רשתות מחשבים

עומר רוזנבוים
ברק גונן
שלומי הוד

המרכז לחינוך סייבר
CYBER EDUCATION CENTER

רשתות מחשבים

גרסה 3.2 נובמבבר 2020

<u>כתיבה</u>

עומר רוזנבוים – כותב ראשי

ברק גונן

תומר גביש

מתן זינגר

רמי עמר

שלומי בוטנרו

שלומי הוד

<u>עריכה</u>

עומר רוזנבוים

ברק גונן

שלומי הוד

אין לשכפל, להעתיק, לצלם, להקליט, לתרגם, לאחסן במאגר מידע, לשדר או לקלוט בכל דרך או אמצעי אלקטרוני, אופטי או מכני או אחר – כל חלק שהוא מהחומר שבספר זה. שימוש מסחרי מכל סוג שהוא בחומר הכלול בספר זה אסור בהחלט, אלא ברשות מפורשת בכתב ממטה הסייבר הצה"לי.

© כל הזכויות על החומרים ממקורות חיצוניים ששובצו בספר זה שמורות לבעליהן. פירוט בעלי הזכויות – בסוף הספר.

© תשע"ד - תש"פ, 2014 - 2020. כל הזכויות שמורות למרכז לחינוך סייבר של קרן רש"י.

הודפס בישראל.

http://www.cyber.org.il

תוכן עניינים

תוכן עניינים ... 4

מבוא ... 10
קהל היעד של הספר ... 10
שיטת הלימוד של הספר ... 11
צעדים להמשך ... 11
ידע מקדים נדרש ... 11
התקנות נדרשות ... 12
אייקונים ... 19
תודות ... 19

תוכן עניינים מצגות ... 20
פייתון ... 20
רשתות ... 21

פרק 1 ... 22

תחילת מסע – איך עובד האינטרנט? ... 22
World Wide Web – WWW ... 23
כתובות IP ... 24
GeoIP ... 28
ענן האינטרנט ... 30
DNS ... 34
איך עובד האינטרנט – סיכום ... 36

פרק 2 ... 37

תכנות ב-Sockets ... 37
שרת-לקוח ... 37
אז מה זה בעצם Socket ? ... 39
כתובות של Socket ... 39
סיכום תכנות ב-Sockets ... 59

פרק 3 ... 60

Wireshark ומודל חמש השכבות ... 60
מודל חמש השכבות ... 60
כיצד ניתן לארגן את כל המידע הזה? ... 61
מה זה בעצם אומר מודל של שכבות? ... 61
פרוטוקול – הגדרה ... 64
כיצד בנויה פקטה (Packet)? ... 66
פירוט חמש השכבות ... 68
שכבה ראשונה – השכבה הפיזית ... 68
שכבה שנייה – שכבת הקו ... 68
שכבה שלישית – שכבת הרשת ... 69
שכבה רביעית – שכבת התעבורה ... 70
שכבה חמישית – שכבת האפליקציה ... 70
מודל השכבות ו-Sockets ... 72

למה המידע מחולק לפקטות? 73
Wireshark 74
Capture Options 75
התחלה ועצירה של הסנפה 76
שמירה ופתיחה של קבצי הסנפה 77
מסננים (Filters) 78
סוגי מסננים 78
דוגמאות 78
שימוש ב-Wireshark לניתוח מודל חמש השכבות 82
Follow TCP/UDP Stream 87
סטטיסטיקות 88
סיכום Wireshark ומודל חמש השכבות 90
פרק 4 91
שכבת האפליקציה 91
פרוטוקול HTTP – בקשה ותגובה 91
מהו משאב רשת? 92
עיקרון "בקשה-תגובה״ 93
שורת הבקשה של HTTP 93
שורת התגובה של HTTP 93
סיכום ביניים של התקשורת שראינו 94
מה מסתתר בתוך הבקשה/תגובה? Headerים ("מבוא") ב-HTTP 94
מבנה פורמלי של בקשת HTTP 95
מבנה פורמלי של תשובת HTTP 96
תוכן (מידע – Data) ב-HTTP 97
פרוטוקול HTTP – תכנות צד שרת בפייתון, הגשת קבצים בתגובה לבקשת GET 98
שלד קוד של תרגיל כתיבת שרת HTTP 101
הוספת אייקון לדפדפן 102
כלי דיבוג – breakpoints 102
כלי דיבוג – דפדפן כרום 103
כלי דיבוג – Wireshark 104
פרוטוקול HTTP – תשובה "תכנותית" לבקשות GET, ופרמטרים של בקשת GET 107
פרוטוקול HTTP – בקשות POST ותכנות צד לקוח בפייתון (הרחבה) 112
האם ניתן להשתמש בפרמטרים של בקשת GET כדי להעביר את כל התמונה? 113
HTTP – סיכום קצר 114
HTTP – נושאים מתקדמים ותרגילי מחקר 116
Cache (מטמון) 116
Cookies ("עוגיות") 118
פרוטוקול DNS – הסבר כללי 122
הירככיית שמות 123
גלישה לאתרים שנמצאים על שרתי אחסון 127
מחקר פרוטוקול – SMTP 132
שכבת האפליקציה – סיכום 134
שכבת האפליקציה – צעדים להמשך 135
קריאה נוספת 135
נושאים מתקדמים בפרוטוקול HTTP 135
נושאים מתקדמים בפרוטוקול DNS 135
פרוטוקולים נוספים בשכבת האפליקציה 135
פרק 5 136
Scapy 136
מבוא ל-Scapy 136

מה למשל אפשר לעשות עם Scapy? 136
בואו נתחיל – הרצת Scapy 137
קבלה של פקטות (או – הסנפה) 138
כיצד נוכל לסנן חבילות לא מזוהות? 147
הסנפה – סיכום 148
יצירת חבילות 148
שכבות 150
Resolving 152
שליחת פקטות 152
שימוש ב-Scapy מתוך קובץ סקריפט 153
Scapy – סיכום 154
Scapy – צעדים להמשך 154
קריאה נוספת 154
תרגיל מתקדם 154
פרק 6 156
שכבת התעבורה 156
מה התפקיד של שכבת התעבורה? 156
ריבוב אפליקציות – פורטים 156
מי מחליט על מספרי הפורטים? 161
העברה אמינה של מידע 162
מיקום שכבת התעבורה במודל השכבות 162
מה השירותים ששכבת התעבורה מספקת לשכבה שמעליה? 162
מה השירותים ששכבת התעבורה מקבלת מן השכבה שמתחתיה? 163
פרוטוקולים מבוססי קישור ולא מבוססי קישור 163
פרוטוקולים מבוססי קישור 163
פרוטוקולים שאינם מבוססי קישור 164
מתי נעדיף פרוטוקול מבוסס קישור ומתי פרוטוקול שלא מבוסס קישור? 165
מקרה מבחן: תוכנה להעברת קבצים גדולים 165
מקרה מבחן: פרוטוקול DNS 165
מקרה מבחן: תוכנה לשיתוף תמונות 166
מקרה מבחן: Skype 166
UDP – User Datagram Protocol 168
מה גודל ה-Header של חבילת UDP? 168
אילו שדות יש ב-Header של חבילת UDP? מה התפקיד של כל שדה? 169
מה מציין שדה האורך ב-UDP? 169
מה זה Checksum? 170
Socket של UDP 173
UDP ב-Scapy 177
TCP – Transmission Control Protocol 185
כיצד ניתן לוודא שהמידע מגיע אל היעד? כיצד ניתן לוודא שהוא מגיע בסדר הנכון? 185
איך TCP משתמש ב-Sequence Numbers? 188
איך TCP משתמש ב-Acknowledgement Numbers? 191
מענה ב-ACK יחיד למספר חבילות 194
הקמת קישור ב-TCP 195
חבילה ראשונה – SYN 196
חבילה שנייה – SYN + ACK 197
חבילה שלישית – ACK 198
העברת מידע על גבי קישור שנוצר 209
תפקידים ושיפורים נוספים של TCP 211

שכבת התעבורה – סיכום 212
שכבת התעבורה – צעדים להמשך 213
קריאה נוספת 213
נספח א' – TCP Header 214
פרק 7 216
שכבת הרשת 216
מה תפקידה של שכבת הרשת? 216
אילו אתגרים עומדים בפני שכבת הרשת? 218
מיקום שכבת הרשת במודל השכבות 218
מה השירותים ששכבת הרשת מספקת לשכבה שמעליה? 218
מה השירותים ששכבת הרשת מקבלת מהשכבה שמתחתיה? 219
מסלולים ברשת 219
מה צריכה שכבת הרשת לדעת בכדי להחליט כיצד לנתב? 221
עשה זאת בעצמך – תכנון מסלול ניתוב 222
פרוטוקול IP 223
כתובות IP 223
מה כתובת ה-IP שלי? 223
מה זו כתובת IP? 224
כמה כתובות IP אפשריות קיימות? 225
מהו מזהה הרשת שלי? מהו מזהה הישות? 228
כתובות IP מיוחדות 231
כתובות פרטיות ו-NAT 233
ניתוב 238
נתב (Router) 238
כיצד נתב מחליט לאן לנתב? 239
מהי טבלת הניתוב שלי? 241
ICMP 243
איך Ping עובד? 243
איך Traceroute עובד? 254
DHCP 261
שכבת הרשת – סיכום 271
שכבת הרשת – צעדים להמשך 272
קריאה נוספת 272
נספח א' – IP Header 274
נספח ב' – IPv6 278
כתובות IPv6 278
IPv6 Header 279
פרק 8 281
שכבת הקו 281
מה התפקיד של שכבת הקו? 281
איפה ממומשת שכבת הקו? 282
פרוטוקול Ethernet 283
מהי הכתובת שלי בשכבת הקו? 283
איך בנויה כתובת Ethernet? 287
כתובת Broadcast 288
מבנה מסגרת Ethernet 289
פרוטוקול Address Resolution Protocol – ARP 294
מטמון (Cache) של ARP 296

שליחת מסגרות בשכבה שנייה באמצעות Scapy 300
רכיבי רשת בשכבת הקו ובשכבה הפיזית 302
Hub (רכזת) 302
Switch (מתג) 304
כיצד Switch פועל? 305
שכבת הקו – סיכום 310
נספח א' – התנגשויות 311
ערוץ משותף – הגדרה 311
מהי התנגשות? 312
מה תפקידה של שכבת הקו בנוגע להתנגשויות? 312
מניעת התנגשויות 313
שכבת הקו – צעדים להמשך 315
קריאה נוספת 315
נספח א' – כתובות Ethernet של קבוצות 317
פרק 9 321
רכיבי רשת 321
Hub (רכזת) 321
Switch (מתג) 322
Router (נתב) 323
טבלת סיכום 324
פרק 10 325
השכבה הפיזית (העשרה) 325
מבוא 325
עמודי התווך של התקשורת 326
העברת מידע באוויר 327
גלים נושאי מידע 329
כבלי נחושת כתווך תקשורת 333
הרשת בבית 333
קו הטלפון הביתי, או למה צריך פילטר למודם ADSL? 334
איך עובד טלפון? 334
איך עובד מודם? 336
אנלוגי, דיגיטלי ומה שביניהם 338
סיכום ביניים 341
הרשת המשרדית 342
משרד קטן 342
משרד גדול 345
ספק תקשורת 347
השכבה הפיזית – סיכום 349
נספח א' – הרשת האלחוטית 350
פרק 11 352
איך הכל מתחבר, ואיך עובד האינטרנט? 352
מה קורה כשאנו גולשים ל-Facebook? 352
מה המחשב שלנו צריך לעשות בכדי להצליח לתקשר עם האינטרנט? 352
איזה מידע יש למחשב שלנו על הרשת? 353
איך המחשב שלנו משיג את כתובת ה-IP ושאר פרטי הרשת שלו? 353
איך ההודעה הגיעה אל שרת ה-DHCP? 354
מה השלב הבא? 354

מהו שרת ה-DNS שלנו? כיצד המחשב יודע זאת? 355
כיצד המחשב שלנו יודע לפנות אל שרת ה-DNS? 355
איך נצליח לתקשר עם הנתב? 356
איך ה-Switch יודע להעביר את ההודעות? 357
מהן הכתובות בחבילה? 358
כתובות ה-MAC 358
כתובות ה-IP 360
מהן הכתובות בתחנה הבאה? 360
כתובות ה-MAC 362
כתובות ה-IP 362
מה עושה כל נתב עם החבילה? 363
כיצד מוצא המחשב את כתובת ה-IP של Facebook? 364
מה השלב הבא? 364
באילו פורטים תתבצע התקשורת? 365
כיצד נראית הרמת הקישור? 366
איך נראית בקשת ה-HTTP? 369
איך נראית תשובת ה-HTTP? 369
מה קורה כאשר המחשב שלנו נמצא מאחורי NAT? 370
איך הכל מתחבר, ואיך עובד האינטרנט – סיכום 373

פרק 12 374

תכנות Sockets מתקדם: ריבוי משתמשים (הרחבה) 374
שירות לכמה לקוחות במקביל 375
הפתרון – Select 376
קריאה לפונקציה select 378
תיאור הפרוטוקול 392
צד הלקוח 392
צד השרת 395
תכנות Sockets מתקדם – סיכום 396

פרק 13 397

מילון מושגים 397
פרק 1 – תחילת מסע – איך עובד האינטרנט? 397
פרק 2 – תכנות ב-Sockets 397
פרק 3 – Wireshark ומודל חמש השכבות 398
פרק 4 – שכבת האפליקציה 398
פרק 5 – Scapy 400
פרק 6 – שכבת התעבורה 400
פרק 7 – שכבת הרשת 401
פרק 8 – שכבת הקו 403
פרק 10 – השכבה הפיזית 403

פרק 14 407

פקודות וכלים 407
הרשימה 407

זכויות יוצרים – מקורות חיצוניים 409

מבוא

רשתות תקשורת הן דבר מדהים. מאז ומעולם, בני אדם רצו להעביר מסרים ביניהם. בעבר הרחוק, בהיעדר אמצעים אחרים, התבססה התקשורת בעיקר על תקשורת מילולית ושפת גוף. על מנת להעביר מסרים למרחקים גדולים יותר, נעשה שימוש בשליחים. בתקופת המערות, החל האדם הקדמון לעשות שימוש בציורי מערות בכדי לתקשר. בשלב מאוחר יותר הופיע הכתב – מערכת סימנים מוסכמת שאפשרה העברת מסרים בצורה רחבה יותר. המצאת הדפוס, במאה ה-15, אפשרה להעביר ידע ומסרים לאנשים רבים ובעלות העתקה נמוכה יחסית.

במאה ה-19 החלה מתפתחת התקשורת האלקטרונית, המאפשרת תקשורת המונים מהירה ויעילה. כבני אדם, עברנו דרך ארוכה מאז שהתקשורת התבססה על תקשורת מילולית, ועד לשימוש היום יומי בגלישה באינטרנט, בדואר אלקטרוני או בתוכנות העברת מסרים כגון WhatsApp. המהפכה התקשורתית משפיעה על כל תחומי החיים שלנו – על הדרך בה אנו מדברים זה עם זה, על הדרך בה אנו מחפשים מידע כדי ללמוד, על מידת ההיכרות שלנו עם העולם הסובב אותנו ואף על החלטות שאנו מקבלים בחיינו.

התפתחות התקשורת והתפתחות המחשוב שזורים זה בזה. עולם המחשבים משתנה ומתפתח בקצב מהיר. בעוד המחשבים הראשונים היו מבודדים זה מזה, רוב מכריע של המחשבים כיום מחוברים זה לזה דרך רשת האינטרנט, שהיא למעשה רשת של תת רשתות. לכל רשת יש את המאפיינים שלה: יש רשתות קטנות (בהן מחוברים למשל שני מחשבים), רשתות בינוניות (כמו רשת של בית ספר, שיכולה לחבר כמה עשרות או מאות מחשבים) ורשתות גדולות (כמו רשת של חברה עם אלפי מחשבים). יש רשתות קוויות ורשתות אלחוטיות, רשתות מהירות ורשתות איטיות. **מטרת כל הרשתות הינה להעביר מידע בין מחשבים.**

הספר נכנס אל תוך העולם המדהים של רשתות מחשבים. נלמד כיצד עובדת התקשורת בין מחשבים, נכיר סוגים שונים של רשתות, נבין איך הן בנויות ואיך הכל מתחבר לעולם הווירטואלי שאנו מכירים כיום.

קהל היעד של הספר

הספר מיועד לשני קהלי יעד עיקריים:

1. תלמידים ומורים הלומדים במסגרת חלופת הגנת סייבר במגמת הנדסת תוכנה.
2. כל מי שמעוניין ללמוד את תחום רשתות מחשבים באופן עצמאי.

שיטת הלימוד של הספר

ספר זה ככל הנראה שונה מספרי לימוד אחרים שהכרתם. הספר נועד לאפשר לימוד עצמאי, והוא **פרקטי** מאוד וכולל תרגול רב. רשתות מחשבים הינו נושא עצום ומורכב, ולא ניתן לכסות את כולו או מרביתו בספר אחד. הקו שהנחה אותנו בהחלטה אילו נושאים ייכללו בספר זה ואילו יישארו מחוצה לו היה **מעשיות** – הספר מתמקד באותם נושאים אשר ניתן ליישם ולתרגל בקלות יחסית. הספר כולל מעט מאוד נושאים שהם תיאורטיים בלבד, ורובו ככולו מתעסק במימושים אמיתיים וביטויים ברשת האינטרנט. כבר מהפרק השני של הספר, הלימוד ילווה בכתיבת קוד מצד הלומד.

במהלך הלימוד בספר, עליכם לתפקד כ**לומדים פעילים** – כלומר, לא רק לקרוא ולהבין את החומר, אלא גם לתרגל אותו. בספר משולבים תרגילים רבים, חלק מודרכים וחלק לביצוע עצמי. בכדי לרכוש שליטה בחומר הלימוד, יש לבצע את כל התרגילים ולוודא שאתם מבינים אותם. התרגילים המודרכים נבנו כך שהם מפורקים למשימות מובנות המתבססות זו על זו, ומניחות שהלומד מבצע בפועל את ההנחיות. אל תסתפקו בקריאת התרגילים המודרכים, והקפידו לבצע אותם שלב אחרי שלב.

אחת ממטרות הספר היא **להקנות כלים** בהם תוכלו להשתמש בעצמכם, כדי להרחיב את אופקיכם, לחקור וללמוד באופן עצמאי. אי לכך, תזכו במהלך הקריאה להיחשף לתוכנות, כלי תכנות ודרכי חשיבה שיאפשרו לכם להרחיב את הידע גם מעבר למה שמופיע בספר זה.

צעדים להמשך

כחלק משיטת הלימוד של הספר, המעודדת למידה עצמאית, בסוף חלק מהפרקים הוספנו סעיף "צעדים להמשך". סעיף זה נועד לתלמידים סקרנים המעוניינים להרחיב את הידע, וכולל מקורות מידע נוספים ותרגילים מתקדמים.

ידע מקדים נדרש

ספר זה מניח כי לקורא היכרות בסיסית לפחות עם **השפה Python**. בהתאם לצורך, אתם מוזמנים להשתמש בספר הלימוד של שפת Python מאת ברק גונן, הזמין בכתובת:

https://data.cyber.org.il/python/python_book.pdf

התקנות נדרשות

כאמור ספר הלימוד מבוסס על שפת פייתון. ישנן התקנות רבות של פייתון, לכן נרצה להמליץ על סביבת עבודה ושימוש נכון בסביבת העבודה. גרסת הפייתון של ההתקנה היא 3.8, בתואם לספר הלימוד. התקנה זו כוללת את כל הכלים הנוספים שנצטרך להשתמש בהם במהלך הלימוד (כגון wireshark, scapy).

שימו לב:

התוכנות הן עבור **מערכת ההפעלה Windows10**, עבור מערכות הפעלה אחרות ישנן גרסאות ספציפיות של התוכנות ויש להוריד אותן באופן עצמאי. יתר ההתקנה צפוי להיות דומה.

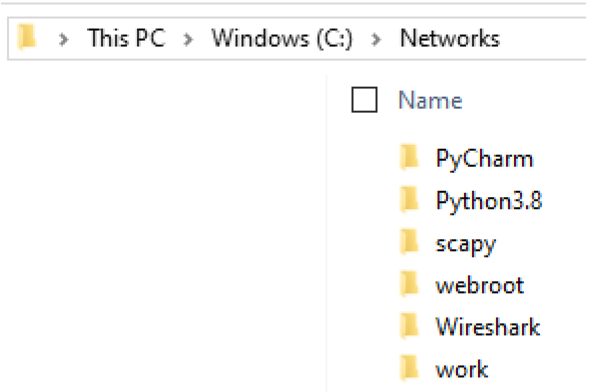
התוכנות הן (לפי סדר ההתקנה):

1. פייתון, גרסה 3.8.0
2. סביבת פיתוח PyCharm, גרסה 2019.2.5
3. דרייבר הסנפה מכרטיס רשת, Npcap גרסה 0.9984
4. תוכנת הסנפה Wireshark גרסה 3.0.6
5. מודול יצירת פקטות של פייתון, Scapy גרסה 2.4.3

קובץ התקנה מרוכז נמצא בכתובת:

https://data.cyber.org.il/networks/installs.zip

אפשר להתקין את התוכנות בכל תיקיה, אך מנסיון רב שנים התקנה "מפוזרת" של התוכנות גורמת אינספור בעיות ללומדים. לדוגמה, חלק מהתוכנות לא מסוגלות לזהות תיקיות שיש בהן עברית. חלק מהתוכנות דורשות התאמה של משתני סביבה. אם תפעלו לפי ההוראות הבאות, תחסכו לעצמכם בעיות. שימו לב למבנה התיקיות המומלץ בסיום ההתקנות:



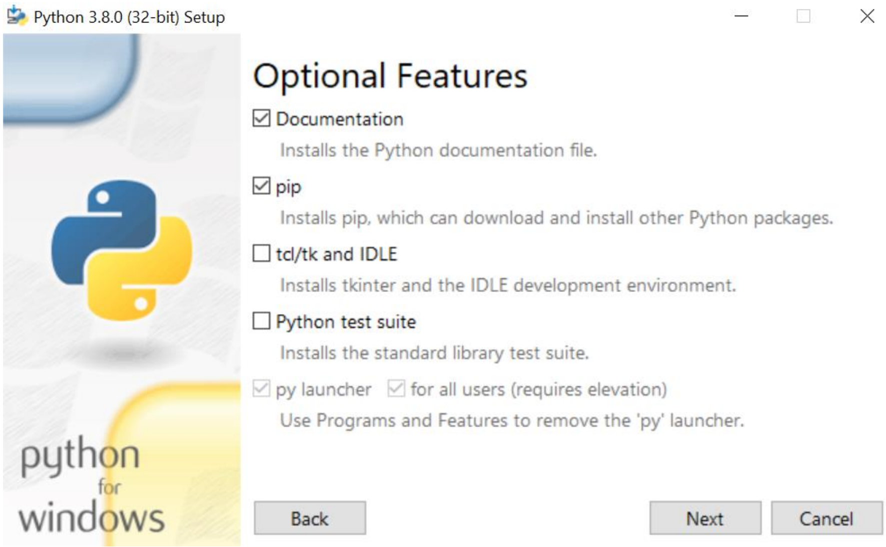
התיקיה הראשית היא Networks, שם נשים את כל תתי התיקיות. ישנן חמש תיקיות המיועדות להתקנות ותיקיה נוספת בשם Work המיועדת לקבצי הפייתון שלכם.

א. <u>התקנת פייתון</u>

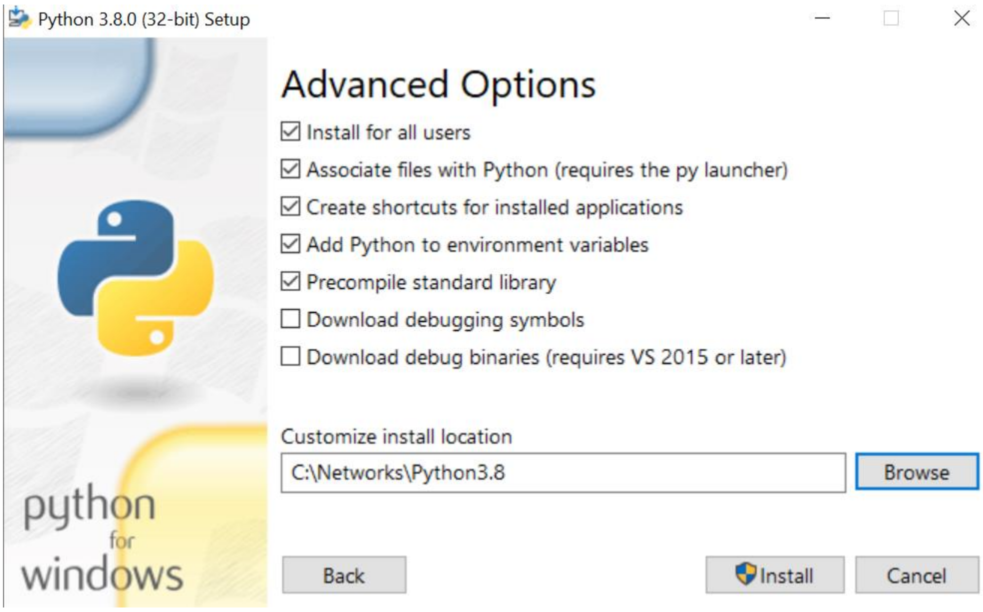
הקליקו על הקובץ python-3.8.0.exe, יופיע מסך ההתקנה הבא:

בחרו באפשרות "customize installation".

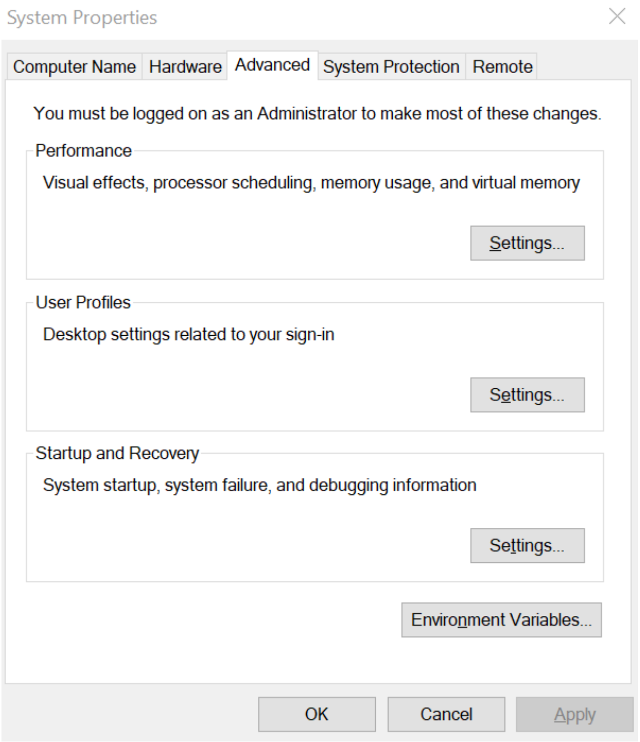
בחרו באפשרויות הבאות:



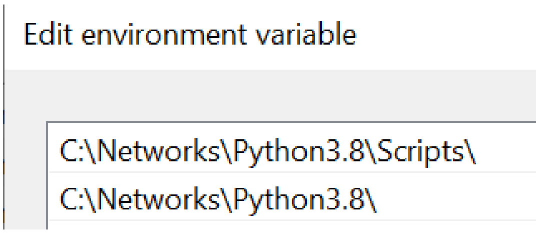
ובמסך הבא בחרו את האפשרויות המסומנות, והזינו בתור תיקיית התקנה את c:\Networks\Python3.8:



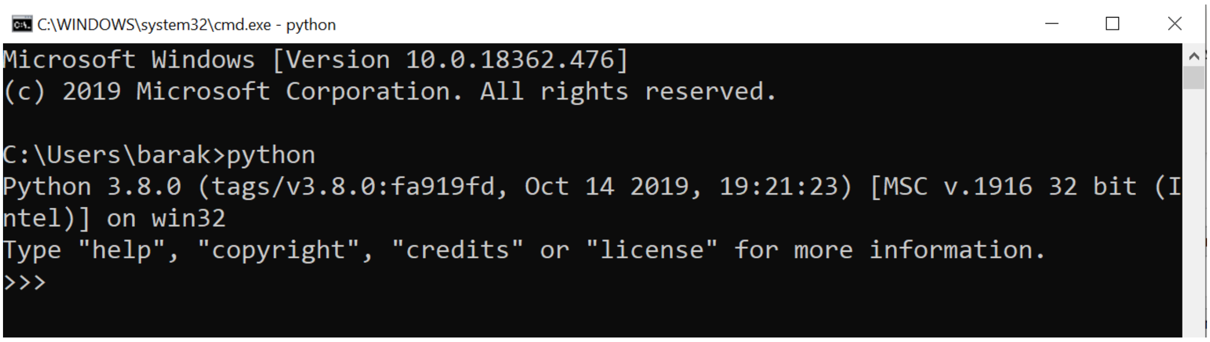
כעת בידקו שפייתון מוגדר בתוך ה-PATH של משתני הסביבה שלכם. במסך החיפוש של windows הקלידו env ובחרו באפשרות Edit The System Environment Variables. הקליקו על הלחצן Environment Variables.



וודאו שבתוך המשתנה PATH יש את שתי הכניסות הבאות. אם הן אינן, הוסיפו אותן ידנית (בהנחה שהתקנתם את פייתון בתוך c:\networks\python3.8)

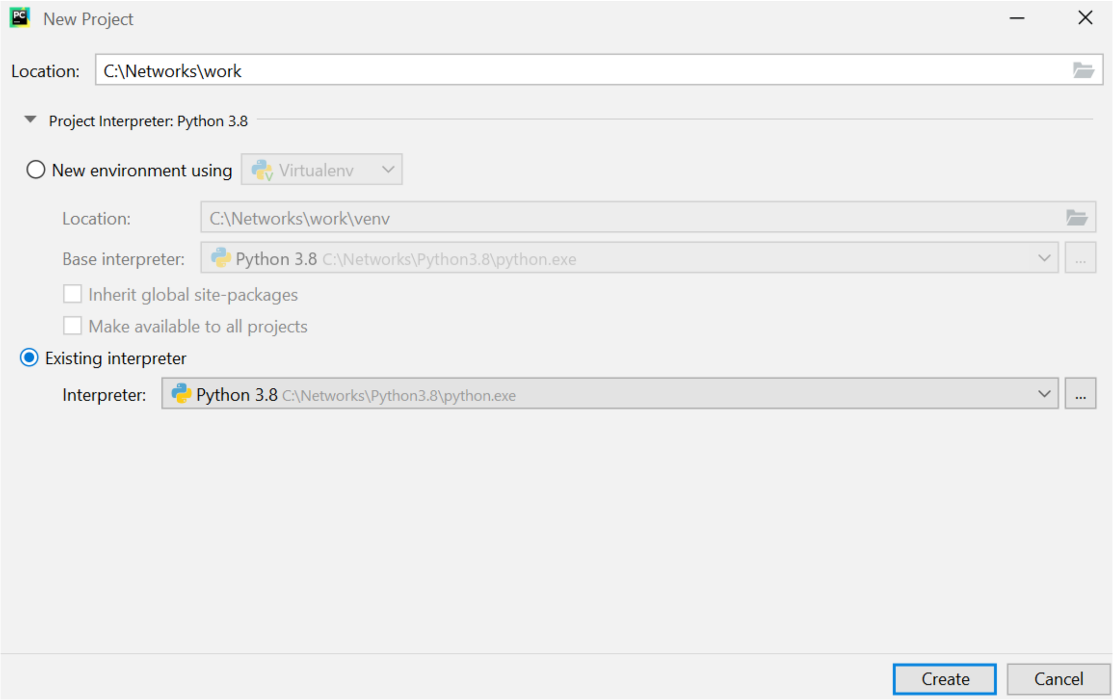


כדי לוודא שהכל עובד כשורה, פיתחו cmd והקלידו python. צפוי שתקבלו את ההדפסה הבאה:



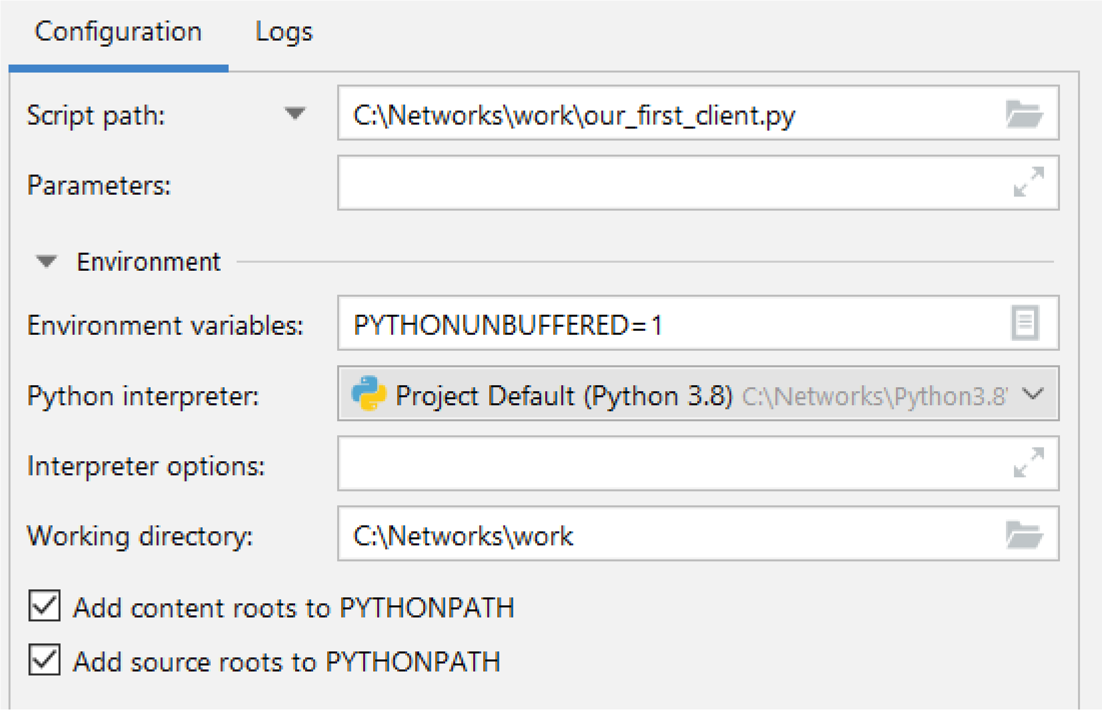
ב. PyCharm

הקליקו על קובץ ההתקנה, היא תתבצע מעצמה.
עם ההפעלה, בחרו באפשרות "Create new project" ופיתחו פרוייקט בספריה c:\networks\work.
תצטרכו להגדיר היכן נמצא ה-python interpreter שלכם, בצעו זאת כך:



אפשרות נוספת היא להגדיר את ה- Interpreter בעצמכם, עבור כל קובץ אותו אתם מריצים. מתי נכון לעשות זאת? אם קבלתם הודעת שגיאה "Invalid Python Interpreter selected for the project".ההודעה מופיעה שורות הקוד של הקובץ אותו אתם מנסים להריץ:

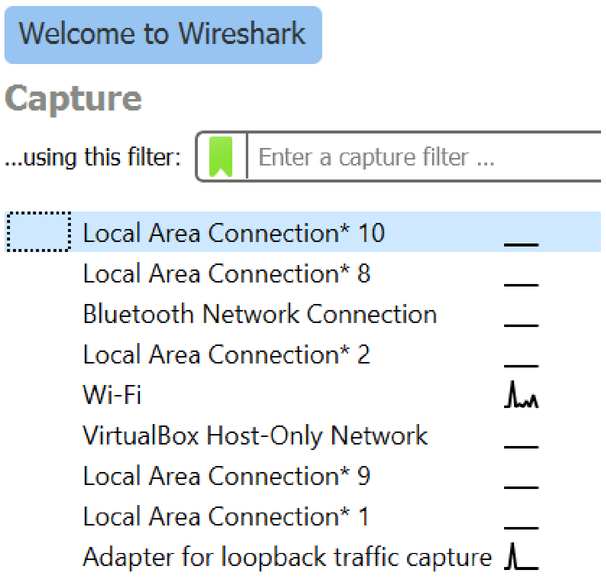
הקליקו על "Configure Python Interpreter" בצד הימני של ההודעה ותעברו למסך ההגדרות הבא:



שנו את הערך של ה-Python interpreter כך שיצביע על הקובץ Python.exe שבתיקיה c:\networks\work\python3.8, כעת הקליקו על כפתור ה-OK והבעיה נעלמה.

ג. <u>Wireshark ו-npcap</u>

הקליקו על תוכנת ההתקנה של Wireshark. התוכנה תשאל אם ברצונכם להתקין על הדרך את npcap, בחרו באפשרות הזו.
הקליקו על האייקון של Wireshark. התוכנה תגלה באופן אוטומטי את כל ממשקי הרשת שלכם. כרטיס רשת פעיל יראה גרף עולה ויורד, בהתאם לרמת הפעילות.



במקרה זה, המחשב מחובר דרך Wi-Fi. בחרו בממשק הפעיל והתחילו הסנפה.

<u>ה. scapy</u>

העתיקו את התוכן של התיקיה scapy-master מקובץ ה-zip אל תיקיית c:\networks\scapy.

פיתחו cmd והקלידו

cd c:\networks\scapy

כדי להתקין הקלידו python setup.py install

כדי לבדוק אם ההתקנה הצליחה, הקלידו scapy. תקבלו את המסך הבא, אין מה להיות מוטרדים מהודעות השגיאה- הן שייכות למודולים שאין לנו צורך בהם.

```
C:\WINDOWS\system32\cmd.exe - scapy
c:\Networks\scapy>scapy
INFO: Can't import matplotlib. Won't be able to plot.
INFO: Can't import PyX. Won't be able to use psdump() or pdfdump().
WARNING: No route found for IPv6 destination :: (no default route?)
INFO: Can't import python-cryptography v1.7+. Disabled WEP decryption/encryption. (Dot11)
INFO: Can't import python-cryptography v1.7+. Disabled IPsec encryption/authentication.
WARNING: IPython not available. Using standard Python shell instead.
AutoCompletion, History are disabled.
WARNING: On Windows, colors are also disabled

                     aSPY//YASa
             apyyyyCY//////////YCa       |
            sY//////YSpcs  scpCY//Pp     | Welcome to Scapy
 ayp ayyyyyyySCP//Pp           syY//C    | Version git-archive.devd31378c886
 AYAsAYYYYYYYY///Ps              cY//S   |
         pCCCCY//p          cSSps y//Y   | https://github.com/secdev/scapy
         SPPPP///a          pP///AC//Y   |
              A//A            cyP////C   | Have fun!
              p///Ac            sC///a   |
              P////YCpc           A//A   | Craft packets like I craft my beer.
       scccccp///pSP///p          p//Y   |               -- Jean De Clerck
      sY/////////y  caa           S//P   |
       cayCyayP//Ya              pY/Ya
        sY/PsY////YCc          aC//Yp
         sc  sccaCY//PCypaapyCP//YSs
                  spCPY//////YPSps
                       ccaacs
```

כדי לוודא שה-scapy שלכם עובד כשורה, הסניפו באמצעותו 2 פקטות והציגו אחת מהן:

```
>>> p = sniff(count = 2)
>>> p[0].show()
```

זהו, סיימנו. מוכנים להתחיל לעבוד!

אייקונים

בספר זה אנו משתמשים באייקונים הבאים בכדי להדגיש נושאים ובכדי להקל על הקריאה:

שאלה

הגדרה למונח

הדגשת נקודה חשובה

"תרגיל מודרך". עליכם לפתור תרגילים אלו תוך כדי קריאת ההסבר

תרגיל לביצוע. תרגילים אלו עליכם לפתור בעצמכם, והפתרון בדרך כלל לא יוצג בספר

פתרון מודרך לתרגיל אותו היה עליכם לפתור בעצמכם

רקע היסטורי, או מידע העשרתי אחר

הפניה לסרטון

תודות

אנשים רבים תרמו לתהליך יצירתו של ספר זה. איריס צור ברגורי ליוותה את הספר מתהליך התכנון ועד לשלביו הסופיים והשפיעה עליו רבות. תומר גלון מימש את צד השרת עבור תרגילים במהלך הספר. מיכל לשם סייעה באופן משמעותי בהבאת הספר לידי גרסה להפצה. משובים שקיבלנו על הספר לאורך הזמן שיפרו אותו ותרמו לו מאוד. באופן מיוחד אנו מבקשים להודות ליוסי ממו ומתן עבודי על הערותיהם הבונות. יוחאי אייזנרייך כתב פתרונות רבים לתרגילים הניתנים בספר, ובכך סייע לשפר אותם וכן סיפק פתרונות לדוגמה עבור תלמידים. כמו כן אנו מבקשים להודות לדניאל גולדברג, נעם ארז, ליאור גרנשטיין, יהודה אור ואנטולי פיימר על משוביהם. לכל המורים, התלמידים והחברים שהשפיעו וסייעו בתהליך יצירת הספר – תודה רבה.

עומר רוזנבוים, ברק גונן, שלומי הוד

תוכן עניינים מצגות

פייתון

Before we start:	http://data.cyber.org.il/python/1450-3-00.pdf
Intro and CMD:	http://data.cyber.org.il/python/1450-3-01.pdf
Pycharm:	http://data.cyber.org.il/python/1450-3-02.pdf
Variables, conditions and loops:	http://data.cyber.org.il/python/1450-3-03.pdf
Strings:	http://data.cyber.org.il/python/1450-3-04.pdf
Functions:	http://data.cyber.org.il/python/1450-3-05.pdf
Assert:	http://data.cyber.org.il/python/1450-3-06.pdf
Files and script parameters:	http://data.cyber.org.il/python/1450-3-07.pdf
Lists and tuples:	http://data.cyber.org.il/python/1450-3-08.pdf
Dictionaries:	http://data.cyber.org.il/python/1450-3-09.pdf
Object Oriented Programming:	http://data.cyber.org.il/python/1450-3-10.pdf
Utilities and exceptions:	http://data.cyber.org.il/python/1450-3-11.pdf
Regular expressions:	http://data.cyber.org.il/python/1450-3-12.pdf

רשתות

מבוא לשנת הלימודים:	http://data.cyber.org.il/networks/1450-2-00.pdf
פרק 1 – מבוא לרשתות מחשבים:	http://data.cyber.org.il/networks/1450-2-01.pdf
פרק 2 – תכנות סוקטים:	http://data.cyber.org.il/networks/1450-2-02.pdf
פרק 3א – מודל חמש השכבות:	http://data.cyber.org.il/networks/1450-2-03.pdf
פרק 3ב – wireshark:	http://data.cyber.org.il/networks/1450-2-04.pdf
פרק 4א – שכבת האפליקציה HTTP:	http://data.cyber.org.il/networks/1450-2-05.pdf
פרק 4ב – HTTP נושאים מתקדמים:	http://data.cyber.org.il/networks/1450-2-06.pdf
פרק 4ג – פרוטוקול DNS:	http://data.cyber.org.il/networks/1450-2-07.pdf
פרק 4ד – אבחון פרוטוקול SMTP:	http://data.cyber.org.il/networks/1450-2-08.pdf
פרק 5 – Scapy:	http://data.cyber.org.il/networks/1450-2-09.pdf
פרק 6א – שכבת התעבורה:	http://data.cyber.org.il/networks/1450-2-10.pdf
פרק 6ב–מבוא לפרוטוקולים של שכבת התעבורה:	http://data.cyber.org.il/networks/1450-2-11.pdf
פרק 6ג – UDP:	http://data.cyber.org.il/networks/1450-2-12.pdf
פרק 6ד – TCP:	http://data.cyber.org.il/networks/1450-2-13.pdf
פרק 7א – שכבת הרשת מבוא לניתוב:	http://data.cyber.org.il/networks/1450-2-14.pdf
פרק 7ב – כתובות IP וראוטר:	http://data.cyber.org.il/networks/1450-2-15.pdf
פרק 7ג – פרוטוקול ICMP:	http://data.cyber.org.il/networks/1450-2-16.pdf
פרק 7ד – פרוטוקול DHCP:	http://data.cyber.org.il/networks/1450-2-17.pdf
פרק 8 – שכבת הקו:	http://data.cyber.org.il/networks/1450-2-18.pdf
פרק 10 – השכבה הפיזית:	http://data.cyber.org.il/networks/1450-2-19.pdf
פרק 11 – איך הכל מתחבר:	http://data.cyber.org.il/networks/1450-2-20.pdf

פרק 1

תחילת מסע – איך עובד האינטרנט?

ניתן לצפות בסרטון המלווה פרק זה בכתובת: http://youtu.be/ad8EOsXFuxE.

ספר זה עוסק ברשתות מחשבים. מה זה בעצם אומר? איך רשת האינטרנט עובדת?

בפרק זה נתחיל לענות על שאלות אלו באופן כללי, ונקבל תמונה כללית על איך עובד האינטרנט. בהמשך הספר, נרד לפרטים ונזכה לקבל תמונה הרבה יותר מעמיקה ומדויקת. על מנת להתחיל את ההסבר, נפתח בשאלה:

מה קורה כשאנו גולשים לאתר Facebook?

רובנו גלשנו ל-Facebook, הרשת החברתית העצומה שמונה מעל למיליארד משתמשים. אך האם עצרנו לשאול את עצמנו – מה בעצם קורה מאחורי הקלעים כשגולשים? איך יתכן שאנו נמצאים בבית, מקישים בדפדפן (Browser) את הכתובת "www.facebook.com", מקישים Enter, ומקבלים תמונת מצב של כל החברים שלנו?

על מנת לענות על שאלה זו, עלינו להבין ראשית מה האתר Facebook צריך כדי לתפקד.

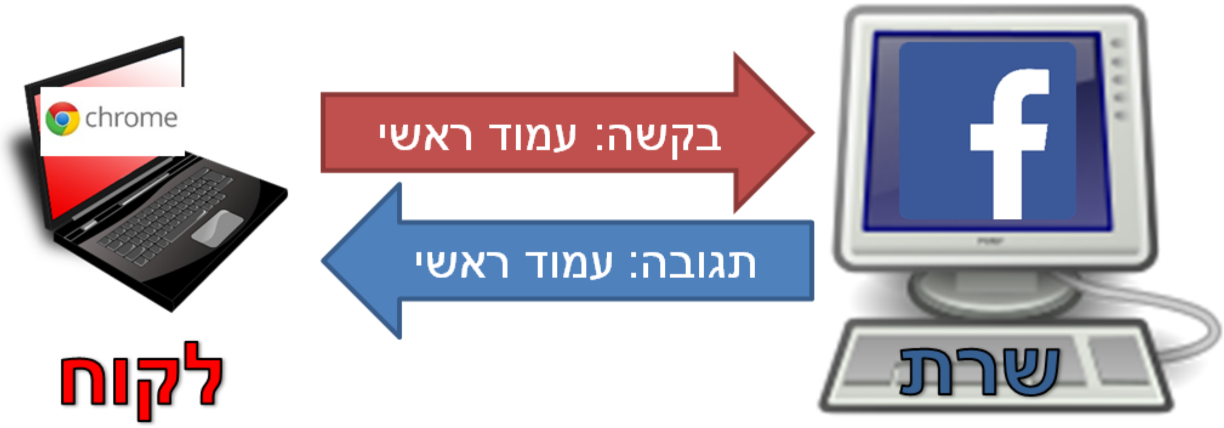
כל אתר, וכך גם אתר Facebook, זקוק ל**אחסון (Hosting)** – הכוונה למקום בו יימצאו דפי האתר ואליו יפנו המשתמשים. אתר Facebook ישמור גם את המידע על כל המשתמשים כגון: מי חבר של מי, התמונות שהועלו לאתר, סטטוסים וכו'. בנוסף, אתר Facebook זקוק ל**עיצוב**. יש לעצב לוגו, להחליט היכן תוצג רשימת החברים, היכן יוצגו העדכונים שלהם, איפה יוצגו הפרסומות ועוד. האתר זקוק גם ל**אימות (Authentication)** – עליו לזהות את המשתמש שפונה אליו. לשם כך, Facebook צריך לזכור את כל המשתמשים והסיסמאות שלהם, ולאפשר לכל משתמש להתחבר ולהזדהות. כלומר, Facebook נזקק ל**תקשורת**. צריכה להיות דרך שתאפשר לאדם לתקשר עם האתר של Facebook, בין אם מהמחשב שלו בישראל, ובין אם מהסמארטפון שלו כשהוא נמצא בטיול בחו"ל.

ספר זה יתמקד בתקשורת שבין רכיבים אלקטרוניים שונים ודרך העברת המידע ביניהם. בפרק הנוכחי, נתאר באופן כללי מה קורה מהרגע שאנו כותבים "www.facebook.com" בדפדפן ומקישים Enter, ועד שמופיע דף הבית של Facebook על המסך.

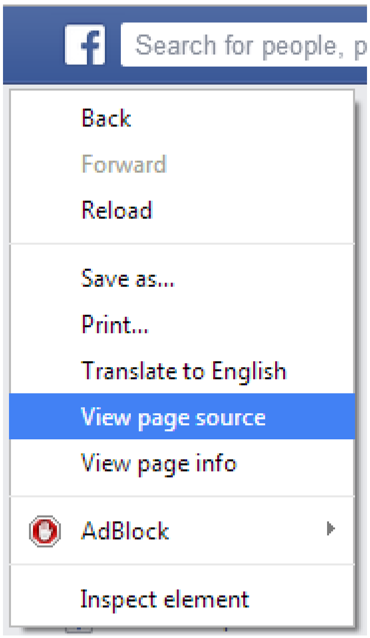
World Wide Web – WWW

כשאנו אומרים "אינטרנט", אנו מתכוונים בדרך כלל ל-WWW (World Wide Web). זהו אוסף עמודי האינטרנט אליהם אנו גולשים בדפדפן. משמעות המילה Web היא רשת. עמודי האינטרנט מקושרים אחד לשני כמו רשת של קורי עכביש המקושרים זה לזה. עמודי האינטרנט מפוזרים בכל רחבי העולם – World Wide – בעוד אתר אחד נמצא בחיפה, שני יכול להיות בטוקיו ושלישי בניו-יורק. העמודים השונים מקושרים ביניהם באמצעות לינקים.

על מנת שהדפדפן יוכל להציג את העמוד הראשי של Facebook, עליו לדעת כיצד העמוד נראה. הכוונה היא לדעת איזה טקסט קיים בעמוד, היכן כל חלק מהעמוד ממוקם, באיזה גופן הטקסט כתוב ובאיזה גודל, האם צריך להציג תמונות על המסך, אילו תמונות ועוד. את כל המידע הזה, הדפדפן משיג מאתר Facebook עצמו. בכדי ש-Facebook ישלח לדפדפן את המידע, על הדפדפן להודיע שהוא מעוניין בו. כלומר, הדפדפן צריך לשלוח הודעת **בקשה (Request)** אל Facebook, ובה להגיד: "שלח לי את המידע הנדרש כדי להציג את העמוד הראשי של האתר www.facebook.com". כאשר Facebook מקבל את הבקשה, הוא שולח **תגובה (Response)** שמכילה את המידע הדרוש.



האתר של Facebook מקבל בקשה ושולח תגובה. כלומר, הוא מספק שירות לדפדפן. כלומר, האתר של Facebook הינו **שרת (Server)**, והדפדפן הינו **לקוח (Client)**. באופן כללי, כאשר גולשים ב-WWW, ישנם **לקוחות** (דפדפנים) ששולחים **בקשות** אל **השרתים** (אתרי האינטרנט), והשרתים מחזירים **תגובות** ללקוחות. אותו מידע שמגיע בתגובות, משמש את הדפדפנים בכדי להציג על המסך את אתרי האינטרנט למשתמשים.



על מנת לראות את המידע שהשרת שלח ללקוח (או לפחות חלק ממנו), ניתן ללחוץ על הכפתור הימני של העכבר באזור ריק באתר האינטרנט, ולבחור באפשרות "View page source" (בעברית: "הצג מקור"):

בחלון שייפתח יוצג טקסט שמכיל את המידע ששלח השרת כתגובה לבקשת הלקוח (הטקסט שנמצא באתר, איפה כל דבר נמצא, אילו תמונות נמצאות וכו'). בשלב זה לא נתעמק במה אנחנו רואים בדיוק. עם זאת, נסו לזהות בעצמכם חלקים מעמוד האינטרנט – האם אתם יכולים לזהות טקסט שמוצג לכם בדפדפן?

היכנסו אל האתר https://www.themarker.com. הקליקו עם הכפתור הימני של העכבר, ובחרו באופציה **View page source**. מצאו בחלון שייפתח את הכותרת הראשית שמופיעה בעמוד של TheMarker.

כתובות IP

על מנת שנוכל לשלוח ולקבל הודעות באינטרנט, עלינו לדעת לאן לשלוח את הבקשות ועל השרת לדעת להיכן לשלוח את התגובות. כאשר אנו שולחים מכתב בדואר, אנו מציינים על המעטפה את **כתובת היעד** (נמען) ואת **כתובת המקור** (מוען). באופן דומה, גם כאשר נשלח מידע ברחבי האינטרנט, יש צורך בכתובות מתאימות שיזהו את השולח ואת היעד של ההודעה. למשל, בדוגמה לעיל, נרצה לדעת מה הכתובת של המחשב ששלח את הבקשה (כתובת המקור), ומה הכתובת של Facebook (כתובת היעד). באינטרנט, כתובות אלו נקראות **כתובות IP (IP Addresses)**.

על IP בכלל וכתובות IP בפרט, נלמד בהרחבה ב[פרק שכבת הרשת](#). בשלב זה רק נבין כיצד כתובות אלו נראות. כשם שלכתובות דואר יש מבנה קבוע, שכולל את שם הנמען (למשל: משה כהן), רחוב ומספר בית (הרצל 1), עיר (ירושלים) ומיקוד (1234567), כך גם לכתובות IP יש מבנה קבוע. כתובות IP מורכבות מארבעה בתים (bytes). כל בית יכול לקבל ערך בין 0 ל-255, ונהוג להפריד את הבתים בנקודה. מכאן ש-כתובת IP יכולה להיות 0.0.0.0, 255.255.255.255, והטווח שביניהן, למשל 1.2.3.4 או 10.42.2.3.

אם נחזור לדוגמה של הדפדפן, הרי שהודעת ה**בקשה** שהדפדפן שולח לאתר Facebook מכילה ב**כתובת המקור** את כתובת ה-IP של המחשב ממנו מתבצעת הגלישה, וב**כתובת היעד** את כתובת ה-IP של האתר Facebook. כאשר נשלחת הודעת **תגובה**, היא נשלחת מ-Facebook לדפדפן, ולכן **כתובת המקור** תכיל את כתובת ה-IP של האתר Facebook, בעוד **כתובת היעד** תכיל את כתובת ה-IP של המחשב ממנו מתבצעת הגלישה.

גלו את כתובת ה-IP שלכם! נבצע זאת בשתי שיטות.

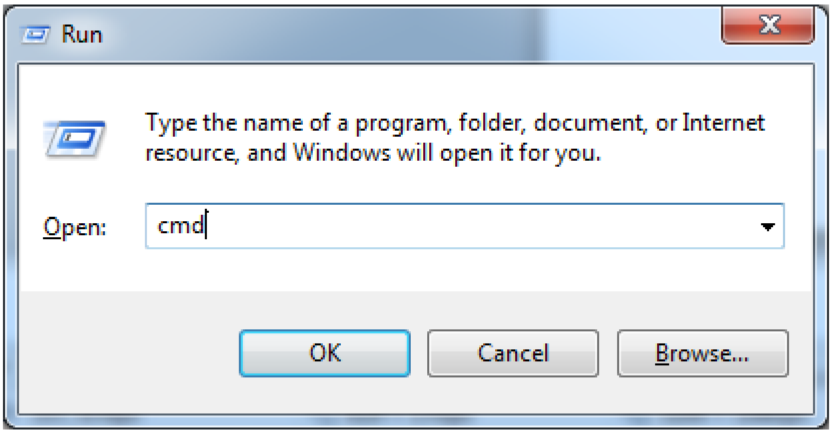
א. היכנסו אל האתר http://www.whatismyip.com.
ב. את שיטה ב' נלמד לאחר ביצוע התרגיל המודרך

תרגיל מודרך – מציאת כתובת ה-IP של אתר אינטרנט מסוים

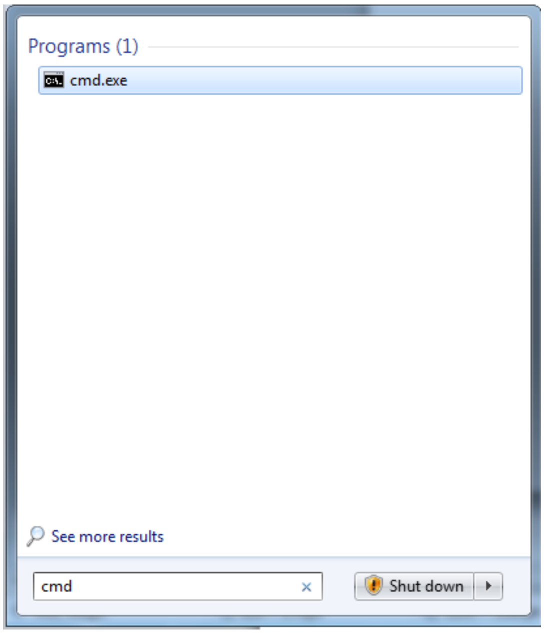
נאמר ונרצה לגלות את כתובת ה-IP של Facebook, או של Google. איך נוכל לעשות זאת?

דרך אחת לגלות את כתובת ה-IP של אתר אינטרנט היא להשתמש בכלי **ping**. לצורך כך, הריצו את **שורת הפקודה (Command Line)**, בה נשתמש רבות לאורך הספר. לחצו על צירוף המקשים (WinKey+R) בכדי להגיע לשורת ההפעלה. Winkey הינו המקש במקלדת שנמצא בין המקש Ctrl למקש Alt ומופיע עליו הלוגו של מערכת הפעלה Windows:

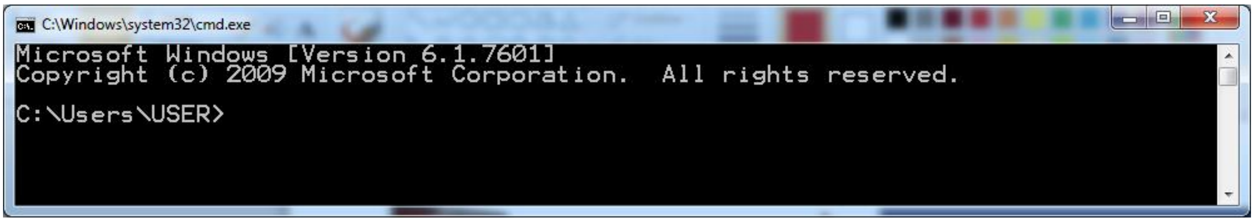
לחלופין, תוכלו לגשת אל Start (התחל) ולבחור באופציה Run (הפעל). הקישו בה את האותיות **cmd** ולחצו על OK:



לחלופין, תוכלו להקיש על מקש ה-WinKey, לכתוב **cmd** ולבחור בו כאשר הוא יוצא למסך:



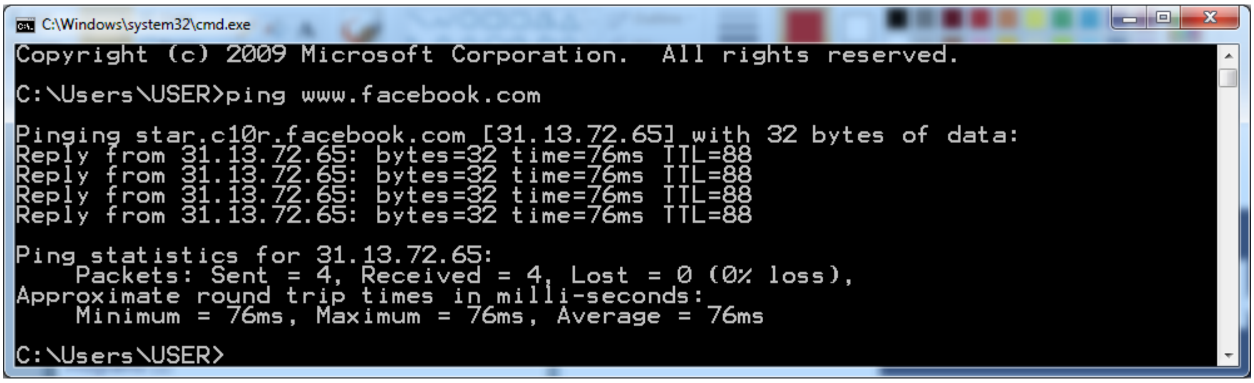
בשלב זה צפוי להיפתח חלון ה-cmd ובו שורת הפקודה:



כעת, הקלידו את הפקודה הבאה:

ping www.facebook.com

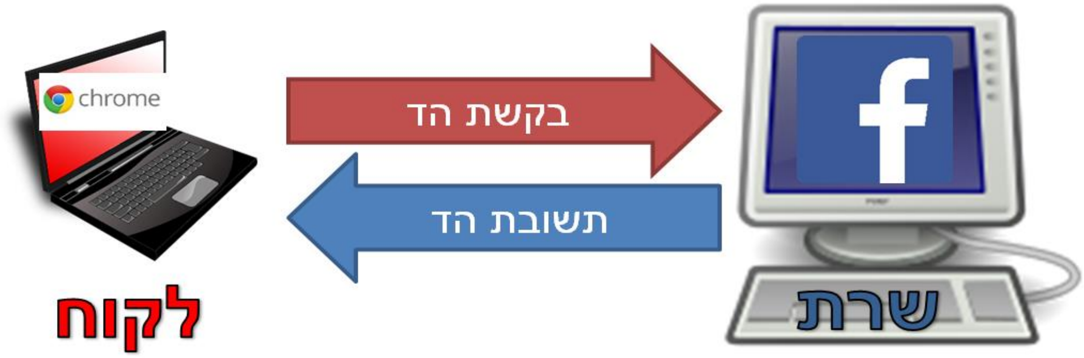
והקישו Enter. על המסך יודפסו נתונים שונים. בין השאר, תוכלו למצוא את כתובת ה-IP של Facebook:



בדוגמה זו, כתובת ה-IP הינה 31.13.72.65.[1]

כעת, נסו בעצמכם – מה היא כתובת ה-IP של Google?

הכלי **ping** לא נועד בכדי למצוא כתובות IP של אתרים. הוא נועד בכדי לבדוק האם מחשב בעל כתובת IP מסוימת מחובר לרשת, ולמדוד כמה זמן לוקח להודעה להישלח אל אותו מחשב, ואז לחזור אליי. על מנת לעשות זאת, הכלי **ping** שולח **בקשת הד** (כמו לצעוק במערה כדי לגלות כמה זמן לוקח להד לחזור אלינו). כאשר מחשב היעד מקבל את הבקשה הזו, הוא משיב מיד ב**תשובת הד**:



כך ניתן לבדוק האם המחשב בכתובת ה-IP המסוימת קיים, ולדעת כמה מהיר החיבור בינינו לבין אותו מחשב מרוחק. בהמשך הספר, בפרק שכבת הרשת, נלמד לעומק איך הכלי **ping** עובד – ואף תממשו אותו בעצמכם!

[1] מסיבות שלא נפרט כעת, כתובת ה-IP עשויה להשתנות. לכן, ייתכן שכשתריצו את הפקודה במחשב שלכם, תהיה ל-Facebook כתובת IP שונה.

גלו את כתובת ה-IP שלכם! נבצע זאת בשתי שיטות.

א. היכנסו אל האתר http://www.whatismyip.com.

ב. בתוך cmd, אותו למדנו כרגע לפתוח, כיתבו ipconfig וליחצו enter. תופיע לפניכם רשימת כל ממשקי הרשת שלכם. נצטרך לזהות מהו החיבור הפעיל שיש לכם, מכיוון שישנם ממשקי רשת רבים אך רק אחד פעיל. אתם מחוברים או באמצעות חיבור שנקרא Ethernet Adapter (אם ישנו כבל רשת שמחובר למחשב שלכם) או באמצעות חיבור שנקרא Wireless LAN Adapter. רק לאחד הממשקים הללו יש ערך בשדה ה- Default Gateway. זהו הממשק הפעיל שלכם. הערך שרשום בשדה ה-"IPv4 Address" הוא כתובת ה-IP שלכם. לדוגמה (מודגש בצהוב):

```
Wireless LAN adapter Wi-Fi:

   Connection-specific DNS Suffix  . : lan
   Link-local IPv6 Address . . . . . : fe80::30ff:3f09:cdf8:5330%12
   IPv4 Address. . . . . . . . . . . : 192.168.1.103
   Subnet Mask . . . . . . . . . . . : 255.255.255.0
   Default Gateway . . . . . . . . . : 192.168.1.1
```

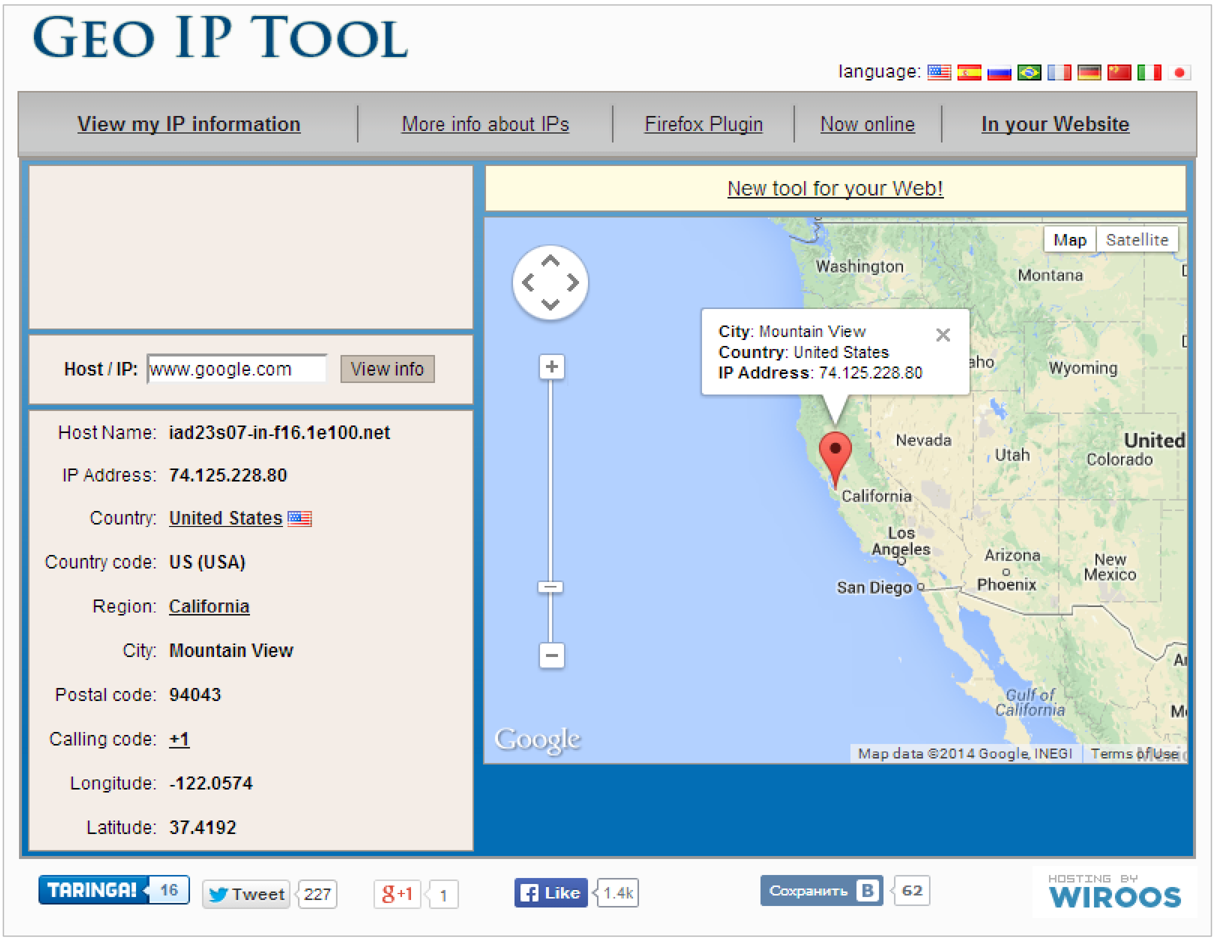
כפי שהנכם רואים, שתי השיטות מראות כתובות IP שונות. מדוע? נבין זאת כאשר נלמד מהו NAT. בקיצור רב- הכתובת שמצאנו בשיטה ב היא כתובת IP פרטית ואילו בשיטה א מצאנו את כתובת ה-IP החיצונית שבה משתמשת ספקית האינטרנט שלנו. שתי הכתובות זהות רק אם רכשתם מספקית האינטרנט כתובת IP חיצונית. כאמור, בשלב זה רק שימו לב להבדלים, בהמשך נדון בכך בפירוט.

GeoIP

עד כה דיברנו על שרתים, לקוחות וכתובות. כאשר גולשים לאתר אינטרנט מסוים, הודעת הבקשה צריכה להגיע בסופו של דבר אל השרת שיטפל בה. אותו שרת נמצא במקום כלשהו בעולם. האם ניתן לגלות היכן הוא נמצא?

ישנם מאגרי נתונים הכוללים מידע על המיקום הגיאוגרפי של כתובות IP. מכאן שבהינתן כתובת IP, ניתן לדעת באיזו מדינה ובאיזו עיר היא נמצאת. מאגרים אלו לא רשמיים ולא מדויקים, אך הם נותנים מענה נכון ברוב המקרים. בגלל האופן שבו בנוי האינטרנט, לא ניתן לייצר מאגר רשמי של מיפוי בין כתובת IP למקום גיאוגרפי, אבל על כך נלמד בהמשך הספר.

אתר לדוגמה שמאפשר למפות בין כתובת IP למיקום הגיאוגרפי שלה, הוא http://www.geoiptool.com.
נוכל להשתמש באתר זה כדי לגלות, למשל, את המיקום של Google:



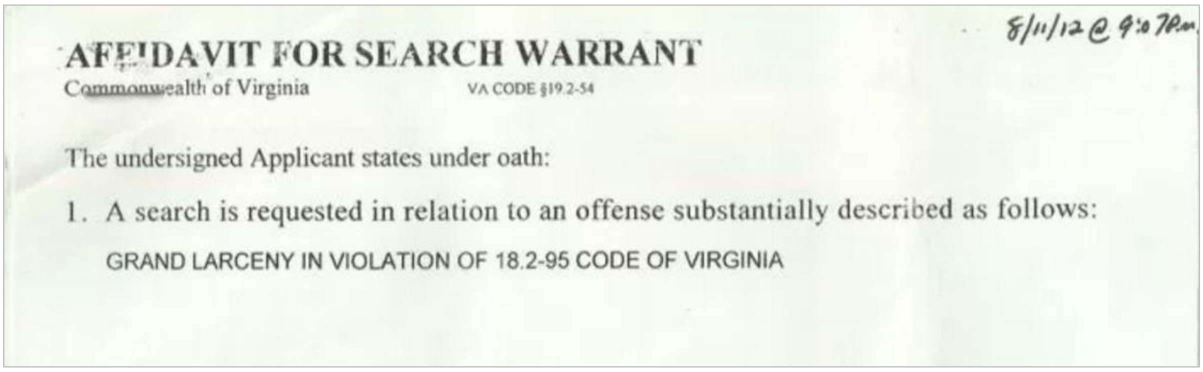
נסו זאת בעצמכם: מיצאו את המיקום הגאוגרפי של האתרים הבאים (כל אחד מהם נמצא במדינה אחרת):

- www.yahoo.com
- www.bbc.com
- www.newtoholland.nl
- www.yahoo.co.jp
- www.southafrica.co.za
- www.webawards.com.au

כאמור, המיפוי בין כתובת IP למיקום גיאוגרפי אינו מדויק. הכתבה הבאה ממחישה מדוע במקרים מסוימים זו עלולה להיות בעיה קשה: נניח שפושע מבצע פשע, ורשויות החוק מגלות את כתובת ה-IP שלו ומשתמשות בה על מנת לאתר את המיקום הפיזי שלו. אם המיפוי אינו מדויק, יש אפשרות שכתובת ה-IP של הפושע תמוקם ליד ביתו של אדם תמים. במקרה זה, רשויות החוק עלולות לפשוט על האזרח התמים, אשר אמנם

יוכל להסביר שחלה טעות, אך עדיין יסבול מהטרדה: דמיינו שהמשטרה פושטת על הבית שלכם עם צו חיפוש, רק משום שחלה טעות באיתור מיקום כתובת IP של פושע.

http://fusion.net/story/287592/internet-mapping-glitch-kansas-farm/


8/11/12 @ 9:07Pm
AFFIDAVIT FOR SEARCH WARRANT
Commonwealth of Virginia VA CODE §19.2-54

The undersigned Applicant states under oath:

1. A search is requested in relation to an offense substantially described as follows:
GRAND LARCENY IN VIOLATION OF 18.2-95 CODE OF VIRGINIA

צו חיפוש משטרתי, בחשד לגניבה, שניתן עקב מיקום IP שגוי (מתוך הכתבה)

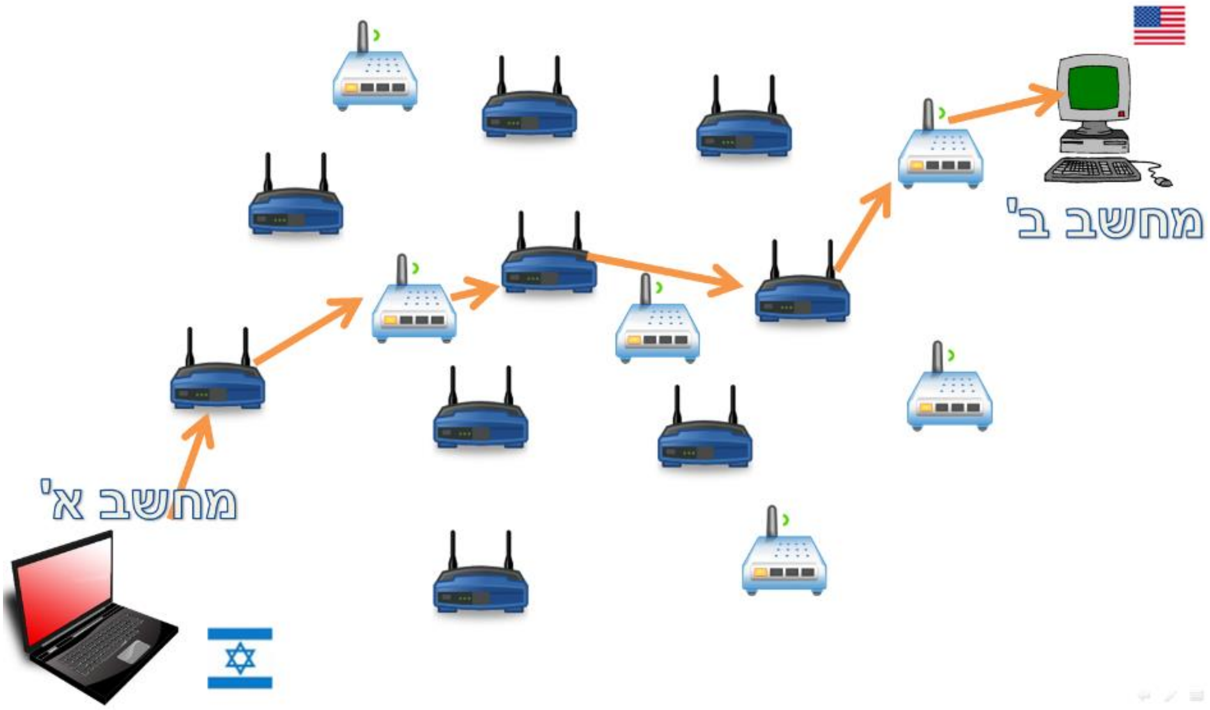
ענן האינטרנט

אם בדקתם מה המיקום הגיאוגרפי של Google, גיליתם שהוא נמצא בקליפורניה שבארצות הברית. איך קורה שהודעת הבקשה מהלקוח מגיעה עד לשרת שבארצות הברית, והודעת התגובה חוזרת בחזרה? ההודעות עוברות מרחק של חצי כדור הארץ הלוך ושוב! אם היינו צריכים לממן טיסה כה ארוכה עבור כל הודעה שנשלחת, השימוש באינטרנט היה עשוי להיות יקר מדי גם עבור ג'ף בזוס[2]. מעבר לכך, לא הגיוני שהמחשב שלנו יהיה מחובר בכבל הישר אל השרת של Google. אם כך, כיצד המידע מצליח להגיע אל Google שבארצות הברית?

האינטרנט הוא למעשה אוסף של הרבה רכיבים שמחוברים זה לזה. הודעה שנשלחת בין שני מחשבים המחוברים לרשת האינטרנט, עוברת בדרך בין הרבה רכיבים שמחוברים זה לזה באופן ישיר. כך כל רכיב מעביר את ההודעה הלאה אל הרכיב הבא, עד שהיא מגיעה אל היעד. העברה של הודעה בין רכיב אחד לרכיב אחר המחוברים ישירות נקראת **קפיצה (Hop)**.

---

[2] מייסד חברת Amazon והאיש העשיר ביותר בעולם, נכון לזמן עדכון ספר זה.



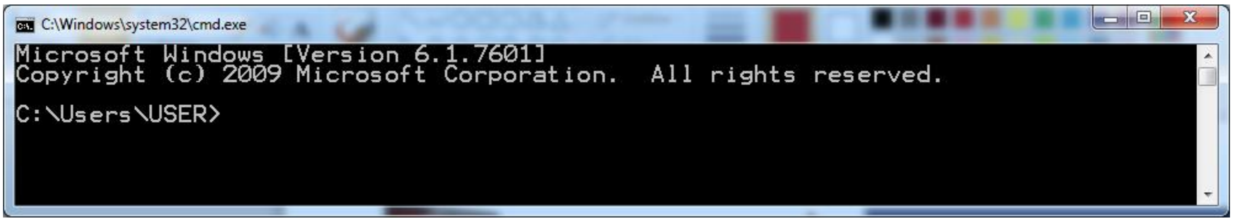
מה הם בדיוק הרכיבים האלו? איך הם עובדים? על שאלות אלו נענה בהרחבה בהמשך הספר. שימו לב שעל רכיבים אלו מוטלת משימה מורכבת: עליהם למצוא את הדרך הנכונה להגיע מהמקור (מחשב א' שנמצא בישראל) אל היעד (מחשב ב' שנמצא בארצות הברית). כל אותם רכיבים משתמשים בכתובות ה-IP של ההודעה בכדי לדעת לאן היא צריכה להגיע.

היות שהאינטרנט הינה רשת גדולה ומורכבת, לעיתים משתמשים בעברית במונח **ענן האינטרנט** כדי לתאר אותה. ענן האינטרנט מקבל מידע מצד אחד (למשל, מחשב של אדם בישראל), ומעביר אותו עד לצד השני (למשל, השרת של Google בארצות הברית).

תרגיל מודרך – מציאת הדרך בה עוברת הודעה

האם נוכל לגלות את המסלול שההודעה עוברת? אילו רכיבים מעבירים אותה בדרך בין המקור אל היעד?

הכלי **tracert** (קיצור של **traceroute** – מעקב אחר מסלול) מאפשר לנו לעשות זאת. לשם כך, הריצו שוב את **שורת הפקודה (Command Line)** אותה פגשנו קודם לכן:



הקלידו את הפקודה הבאה:

tracert -d www.google.com

```
C:\Users\barak>tracert -d www.google.com

Tracing route to www.google.com [172.217.16.164]
over a maximum of 30 hops:

  1     1 ms     1 ms     1 ms  192.168.1.1
  2     5 ms     1 ms     1 ms  176.230.251.233
  3     *        *        *     Request timed out.
  4     *        *        *     Request timed out.
  5     *        *        *     Request timed out.
  6     *        *        *     Request timed out.
  7     7 ms     4 ms     3 ms  82.102.132.78
  8   126 ms    98 ms    99 ms  80.179.166.38
  9   114 ms    99 ms    98 ms  72.14.216.121
 10   117 ms    97 ms   100 ms  216.239.57.31
 11    61 ms    61 ms    93 ms  216.239.63.255
 12   111 ms   100 ms   100 ms  172.217.16.164

Trace complete.
```

קיבלנו רשימה של כתובות ה-IP של הרכיבים שדרכם עברה ההודעה ששלחנו אל Google.[3]
כפי שאתם רואים, חלק מההנתיב אינו ידוע. מבלי להכנס לאופן בו עובד Tracert, שנלמד עליו כאשר נלמד על פרוטוקול IP, חלק מרכיבי הרשת לא סיפקו לנו מידע שמאפשר לדעת מה כתובת ה-IP שלהם. עם זאת, הצלחנו למצוא חלקים מהנתיב ועל הדרך לגלות את כמות ה-Hopים בין המחשב שלנו לבין Google (במקרה זה- 11 Hopים).

tracert מודיע שהוא לא ימשיך לעקוב אחרי המסלול של ההודעה אם המסלול ארוך יותר מ-30 קפיצות (hops). המרחק בין מקור ליעד נמדד לעיתים על ידי מספר הקפיצות ביניהם (כלומר, כמות הרכיבים שההודעה עברה בדרך מהמקור ליעד).

כמו כן, הכלי **tracert** מציג מדידות של זמן ב-MS (מילי שניות) שלקח להגיע מהמקור אל כל רכיב בדרך ובחזרה ממנו. עבור כל אחד מהרכיבים, מוצגות שלוש מדידות כדי להגביר את האמינות של המדידה. ההבדלים בין משך הזמן שלוקח להגיע לכל רכיב, יכולים להעיד על המרחק הגאוגרפי בין הרכיבים השונים.

[3] בגלל הדרך בה עובד האינטרנט, הדרך הזו נכונה עבור הודעה מסוימת אך עשויה להשתנות עבור הודעה אחרת. על כך נעמיק בהמשך הספר.

למשל, אם נראה לפתע הפרש גדול מאוד בין זמנים, נוכל לשער שעברנו יבשת. עבור הפרשים קטנים במיוחד, כנראה שהרכיבים סמוכים יחסית זה לזה.

נסו בעצמכם להריץ את הכלי **tracert** בכדי לגלות את המסלול שהודעה עוברת מהמחשב שלכם אל Facebook.

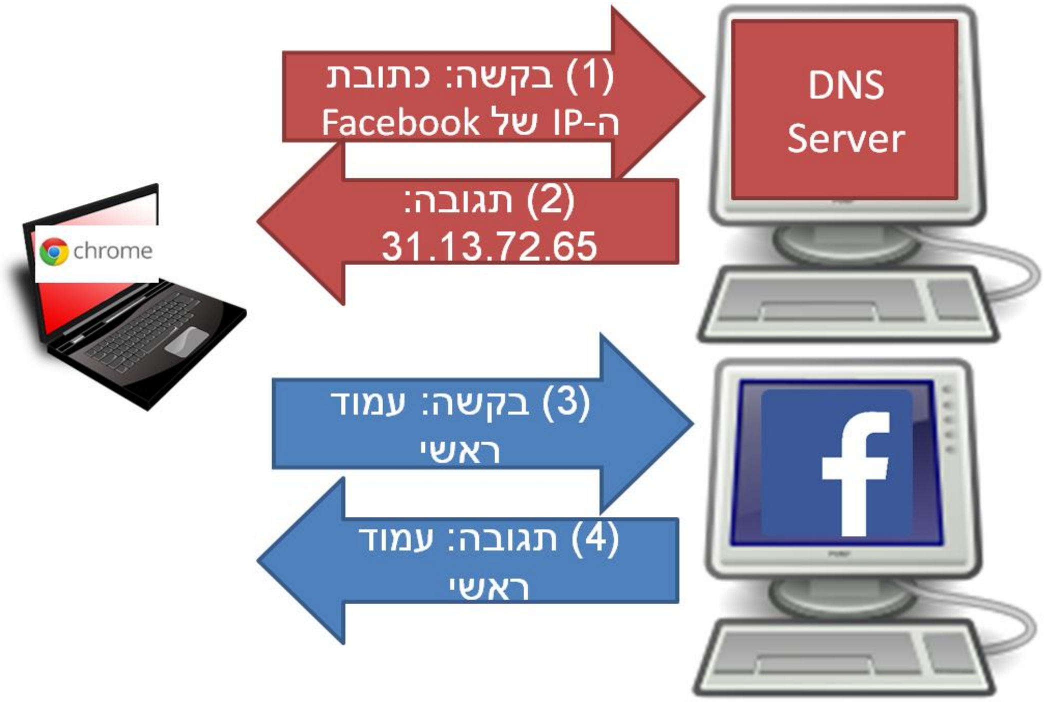
בהמשך הספר, נלמד להבין איך הכלי **tracert** פועל מאחורי הקלעים, וגם תבנו כלי כזה בעצמכם!

DNS

קודם לכן, דיברנו על כתובות IP, ואמרנו שאלו הכתובות בעזרתן הודעות נשלחות באינטרנט. עם זאת, בתור משתמשים, כמעט ולא נתקלנו בכתובות IP. על מנת לזהות שרתים באינטרנט, השתמשנו בעיקר בכתובות מהצורה "www.facebook.com". כתובות אלו נקראות **שמות דומיין (Domain Name)**, או **שמות מתחם**. הסיבה העיקרית לכך שמעדיפים להשתמש בשמות דומיין, היא שלבני אדם קל יותר לזכור אותם מאשר כתובות IP.

הדבר דומה לדרך בה אנו שומרים מספרי טלפון: כאשר אנו מתקשרים לאדם, בדרך כלל נעדיף שהטלפון יזכור עבורנו את המספר שלו, וכך נעדיף לחייג אל "משה כהן" מאשר לזכור את המספר 054-1234567. עם זאת, כשם שהטלפון צריך בסופו של דבר להשתמש במספר בכדי להתקשר למשה, כך גם המחשב צריך לדעת את כתובת ה-IP של מחשב היעד בכדי לפנות אליו. לשם כך, יש צורך בדרך לתרגם בין השניים: בין השם "משה כהן" למספר 054-1234567, או בעולם האינטרנט: בין שם הדומיין "www.facebook.com" לכתובת ה-IP הרלוונטית, למשל 31.13.72.65.

באינטרנט, הדרך לתרגם שמות דומיין לכתובות IP היא באמצעות המערכת **DNS (Domain Name System)**. ההמרה הזו, בין שמות דומיין אל כתובות IP, מתרחשת בכל פעם שאנו מציינים שם דומיין – בין אם כשאנו גולשים לאתר בדפדפן, ובין אם כשאנו משתמשים בכלים כמו **ping** או **tracert**. המחשב מבין בשלב ראשון מה היא כתובת ה-IP של היעד, ורק לאחר מכן שולח אליו את ההודעה. כך למשל, כאשר אנו גולשים אל Facebook, המחשב קודם צריך להבין מה היא כתובת ה-IP של אתר Facebook, ולאחר מכן לבקש ממנו להציג את העמוד:

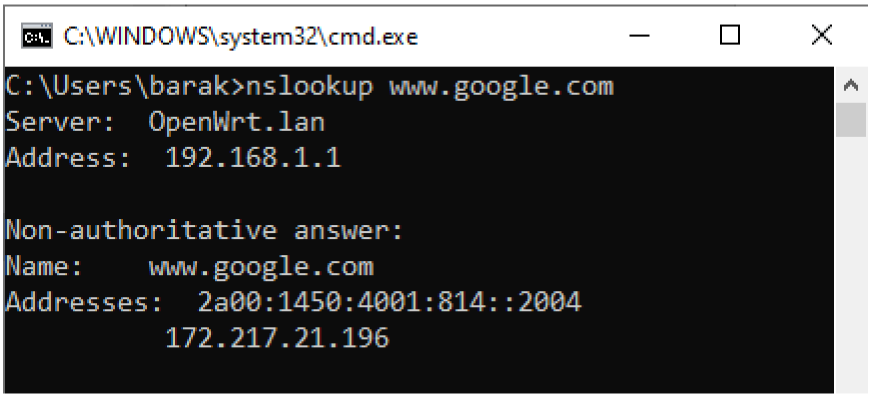


שימו לב כי הבקשה למציאת כתובת IP (המוצגת ב**אדום** בשרטוט, בחלק העליון) נשלחת אל שרת ה-DNS, בעוד בקשת העמוד (המוצגת ב**כחול** בשרטוט, בחלק התחתון) נשלחת אל כתובת ה-IP של Facebook. על הדרך שבה עובדת מערכת ה-DNS, נלמד בהמשך הספר.

תרגיל מודרך – תרגום בין שמות דומיין לכתובות IP

על מנת להשתמש במערכת ה-DNS כדי לתרגם שמות דומיין לכתובות IP, נוכל להשתמש בכלי **nslookup**. הריצו את שורת הפקודה, אתם כבר יודעים כיצד לעשות זאת. כעת, הריצו את הפקודה הבאה:

nslookup www.google.com



על המסך מוצגת כתובת ה- IP השייכת ל-Google במועד כתיבת מילים אלו, 172.217.21.196. ייתכן שקיבלתן כתובת IP שונה, זאת מכיוון שכתובות IP של Domain Names עשויות להשתנות עם הזמן. כמו שאנשים עוברים כתובת, גם Domain Names עוברים כתובת. נסו לגלוש בדפדפן אל כתובת ה-IP שקיבלתם. כלומר, במקום לכתוב בשורת הכתובת "www.google.com", כתבו 172.217.21.196 (או את הכתובת שאתם קבלתם, אם היא שונה מכתובת זו). האם הגעתם אל Google?

כעת, השתמשו בכלי **nslookup** כדי לגלות כתובות של אתרים נוספים. האם כתובות ה-IP שמחזיר **nslookup** זהות לכתובות שמוצגות כאשר אתם משתמשים בכלי **ping** עבור אותם האתרים?

התשובה היא שלעיתים כן ולעיתים לא. לחברות גדולות יש בדרך כלל מספר רב של שרתים, לכן הן עושות שימוש במספר כתובות IP, תוך ניתוב העומס לשרת פנוי. לכן אפשרי שתקבלו כתובות שונות. לעומת זאת אם תבצעו את ההשוואה בין תוצאות של ping ותוצאות של nslookup על אתר צנוע כגון www.themarker.com, צפוי שתקבלו תוצאות זהות.

איך עובד האינטרנט – סיכום

בפרק זה התחלנו לענות על השאלה – כיצד האינטרנט עובד? ניסינו להבין מה קורה כאשר מקלידים בדפדפן את הכתובת "www.facebook.com". הבנו כי הדפדפן הינו תוכנת **לקוח**, ששולחת הודעות **בקשה** אל **השרת** של Facebook, שבתורו מחזיר **תגובה**. למדנו כי להודעות באינטרנט, בדומה להודעות דואר, יש כתובת מקור וכתובת יעד, ושלכתובות אלו קוראים באינטרנט **כתובות IP**. כמו כן, למדנו שניתן לעיתים להבין מכתובת IP מה המיקום הגיאוגרפי שלה, באמצעות **GeoIP**.

לאחר מכן, למדנו על **ענן האינטרנט**, והבנו שהמחשב שלנו לא מחובר ישירות לאתר של Facebook, אלא לרכיב המחובר לרכיבים אחרים שיוצרים רשת של הרבה רכיבים שמעבירים את ההודעה שלנו מהמחשב ועד לשרת היעד. הכרנו את הכלי **traceroute** שהציג את הדרך שעוברת הודעה מהמחשב שלנו אל Facebook, ונעזרנו בכלי **visual traceroute** בכדי להציג את הדרך הזו על מפת העולם. לסיום, הבנו שיש צורך בתרגום בין **שמות דומיין**, כמו www.facebook.com, אל כתובות IP, דבר הנעשה באמצעות מערכת ה-**DNS**.

במהלך הפרק, הכרנו מושגים רבים וכן כמה כלים ראשונים. למדנו להשתמש ב**שורת הפקודה (Command Line)**, הכרנו את הכלים **nslookup , tracert ,ping** וכן את **visual traceroute**. עם זאת, רק התחלנו לענות על השאלות שלנו, וציירנו תמונה כללית בלבד. בעיקר, נפתחו בפנינו שאלות חדשות – איך עובדת מערכת ה-DNS? מה זו בדיוק כתובת IP? מי הם הרכיבים האלו שמחברים את המחשבים באינטרנט, וכיצד הם פועלים? איך עובד traceroute? איך Facebook מחזיר לדפדפן תשובה? על כל שאלות אלו, כמו גם שאלות רבות אחרות, נענה בהמשך הספר.

פרק 2

תכנות ב-Sockets

בפרק הקודם התחלנו את מסענו בעולם המופלא של התקשורת תוך הצגת אופן הגלישה שלנו באינטרנט. בפרק זה נלך שלב אחד קדימה וננסה לתפוס את המקום של כותבי האפליקציה. עד סוף הפרק, תדעו לכתוב בעצמכם צ'אט בסיסי, ואף שרת שיודע לבצע פקודות בהתאם לבקשות שהוא מקבל.

שימו לב: לאורך הפרק נציג דוגמאות קוד ב-Python. עליכם להקליד ולהריץ שורות קוד אלו במקביל לתהליך הקריאה, על מנת להבין את החומר באופן מלא.

שרת-לקוח

צורת התקשורת הנפוצה ביותר כיום באינטרנט נקראת **שרת-לקוח** (**Client-Server**). בצורת תקשורת זו קיים רכיב כלשהו המשמש כשרת (Server), דהיינו מספק שירות כלשהו, ורכיב כלשהו הנקרא לקוח (Client) אשר משתמש בשירות המסופק. לדוגמה, כאשר אתם פונים אל האתר של YouTube בכדי לצפות בסרטון, הלקוח הוא המחשב שלכם (או הדפדפן שבמחשב שלכם), והשרת הוא אותו שרת של YouTube. במקרה זה, הדפדפן שלכם שולח בקשה לשרת של YouTube, והשרת מחזיר לכם תשובה, כפי שנראה בשרטוט הבא:

באופן דומה, במידה שאתם גולשים מהטלפון הנייד שלכם אל כתבה ב-TheMarker, הלקוח הינו הטלפון שלכם (או הדפדפן שבטלפון שלכם), והשרת הינו השרת של TheMarker.

מי הלקוח ומי השרת בדוגמה הבאה?

בשרטוט להלן ניתן לראות דוגמה בה טלפון חכם "גולש" למיקרו-גל על מנת לדעת עוד כמה זמן נשאר לחימום של השניצל האהוב. נסו לחשוב בעצמכם – מי הלקוח ומי השרת?

בדוגמה זו, המיקרו-גל הינו השרת, שכן הוא נותן שירות של הערכה של משך הזמן הנותר עד לתום תהליך חימום השניצל. הלקוח הינו הטלפון החכם, אשר משתמש בשירות של המיקרו-גל.

במהלך הפרק, נבנה אפליקציה בתצורת שרת-לקוח ונבצע זאת תוך שימוש במודול של פייתון בשם **socket**. נכתוב בעצמנו לקוח, ולאחר מכן נכתוב גם שרת. הלקוח והשרת שנכתוב ידעו לשלוח ולקרוא מידע אחד לשני, בדומה לצ'אט בו יש שני משתתפים. לאחר מכן, הלקוח והשרת ידעו לבצע פעולות בהתאם לתקשורת ויהפכו למורכבים יותר ויותר ככל שהפרק יתקדם.

אז מה זה בעצם Socket ?

Socket הוא נקודת קצה של חיבור בין שני רכיבים. כאשר אנחנו רוצים להעביר מידע בין שני מחשבים, אנחנו צריכים לקשר ביניהם, בדומה להעברת מידע בין שני צדדים של צינור.

מידע שנכנס מצד אחד של הצינור יצא בצד השני. כל קצה של הצינור נקרא כאמור **Socket**. הצינור יכול לחבר בין תוכנות הנמצאות על אותו רכיב (לדוגמה: Word והדפדפן על המחשב שלכם), או בשני רכיבים נפרדים (למשל: הדפדפן שלכם והשרת של Google). במידה ששתי תוכניות מעוניינות להעביר ביניהן מידע, נקודת הכניסה של מידע למערכת ונקודת היציאה שלו ממנה יכונו Socket.

על מנת שיהיו לנו צינור, אנו זקוקים למספר דברים:

- לבנות את הצינור – פעולה זו יבצעו השרת והלקוח שלנו.
- התחלה וסוף – לצינור המעביר מים יש נקודת התחלה ונקודת סיום. הדבר נכון גם לגבי צינור המעביר מידע בין שתי תוכנות. על מנת להגדיר את נקודת ההתחלה והסיום, אנו זקוקים לכתובות, עליהן נפרט בעוד רגע קצר.
- גודל – לכל צינור יש קוטר ואורך, לפיהם ניתן לדעת כמה מידע יכול הצינור להכיל ברגע נתון. ה"קוטר" של הצינור שלנו, הוא ה-Socket, הינו בית (byte) אחד – אנו נעביר ב-Socket זרם של בתים מצד לצד.

כתובות של Socket

כאמור, לכל צינור יש נקודת התחלה ונקודת סיום. עלינו להגדיר את נקודות ההתחלה והסיום של ה-Socket שלנו. לשם כך, נצטרך להשתמש במזהה של התוכנה שאיתה נרצה לתקשר. ל-Socket יש שני מזהים:

- מזהה הרכיב – מזהה את הרכיב (מחשב, שרת וכדומה) שאיתו נרצה לתקשר.
- מזהה התוכנית (התהליך) – מזהה את התוכנה על הרכיב (למשל – שרת המייל, שרת ה-Web) שאיתה נרצה לתקשר.

ניתן לדמות זאת לשליחת מכתב דואר בין שתי משפחות הגרות בשכונה של בתים רבי קומות. מזהה הרכיב הינו מזהה הבניין של המשפחה – למשל "רחוב הרצל בעיר תל אביב, בית מספר 1". מזהה התהליך הוא מזהה הדירה הספציפית בבניין, למשל "דירה 23".

ה-Socket מבצע עבורנו את השירות של רשות הדואר – אנו, כמתכנתים, לא צריכים להיות מודעים לכל הפעילות שמתרחשת ברשת בכדי להעביר את ההודעה מצד אחד של ה-Socket (תיבת הדואר של משפחה אחת) לצד השני של ה-Socket (תיבת הדואר של משפחה שנייה). בעוד בעולם הדואר הקישור בין משפחה לבין התיבה הינו כתובת הבניין ומספר הדירה, הקישור בין Socket ובין התוכנה שמנהלת אותו (כלומר – התוכנה שאתחלה את אובייקט ה-Socket) הינו **כתובת IP** ומספר **פורט (Port)**. את המושג "כתובת IP" הכרנו מוקדם יותר בספר, ואת המושג **"פורט"** אנו פוגשים עכשיו לראשונה. את שניהם נזכה להכיר בצורה מעמיקה יותר בהמשך. בשלב זה, חשוב שנכיר שפורט יכול להיות מספר בטווח שבין 0 ל-65,535.

תרגיל 2.1 מודרך – הלקוח הראשון שלי

כעת נכתוב את הלקוח הראשון שלנו, באמצעות פייתון.

פיתחו קובץ py חדש, קיראו לו בשם כלשהו. שימו לב!

א. אל תקראו לו בשם בעברית ואל תשמרו אותו בתיקיה שחלק מהנתיב אליה הוא בעברית. סביבת הפיתוח PyCharm לא מסתדרת טוב עם תווים בעברית

ב. אל תקראו לקובץ socket.py! אם תקראו לו כך, למעשה "תדרסו" את המודול socket של פייתון ולא תוכלו לעשות בו שימוש

הדבר הראשון שעלינו לעשות הוא לייבא את המודול של **socket** לפייתון:

```
import socket
```

כעת, עלינו ליצור אובייקט מסוג **socket**. נקרא לאובייקט זה בשם my_socket:

my_socket = socket.socket()

הערה: ל-**socket()** ניתן לתת פרמטרים שכרגע איננו מרחיבים עליהם, אך נעשה זאת בהמשך.

כעת יש לנו אובייקט **socket**. בשלב הבא, נדאג ליצור חיבור בינו לבין שרת אחר. לשם כך נשתמש במתודה **connect**. קודם לכן הסברנו שהכתובת של Socket כוללת כתובת IP ומספר פורט. ובהתאם לכך, המתודה **connect** מקבלת tuple (היזכרו בשיעורי הפייתון) שמכיל כתובת IP ומספר פורט. לצורך הדוגמה, נתחבר לשרת שכתובת ה-IP שלו היא "127.0.0.1", לפורט 8820:

my_socket.connect((‘127.0.0.1, 8820))

כעת יצרנו חיבור בינינו לבין התוכנה שמאזינה בפורט 8820 בשרת בעל הכתובת "127.0.0.1".
הכתובת המיוחדת "127.0.0.1" מציינת שהמידע נשלח אל המחשב שלנו ולא יוצא מכרטיס הרשת, ולכן היא מאפשרת ללקוח להתחבר לשרת שנמצא איתו על אותו המחשב. יפה, יש לנו את הצינור – ועכשיו אפשר לשלוח ולקבל דרכו מידע!
על מנת לשלוח מידע אל התוכנה המרוחקת, נשתמש במתודה **send**:

my_socket.send(‘Omer’.encode())

כפי שאתם שמים לב לאחר הטקסט שאנחנו שולחים מופיעה המתודה encode. מהי ומה תפקידה? בפייתון גרסה 3, שאנחנו משתמשים בו, ה-Socket דורש לקבל פרמטר מסוג 'bytes', שהינם רצף בינארי. אי אפשר לשלוח מחרוזת, שהינה מטיפוס 'str'.
כפי שאפשר לראות מההמחשה הבאה, מתודת ה-encode הופכת טיפוס מחרוזת לטיפוס מסוג 'bytes':

```
>>> msg = 'Omer'
>>> type(msg)
<class 'str'>
>>> byte_msg = msg.encode()
>>> type(byte_msg)
<class 'bytes'>
```

כדי להבין יותר טוב את פעולת המתודות encode ו-decode (שמיד נשתמש בה) אתם מוזמנים לקרוא את המדריך הבא, אך קריאת המדריך הינה העשרה ואינה חובה:

https://pymotw.com/2/codecs/

נחזור להתמקד בפיתוח הלקוח שלנו. אם כך, שלחנו הודעה אל השרת.
על מנת לקבל מידע, נוכל להשתמש במתודה **recv** (קיצור של receive):

data = my_socket.recv(1024).decode()

המשתנה data הוא גם מטיפוס bytes, כיוון שכל מה שיוצא מה-Socket הוא מטיפוס זה, ולכן נבצע לו decode, שהיא המתודה ההפוכה ל-encode.

Print(‘The server sent: ‘ + data)

ולבסוף, "נסגור" את אובייקט ה-**socket** שיצרנו בכדי לחסוך במשאבים:

my_socket.close()

זהו, פשוט וקל! ראו כמה קצר כל הקוד שכתבנו:

```
import socket

my_socket = socket.socket()
my_socket.connect(("127.0.0.1", 8820))

my_socket.send("Omer".encode())
data = my_socket.recv(1024).decode()
print("The server sent " + data)

my_socket.close()
```

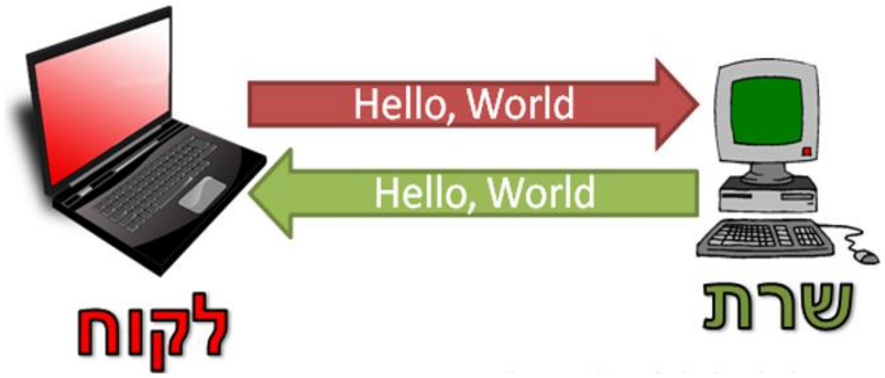
בשבע שורות קוד אלו יצרנו אובייקט **socket**, התחברנו לשרת מרוחק, שלחנו אליו מידע, קיבלנו ממנו מידע, הדפסנו את המידע שהתקבל וסגרנו את האובייקט. שימו לב עד כמה נוח לכתוב קוד כזה בשפת פייתון.

תרגיל 2.2 – הפעלת הלקוח במקביל לשרת הדים: שליחה וקבלה של מידע

בתרגיל זה תפעילו את הלקוח שיצרתם בתרגיל הקודם, כאשר השרת כבר מומש עבורכם. השרת משכפל כל מידע שתשלחו לו, ושולח אותו אליכם בחזרה, כמו הד. כך למשל, אם תכתבו אל השרת את המידע:

"Hello, World"

השרת יענה: "Hello, World":



הורידו את השרת מהכתובת:

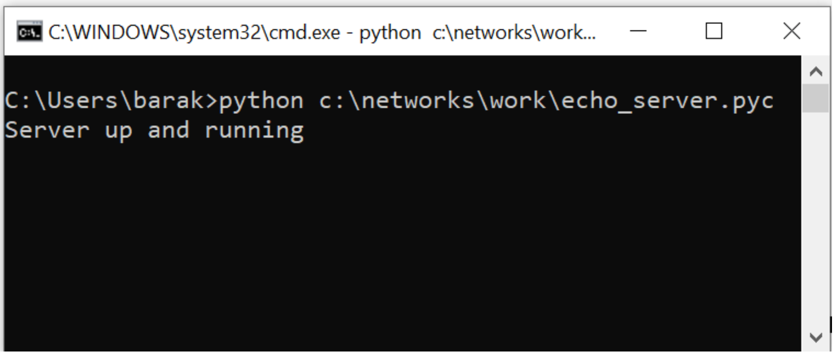
https://data.cyber.org.il/networks/echo_server.pyc

שמרו את הקובץ למיקום הבא (לא הכרחי אבל מומלץ למען הסדר הטוב):

C:\networks\work\echo_server.pyc

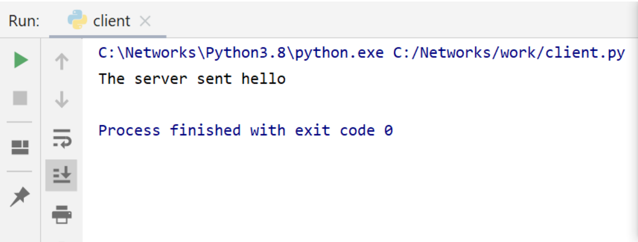
על מנת להריץ את השרת, היכנסו אל ה-Command Line, והריצו את שורת הפקודה:

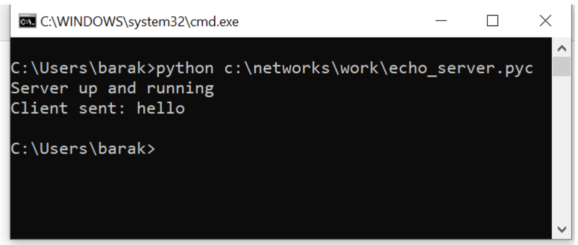
python C:\networks\work\echo_server.pyc



הרגע הרצתם את שרת "הדים" (echo). כאמור, שרת זה מחקה כמו הד את המידע שהוא מקבל. השרת מאזין לחיבורים בפורט קבוע – פורט 8820.

כעת הריצו באמצעות pycharm את הלקוח שכתבתם בסעיף הקודם (שימו לב- הכרחי להפעיל את השרת *לפני* שמפעילים את הלקוח) –





ה-client וה-server מתקשרים זה עם זה!

באג נפוץ:

השרת שלנו "תופס" את פורט 8820 ומאזין לו. אם ננסה להריץ שרת נוסף בו זמנית, לפני שהשרת הראשון סיים לרוץ, השרת השני לא יצליח לתפוס את פורט 8820 שכבר תפוס על ידי השרת הראשון. במקרה זה נקבל שגיאה כזו:

```
C:\WINDOWS\system32\cmd.exe

C:\Users\barak>python c:\networks\work\echo_server.pyc
Traceback (most recent call last):
  File "echo_server.py", line 19, in <module>
    main()
  File "echo_server.py", line 6, in main
    s_socket.bind(('0.0.0.0', 8820))
OSError: [WinError 10048] Only one usage of each socket address (protocol/network address/port) is normally permitted

C:\Users\barak>
```

פשוט סיגרו את השרת הראשון והריצו שוב, הפעם התוכנית תעבוד.

לאחר שהצלחתם לשלוח הודעה לשרת ולקבל ממנו תשובה, כיתבו סקריפט פייתון שמבקש מהמשתמש הודעה, שולח אותה לשרת "הדים", מקבל מהשרת את התשובה ומדפיס אותה למשתמש.

דגשים:

- על מנת לקרוא מידע מהמשתמש, תוכלו להשתמש בפונקציה **input()**.
- הפונקציה recv היא blocking – כלומר, אם קראתם לפונקציה אך עדיין לא הגיע מידע, התוכנית תיעצר עד שיגיע מידע חדש.
- במידה שהחיבור התנתק, recv תחזיר מחרוזת ריקה (''). יש לבדוק מקרה זה בקוד[4].
- חשוב לסגור את ה-socket בסוף השימוש (כלומר – לקרוא ל-**close()**). כך מערכת ההפעלה תוכל לשחרר את המשאבים שהיא הקצתה עבור ה-Socket שיצרנו.

תרגיל 2.3 מודרך – השרת הראשון שלי

אז הצלחנו ליצור לקוח שמתחבר לשרת, שולח אליו מידע ומקבל ממנו מידע. כעת הגיע הזמן לכתוב גם את השרת.

מוקדם יותר, יצרנו לקוח ששולח לשרת את שמו, לדוגמה: "Omer". כעת, נגרום לשרת לקבל את השם שהלקוח שלח, ולענות לו בהתאם. לדוגמה, השרת יענה במקרה זה: "Hello Omer".

הדרך לכתיבת שרת דומה מאוד לכתיבה של לקוח. גם הפעם, הדבר הראשון שעלינו לעשות הוא לייבא את המודול של **socket** לפייתון:

```
import socket
```

כעת, עלינו ליצור אובייקט מסוג **socket**. נקרא לאובייקט זה בשם server_socket:

```
server_socket = socket.socket()
```

[4] מקרה זה לא מתאר מצב שבו השרת באמת ניסה לשלוח מחרוזת ריקה וקרא למתודה send(''). קבלת מחרוזת ריקה מעידה שהחיבור בין הלקוח אל השרת התנתק.

בשלב הבא, עלינו לבצע קישור של אובייקט ה-**socket** שיצרנו לכתובת מקומית. לשם כך נשתמש במתודה **bind**. מתודה זו, בדומה ל-**connect** אותה פגשנו קודם, מקבלת tuple עם כתובת IP ומספר פורט. נשתמש בה, לדוגמה, כך:

```
server_socket.bind((‘0.0.0.0’, 8820))
```

בצורה זו יצרנו קישור בין כל מי שמנסה להתחבר אל הרכיב (במקרה זה, המחשב) שלנו (זו המשמעות של הכתובת "0.0.0.0", נרחיב על כך בהמשך) לפורט מספר 8820 – אל האובייקט server_socket.

אך יצירת הקישור הזו עדיין אינה מספקת. על מנת לקבל חיבורים מלקוחות, נצטרך להאזין לחיבורים נכנסים. לשם כך נשתמש במתודה **listen**:

```
server_socket.listen()
```

שימו לב שהמתודה **listen** מקבלת פרמטר מספרי[5], אנחנו לא מזינים ערך כדי להשתמש בערך ברירת המחדל של המתודה, שהינו 0.

כדי לקבל חיווי על מצב השרת שלנו, נדפיס למסך הודעה. כמובן שהשרת יעבוד היטב גם ללא ההודעה הזו, אך היא תסייע לנו לדעת שהכל עובד כשורה:

```
print("Server is up and running")
```

בשלב זה אנו מחכים לחיבורים נכנסים. עכשיו, עלינו להסכים לקבל חיבור חדש שמגיע. לשם כך, נשתמש במתודה **accept**:

```
(client_socket, client_address) = server_socket.accept()
```

המתודה **accept** הינה blocking – כלומר, הקוד "יקפא" ולא ימשיך לרוץ עד אשר יתקבל בשרת חיבור חדש.

כמו שניתן להבין, המתודה **accept** מחזירה tuple שכולל שני משתנים: אובייקט **socket** שנוצר בעקבות תקשורת עם לקוח שפנה, ו-tuple נוסף שכולל את הכתובת של הלקוח שפנה אלינו.

נבצע הדפסה כדי לקבל חיווי על כך שלקוח פנה אל השרת שלנו (שוב, זה רק חיווי למשתמש, השרת אינו נזקק לשורה זו כדי לפעול באופן תקין):

```
print("Client connected")
```

---

[5] פרמטר זה לא מציין כמה חיבורים ה-socket יכול לקבל, אלא כמה חיבורים יכולים לחכות בתור במערכת ההפעלה מבלי שביצעו להם **accept** (למתודה זו נגיע בעוד רגע).

לאחר שלקוח יפנה אל השרת שלנו, נגיע למצב שיש לנו את client_socket ובאמצעותו נוכל לתקשר עם הלקוח. לדוגמה, נוכל לקבל ממנו מידע. בהקשר זה, נרצה לקבל את השם של הלקוח ולהדפיס אותו:

```
data = client_socket.recv(1024).decode()
print("Client sent: " + data)
```

נוכל גם לשלוח אליו מידע:

```
reply  = 'Hello ' + data
client_socket.send(reply.encode())
```

המימוש זהה למעשה לקבלה ושליחה של מידע בצד הלקוח, ומשתמש במתודות **send** ו-**recv** אשר פגשנו קודם לכן. כשנסיים את התקשורת עם הלקוח, עלינו לסגור את החיבור איתו:

```
client_socket.close()
```

כעת, נוכל להתפנות ולתת שירות ללקוח אחר. לחלופין, נוכל לסגור את אובייקט ה-**socket** של השרת:

```
server_socket.close()
```

כשאנו קוראים למתודה **close** אנו מודיעים למערכת ההפעלה שסיימנו להשתמש באובייקט ה-**socket** שיצרנו, ולכן מערכת ההפעלה יכולה לשחרר את המשאבים שקשורים אליו. בין השאר, אחד המשאבים הוא מספר הפורט שמערכת ההפעלה הקצתה עבור ה-Socket. לדוגמה, בשרת שיצרנו לעיל, מערכת ההפעלה הקצתה את הפורט 8820 עבור server_socket. באם תוכנה אחרת תבקש להשתמש בפורט הזה, היא לא תוכל לעשות זאת, שכן הוא תפוס. רק לאחר שהפקודה:

```
server_socket.close()
```

תרוץ, הפורט ייחשב שוב "פנוי" ויוכל להיות בשימוש בידי תוכנה או Socket אחר.

לעיתים, תוכנה תסיים לרוץ מבלי שהיא קראה למתודה **close**. דבר זה יכול לנבוע ממספר סיבות – למשל, מתכנת ששכח לקרוא למתודה זו. דוגמה נוספת, שייתכן שתקרה במהלך עבודתכם על התרגיל, היא שהתוכנה (או הסקריפט) תקרוס לפני שהקריאה ל-**close** תתרחש. במקרה זה, מערכת ההפעלה לא יודעת שהפורט שוחרר, ולכן מתייחסת אליו כתפוס. בשלב זה, תוכנית חדשה לא תוכל להשתמש במספר הפורט הזה. דבר זה יהיה נכון גם, למשל, אם תנסו להריץ שוב את הסקריפט שלכם לאחר שהוא קרס. על מנת להתגבר על בעיה זו, תוכלו לשנות את הפורט בו אתם משתמשים. זכרו לשנות את מספר הפורט גם בצד הלקוח וגם בצד השרת.
בנוסף, לאחר שמתבצעת קריאה ל-**close** עשוי לעבור זמן מסוים עד שמערכת ההפעלה באמת תשחרר את הפורט. ייתכן שתיתקלו בכך במהלך העבודה על התרגיל.

שימו לב לא להתבלבל בין שני אובייקטי ה-**socket** השונים שיצרנו. האובייקט הראשון שיצרנו, שנקרא בדוגמה server_socket, הינו האובייקט של השרת. תפקידו רק להאזין ללקוחות חדשים. ברגע שלקוח פונה אל serve_socket מוקם מולו client_socket, שמאפשר את התקשורת הספציפית עם הלקוח. ניתן להמשיל זאת למורה בכיתה, שעונה לשאלות של תלמידים. סגירה של client_socket היא כמו לסיים לענות לתלמיד מסויים, במקרה זה המורה יכול לענות לשאלה של תלמיד אחר. סגירה של server_socket היא כמו שהמורה יוצא מהכיתה, כעת הוא כבר לא מנמצא במצב בו הוא מאזין לשאלות התלמידים.

הנה, יש לנו שרת עובד. להלן הקוד המלא של השרת הפשוט שכתבנו:

```
import socket

server_socket = socket.socket()
server_socket.bind(("0.0.0.0", 8820))
server_socket.listen()
print("Server is up and running")

(client_socket, client_address) = server_socket.accept()
print("Client connected")

data = client_socket.recv(1024).decode()
print("Client sent: " + data)

reply = "Hello " + data
client_socket.send(reply.encode())

client_socket.close()
server_socket.close()
```

כעת הריצו ב-Pycharm את השרת ו*רק לאחר מכן* את הלקוח. הנה הם מתקשרים זה עם זה במחשב שלכם!

המלצה חשובה – כפי שנכתב במבוא לספר זה, מומלץ להשתמש בעורך PyCharm עבור עבודה עם פייתון. באמצעות עורך זה ניתן לעבוד בצורה נוחה יותר עם קוד, וכן לדבג אותו באמצעות breakpoints, דבר הצפוי לסייע לכם רבות בפתרון התרגילים. ניתן להיעזר במצגת על PyCharm, הזמינה בכתובת הבאה:

http://data.cyber.org.il/python/1450-3-02.pdf

תרגיל 2.4 קישור בין שני מחשבים באותה רשת מקומית

עד כה הרצנו את השרת והלקוח שלנו מקומית על אותו מחשב. נחמד, אבל למה לא נריץ אותם על שני מחשבים שונים? לצורך זה, תצטרכו שני מחשבים כמובן. שלבי העבודה:

א. וודאו שבשני המחשבים מותקנת סביבת העבודה של פייתון

ב. בחרו מחשב כלשהו, מעכשיו נקרא לו "שרת". מיצאו את כתובת ה-IP שלו. איך עושים זאת? הכנסו ל-cmd וכיתבו ipconfig. לפניכם תופיע כתובת ה-IP שלכם, היכן שרשום "IPV4 address". טיפ: הכתובת צריכה להתחיל במספר 10 או 192:

```
Wireless LAN adapter Wi-Fi:

   Connection-specific DNS Suffix  . : lan
   Link-local IPv6 Address . . . . . : fe80::30ff:3f09:cdf8:5330%12
   IPv4 Address. . . . . . . . . . . : 192.168.1.103
   Subnet Mask . . . . . . . . . . . : 255.255.255.0
   Default Gateway . . . . . . . . . : 192.168.1.1
```

ג. וודאו ששני המחשבים מחוברים לאותה רשת ביתית. איך עושים זאת? גשו למחשב השני. מעכשיו נקרא לו "לקוח". הכנסו ל-cmd וכיתבו ping ולאחר מכן את כתובת ה-IP של השרת. לדוגמה, נניח שכתובת ה-IP של השרת היא 192.168.1.109:

```
C:\Users\barak>ping 192.168.1.109

Pinging 192.168.1.109 with 32 bytes of data:
Reply from 192.168.1.109: bytes=32 time=327ms TTL=64
Reply from 192.168.1.109: bytes=32 time=127ms TTL=64
Reply from 192.168.1.109: bytes=32 time=42ms TTL=64
Reply from 192.168.1.109: bytes=32 time=57ms TTL=64
```

אם קבלתם הודעות דומות, שמתחילות במילה Reply, סימן שאתם מחוברים ועל אותה רשת. אם לא, בידקו ששני המחשבים מחוברים לאותה רשת WiFi. שימו לב- ייתכן שתצטרכו לבטל גם בשרת וגם בלקוח את האנטיוירוס, שעלול לחסום הן את ה-ping והן את העברת המידע בין השרת והלקוח.

ד. במחשב הלקוח הכנסו לקוד הלקוח, ובמקום הכתובת 127.0.0.1 הזינו את כתובת ה-IP של השרת.

זהו, זה הכל :)

תרגיל מסכם 2.6 – שרת פקודות בסיסי

(נכון, מספר התרגיל 2.6 אינו רציף למספר התרגיל הקודם, הסיבה לכך שתרגיל זה נקרא "2.6" היא כדי לשמור על תאימות עם גרסאות קודמות של ספר הלימוד).

בתרגיל הראשון התבקשתם לכתוב לקוח, ובתרגיל השני התבקשתם לכתוב שרת. בתרגיל זה עליכם לכתוב הן את השרת, והן את הלקוח. חלק מהאתגר בתרגיל הינו כתיבת פרוטוקול למשלוח הודעות בין השרת והלקוח.

שלד של פתרון ניתן למצוא בלינק:

https://data.cyber.org.il/networks/ex26.zip

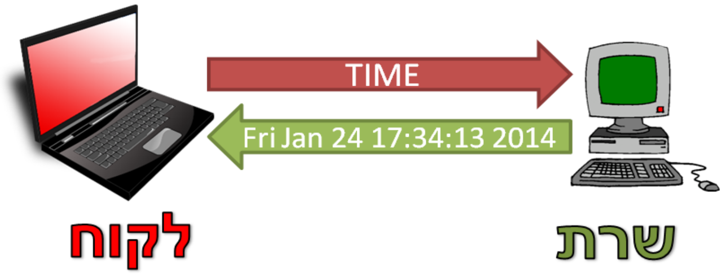
<u>שלב 1:</u>

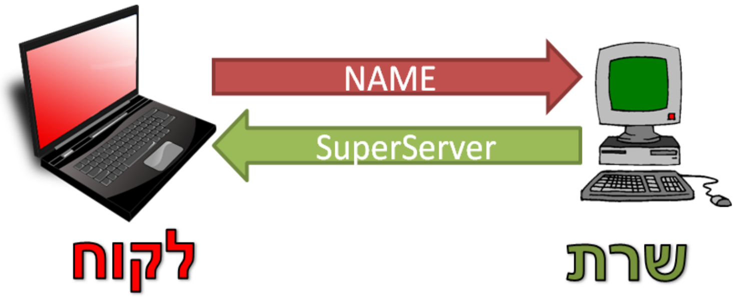
עליכם לכתוב מערכת שרת-לקוח, כאשר השרת מבצע פקודות שהלקוח שולח אליו, ומחזיר ללקוח תשובה בהתאם. על כל בקשה של הלקוח להיות באורך של ארבעה בתים בדיוק. אורך התגובה יכול להיות שונה בהתאם לבקשה.

להלן רשימת הבקשות שיש לתמוך בהן:

- TIME – בקשת הזמן הנוכחי. על השרת להגיב עם מחרוזת שכוללת את השעה הנוכחית אצלו.
  - היעזרו במודול datetime שמובנה בפייתון.
- WHORU – בקשת שם השרת. על השרת להגיב עם מחרוזת שמייצגת את שמו. השם יכול להיות כל מחרוזת שתבחרו.
- RAND – בקשת מספר רנדומלי. על השרת להגיב עם מספר רנדומלי שנע בין הערכים 1 ל-10.
  - היעזרו במודול random המובנה בפייתון.
- EXIT – בקשת ניתוק. על השרת לנתק את החיבור עם הלקוח.

להלן דוגמאות לתקשורת:





לפני שאתם ניגשים לכתוב את הקוד, בצעו תכנון ראשוני. חשבו מה בדיוק תממשו בצד הלקוח ומה בצד השרת, ונסו לצפות מראש בעיות בהן תתקלו.

בצד הלקוח – ראשית, עליכם לבקש מהמשתמש לבחור באחת מהפקודות שצוינו לעיל.

טיפ: היעזרו בפונקציה **input**. לאחר מכן, שלחו את הבקשה לשרת, קבלו את התשובה והציגו אותה למשתמש.

בצד השרת – עליכם לקבל חיבור מהלקוח, להבין את הבקשה שלו ולהגיב אליה בהתאם. שימו לב שהשרת צריך להמשיך לקלוט בקשות מהלקוח עד שהוא מקבל פקודת EXIT. במילים אחרות, כל עוד השרת לא קיבל EXIT עליו להמשיך להשיב לפקודות של הלקוח. טיפ: לולאת while תהיה שימושית למדי בצד השרת.

<u>שלב 2:</u>

תרגול כתיבת פרוטוקול – האם בשלב 1 הלקוח והשרת שלכם ביצעו socket.recv(1024)? כעת, צרו פרוטוקול שמאפשר לשלוח מהשרת ללקוח ומהלקוח לשרת הודעות באורך שונה. יכולת זו תשמש אתכם בתרגילים בהם לא תוכלו להניח שאורך ההודעה מהשרת ללקוח הוא 1024 בתים או מספר קבוע כלשהו.

נסקור שיטה לעשות זאת, באמצעות שליחת אורך ההודעה.

הרעיון הכללי הוא שכל צד יודיע לצד השני מה אורך ההודעה שהוא שולח. לדוגמה:

- צד א': אני שולח הודעה באורך 12
- צד א': hello world!

בשלב זה צד ב' קולט את ההודעה הראשונה, מפענח אותה ומבין שכדי לקלוט את ההודעה הבאה עליו לבצע recv(12)

- צד ב': אני שולח הודעה באורך 2
- צד ב': OK

בשלב זה צד א' קולט את ההודעה הראשונה, מפענח אותה ומבין שכדי לקלוט את ההודעה הבאה עליו לבצע recv(2)

הדרך לממש את הרעיון היא פשוט להצמיד את המספר שמהווה אורך ההודעה אל ההודעה עצמה. לדוגמה, צד א ישלח את הרצף הבא:

12hello world!

חשוב להבין, שצד ב' יכול לקבל את ההודעה הזו בחלקים. אם צד ב' יבצע

socket.recv(2)

אז הוא ימשוך מה-Socket את הרצף "12", בעוד יתר ההודעה נותר ב-Socket. יתר ההודעה לא נמחק מה-Socket, המידע נותר שם עד לפעם הבאה בה צד ב' יבצע recv.

כיצד נראית יצירת ההודעה "12Hello world!"? נסו לבצע זאת בעצמכם.

פתרון:

הריצו את הפקודות הבאות, בצעו בסוף print למשתנה message:

```
message = "Hello World!"
length = len(message)
message = str(length) + message
```

כעת אפשר לבצע encode למשתנה message, שהינו מטיפוס מחרוזת, ולשלוח את התוצאה באמצעות send:

```
my_socket.send(message.encode())
```

ומה יבצע הצד הקולט? יהיה עליו לבצע recv פעמיים. בפעם הראשונה תיקלט אורך ההודעה, במקרה שלנו 12. בפעם השניה, תיקלט ההודעה עצמה:

```
length = my_socket.recv(2).decode()
message = my_socket.recv(int(length)).decode()
```

האם סיימנו? או שאתם שמים לב לבעיה?

הקוד הנ"ל יעבוד היטב אם אורך ההודעה הוא מספר בן 2 ספרות. במקרה זה הצד הקולט יקלוט את שתי הספרות, 12 בדוגמה שלנו, וימיר אותן בהצלחה לאורך של הודעה. אולם, מה אם אורך ההודעה אינו נכנס במספר בן 2 ספרות?

ניקח את הדוגמה של ההודעה "OK" אותה צד ב' משיב.

לפי הקוד שלנו, צד ב' ישלח

2OK

צד א' יקלוט את 2 התווים הראשונים בהודעה, במקרה זה "2O". לא ניתן להמיר את הערך הזה למספר (שימו לב זהו התו O לא הספרה 0) והתוכנית תסתיים בקריסה.

המסקנה היא שהשרת והלקוח צריכים לתאם ביניהם מראש כמה ספרות ישמשו לייצוג של אורך הודעה ולדבוק תמיד באורך הודעה זה. כלומר אם קבענו שאורך ההודעה ייוצג על ידי 2 ספרות, אז נצטרך לשמור על הכללים הבאים:

- לא שולחים הודעה שאורכה למעלה מ-99 (המספר המקסימלי שניתן לייצוג על ידי 2 ספרות).
- אם אורך ההודעה הוא בן ספרה אחת בלבד, נוסיף 0 מוביל. לדוגמה, במקום "2" נשלח "02".

כך, הצד הקולט יכול להיות בטוח שאם הוא מוציא מה-socket הודעה באורך 2, ההודעה תמיד מכילה את האורך של יתר ההודעה.

כדי להפוך את הספרה "2" לרצף "02" נשתמש במתודה zfill, קיצור של "zero-filling", מילוי באפסים. המתודה הזו פועלת על מחרוזות, ומקבלת כפרמטר מספר שהוא אורך המחרוזת שיש להגיע אליו לאחר הוספת אספים.

כעת הקוד בצד השולח נראה כך:

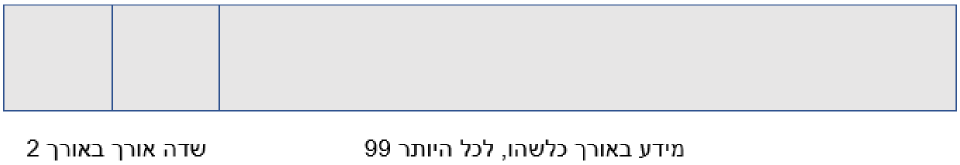
```
length = str(len(message))
zfill_length = length.zfill(2)
message = zfill_length + message
```

נסו זאת. הזינו לתוך message ערך התחלתי כגון "hi" ותראו שלאחר קטע הקוד הזה הערך הינו "02hi".

כעת הצד המקבל יכול לבצע recv(2) ללא חשש. שני הצדדים תיאמו ביניהם מראש שלא משנה מה, אורך ההודעה תמיד יהיה באורך 2.

מה שיצרנו נקרא בשפה המקצועית "שדה אורך". שדה אורך הוא שדה נפוץ למדי בפרוטוקולים, כיוון שהוא מאפשר לשני הצדדים לדעת בדיוק כמה מידע עליהם לקלוט.

הפרוטוקול שלנו נראה כעת כך:



זהו, כעת יש לכם את הכלים להוסיף את שדה האורך לפרוטוקול שלכם!

שלב 3:

עלינו לכתוב שרת יציב, כלומר גם אם הלקוח שולח "זבל", לשרת שלנו אסור לקרוס. בידקו את יציבות השרת באמצעות הודעות מסוגים שונים. בעיות איתן השרת צריך להתמודד:

- הלקוח שלח פקודה שאינה מוכרת לשרת
- שדה האורך של הלקוח אינו כולל ספרות

במקרים הללו השרת פשוט צריך לענות בהודעה כגון "Wrong protocol". אם שדה האורך לא היה תקין השרת אינו יודע כמה בתים נותרו ב-socket ויש אפשרות שנותר שם "זבל", שימנע מההודעה הבאה להגיע בצורה תקינה. לכן במקרה זה השרת יוציא מה-socket כ-1024 בתים, בלי לעשות איתם דבר, בתקווה שהוצאת ה"זבל" תאפשר להודעה הבאה להתקבל תקין.

גם הלקוח צריך להיות יציב: עלול להיות מצב שבו עקב שגיאה כלשהי שדה האורך אינו תקין, אסור ללקוח לקרוס במקרה זה.

שלד הפתרון יכווין אתכם לפתרון ומה נדרש לממש.

בהצלחה!

תרגיל אתגר 2.7 – שרת פקודות מתקדם- טכנאי מרוחק (אתגר)

עד כה בניתם שרתים ולקוחות שתקשרו עם זה עם זה וביצעו פעולות פשוטות. בתרגיל זה עליכם לתכנן מערכת שרת-לקוח, ולאחר מכן לממש אותה. המערכת תאפשר לטכנאי לתקשר עם מחשב מרוחק ולבצע עליו פעולות שונות. הפעם, עליכם לתכנן כיצד ייראו הפקודות והתשובות, ואיך התרגיל מציין זאת עבורכם.

השרת, המחשב המרוחק, יבצע פקודות בהתאם לבקשת הלקוח (הטכנאי), כמו בתרגיל הקודם. כלומר: הלקוח ישלח פקודה לשרת, השרת יקבל את הפקודה, יעבד אותה וישלח את התשובה אל הלקוח. כל ההודעות בין השרת והלקוח יעברו יחד עם שדה אורך, כפי שראינו בתרגיל 2.6. הפעם מומלץ להשתמש בשדה אורך גדול יותר, באורך 4.

שימו לב, לרשותכם קבצים המכילים את שלד התרגיל הן בשרת והן בלקוח. עליכם למלא בתוכן את הפונקציות, לפי התיעוד שנמצא בפונקציות ולפי ההדרכה בהמשך.

https://data.cyber.org.il/networks/ex27.zip

להלן הפקודות שעליכם לממש:

DIR - 2.7.1

הטכנאי שלנו חושד שהקבצים של תוכנה כלשהי לא נמצאים במקום או שלא כל הקבצים נמצאים. הטכנאי מעוניין להציג תוכן של תיקייה מסוימת במחשב המרוחק. לדוגמה, הצגת רשימת הקבצים שבתיקייה C:\Cyber. תמכו בפקודת DIR – השרת ישלח ללקוח את התוכן של תיקייה מבוקשת. לדוגמה:

DIR C:\Cyber

לחיפוש על פי שמות קבצים, ניתן להשתמש במודול **glob**.

לדוגמה, להצגת כל הקבצים בתיקייה C:\Cyber, ניתן להריץ:

```
import glob
files_list = glob.glob(r’C:\Cyber\*.*’)
```

DELETE - 2.7.2

הטכנאי הגיע למסקנה שאחד הקבצים אינו צריך להיות בתיקייה ויש למחוק אותו. צרו בשרת ובלקוח אפשרות להורות על מחיקת קובץ כלשהו, באמצעות פקודת DELETE, לדוגמה:

DELETE C:\Cyber\blabla.txt

למחיקת קובץ ניתן להשתמש במודול OS, לדוגמה:

```
import os
os.remove(r'C:\Cyber\blabla.txt')
```

COPY – 2.7.3

כעת הטכנאי שלנו רוצה להעתיק קובץ כלשהו לתיקייה עליה הוא עובד. הקובץ המבוקש נמצא בתיקייה אחרת במחשב אליו מתבצעת ההעתקה. הוסיפו תמיכה בפקודת COPY (לדוגמה: העתק את הקובץ C:\Cyber\1.txt אל C:\Cyber\2.txt). אין הכוונה לשליחת הקובץ אל הלקוח, אלא לביצוע הפעולה על השרת בלבד. במקרה זה, השרת יחזיר ללקוח האם הפעולה הצליחה או לא. לדוגמה:

COPY C:\Cyber\1.txt C:\Cyber\2.txt

להעתקה של קבצים, ניתן להשתמש במודול **shutil**.

לדוגמה, להעתקת הקובץ C:\1.txt אל C:\2.txt, ניתן להריץ:

```
import shutil
shutil.copy(r'C:\1.txt', r'C:\2.txt')
```

EXECUTE – 2.7.4

הטכנאי שלנו רוצה לבדוק שהתוכנה עובדת עכשיו היטב. תמכו בפקודת EXECUTE אשר תגרום להפעלת תוכנה אצל השרת (לדוגמה – הרצה של תוכנת Word). במקרה כזה, על השרת להגיב ללקוח האם הפעולה הצליחה או נכשלה.

EXECUTE notepad.exe

על מנת להריץ תוכנות, נוכל להשתמש במודול **subprocess**.

לדוגמה, כדי להריץ את notepad, נוכל לבצע:

```
import subprocess
subprocess.call(r'C:\Windows\system32\notepad.exe')
```

ישנן תוכנות כמו notepad שניתן להריץ גם ללא ציון הנתיב המלא, אולם נדרוש מהמשתמש נתיב מלא, כדי להקל על השרת לבדוק אם התוכנה קיימת לפני הנסיון להריץ אותה.

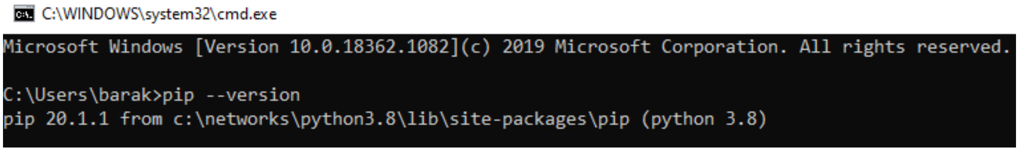
TAKE_SCREENSHOT – 2.7.5

הטכנאי שלנו רוצה לקבל תצלום מסך של המחשב המרוחק. עליכם לתמוך בכך בשרת ובלקוח.

- המשתמש בלקוח יוכל להקליד פקודות, שהלקוח יעביר לשרת.
- תמכו בפקודה TAKE_SCREENSHOT, שתגרום לכך שהשרת יבצע צילום מסך וישמור את הקובץ במחשב השרת, במיקום קבוע לפי בחירתכם
- להלן הדרכה שתסייע לכם להכניס לסקריפט הפייתון שלכם קוד שמבצע צילום מסך:
  1. בידקו שהמודול pip מותקן אצלם. כדי לבדוק, הכנסו לתוך cmd וכיתבו

```
pip --version
```

אם הכל בסדר, יופיע לכם מסך כדוגמת הסמך הבא:



2. אם pip אינו מותקן אצלכם מסיבה כלשהי, הורידו אותו מהקישור הבא:

https://bootstrap.pypa.io/get-pip.py

בתוך cmd הכנסו לתיקיה בה שמרתם את קובץ ה-py שהורדתם.

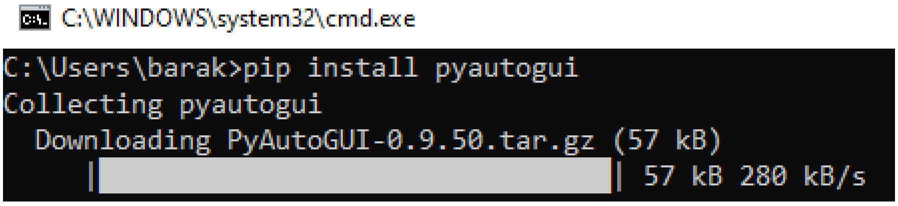
הריצו את קובץ ה-py:

python get-pip.py

3. כעת כאשר יש לנו pip אפשר להתקין את המודול pyautogui. בתוך cmd (לא בתוך פייתון) כיתבו:

pip install pyautogui

כמו בדוגמה הבאה, והמודול יותקן לכם:



4. כעת בתחילת סקריפט הפייתון כיתבו:

import pyautogui

5. השתמשו בקוד הבא כדי לצלם את המסך ולשמור את התמונה לקובץ (הקוד שומר את התמונה למיקום: C:\screen.jpg), שנו אותו לפי הצורך:

image = pyautogui.screenshot()

image.save(r'C:\screen.jpg')

<u>2.7.6</u>

צילום המסך נוצר בשרת, אולם כדי שהטכנאי יוכל להשתמש בו, עליכם לשלוח אותו ללקוח. פקודת SEND_PHOTO צריכה לגרום לשרת לשלוח את הקובץ המבוקש ללקוח. אין צורך להוסיף פרמטר לפקודה, כיוון שהשרת יודע היכן הוא שמר את התמונה.

שימו לב – צילום המסך גדול למדי ושדה האורך שקבענו עלול שלא להיות מספיק. לכן נשלח קודם כל את האורך של הקובץ. הדרכה:

נניח שגודל התמונה הוא 2000000 (שני מליון) בתים. ראשית נשלח ללקוח את גודל התמונה. לפני כן צריך לבוא שדה אורך. נניח שכמות הספרות בשדה האורך היא 4, כלומר האורך המקסימלי של הודעה יכול להיות 9999 בתים. האורך של המספר 2000000 הוא 7 ספרות, לכן שדה האורך יהיה 0007.
לאחר שדה האורך ושדה גודל התמונה השרת ישלח את המידע עצמו.
הלקוח יקלוט 4 בתים, יפענח אותם ויידע שעליו לקלוט עוד 7 בתים. הלקוח יקלוט 7 בתים נוספים ויידע שאורך הקובץ הצפוי הוא 2000000.
בשלב זה הלקוח יפתח קובץ עם סיומת jpg, לדוגמה screenshot.jpg.
עם סיום העברת התמונה, הלקוח יסגור את הקובץ jpg. זהו, התמונה שמורה בצד הלקוח!

2.7.7
לבסוף הטכנאי שלנו רוצה להודיע לשרת לסגור את החיבור. תמכו בפקודת EXIT, שתגרום לשרת לסגור את הסוקט מול הלקוח

טיפים לפתרון התרגיל

1. שימו לב שפתיחת קובץ התמונה בשרת צריכה להיות במצב קריאה של קובץ בינארי לא טקסטואלי. לאחר הקריאה, התמונה כבר נמצאת בקידוד בינארי ולכן אין צורך לבצע לה encode לפני השליחה ללקוח. גם בצד הלקוח, כמובן, יש לפתוח את הקובץ שלתוכו תישמר התמונה במצב בינארי
2. שימו לב שבפייתון, על מנת לייצג מחרוזת שכוללת את התו backslash ("\"), נצטרך לכתוב את התו פעמיים – כלומר "\\". זאת היות שהתו backslash יכול להיות חלק מתווים מיוחדים, כגון "\n" שמסמל התחלה של שורה חדשה. לחלופין, ניתן להשתמש באות r לפני הגדרת המחרוזת. שימו לב לדוגמה הבאה:

```
>>> not_what_we_mean = "C:\file.txt"
>>> not_what_we_mean
'C:\x0cile.txt'
>>> what_we_mean_1 = "C:\\file.txt"
>>> what_we_mean_1
'C:\\file.txt'
>>> what_we_mean_2 = r"C:\file.txt"
>>> what_we_mean_2
'C:\\file.txt'
>>> what_we_mean_1 == what_we_mean_2
True
```

3. אם אתם מכירים את השימוש ב-Exceptions בפייתון, מומלץ להשתמש בהם.

תרגיל 2.8 – קרוסלת הפורטים (אתגר)

קרדיט: אייל אבני

הגיע הזמן לעוף על גבי קרוסלת הפורטים! עד כה כתבנו שרת ולקוח שעבדו מעל socket עם פורט קבוע. כעת, "נסחרר" קצת את העניינים.

עליכם לכתוב שרת ולקוח, כך שלאחר כל שליחה של מידע – הם יחליפו מספרי פורטים, וגם יתחלפו בתפקידים.

<u>דוגמה:</u>

צד א' מתחיל בתור שרת, שמאזין בפורט כלשהו, נניח 8111.

צד ב' מתחיל בתור לקוח, שמתחבר לצד א' בפורט 8111, ושולח לו הודעה. ההודעה היא מחרוזת כלשהי שהמשתמש הזין (input). בסיום ההודעה שולח צד ב' מספר פורט, לדוגמה 8055, ומתנתק.

צד ב' הופך להיות שרת ופותח האזנה לפורט 8055.

צד א' הופך להיות לקוח, מתחבר לצד א' בפורט 8055, שולח לו הודעת תשובה, מחרוזת תשובה מאת המשתמש. ביום ההודעה שולח צד א' מספר פורט, לדוגמה 8071, ומתנתק.

חוזר חלילה על השלבים הקודמים, עד שהמשתמש מזין הודעת "exit".

חשבו על כל מקרי הקצה שעשויים לקרות, וכתבו קוד שמסביר מה המקרים הללו וכיצד התמודדתם איתם.

כמובן שאת כל ההתכתבות יש להציג על המסך, כולל הפורט שבו נעשה כרגע שימוש. לדוגמה:

Side A listening to port 8111

Side B connecting to port 8111

Side B: Hello

Side B disconnected

Side B listening to port 8055

Side A connecting to port 8055

Side A: Hi There

Side A disconnected

Side B listening to port 8071

הצלחתם? יפה מאוד. כעת חזרו על התהליך, אך בלי להמתין לכך שהמשתמש יזין הודעה. כלומר, צרו הודעה קבועה שעוברת בין צד א' לצד ב' וחזרה, ובדקו כמה מהר אתם יכולים להעביר אותה. גרמו לתוכנה שלכם להיות מהירה, נסו לבדוק גבולות ולהבין היכן הדברים מפסיקים לעבוד ומדוע.

סיכום תכנות ב-Sockets

בזאת מגיע לסיומו פרק תכנות ב-Sockets. במהלך הפרק למדנו מהו מודל שרת-לקוח, והסברנו מהו Socket. לאחר מכן, בנינו יחד לקוח ראשון ושרת ראשון. בהמשך, מימשתם בעצמכם לקוח ושרת "הדים", פיתחתם שרת פקודות בסיסי וכן שרת פקודות מתקדם. לאורך הפרק הצלחתם ליצור מספר מימושים במודל שרת-לקוח, ותרגלתם הן את יכולות הפייתון שלכם והן עבודה מול Socket.

כעת, יש ברשותכם כלי עוצמתי. אתם יכולים להשתמש ב-Socket כדי לתקשר מנקודת קצה אחת לנקודת קצה שנייה, ולכתוב כל שרת-לקוח שתרצו. עם זאת, למדנו רק חלק מהאפשרויות שניתן לבצע באמצעות Socket. לא דיברנו על סוגים שונים של Sockets, וכן ראינו רק מודל שבו יש שרת יחיד ולקוח יחיד. עדיין לא ראינו כיצד ניתן לממש שרת שמספק שירות למספר לקוחות במקביל. על זאת ועוד נלמד בפרקים הבאים. הידע שרכשנו בפרק זה יסייע לנו בהמשך הספר ללמוד קונספטים חדשים, לראות דוגמאות וכן לכתוב בעצמנו קוד שיממש אותן.

פרק 3

Wireshark ומודל חמש השכבות

בפרק הקודם השתמשנו ב-Sockets כדי לתקשר בין שני מחשבים שונים. בתור מתכנתים, היה לנו מאוד נוח להשתמש ב-Sockets על מנת לדבר עם מחשב מרוחק: חוץ מלתת את כתובת המחשב המרוחק, הפורט שעליו הוא מאזין, להתחבר ל-Socket המרוחק ולשלוח אליו מידע – לא היינו צריכים לעשות כלום. אך עלינו לזכור ש-Socket הוא בסך הכל ממשק שמאפשר לנו לתקשר בקלות ומפשט לנו את כל תהליך התקשורת שקורה בפועל (כלומר – איזה מידע באמת עובר ברשת כשאנחנו משתמשים ב-Sockets). אז מה קורה בפועל כשאנחנו מדברים עם מחשב מרוחק? בכך עוסק פרק זה.

בפרק זה נבין טוב יותר כיצד נראית התעבורה שיוצאת מכרטיס הרשת שלנו בדרכה אל מחשב אחר בעולם או נכנסת אליו, ותוך כדי נציג את מודל חמש השכבות (מודל לוגי שמחלק את הפעולות של מערכת תקשורת לחמישה חלקים שונים) ומדוע צריך אותו. בתחילת הפרק נראה שימוש בסיסי בכלי חזק מאוד שנקרא **Wireshark**, שיאפשר לנו לחקור ולהבין מה זה בדיוק "המידע שעובר דרך כרטיס הרשת שלנו", ולגלות דברים חדשים שלא היינו יכולים לגלות ללא התוכנה. אם יש לכם שאלות נוספות על Wireshark – שמרו אותן להמשך הפרק, שם נציג שימושים מתקדמים יותר.

פרק זה בספר יהווה מבוא ויעזור לכם להבין את שאר הפרקים שיסתמכו עליו. שימו לב שעדיין לא נרד לעומק של כל נושא, אלא ניתן סקירה כללית כיצד בנוי כלל עולם התקשורת ברשת האינטרנט, ורק בפרקים הבאים נסביר ביתר בפירוט על כל נושא ונושא. בינתיים, תזכו לרכוש כלים שיאפשרו לכם לבחון את עולם תקשורת האינטרנט בצורה שלא הכרתם לפני כן!

מודל חמש השכבות

בפרק הראשון הבנו כמה ענן האינטרנט הוא גדול ומורכב, ושהוא מכיל אינספור **ישויות (Entities)**. ישות ברשת היא כל דבר המחובר לרשת – בין אם זה סמארטפון, מחשב נייד, שרת של Google, רכיב רשת שנמצא בדרך בין ישויות אחרות, או רכיב בקרה של תחנת כוח המחובר גם הוא לרשת לצורך שליטה מרחוק. העברת המידע בין כל הישויות הללו זו משימה כלל לא פשוטה: קצבי התקשורת הגבוהים, מספר המשתמשים הרב בהם צריך לתמוך בו זמנית, והכמות העצומה של המידע העובר בכל רגע נתון באינטרנט וצריך לחצות את הגלובוס כדי להגיע לצד השני של העולם – כל אלו הם רק חלק מהאתגרים איתם צריך להתמודד ענן האינטרנט.

כיצד ניתן לארגן את כל המידע הזה?

נשאלת השאלה: איך אפשר לארגן את כל המידע הזה, כך שיאפשר למערכת המורכבת הזו לעבוד – ולעבוד בצורה טובה?

ובכן – על השאלה הזו ניסו לענות רבים וטובים, אך ברור שאם כל אחד יתן את פתרונו נגיע למצב שבו כל ישות יודעת לדבר ב"שפה שלה" ואין אף "שפה משותפת" לכל רכיבי הרשת בעולם.

אך מה קרה בפועל? נוצר מצב בו הרבה יצרניות חומרה שיווקו מכשירים אשר תומכים אך ורק בתקן אחד ספציפי (תקן אשר החברה עצמה ייצרה), מה שחייב את הלקוחות להמשיך ולרכוש מוצרים נוספים מאותה היצרנית אם הם היו רוצים לתקשר בין שני מכשירים שונים. ברור כי המצב הזה אינו רצוי, והוא אינו מאפשר למכשירים שונים באמת לדבר ב"שפה אחידה" המשמשת את כל רשת האינטרנט.

כדי לפתור את הבעיה הזו וליצור סטנדרטיזציה (תִקְנוּן) של המידע העובר על גבי רשת האינטרנט, יצר ארגון התקינה הבינלאומי (**I**nternational **O**rganization for **S**tandardization – **ISO**), שאחראי על פיתוח ופרסום של תקנים בינלאומיים) את מודל שבע השכבות (המכונה גם מודל שבע הרמות). מטרת מודל זה, שנקרא **OSI** (**O**pen **S**ystems **I**ntercommunications), הינה לתת קווים מנחים כיצד צריכה להיראות תקשורת בין כל מערכת מחשב אחת לשנייה, ללא תלות ביצרן של אותה מערכת.

שימו לב: מודל השכבות הינו מודל, ולא תקן. הוא לא מחייב את מערכות התקשורת לדבר בצורה הזו אחת עם השנייה, אלא מנחה את יצרניות החומרה כיצד לממש את מערכות התקשורת כך שתהיה אחידות בין כולם.

כפי שתראו, בספר זה לא נעשה שימוש במודל שבע השכבות אלא במודל חמש השכבות[6].

מה זה בעצם אומר מודל של שכבות?

לפני שנסביר את משמעות השכבות בעולם המחשבים, נשתמש בדוגמה מהחיים עליה נחיל את מודל השכבות כדי להבין על מה מדובר. ניקח למשל מערכת מורכבת כמו טיסה בשדה תעופה: כיצד תוכלו לתאר את התהליך שעובר אדם מהרגע שמגיע לשדה התעופה במדינת המקור ועד שיוצא משדה התעופה במדינת היעד? דרך אחת לעשות זאת היא לתאר את הפעולות שהוא עושה (או שעושים בשבילו) בצורה כרונולוגית: בידוק ביטחוני והפקדת מזוודות, החתמת דרכון ועלייה למטוס (Boarding). מרגע שהמטוס

---

[6] ISO יצרו את מודל שבע השכבות באופן תיאורטי. מודל חמש השכבות (לעתים מכונה Protocol Stack) נוצר לאחר העבודה עם רשת האינטרנט, מתוך השימוש היישומי, והוא דומה למודל שבע השכבות אולם הוא מוותר על שתי שכבות (השכבה החמישית והשישית) שבפועל התגלו כמיותרות ולכן הושמטו מהמודל.

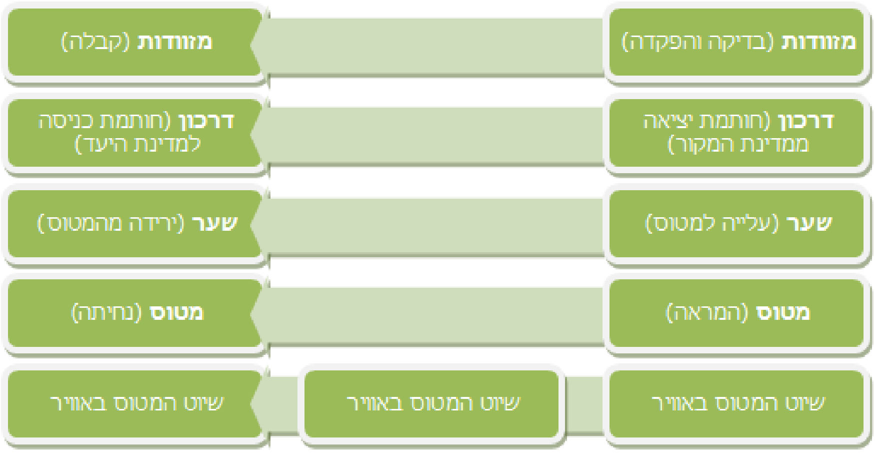
המריא, הוא מנווט את דרכו ונוחת במדינת היעד. לאחר מכן הנוסע יורד מהמטוס, מחתים שוב את הדרכון ומקבל בחזרה את המטען שלו.

שימו לב שהשתמשנו כאן באנלוגיה ולכן חלק מהפרטים עשויים להיראות "מאולצים" כדי שיתאימו למודל, אך הדבר החשוב הוא להבין את הרעיון הכללי, כפי שמתואר באיור הבא:

השלבים השונים בשדה התעופה

אם נבחן שוב את התהליך, נגלה שהוא מורכב ממספר שלבים, כאשר כל שלב מופיע הן בחלק הראשון של התהליך (במדינת המקור) והן בחלק השני של התהליך (במדינת היעד). יש פונקציה של מזוודות (במדינת המקור – הפקדה, במדינה היעד – קבלה), פונקציה של החתמת דרכון (במדינת המקור – חתימה יוצאת, במדינת היעד – חתימה נכנסת), וכן הלאה.

בצורה זו ניתן להסתכל על התהליך ככזה הבנוי ממספר שכבות, כפי שמתואר באיור הבא:



השלבים השונים בשדה התעופה, הפעם בחלוקה לשכבות מוגדרות

האיור הנ"ל מספק לנו תשתית כדי שנוכל לדבר על המבנה ממנו בנוי התהליך המורכב של טיסה. נשים לב שכל שכבה, יחד עם כל השכבות מתחתיה, מספקת שירות כלשהו.

אם נסתכל על הקשר האופקי בכל שכבה, נראה שכל משבצת שמספקת פונקציונליות כלשהי מתבססת אך ורק על השכבות שמתחתיה כדי להשלים "מסלול מלא אל היעד", ומוסיפה פעולות הקשורות לשכבה הספציפית. למשל:

- בשכבת השערים, החל משער העלייה למטוס ועד לשער הירידה מהמטוס, משתמשים בשירות העברת המטוס ממסלול ההמראה למסלול הנחיתה שמסופק על-ידי השכבות שמתחת, ובנוסף דואגים להעלות ולהוריד את הנוסעים דרך השער. בכך מתבצעת העברה מלאה של האדם בלבד מנמל התעופה במדינת המקור לנמל התעופה במדינת היעד.

  שימו לב: שכבה זו לא יודעת כיצד הגיעו הנוסעים אל השער, והיא אינה מודעת לעצם קיום המזוודות או הדרכונים. היא רלוונטית אך ורק לשערים, ורק בזה היא "מבינה". כלומר – שכבה זו אחראית על השערים בלבד, לא מכירה את השכבות שמעליה, ולא יודעת איך השכבות מתחתיה מתפקדות. היא אינה מעוניינת להכיר דברים אחרים שהיא לא אחראית עליהם.
- בשכבת הדרכונים, החל מהדלפק היוצא לדלפק הנכנס, מתבצעת העברה מלאה של האדם (על-ידי שימוש בשכבות שמתחת), וכן החתמת הדרכון במדינת המקור ובמדינת היעד.
- בשכבת המזוודות ומטה, החל מהפקדת המזוודות ועד קבלת המזוודות, כבר מתבצעת העברה מלאה למדינת היעד של האדם, המטען שלו, וכן החתמת הדרכון בשתי המדינות.

בנוסף, ניתן לשים לב שכל שלב מסתמך על השלבים הקודמים לו: החתמת הדרכון היא רק עבור מי שהפקיד כבר את המזוודות ועומד לצאת מגבולות המדינה; העלייה למטוס היא רק עבור מי שהפקיד את המזוודות וכבר החתים את הדרכון; ההמראה היא רק עבור מי שהפקיד את המזוודות, החתים את הדרכון, וכמובן – עלה למטוס. עובדה זו מאפשרת לנו לזהות סדר מוגדר לתהליך: החל מהשכבה העליונה בצד השולח (דרך כל השכבות התחתונות של הצד השולח) ועד לשכבה העליונה בצד המקבל (דרך כל השכבות התחתונות של הצד המקבל).

למודל של שכבות יש יתרון חשוב שלא דיברנו עליו: **כל שכבה אינה תלויה במימוש של שכבה אחרת.**

חשבו, למשל, שיום אחד יחליטו בממשלה על מעבר לדרכונים אלקטרוניים. התחנה של החתמת הדרכונים עדיין תמלא אחרי הייעוד שלה: רישום של אדם היוצא ממדינה אחת ונכנס למדינה אחרת. היתרון הגדול הוא שלשאר התחנות בשרשרת לא אכפת כיצד היא עושה את זה – העיקר שהיא תדאג לרשום שהנוסע יצא ממדינת המקור כדי שאפשר יהיה להעלות אותו למטוס דרך השער, ושהוא נכנס למדינת היעד כדי שיוכל להמשיך ולאסוף את המזוודות שלו. עם זאת, הדבר היחיד שכן צריך לוודא הוא שמדינת היעד יודעת להתמודד עם תיירים בעלי דרכונים אלקטרוניים.

אם נחשוב על שכבת השערים, הרי שמבחינתה, העובדה שהאדם עבר מנמל לנמל הייתה יכולה להתבצע לא באמצעות מטוס, אלא באמצעות מכונית, אונייה או סוס מעופף. היא אחראית על השערים בלבד. כך שכבה זו לא צריכה להיות מודעת ל**מימוש (Implementation)** של השכבות מתחתיה.

דוגמה נוספת יכולה להיות שליחת המזוודות דרך ספינה במקום בבטן המטוס – אנחנו עדיין מספקים את אותו שירות, העברת מזוודות אל מדינת היעד, אולם מממשים אותו בדרך אחרת.

במערכות מורכבות כמו רשת האינטרנט, שמתעדכנות לעיתים תכופות ומתפתחות מעת לעת, היתרון אותו הצגנו הוא חיוני. גם אם מימוש של שכבה מסוימת ישתנה, שאר המערכת לא תושפע משינוי זה ונוכל להמשיך לדבר עם ישויות אחרות ברשת כאילו כלום לא קרה. זאת מכיוון שהשכבה עדיין תספק את אותו **שירות** לרמות שמעליה (שימו לב שמדובר על שינוי ב**מימוש** השירות, ולא בשינוי ה**שירות** עצמו). מאפיין זה של הסתרת המימוש אותו הצגנו כרגע הוא מאפיין חיוני בבניית מודל מודולרי כגון מודל חמש השכבות.

איך זה קורה בפועל?

דיברנו מספיק על מטוסים, כעת נשוב לדבר על רשתות. הבנו מדוע בנויה מערכת מורכבת שכזו ממודל של שכבות, ומה היתרונות שמודל זה מספק. כעת נסביר כיצד מודל השכבות מיושם בעולם הרשתות. כפי שכבר אמרנו, אנו נתעסק במודל חמש השכבות, שמחלק את מימוש מערכת התקשורת לחמש שכבות לוגיות. כל שכבה במודל השכבות מספקת שירות לרמה שמעליה, מבלי לחשוף אותה לאופן בו השירות שהיא מספקת ממומש (ובכך מאפשרת לשכבה שמעליה להתייחס אליה בתור "קופסה שחורה" שבסך הכל מציעה שירות כלשהו).

כששכבה מסוימת (נניח שכבה n) על ישות אחת רוצה לדבר עם שכבה n על ישות אחרת, היא עושה זאת בעזרת **פרוטוקול** ששייך לרמה n.

פרוטוקול – הגדרה

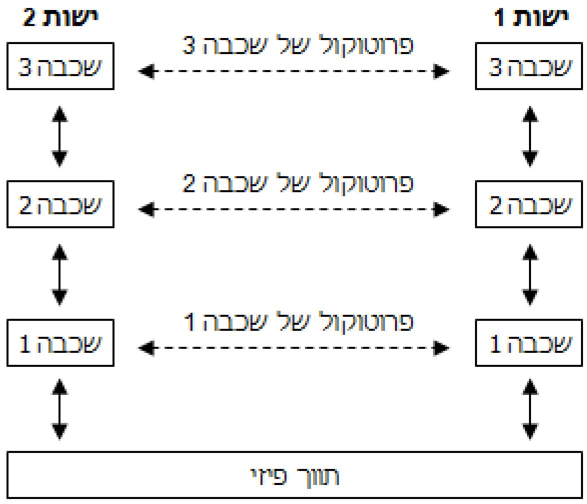
פרוטוקול (Protocol, תקן) הינו סט מוגדר של חוקים, הקובע כללים ברורים כיצד צריכה להיראות התקשורת בין הצדדים השונים. אם תחשבו על כך, אנחנו מכירים לא מעט תקנים בחיי היומיום שלנו: השפה העברית, למשל, היא תקן. היא קובעת כללי תחביר ואוצר מילים המוגדרים מראש, אשר מנחים את שני הצדדים כיצד עליהם לדבר זה עם זה על מנת שיבינו אחד את השני. היזכרו בתרגילי השרת והלקוח אותם ביצענו כשלמדנו סוקטים: איך, לדוגמה, יודע השרת שהלקוח מבקש לבצע צילום מסך? באמצעות פרוטוקול התקשורת שהגדרתם ביניהם.
גם HTTP, הפרוטוקול המשמש להעברת דפי האינטרנט אליהם אנחנו גולשים בדפדפן, הוא תקן. הוא קובע כיצד ידבר הדפדפן עם השרת המרוחק, ובאיזו צורה יציג חזרה למשתמש את הדף שהגיש לו השרת.

למעשה, בפרק הראשון בספר, כשראינו שהדפדפן שולח הודעת בקשה לשרת, הודעה זו הייתה בפועל הודעה בפרוטוקול HTTP, עליו נלמד לעומק בהמשך הספר.

בלי פרוטוקולים היינו מקבלים סיטואציה נוסח "מגדל בבל", בו כל רכיב מדבר בשפה שלו ואף אחד לא יכול לדבר עם השני. חשבו על שני אנשים שונים שנפגשים, האחד יודע רק סינית והשני יודע רק אנגלית. יהיה להם קשה מאוד לתקשר אחד עם השני בצורה הזו. כדי שיצליחו לדבר אחד עם השני, עליהם להחליט מראש על "שפה משותפת" אותה שניהם יודעים.

פרוטוקול מחייב את שני הצדדים בשיחה לסט מסוים של חוקים הקובעים כיצד יראה תהליך התקשורת ביניהם. בצורה זו הם יכולים לדבר ולהבין האחד את השני.

נחזור לתקשורת בין השכבות. בין שכבה n בישות אחת לשכבה n בישות אחרת אין אף מידע שמועבר **ישירות**. במקום זאת, כל שכבה מעבירה את המידע שקיבלה (ונתונים נוספים שהיא מוסיפה, כפי שנראה בהמשך) לשכבה שנמצאת ישירות מתחתיה, עד שמגיעים לשכבה התחתונה ביותר. מתחת לשכבה זו נמצא המימד הפיזי, ורק שם עובר המידע בפועל. ניתן לראות זאת בתרשים הבא, כאשר תקשורת וירטואלית מיוצגת על-ידי קווים מקווקווים ותקשורת פיזית מיוצגת על-ידי קווים רציפים.



הקשרים בין השכבות

אם ניישם את המסקנות מהתרשים הנ"ל על הדוגמה של Sockets שראינו בפרק הקודם, נבין את הדבר הבא: בעוד שכל Socket מדבר עם ה-Socket השני בפרוטוקול של אותה שכבה, הוא "חושב" שהוא מדבר איתו ישירות (על-ידי שימוש בפונקציות send ו-recv) – אולם התקשורת ביניהם היא <u>וירטואלית</u>, ובפועל היא מסתמכת על העברת המידע לשכבות התחתונות ושימוש בשירות שהן מספקות. המידע יורד עד לכרטיס הרשת, יוצא אל הכבל (או כל תווך פיזי אחר) ומוצא את דרכו אל היעד – שם הוא נקלט בכרטיס הרשת (כפי שנוכחנו לדעת על-פי מה שראינו ב-Wireshark) ועולה חזרה אל השכבה הרלוונטית.

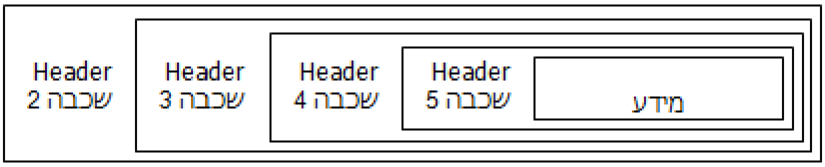
כיצד בנויה פקטה (Packet)?

הפקטה היא כזכור חבילת מידע שעוברת ברשת ממקום למקום. מה שלא נגענו בו קודם הוא הקשר בין הפקטה לבין מודל חמש השכבות. מה הקשר ביניהם? התשובה הפשוטה היא שהפקטה מכילה בתוכה מידע של כל שכבה ממודל חמש השכבות שהשתתפה בתהליך התקשורת[7], אבל מה זאת אומרת?

מוקדם יותר בפרק, הזכרנו שבתהליך השליחה כל שכבה מעבירה את הפקטה לשכבה שמתחתיה. בסופו של דבר, הפקטה מורכבת ממספר פרוטוקולים הבנויים זה מעל זה, כאשר כל שכבה מוסיפה את המידע שלה (הקשור לשירות אותו היא מספקת) לתחילת הפקטה של הרמה שמעליה, ובכך למעשה עוטפת אותה בעוד שכבה. חשבו על זה כמעין משחק של חבילה עוברת – בכל שלב בו נבנית החבילה היא נעטפת בעוד ועוד שכבות (כאשר כל שכבה לא יודעת מה יש בפנים), ולאחר שהחבילה נשלחת ועוברת בין המשתתפים – בכל שלב מקלפים אותה, שכבה אחר שכבה. לתהליך של עטיפת המידע בכל שכבה ושכבה בצד השולח קוראים **Encapsulation (כימוס)**[8], ואילו תהליך קילוף הפקטה בצד המקבל נקרא **Decapsulation (קילוף)**.

המידע שמוסיפה כל שכבה בתחילת הפקטה נקרא **Header (תחילית)**, והוא מכיל מידע שמשמש לשליטה ובקרה על הפקטה הרלוונטי לשירות שמספקת אותה שכבה (למשל: כתובת ה-IP אליה מיועדת הפקטה, בקרת שגיאות וכו').

האיורים הבאים מתארים כיצד נראית פקטה במודל חמש השכבות, ואיפה נמצא המידע של כל פרוטוקול בפקטה השלמה[9].



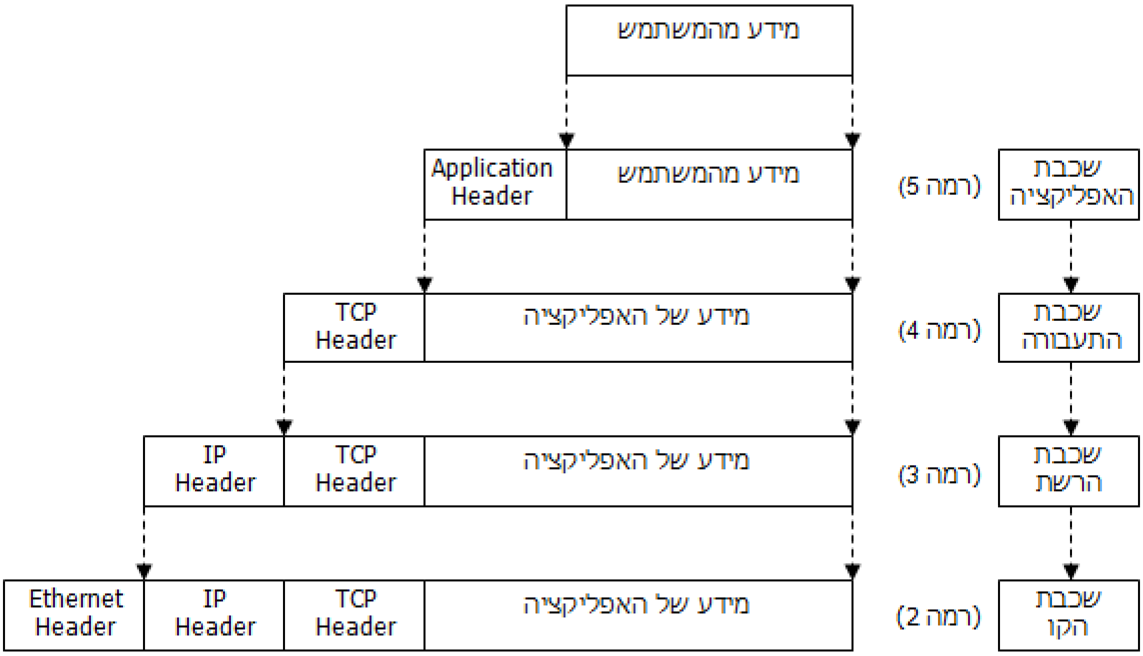
מבנה פקטה במודל חמש השכבות #1

---

[7] לא כל השכבות חייבות להשתתף בתהליך התקשורת. במקרים מסוימים יש פקטות שמכילות רק את שכבות 1-3, למשל, וזה הגיוני לחלוטין כשהשכבות מעליהן כלל לא היו רלוונטיות לתהליך התקשורת הספציפי בין שני הצדדים.

[8] למונח Encapsulation יש משמעות נוספת בתכנות מונחה עצמים: הסתרת מימוש וחשיפת ממשק תכנותי. שימו לב לא להתבלבל בין השניים.

[9] בשכבה השנייה בלבד מוסף מידע גם לסוף המסגרת (הוא נקרא **Trailer**), אך התעלמנו ממנו במכוון ובחרנו להציג רק את ה-Header כדי לפשט את התרשים. בנוסף, לא כללנו את השכבה הראשונה (השכבה הפיזית), משום שלרוב נוטים להתעלם ממנה בהסנפה (הסנפה היא הפעולה בה אנו מסתכלים על חבילות המידע בדיוק כפי שנשלחו או התקבלו בכרטיס הרשת).



מבנה פקטה במודל חמש השכבות #2

באיור האחרון, בכל שכבה נתנו דוגמה לפרוטוקול השייך לאותה השכבה (TCP בשכבה הרביעית, IP בשכבה השלישית ו-Ethernet בשכבה השנייה), אך ברור שאלו לא הפרוטוקולים היחידים של אותה שכבה. בהמשך הספר נלמד לעומק על כל אחד מהפרוטוקולים הללו.

דבר מעניין ששווה לשים עליו דגש הוא שה-Data של כל שכבה (כלומר המידע עצמו, לא ה-Headerים) **זהה** לפקטה של השכבה שמעליה; בתהליך השליחה כל שכבה מקבלת מהשכבה שמעליה את הפקטה בדיוק כפי שהיא, מוסיפה לה את Header על-פי התקן (פרוטוקול) של אותה שכבה ומעבירה אותה הלאה לשכבה שמתחת.

כך למשל, בשכבה השלישית, ה-Data של הפקטה כולל בין היתר את ה-Header של השכבה הרביעית (בדוגמה לעיל, ה-TCP Header). בשכבת הקו, ה-Data של הפקטה כולל את ה-Header של השכבה השלישית, והן של השכבה הרביעית (בדוגמה זו, את ה-IP Header ואת ה-TCP Header).

פירוט חמש השכבות

ובכן, בדומה לדוגמת המטוסים – נרצה לדעת מה עושה כל שכבה (או למעשה **איזה שירות** היא מספקת לרמות שמעליה). השכבות במודל חמש השכבות הן: השכבה הפיזית, שכבת הקו, שכבת הרשת, שכבת התעבורה ושכבת האפליקציה. כעת נסקור אותן מלמטה למעלה, החל מהשכבה התחתונה ועד לשכבה העליונה:

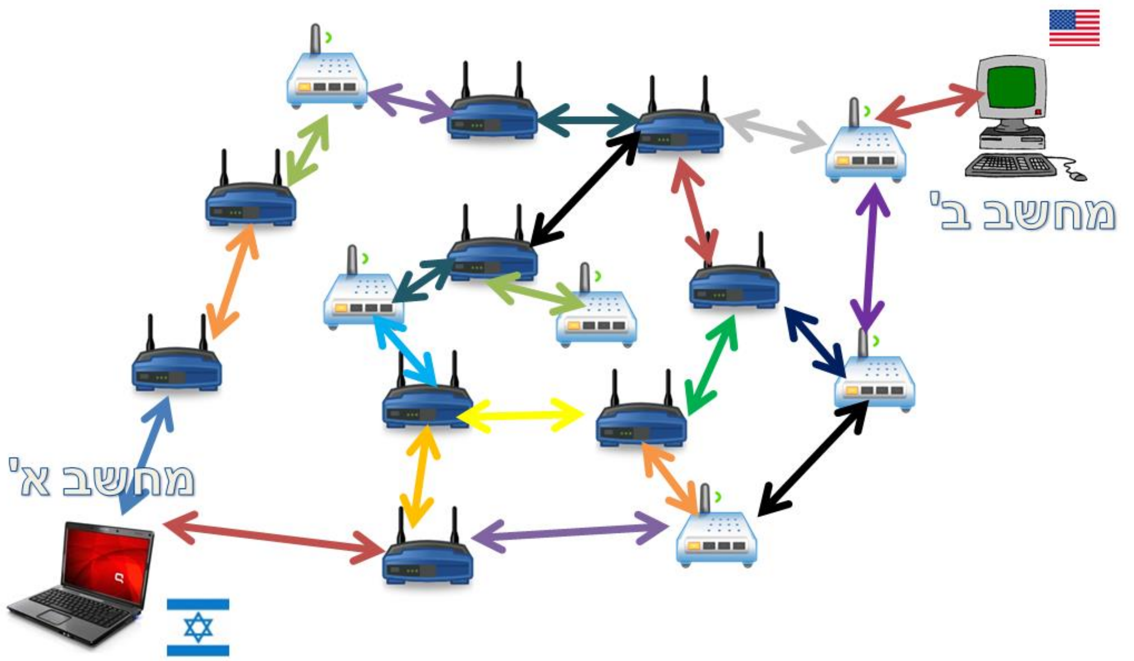
שכבה ראשונה – השכבה הפיזית

תפקידה של שכבה זו הוא להעביר את הביטים מנקודה אחת לנקודה שנייה עם כמה שפחות שגיאות. השכבה הפיזית רק מעבירה 0 או 1 מצד לצד. שימו לב ששכבה זו אינה מודעת לרצפים של ביטים, פקטות או כל דבר כזה. מבחינתה עליה להעביר ביט אחד בלבד בכל פעם. ההעברה הפיזית יכולה להתבצע על גבי מגוון של תווכים: כבלי רשת, סיבים אופטיים, באוויר (גלים אלקטרומגנטיים, לווין) וכו' – העיקר שהמידע יגיע ליעד.

שכבה שנייה – שכבת הקו

שכבת הקו מסתמכת על ההעברה הפיזית של המידע שנעשה ברמה שמתחתיה, ומאפשרת לנו לדבר עם ישויות אחרות שסמוכות אלינו.

באיור שלפנינו תחום האחריות של השכבה השנייה מתבטא בכל חץ צבעוני שמקשר ישויות רשת סמוכות אחת לשנייה:



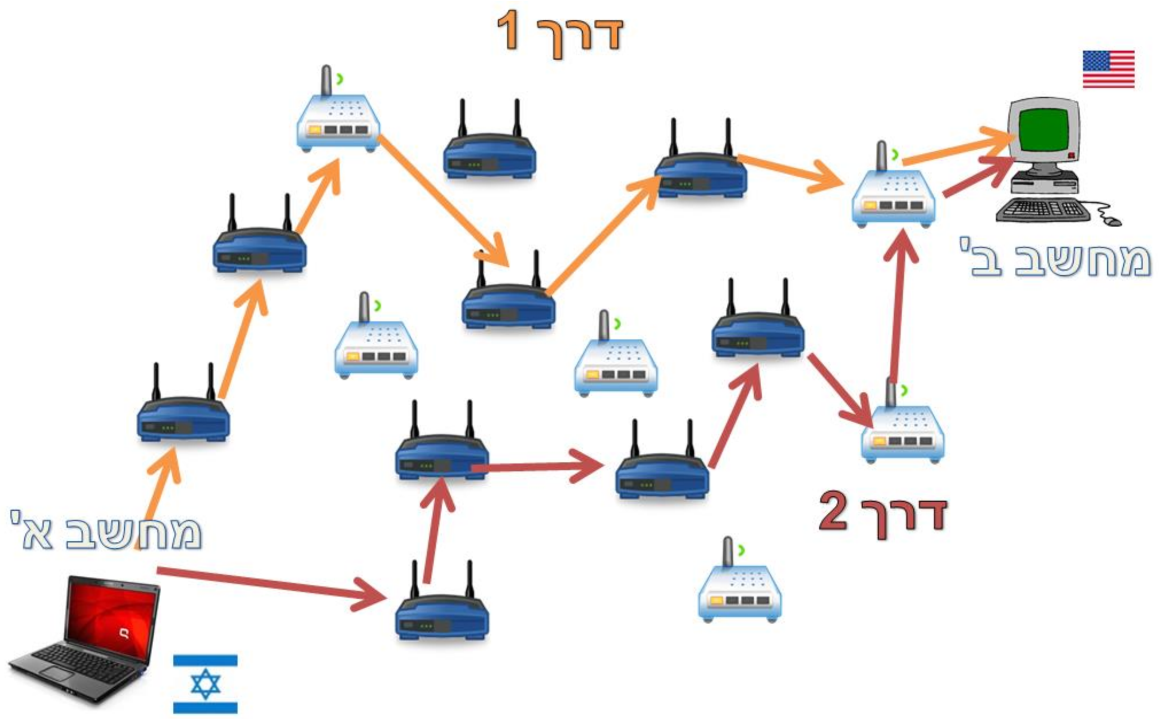
בנוסף, השכבה השנייה מוסיפה מספר יכולות חשובות:

- ארגון המידע בגושים (המכונים **מסגרות – Frames**, כמו שראינו בפילטר של Wireshark), בהן תוכלנה השכבות הגבוהות יותר לטפל.
- טיפול במקרים שבהם מספר ישויות מנסות לשלוח מידע על אותו תווך פיזי (למשל: מספר מחשבים על אותו כבל רשת, או על אותה רשת WiFi ביתית). השכבה השנייה תמנע התנגשויות.
- טיפול ראשוני בשגיאות (או לכל הפחות זיהוי השגיאות, כדי שאפשר יהיה לשלוח את המסגרת מחדש).

שכבה שלישית – שכבת הרשת

שכבת הקו מאפשרת לנו לדבר עם ישויות אחרות הסמוכות אלינו, אך מה אם נרצה לדבר עם מישהו בקצה השני של העולם? תפקידה של שכבת הרשת הוא למצוא את המסלול הטוב ביותר מאיתנו אל היעד ובחזרה. שכבה זו לא מתעסקת בתקשורת בין ישויות סמוכות, אלא אחראית על המסלול המלא בין שתי נקודות קצה. ניתוב המידע מתבצע על-ידי רכיבים המכונים **Routerים (נתבים)**, אשר מנתבים את הפקטות בין הרשתות השונות. כך יכולה פקטה לצאת מקו אחד, לעבור דרך מספר קווים שונים ולבסוף להגיע אל הרשת אליה היא מיועדת.

באיור שלפנינו, כל מסלול חיצים בצבע מסוים מסמן מסלול שעשויה לבחור שכבת הרשת עבור הפקטה:

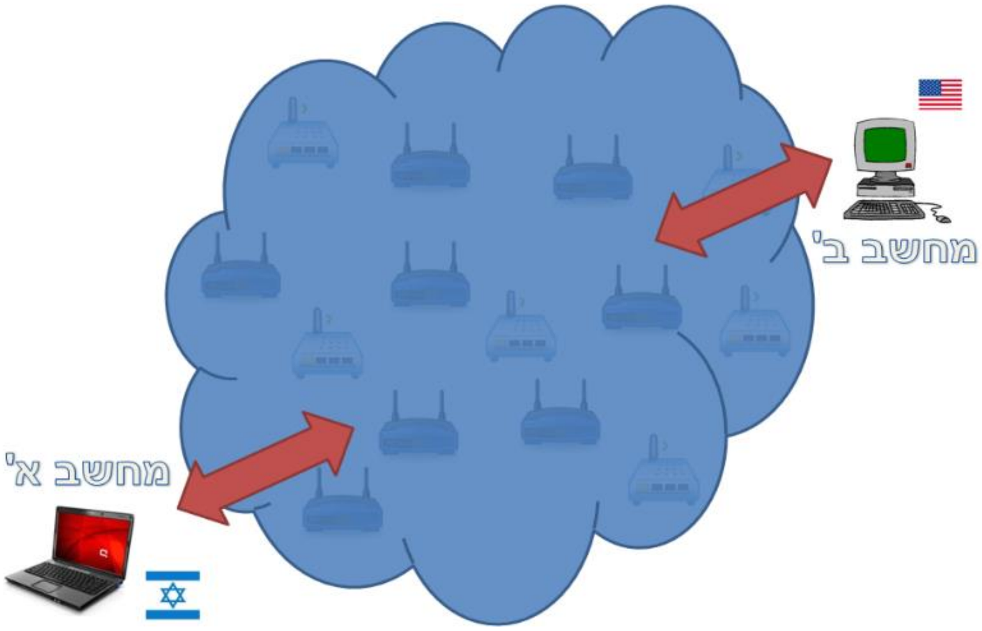


שכבה רביעית – שכבת התעבורה

שכבת התעבורה מסתמכת על שכבת הרשת שתנתב עבורה פקטות מישות כלשהי ברשת לישות אחרת, נאמר אפילו בקצה השני של העולם. אך שירות זה עדיין לא מספיק. עלינו לזכור שהנחה בסיסית ברשת האינטרנט היא שהחיבור עצמו אינו אמין – פקטות יכולות להישלח ולא להגיע ליעדן, או להגיע ליעדן באיחור רב. מה יקרה אם נשלחו שתי פקטות אחת אחרי השנייה, אך הן התחלפו והגיעו ליעד בסדר הפוך? כדי להיות מסוגלים להתקיים ברשת האינטרנט אנחנו צריכים בחלק מהמקרים ליצור קישור אמין ורציף בין שתי נקודות הקצה, כזה שייתן סדר ומשמעות למידע שיישלח – ולא סתם אוסף של חבילות מידע שלא בהכרח קשורות אחת לשנייה.

בעוד שהמטרה הקודמת היא אופציונלית (יש מקרים בהם לא נהיה חייבים להבטיח את סדר הפקטות שנשלחות או את הגעתן כלל), שכבת התעבורה מספקת תמיד דבר חשוב נוסף: האפשרות לפנות אל מספר שירותים הנמצאים על אותה ישות. דמיינו שעל שרת מסוים רצה גם תוכנה המספקת שירות מיילים וגם תוכנה המספקת שירות WEB (כלומר מגישה דפי אינטרנט). כיצד יוכל השרת להבדיל בין שתי הבקשות שמתקבלות אליו מהלקוח, האחת אל שירות המייל והאחת אל שירות ה-WEB? לשם כך, השכבה הרביעית מוסיפה לנו פורטים (דיברנו על המושג כבר בפרק תכנות ב/ Sockets-כתובות שלSocket, שם דימינו את הפורט למזהה דירה בתוך בניין), כדי שנוכל להבדיל בין השירותים השונים ולהשתמש בכמה שירותים על אותה ישות.

באיור שלפנינו ניתן לראות ששכבת הרשת "העלימה" את הצורך של שכבת התעבורה להכיר את המסלול אל היעד. מבחינת השכבה הרביעית, ישנו "ענן" המחבר בין היעד לבינה – בו היא משתמשת כדי לשלוח פקטות לישות בצד השני:



שכבה חמישית – שכבת האפליקציה

שכבה זו מסתמכת על שכבת התעבורה כדי לקבל קישור לוגי בין שתי נקודות הקצה. איך היא עושה זאת? כפי שכבר הבנו ממודל השכבות – זה לא באמת מעניין אותה. כל עוד השכבה שמתחתיה מספקת לה את השירות של יצירת קישור שכזה, היא משתמשת בו לצרכיה השונים של האפליקציה. לשכבה זו קיימים פרוטוקולים רבים, המוכרים שבהם: HTTP (פרוטוקול הגלישה באינטרנט עליו דיברנו קודם), SMTP (פרוטוקול דואר), FTP (העברת קבצים), ועוד רבים אחרים. למעשה, כמעט כל אפליקציה שמשתמשת בחיבור רשתי כלשהו מדברת בפרוטוקול של שכבת האפליקציה.

לסיכום סקירת השכבות, להלן טבלה המתארת את כל השכבות יחד עם מעט פרטי מידע שיאפשרו לכם להשוות ביניהן:

מספר השכבה	שם השכבה	מטרה (בקצרה)	פרוטוקול לדוגמה	שם של גוש מידע
1	השכבה הפיזית (Physical Layer)	העברת המידע ביט אחר ביט – 0 או 1 בכל פעם		**ביט** (bit, סיבית)
2	שכבת הקו (Data Link)	תקשורת בין ישויות סמוכות זו לזו	Ethernet	**מסגרת** (frame)
3	שכבת הרשת (Network Layer)	החלטה על המסלול שתעבור חבילת מידע בין המקור אל היעד	IP	**פקטה** (packet, חבילה)
4	שכבת התעבורה (Transport Layer)	ריבוב אפליקציות על אותה ישות (תמיד) + מתן אמינות לקישור (אופציונלי)	TCP	**סגמנט** (segment)
5	שכבת האפליקציה (Application Layer)	שימושים שונים בהתאם לאפליקציה	HTTP	* אין שם מיוחד *

שימו לב: כשמשתמשים באחד הכינויים לגוש מידע של אחת השכבות, מתכוונים לרצף המידע משכבה זו ומעלה. למשל: כשמשתמשים במושג "פקטה" מתכוונים לפקטה בשכבה השלישית, אך גם לכל המידע של השכבות הרביעית והחמישית (שמוכלות בתוך הפקטה, כפי שהזכרנו באיורים של מבנה הפקטה). המונח "מסגרת" מתאר את כל המידע השייך לשכבה השנייה, אך גם לשכבה השלישית, הרביעית והחמישית.

בהתאם להסבר לעיל, כל מסגרת היא גם פקטה (שהרי אין חבילה בשכבה השלישית בלי שכבה שנייה), אך לא כל פקטה היא מסגרת (שכן יש מסגרות שהן רק בשכבה השנייה).

מודל השכבות ו-Sockets

גם ה-Sockets עליהם למדנו בפרק הקודם שייכים לשכבת האפליקציה. נזכיר כי Sockets הם בסך הכל API (ממשק תכנותי) שמספקת מערכת ההפעלה כדי שאפליקציות יוכלו ליצור חיבור רשתי לישויות אחרות ברשת. הם אינם שכבה במודל חמש השכבות. האפליקציות משתמשות ב-Sockets שיצרו כאל "צינור" להעברת המידע, ומדברות מעליהם בפרוטוקולים השונים של רמת האפליקציה (בדיוק כמו הפרוטוקול שאתם כתבתם בתרגיל בפרק הקודם).

כעת, כשאנחנו מכירים את מודל חמש השכבות, נוכל לשים לב לדבר הבא: כשהשתמשנו ב-Socket, בכלל לא נתנו לו פרמטרים שרלוונטים לשכבה שלו – אלא נתנו לו פרמטרים שעוזרים לו לפתוח את החיבור בהתבסס על הרמות שמתחתיו! בפועל, הפרמטרים שנתנו היו רלוונטיים ישירות לשכבות שמתחת ל-Socket – שכבת הרשת (השכבה השלישית) ושכבת התעבורה (השכבה הרביעית). נמחיש זאת באמצעות דוגמה:

```
s = socket.socket()
s2 = socket.socket()
s.bind(("1.2.3.4", 80))
s2.connect(("5.6.7.8", 8820))
```

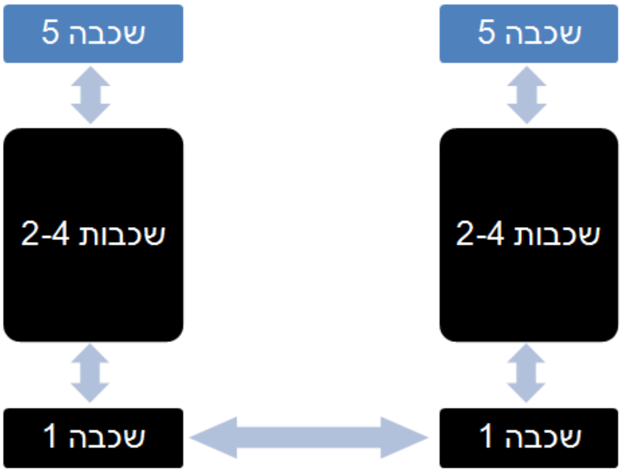
בדוגמה זו, סיפקנו את הפרמטרים של שכבת הרשת (Network Layer): באיזו כתובת IP להשתמש. ("1.2.3.4" ו-"5.6.7.8")
כמו כן, סיפקנו את הפרמטרים של שכבת התעבורה (Transport Layer): באיזה פורט להשתמש. (80 ו-8820)

למעשה, Socket הינו ממשק שמאפשר את התקשורת מהשכבה הפיזית ועד שכבת התעבורה, ומעליו מדברים בפרוטוקולים שונים בשכבת האפליקציה.[10]

בספר זה, נסקור את חמש השכבות מלמעלה למטה, כלומר החל משכבת האפליקציה שלמעלה ועד לשכבה הפיזית שלמטה. שימו לב שייתכן מצב בו לא תבינו בדיוק כיצד עובר המידע בשכבות שמתחת, ותצטרכו להתייחס אליהן כאל "קופסה שחורה", כזו שרק מספקת שירותים ולא ברור כיצד היא פועלת (בדיוק כפי

[10] למעט Raw Sockets, עליהם לא נפרט בספר זה.

שהפרוטוקולים ברמות הגבוהות מניחים שהרמות שמתחתיהן הן "קופסאות שחורות" וסגורות שמספקות להם שירותים שונים). ככל שנצלול לעומק ונגיע לרמות התחתונות, כך תבינו יותר כיצד עובר המידע בפועל.
כך למשל, כשנלמד על שכבת האפליקציה, הגיוני שלא יהיה ברור כיצד מובטח שהמידע עובר מישות אחת לישות שנייה. נושא זה יתבהר בהמשך הספר, כשנלמד על השכבות התחתונות.



אתם מוזמנים לרשום לעצמכם בצד דף עם שאלות, כדי שתוכלו לחזור אליו אחר כך כשנלמד על השכבות הבאות ולבדוק אם שאלותיכם אכן נענו.

למה המידע מחולק לפקטות?

למדנו על מודל השכבות, הבנו את החשיבות שלו והכרנו את התפקיד של כל שכבה ושכבה במודל – אך עדיין לא שאלנו את עצמנו שאלה בסיסית, שאולי תהיתם לגביה: מדוע בכלל לחלק את המידע לפקטות? למה לא להעביר את כל המידע כרצף ארוך של ביטים, שמתחיל כשישות אחת רוצה לשלוח מידע לישות אחרת ומסתיים רק כאשר כל המידע הועבר לצד השני?

לשאלה הזו יש מספר תשובות. נזכיר את הבולטות שבהן:

- בקרת שגיאות טובה יותר: בחלק מהשכבות נעשית בקרת שגיאות על המידע שנשלח, כדי לזהות שגיאות ולאפשר שליחה מחודשת של המידע אם הוא לא הגיע ליעדו כראוי. חלוקת המידע לקבוצות קטנות, אותן אנחנו מכירים כפקטות, מאפשרת לזהות את השגיאות מוקדם יותר (לאחר שנשלח רק חלק קטן מרצף מהידע השלם), ובמידה שקרתה שגיאה – לשלוח מחדש אך ורק את החלק הפגום, במקום לשלוח את כל המידע מחדש.
- שילוב מספר זרמי מידע (Streams) במקביל: חלוקת המידע לפקטות מאפשרת לכמה אפליקציות לשלוח במקביל את המידע שלהן ללא צורך להמתין קודם שאפליקציה אחרת תסיים לשלוח את המידע שלה. חשבו על כך: כל כרטיס רשת יכול להוציא בכל זמן נתון אך ורק זרם נתונים אחד אל התווך אליו הוא מחובר. אם לא היינו מפצלים את המידע לפקטות, כל תוכנה הייתה צריכה לחכות עד

שהקו יתפנה, ולשלוח בתורה מידע דרך כרטיס הרשת. בצורה זו לא הייתה מתאפשרת שליחה במקביל בין מספר תוכנות[11].

הדבר נכון גם לגבי מספר מחשבים המשדרים על אותו הקו, שכן גם במקרה הזה לא ניתן להעביר על אותו קו יותר מרצף מידע אחד בו-זמנית. אם המידע היה עובר באופן רציף ולא מחולק למסגרות – בכל פעם שמחשב היה שולח מידע כלשהו, שאר המחשבים שנמצאים איתו על אותו קו היו מנועים מלשלוח מידע והיו צריכים לחכות שהוא יסיים את השליחה. חלוקת המידע למסגרות גורמת לכך שבסוף כל מסגרת ניתנת הזדמנות לישות אחרת ברשת להתחיל לשדר מסגרת משלה, ומונעת מישות אחת להציף את הקו ברצף ארוך מאוד של ביטים[12].

- מניעת בעיות סנכרון ברמת החומרה: לא ניכנס לסיבה הזו לעומק, אולם נציין שברמת החומרה קל יותר לסנכרן בין מספר ישויות המחוברות על אותו קו כל עוד המידע מחולק למנות קטנות, וכך יש פחות סיכוי להתנגשויות. נושא זה יורחב בהמשך הספר, בסוף הפרק על שכבת הקו.

כאמור – לא ציינו את כל הסיבות לחלוקת המידע לפקטות. חלק מהסיבות הנוספות יוזכרו בהמשך הספר, ועל חלק מהן לא נדבר כלל.

Wireshark

כדי להבין כיצד עובר המידע ברשת, נרצה להסניף את התעבורה היוצאת והנכנסת אל המחשב שלנו ולנתח אותה (**הסנפה** היא הפעולה בה אנו מסתכלים על חבילות המידע בדיוק כפי שנשלחו או התקבלו בכרטיס הרשת). כך נוכל לגלות בדיוק מה קורה ברשת ולהבין דברים שאין לנו דרך אחרת לראותם. לשם כך נשתמש בתוכנה **Wireshark**, תוכנת הסנפה מהטובות בעולם – והיא גם חינמית!

התוכנה נמצאת בתוך חבילת ההתקנות שבמבוא לספר, אם טרם התקנתם אותה עשו זאת כעת.

קיימות מספר דרכים לפתוח את התוכנה:

- לחיצה כפולה על ה-icon של Wireshark שנמצא על ה-Desktop.
- דפדוף למיקום המלא של התוכנה (בדרך כלל C:\Program Files\Wireshark\Wireshark.exe).
- פתיחת שורת ההפעלה (WinKey + R), הקלדת המילה Wireshark והקשת Enter.

---

[11] התהליך שבו מידע ממספר מקורות משולב אל תווך משותף אחד נקרא **ריבוב (Multiplexing)**. בדוגמה זו המידע מתקבל ממספר אפליקציות, ויוצא אל כרטיס הרשת (שהוא משאב יחיד המשותף לכל האפליקציות על אותה ישות).

[12] תופעה זו מכונה **הרעבה (Starvation)**.

ניתן לצפות בסרטון ההסבר לעבודה בסיסית עם Wireshark בכתובת:

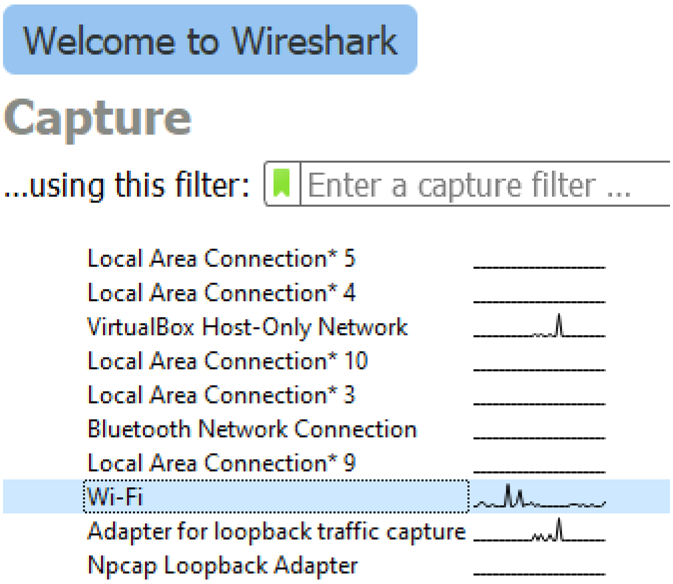
http://data.cyber.org.il/networks/videos/wireshark-basic.html

ניתן לצפות בסרטון ההסבר על שימוש מתקדם יותר ב-Wireshark בכתובת:

http://data.cyber.org.il/networks/videos/wireshark-advanced.html

הדרכה קצרה לתחילת עבודה:

עם הפעלת Wireshark יופיע לפניכם תפריט, שבו תתבקשו לבחור את כרטיס הרשת שעליו תבצעו את ההסנפה. למחשב שלנו יש כמות רבה של כרטיסי רשת, חלקם וירטואליים או תוכנתיים, ולכן יש לבחור רק את כרטיס הרשת שדרכו מתבצעת בפועל התקשורת בין המחשב שלנו לאינטרנט. כדי להקל עלינו לזהות, יש גרף קטן ליד כל שם של כרטיס רשת. ממשק שהגרף שלו אינו קבוע הוא ממשק שעוברת דרכו תקשורת. כרטיס הרשת שלכם הוא או מסוג Local Area Connection או מסוג Wi-Fi. בדוגמה שלפניכם, כרטיס הרשת הפעיל הוא Wi-Fi. יש להקליק עליו כדי להתחיל הסנפה חדשה:



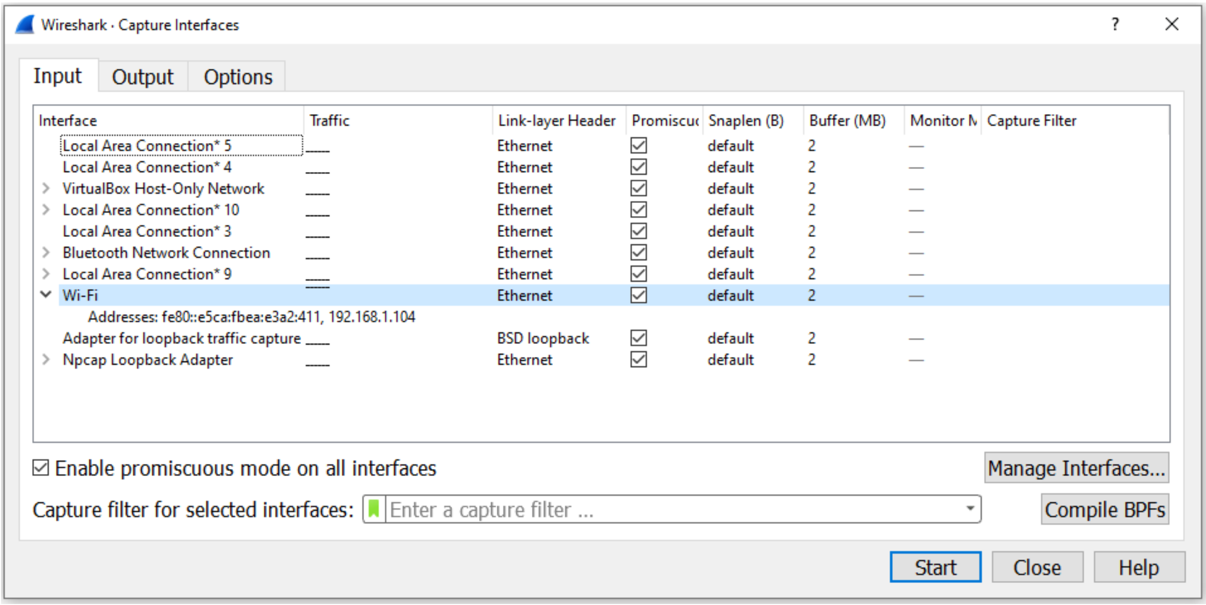
Capture Options

בתפריט זה ניתן לקבוע הגדרות שונות עבור ההסנפה. ניתן להגיע אליו במספר דרכים:

- דרך הכפתור בסרגל הכלים העליון, בכל שלב בו התוכנה פתוחה (לא רק במסך הפתיחה):

- דרך התפריט Capture < Options… (יש גם קיצור מקלדת: Ctrl + K)

ההגדרות הבולטות במסך זה:



- בחירת האפשרות Promiscous Mode: הכנסת כרטיס הרשת ל"מצב פרוץ", מה שיגרום לכך שנראה בהסנפה את כל המסגרות שרואה כרטיס הרשת, גם כאלו שלא מיועדות אליו (את המשמעות של משפט זה נבין בהמשך הספר).
- שורת ה-Capture Filter: מגדיר מסנן של פקטות להסנפה עבור ה-Driver (נרחיב על משמעות מסנן זה בהמשך הפרק).
- תפריט ה-Options: מאפשר שמירה של ההסנפה לקובץ ואף חלוקה אוטומטית שלה לקבצים על-פי גודל או זמן (למשל: חלוקת ההסנפה לקבצים בגודל 50MB כל אחד, או סגירת קובץ הסנפה ופתיחה של קובץ חדש בכל דקה).

התחלה ועצירה של הסנפה

כדי להתחיל את ההסנפה יש לבצע אחת מהפעולות הבאות:

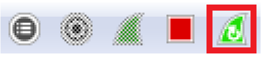
- לחיצה על כפתור Start באחד התפריטים הקודמים שהוצגו.
- דרך הקיצור במסך הפתיחה (בחירה בכרטיסי הרלוונטי מהרשימה ולחיצה על Start).
- דרך הכפתור בסרגל הכלים העליון, בכל שלב בו התוכנה פתוחה (לא רק במסך הפתיחה):

- דרך התפריט Capture < Start (יש גם קיצור מקלדת: Ctrl + E).

שימו לב שלאחר שההסנפה פועלת, מצב הכפתורים משתנה, וכעת יש 2 פעולות נוספות שניתן לעשות:

- <u>עצירת הסנפה</u> – דרך הכפתור בסרגל הכלים העליון או דרך התפריט Capture < Stop (יש גם קיצור מקלדת: Ctrl + E):

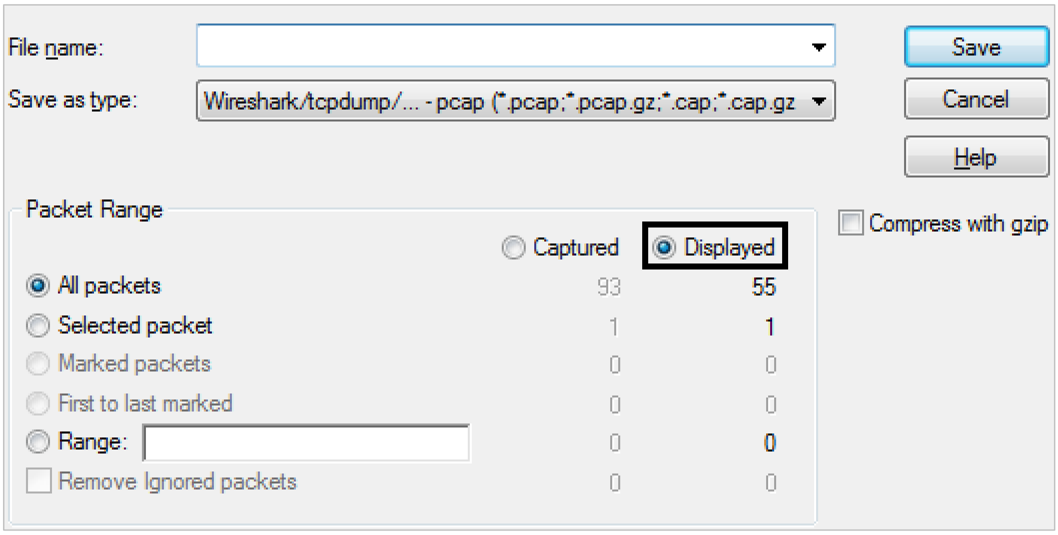
- <u>התחלה מחדש של ההסנפה</u> – ניקוי רשימת הפקטות והתחלת הסנפה חדשה על אותו interface (כרטיס רשת), עם אותם מאפיינים, ואותו Display Filter. גם את האפשרות הזו ניתן להפעיל דרך הכפתור בסרגל הכלים העליון או דרך התפריט Capture < Restart (יש גם קיצור מקלדת: Ctrl + R):



שמירה ופתיחה של קבצי הסנפה

כדי לשמור את הפקטות שנקלטו, נעצור את ההסנפה ונלחץ על File < Save (קיצור מקלדת: Ctrl + S). הקובץ שיישמר יהיה בעל סיומת pcap.

אם נרצה לשמור רק חלק מהפקטות שנקלטו (שימושי בעיקר במקרים בהם רוצים לשמור רק את הפקטות שעונו על הפילטר הנוכחי ומוצגות כעת למסך), נפתח את התפריט File < Export Specified Packets..., שיציג לנו את חלון השמירה הרגיל אך יוסיף לנו את אזור ה-Packet Range שמאפשר לבחור אילו פקטות לשמור:



במידה שנרצה לשמור רק את הפקטות שעונות על הפילטר הנוכחי, נבחר באפשרות Displayed.

כדי לטעון לתוכנה קובץ הסנפה, ניתן ללחוץ לחיצה כפולה על קובץ pcap או לבחור מהתפריט File < Open (קיצור מקלדת: Ctrl + O).

מסננים (Filters)

סוגי מסננים

כפי שכבר הוזכר, יש שני סוגים של מסננים: **מסנן תצוגה** (Display Filter) **ומסנן הסנפה** (Capture Filter).
Display Filter, בשונה מה-Capture Filter (שמיועד עבור ה-Driver), משפיע על התצוגה בלבד – כלומר פקטות שלא עברו את הפילטר עדיין קיימות בהסנפה, ואם נשנה את הפילטר נוכל להחזירן לתצוגה. עם זאת, פילטר זה איטי בהרבה מהפילטר של ה-Driver.

בספר זה לא ניגע בתחביר לכתיבת Capture Filters, אולם נסקור בקצרה את ההבדלים ביניהם:

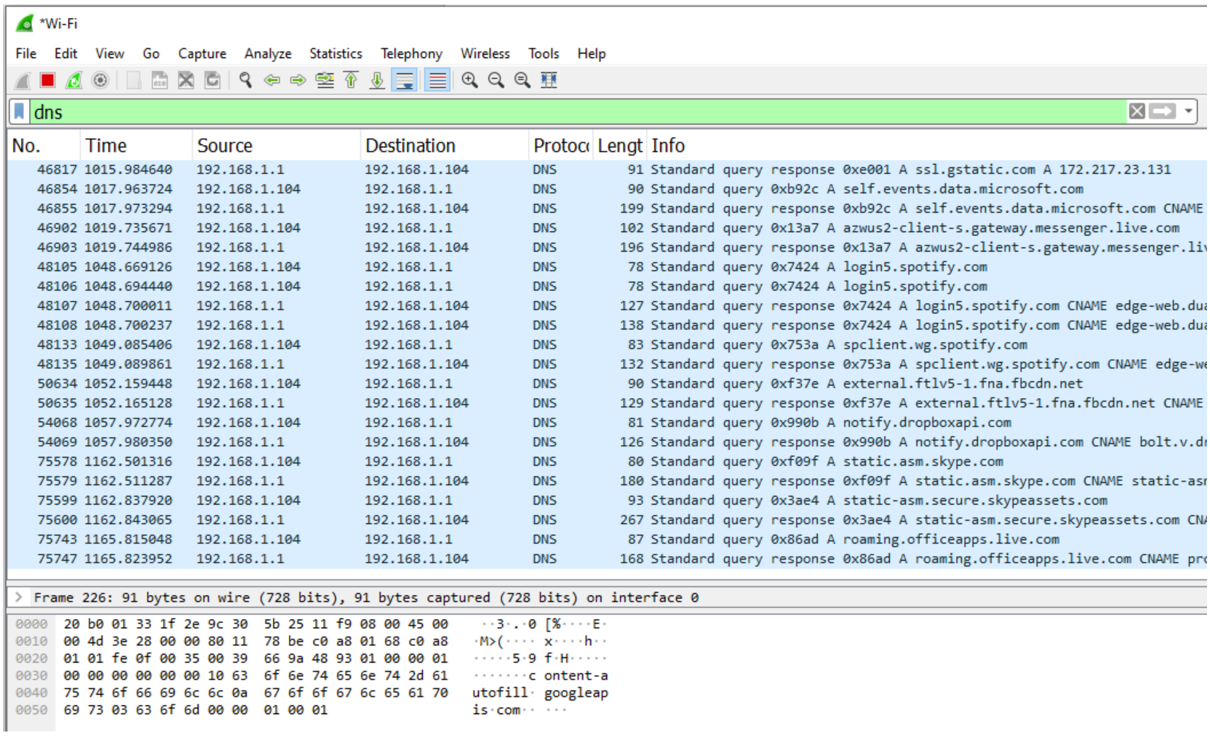
קריטריון	מסנן הסנפה (Capture Filter)	מסנן תצוגה (Display Filter)
רמה בה רץ	מערכת ההפעלה (Driver)	התוכנה (Wireshark)
המקום ממנו מפעילים אותו	מסך Capture Options	מסך ההסנפה הראשי
מתי מפעילים אותו	לפני ההסנפה	במהלך ההסנפה
האם ניתן לשנות בזמן הסנפה	לא	כן
תחביר	מצומצם	עשיר ורחב
מהירות	מהיר	איטי

מידע נוסף על Capture Filters תוכלו למצוא כאן: http://wiki.wireshark.org/CaptureFilters.

דוגמאות

מסנני תצוגה מאפשרים יכולות סינון מתקדמות, ונותנים אפשרות לסנן גם על-פי השדות הפנימיים של כל פרוטוקול (יש אפילו אפשרות לסנן על-פי פרמטרים מתקדמים יותר שאינם מופיעים בפקטה המקורית, הודות לניתוח המעמיק שעושה Wireshark לכל פקטה).

- אם נרצה לפלטר על פרוטוקול מסוים, נוכל פשוט לרשום את שמו ויופיעו פקטות מפרוטוקול זה בלבד.
    - למשל: ip, arp, tcp, http, dns



- ניתן לפלטר על שדה מסוים של פרוטוקול, כאשר אופרטור ההשוואה יכול להיות == (שווה), =! (שונה), > (גדול מ..), < (קטן מ..), contains "..." (בדיקה אם השדה מכיל את המחרוזת "..."), ועוד.

כדי לציין איזה שדה אנו רוצים, נרשום את שם הפרוטוקול, לאחריו נקודה, ולאחר מכן את שם השדה – בצורה הבאה:

<ProtocolName>.<FieldName> Operator <Value>

  - למשל:

ip.src == 192.168.1.1 (הצגת כל הפקטות שכתובת המקור שלהן היא 192.168.1.1)

ip.dst != 192.168.1.1 (הצגת כל הפקטות שאינן מיועדות ל- 192.168.1.1)

ip.addr == 192.168.1.1 (הצגת כל הפקטות שכתובת המקור או כתובת היעד שלהן היא 192.168.1.1)

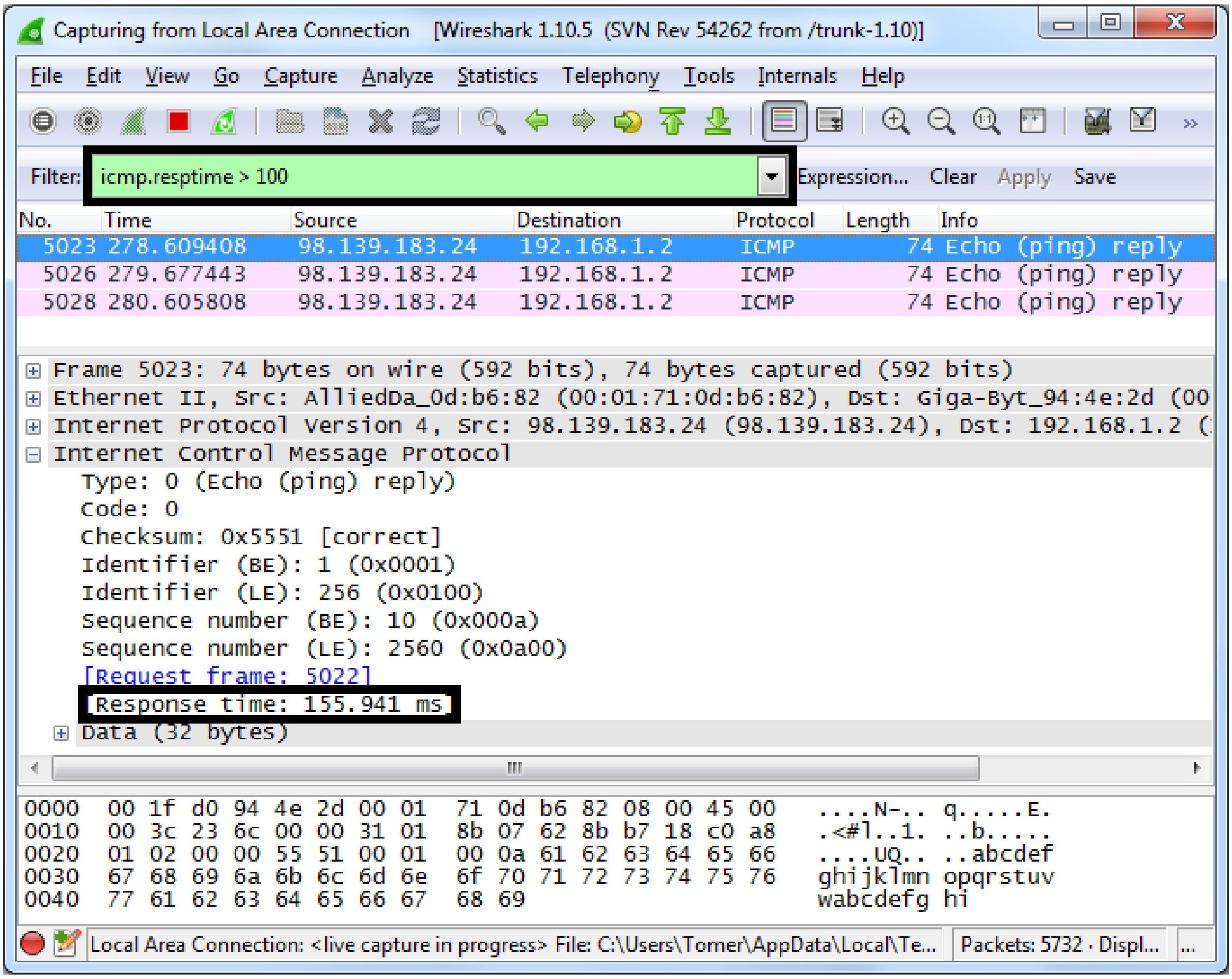
- ניתן לשלב מספר ביטויים ביחד, ולקשר ביניהם בעזרת קשר לוגי: or / and.
  - למשל:

ip.addr == 192.168.1.1 or tcp.port == 22

(הצגת כל הפקטות שנשלחו או התקבלו מכרטיס הרשת שכתובתו 192.168.1.1, או פקטות tcp שנשלחו או התקבלו בפורט 22)

- הניתוח של Wireshark מאפשר לנו להשתמש בשדות שלא באמת קיימים בפקטה, אלא הם פרי ניתוח התוכנה עצמה:
  - למשל:

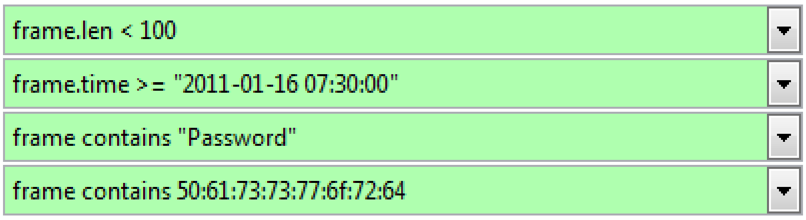
icmp.resptime > 100 (פקטות ICMP שזמן התגובה שלהן היה גדול מ-100 מילישניות)



שימו לב שזמן התגובה הוא אינו שדה אמיתי בפקטה (הוא לא חלק מפרוטוקול ICMP), אלא מחושב בידי Wireshark על-פי ההפרש בין הזמן שבו נקלטה פקטת התשובה לבין הזמן שבו נשלחה פקטת הבקשה. במקרה ש-Wireshark מציג שדה שנובע מהניתוח שלו ולא קיים בפרוטוקול המקורי, השדה יוקף בסוגריים מרובעים – [ ].

- דבר נוסף שחשוב להכיר הוא האובייקט frame (המסמן כל מסגרת שנקלטה בכרטיס הרשת, לא משנה באיזה פרוטוקול היא). כך ניתן לפלטר על פרמטרים כמו אורך המסגרת (frame.len), הזמן שבו היא נקלטה (frame.time) או פשוט על תוכן שמופיע בה במקום מסוים

frame contains "some text"

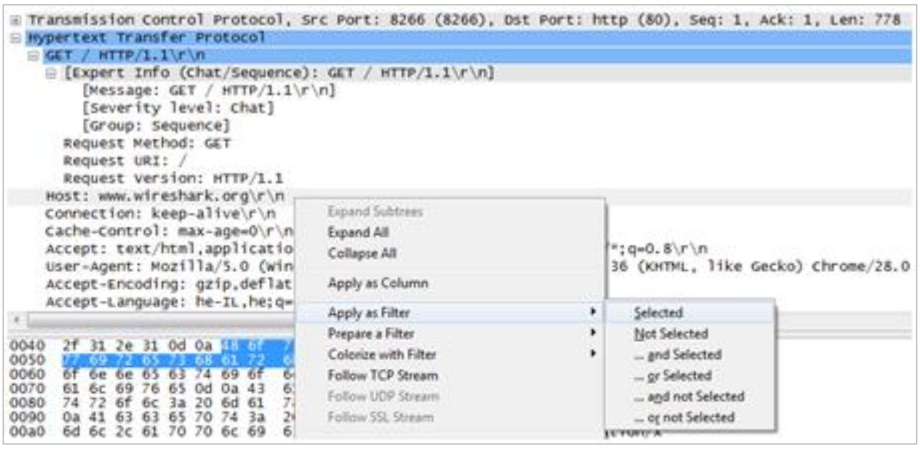


(הדוגמה האחרונה זהה לזו שלפניה, רק שהיא מאפשרת לתת ערכי הקסה של הבתים במקום ייצוג ה-ASCII שלהם)

מגוון הפילטרים ש-Wireshark מציע הוא כל כך רחב, כך שלא נוכל להזכיר כאן את כל הפילטרים הקיימים. ניתן שני טיפים ששווה להכיר אם אתם לא יודעים כיצד לרשום ביטוי כלשהו. הם שימושיים גם למקרה שבו שכחתם איך קוראים לפילטר שאתם מחפשים, וגם כדי ללמוד לבד על עוד פרוטוקולים ושדות בעצמכם:

1. ניתן להיעזר בתפריט ה-Expression, שעוזר לנו לבנות ביטויים כאלו.
2. בעזרת לחיצה ימנית על שדה כלשהו בחלון ניתוח הפקטה, ובחירה ב- Apply As Filter > Selected.

<u>למשל</u>: נניח שאנחנו רוצים לסנן על-פי שדה ה-Host בפרוטוקול HTTP, אך איננו יודעים כיצד קוראים לפילטר הזה. מספיק שנמצא פקטת HTTP אחת שבה מופיע השדה הזה, ונוכל להגדיר אותו כפילטר באמצעות התפריט:



עכשיו ניתן לראות את הפילטר ש-Wireshark יצר עבור השדה הזה:

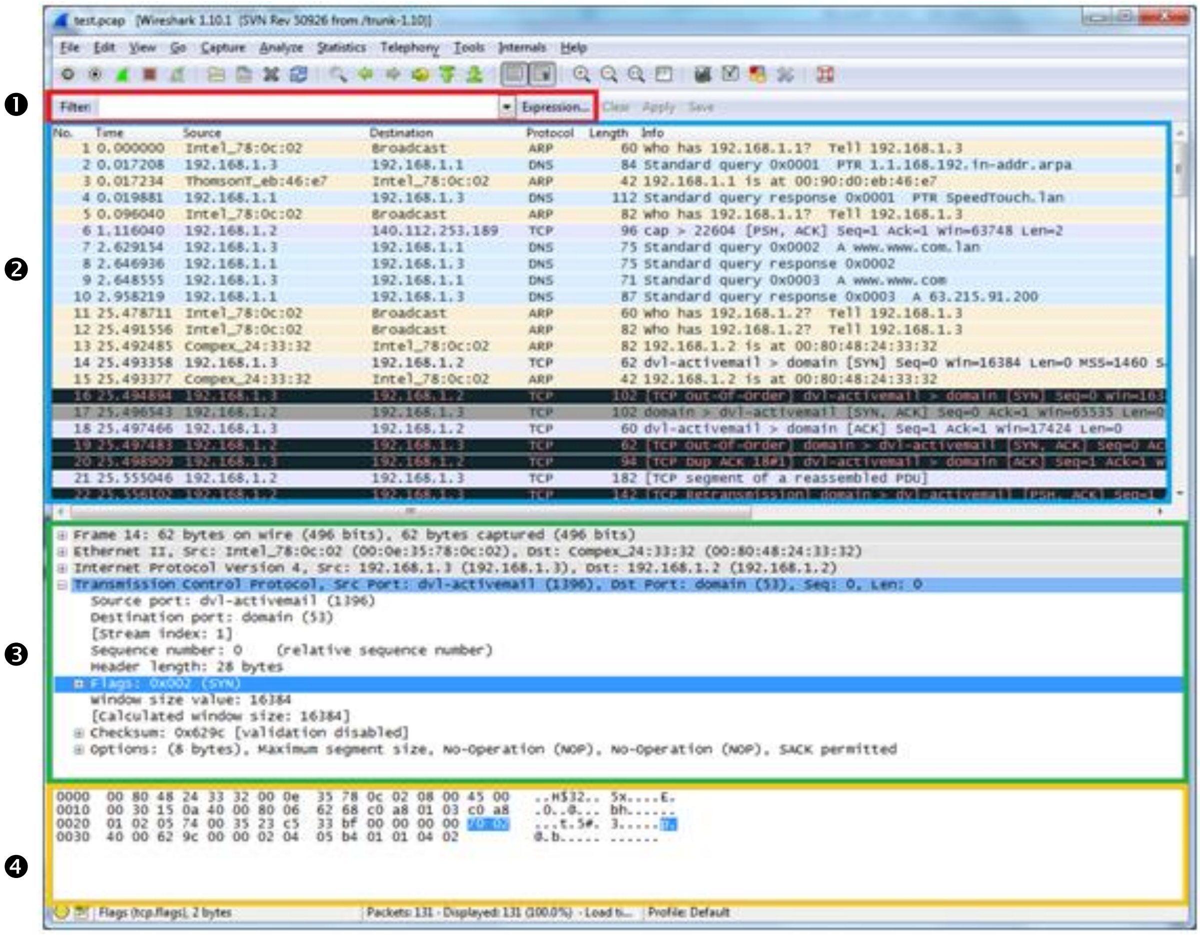
שימו לב שבאמצעות שיטות אלו אתם יכולים **ללמד את עצמכם** כיצד להשתמש ב-Wireshark, לגלות פילטרים חדשים ואפשרויות שלא הכרתם עד כה.

שימוש ב-Wireshark לניתוח מודל חמש השכבות

לאחר פתיחת התוכנה, יתקבל מסך הפתיחה, שמכיל קיצורים שימושיים כמו פתיחת קובץ הסנפה שנשמר בעבר, גישה מהירה לעזרה או התחלת הסנפה חדשה. כדי להתחיל הסנפה חדשה, בחרו את כרטיס הרשת שלכם ולחצו על Start.

כעת נפתח לכם מסך ההסנפה הראשי. אם תמתינו מספר שניות תוכלו לראות שבמרכז המסך מתמלאות להן שורות-שורות (אם לא – כנראה שבחרתם בכרטיס הרשת הלא נכון. נסו לפתוח הסנפה חדשה על כרטיס אחר). השורות הללו מציגות **פקטות (חבילות מידע, Packets)** שכרטיס הרשת שלכם מוציא או מקבל. כזכור מההסבר אודות מודל חמש השכבות, פקטה היא מעין מעטפה שמכילה מוען, נמען ואת תוכן ההודעה – וכך מתאפשרת העברת המידע ברשת ממקום למקום. זוכרים ש[בפרק הראשון](#) הראינו כי נשלחת הודעת בקשה מהדפדפן אל האתר של שרת Facebook, והודעת תגובה מהאתר של Facebook אל הדפדפן? למעשה, הודעות אלו הן פקטות.

כעת נוכל להסתכל על החלקים מהם מורכב מסך ההסנפה הראשי של Wireshark:



1. **באדום** (1) – מסנן תצוגה (Display Filter): בחלון זה ניתן לסנן את הפקטות ולהציג רק את אלו שעונות על תנאי מסוים. כרגע זו עלולה להיראות לכם כיכולת לא ממש חשובה, אבל בפעם הראשונה שתסניפו

תראו כל כך הרבה פקטות – ותבינו שבהרבה מאוד מקרים נרצה לפלטר (לסנן) רק את אלו שמעניינות אותנו. דוגמה לפילטר כזה יכולה להיות "רק התקשורת ביני לבין השרת של Google", כדי להסתיר את כל הגלישות שלי לשאר אתרי האינטרנט.

ניתן לכתוב ביטויים לפילטר בעצמנו, או להשתמש בחלון ה-Expression... שעוזר לבנות פילטרים שאנחנו לא יודעים כיצד לכתוב.

2. **בכחול** (2) – רשימת הפקטות שנקלטו: במרכז המסך ניתן לראות את כל הפקטות שנקלטו דרך כרטיס הרשת (ושעונות על הפילטר שהגדרנו). נפרט על השדות המופיעים באופן ברירת המחדל עבור כל פקטה:
   - .No – מספר הפקטה בהסנפה (מס' סידורי).
   - Time – משך הזמן שעבר מתחילת ההסנפה ועד שנקלטה הפקטה.
   - Source – כתובת המקור של הפקטה. לפקטות IP, תוצג כתובת ה-IP, וזו הכתובת שניתן לראות בדרך כלל בשדה זה.
   - Destination – כתובת היעד של הפקטה. לפקטות IP, תוצג כתובת ה-IP, וזו הכתובת שניתן לראות בדרך כלל בשדה זה.
   - Protocol – באיזה פרוטוקול הפקטה מועברת.
   - Length – אורך הפקטה בבתים (bytes).
   - Info – מידע נוסף על הפקטה. משתנה לפי סוג הפרוטוקול.

3. **בירוק** (3) – ניתוח הפקטה: בחלק זה ניתן לראות ניתוח של פקטה מסומנת מרשימת הפקטות. הניתוח מחלק את הפקטה לשכבותיה השונות, ומציג מידע על כל אחת מהשכבות (עליהן נלמד בהמשך).

4. **בכתום** (4) – תוכן הפקטה: הצגת תוכן הפקטה בייצוג הקס-דצימלי משמאל, ובייצוג ASCII מימין. Wireshark הוא כלי חזק, והוא מקשר לנו בין כל השדות של הפרוטוקול למיקום שלהם בפקטה. שימו לב שלחיצה על בית (byte) כלשהו של הפקטה תקפיץ את חלון הניתוח לחלק הרלוונטי בו נמצא הבית הזה, ולהפך – לחיצה על נתון כלשהו בחלון הניתוח תסמן לכם היכן הוא נמצא בפקטה השלמה ותראה לכם את הייצוג שלו. בהמשך נבין טוב יותר את הקשר בין שני החלונות הללו.

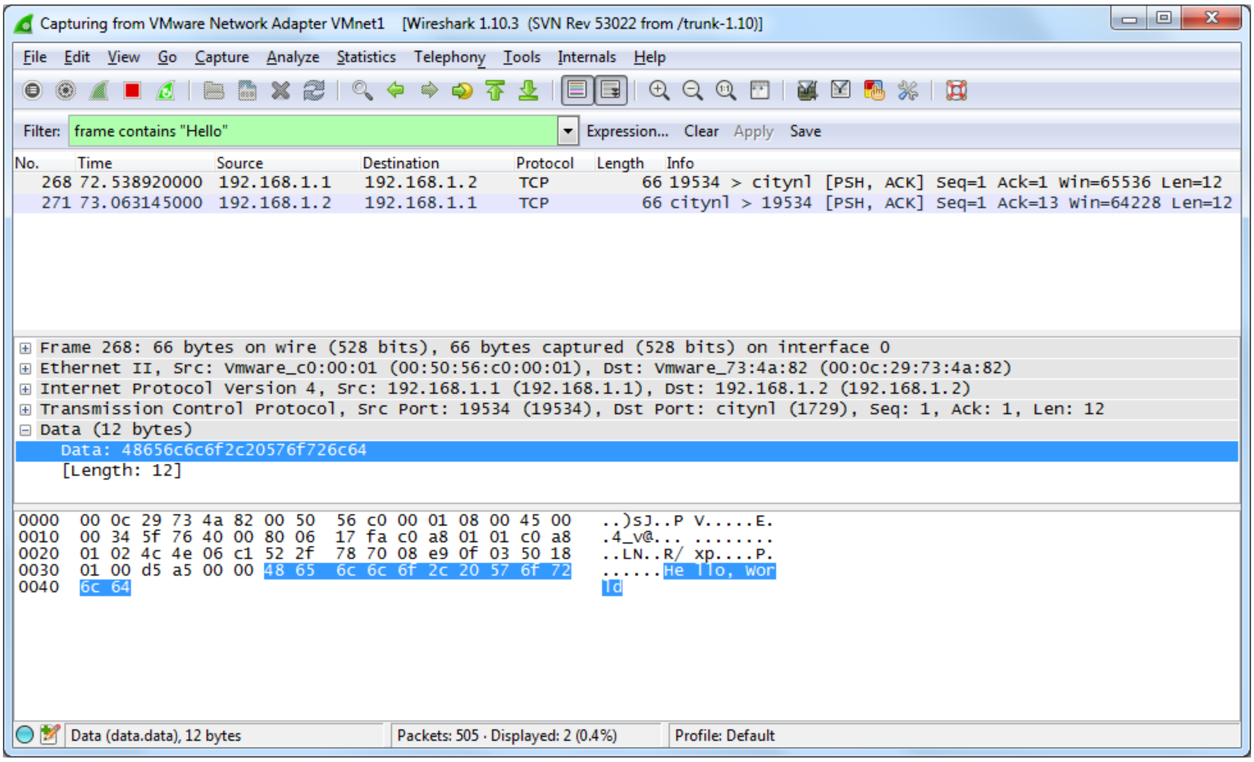
תרגיל 3.1 מודרך – הסנפה של שרת ההדים מהפרק הקודם

בטרם נתמקד בהסנפה, הפעילו את הסקריפטים שכתבתם בתרגילים 2.2 ו-2.5 בפרק הקודם (סקריפט הלקוח וסקריפט שרת ההדים) על מחשבים נפרדים. הריצו את Server.py על מחשב מרוחק ואת Client.py על המחשב הנוכחי (הסיבה לכך שאנחנו מריצים את סקריפט הלקוח וסקריפט השרת על מחשבים נפרדים ולא על אותו מחשב נעוצה בעובדה שהרבה יותר קשה להסניף את המידע שנשלח לאותו המחשב

דרך הכתובת 127.0.0.1 עליה דיברנו קודם)[13]. לאחר שווידאתם כי הסקריפטים של השרת והלקוח מצליחים לתקשר ביניהם על גבי מחשבים נפרדים, הריצו אותם שוב בעוד ההסנפה פועלת ברקע.

עכשיו הגענו לחלק המעניין: כבר ראינו כי המידע שנשלח דרך ה-Sockets מודפס למסך ("Hello, World"), אך האם נוכל למצוא אותו בהסנפה? התשובה היא, כמובן, חיובית – משום שהמידע נשלח דרך כרטיס הרשת שלנו ולכן נקלט בהסנפה. אם הייתם זריזים, אולי הצלחתם לזהות את הפקטות הרלוונטיות מבין כל הפקטות שהוצגו במסך ההסנפה, אך גם אם לא – אנו נשתמש באופציית סינון הפקטות כדי להציג רק את הפקטות הרלוונטיות אלינו. רשמו בשדה ה-Filter את ה-Display Filter הבא:

frame contains "Hello" (הפילטר הזה גורם לכך שיוצגו רק הפקטות שמופיעה בהן המילה Hello)



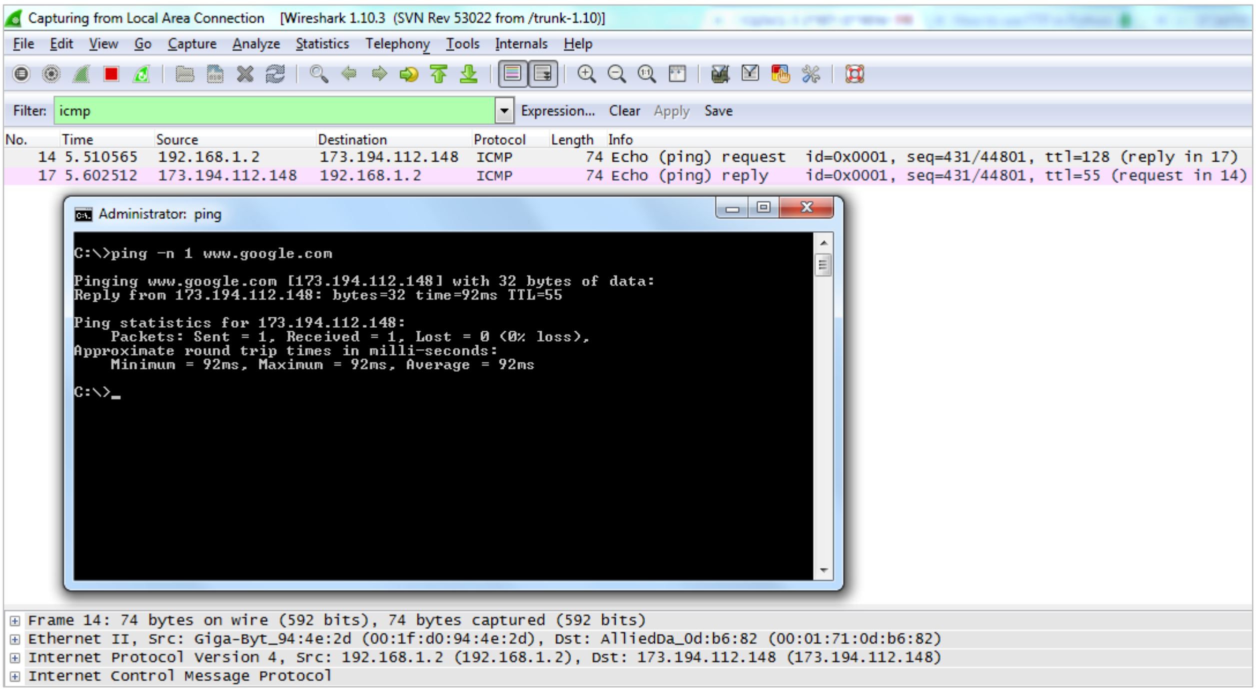
אפשר לראות שמופיעה גם הפקטה שהכילה את המידע מה-client (בעל כתובת ה-IP 192.168.1.1 במקרה שלנו) ל-server (192.168.1.2), וגם הפקטה שחזרה מה-server ל-client!

[13] במערכת ההפעלה Linux (בשונה מ-Windows), ההסנפה על התעבורה שנשלחת למחשב עצמו דרך 127.0.0.1 היא פשוטה וזהה להסנפה על כל כרטיס רשת אחר. גם ב-Windows ניתן להסניף תעבורה שנשלחת אל 127.0.0.1, אולם התהליך מסובך יותר ודורש התקנה של תוכנות חיצוניות. אי לכך, לא נעשה זאת בספר זה.

דוגמה נוספת יכולה להיות הסנפה של פקטת ping, אותו ping עליו דיברנו בפרק הראשון. פתחו חלון cmd והריצו את הפקודה הבאה (וודאו, כמובן, שיש לכם הסנפה ברקע):

ping –n 1 www.google.com

ה-flag בשם n- קובע כי תישלח רק בקשת ping אחת, ולא ארבע בקשות כמו שנשלחות בדרך כלל.



שימו לב לתוצאה שהתקבלה: שלחנו בקשה אחת (המכונה "Echo (ping) request", ניתן לראות זאת בעמודת ה-Info) אל השרת, וקיבלנו ממנו תשובה (Echo (ping) reply) מיד לאחר מכן. שימו לב גם אל כתובות ה-IP של הפקטות – הפקטה הראשונה ובה הבקשה נשלחה מכתובת ה-IP שלנו (192.168.1.2) אל השרת של Google (173.194.112.148), ואילו הפקטה השנייה ובה התשובה נשלחה בדיוק בכיוון ההפוך (מ-173.194.112.148 אל 192.168.1.2).

דרך אגב, הפילטר שהשתמשנו בו כדי להציג אך ורק את פקטות ה-ping הוא הפילטר **icmp**, שהוא הפרוטוקול בו עוברות בקשות ותשובות ping. בהמשך הספר נרחיב את הדיבור על פרוטוקול זה ונלמד לעומק איך ping עובד. בנוסף, אפילו נכתוב כלי דמוי ping בעצמנו!

תרגיל 3.3 – שימוש בסיסי ב-Wireshark בעזרת שליחת בקשת ping

(מספר התרגיל אינו עוקב כדי לשמור על תאימות עם גרסאות קודמות של הספר)

1. פתחו חלון cmd ורשמו **ping 8.8.8.8 –n 1**. הסתכלו על ההסנפה ונסו למצוא את הפקטות שנשלחו (תזכורת: היעזרו בפילטר "icmp", שהוא הפרוטוקול שבו עוברות בקשות ה-ping). כעת, חשבו את ה-round trip (משך הזמן החל מרגע שליחת הבקשה ועד קבלת התשובה) על-פי שדה ה-Time ב-Wireshark (שכזכור מציג את הזמן בו עברו הפקטות בכרטיס הרשת, בפורמט של שניות מתחילת ההסנפה).
בדקו את עצמכם – האם הגעתם לאותה תוצאה שהופיעה בחלון ה-cmd?

2. השתמשו ב-flag בשם l- (שקובע את גודל המידע שיישלח בפקטת ה-ping), והריצו את השורה הבאה:
ping 8.8.8.8 –n 1 –l 500
גודל הפקטה שנשלחה כעת שונה מגודל הפקטה ששלחנו קודם (כאשר לא השתמשנו ב-flag שמציין את גודל המידע שנשלח). נסו למצוא לפחות שני מקומות ב-Wireshark בהם אפשר לראות שגודל הבקשה הזו שונה מגודל הבקשה הקודמת.

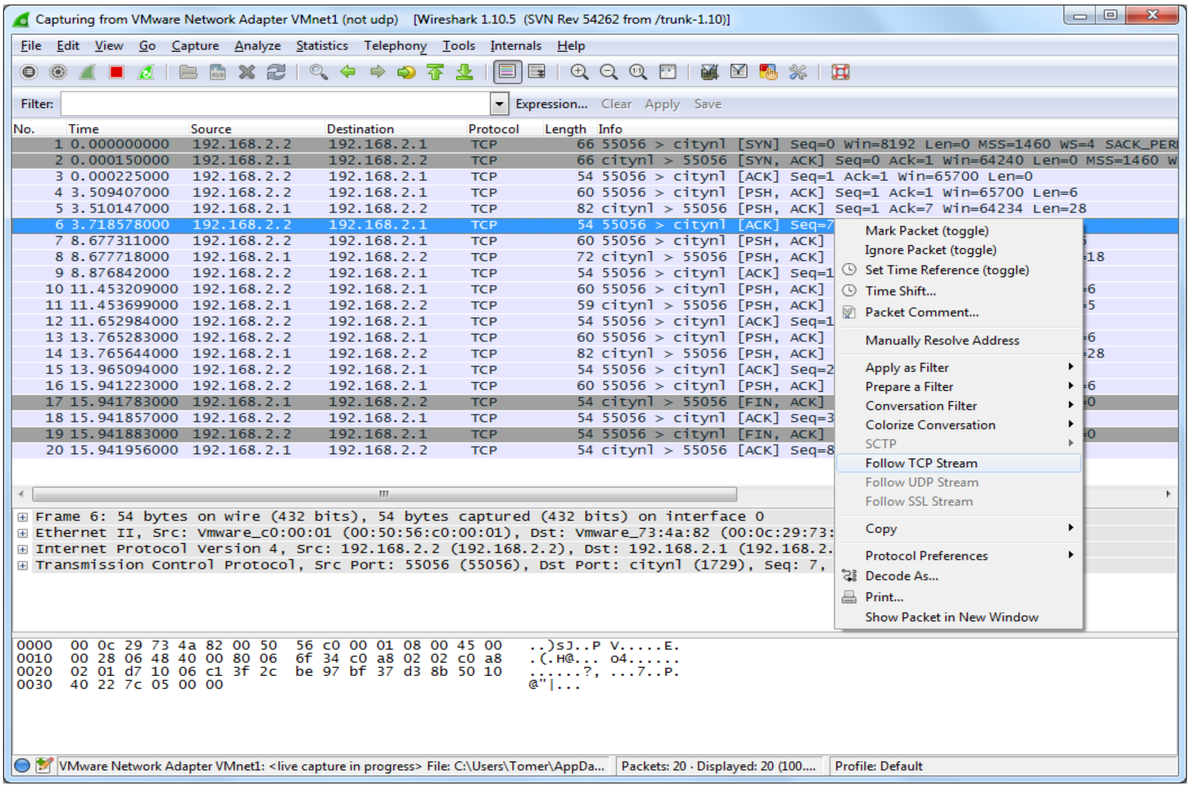
3. הסתכלו על תוכן הפקטה שנשלחה בסעיף 2, בחלון המציג את תוכן הפקטה. בקשת ping מכילה בתוכה מידע, והיא מצפה לקבל את תשובת ה-ping עם אותו מידע בדיוק (בדומה למה שעשיתם בפרק תכנות ב-Sockets כשמימשתם שרת הדים). ב-Wireshark, ניתן לראות את המידע הזה בשדה "Data" של פקטת ה-ping. תוכלו לזהות מהו המידע שנשלח בבקשת ping אל השרת?

Follow TCP/UDP Stream

אופציה נוספת אך חשובה להפליא היא Follow TCP Stream או Follow UDP Stream (השימוש בראשון הוא הרבה יותר נפוץ). אפשרות זו שמופעלת על פקטה, מפלטרת את כל הפקטות השייכות לאותו "Stream" (כלומר נשלחו והתקבלו דרך אותו ה-Socket[14]), ובכך מאפשרת לראות את התעבורה "דרך העיניים" של ה-Socket באפליקציה". נדגים את השימוש באפשרות זו בעזרת התרגיל שכתבתם ב[פרק תכנות ב- Sockets / תרגיל 2.6 – שרת פקודות בסיסי](#):

תרגיל 3.4 מודרך – צפייה במידע של Socket בעזרת Follow TCP Stream

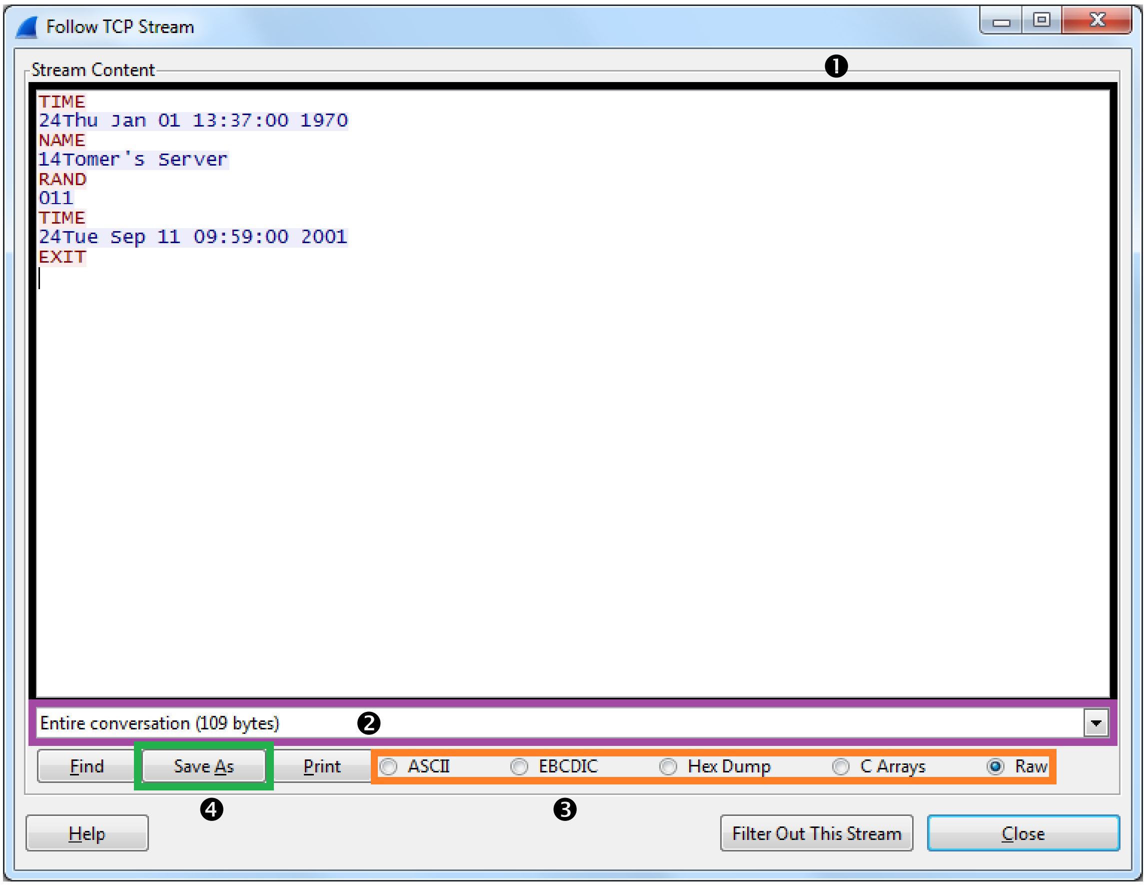
- השתמשו בלקוח שכתבתם בתרגיל שרת פקודות בסיסי (TIME, RAND, NAME, EXIT).
- שנו את קוד הלקוח כך שהוא יפנה אל שרת גבהים שנמצא בכתובת networks.cyber.org.il בפורט 8850 (מה כתובת ה-IP אליה יש לפנות? מצאו אותה בעצמכם בעזרת הכלים שלמדנו).
- התחילו הסנפה חדשה והריצו את הלקוח. פתחו את ההסנפה.
- בחרו את אחת הפקטות שהועברו בין שני המחשבים כחלק מאותו ה-Socket, לחצו על הכפתור הימני של העכבר ובחרו באופציה "Follow TCP Stream".



---

[14] למעשה, הכוונה היא לכל הפקטות שהן חלק מה-Stream בשכבת התעבורה. על המשמעות של מושג זה, וכיצד Wireshark מצליח להבין אילו פקטות שייכות לאיזה Stream – נלמד ב[פרק שכבת התעבורה](#).

- כעת נפתח לכם חלון המציג את כל המידע שעבר על גבי ה-Socket:

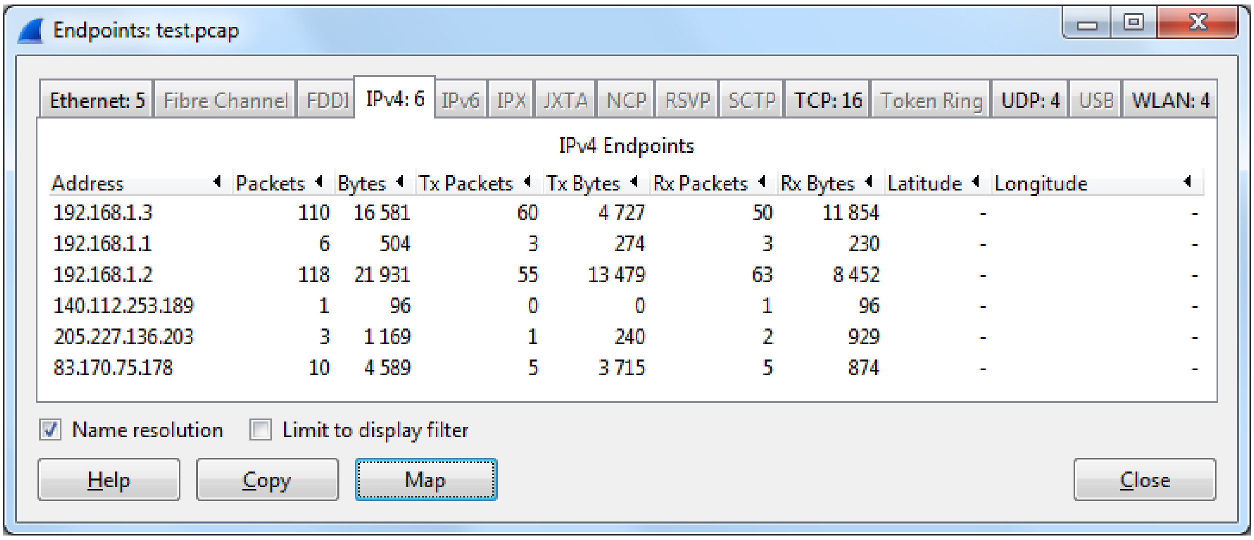


שימו לב, כי מופיע כאן רק ה-Data השייך לשכבת האפליקציה, ללא כל ה-Headerים והפרמטרים ששימשו לחיבור בין שתי נקודות הקצה. אם תסתכלו על המסגרת **השחורה** (1), תראו שכל צד בתקשורת מופיע בצבע אחר – ב**אדום** התקשורת בין הלקוח לשרת (שורות 1,3,5,7,9), וב**כחול** (שורות 2,4,6,8) התקשורת בין השרת ללקוח. ניתן לזהות את הפרוטוקול בו בחרנו להעביר את המידע חזרה מהשרת ללקוח: שליחה של 2 ספרות המסמלות את אורך התשובה, ומיד לאחריהן נשלחת התשובה עצמה.

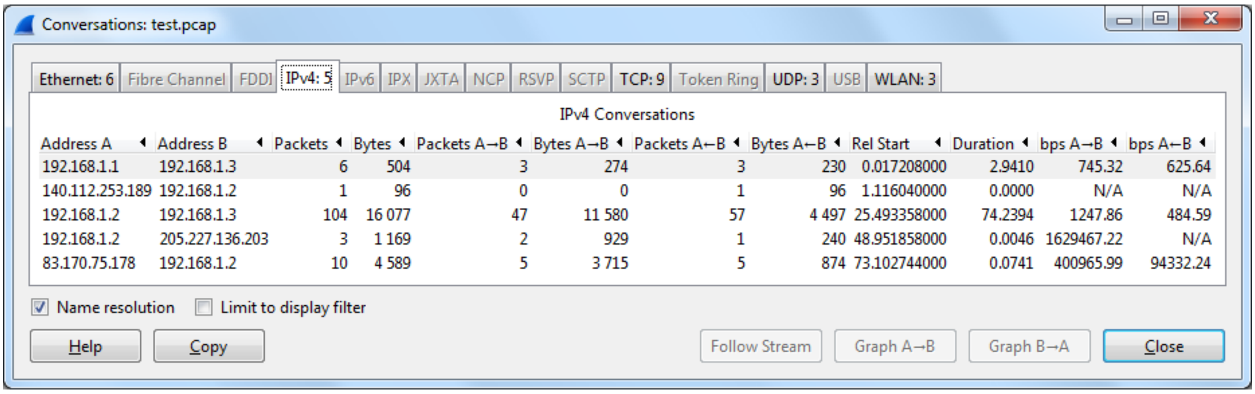
בנוסף, ניתן להגביל את הצפייה רק לאחד הצדדים של התקשורת (המסגרת ה**סגולה**, 2), לבחור את הייצוג בו נרצה לצפות במידע (המסגרת ה**כתומה**, 3) ואף לשמור את המידע לקובץ ובכך לקבל את כל המידע שעבר ב-Socket (המסגרת ה**ירוקה**, 4).

סטטיסטיקות

Wireshark היא תוכנה חכמה – היא מבצעת ניתוח לכל הפקטות שמגיעות ומסיקה מהן מסקנות. כתוצאה מכך, אנו יכולים לקבל תצוגות סטטיסטיות נחמדות שנמצאות תחת תפריט Statistics: אורכי הפקטות בהפנסה (Packet Lengths), גרף שמתעדכן בזמן אמת ומציג את כמות התעבורה (IO Graph) וישויות רשתיות שנקלטו בהפנסה (Endpoints).



בנוסף, יש את חלון ה-Conversations, שמציג את כל השיחות (תקשורת דו-צדדית בין שתי ישויות ברשת) שעלו בהסנפה.



תרגיל 3.5 – זיהוי שרתים בחלון ה-Conversations וה-Endpoints

השתמשו בכלי **ping** שהכרתם בפרקים הקודמים, על מנת לבדוק קישוריות אל google.com, themarker.com. נסו למצוא את השיחות שלכם עם אותם האתרים בחלון ה-Conversations (כבר למדתם כיצד להמיר בין ה-Domain לבין IP, כך שתצטרכו לזהות שיחות בין ה-IP שלכם ל-IP של השרת המרוחק). בנוסף, נסו לזהות אותם גם בחלון ה-Endpoints.

סיכום Wireshark ומודל חמש השכבות

בפרק זה נחשפנו לראשונה לכלי **Wireshark**, שהוא כלי רב-עוצמה שמאפשר לנו לבחון את המידע שיוצא ומתקבל בכרטיס הרשת שלנו. בהמשך הספר נעשה שימוש נרחב ב-Wireshark, כדי ללמוד עוד על הפרוטוקולים השונים ולהבין כיצד הם עובדים.

התחלנו מהסנפה של שרת ההדים שכתבתנו בפרק הקודם, הסנפנו גם שימוש בכלי ping ואפילו סיסמאות שעוברות בגלוי ברשת האינטרנט – הכל כדי שתיווכחו כמה כוח יש ל-Wireshark ומה אפשר לגלות באמצעותו.

לאחר מכן דיברנו על ענן האינטרנט, ועל הצורך לארגן את המידע שעובר בו בצורה טובה. הצגנו את **מודל חמש השכבות**, מדוע צריך אותו ואילו יתרונות הוא מספק. הראינו כיצד השכבות מדברות האחת עם השנייה ואיך המידע עובר בפועל (כלומר כיצד בנויה פקטה במודל חמש השכבות).
סיימנו את החלק ב**הצגה קצרה של כל אחת מהשכבות** (איזה שירות היא מספקת לשכבות שמעליה, ואיזה שירות היא מקבלת מהשכבות שמתחתיה), באופן שיהווה מפת דרכים להמשך הלימוד בספר. כעת, כשנצלול לעומק ונלמד על כל שכבה, תוכלו להבין היכן היא ממוקמת במודל חמש השכבות ומה תפקידה באופן כללי.

לסיום, הצגנו **אפשרויות מתקדמות** של Wireshark כמו התעסקות עם קבצים, מסננים וסטטיסטיקות. אנו מקווים שספגתם קצת מההתלהבות לגבי התוכנה, ושאתם רק מחכים להשתמש בה כדי לחקור וללמוד עוד על נושאים שאתם לא מכירים ברשתות.

בהמשך הספר נשתמש בידע שרכשנו בפרק זה בכדי ללמוד לעומק על שכבות, פרוטוקולים ורכיבי רשת שונים.

פרק 4

שכבת האפליקציה

מצוידים בכלים שלמדנו – הסנפת תקשורת בין מחשבים ותכנות Sockets בפייתון, ולאחר שלמדנו על מודל חמש השכבות, אנחנו מוכנים להתחיל ולחקור את השכבה הראשונה שלנו – שכבת האפליקציה.

ניתן לצפות בסרטון "מבוא לשכבת האפליקציה" בכתובת: http://youtu.be/yftdGTiEP8A.

אפליקציות (**יישומים**, בעברית) – כינוי לתוכנות שבהן אנחנו עושים שימוש במחשב, וזהו מושג שנעשה הרבה יותר נפוץ ומוכר מאז שהתחיל השימוש הנרחב בסמארטפונים ובטאבלטים. גם יישומים שרצים על המחשב שלנו (דוגמה נפוצה – דפדפנים), וגם אפליקציות ב-iPhone או ב-Android שלנו (כמו Whatsapp או האפליקציה של Facebook), עושים שימוש בתקשורת דרך האינטרנט כדי לשלוח ולקבל הודעות (Whatsapp), להעלות תמונות (Facebook / Instagram), לקבל מיילים (Gmail) ועוד.

שכבת האפליקציה היא אוסף הפרוטוקולים בהם עושות האפליקציות שימוש באופן ישיר, והיא מספקת הפשטה מעל תקשורת הנתונים ברשת האינטרנט.

עד סוף הפרק, נבין בדיוק מה המשפט הזה אומר.

מעבר לכך, נכיר לעומק איך עובד פרוטוקול HTTP ואת היכולות שהוא מספק, ונתבונן כיצד אתרים ואפליקציות כמו Facebook או המפות של Google משתמשות בו. בנוסף – נממש בעצמנו שרת אינטרנט, ולקוח שמתקשר איתו. בהמשך, נלמד על פרוטוקול DNS ודרך הפעולה שלו.

פרוטוקול HTTP – בקשה ותגובה

נתחיל עם הפרוטוקול המוכר ביותר של שכבת האפליקציה – HTTP – המשמש לגלישה באינטרנט.

חשוב לציין שפרוטוקול HTTP כמעט אינו בשימוש כיום, הוא הוחלף על ידי פרוטוקול HTTPS (ה-S מציין "Secure", מאובטח). הסיבה לכך שאנחנו לומדים על HTTP ולא HTTPS היא פשוטה: בפרוטוקול HTTPS שכבת האפליקציה היא מוצפנת ולכן מורכב לבחון אותה.

לכן כל עוד נתמקד בשכבת האפליקציה, נעסוק ב-HTTP. ההבנה של השדות השונים של הפרוטוקול רבלנטית גם להבנת HTTPS. כאשר נעבור לשכבות נמוכות יותר, נגלוש גם לאתרים שמשתמשים ב-HTTPS.

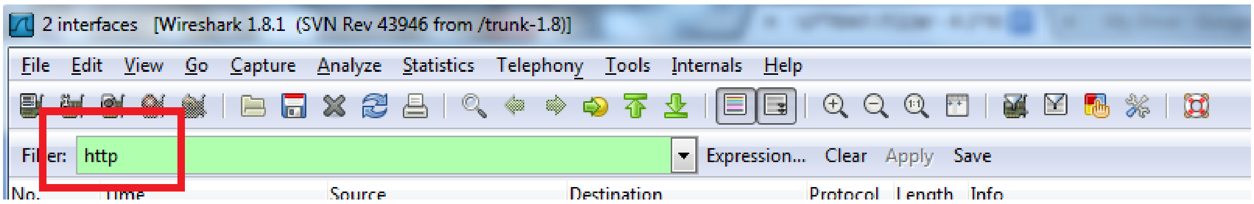
לפני שנתחיל, ניתן לצפות בסרטון "מבוא ל-HTTP" בכתובת: http://youtu.be/BS46e9GYHNI.

מהו משאב רשת?

פרוטוקול HTTP מיועד לאפשר לאפליקציות לגשת ולעשות שימוש במשאבי-רשת באינטרנט. בשלב זה יש לנו הבנה של מהי אפליקציה. כעת ננסה להבין מהו משאב-רשת. למשל: שירות המפות של Google, הוא משאב ברשת. כמו כן, עמוד ה-Facebook (כיום נקרא timeline) שלי הוא משאב ברשת. כך גם חשבון ה-Twitter שלי, וכך גם כתבה ב-TheMarker .

תרגיל 4.1 מודרך – התנסות מעשית בתקשורת HTTP

לצורך כך הפעילו את Wireshark והתחילו הסנפה עם "http" בתור פילטר.



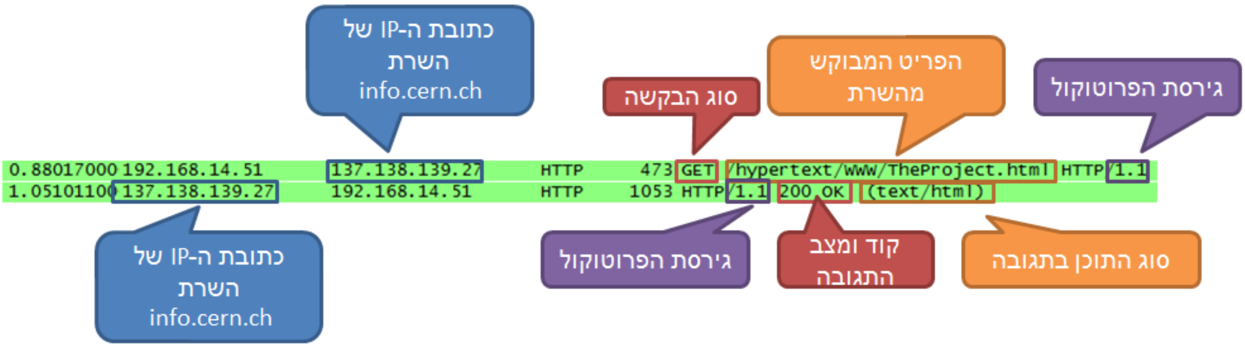
כעת פתחו את הדפדפן החביב עליכם, והכניסו את הכתובת הבאה:

http://info.cern.ch/hypertext/WWW/TheProject.html

לאחר שהדף ייטען, תראו עמוד קצר ובו מספר שורות.

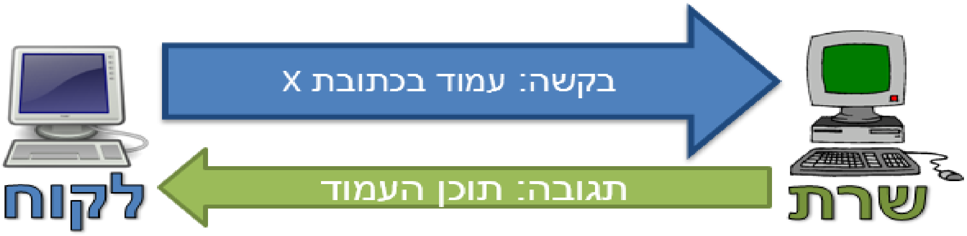
איך הדף נטען בדפדפן שלנו? מה בעצם קרה כאן?

נחזור אל Wireshark כדי למצוא את התשובה – עצרו את ההסנפה, ושימו לב שנוצרו שם שתי שורות:



עיקרון "בקשה-תגובה"

סוג התקשורת שראינו כאן משתמש בשיטת תקשורת שנקראת "בקשה-תגובה", ובה מחשב מבקש מידע מסוים ממחשב אחר על ידי שליחת בקשה מסוימת, ובתגובה המחשב השני מחזיר לו את המידע הרלוונטי. במקרה שלנו, הלקוח מעוניין בדף אינטרנט בעל כתובת כלשהי, ועל כן שולח בקשה אל השרת (שאליה אפשר להתייחס כ-"אנא השב לי את העמוד בכתובת הזו"). לאחר מכן השרת ישלח תגובה (שאליה אפשר להתייחס כ-"הנה עמוד האינטרנט").



כאשר דיברנו בפרק הראשון על שליחת בקשה ל-Facebook וקבלת התשובה ממנו, הפעולה שתארנו הייתה למעשה בקשה ותגובה בפרוטוקול HTTP.

שורת הבקשה של HTTP

הפקטה בשורה הראשונה היא הבקשה שנשלחה מהלקוח (במקרה הזה – מהדפדפן שלכם) אל שרת באינטרנט. בשורה השנייה אנחנו רואים את התשובה שקיבל הדפדפן שלכם מאותו שרת, ואותה ננתח בהמשך.

נתבונן ביחד ונבין איך הפרוטוקול בנוי – שימו לב שבפקטת הבקשה ניתנה הכותרת:

GET /hypertext/WWW/TheProject.html

המילה GET מציינת שזו בקשת HTTP מסוג GET (בהמשך גם נלמד על סוגי הבקשות הנוספים ב-HTTP), שנועדה להביא פריט מידע כלשהו מהשרת באינטרנט שמסתתר מאחורי הכתובת info.cern.ch.

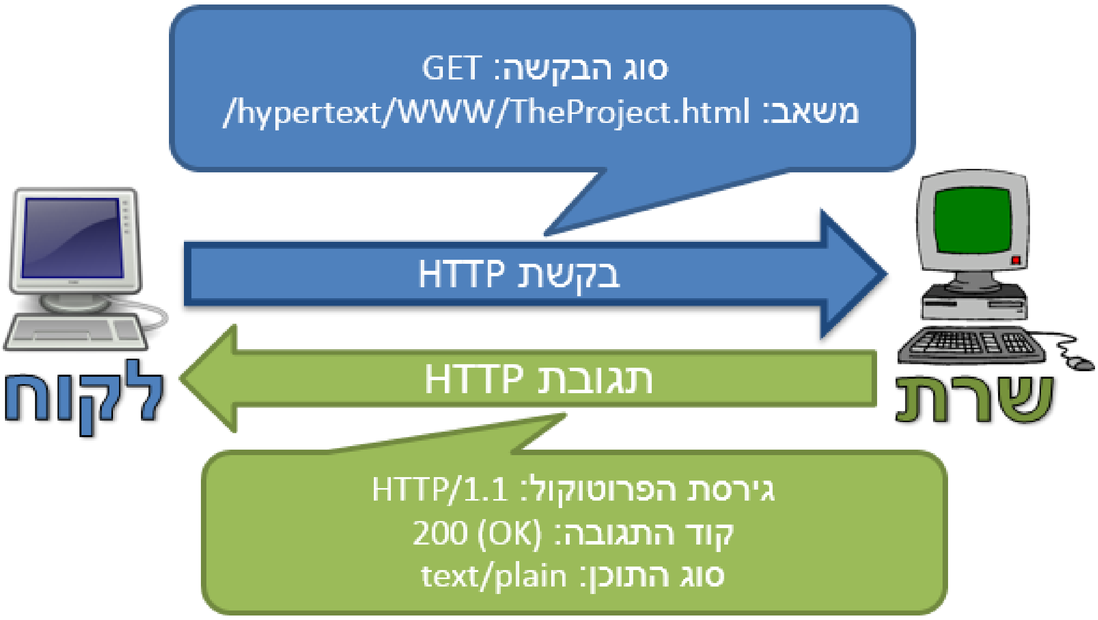
שורת התגובה של HTTP

בשורת התגובה מופיע "HTTP/1.1 200 OK (text/plain)" – למעשה יש בשורה הזו שלושה נתונים:

1. גרסת פרוטוקול HTTP שבה השתמשנו (1.1)
2. מצב התשובה – בפרוטוקול יש מספר קודים מוגדרים מראש כדי לתאר את מצב התשובה; כשמתקבלת תשובה תקינה, נקבל תשובה עם קוד 200 שאומר שהכל בסדר (OK), אם היתה בעיה כלשהי נקבל קוד אחר שאמור לרמז על סוג השגיאה (לדוגמה, קוד 404 המפורסם שאומר שהמשאב המבוקש לא נמצא). את הרשימה המלאה ניתן למצוא בעמוד: http://goo.gl/COC4J7.

3. סוג התוכן שבתשובה – במקרה זה התקבל טקסט. במקרים אחרים ניתן לקבל בתגובה תמונה, סרטון וידאו או קוד.

סיכום ביניים של התקשורת שראינו



תרגיל 4.2

הסניפו באמצעות Wireshark עם פילטר "http" בזמן שתכניסו בדפדפן את הכתובת:

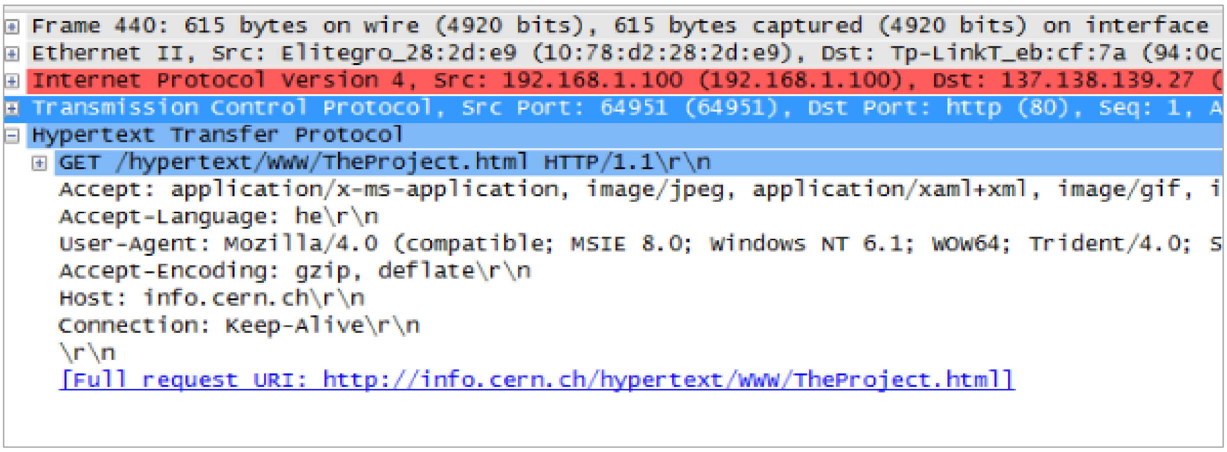
http://www.lilmodtikshoret.com/notfound

מצאו את פקטות הבקשות והתשובות, ובדקו בפקטת התגובה מהו הקוד וסוג התוכן שהתקבל.

מה מסתתר בתוך הבקשה/תגובה? Header‏ים ("מבוא") ב-HTTP

בנוסף לשלושת הפרמטרים שראינו בשורת הבקשה, תקשורת HTTP לרוב מכילה שדות מידע נוספים, מעבר לתוכן שעובר. שדות אלה נשמרים בשורות שמופיעות אחרי שורת הבקשה/תגובה, ולפני ה-Data (תוכן), ונקראות HTTP Headers (בעברית – "שורות כותרת" או "מבוא"). למעשה, כל שורת Header מכילה שם של שדה והערך שלו, כשהם מופרדים על ידי נקודותיים (:).

למשל, בדוגמה הבאה ניתן למצוא, מיד לאחר שורת הבקשה, שדה Header בשם "Accept", לאחריו שדה Header בשם "Accept-Language" (שהערך שלו הוא 'he', כלומר – עברית), ושדות נוספים. שדה ה-Header האחרון בבקשה זו נקרא "Connection":



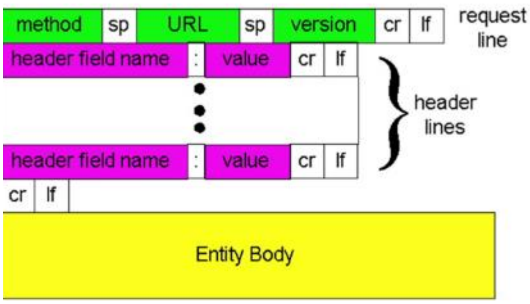
חלק משדות ה-Header יכולים להופיע גם בבקשה וגם בתגובה (למשל: אורך התוכן), חלק יופיעו רק בבקשה (למשל: סוגי התוכן שהלקוח מוכן לקבל בחזרה,"סוג" הלקוח – לדוגמה: דפדפן כרום) וחלק יופיעו רק בתגובה (למשל: "סוג" השרת).

רשימה של שדות Header ניתן למצוא בכתובת:

http://en.wikipedia.org/wiki/List_of_HTTP_header_fields

מבנה פורמלי של בקשת HTTP

```
GET /index.html HTTP/1.1\r\n
Host: www-net.cs.umass.edu\r\n
User-Agent: Firefox/3.6.10\r\n
Accept: text/html,application/xhtml+xml\r\n
Accept-Language: en-us,en;q=0.5\r\n
Accept-Encoding: gzip,deflate\r\n
Accept-Charset: ISO-8859-1,utf-8;q=0.7\r\n
Keep-Alive: 115\r\n
Connection: keep-alive\r\n
\r\n
```



הבקשה היא מחרוזת טקסטואלית, שמורכבת משלושה חלקים: שורת הבקשה, שדות ה-Header ותוכן הבקשה (בשלב זה, נתעלם מהחלק השלישי, שכן בקשות GET לא מכילות תוכן).

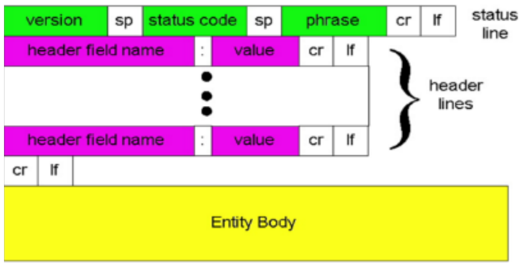
כדי להפריד בין שורה לשורה, נהוג להשתמש ברצף של שני תווים – \r ומיד אחריו \n[15].

מכיוון ששורת הבקשה היא שורה אחת בלבד, ניתן לצפות שמיד עם סיום השורה (הופעת התווים \r\n) יתחילו שדות ה-Header. כל שורה בחלק זה מכילה שדה אחד ואת הערך שלו. לאחר שדה ה-Header האחרון, יופיע כרגיל "סיום שורה" (\r\n), והשורה שלאחר מכן תהיה ריקה – כלומר מיד יופיעו שוב \r\n. זהו סימון לכך ששלב ה-Header הסתיים, וכל שאר המחרוזת תכיל את תוכן הבקשה, בו נדון בשלב מאוחר יותר בפרק זה.

כדי להבין טוב יותר את השרטוט שמוצג בצד ימין: הקיצור sp משמעותו התו רווח (space), הקיצור cr הוא התו \r, והקיצור lf הוא התו \n.

בצד שמאל נתונה דוגמה לבקשת GET – נסו לזהות את החלקים השונים בבקשה (תזכורת: חלק התוכן יהיה ריק), את התווים המפרידים ביניהם, ונסו לזהות שדות Header שדנו בהם בפסקה הקודמת.

מבנה פורמלי של תשובת HTTP




```
HTTP/1.1 200 OK\r\n
Date: Sun, 26 Sep 2010 20:09:20 GMT\r\n
Server: Apache/2.0.52 (CentOS)\r\n
Last-Modified: Tue, 30 Oct 2007 17:00:02 GMT\r\n
ETag: "17dc6-a5c-bf716880"\r\n
Accept-Ranges: bytes\r\n
Content-Length: 2652\r\n
Keep-Alive: timeout=10, max=100\r\n
Connection: Keep-Alive\r\n
Content-Type: text/html; charset=ISO-8859-1\r\n
\r\n
data data data data data ...
```

בדיוק כמו הבקשה, גם תגובת HTTP היא מחרוזת טקסטואלית שבנויה בדיוק באותה צורה – אותם שלושה חלקים, ואותה דרך להפריד ביניהם. עברו על הדוגמה בצד שמאל, וודאו שאתם מזהים את החלקים והמפרידים ביניהם, חפשו שדות Header מוכרים ונסו להבין את המשמעות שלהם.

בדוגמאות שראינו עד כה, יש משמעות גם לחלק השלישי – התוכן בתגובת ה-HTTP (שם מועבר תוכן דף האינטרנט) – ועל כך בסעיף הבא.

---

[15] משמעות התו \r הוא carriage return, לרוב מיוצג על ידי הסמל ↲, ומקורו במכונות הכתיבה. תפקידו של מקש זה הוא להחזיר את ראש הכתיבה לתחילת השורה. משמעות התו \n הוא line feed – ירידת שורה. צירוף שני המקשים הללו מאפשר להתחיל שורת כתיבה חדשה.

תוכן (מידע – Data) ב-HTTP

ראינו תקשורת בסיסית של בקשה ותגובה, אבל למרות שקוד התגובה, סוג התוכן ושדות ה-Header משמשים כמידע די מעניין, אנחנו (בתור משתמשים) מעוניינים בתוכן עצמו. בסופו של דבר, האפליקציה צריכה לקבל את התוכן כדי להציג לנו תמונה, הודעה או מסמך.

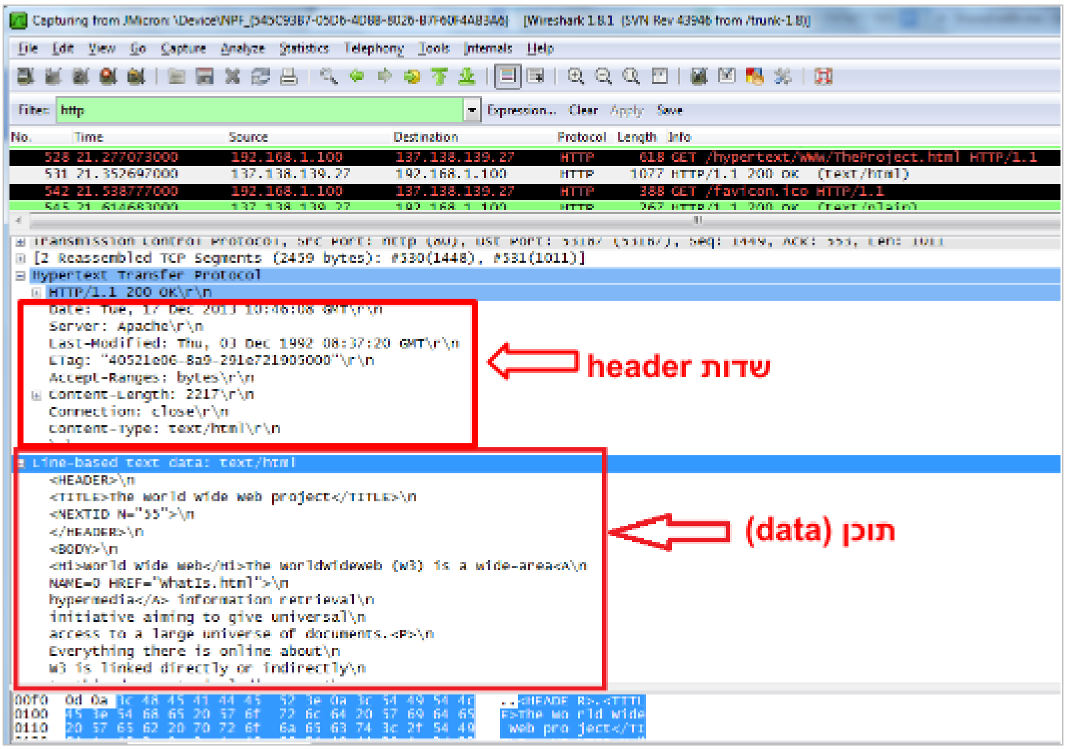
לכן, כשאפליקציה מחליטה לבקש משאב מסוים, היא תבדוק את קוד התגובה כדי לוודא שהיא תקינה – למשל, תוודא שהתקבל קוד 200 OK (כמו שראינו בדוגמה הקודמת). אם למשל התקבל קוד 404 (משאב לא נמצא), היא תציג הודעת שגיאה מתאימה.

לאחר מכן, האפליקציה תבדוק את סוג התוכן כדי לוודא שהוא תואם את מה שהיא ציפתה לקבל – למשל, אם היא ציפתה לקבל תמונה וחזר תוכן מסוג text, כנראה שיש בעיה.

לבסוף, האפליקציה תציג את התוכן שנמצא בתגובה – למשל תנגן סרטון וידאו או תציג הודעת טקסט.

התבוננות מודרכת בתגובת HTTP

הגיע הזמן שנתבונן בתוכן של תגובת HTTP – נחזור לדוגמה שראינו ב-Wireshark, ועכשיו "נפתח" את פקטת התגובה (ונראה את מה שקראנו לו בתרשים "תגובת HTTP" – ... data data data):



הבנו קודם לכן שכאשר שולחים בקשת GET, מבקשים משאב ספציפי מן השרת. גם למשל, כאשר מבקשים את המשאב a.html, הדפדפן שולח GET בצורה הבאה:

GET /a.html HTTP/1.1

במידה שלא נבקש אף משאב בצורה ספציפית, הדפדפן יפנה למשאב שנמצא בכתובת "/", כך:

GET / HTTP/1.1

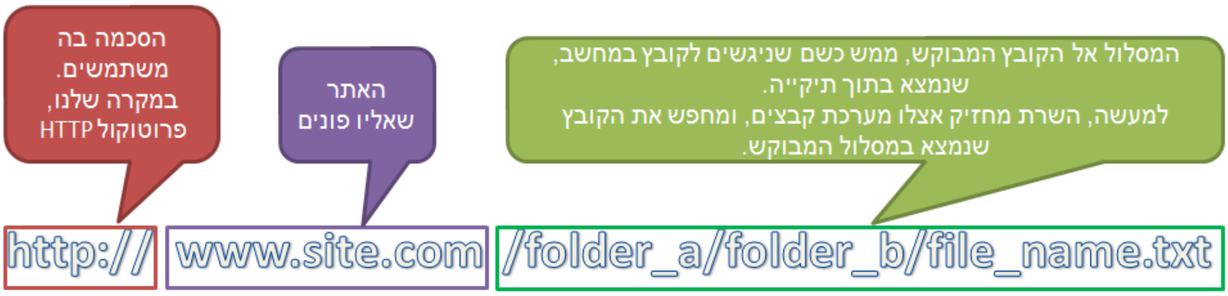
כאשר אנו גולשים בדרך כלל לאתר אינטרנט, איננו מציינים משאב ספציפי. אי לכך, אנו מבקשים למעשה את המשאב אשר נמצא בכתובת "/". כשהשרת מקבל פנייה למשאב הזה, הוא יכול להציג את העמוד שברצונו להציג בעת גלישה של משתמש לאתר.

פרוטוקול HTTP – תכנות צד שרת בפייתון, הגשת קבצים בתגובה לבקשת GET

אם ניזכר לרגע בפרק תכנות Sockets /שרת-לקוח, נבין ששני סוגים של רכיבים משתתפים בכל תקשורת כזו – צד השרת, שמספק גישה למשאבים (נניח, מיילים), וצד הלקוח, שהוא למעשה אפליקציית הקצה שבה אנחנו משתמשים, ומבקש משאבים מן השרת. בדוגמה שראינו זה עתה, המשאבים היו דפי אינטרנט טקסטואליים, אותם סיפק השרת בתגובה לבקשה של הלקוח. למעשה, השרת עונה לבקשות שמגיעות מהרבה לקוחות, ובינתיים אנחנו מפשטים ומדברים על לקוח בודד. נדבר בהמשך על ההשלכות של "טיפול" בלקוחות רבים.

מעט רקע היסטורי – בתחילת דרכה של השימוש הביתי ברשת האינטרנט, בסביבות שנות ה-90', שרתי אינטרנט שימשו בעיקר ל"הגשה" של עמודי תוכן סטטיים. הכוונה ב"הגשה" היא להעביר את התוכן של הקובץ שנמצא בכתובת שביקשה האפליקציה. למשל, בדוגמה שראינו בחלק הקודם, השרת הגיש לדפדפן קובץ טקסט.

נתעכב לרגע כדי להבין איך בנויות הכתובות הללו: לכל אתר ברשת יש כתובת ייחודית, שמכונה **URL** – ראשי תיבות של Universal Resource Locator (בעברית: "כתובת משאב אוניברסלית"). למעשה, ה"כתובות" שהכנסנו לדפדפן בסעיף הקודם היו URLים. על משמעות המילה "משאב" נרחיב עוד בהמשך. URL בסיסי בנוי באופן הבא:



במקרה הזה, השרת שמאחורי הכתובת www.site.com יחפש בתיקייה folder_a תיקייה בשם folder_b, ובתוכה יחפש קובץ בשם file_name.txt. הפנייה מתבצעת בסכמה של HTTP[16].
אם אכן קיים קובץ כזה, השרת יגיש אותו בתור תגובה לבקשה.

כדי להבין טוב יותר את הרעיון, תממשו כעת בעצמכם שרת כזה, שמגיש קבצים בתגובה לבקשות.
אל דאגה, התרגיל מפורט, ומפורק למשימות קטנות מאוד.

לפני כן, נסביר מעט יותר על המימוש של **root directory**. כפי שציינו קודם, הפנייה למשאב מתבצעת כמו במערכת קבצים – פנייה ל-"/folder_a/folder_b/file_name.txt" למעשה פונה לקובץ file_name.txt בתיקייה folder_b שבתיקייה folder_a. אך היכן נמצאת תיקייה folder_a? היא נמצאת בתיקיית השורש, root directory, של האתר. בפועל, התיקייה הזו היא פשוט תיקייה כלשהי במחשב שהמתכנת הגדיר אותה בתור root directory. בחירה נפוצה היא להגדיר את C:\wwwroot בתור ה-root directory. כעת, כאשר הלקוח פונה ושולח בקשה מסוג:

GET /folder_a/folder_b/file_name.txt HTTP/1.1

הבקשה תתבצע למעשה לקובץ הבא אצל השרת[17]:

C:\wwwroot\folder_a\folder_b\file_name.txt

תרגיל 4.4 – כתיבת שרת HTTP

לאחר כל אחד מהשלבים הבאים, הקפידו לבדוק את השרת שלכם על ידי הרצה של התוכנית, ושימוש בדפדפן כלקוח; היזכרו במשמעות של הכתובת 127.0.0.1 אותה הזכרנו בפרק תכנות ב-Sockets – הכתובת שאליה נתחבר באמצעות הדפדפן תהיה http://127.0.0.1:80 (כאשר 80 הוא הפורט בו נשתמש). על מנת לבדוק את הפתרון שלכם, אנו ממליצים להוריד אתר לדוגמה מהכתובת:

---

[16] ברוב המקרים בסכמה יצוין פרוטוקול – כגון HTTP או FTP. עם זאת, במקרים מסוימים, היא לא תכלול פרוטוקול, כמו הסכמה "file".

[17] זאת בהנחה שהשרת מריץ מערכת הפעלה Windows. כאמור, במערכות הפעלה שונות ה-path עשוי להיראות בצורה שונה.

https://data.cyber.org.il/networks/webroot.zip. העתיקו את תוכן קובץ ה-ZIP אל תיקייה כלשהי (כמובן שיש לפתוח את הקובץ) והשתמשו בה בתור ה-root directory שלכם. המטרה היא שהשרת ישלח ללקוח את index.html ויתמוך באפשרויות השונות שיש בעמוד אינטרנט זה.

1. כתבו שרת המחכה לתקשורת מהלקוח בפרוטוקול TCP בפורט 80. לאחר סגירת החיבור על ידי הלקוח, התוכנית נסגרת.
2. הוסיפו תמיכה בחיבורים עוקבים של לקוחות. כלומר, לאחר שהחיבור מול לקוח נסגר, השרת יוכל לקבל חיבור חדש מלקוח.
3. גרמו לשרת לוודא כי הפקטה שהוא מקבל היא HTTP GET, כלומר – ההודעה שהתקבלה היא מחרוזת מהצורה שראינו עד כה: מתחילה במילה GET, רווח, URL כלשהו, רווח, גרסת הפרוטוקול (HTTP/1.1), ולבסוף התווים \r ו- \n.
   - אם הפקטה שהתקבלה אינה HTTP GET – סגרו את החיבור.
4. בהנחה שהשרת מקבל בקשת HTTP GET תקינה ובה שם קובץ, החזירו את שם הקובץ המבוקש אל הלקוח. בשלב זה, החזירו את שם הקובץ בלבד, ולא את התוכן שלו.
   - שימו לב שב-Windows משתמשים ב-"\" כמפריד בציון מיקום קובץ, בעוד שבאינטרנט וגם בלינוקס משתמשים ב-"/".
   - הערה: את שם הקובץ יש להעביר בתור שם משאב מבוקש, ולא ב-Header נפרד.
   - הערה נוספת: בשלב זה, אל תעבירו Headerים של HTTP כגון הגירסה או קוד התגובה.
5. כעת החזירו את הקובץ עצמו (כלומר, את התוכן שלו).
   - אם מתבקש קובץ שלא קיים – פשוט סגרו את החיבור (היעזרו ב- os.path.isfile).
   - הערה: בניגוד לכמה מהתרגילים הקודמים, כאן יש לשלוח את כל הקובץ מייד, ולא לחלק אותו למקטעים בגודל קבוע (כפי שעשינו, למשל, בתרגיל 2.7).
6. הוסיפו את שורת התגובה ו-Headerים של HTTP:
   - גרסה HTTP 1.0.
   - קוד תגובה: 200 (OK).
   - השורה Content-Length: (מלאו בה את גודל הקובץ שמוחזר).
7. במקרה שבו לא קיים קובץ בשם שהתקבל בבקשה, החזירו קוד תגובה 404 (Not Found).
8. אם השרת מקבל בקשת GET ל-root (כלומר למיקום "/") – החזירו את הקובץ index.html (כמובן, וודאו שקיים קובץ כזה; תוכלו ליצור קובץ בשם index.html שמכיל מחרוזת קצרה, רק לשם הבדיקה).
9. אם השרת מקבל בקשות לקבצים מסוגים שונים, הוסיפו ל-Header של התשובה את השדה Content Type, בהתאם לסוג הקובץ שהתבקש. תוכלו להעזר בנתונים הבאים:
   - קבצים בסיומת txt או html:

Content-Type: text/html; charset=utf-8

   - קבצים בסיומת jpg:

Content-Type: image/jpeg

- קבצים בסיומת js:

Content-Type: text/javascript; charset=UTF-8

- קבצים בסיומת css:

Content-Type: text/css

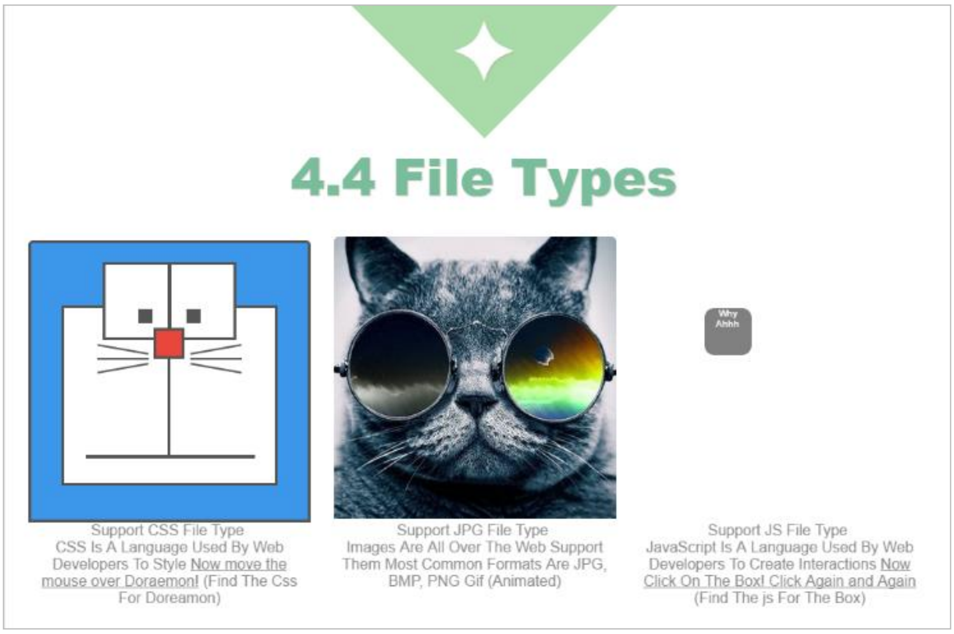
10. כעת הוסיפו תמיכה במספר Status Codes נוספים (היעזרו בוויקיפדיה):

1) 403 Forbidden – הוסיפו מספר קבצים שאליהם למשתמש אין הרשאה לגשת.

2) 302 Moved Temporarily – הוסיפו מספר קבצים שהמיקום שלהם "זז". כך למשל, משתמש שיבקש את המשאב page1.html, יקבל תשובת 302 שתגרום לו לפנות אל המשאב page2.html. לאחר מכן, הלקוח יבקש את המשאב page2.html, ועבורו יקבל תשובת 200 OK.

3) 500 Internal Server Error – במקרה שהשרת קיבל בקשה שהוא לא מבין, במקום לסגור את החיבור, החזירו קוד תגובה 500.

נסו את השרת שלכם באמצעות הדפדפן- גרמו לשרת לשלוח את המידע שנמצא ב-webroot ותוכלו לצפות באתר הבא:



*אתר בדיקת תרגיל שרת HTTP – קרדיט **תומר טלגם***

שלד קוד של תרגיל כתיבת שרת HTTP

קוד של שרת HTTP הוא קוד מורכב יחסית לתרגילים קודמים ולכן מומלץ לתכנן אותו מראש ולחלק אותו לפונקציות. יש דרכים רבות לכתוב את קוד השרת, תוכלו להתבסס על שלד התוכנית הבא:

https://data.cyber.org.il/networks/http_server.py

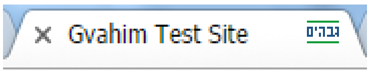
המקומות שעליכם להשלים בעצמכם נמצאים תחת הערה "TO DO". שימו לב שכדי שהתוכנית תעבוד תצטרכו להוסיף לה קבועים ופונקציות נוספות, השלד אמור רק לסייע לכם להתמקד.

טיפים לכתיבה:

1. שימו לב שהתשובות שאתם מחזירים הם לפי כל השדות של HTTP. אל תפספסו אף סימן רווח או ירידת שורה...
2. לעיתים עלול להיות מצב שבו הן השרת והן הדפדפן מצפים לקבל מידע. כדי לצאת ממצב זה, ניתן להגדיר SOCKET_TIMEOUT. שימו לב שאם הזמן שהגדרתם עבר, תקבלו exception. עליכם לטפל בו כדי שהשרת לא יסיים את הריצה.

הוספת אייקון לדפדפן

מרבית אתרי האינטרנט כוללים אייקון מעוצב שמופיע בלשונית של הדפדפן. גם הדפדפן שלכם יבקש את האייקון מהשרת. כדי למצוא היכן נמצא האייקון, תוכלו לפתוח את העמוד index.html בתוכנת notepad++ ולחפש את favicon.ico.



רוצים ליצור לעצמכם אייקון אישי? תוכלו להשתמש באתר http://www.favicon.cc/

תוכלו להעלות לאתר תמונה כלשהי או להמציא אייקון משלכם. לאחר שסיימתם לעצב אייקון, לחצו על כפתור download icon ושימרו אותו במקום המתאים.

כלי דיבוג – breakpoints

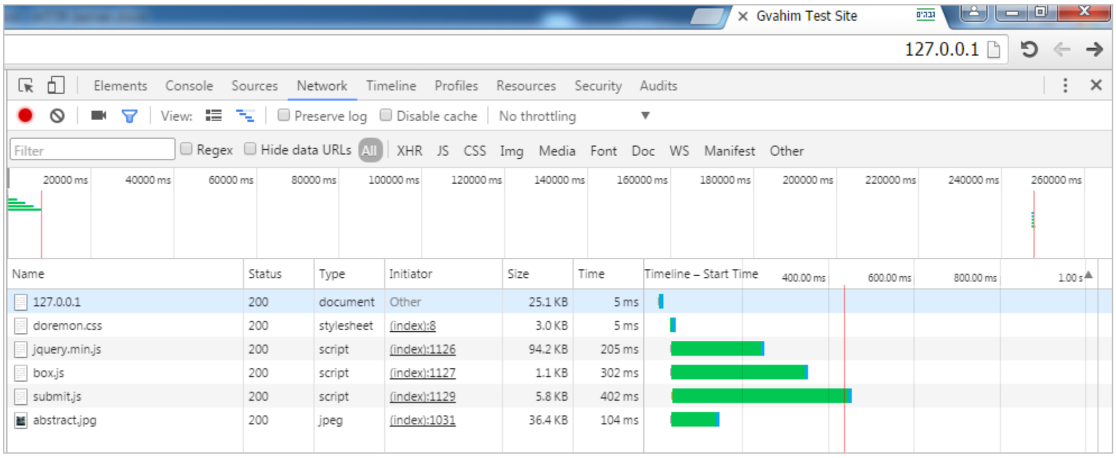
שימוש ב-breakpoints לטובת דיבוג קוד השרת שלכם הוא הכרחי. צרו breakpoints במיקום שבו הקוד עדיין מבוצע באופן תקין ועיקבו אחרי הערכים שמקבלים משתנים ושפונקציות מחזירות כדי לבדוק בעיות בקוד שלכם.

באמצעות בדיקת ערכי משתנים תוכלו לבדוק, לדוגמה, מה הבקשה ששלח הלקוח.

```
    while True:
        client_socket, client_address = server_socket.accept()
        print('New connection received')
        client_socket.settimeout(SOCKET_TIMEOUT)
        handle_client(client_socket)

```

כלי דיבוג – דפדפן כרום

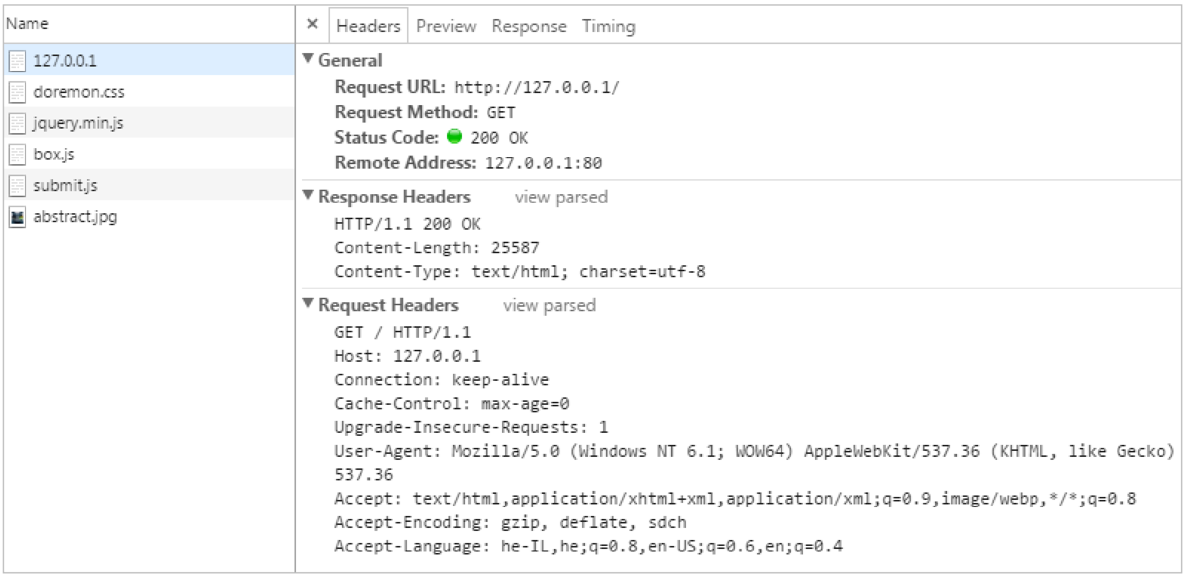
דפדפן כרום כולל אפשרות לעקוב אחרי כל התעבורה בין הדפדפן לשרת. איך עושים את זה?
בתוך הדפדפן לחצו על F12 ולאחר מכן על טאב network. ייפתח לכם המסך הבא:



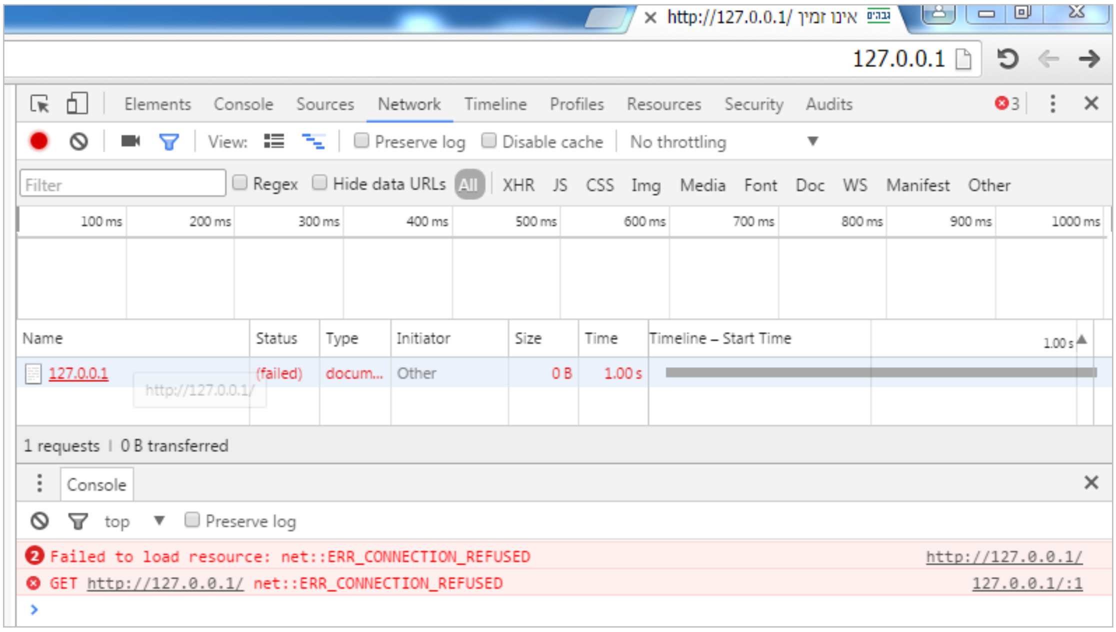
לחצו על הנקודה האדומה כדי להתחיל הקלטה חדשה.

אם תעמדו על שם של קובץ כלשהו תוכלו לקבל פרטים נוספים בטאבים החדשים שמופיעים לצידו – Headers, Preview, Response, Timing.

המידע המעניין ביותר לטובת דיבוג השרת שלכם מופיע ב-headers, שם ניתן לראות את הבקשות שנשלחו לשרת ואת התשובות של השרת.



אם מסיבה כלשהי לא התקבלו מהשרת כל הקבצים, תוכלו לראות מה לא התקבל באזור מיוחד שמוקצה לתיעוד שגיאות. לדוגמה, כאשר השרת לא מקבל את בקשת ההתחברות של הלקוח:



כלי דיבוג – Wireshark

כלי נפלא זה, איתו עשינו היכרות בעבר, יכול לספק לכם את כל המידע שעבר בין השרת ללקוח שלכם – בתנאי שתריצו את השרת ואת הלקוח על שני מחשבים נפרדים, שמחוברים ע"י ראוטר או switch כפי שוודאי יש לכם בבית.

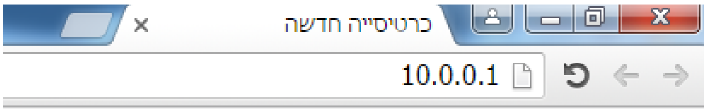
בשלב ראשון, וודאו שהשרת שלכם מאזין לכתובת IP 0.0.0.0 (ולא 127.0.0.1). כתובת 0.0.0.0 אומרת למערכת ההפעלה שלכם לשלוח לשרת שבניתם לא רק פקטות שמגיעות מתוך המחשב עצמו (127.0.0.1) אלא גם פקטות שמגיעות ממחשבים אחרים – בתנאי שהם פונים לפורט הנכון.

לאחר מכן בדקו באמצעות ipconfig מה כתובת ה-IP של השרת שלכם (שימו לב, כתובת IP ברשת הפנימית שלכם, בהמשך נלמד מה היא כתובת פנימית). בדוגמה הבאה הכתובת היא 10.0.0.1:

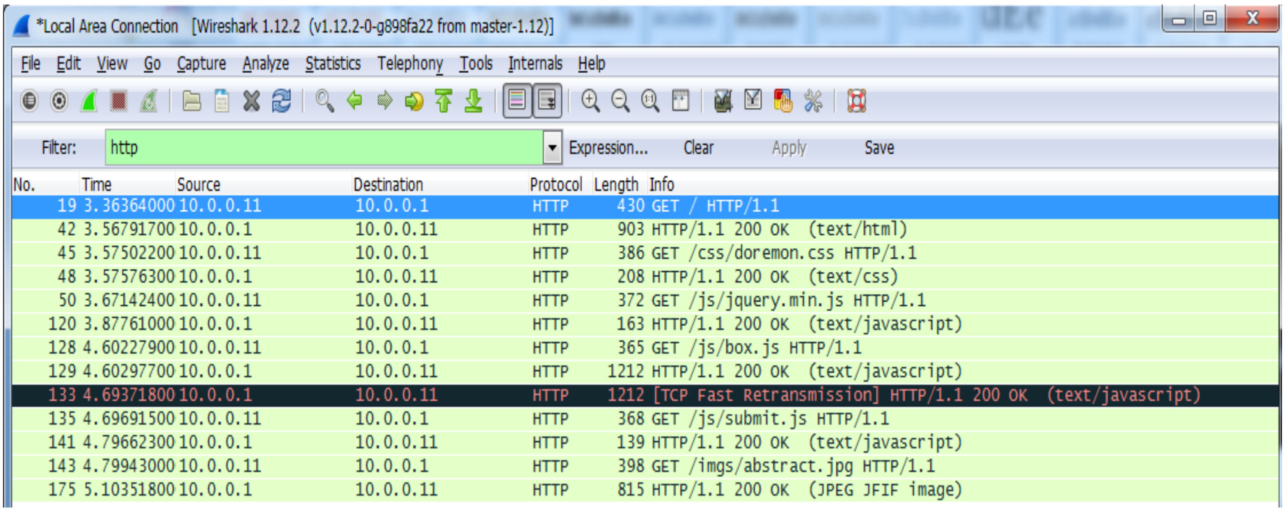
```
Copyright (c) 2009 Microsoft Corporation.  All rights reserved.
C:\Users\ADMIN>ipconfig
Windows IP Configuration

Ethernet adapter Local Area Connection:

   Connection-specific DNS Suffix  . : Home
   Link-local IPv6 Address . . . . . : fe80::48ac:d008:caaf:7415%12
   IPv4 Address. . . . . . . . . . . : 10.0.0.1
   Subnet Mask . . . . . . . . . . . : 255.255.255.0
   Default Gateway . . . . . . . . . : 10.0.0.138
```

לפני שנתקדם לשלב הבא, הפעילו wireshark.

כעת עליכם לקבוע בהתאם את כתובת ה-IP שהלקוח פונה אליה:

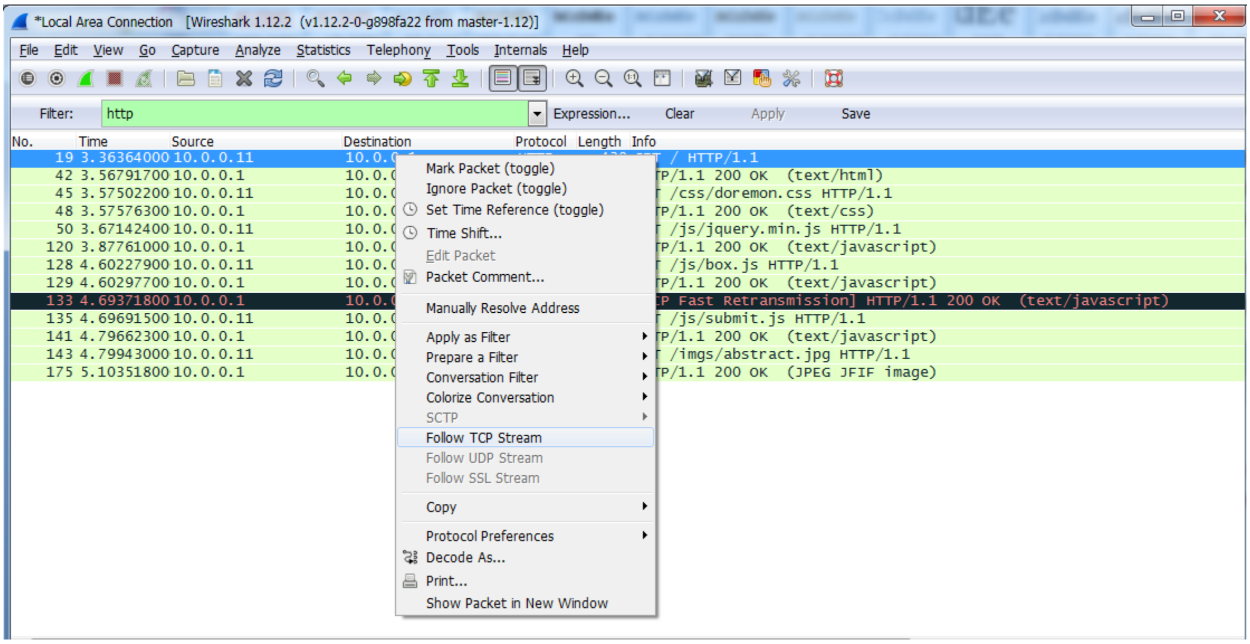


תוכלו למצוא בתוך wireshark את התעבורה בין הדפדפן בלקוח לבין שרת ה-HTTP שלכם. נפלטר את התעבורה לפי http:



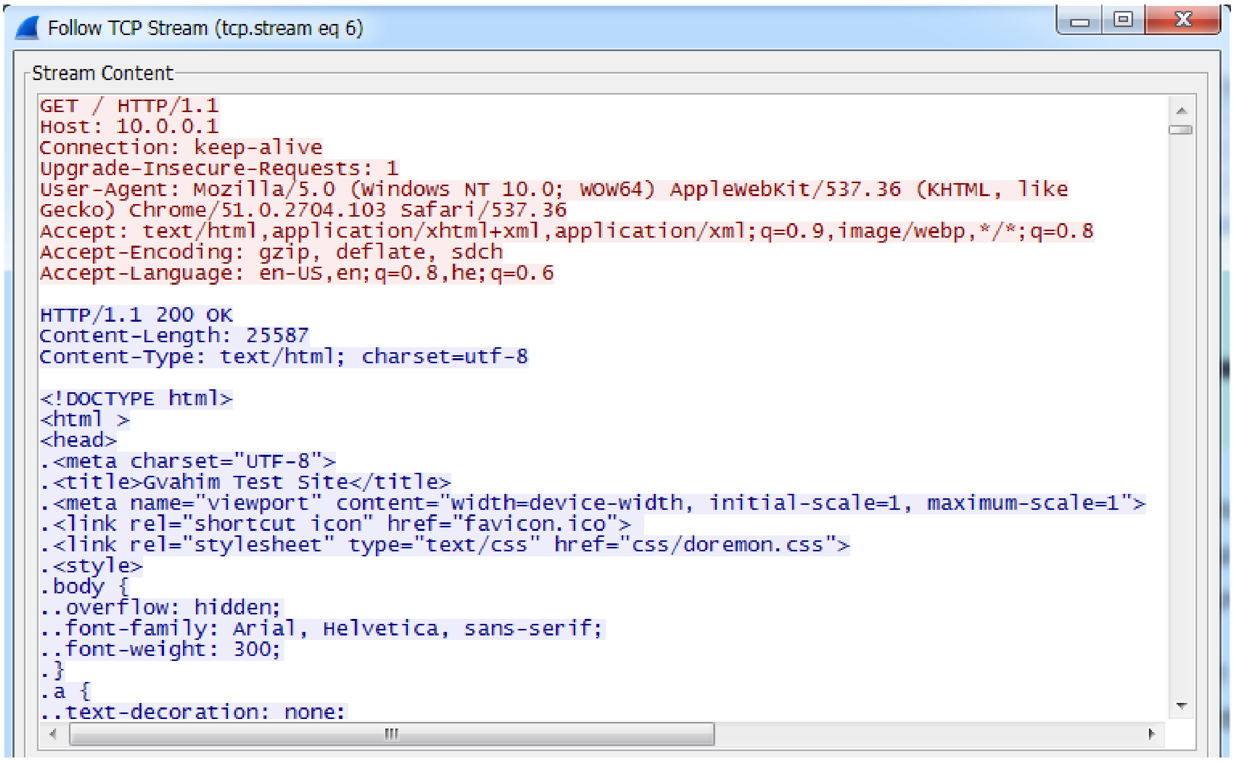
נוכל לראות את כל הבקשות של הלקוח ואת תשובות השרת. בצילום מסך זה ניתן לראות שחלק מהמידע (השורה הצבועה על ידי wireshark בשחור) שודר פעמיים מהשרת ללקוח – Retransmission – שמעיד על כך שהשרת לא קיבל אישור על המידע ששלח ללקוח (האישור הוא ברמת שכבת התעבורה, פרוטוקול TCP, עליו נלמד בהמשך).

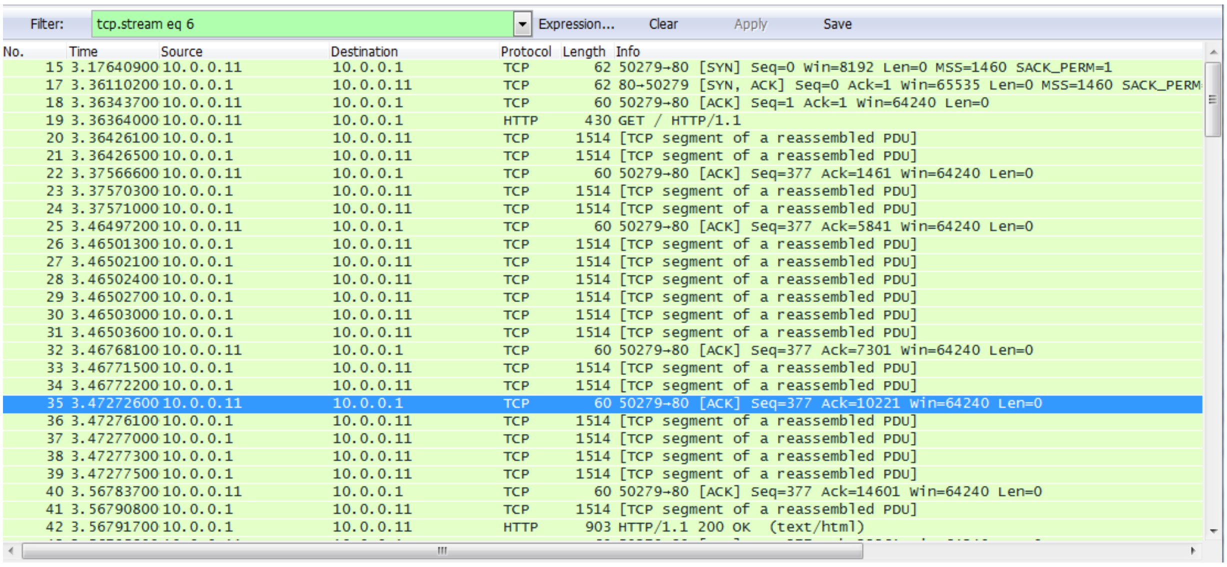
אם נרצה לעקוב אחרי כלל הפקטות שעברו בין השרת והלקוח שלנו, כולל הודעות הקמת וסיום קשר ושליחה מחודשת של פקטות, נוכל להקליק על שורה כלשהי ולבחור follow TCP stream.



יוצגו בפנינו מסכים שמראים את כל המידע שעבר, הן בשכבת האפליקציה (השרת והלקוח בצבעים שונים) והן בשכבת התעבורה.

נוכל להשתמש במידע שעבר ברשת כדי לאתר בעיות שאולי גרמו לכך שהשרת שלנו לא עובד כפי שתכננו.





... זהו, בהצלחה!

פרוטוקול HTTP – תשובה "תכנותית" לבקשות GET, ופרמטרים של בקשת GET

במהלך עשרים השנים האחרונות, "אפליקציות web" (כינוי לאפליקציות ושרתים שעושים שימוש בפרוטוקולי אינטרנט) צברו יותר ויותר פופולריות, ובמקביל התפתחו באופן מואץ היכולות, רמת התחכום והמורכבות של מערכות אלו. אתרי אינטרנט כבר לא הסתפקו בהגשה של דפי html ותמונות ערוכים מראש, אלא עשו שימוש בקוד ותכנות כדי לייצר דפי תוכן "דינמיים".

לצורך המחשה, חשבו מה קורה כשאתם מתחברים ל-Facebook – אתם רואים דף אינטרנט ובו דברים קבועים – סרגל כלים, לינקים – והרבה דברים שנקבעים מחדש בכל כניסה – למשל הודעות על פעילות של החברים שלכם ופרסומות. למעשה מאחורי הקלעים מסתתר שרת, שמקבל החלטות איך להרכיב את דף התוכן עבור כל בקשה של כל משתמש.
כך כאשר אתם פונים ל-Facebook, השרת צריך להחליט כיצד לבנות עבורכם את הדף. לאחר שהוא מבין מי המשתמש שפנה ומה אותו משתמש צריך לראות, הוא בונה עבורו את הדף ומגיש אותו ללקוח.

השירות ש-Facebook מספק מתוחכם בהרבה מעמודי אינטרנט בראשית הדרך, שבהם היה תוכן סטטי בלבד – קבוע מראש, והשרת רק היה מגיש את העמודים הללו ללקוחות.

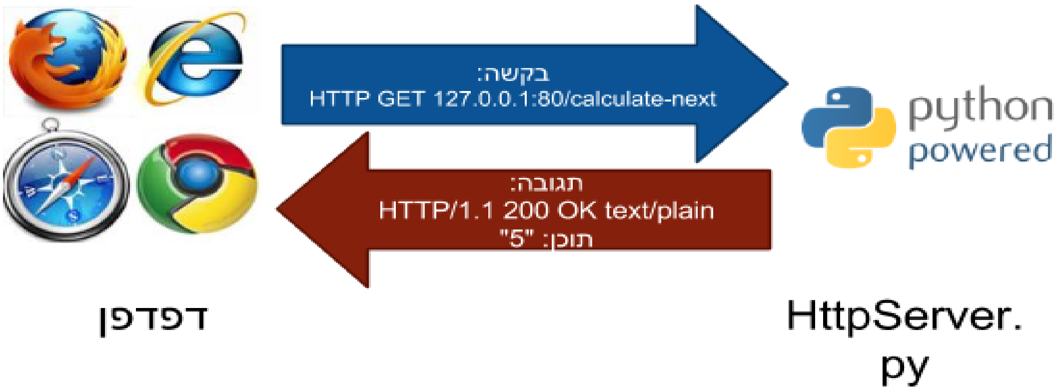
כעת נממש שרת שמגיב בתשובות תכנותיות לבקשות GET. כלומר, השרת שלנו יבנה תשובה באופן דינאמי בהתאם לבקשה שהגיעה מהלקוח. בינתיים, נממש את צד השרת, ולצורך צד הלקוח נמשיך להשתמש בדפדפן.

תרגיל 4.5

עדכנו את קוד השרת שכתבתם, ככה שפנייה לכתובת "/calculate-next" לא תחפש קובץ בשם הזה, אלא תענה על השאלה "מה המספר שמגיע אחרי 4". כלומר, השרת פשוט יחזיר תמיד "5" בתור תוכן התגובה לבקשה הזו.
הריצו את השרת ובדקו שהמספר 5 מתקבל כתגובה לפנייה לכתובת זו.

בתרגילים הבאים, שימו לב לכלול Headerים רלוונטיים בתשובותיכם, ולא "סתם" Headerים שאתם רואים בהסנפות. כדי שהתשובה תהיה תקינה, ציינו קוד תשובה (כגון 200 OK). עליכם לכלול גם Content-Length (שצריך להיות גודל התשובה, בבתים, ללא גודל ה-Headerים של HTTP). ציינו גם את ה-Content-Type (במקרה שתחזירו קובץ – סוג הקובץ שאתם מחזירים, שאפשר להסיק לפי סיומת הקובץ). תוכלו להוסיף גם Headerים נוספים, אך עשו זאת בצורה שתואמת את הבקשה והתשובה הספציפיות.

אם ביצעתם את התרגיל נכון, זה מה שבעצם קרה כאן:



בטח כבר עליתם בעצמכם על כך שהשרת במקרה הזה הוא מאוד לא שימושי, כי הוא מניח שהמשתמש תמיד מעוניין לדעת מה המספר שבא אחרי 4. בתוכניות שכתבנו בפייתון, בדרך כלל עשינו שימוש בקלט מהמשתמש כדי לעשות איתו חישוב כלשהו במהלך התוכנית. השרת שלנו היה הופך להרבה יותר שימושי, אם האפליקציה הייתה יכולה לבקש את המספר שבא אחרי מספר כלשהו; למעשה, אפשר לחשוב על השרת שלנו כמו על כל תוכנית בפייתון שכתבנו עד היום, למעט העובדה שלא ניתן להעביר לתוכנית קלט, ולכן היא מחזירה תמיד את אותו הפלט. אם נוכל להעביר לה בכל בקשה קלט שונה, נקבל את הפלט המתאים לכל קלט.

אם כך, כעת נעביר קלט לשרת באמצעות פרוטוקול ה-HTTP – למעשה נשתמש במה שנקרא " HTTP GET Parameters" – פרמטרים בבקשת ה-GET.

הדרך שבה מעבירים פרמטרים בבקשת GET היא באמצעות תוספת התו "?" בסיום הכתובת (זה התו שמשמש בפרוטוקול HTTP כמפריד בין הכתובת של המשאב לבין משתני הבקשה), ולאחר מכן את שם הפרמטר והערך שלו.

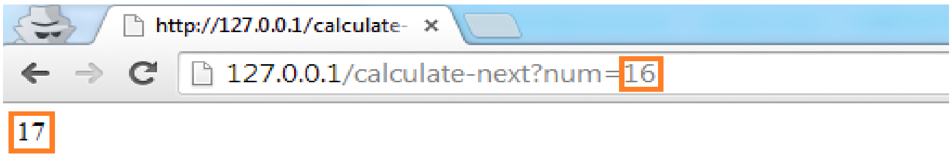
תרגיל 4.6

עדכנו את קוד השרת שכתבתם, כך שפנייה לכתובת "/calculate-next?num=4" תתפרש כפנייה לכתובת calculate-next עם השמה של הערך '4' במשתנה num, ולכן תחזיר את המספר 5.

אבל, "/calculate-next?num=16" תחזיר את התשובה '17',

"/calculate-next?num=10000" תחזיר את התשובה '10001',

והבקשה "/calculate-next?num=-200" תחזיר את התשובה '-199',

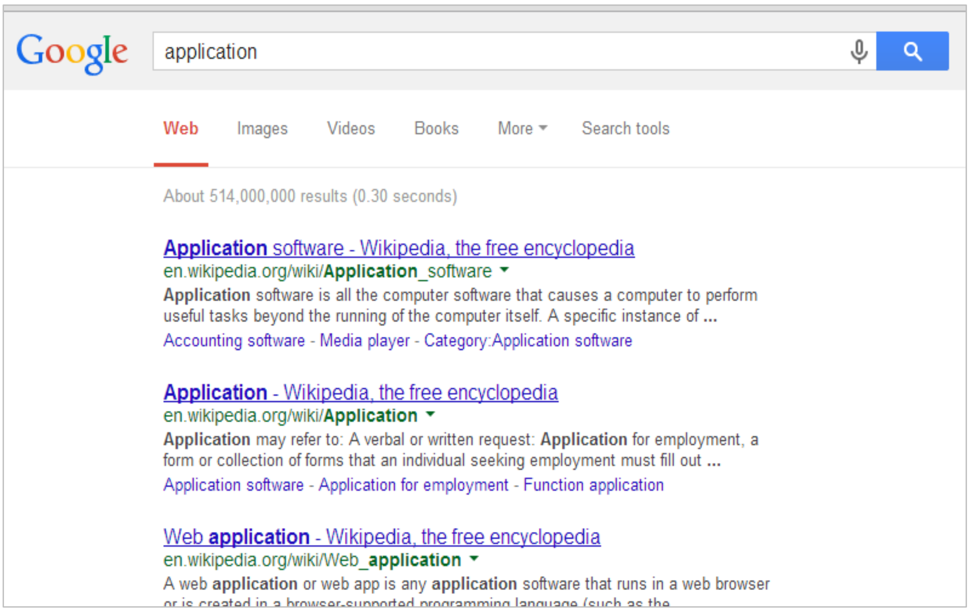


זוהי למעשה הדרך שבה יכולה אפליקציה להעביר פרמטרים כחלק מהבקשה שלה.

כדי להמחיש את העניין הזה, נתבונן בדוגמה מעניינת יותר, ולאחר מכן נחזור להבין את קטע הקוד של השרת שבו השתמשנו:

הכניסו את השורה הבאה בדפדפן: http://www.google.com/search

זוהי למעשה בקשת GET שנשלחת אל שרת החיפוש של Google. אבל, אתם עדיין לא רואים תוצאות חיפוש, כי ה"אפליקציה" (במקרה זה, האתר של Google בדפדפן שלכם), לא יודעת מה אתם רוצים לחפש. ברגע שתתחילו להכניס מילת חיפוש (נניח: "application"), נתחיל לראות תוצאות חיפוש במסך.



איך הגיעו התוצאות? הדפדפן שלח בקשת GET אל השרת של גוגל.

איך נראתה הבקשה? טוב ששאלתם, היא נראתה בדיוק כך[18]:

GET www.google.com/search?q=application HTTP/1.1

נסו בעצמכם להכניס את ה-URL בדפדפן (URL בלבד, החלק הצבוע בכחול, ללא המילה GET וללא גרסת הפרוטוקול), ותראו בעצמכם מה קורה.

[18] ברוב המקרים ה-Host (בדוגמה זו "www.google.com") לא יכלל בשורת ה-GET, אלא ב-HTTP Header נפרד. עם זאת, על מנת שהדוגמאות תהיינה ברורות, אנו נציין אותו באופן מפורש.

תרגיל 4.7

בכל אחת מהשורות בטבלה הבאה, תמצאו כתובת של משאב באינטרנט (אמנם עדיין לא הסברנו את המשמעות הפורמלית של המילה "משאב", אך ראיתם דוגמאות למשאבים, למשל www.google.com/search), שם של פרמטר, ורשימת ערכים.

מה שעליכם לעשות הוא להרכיב בקשת GET בכל אחד מהמקרים (כפי שראינו שבונים בקשה ממשאב, שם של פרמטר וערך שלו), לנסות אותה בדפדפן שלכם, ולכתוב בקצרה מה קיבלתם.

משאב	פרמטר	ערכים	בקשת GET	הסבר
google.com/maps	q	jerusalem, new-york, egypt		
twitter.com/search	q	madonna, obama		
wikipedia.org/w/index.php	search	israel, http, application		
youtube.com/results	search_query	obama, gangam, wolf		

נקודה למחשבה: שימו לב להבדל המרכזי בין ה-URLים הללו לבין URLים שעליהם התבססתם בתרגיל שרת http שמימשתם בסעיף הקודם – כיום "נעלמו" החלקים מה-URL שמתארים מיקום במערכת הקבצים, ואת מקומם החליפו הפרמטרים. זוהי תוצאה של השינוי שהתרחש באינטרנט בעשרים השנים האחרונות – מעבר מהגשה של דפי תוכן שנוצרו מראש, לבנייה של תשובות על סמך פרמטרים וקוד תוכנה שמרכיב את התשובה.

כעת נחזור לאחד החיפושים שעשינו ב-Twitter בתרגיל האחרון; שימו לב שהתוצאות שקיבלנו מכילות ציוצים (tweets) שהכילו את המילה obama. מה אם הייתי רוצה לחפש את הפרופיל של אובמה? (שימו לב שיש בצד שמאל כפתור שעושה את זה). האופן שבו ה"אפליקציה" (במקרה שלנו, האתר) של Twitter עושה את זה, הוא שהוא שולח פרמטר נוסף לשרת.

פרמטר אחד יכיל את מילת החיפוש (obama), והפרמטר השני יכיל את סוג החיפוש – חיפוש משתמשים (users). בין שני הפרמטרים יופיע המפריד &. נראה ביחד איך זה עובד.

הכניסו את השורה הבאה בדפדפן: https://twitter.com/search?q=obama&mode=users

באופן דומה, ניתן לחפש תמונות של אובמה, על ידי החלפת הערך של הפרמטר mode בערך photos – בנו את הבקשה ונסו זאת בעצמכם.

תרגיל 4.8

כמו שאתם וודאי מתארים לעצמכם, ניתן להעביר בבקשת ה-GET גם יותר משני פרמטרים, באותה הצורה. הרכיבו את בקשת ה-GET לשרת המפות של Google (בדומה לדוגמה שעשיתם בתרגיל הקודם), אך במקום חיפוש כתובת, בקשו את הוראות הנסיעה מתל אביב לירושלים בתחבורה ציבורית, על ידי שימוש בפרמטרים הבאים:

- saddr – כתובת המוצא (למשל: 'Tel+Aviv').
- daddr – כתובת היעד.
- dirflg – אמצעי התחבורה. תוכלו להעביר 'r' עבור תחבורה ציבורית (או 'w' אם אתם מתכוונים ללכת ברגל).

בנו את בקשת ה-GET, הכניסו אותה בדפדפן שלכם, וודאו שאתם מקבלים את הוראות הנסיעה.

תרגיל 4.9

כתבו שרת HTTP משלכם, שמחשב את השטח של משולש, על סמך שני פרמטרים: גובה המשולש והרוחב שלו.

למשל, עבור שורת הבקשה הבאה: http://127.0.0.1:80/calculate-area?height=3&width=4, יחזיר "6.0".

בדקו אותו גם עבור קלטים נוספים.

פרוטוקול HTTP – בקשות POST ותכנות צד לקוח בפייתון (הרחבה)

נפתח בשאלה – האם העליתם פעם תמונה ל-Facebook? קל להניח שכן (ואם תתעקשו שמעולם לא עשיתם זאת, ניתן להניח במקום כי שלחתם קובץ PDF או מסמך Word למישהו במייל, או לפחות מילאתם טופס "יצירת קשר" באתר כלשהו). המשותף לכל המקרים שתיארנו הוא – הצורך להעביר כמות מידע **מ**האפליקציה **אל** השרת, כמות של מידע שלא ניתן להעביר בבקשת GET.

נתעכב כדי להבין טוב יותר את ההבדל בין בקשת GET עם פרמטרים לשרת של Twitter, שמחזירה דף אינטרנט עם תמונות של אובמה (כמו שראינו בסעיף הקודם), לבין בקשה לשרת של Facebook, שתאחסן שם תמונה.

במקרה שכבר ראינו – שליחת בקשה לתמונות באתר Twitter – האפליקציה (הדפדפן שמציג את עמוד האינטרנט של Twitter) רוצה להציג למשתמש את התמונות שמתאימות לחיפוש "אובמה":

כדי לעשות זאת, נשלחת לשרת בקשה די קצרה, שמכילה בסך הכל את הכתובת "twitter.com/search", ושני פרמטרים – obama, photos. כל זה נכנס בפקטה אחת. התשובה, לעומת זאת, מכילה הרבה מידע, ואם נסניף את התקשורת, נראה שהתשובה מורכבת ממספר גדול של פקטות. עשו זאת, ומצאו את פקטת הבקשה, וכמה פקטות היה צריך כדי להעביר את כל התשובה.

מה ההבדל לעומת המקרה שבו נעלה תמונה ל-Facebook?

במקרה זה, המידע נמצא בצד האפליקציה (ספציפית – התמונה. בטלפון ממוצע תמונה יכולה להיות בגודל של יותר מ-1MB), והיא מעוניינת להעביר את כולו לשרת.

האם ניתן להשתמש בפרמטרים של בקשת GET כדי להעביר את כל התמונה?

אם ניקח בחשבון את העובדה שהאורך המקסימלי של URL בו תומכים רוב הדפדפנים הוא 2,000 תווים, וה-URL הוא כל מה שניתן להעביר בבקשת GET, די ברור שלא נוכל להעביר את התמונה לשרת באמצעות בקשת GET.

מה לגבי מקרה שבו אנחנו שולחים תגובה (טוקבק) לאתר כלשהו? תגובה שאנחנו שולחים כוללת את השם שלנו, כתובת, כותרת ותוכן התגובה עצמו.

האם היה ניתן לשלוח את כל הנתונים האלה לשרת באמצעות בקשה GET? לדוגמה כך-

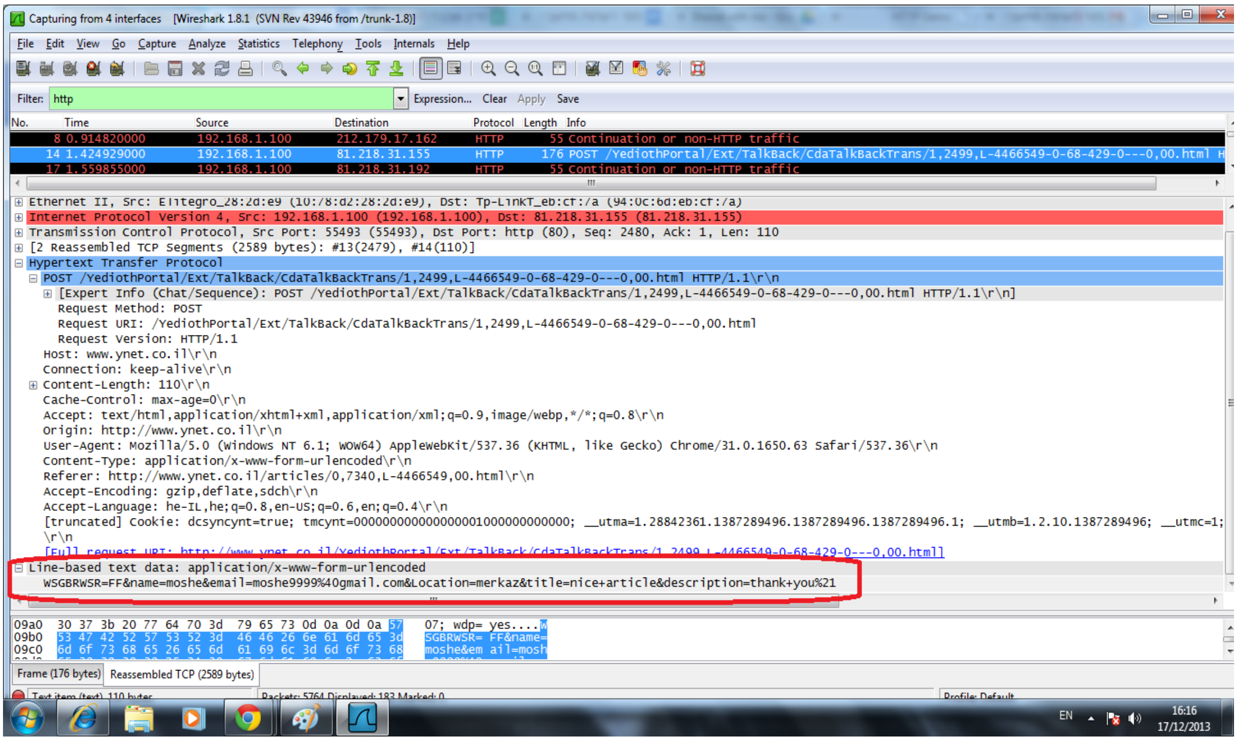
themarker.com/articles/17773?name=Moshe&address=Rehovot&mail=moshe&rehovot.co.il &title=nice_article&body=thank_you_this_was_a_great_article_...

גם אם באופן תיאורטי ניתן לנצל את שדה ה-URL והפרמטרים כדי להעביר מידע, המקום הנכון להעביר בו מידע הוא אזור ה-Entity Body של פרוטוקול ה-HTTP. זה המקום בו היינו רוצים שהמידע שאנחנו מעבירים יימצא. קל יותר לתכנת כך, אין מגבלה של מקום, והדבר מסייע במניעת שגיאות ופרצות בקוד.

לכן, נעשה שימוש בסוג חדש של בקשה שקיים בפרוטוקול HTTP – בקשת POST. בנוסף לפרמטרים שעוברים כחלק מהכתובת (URL), בבקשת POST ניתן לצרף **תוכן**, ושם יועבר תוכן התגובה.

באופן כללי, אחד השימושים הנפוצים בבקשות POST הוא בטפסים – הכוונה היא לטופס שמכיל מספר שדות, שאותם המשתמש ממלא ואז לוחץ על כפתור כדי לשלוח את הטופס – וזהו בדיוק המקרה עם תגובות באתר. דוגמה נוספת – אם תלחצו על "צור קשר" באתר של חברת הסלולרי שלכם, ותמלאו שם את הטופס, התוכן יישלח לשרת של אתר האינטרנט שלהם באופן דומה.

אם תרצו "לראות" היכן עובר התוכן עצמו בבקשת ה-POST – לטפסים יש דרך די סטנדרטית להעביר את הנתונים האלה. הדרך הזו קצת מזכירה את האופן שבו משתמשים בפרמטרים ב-URL, רק שבמקרה זה, אין מגבלה של מקום. כדי לראות זאת – התבוננו ב-Wireshark בתוכן של פקטת ה-POST, וחפשו את text-data. נסו למצוא שם את השדות השונים שמילאתם בתגובה (שמכם, מקום המגורים, הכותרת וכו').



HTTP – סיכום קצר

עד כה, למדנו כיצד להתנהל עם משאבים באמצעות פרוטוקול HTTP. בתחילה, למדנו לבקש משאבים על סמך השם שלהם באמצעות GET; לאחר מכן למדנו לצרף פרמטרים נוספים כדי "לחדד" את הבקשה שאיתה

אנו פונים אל משאב הרשת – לדוגמה, כשפנינו אל שירות המפות עם חיפוש המסלול מת"א לירושלים, הוספנו פרמטר נוסף לבקשה שמסמן שהמסלול צריך להיות בתחבורה ציבורית.
הבנו כי בקשה של משאב לא אמורה לשנות דבר בצד השרת – רק להחזיר תוצאה מסוימת. ניתן לחזור על אותה הבקשה מספר רב של פעמים, והתוצאה אמורה להיות זהה.

לעומת זאת, בהמשך למדנו להעלות מידע מצד האפליקציה אל השרת באמצעות POST. פעולה זו בהחלט גורמת לשינוי במידע שנמצא בשרת, כפי שראינו בשרת אחזור התמונות שכתבנו בסעיף הקודם. אני יכול לבקש תמונה בשם "december-21" ולקבל הודעת שגיאה שאומרת שהמשאב אינו נמצא (404), וביום שלאחר מכן לבצע שוב את אותה הבקשה, אלא שהפעם אקבל תמונה בחזרה.
מה ההסבר לכך? כנראה שבינתיים מישהו ביצע פקודת POST והעלה תמונה בשם הזה.

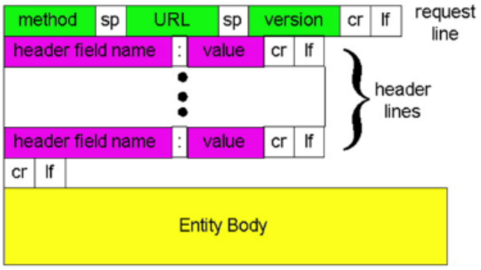
הבנו כי מאחורי משאבי האינטרנט נמצאים שרתים שמריצים קוד – בדומה לשרת שכתבנו בפרק זה – ומשתמשים בו כדי לייצר תגובות לבקשות GET ו-POST. אבל, שרתים כאלה הם לרוב מורכבים הרבה יותר מהשרת שכתבנו, ולרוב ישתמשו גם בבסיס נתונים (database) – יכתבו אליו בבקשות POST, ויקראו ממנו בבקשות GET. הם גם יידעו איך לתמוך במספר משתמשים (לקוחות) שמבקשים בקשות בו זמנית, יפעילו מנגנוני אבטחה והרשאות, ויתמכו בסוגי בקשות נוספים (כמו למשל בקשות למחיקה).

בגלל ששרתי HTTP הם כל כך נפוצים ופופולריים, רוב אתרי האינטרנט לא מממשים את פרוטוקול HTTP בעצמם, אלא עושים שימוש במימושים נפוצים של שרתים כאלה (למשל, בפייתון ניתן להשתמש במימוש SimpleHTTPServer).
נושאים אלו לא מכוסים בפרק זה לעת עתה, אך אנחנו מעודדים קוראים סקרנים לחקור וללמוד בעצמם, בסעיף צעדים להמשך של פרק זה.

HTTP – נושאים מתקדמים ותרגילי מחקר

בחלק זה נלמד על יכולות מתוחכמות יותר של פרוטוקול HTTP, שיהיו מבוססות על מנגנון ה-Header שלמדנו בחלק הבסיסי. בשלב זה של הלימוד, ננצל את היכולות שרכשנו ב--wireshark על מנת לחקור בעצמנו חלק מהנושאים.

בתחילה, ניזכר במבנה המלא של בקשת HTTP – ה-Header מופיע לאחר שורת הבקשה ולפני המידע עצמו (data):




```
GET /index.html HTTP/1.1\r\n
Host: www-net.cs.umass.edu\r\n
User-Agent: Firefox/3.6.10\r\n
Accept: text/html,application/xhtml+xml\r\n
Accept-Language: en-us,en;q=0.5\r\n
Accept-Encoding: gzip,deflate\r\n
Accept-Charset: ISO-8859-1,utf-8;q=0.7\r\n
Keep-Alive: 115\r\n
Connection: keep-alive\r\n
\r\n
```

Cache (מטמון)

דרך אחת "להאיץ" את חוויית השימוש באינטרנט היא באמצעות ההבנה שיש משאבים שאותם מבקשים שוב ושוב, וכך למרות שאין בהם כל שינוי, המידע שלהם עובר מספר רב של פעמים על גבי רשת האינטרנט.

לצורך הדוגמה, חשבו על כך שבכל פעם שאתם גולשים לאתר TheMarker כדי להתעדכן בחדשות, תמונת הלוגו של TheMarker תעבור מהשרת אל הדפדפן שלכם (הלקוח). הדבר נכון גם לגבי שאר הכותרות ואמצעי העיצוב שקבועים בדף – כל זה גורם לתעבורה "מיותרת", שהרי המידע היה כבר בדפדפן שלכם בעבר, ומאז הוא לא השתנה! התעבורה המיותרת גורמת גם לבזבוז של עלויות (רוחב הפס וכמות המידע שעוברת על גביו), וגם של זמן (בהמתנה למשאב שיגיע).

הרעיון של מנגנון ה-**Cache (מטמון)** הוא לשמור משאבים כאלה על המחשב של הלקוח, וכל עוד הם לא משתנים, לטעון אותם מהדיסק המקומי ולא על גבי הרשת.
המנגנון שמאפשר לבצע זאת בפרוטוקול HTTP נקרא Conditional-Get (בקשת GET מותנית). משמעות השם: בקשת ה-GET תתבצע רק **בתנאי** שלא קיים עותק מקומי ועדכני של המשאב.

איך יידע הלקוח האם העותק שקיים אצלו הוא עדכני, או שבשרת כבר יש גרסה חדשה יותר? באמצעות כך שיעביר לשרת (יחד עם בקשת ה-GET) את הזמן בו שמר את המשאב. השרת יחזיר את המשאב רק אם יש לו גרסה חדשה יותר (כלומר, שנוצרה לאחר הזמן שמצוין בבקשה).

לצורך ההמחשה – נחשוב על המקרה בו נגלוש באמצעות דפדפן לאתר TheMarker. הדפדפן יבקש את הלוגו של TheMarker, אבל רק אם הוא השתנה מאז הפעם האחרונה שהדפדפן הציג את האתר ושמר אצלו את הלוגו.

ברוב המוחלט של הפעמים, הלוגו לא השתנה, והשרת פשוט יורה לדפדפן: "תשתמש בעותק שקיים אצלך". כך נחסכה העברה של הלוגו של TheMarker שוב ושוב ברשת.

ננסה להבין את המנגנון הזה באמצעות תרגיל מעשי:

תרגיל 4.12 מודרך – הבנת Caching באמצעות Wireshark

שימו לב שרוב הדפדפנים כיום מבצעים **Caching** – כלומר הם שומרים ב"מטמון" דפים שהם הורידו ולא מורידים את הדף מחדש בכל פעם, אלא רק כשהם חושבים שיש בזה צורך.מה, לא פתחתם הסנפה עדיין? קדימה!

1. חפשו בגוגל "אתרי http" (כיוון שכיום כמעט כל האתרים הם https, שם התעבורה מוצפנת ולכן קשה יותר לחקור את התעבורה באמצעות Wireshark). נכון למועד עדכון ספר זה, נותרו מספר זעום של אתרי http בישראל, כגון אתר רשות העתיקות ואתר מוזיאון הפלמ"ח. אתם מוזמנים להכנס לאתר הפלמ"ח ובזמן שהנכם שם כבדו את לוחמי הפלמ"ח המעטים שעדיין איתנו וקיראו על פועלם ועל מורשתם - http://www.palmach.org.il/
2. חכו עד שהדפדפן יסיים לטעון את העמוד.
3. הזינו פילטר http ב-Wireshark. כמה פקטות HTTP עברו ביניכם לבין השרת?
4. התחילו הסנפה חדשה, כעת היכנסו שוב לעמוד הזה (או לחצו על **Refresh / F5**).
5. הפסיקו את ההסנפה. כמה פקטות HTTP עברו ביניכם לבין השרת?
6. בפקטות ששלח השרת, חפשו בתוך הודעת 200 OK שדה בשם "Last Modified". השדה הנ"ל אומר לדפדפן לבדוק האם יש ברשותו כבר את הגרסה הזו, ואם כן- הדפדפן יכול להשתמש ב-cache שלו ואין צורך לבקש את המידע מהשרת.
7. התחילו הסנפה נוספת. חיזרו לדפדפן ולחצו על CTRL+F5. פעולה זו מאלצת את הדפדפן לבקש את המידע מהשרת, גם אם קיים כבר מידע ב-cache. כמה פקטות עברו כעת ביניכם לבין השרת?

Cookies ("עוגיות")

במהותו, HTTP הוא פרוטוקול "חסר-מצב" (באנגלית: stateless) או "חסר-זיכרון". המשמעות היא שכל בקשה מטופלת בפני עצמה, ללא קשר לבקשות הקודמות – בקשות מסוג GET מאפשרות גישה למשאבי אינטרנט (תמונות, מסמכים וכו'), ובקשות POST מאפשרות להעלות מידע מאפליקציית לקוח אל השרת (טפסים, תמונות, מיילים וכו').

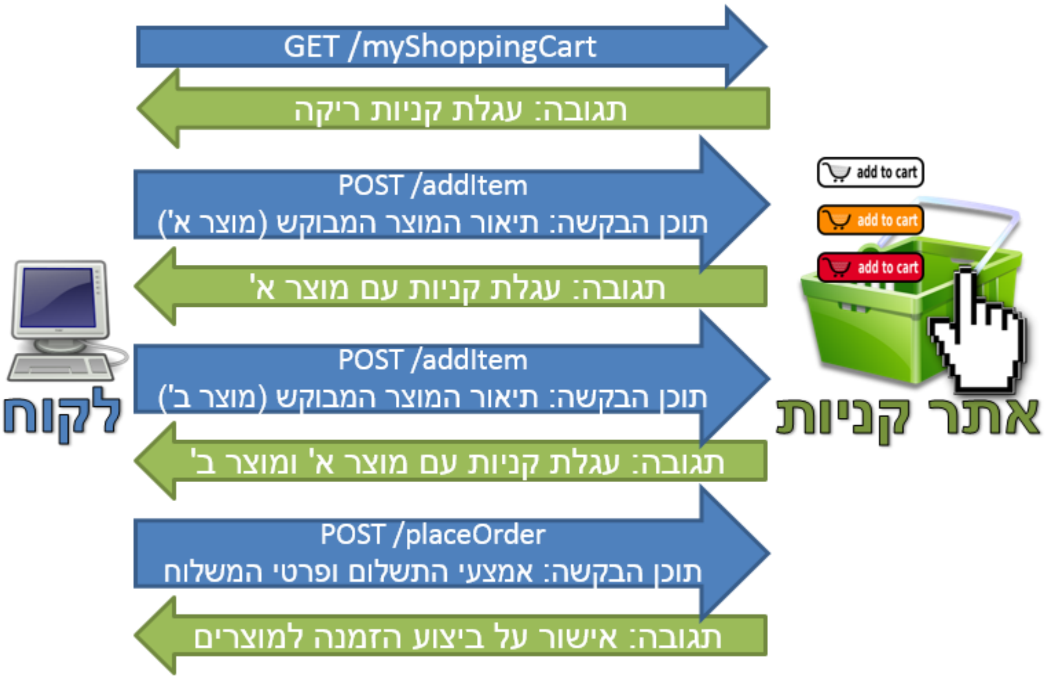
פרוטוקול HTTP הוא משולם עבור אתרים כמו Wikipedia, מכיוון שאם אנחנו נמצאים בערך כלשהו לא משנה לשרת כלל איך הגענו לערך הזה. במילים אחרות, בכל דרך שבה נגיע אל הערך המבוקש, השרת יציג בפנינו את אותו ערך. השרת לא יציג ללקוח גרסאות שונות של אותו הערך כתוצאה מהגעה מלינקים שונים.

עם זאת, בהרבה אפליקציות ואתרי אינטרנט נפוץ הרעיון של session (פעילות ממושכת) – למשל באתר Amazon, ניתן להוסיף עוד ועוד מוצרים לעגלת הקניות, וכן לקבל הצעות למבצעים על סמך המוצרים שבעגלה, ולאחר מכן לבצע תשלום כשרשימת המוצרים שבחרנו לאורך ה-session כבר סוכמה.

בניגוד לשרת של Wikipedia, שרת כמו של Amazon חייב לדעת מה מסלול הגלישה שהוביל את הלקוח לעמוד מסויים. הרי לכל לקוח אמורה להיות עגלת קניות אישית, אי אפשר להציג לכל הלקוחות את אותה רשימת קניות בעגלה.

תרגיל לחשיבה עצמית: כיצד ניתן לממש מנגנון של עגלת קניות? מה נדרש לצורך כך בשרת? מה נדרש באפליקציית הלקוח?

השרת צריך "לזכור" עבור כל session של משתמש אילו מוצרים כבר הוספו לעגלת הקניות ואילו מבצעים הוצעו לו; בכל תשובה לבקשת GET של הלקוח, על השרת לצרף את רשימת המוצרים שכרגע בעגלת הקניות, וכן מבצעים רלוונטיים על סמך המוצרים שכבר בעגלה; בכל בקשת POST (להוספת מוצר לעגלה), על השרת להוסיף את המוצר לרשימת המוצרים שבעגלת הקניות של ה-session.



שימו לב שהמידע אודות ההזמנה מגיע אל השרת בשלבים, כלומר, באמצעות מספר בקשות שונות; ראשית מגיעה הבקשה להוסיף את מוצר א׳ לרכישה, לאחר מכן הבקשה להוסיף את מוצר ב׳, ולבסוף הבקשה לביצוע ההזמנה, המספקת את פרטי אמצעי התשלום וכתובת המשלוח.

תקשורת מהצורה הזו מחייבת את השרת ״לזכור״ מידע בין טיפול בבקשות שונות.

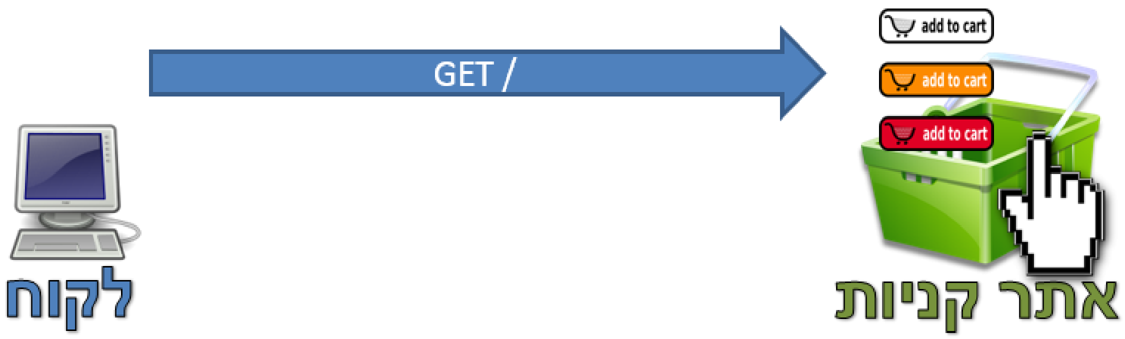
חשוב גם לשים לב שהשרת צריך "לזכור" בנפרד את עגלת הקניות לכל session של כל משתמש; אסור שיבלבל בין עגלות של sessionים שונים.

תרגיל לחשיבה עצמית: כיצד ניתן לעשות זאת? כשמתקבלת בקשת HTTP, כיצד "יידע" השרת לאיזה session היא שייכת?

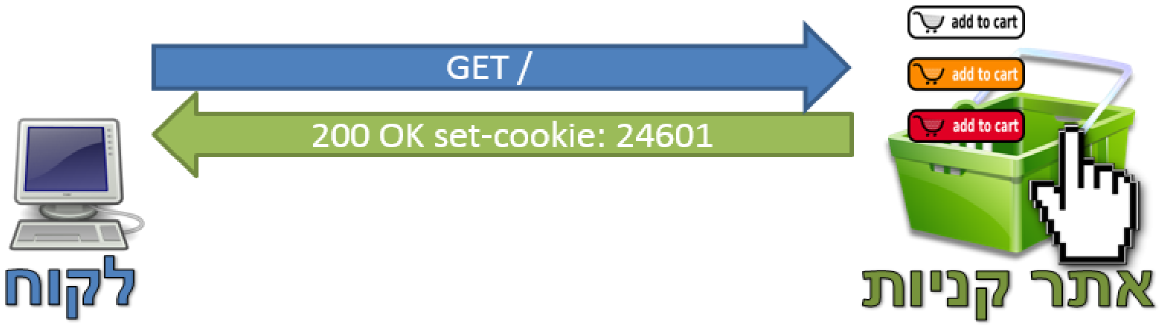
הפתרון לכך הוא שימוש במנגנון ה-cookies. ה-cookie ("עוגייה") היא מחרוזת שמשותפת לשרת וללקוח; השרת קובע את המחרוזת הזו בתחילת session, ולכל אורך ה-session הלקוח יצרף את המחרוזת הזו לכל בקשה שלו (בשני המקרים עושים שימוש ב-HTTP Header fields כדי להעביר את המחרוזת הזו).

את ה-cookie ישמור הלקוח בדיסק המקומי, וכן היא תישמר בבסיס הנתונים של השרת. ל-cookie נקבע אורך חיים, כך שלאחר זמן מסוים פג התוקף שלה, והשרת יקבע מזהה session חדש.

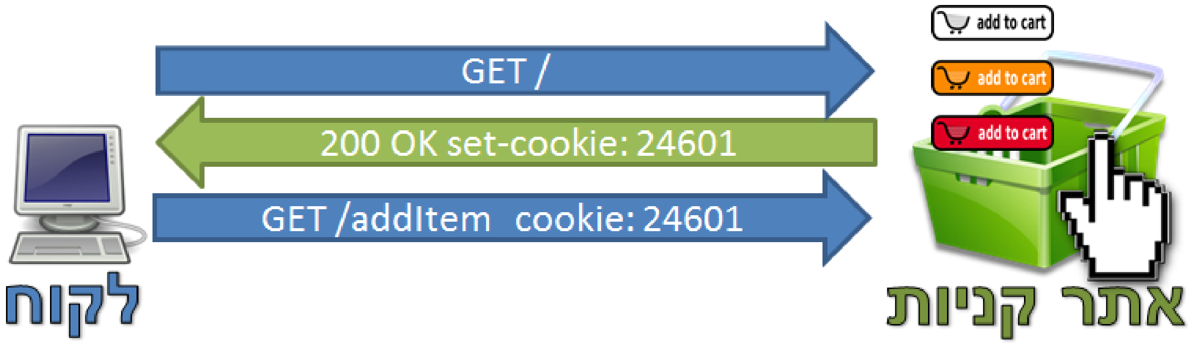
על מנת להבין את האופן בו עובד מנגנון ה-cookies, נשתמש בדוגמה. נאמר שיש לנו לקוח בשם client123 שניגש אל שירות קניות באינטרנט:



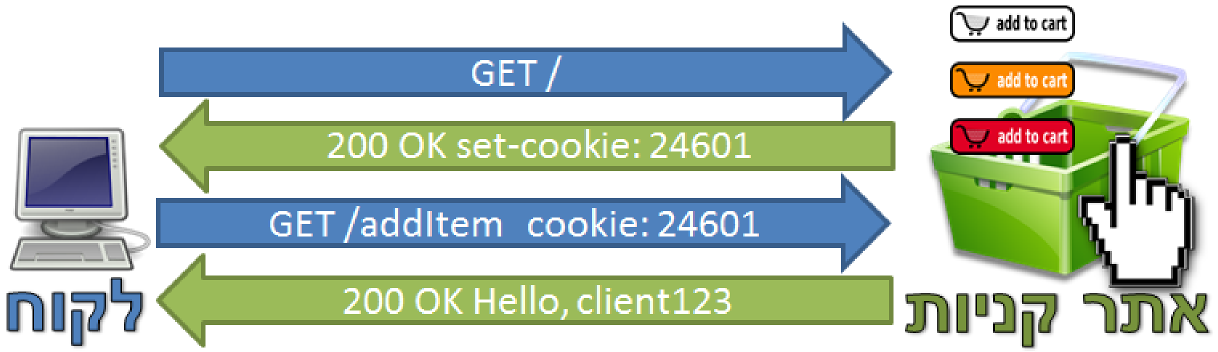
כעת, האתר מעוניין לשמור מזהה של הלקוח אצלו. לכן, הוא מייצר עבורו מזהה. נאמר שהשרת בחר במזהה 24601. כעת, השרת יאמר ללקוח להשתמש מעתה במזהה זה:



מעתה ואילך, כל עוד זמן התוקף של העוגיה לא פג, בכל פנייה שהלקוח יבצע אל אתר הקניות הזה, הוא ישלח גם את המזהה שלו:



באופן זה, בכל פעם שהלקוח יפנה לשרת, השרת יוכל לראות את המזהה ולזהות שמדובר בלקוח client123:


GET /
200 OK set-cookie: 24601
GET /addItem cookie: 24601
200 OK Hello, client123
לקוח
אתר קניות
add to cart
add to cart
add to cart

פרוטוקול DNS – הסבר כללי

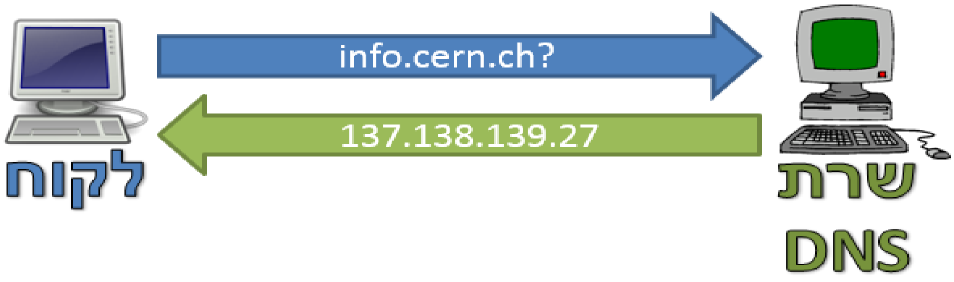
DNS (ראשי תיבות של Domain Name System) הוא פרוטוקול נפוץ נוסף בשכבת האפליקציה. תפקידו העיקרי הינו לתרגם שמות דומיינים (כגון www.google.com, www.facebook.com או http-demo.appspot.com) לבין כתובת ה-IP הרלוונטית. לבני אדם נוח יותר לזכור שמות טקסטואליים כגון "www.google.com" מאשר כתובת IP כגון "173.194.39.19". לשם כך נועד פרוטוקול ה-DNS. למעשה, ניתן לחשוב על DNS כמעין ״ספר טלפונים״ – כמו שספר הטלפונים מתרגם בין שם של אדם או עסק (שאותו יותר קל לבני אדם לזכור) לבין מספר טלפון, כך ה-DNS שמאפשר לאתר כתובת IP באמצעות שם של אתר.

על מנת להבין את הצורך בפרוטוקול זה, ננסה להבין כיצד הדפדפן פועל כאשר מנסים לגלוש לאתר מסוים. ניזכר בדוגמה הראשונה שראינו בתחילת פרק זה, בה הכנסנו את הכתובת http://info.cern.ch/hypertext/WWW/TheProject.html, וראינו את ההסנפה הבאה ב-Wireshark:

528	21.277073000	192.168.1.100	137.138.139.27	HTTP	618 GET /hypertext/WWW/TheProject.html HTTP/1.1
531	21.352697000	137.138.139.27	192.168.1.100	HTTP	1077 HTTP/1.1 200 OK (text/html)

שימו לב שהדומיין בכתובת הזו הינו info.cern.ch, אך אם ניזכר בפרק תכנות ב/ Sockets-כתובות של Socket, נבין שבתקשורת אינטרנט לא ניתן להתחבר אל דומיין, אלא דרושה כתובת IP.
בהסנפה לעיל ניתן לראות שהדפדפן פתר את הבעיה הזו בדרך כלשהי, ופנה אל הכתובת 137.138.139.27. השלב שעליו דילגנו בהסבר שמופיע בתחילת הפרק, הוא שלב התרגום – תרגום שם של דומיין (במקרה זה – info.cern.ch) לכתובת IP (במקרה זה – 137.138.139.27). תרגום זה נעשה באמצעות פרוטוקול DNS.

בדומה לפרוטוקול HTTP, גם פרוטוקול DNS פועל באמצעות בקשה (Request), שנקראת גם **שאילתה (Query)**, **ותשובה (Response)**. בטרם הדפדפן ניגש אל האתר המבוקש (info.cern.ch), על מערכת ההפעלה למצוא את כתובת ה-IP הרלוונטית. לשם כך, המחשב שלנו מתשאל שרת DNS:



כעת, כאשר ללקוח יש את כתובת ה-IP של שרת היעד, הוא יכול לפנות אליו באמצעות פרוטוקול HTTP.
על מנת להבין את דרך הפעולה של שרת DNS כדי להמיר את שם הדומיין לכתובת IP, עלינו להכיר כיצד שמות דומיין מורכבים. לשם כך, עלינו להכיר את היררכיית השמות של DNS.

היררכיית שמות

DNS משתמש במבנה היררכי של **אזורים (Zones)**. התו המפריד שיוצר את ההיררכיה הוא התו נקודה ("."). כך למשל, הדומיין www.facebook.com מתאר שרת בשם "www" בתוך האזור "facebook" שבתוך האזור "com". הדומיין "he.wikipedia.org" מתייחס לשרת בשם "he" בתוך האזור "wikipedia", שבתוך האזור "org".

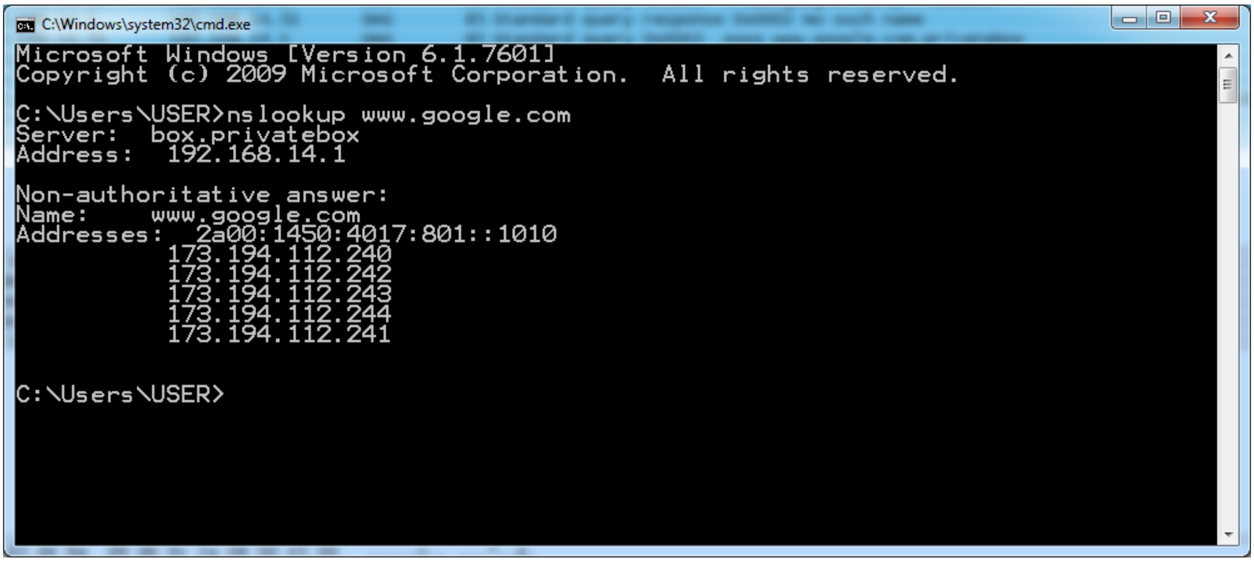
חלוקת ה-DNS לאזורים מאפשרת חלוקת אחריות ומשאבים. במצב זה, אף שרת DNS לא צריך לטפל בכל הדומיינים באינטרנט. לכל אזור יכול להיות שרת DNS שידאג אך ורק לאזורים והדומיינים שנמצאים תחתיו. כך למשל, השרת שאחראי על כל הדומיינים ותת-הדומיינים (subdomains) של google.com כגון: mail.google.com ו-www.google.com, לא צריך להכיר את www.facebook.com, ובוודאי שלא את he.wikipedia.org. שרת זה מכיר רק את הדומיינים ותת הדומיינים שתחת google.com.

האזור הראשי ברשת הינו האזור Root המיוצג בידי התו נקודה ("."). למעשה, האזור com, כמו גם האזור org, מוכלים בתוך האזור Root. כך ששם הדומיין המלא עבור "www.google.com" הינו למעשה "www.google.com." (שימו לב לנקודה שמופיעה בסוף הכתובת).

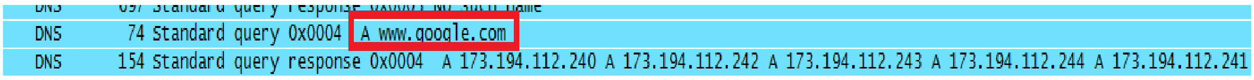
תרגיל 4.14 מודרך – התבוננות בשאילתת DNS

על מנת לראות שאילתת DNS, פתחו את Wireshark והתחילו הסנפה. אתם יכולים להשתמש במסנן התצוגה "dns". כעת, היעזרו בכלי **nslookup** אותו פגשנו לראשונה בפרק תחילת מסע – איך עובד האינטרנט / DNS. הריצו את שורת הפקודה (Command Line), ולאחר מכן, הריצו את הפקודה הבאה:

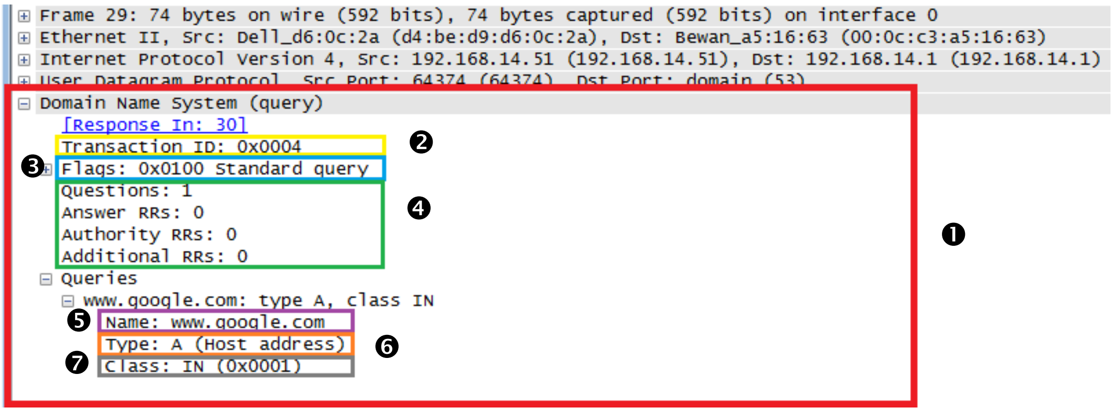
nslookup www.google.com



הערה: ייתכן שתראו יותר מאוסף אחד של שאילתה ותשובה. חפשו רק את החבילה שכוללת את השאלה עבור שם הדומיין "www.google.com":



כעת נתמקד בחבילת השאילתה (Query):



כפי שניתן לראות, פרוטוקול ה-DNS (שהחלק שלו בחבילה מסומן ב**אדום**, 1) נשלח בשכבה החמישית, מעל פרוטוקול UDP בשכבה הרביעית. כעת נתמקד בשדות של פרוטוקול זה.

השדה הראשון, המסומן ב**צהוב** (2), הוא שדה ה-Transaction ID. שדה זה כולל מזהה של השאילתה הנוכחית, על מנת להפריד אותה משאילתות שונות. כך למשל, השאילתה הזו קיבלה את המזהה 4. ייתכן שהשאילתה הבאה תקבל את המזהה 5. דבר זה מקל על המחשב להפריד בין התשובות שיקבל מהשרת, ולדעת איזו מהן שייכת לאיזו שאלה.

השדה השני, המסומן ב**כחול** (3), הינו שדה הדגלים (Flags). שדה זה מורכב משמונה דגלים בעלי משמעויות שונות. בדוגמה לעיל, הדגלים מציינים כי מדובר בשאילתה סטנדרטית. עבור תשובה, למשל, יהיו דגלים שונים.

השדות הבאים, המסומנים ב**ירוק** (4), מתארים כמה רשומות מכילה חבילת ה-DNS. בשאילתות ותשובות של DNS ישנן רשומות, הנקראות **Resource Records** (או בקיצור **RR**, כפי ש-Wireshark מציין). כל רשומה כזו מכילה מספר פרטים, כפי שתכף נראה. בחבילת השאלה שלפנינו, ישנה רשומת שאלה אחת, ואין רשומות נוספות.

לבסוף, אנו רואים את רשומת השאלה. ב**סגול** (5), שדה השם (Name), שכולל את שם הדומיין המלא. בדוגמה זו, השאלה היא עבור השם www.google.com. ב**כתום** (6), אנו רואים את סוג (Type) הרשומה שעליה שואלים. כאן מדובר בסוג A, המתאר רשומה הממפה בין שם דומיין לכתובת IP. ישנן גם סוגי רשומות נוספים, כמו הרשומה PTR העושה בדיוק את הדבר ההפוך – ממפה בין כתובת IP לבין שם הדומיין הרלוונטי. ב**אפור** (7), אנו רואים את סוג הרשת (Class). בכל המקרים שאתם צפויים לראות, הסוג יהיה תמיד IN, ולכן לא נתעכב על שדה זה.

תרגיל 4.15 – תשאול רשומות מסוגים שונים

בתרגיל הקודם, השתמשנו ב-**nslookup** על מנת לתשאל מה כתובת ה-IP של הדומיין www.google.com. כעת, אנו מבינים שלמעשה שלחנו שאילתה עבור רשומה מסוג A על השם www.google.com. ניתן לציין בפני **nslookup** במפורש עבור איזה סוג רשומה לתשאל, בצורה הבאה:

nslookup -type=<TYPE> <host/address>

כך לדוגמה, עבור תשאול רשומה מסוג A, ניתן לכתוב:

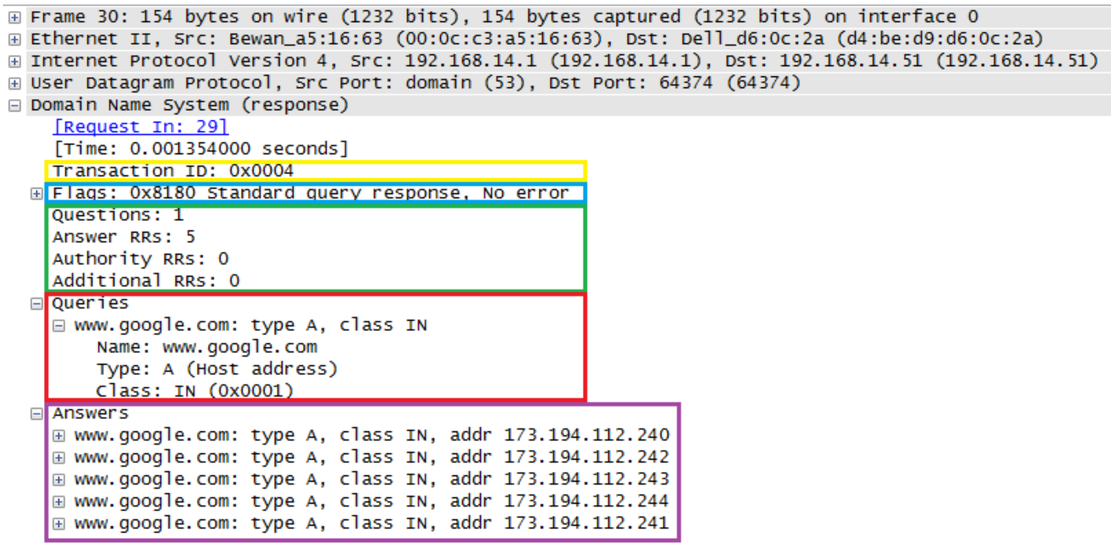
nslookup -type=A www.google.com

כפי שציינו קודם, קיימים גם סוגי רשומות נוספים, כגון PTR – שמתאר רשומה שממפה בין כתובת IP לבין שם הדומיין שלה.

השתמשו ב-nslookup וגלו מהו שם ה-DNS עבור כתובת ה-IP הבאה: 8.8.8.8. מהו שם הדומיין שמצאתם?

תרגיל 4.16 מודרך – התבוננות בתשובת DNS

בתרגיל המודרך הקודם, שלחנו שאילתה מסוג A עבור הדומיין www.google.com, וצפינו בחבילת השאילתה שנשלחה. כעת, נתמקד בחבילת התשובה מאותה ההסנפה:



השדה הראשון, המסומן ב**צהוב** (1), הוא שדה ה-Transaction ID. באופן הגיוני, הערך הינו 4, הזהה לערך עבור השאילתה שראינו קודם. כך המחשב יכול לדעת שהתשובה הזו שייכת לשאילתה שראינו קודם.

השדה השני, המסומן ב**כחול** (2), הינו שדה הדגלים (Flags). ניתן לראות שכעת הדגלים שונים מבמקרה של שאילתה. במקרה זה, הדגלים מעידים על תשובה שחזרה ללא שגיאות.

השדות הבאים, המסומנים ב**ירוק** (3), מתארים כמה רשומות מכילה חבילת ה-DNS. במקרה הזה ניתן לראות שישנה רשומת שאילתה אחת, ועוד חמש שאלות תשובה.

לאחר מכן, ב**אדום** (4), אנו רואים את רשומת השאילתה שראינו קודם לכן. בחבילות תשובה, שרתי ה-DNS משכפלים את השאילתה שנשלחה אליהם ושולחים אותה חזרה אל השולח.

לסיום, ב**סגול** (5), ניתן לראות את חמש רשומות התשובה. אפשר לראות שישנן חמש רשומות שונות, כאשר כל אחת מכילה כתובת IP שונה. הסיבה לכך היא שלעיתים רשומת DNS מצביעה על יותר מכתובת IP אחת. במקרה זה, למשל, ייתכן שאחד השרתים של Google לא יהיה זמין. במקרה זה, מערכת ההפעלה תוכל לפנות אל כתובת IP אחרת, שאולי תקשר לשרת שכן זמין. כאן אנו לומדים על יתרון נוסף של מערכת ה-DNS – היכולת לקשר כתובת שמית אחת ליותר מכתובת IP אחת.

נתמקד באחת מרשומות התשובה:

```
⊟ Answers
  ⊟ www.google.com: type A, class IN, addr 173.194.112.240
      Name: www.google.com
      Type: A (Host address)
      Class: IN (0x0001)
      Time to live: 3 minutes, 30 seconds
      Data length: 4
      Addr: 173.194.112.240 (173.194.112.240)
  ⊞ www.google.com: type A, class IN, addr 173.194.112.242
  ⊞ www.google.com: type A, class IN, addr 173.194.112.243
```

המבנה דומה מאוד למבנה רשומה של שאילתה. אלו הם השדות:

- **בסגול** (1) – שדה השם (Name), שכולל את שם הדומיין המלא. בדוגמה זו, התשובה היא עבור השם www.google.com.
- **בכתום** (2) – שדה סוג (Type) הרשומה. כאמור, מדובר בסוג A.
- **באפור** (3) – סוג הרשת (Class). הערך הוא IN.
- **באדום** (4) – Time To Live. שדה זה קובע כמה זמן יש לשמור את הרשומה ב-Cache של הלקוח. בדומה ל-HTTP, גם עבור שאילתות DNS לא נרצה לשאול שאלות סתם. במקרה זה, על הלקוח לזכור את כתובת ה-IP של Google במשך שלוש וחצי הדקות הקרובות, ורק לאחר מכן – לשאול שוב.
- **בירוק** (5) – אורך (Length) המידע. שדה זה משתנה בהתאם לסוג השאילתה. כאן, הגודל הוא 4 – מכיוון שכתובת IP היא באורך של ארבעה בתים (bytes).
- **בכחול** (6) – המידע עצמו (Data). במקרה של רשומת A, מדובר בכתובת ה-IP הרלוונטית לשם הדומיין עליו נשלחה השאילתה.

כעת, ברשות הלקוח יש את כתובת ה-IP של www.google.com, והוא יכול להשתמש בה בכדי לתקשר עם השרת.

גלישה לאתרים שנמצאים על שרתי אחסון

לאחר שמצאתם באמצעות שאילתת DNS את כתובת ה-IP של www.google.com, הכניסו אותה אל הדפדפן ותגיעו אל האתר המבוקש. גם אם תזינו לדפדפן את כתובת ה-IP של פייסבוק, תגיעו אל האתר המבוקש. ישנם אתרים נוספים בהם כדי לגלוש אל האתר די בהכנסת כתובת ה-IP של השרת. המשותף לאתרים אלו הוא שהשרת שעליהם נמצא האתר שלהם שייך רק להם. כלומר, השרת של גוגל משרת רק את גוגל והשרת של פייסבוק משרת רק את פייסבוק.

אך זה אינו המצב תמיד. למעשה, ברוב האתרים זה אינו המצב. אם תחפשו לדוגמה את כתובת ה-IP של האתר www.themarker.com, של www.twitter.com או של www.amazon.com ולאחר מכן תנסו להזין לדפדפן את כתובות ה-IP שקיבלתם, תקבלו שגיאות. מדוע?

מרבית האתרים בעולם נמצאים על שרתי אחסון. כלומר שרתים שמשמשים אתרים רבים. לכל שרת כזה יש כתובת IP, שהיא כתובת ה-IP שמתקבלת כאשר מבצעים nslookup, אך כיוון שהשרת משרת אתרים רבים, יש צורך למסור לו איזה URL ספציפי אנחנו מבקשים. אחרת, השרת יחזיר לנו שגיאה.

תרגיל 4.17 מודרך – תשאול DNS רקורסיבי

הסרטון הבא שהכנו עבורכם מסביר כיצד משתמשים בכלי nslookup כדי לבצע תשאול רקורסיבי. צפו בסרטון ההדרכה:

https://data.cyber.org.il/networks/DNS.mp4

לאחר שהבנתם כיצד עובד תשאול רקורסיבי של DNS – הגיע הזמן לממש זאת בעצמכם בכדי לגלות מהי כתובת ה-IP של הדומיין maps.google.com. היעזרו בכלי **nslookup** ובתיעוד שקיים אודותיו ברחבי האינטרנט. בצעו את השלבים הבאים:

- בחרו בשרת Root כלשהו. איך מצאתם את כתובת ה-IP שלו? מה היא כתובת ה-IP?
- באמצעות תשאול שרת ה-Root, גלו מי הוא שרת ה-NS האחראי על ה-Zone של com. מי הוא שרת ה-NS? מה היא כתובת ה-IP שלו? מה הרצתם בכדי לגלות זאת?
- באמצעות שרת ה-NS האחראי על ה-Zone של com, גלו מי הוא שרת ה-NS האחראי על ה-Zone של google. מי הוא שרת ה-NS? מה היא כתובת ה-IP שלו? מה הרצתם בכדי לגלות זאת?
- באמצעות שרת ה-NS האחראי על ה-Zone של google.com, גלו מה היא כתובת ה-IP של maps.google.com. מה היא הכתובת? מה הרצתם בכדי לגלות זאת?

תרגיל 4.18 – elgoog (אתגר)

קרדיט: **אייל אבני**

.הבוט הביס הזל שי הארנכ ,אל םאו – החיתפ טפשמב ליחתמ ליגרת לכ

ישנם דברים שעושים בסדר הנהוג, וישנם דברים שנעשים לחלוטין הפוך. אנחנו נתחיל מהסוף, ובתקווה – נצליח לשנות אותו!

בתרגיל זה ניצור שרת DNS, שיגרום לכך שכל פנייה שלנו ל-www.google.co.il תנותב לכתובת אחרת.

בטרם תתחילו לעבוד על התרגיל, שימו לב לקרוא היטב את ההוראות ולוודא שאתם מבינים אותן.

הכנות:

בחלק זה ניצור הסנפה של DNS QUERY ו-DNS RESPONSE עבור www.google.co.il, כדי שנוכל בהמשך לזהות את ה-QUERY המבוקש ולגרום לשרת שלנו להחזיר תשובה תקינה.

ראשית, ודאו שאתם מחוברים דרך כבל הרשת.

כעת, הריצו את שורת הפקודה הבאה (חשוב: פתחו CMD בהרשאות Administrator). השורה תגרום למחיקת ה-cache של ה-DNS שלכם, כיוון שסביר שכתובת ה-IP של גוגל כבר נמצאת שם ולכן לא תבוצע DNS QUERY אותה אנחנו רוצים למצוא:

```
ipconfig /flushdns && taskkill /F /IM iexplore.exe
```

פתחו את Internet Explorer.

פתחו Wireshark והפעילו הסנפה עם ה-display filter הבא:

```
udp.port == 53
```

גלשו ל: www.google.co.il באמצעות Internet Explorer.

עצרו ושמרו את ההסנפה בWireshark.

הורידו את הקובץ

data.cyber.org.il/networks/gvahimchallenge/elgoog.rar

חלצו את הקבצים מתוך ה-rar שהורדתם לתיקיית התרגיל והריצו את הקובץ elgooG_set.bat בהרשאות Administrator.

הסקריפט יגדיר את שרת ה-DNS הראשי שלכם כ-127.0.0.1, ואת המשני בתור השרת הקודם שהיה לכם. השרת המשני נכנס לפעולה במידה ששרת הראשי אינו מחזיר תשובה, וכך תוכלו להמשיך להשתמש באינטרנט.

התרגיל:

עליכם לכתוב שרת שיקבל את בקשות ה-DNS עבור כתובת ה-IP של www.google.co.il, ויחזיר כתובת IP של שרת אחר.

מצאו בהסנפה את בקשת ה-A עבור שם הדומיין www.google.co.il ואת התשובה לה.

* מומלץ להיעזר באופציה של Follow UDP/TCP Stream על הבקשה שמצאתם.

בשלב הבא, צרו שרת UDP שיאזין לבקשות ה-DNS. כיוון שטרם למדנו לכתוב שרת UDP, תוכלו להשתמש בשלד התוכנית הבאה. עליכם לגרום לשרת להאזין על הפורט של UDP ולכתוב את הפונקציה dns_handler. בפרק הבא, תלמדו לעומק על פרוטוקול UDP ותבינו כיצד הקוד פועל.

```
import socket

DNS_SERVER_IP = '0.0.0.0'
DNS_SERVER_PORT = ???
DEFAULT_BUFFER_SIZE = 1024


def dns_udp_server(ip, port):
    """
        Starts a UDP server on a given IP:PORT, and calls
        dns_handler(data, client_address)
        prototyped function on any client request data.
    """
    server_socket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
    server_socket.bind((ip, port))
    print "Server started successfully! Waiting for data.."
    while True:
        try:
            data, addr = server_socket.recvfrom(DEFAULT_BUFFER_SIZE)
            dns_handler(data, addr)
        except Exception, ex:
            print "Client exception! %s" % (str(ex), )


def main():
    """
        Main execution point of the program
    """
    print "Starting UDP server.."
    dns_udp_server(DNS_SERVER_IP, DNS_SERVER_PORT)


if __name__ == '__main__':
    main()
```

בפעם הבאה שהבקשה הספציפית הזו מופיעה, החזירו את הכתובת 212.143.70.40 במקום את כתובות ה-IP שחזרו במקור.

ברגע שאתם חושבים שסיימתם והכל רץ כמו שצריך, גלשו מחדש לכתובת. מה קרה?

הערות וטיפים:

כשסיימתם את העבודה, לפני שהולכים הביתה, עליכם להריץ את הקובץ המצורף: elgooG_revert.bat בהרשאות Administrator.

שימו לב שהשורה של flushdns עשויה להחזיר שגיאה במידה ו-Internet Explorer כבר סגור – זה בסדר גמור וצפוי.

מומלץ להריץ את כל התרגיל בתור Administrator, גם כשבודקים את הסקריפט שלכם דרך cmd, וגם כאשר שעובדים עם PyCharm.

לפני שאתם עונים לבקשה הספציפית שאתם רוצים לענות לה עם התשובה שלכם – מומלץ לראות שאתם מקבלים מידע כמו שצריך.

חשבו טוב לפני שאתם רצים לכתוב – איך אתם רוצים שהקוד ייראה, ומה בסופו של דבר התוכנה שלכם אמורה לעשות.

בכל פעם שתרצו לבדוק את התוכנה שלכם, מומלץ שתבצעו את ההכנות מחדש עד לשלב פתיחת הדפדפן.

בכדי להכניס מידע בינארי למחרוזת בפייתון, עליכם להוסיף ¥x לפני כל בית – ואז את ערכו ההקסאדצימלי.

לדוגמה, במידה ונרצה להכניס את הערכים 0x13 0x37, נכתוב בפייתון:

```
my_binary_string = '\x13\x37'
```

שימו לב שפרוטוקול DNS הוא לא פרוטוקול טקסטואלי. היעזרו בפונקציה hex בפייתון. לדוגמה –

```
hex(213) = 0xD5
```

בתרגיל אין צורך לבצע פעולות על שכבת התעבורה, שכן פרוטוקול DNS עובר בשכבת האפליקציה. לכן, הדבר היחידי שצריך לבצע בקשר לשכבת התעבורה הוא פתיחת UDP socket.

אתגרים נוספים:

1. גרמו לבקשה הקודמת להפנות ל-127.0.0.1, פתחו את שרת ה-HTTP מהתרגילים הקודמים שלכם. מה קרה?
2. גרמו לתוכנה שלכם להפנות ל-IP לפי כתובת מוגדרת מראש – בצורה גנרית. כך שלדוגמה גלישה ל-nana10.co.il תפנה לכתובת ה-IP של גוגל, וגלישה ל-google.co.il תפנה אותנו אל nana10.co.il. נוסף, נוכל ללא מאמץ רב להוסיף כתובות נוספות בעתיד.

מחקר פרוטוקול – SMTP

עד כה למדנו על שני פרוטוקולים נפוצים בשכבת האפליקציה. כעת יש בידיכם את הכלים להבין בעצמכם איך פועלים פרוטוקולים בשכבת האפליקציה. כדי לתרגל זאת נבצע מחקר של פרוטוקול שנקרא SMTP. כל מה שעליכם לדעת בשלב זה, הוא ש-SMTP הוא פרוטוקול שהיה נפוץ מאוד בעבר ושימש לשליחת דואר אלקטרוני. את יתר הפרטים אודות הפרוטוקול תלמדו בעצמכם –

בתור התחלה, **תאבחנו** את הפרוטוקול, כלומר תנתחו בעצמכם, מתוך דוגמה של שימוש בפרוטוקול, את הפקודות, השלבים והפרמטרים שלו. לאחר מכן, **תממשו** לקוח SMTP בפייתון שיישלח בעצמו דואר אלקטרוני.

הורידו את קבצי ההסנפה smpt1.pcap מהכתובת
http://data.cyber.org.il/networks/c04/smtp1.pcap

ואת smtp2.pcap מהכתובת http://data.cyber.org.il/networks/c04/smtp2.pcap

קבצים אלו מכילים הסנפות של מחשב מסוים. בכל הסנפה יש משלוח מייל באמצעות פרוטוקול SMTP. מטרתכם היא להבין את פרוטוקול SMTP. כלומר – להבין את מבנה הפרוטוקול, הדרך שבה הוא עובד ואיך יש לרשום בו הודעות על מנת להצליח לשלוח אימיילים באמצעותו. תעשו זאת באמצעות שימוש ב-Wireshark, ניסוי וטעייה, וללא שימוש במקורות חיצוניים. פעולה זו, של הבנת דרך הפעולה מהתבוננות בהסנפות, נקראת "אבחון" והיא שימושית במקרים בהם או צריכים להבין איך עובדת מערכת מבלי לנו תיעוד מלא שלה.

התרכזו ושימו דגש בצד הלקוח של הפרוטוקול (הצד **ששולח** את המייל), אין צורך להיכנס לפרטים עבור צד השרת.

הנחיה חשובה: אין להשתמש או להיעזר באינטרנט עבור תרגיל זה!

הנחיות נוספות:

- ראשית, הבינו מה היא כתובת ה-IP של המחשב שממנו מתבצעת ההסנפה.
- סננו את הפקטות כך שתראו רק פקטות של פרוטוקול SMTP.

שימו לב: לאחר הסינון ייתכן שתראו גם פקטות מסוג פרוטוקול IMF. התעלמו מהן במהלך התרגיל.

- היעזרו ב-Follow TCP Stream, אותו הכרתם בפרק Wireshark ומודל חמש השכבות/Follow TCP/UDP Stream, כדי לראות את מהלך הפרוטוקול בצורה נוחה.
- נסו להבין – מה הן הפקודות שבפרוטוקול? מה הן הבקשות והתגובות? אילו פרמטרים יש לכל פקודה?

- לאורך התרגיל, זכרו את מטרת פרוטוקול SMTP – לשלוח מיילים. עצרו וחשבו כיצד אתם שולחים דואר אלקטרוני – איזה מידע אתם נדרשים לספק לשם כך? איזה מידע משתנה בין מייל למייל ואיזה מידע לא משתנה? חפשו זאת בקבצי ההסנפה.
- נסו להבין את התמונה הכוללת לפני שאתם צוללים לפרטים. למשל, לו הייתם נדרשים לאבחן את פרוטוקול HTTP, חשוב קודם להבין שהמבנה הבסיסי הוא בקשה-תגובה, וכי יש סוגי בקשות שונים כמו GET ו-POST, הרבה לפני שצוללים לסוגי ה-Headers השונים.
- במהלך התרגיל, תצטרכו להשתמש בקידוד Base64. את הקידוד הזה פגשנו באותנטיקציה של HTTP, ואז השתמשנו באתר אינטרנט כדי להמיר אל הקידוד ובחזרה. הפעם נשתמש בפייתון.
על מנת לקודד מחרוזת לקידוד Base64 באמצעות פייתון, נשתמש בפעולה **encode**:

```
>>> my_string = ‘This is a string...’
>>> my_string.enode(‘base64’)
‘VGhpcyBpcyBhIHN0cmluZW==\n’
```

שימו לב כי פייתון מוסיף למחרוזת את '\n' (התו של ירידת שורה) שאינו חלק מקידוד Base64, ולכן עליכם להסירו. סימני השווה (=) הם כן חלק מקידוד Base64.

על מנת לבצע את הפעולה ההפוכה, נשתמש בפעולה **decode**:

```
>>> ‘VGhpcyBpcyBhIHN0cmluZW==\’.decode(‘base64’)
‘This is a string...’
```

שכבת האפליקציה – סיכום

בפרק זה סקרנו לעומק את שכבת האפליקציה. לאחר שהבנו את המטרות של שכבה זו, התמקדנו בפרוטוקול HTTP. הבנו את מבנה הבקשה והתגובה של הפרוטוקול, וכן הכרנו את ה-Headers השונים ואילו ערכים יש בהם. לאורך הפרק כתבנו שרת HTTP שהלך והתפתח. בתחילה הגשנו קבצים בתגובה לבקשות GET. לאחר מכן, תמכנו בסוגים שונים של תשובות, ובהמשך סיפקנו תשובה תכנותית לבקשות, בהתאם לנתונים שהלקוח העביר בפרמטרים של בקשת ה-GET.

כשהסתכלנו על המשמעות של פרמטרים שונים בבקשות GET, ראינו שירותים אמיתיים כגון Google, Twitter, Wikipedia או YouTube. הצלחנו לבנות בעצמנו בקשות עם פרמטרים שהשפיעו על השרת, כגון בקשה לקבלת הוראות נסיעה מכתובת אחת לכתובת אחרת ב-Google Maps. לאחר מכן, למדנו גם על בקשות POST ותכנתנו תוכנת צד לקוח. לבסוף הכרנו גם נושאים מתקדמים ב-HTTP, כגון מטמון (Cache), עוגיות (Cookies) ואותנטיקציה (Authentication).

בהמשך הפרק, הכרנו גם את פרוטוקול DNS והבנו שהוא משמש בעיקר לתרגום בין שמות דומיין לכתובות IP. ראינו כיצד בנויה שאילתת DNS, וכן כיצד בנויה תשובה. למדנו להשתמש בכלי **nslookup** בכדי לתשאל רשומות מסוגים שונים, הכרנו את מושגי ה-Zone וה-Resource Records.

לסיום, הבנו את פרוטוקול SMTP באמצעות תרגיל. בתחילה אבחנו את הפרוטוקול, והבנו מתוך הסנפות את דרך הפעולה שלו. לאחר מכן, הצלחנו לכתוב בפייתון לקוח ששולח מייל באמצעות פרוטוקול SMTP. כך, יחד עם HTTP ו-DNS, הכרנו שלושה מימושים שונים בשכבת האפליקציה.

בפרקים הבאים, נמשיך להתקדם במודל השכבות ולסקור שכבות נוספות – החל משכבת התעבורה, ועד לשכבה הפיזית. לפני שנוכל לעשות זאת, נלמד כלי חשוב נוסף שיעזור לנו לתרגל את החומר שנלמד בשכבות הנמוכות יותר – Scapy.

שכבת האפליקציה – צעדים להמשך

על אף שהעמקנו רבות בידע שלנו בשכבת האפליקציה, נותרו נושאים רבים בהם לא נגענו. שכבה זו מתאפיינת במספר רב מאוד של מימושים ופרוטוקולים אותם לא הכרנו כלל. בנוסף, ישנם נושאים רבים ב-HTTP וב-DNS בהם לא העמקנו את הידע שלנו.

אלו מכם שמעוניינים להעמיק את הידע שלהם בשכבת האפליקציה, מוזמנים לבצע את הצעדים הבאים:

קריאה נוספת

נושאים מתקדמים בפרוטוקול HTTP

- היכרות עם מימושים נפוצים של שרת HTTP – לדוגמה SimpleHTTPServer, בכתובת: http://goo.gl/z2DtzF.
- מנגנוני אותנטיקציה – ניתן לקרוא בדף: http://en.wikipedia.org/wiki/Authentication.
  - באופן ספציפי, על המנגנון Basic Authentication של HTTP, ניתן לקרוא עוד בדף: http://en.wikipedia.org/wiki/Basic_access_authentication.
  - על המנגנון SSL/TLS, ניתן לקרוא בדף:
    http://en.wikipedia.org/wiki/Transport_Layer_Security
    (עדיף לעשות זאת לאחר לימוד פרק שכבת התעבורה).

נושאים מתקדמים בפרוטוקול DNS

על מנת להבין את הדרך שבה באמת מתבצעת שאילתה, מומלץ לקרוא באחד מהמקורות הבאים:

- פרק 2.5.2 (Overview of how DNS works) בספר Computer Networking: A Top-Down Approach (מהדורה שישית) מאת James F. Kurose.
- פרק 2.2.6 (DNS Queries) בספר Pro DNS and BIND, בכתובת:
http://www.zytrax.com/books/dns

פרוטוקולים נוספים בשכבת האפליקציה

בשכבת האפליקציה יש אינספור פרוטוקולים, והמספר רק הולך וגדל. עם זאת, מומלץ לקרוא ולהכיר פרוטוקולים מסוגים שונים, כגון:

- קבלת דואר – פרוטוקול POP. ניתן לקרוא כאן:
http://en.wikipedia.org/wiki/Post_Office_Protocol
- העברת קבצים – פרוטוקול FTP. ניתן לקרוא כאן:
http://en.wikipedia.org/wiki/File_Transfer_Protocol

פרק 5

Scapy

מבוא ל-Scapy

בפרק תכנות ל- Sockets למדנו כיצד ניתן להשתמש בפייתון ובתיקייה **sockets** בכדי להצליח לפתח בעצמנו אפליקציות. בעוד Sockets היו כלי מצוין בכדי לכתוב קוד לשכבת האפליקציה, הם לא עוזרים לנו לבצע פעולות בשכבות נמוכות יותר, עליהן נלמד בפרקים הבאים. לשם כך – נכיר את Scapy.
Scapy היא תיקייה חיצונית ל-python שמאפשרת שימוש נוח בממשקי הרשת, הסנפה, שליחה של חבילות, ייצור חבילות וניתוח השדות של החבילות. עם זאת, שימו לב ש-Scapy רצה "מעל" python, ולכן כל הפונקציות המוכרות לנו – כגון print או dir, עדיין יעבדו ונוכל להשתמש בהן.

מה למשל אפשר לעשות עם Scapy?

באופן כללי – כל מה שעולה בדמיונכם שניתן לעשות ברשת.

באמצעות Scapy, נוכל להסניף ברשת ולבצע פעולות על החבילות שנקבל. על אף ש-Wireshark הינו כלי משמעותי בעבודה שלנו עם רשתות, הוא לא מאפשר לנו דרך לבצע פעולות מורכבות. למשל, מה יקרה אם נרצה להסניף תעבורת HTTP, כפי שעשינו בפרק שכבת האפליקציה/ פרוטוקול HTTP – בקשה ותגובה, ולשמור לקובץ את כל הכתובות אליהן התבצעה גלישה? מה אם נרצה לשמור רק את הכתובות שאליהן התבצעה גלישה והכתובות עונות על תנאי שהגדרנו מראש? מה נעשה אם נרצה לראות את כל התמונות שעברו באותה ההסנפה? מה אם נרצה לשלוח בעצמנו חבילות, כשאנו שולטים בדיוק במבנה שלהן?

פעולות אלו ועוד נוכל לבצע באמצעות Scapy, ותממשו אותן בעצמכם עד סוף פרק זה. במהלך הפרק נלך יחד, צעד אחר צעד, ונבצע לראשונה חלק קטן אך מכובד מסט הפעולות ש-Scapy מציע לנו. בהמשך הספר, נשתמש ב-Scapy על מנת לבדוק מרחוק איזה תוכנות רצות על מחשב מרוחק, נכתוב בעצמנו כלי שדומה ל-Ping שפגשנו קודם לכן, נגלה מה הדרך אותה עוברת חבילת מידע בין שתי נקודות קצה ועוד.

פרק זה נכתב כמדריך אינטראקטיבי, ובמהלכו נתקדם בהדרגה בעבודה עם Scapy. על מנת ללמוד ממנו בדרך היעילה ביותר, **פתחו את Scapy לצידכם והקישו את הפקודות יחד עם הפרק**. ודאו שאתם מצליחים לעקוב אחר הצעדים ומבינים את משמעותם.

שימו לב כי במהלך הפרק ניגע במושגים שעדיין לא הסברנו לעומק – כגון Ethernet, IP ו-TCP. אל דאגה, את הידע על מושגים אלו נעמיק בהמשך הספר. בינתיים, ניעזר בהם בכדי להבין את השימוש ב-Scapy.

בתום פרק זה תכירו את אחד הכלים המשמעותיים ביותר לעבודה עם רשתות בכלל, ובספר זה בפרט. Scapy ילווה אותנו בהמשך הספר כולו.

בואו נתחיל – הרצת Scapy

בהנחה והתקנתם את Scapy בהצלחה, פתחו את ה-Command Line ולאחר מכן הריצו בשורת הפקודה את הפקודה הבאה:

scapy

כעת המסך אמור להיראות כך:

```
C:\WINDOWS\system32\cmd.exe - scapy
C:\Users\barak>scapy
INFO: Can't import matplotlib. Won't be able to plot.
INFO: Can't import PyX. Won't be able to use psdump() or pdfdump().
WARNING: No route found for IPv6 destination :: (no default route?)
INFO: Can't import python-cryptography v1.7+. Disabled WEP decryption/encryption. (Dot11)
INFO: Can't import python-cryptography v1.7+. Disabled IPsec encryption/authentication.
WARNING: IPython not available. Using standard Python shell instead.
AutoCompletion, History are disabled.
WARNING: On Windows, colors are also disabled
      .SYPACCCSASYY
P /SCS/CCS        ACS | Welcome to Scapy
       /A          AC | Version git-archive.devd31378c886
     A/PS       /SPPS |
        YP        (SC | https://github.com/secdev/scapy
       SPS/A.      SC |
   Y/PACC          PP | Have fun!
    PY*AYC        CAA
         YYCY//SCYP
>>>
```

אל תדאגו לגבי כל ההערות ש-Scapy היקר מדפיס למסך כשהוא עולה. מדובר בחבילות שעדיין לא התקנתם כי הן לא נחוצות לכם, או למשל חוסר תמיכה ב-IPv6, אשר לא רלוונטית לתרגילים הנמצאים בספר[19].

---

[19] עוד על IPv6 – בנספח ג' של פרק שכבת הרשת.

אם אתם מעוניינים במסך צבעוני (גם מגניב וגם נוח יותר לעבוד), לפני הרצת scapy הריצו:

pip install ipython

הפעם לאחר שתקלידו scapy יופיע מסך הפתיחה הצבעוני:

```
Scapy vgit-archive.devd31378c886
C:\Users\barak>scapy
INFO: Can't import matplotlib. Won't be able to plot.
INFO: Can't import PyX. Won't be able to use psdump() or pdfdump().
WARNING: No route found for IPv6 destination :: (no default route?)
INFO: Can't import python-cryptography v1.7+. Disabled WEP decryption/encryption. (Dot11)
INFO: Can't import python-cryptography v1.7+. Disabled IPsec encryption/authentication.

                     aSPY//YASa
             apyyyyCY//////////YCa       |
            sY//////YSpcs  scpCY//Pp     | Welcome to Scapy
 ayp ayyyyyyySCP//Pp           syY//C    | Version git-archive.devd31378c886
 AYAsAYYYYYYYY///Ps              cY//S   |
         pCCCCY//p          cSSps y//Y   | https://github.com/secdev/scapy
         SPPPP///a          pP///AC//Y   |
              A//A            cyP////C   | Have fun!
              p///Ac            sC///a   |
              P////YCpc           A//A   | Craft packets like I craft my beer.
       scccccp///pSP///p          p//Y   |               -- Jean De Clerck
      sY/////////y  caa           S//P   |
       cayCyayP//Ya              pY/Ya
        sY/PsY////YCc          aC//Yp
         sc  sccaCY//PCypaapyCP//YSs
                  spCPY//////YPSps
                       ccaacs
                                       using IPython 7.18.1
```

אפשר לשנות ערכת צבעים בקלות:

conf.color_theme = DefaultTheme()

conf.color_theme = RastaTheme()

conf.color_theme = BrightTheme()

conf.color_theme = ColorOnBlackTheme()

conf.color_theme = BlackAndWhite()

נהדר, עכשיו כש-Scapy פועל אפשר להתחיל לעשות דברים מגניבים.

קבלה של פקטות (או – הסנפה)

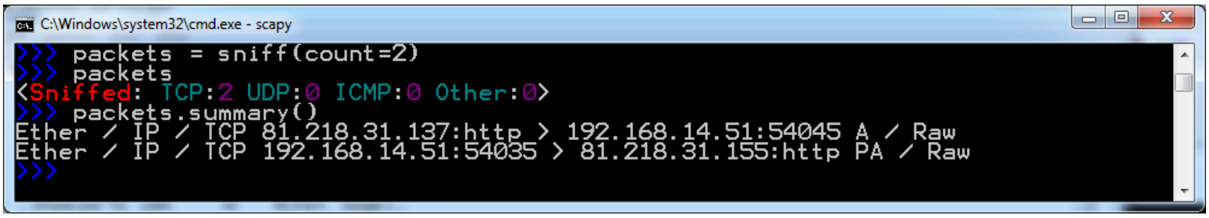
ממש כשם שאנו מסניפים באמצעות Wireshark, נוכל לבצע הסנפה ב-Scapy, וכך לטפל בחבילות שהגיעו אלינו באופן תוכנתי. לשם כך, Scapy מספק לנו את הפונקציה **sniff** (מלשון הסנפה). נתחיל בלהסניף שתי פקטות כלשהן, ונבקש מ-Scapy להציג לנו אותן:

>>> packets = sniff(count=2)

>>> packets

<Sniffed: TCP:2 UDP:0 ICMP:0 Other:0>

Scapy הציג לנו סיכום של הפקטות שהוא קיבל: שתי פקטות מסוג TCP. ניתן לבקש ממנו להציג לנו פירוט גדול יותר באמצעות המתודה **summary**:

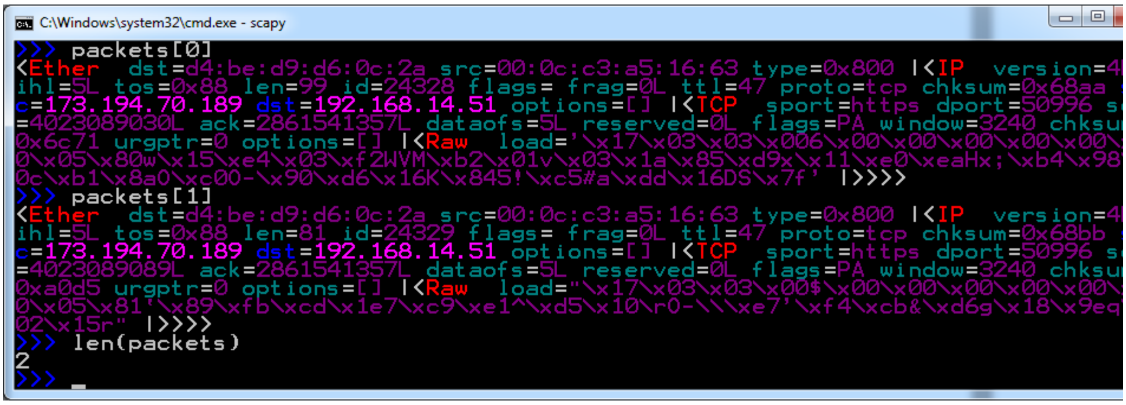


ניתן גם להתייחס לאובייקט packets בתור רשימה:

>>> packets[0]

>>> packets[1]

>>> len(packets)



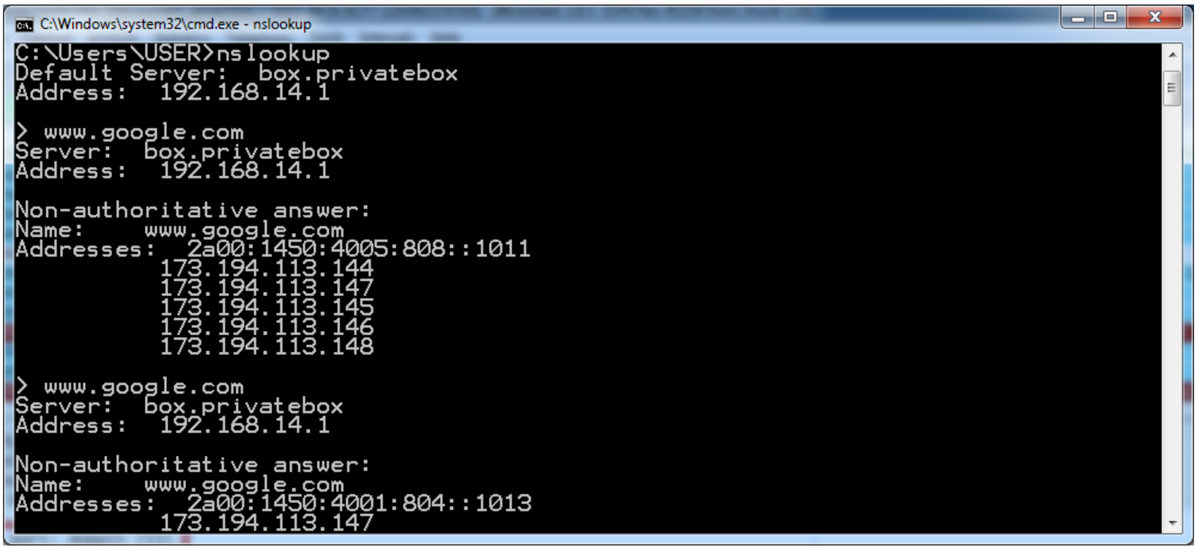
תרגיל 5.1 מודרך – הסנפה של DNS

כעת כשלמדנו להסניף בצורה בסיסית, הגיע הזמן להשתמש בידע שרכשנו בכדי לעשות דברים מועילים. **מטרתנו עכשיו היא להסניף באמצעות Scapy, ולהדפיס למסך את כל שאילתות ה-DNS שהמשתמש שולח.**

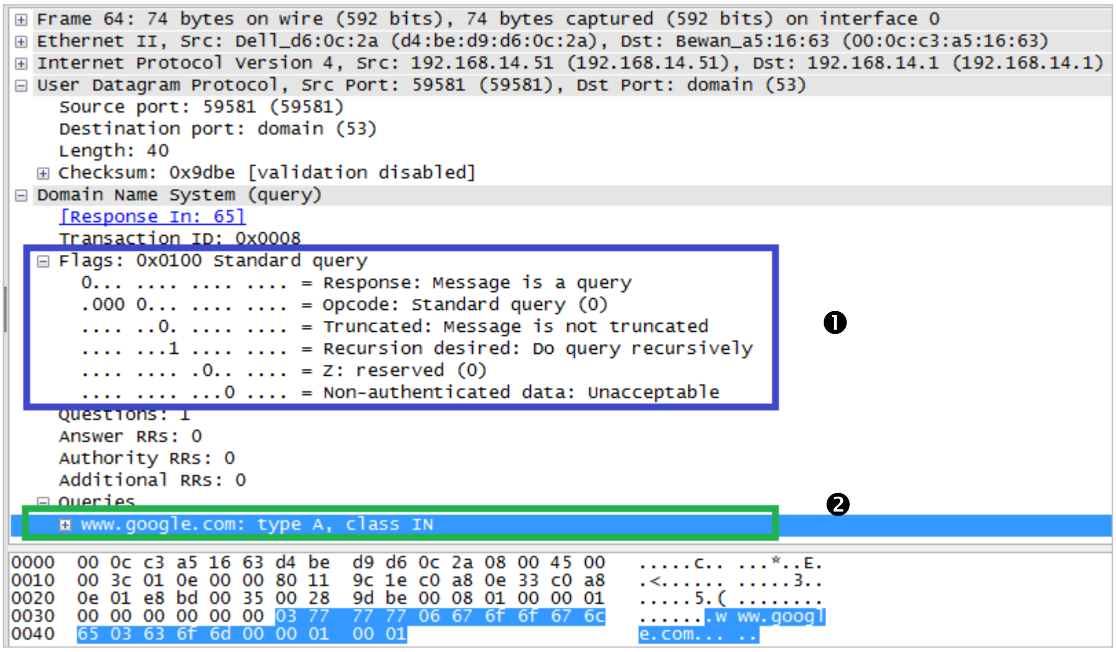
בפרק שכבת האפליקציה, למדנו מעט על פרוטוקול DNS, אשר משמש בעיקר למיפוי בין שמות (כגון www.google.com) לכתובות IP, וכן על דרך העבודה של פרוטוקול זה. כעת נבנה מערכת ניטור גלישה באינטרנט, שתרוץ כל הזמן ברקע ותשמור את כל הדומיינים בבקשות ה-DNS. פעולה זו הייתה קשה לביצוע באמצעות Wireshark, שכן היינו צריכים להסניף לאורך זמן, לבצע סינון, לחפש כל חבילה של DNS, למצוא את כתובת הדומיין מתוך השאילתה וכו'. באמצעות Scapy, נוכל לעשות כל זאת בצורה תכנותית!

בתור התחלה, ננסה ליצור הסנפה ב-Scapy שתציג לנו אך ורק חבילות DNS. ראשית, הריצו את Wireshark, שילווה אותנו במהלך העבודה על מנת להיזכר כיצד פרוטוקול DNS נראה.

פתחו את ה-Command Line, והשתמשו בכלי **nslookup**, כפי שלמדתם ב[פרק תחילת מסע – איך עובד האינטרנט? / DNS](), בכדי למצוא את הכתובת של www.google.com:



על מנת לסנן על חבילות DNS, ניתן להשתמש במסנן התצוגה "dns". מצאו את החבילה הרלוונטית ב-Wireshark. היא אמורה להיראות כך:



נשים לב למספר דברים:

- **בכחול** (1) – שדה הדגלים של ה-DNS, שמראה על שאילתה.

- **בירוק** (2) – השאילתה עצמה, מסוג A (מיפוי שם לכתובת), על הדומיין www.google.com.

כעת נלמד כיצד להשתמש במסנן (filter) בסיסי במהלך ההסנפה. ממש כמו שהשתמשנו במסננים בכדי לסנן חבילות לא רלוונטיות כאשר השתמשנו ב-Wireshark, נרצה לעשות זאת גם כשאנו משתמשים ב-Scapy. נבצע זאת בעזרת הפרמטר **lfilter** של הפונקציה **sniff**.

בשלב ראשון, עלינו להגדיר את הפונקציה שבה נרצה להשתמש כדי לסנן את החבילות. הפונקציה תקבל בכל פעם חבילה אחת, ותחליט האם היא צריכה לעבור את הסינון או לא. בדוגמה זו, נבדוק האם החבילה מכילה שכבת DNS.

להלן הקוד הרלוונטי:

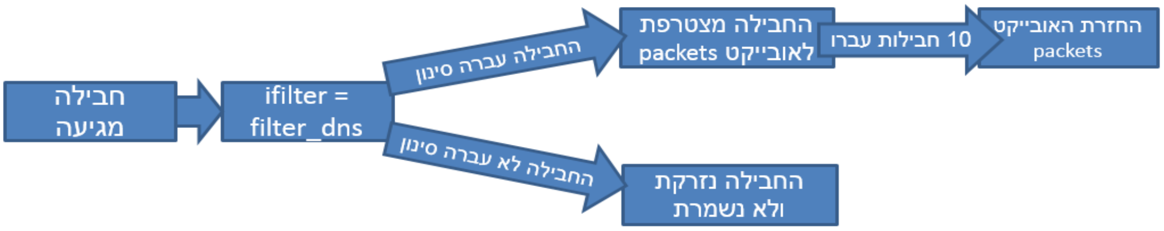
```
>>> def filter_dns(packet):
...     return DNS in packet
```

כלומר, הפונקציה תחזיר True אם קיימת שכבת DNS כחלק מהחבילה, ו-False אם לא קיימת שכבה כזו.

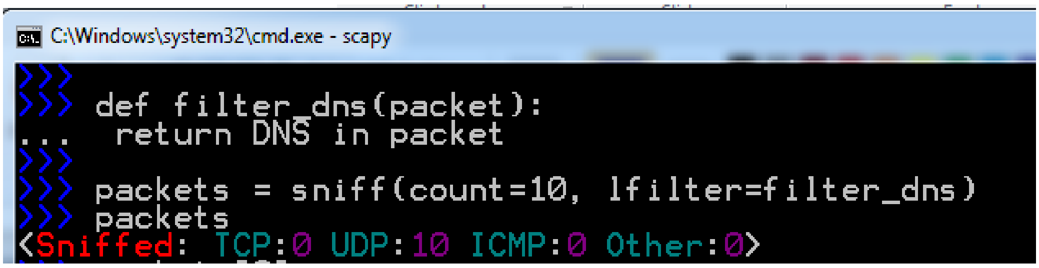
כעת, נשתמש בפונקציה הזו כפרמטר ל-**sniff**, בצורה הבאה:

```
>>> packets = sniff(count=10, lfilter=filter_dns)
```

כלומר, בשלב זה, כל חבילה שתתקבל תישלח אל הפונקציה filter_dns. החבילה היא אובייקט של Scapy. אם החבילה תעבור את הסינון (filter_dns תחזיר True) – החבילה תישמר, ותוחזר לבסוף אל האובייקט packets כאשר יהיו 10 חבילות שעברו את הסינון. שימו לב שהפונקציה sniff היא blocking, כלומר – הפונקציה לא תסיים לרוץ עד אשר יימצאו 10 חבילות שיעברו את הסינון. אם החבילה לא תעבור את הסינון (filter_dns תחזיר False) – החבילה תיזרק:



הריצו שוב את nslookup כפי שעשיתם קודם. כשהפונקציה **sniff** תסיים לרוץ, צפויות להיות בידינו 10 חבילות DNS:



כעת נשמור את אחת החבילות שהסנפנו, ונוכל להביט ולראות איך Scapy מסתכל על חבילה. לשם כך נשתמש במתודה **show()**:

```
>>> my_packet = packets[4]
>>>
>>> my_packet.show()
###[ Ethernet ]###
  dst= 00:0c:c3:a5:16:63
  src= d4:be:d9:d6:0c:2a
  type= 0x800
###[ IP ]###
     version= 4L
     ihl= 5L
     tos= 0x0
     len= 60
     id= 3144
     flags=
     frag= 0L
     ttl= 128
     proto= udp
     chksum= 0x90e4
     src= 192.168.14.51
     dst= 192.168.14.1
     \options\
###[ UDP ]###
        sport= 58636
        dport= domain
        len= 40
        chksum= 0x9dbe
###[ DNS ]###
           id= 12
           qr= 0L
           opcode= QUERY
           aa= 0L
           tc= 0L
           rd= 1L
           ra= 0L
           z= 0L
           rcode= ok
           qdcount= 1
           ancount= 0
           nscount= 0
           arcount= 0
           \qd\
            |###[ DNS Question Record ]###
            |  qname= 'www.google.com.'
            |  qtype= A
            |  qclass= IN
           an= None
           ns= None
           ar= None
>>>
```

ראו איזה יופי! Scapy מראה לנו את כל הפרטים על החבילה, במבט שדומה מאוד ל-Wireshark. אנו רואים את ההפרדה לשכבות (שכבת הקו – Ethernet, שכבת הרשת – IP, שכבת התעבורה – UDP, שכבת האפליקציה – DNS), ואת השדות השונים בכל שכבה.

כעת, נתמקד בשכבת ה-DNS של החבילה. ניתן לעשות זאת באמצעות גישה לשכבת ה-DNS בלבד, בצורה הבאה:

```
>>> my_packet[DNS]
```

```
>>> my_packet[DNS]
<DNS  id=12 qr=0L opcode=QUERY aa=0L tc=0L rd=1L ra=0L z=0L rcode=ok qdcount=1 a
ncount=0 nscount=0 arcount=0 qd=<DNSQR  qname='www.google.com.' qtype=A qclass=I
N |> an=None ns=None ar=None |>
>>> _
```

מכיוון שלא נוח להסתכל על חבילה (או חלק מחבילה) בצורה הזו, נוכל להשתמש שוב במתודה **show()**:

```
>>> my_packet[DNS].show()
###[ DNS ]###
  id= 12
  qr= 0L
  opcode= QUERY
  aa= 0L
  tc= 0L
  rd= 1L
  ra= 0L
  z= 0L
  rcode= ok
  qdcount= 1
  ancount= 0
  nscount= 0
  arcount= 0
  \qd\
   |###[ DNS Question Record ]###
   |  qname= 'www.google.com.'
   |  qtype= A
   |  qclass= IN
  an= None
  ns= None
  ar= None
>>> _
```

אנו רואים כאן כל שדה ושדה של שכבת ה-DNS. נוכל גם לגשת לשדה מסוים. למשל, אם נרצה לבדוק את שדה ה-Opcode, ולדעת האם מדובר בשאילתת DNS או בתשובת DNS, נוכל לעשות זאת כך:

```
>>> my_packet[DNS].opcode
0L
>>> _
```

השדה שווה ל-0. ניתן לראות שלמרות שכאשר ביצענו **show()** נעזרנו ב-Scapy שביצע לנו את ה"תרגום" ואמר לנו ש-0 משמעותו QUERY (שאילתה), בדומה לדרך שבה Wireshark "מסביר" לנו את המשמעות של שדות שונים, כשאנו ניגשים לערך בעצמנו אנו מקבלים את הערך המספרי. אם נחזור ל-Wireshark, נוכל לראות שאכן 0 משמעותו שאילתה.

כעת, נוכל לשפר את פונקציית ה-filter שכתבנו קודם לכן, ולסנן על שאילתות DNS בלבד:

```
>>> def filter_dns(packet):
...     return (DNS in packet and packet[DNS].opcode == 0)
```

נביט שוב בחבילת ה-DNS שלנו. למעשה, ניתן לראות ששכבת ה-DNS מחולקת מבחינת Scapy לשני חלקים: החלק הכללי של ה-DNS, והחלק שכולל את השאילתה עצמה (ומופיע בתור **DNS Question Record**). אל החלק השני ניתן לגשת ישירות בצורה הבאה:

```
>>> my_packet[DNSQR]
<DNSQR  qname='www.google.com.' qtype=A qclass=IN |>
>>> my_packet[DNSQR].show()
###[ DNS Question Record ]###
  qname= 'www.google.com.'
  qtype= A
  qclass= IN
>>>
```

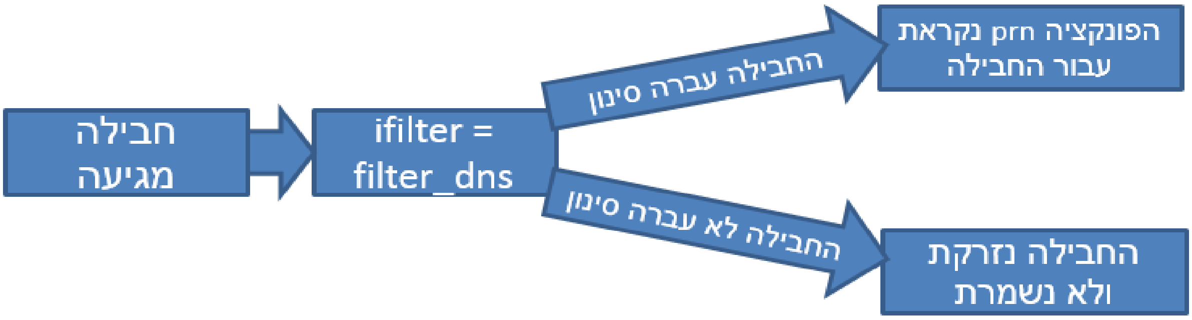
כעת, נוכל גם לגשת לשם שעליו התבצעה השאילתה:

כמו כן, נרצה שהסינון שלנו יתבצע רק על שאילתות מסוג A. נבדוק מה הערך שנמצא בשדה qtype ומשמעותו שאילתת A:

עתה נוכל להשתמש במזהה זה בכדי לשפר את פונקציית ה-filter שלנו:

```
>>> def filter_dns(packet):
...     return (DNS in packet and packet[DNS].opcode == 0 and packet[DNSQR].qtype == 1)
```

עכשיו אנו מסננים אך ורק על שאילתות DNS מסוג A. בשלב הבא, נרצה להדפיס למסך את השם שעליו התבצעה השאילתה. לשם כך, נשתמש בפרמטר של הפונקציה **sniff** שנקרא **prn**. פרמטר זה מקבל פונקציה שמבצעת פעולה על כל פקטה שעוברת את ה-filter. כלומר, כל חבילה שהצליחה לעבור את הסינון שהתבצע קודם לכן באמצעות **lfilter** תישלח אל הפונקציה שניתנה ל-**prn**. מכאן ששרשרת הפעולות שלנו תיראה כך:



ראשית, עלינו להגדיר את הפונקציה שתרוץ. ברצוננו להדפיס את שם הדומיין שעליו התבצעה השאלה. לשם כך, נגדיר את הפונקציה בצורה הבאה:

```
>>> def print_query_name(dns_packet):
...         print(dns_packet[DNSQR].qname)
```

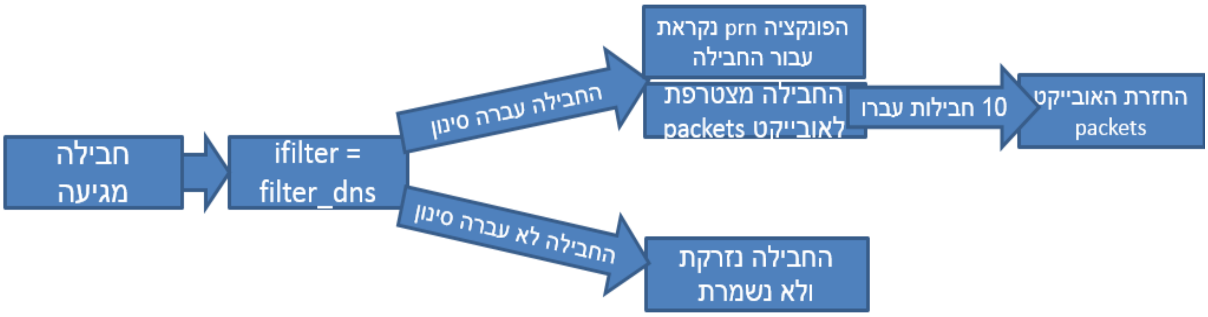
לפני ביצוע ההסנפה, נאפס את המטמון של רשומות ה-DNS באמצעות הפקודה[20] **ipconfig /flushdns**:

```
C:\Windows\system32\cmd.exe
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation.  All rights reserved.

C:\Users\USER>ipconfig /flushdns

Windows IP Configuration

Successfully flushed the DNS Resolver Cache.

C:\Users\USER>
```

כעת, נוכל לקרוא לפונקציה **sniff**, כאשר אנו מסננים על שאילתות DNS מסוג A בלבד באמצעות **lfilter**, ומדפיסים את שמות הדומיינים שעליהם התבצעה השאילתה באמצעות **prn**:

>>> sniff(count=10, lfilter=filter_dns, prn=print_query_name)

זוהי שרשרת הפעולות המלאה:



נסו לגלוש בדפדפן ולראות אילו תוצאות אתם מקבלים. אתם צפויים לראות פלט הדומה לפלט הבא:

```
C:\Windows\system32\cmd.exe - scapy
>>> def print_query_name(dns_packet):
...   print dns_packet[DNSQR].qname
>>>
>>> def filter_dns(packet):
...   return (DNS in packet and packet[DNS].opcode == 0 and packet[DNSQR].qtype =
= 1)
>>>
>>> sniff(count=10, lfilter=filter_dns, prn=print_query_name)
www.google.co.il.
www.google.co.il.
www.google.com.
www.google.com.
apis.google.com.
lh4.googleusercontent.com.
plus.google.com.
apis.google.com.
lh4.googleusercontent.com.
plus.google.com.
<Sniffed: TCP:0 UDP:10 ICMP:0 Other:0>
>>> _
```

[20] באם לא נבצע פעולה זו, מערכת ההפעלה עשויה "לזכור" את התשובות לשאילתות קודמות ששאלנו, ולא לשאול עליהן. כך למשל, אם נגלוש ל-www.google.com, מערכת ההפעלה עשויה "לזכור" את כתובת ה-IP שנמצאה עבורו קודם לכן, ולתת אותה כתשובה מבלי באמת לשאול את שרת ה-DNS.

מכיוון שהפונקציה שניתנת ל-**prn** היא קוד פייתון לכל דבר, נוכל לבצע כל פעולה שנרצה. נוכל, למשל, לשמור את הדומיינים הללו לקובץ. שימו לב כמה קל לעשות זאת באמצעות Scapy.

תרגיל 5.2 מודרך – הספנה של HTTP

ב<u>פרק שכבת האפליקציה</u> למדנו על פרוטוקול HTTP. לצערנו, Scapy לא מכיר את פרוטוקול HTTP באופן מובנה, והוא מתייחס אליו פשוט כאל מידע (כלומר – מחרוזת של תווים). באם נסניף חבילת HTTP, היא תיראה כך:

```
C:\Windows\system32\cmd.exe - scapy
>>> my_packet.show()
###[ Ethernet ]###
  dst= 00:0c:c3:a5:16:63
  src= d4:be:d9:d6:0c:2a
  type= 0x800
###[ IP ]###
     version= 4L
     ihl= 5L
     tos= 0x0
     len= 1400
     id= 20600
     flags= DF
     frag= 0L
     ttl= 128
     proto= tcp
     chksum= 0x6499
     src= 192.168.14.51
     dst= 81.218.31.185
     \options\
###[ TCP ]###
        sport= 50744
        dport= http
        seq= 4059555257L
        ack= 1636415122
        dataofs= 5L
        reserved= 0L
        flags= A
        window= 16660
        chksum= 0x45d9
        urgptr= 0
        options= []
###[ Raw ]###
           load= 'GET /home/0,7340,L-8,00.html HTTP/1.1\r\nHost: www.ynet.co.il\
r\nConnection: keep-alive\r\nCache-Control: no-cache\r\nPragma: no-cache\r\nAcce
pt: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8\r
\nUser-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, lik
e Gecko) Chrome/32.0.1700.102 Safari/537.36\r\nAccept-Encoding: gzip,deflate,sdc
h\r\nAccept-Language: en-US,en;q=0.8,he;q=0.6\r\nCookie: dcsyncynt=true; _dyid=7
82863874; _dyfs=true; _cb_ls=1; _chartbeat2=uxytmbr82xjw5gdk.1391106319990.13911
06319990.1; _chartbeat_uuniq=1; _dycnoabc=1391106320458; tmcynt=0000000000000000
000000000000020; _dyaud_page=47@1294169256@0@3.50@520249338@0@4.95@1392490978@0@2
9.100@-688318138@0@33.101@1972278733@0@34.102@-264413004@0@35.103@412490352@0@36
.104@-990684182@0@37.105@1315227445@0@38.106@-971182034@0@39.107@-663019518@0@40
.108@-1598492479@0@41.211@653122546@0@195.212@-1374571664@0@196.213@1928760588@0
@197.214@1589254879@0@198-199.215@584638648@0@200.216@896448668@0@201.217@-12576
78123@0@202.313@-994981885@0@360-361.326@-1708915932@0@375-384.327@-1695460534@0
@377.328@-1106950800@0@378.342@279212711@0@396-397-398.343@921663746@0@399-400-4
01.344@-1299386719@0@402-403-404.345@2093435358@0@405-406-407.346@-140053136@0@4
08-409-410.347@-927614613@0@411-412-413.368@-1308686476@0@449.369@-2012292496@0@
450.370@-119191347@0@451.371@-118395'
>>> _
```

למעשה, כל שכבת ה-HTTP שלנו הובנה על ידי Scapy בתור מידע Raw, והוא מוצג כרצף מידע ותו לא.

ברצוננו להשיג את כל בקשות ה-HTTP שפונות לעמוד מסוים, כלומר בקשות מסוג GET.

כיצד נוכל לסנן חבילות לא מזוהות?

מכיוון ש-Scapy לא מכיר את HTTP, לא נוכל לכתוב "HTTP in packet" כמו שביצענו קודם. עם זאת, במקרה של HTTP, אנו יודעים כי חבילות GET יתחילו ברצף התווים 'GET'. נוכל, למשל, להניח שכל חבילה ששכבת התעבורה שלה היא TCP[21] והמידע שלה מתחיל במחרוזת 'GET' הינה אכן חבילת GET.

כדי לבדוק האם המידע מתחיל ב-'GET', עלינו ראשית להפוך אותו למחרוזת:

```
>>> data_string = str(my_packet[Raw])
```

כעת נוכל לבדוק האם המידע מתחיל במחרוזת הידועה מראש:

```
>>> data_string.startswith('GET')
```

```
>>> str(my_packet[Raw])
'GET /home/0,7340,L-8,00.html HTTP/1.1\r\nHost: www.ynet.co.il\r\nConnection: ke
ep-alive\r\nCache-Control: no-cache\r\nPragma: no-cache\r\nAccept: text/html,app
lication/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8\r\nUser-Agent: Moz
illa/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/3
2.0.1700.102 Safari/537.36\r\nAccept-Encoding: gzip,deflate,sdch\r\nAccept-Langu
age: en-US,en;q=0.8,he;q=0.6\r\nCookie: dcsyncynt=true; _dyid=782863874; _dyfs=t
rue; _cb_ls=1; _chartbeat2=uxytmbr82xjw5gdk.1391106319990.1391106319990.1; _char
tbeat_uuniq=1; _dycnoabc=1391106320458; tmcynt=00000000000000000000000000000020;
dyaud_page=47@1294169256@0@3.50@520249338@0@4.95@1392490978@0@29.100@-688318138@
0@33.101@1972278733@0@34.102@-264413004@0@35.103@412490352@0@36.104@-990684182@0
@37.105@1315227445@0@38.106@-971182034@0@39.107@-663019518@0@40.108@-1598492479@
0@41.211@653122546@0@195.212@-1374571664@0@196.213@1928760588@0@197.214@15892548
79@0@198-199.215@584638648@0@200.216@896448668@0@201.217@-1257678123@0@202.313@-
994981885@0@360-361.326@-1708915932@0@375-384.327@-1695460534@0@377.328@-1106950
800@0@378.342@279212711@0@396-397-398.343@921663746@0@399-400-401.344@-129938671
9@0@402-403-404.345@2093435358@0@405-406-407.346@-1400531360@0@408-409-410.347@-9
27614613@0@411-412-413.368@-1308686476@0@449.369@-2012292496@0@450.370@-11919134
7@0@451.371@-118395'
>>>
>>> data_string = str(my_packet[Raw])
>>>
>>> data_string.startswith('GET')
True
>>>
```

בשלב זה, נוכל לכתוב פונקציית filter באמצעותה נסנן חבילות כאלו בלבד:

```
>>> def http_get_filter(packet):
…       return (TCP in packet and Raw in packet and str(packet[Raw]).startswith('GET'))
```

תרגיל 5.3 – הספנה

השתמשו במסנן שהגדרנו קודם לכן, והדפיסו באופן מסודר את כל ה-URLים אליהם מתבצעת בקשת GET במהלך ריצת הסקריפט. שימו לב לכתוב URL מלא, הכולל גם את ה-Host. השתמשו באתר שעבדנו איתו בעבר-

http://info.cern.ch/hypertext/WWW/TheProject.html

[21] על פרוטוקול זה, והסיבה שמשתמשים בו כפרוטוקול שכבת התעבורה של HTTP, נלמד בפרק שכבת התעבורה.

הסנפה – סיכום

ובכן, למדנו להסניף באמצעות Scapy. הצלחנו להסניף חבילות DNS וחבילות HTTP, בנינו מסנן באמצעות **lfilter** והצלחנו לבצע פעולות על החבילות באמצעות **prn**. ראינו גם איך נראות חבילות ב-Scapy ולמדנו קצת איך לעבוד איתן. עם זאת, הכרנו חלק קטן מאוד מהיכולות של הפונקציה **sniff**. נלמד יכולות נוספות בהמשך, אך בטרם נעשה זאת – נכיר פעולות נוספות. הרי להסניף חבילות זה טוב ויפה, אך בהחלט לא מספיק.

יצירת חבילות

לעיתים נרצה פשוט ליצור חבילה משלנו, מבלי להסתמך על חבילה קיימת שהסנפנו מהרשת.

נתחיל ביצירת פקטה של IP[22]. הקלידו את הפקודה הבאה:

```
>>> my_packet = IP()
```

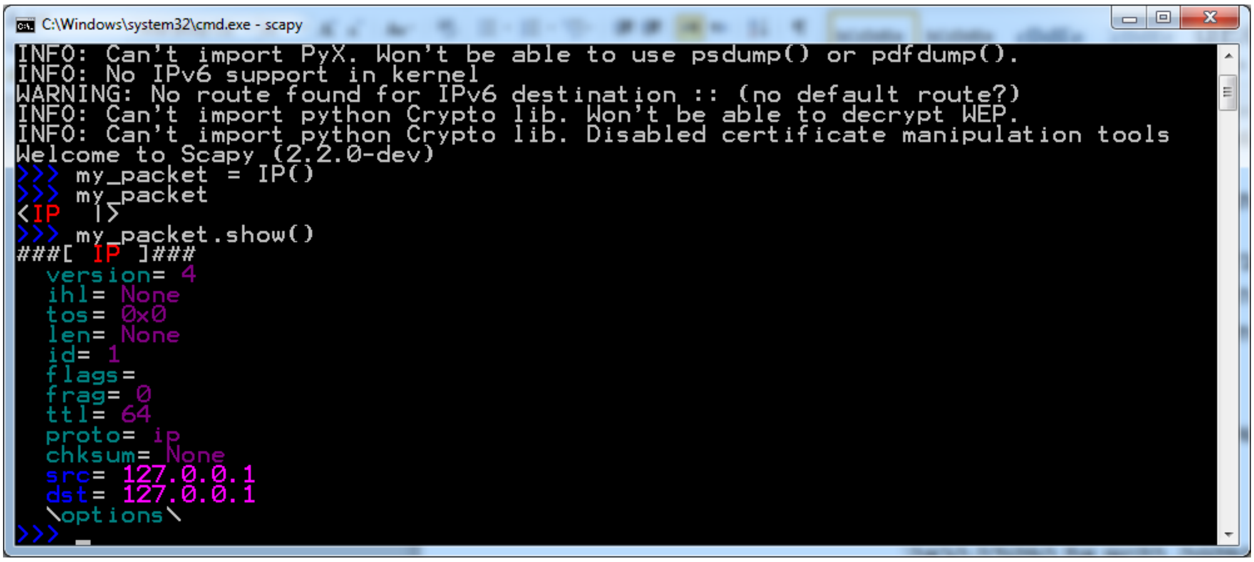
לא קרה הרבה. בואו ננסה לראות את הפקטה. הקלידו את הפקודה הבאה:

```
>>> my_packet
<IP     | >
```

לא הודפס פלט רב למסך. האם הדבר אומר שנוצרה לנו פקטה ללא אף שדה?

בואו ננסה לראות את כל הפרטים על הפקטה, באמצעות המתודה **show()**. עשו זאת כך:

```
>>> my_packet.show()
```



[22] את פרוטוקול IP פגשנו כבר בפרקים הקודמים, ונרחיב את ההיכרות איתו בפרק שכבת הרשת.

איזה יופי! אנחנו רואים את כל השדות. Scapy בנה אותם עבורנו, וגם נתן להם ערכים הגיוניים (למשל הגרסה היא 4, שכן מדובר ב-IPv4. ה-ttl הוא 64, וכך הלאה[23]). מכאן ש-Scapy יודע לייצר עבורנו חבילות עם ערכים הגיוניים, ואנחנו יכולים לדאוג אך ורק לערכים המעניינים אותנו.

האם יש צורך במתודה **show()** בכל פעם שברצוננו לדעת את הערך של שדה יחיד? ממש לא! לדוגמה, לכל חבילה של IP יש כתובת מקור המציינת מי שלח את החבילה, וכתובת היעד – המציינת למי החבילה מיועדת. למשל, חבילה שנשלחה מהמחשב שלנו אל השרת של Google, תכלול בשדה כתובת המקור את כתובת ה-IP של המחשב שלנו, ובשדה כתובת היעד את כתובת ה-IP של השרת של Google.

בואו ננסה לגלות רק מה ערך כתובת המקור (ה-Source Address) של הפקטה:

```
>>> my_packet.src
'127.0.0.1'
```

פשוט וקל.

כעת ננסה לשנות את אחד השדות. נאמר ונרצה שכתובת היעד (ה-Destination Address) של הפקטה תהיה '10.1.1.1'. נוכל לעשות זאת כך:

```
>>> my_packet.dst = '10.1.1.1'
```

כעת אם נסתכל שוב בפקטה, נראה ש-Scapy מציין בפנינו רק את הפרמטר ששינינו:

```
>>> my_packet.dst='10.1.1.1'
>>> my_packet
<IP  dst=10.1.1.1 |>
```

מכאן אנו למדים ש-Scapy מציג לנו רק את הפרמטרים השונים, המעניינים. קודם לכן, כשניסינו להציג את my_packet, הוא בחר שלא להראות לנו את שדה כתובת היעד, שכן כתובת היעד לא השתנתה מערך ברירת המחדל שלה.

כמובן ששינוי כתובת היעד יבוא לידי ביטוי גם כשנבצע my_packet.show(). בצעו זאת בעצמכם כעת.

אם ננסה לשנות יותר משדה אחד, תקרה תופעה דומה:

```
>>> my_packet.ttl = 5
>>> my_packet
```

כאן שינינו את הערך של השדה ttl, וכעת scapy הראה גם את כתובת היעד כשונה, וגם את שדה ה-ttl:

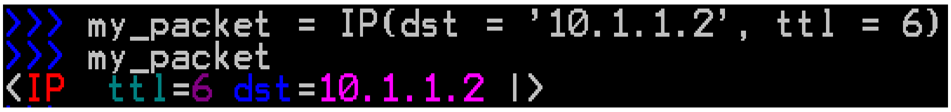
```
>>> my_packet.dst='10.1.1.1'
>>> my_packet
<IP  dst=10.1.1.1 |>
>>> my_packet.ttl = 5
>>> my_packet
<IP  ttl=5 dst=10.1.1.1 |>
```

---

[23] על משמעות מושגים אלו (IPv4 ו-ttl) – נלמד בפרק של שכבת הרשת.

לחלופין, ניתן גם ליצור כך את הפקטה מראש, בצורה הבאה:

```
>>> my_packet = IP(dst = '10.1.1.2', ttl = 6)
>>> my_packet
```



```
>>> my_packet = IP(dst = '10.1.1.2', ttl = 6)
>>> my_packet
<IP  ttl=6 dst=10.1.1.2 |>
```

שכבות

עד כה הצלחנו לבנות פקטה עם שכבה אחת יחידה (IP). כמו שוודאי הבנתם מהפרקים הקודמים בספר, פקטות IP כשלעצמן לא מאוד מעניינות. הן בדרך כלל מגיעות עם שכבה נוספת מעליהן. על מנת להוסיף שכבות אחת מעל השנייה ב-Scapy, נבצע לדוגמה את הפעולה הבאה:

```
>>> my_packet = IP() / TCP()
```

השתמשנו באופרטור / על מנת "להעמיס" שכבה אחת מעל שכבה אחרת. כאן, יצרנו פקטה עם שכבת IP ומעליה שכבה של TCP[24]. בואו נראה כיצד Scapy מציג את הפקטה:

```
>>> my_packet
my_packet
<IP      frag=0 proto=tcp                  |<TCP   |>>
```

(**הערה**: הסוגריים המשולשים נצבעו והודגשו על ידי כותב הספר ולא על ידי Scapy)

שימו לב לדרך הייצוג כאן. שכבת ה-TCP תחומה בידי הסוגריים המשולשים ה**אדומים** (הפנימיים), בעוד שכבת ה-IP תחומה בסוגריים המשולשים ה**כחולים** (החיצוניים). ניתן לראות כי שכבת ה-TCP מוכלת בשכבת ה-IP! היזכרו במושג ה-Encapsulation (או כימוס) עליו דיברנו בפרק מודל השכבות.

IP הוא פרוטוקול שכבה שלישית ו-TCP הוא פרוטוקול שכבה רביעית. בואו ננסה עתה ליצור גם שכבה שנייה, ולשם כך נשתמש בפרוטוקול ה-Ethernet[25]:

```
>>> my_packet = Ether() / IP() / TCP()
my_packet
<Ether type=0x800               | <IP    frag=0 proto=tcp |<TCP                    |>>>
```

---

[24] על פרוטוקול זה נלמד לעומק בפרק שכבת התעבורה.

[25] על פרוטוקול זה נלמד בהרחבה בפרק שכבת הקו.

כעת הפקטה my_packet הינה פקטת Ethernet, שמכילה שכבת IP וכן שכבת TCP. נוכל גם לשנות את הפרמטרים של השכבות השונות בזמן יצירת הפקטה:

```
>>> my_packet = Ether() / IP(ttl = 4) / TCP(dport=80)
>>> my_packet
<Ether  type=0x800 |<IP  frag=0 ttl=4 proto=tcp |<TCP  dport=http |>>>
```

שימו לב לכמה דברים:

1. בשכבת ה-Ethernet מוצא לנו ה-type (ששווה ל-0x800). זאת מכיוון שה-type מצביע על כך שהשכבה הבאה היא אכן שכבת IP.
2. בשכבת ה-IP כעת מוצא גם ה-ttl. זאת מכיוון ששמנו בו ערך לא ברירת מחדל בשורה הקודמת.
3. בשכבת ה-TCP ציינו שפורט היעד (Destination port, או בקיצור dport) יהיה 80. Scapy יודע לבצע את התרגום ולהציג אותו כפורט הייעודי של HTTP[26] – ממש כמו ש-Wireshark יודע לעשות.

דבר נוסף שאפשר לבצע ב-Scapy הוא להוסיף מידע "Raw", שישמש אותנו כ-Payload לשכבה הנוכחית. Payload של שכבה הוא המידע ש"מעליה". כך למשל, ה-Payload של שכבת IP יכול להיות שכבת ה-TCP וכל המידע שלה, וה-Payload של TCP עשוי להיות, למשל, HTTP. ראינו שניתן להוסיף מידע "Raw" באמצעות Scapy כאשר הסנפנו חבילה של HTTP קודם לכן. על אף ש-Scapy, כאמור, לא מכיר את פרוטוקול HTTP, נוכל ליצור חבילת HTTP בצורה הבאה:

>>> Ether()/IP()/TCP()/Raw("GET / HTTP/1.0\r\n\r\n")

ראו מה מתרחש:

```
>>> my_packet = Ether() / IP() / TCP() / Raw("GET / HTTP/1.0\r\n\r\n")
>>> my_packet
<Ether  type=0x800 |<IP  frag=0 proto=tcp |<TCP  |<Raw  load='GET / HTTP/1.0\r\n
\r\n' |>>>>
```

נוכל לבצע זאת גם מבלי לכתוב Raw, אלא לכתוב את השורה בצורה הפשוטה:

>>> Ether()/IP()/TCP()/"GET / HTTP/1.0\r\n\r\n"

ניתן גם לבקש ייצוג Hexdump (הצגת המידע בפורמט הקסדצימלי) של הפקטה, כך שהיא תיראה בדומה לדרך בה היא מוצגת ב-Wireshark:

```
>>> my_packet = Ether() / IP() / TCP() / Raw("Cyber rulez")
>>> hexdump(my_packet)
0000   FF FF FF FF FF FF 00 00  00 00 00 00 08 00 45 00   ..............E.
0010   00 33 00 01 00 00 40 06  7C C2 7F 00 00 01 7F 00   .3....@.|.......
0020   00 01 00 14 00 50 00 00  00 00 00 00 00 00 50 02   .....P........P.
0030   20 00 20 97 00 00 43 79  62 65 72 20 72 75 6C 65    . ...Cyber rule
0040   7A                                                z
```

[26] נדון בנושא הפורטים בפרק שכבת התעבורה.

Resolving

בפרקים הקודמים הזכרנו את התרגום של שמות דומיין (כגון "www.google.com") לכתובות IP (כגון "172.15.23.49"). תהליך זה נקרא Resolving, ו-Scapy יודע לבצע אותו בשבילנו. לכן, אם נכתוב את השורה הבאה:

```
>>> my_packet = IP(dst = "www.google.com")
```

Scapy יציב בשדה כתובת היעד של ה-IP את כתובת ה-IP של Google. נסו זאת בעצמכם!

שימו לב שכאשר תסתכלו על החבילה שיצרתם (באמצעות המתודה **show** למשל), תראו כי Scapy לא רושם את כתובת ה-IP אלא את הדומיין שציינתם. עם זאת, כאשר נשלח את החבילה (כמו שנלמד לעשות בהמשך הפרק), Scapy ידאג להבין את כתובת ה-IP הרלוונטית ולהציב אותה בשדה הרלוונטי.

שליחת פקטות

עד כאן למדנו ליצור בעצמנו פקטות בשכבות שונות, וכן להציג אותן על המסך. כעת נלך צעד אחד קדימה, ונלמד גם **לשלוח** את הפקטות שיצרנו.

Scapy מציע, בגדול, שתי דרכים לשלוח פקטות: שליחה בשכבה שלישית ושליחה בשכבה שנייה. בשלב זה נתעלם מהאופציה לבצע שליחה ברמה שנייה, אך נחזור אליה בהמשך הספר.

ניצור חבילה שמתחילה בשכבה שלישית, למשל חבילת IP ומעליה מידע טקסטואלי בסיסי (זוהי לא חבילה תקינה, ו-Google לא באמת צפוי להגיב אליה):

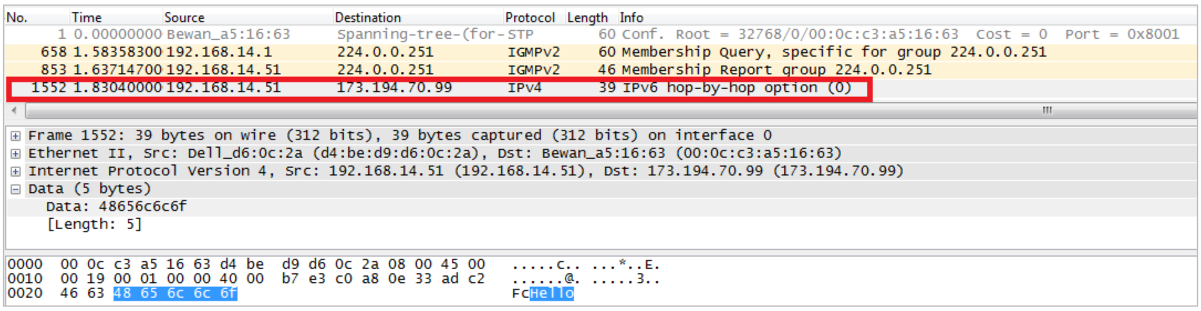
```
>>> my_packet = IP(dst = "www.google.com") / Raw(“Hello”)
```

פתחו את Wireshark והריצו הסנפה.

כעת נוכל לשלוח את החבילה:

```
>>> send(my_packet)
.
Sent 1 packets.
```

נסו לזהות את החבילה בהסנפה שלכם ב-Wireshark.



שימוש ב-Scapy מתוך קובץ סקריפט

כאשר נרצה לכתוב תוכנית / סקריפט שמבצע פעולות באופן קבוע ואוטומטי (כלומר – קובץ בעל הסיומת py, ולא עבודה מתוך ה-Interperter), ניתן להשתמש ב-Scapy לצרכים אלו באופן דומה לשימוש בחבילות אחרות ב-python. לשם כך, יש לבצע import בצורה הבאה:

from scapy.all import *

כעת ניתן להשתמש באובייקטים של Scapy.

לדוגמה, נשתמש בקוד שכתבנו מוקדם יותר בפרק בכדי להדפיס את כל הדומיינים שעבורם מתבצעת שאילתת DNS:

```
from scapy.all import *
def print_query_name(dns_packet):
    ‘”This function prints the domain name from a DNS query”’
    print(dns_packet[DNSQR].qname)

def filter_dns(packet):
    ‘”This function filters query DNS packets”’
    return (DNS in packet and packet[DNS].opcode == 0 and packet[DNSQR].qtype == 1)

print ‘Starting to sniff!’
sniff(count=10, lfilter=filter_dns, prn=print_query_name)
```

היכולת ליצור ולשלוח חבילות תשמש אותנו רבות בהמשך הספר.

תרגילי Scapy – 5.4

1. צרו חבילת IP באמצעות Scapy, ודאגו שכתובת היעד שלה תהיה השרת של "www.google.com". השתמשו במתודה **show**, וודאו שאכן החבילה שיצרתם מיועדת אל Google.
2. שלחו את החבילה שיצרתם בסעיף הקודם. הסניפו באמצעות Wireshark, ובדקו שאתם מצליחים לראות את החבילה, ושהיא אכן נשלחה אל הכתובת של Google.
3. פתחו את הדפדפן שלכם, וגלשו אל Facebook. הסניפו באמצעות Scapy את החבילות שנשלחות בין המחשב שלכם לבין Facebook, והדפיסו סיכום שלהן באמצעות המתודה **summary**. שימו לב לסנן רק חבילות שנשלחות ביניכם לבין השרת של Facebook.

Scapy – סיכום

בפרק זה למדנו **קצת** על Scapy ואיך להשתמש בו. במהלך הפרק ראינו כיצד מתקינים את Scapy, ואחר כך למדנו לבנות באמצעותו חבילות, לשלוח אותן ולהסניף חבילות המתקבלות ברשת. בתקווה שהצלחתם להבין את דרך העבודה עם הכלי הנהדר הזה, וגם קצת לספוג את ההתלהבות מהשימוש בו והכוח הרב שהוא נותן.

כפי שנכתב בתחילת הפרק – Scapy הוא כלי. בדומה לקריאה וכתיבה, הוא לא נותן המון כשלעצמו. אך כשמחברים אותו עם דברים נוספים (כמו הידע הנרכש שלכם ברשתות), בהחלט אפשר להגיע רחוק. Scapy ימשיך ללוות אותנו לאורך הספר כולו. ניעזר בו בכדי לתרגל נושאים קיימים, וכן נעמיק את היכרותנו עימו.

Scapy – צעדים להמשך

אלו מכם שמעוניינים להעמיק את הידע שלהם ב-Scapy, מוזמנים לבצע את הצעדים הבאים:

קריאה נוספת

ניתן ומומלץ לקרוא עוד רבות על Scapy באתר הפרויקט: http://www.secdev.org/projects/scapy/doc.
כמו כן, ניתן להיעזר במסמך Scapy Cheat Sheet, שנמצא בכתובת: http://goo.gl/tUy6Aq.

תרגיל מתקדם

בתרגיל זה תממשו שרת DNS באמצעות Scapy. בצעו את התרגיל בשלבים. בכל שלב, בדקו את השרת שלכם בטרם תמשיכו לשלב הבא. תוכלו להיעזר בכלי **nslookup** אותו פגשנו בפרק תחילת מסע – איך עובד האינטרנט/ DNS, על מנת לבדוק את השרת שלכם. שימו לב שעל מנת לבדוק תרגיל זה, עליכם להשתמש בשני מחשבים שונים – אחד ללקוח ואחד לשרת[27].

[27] באופן תאורטי, יכלנו לעשות זאת מעל loopback device – כלומר מעל הכתובת "127.0.0.1", המוכרת לנו מתרגילים קודמים. עם זאת, עקב Bug של Scapy בשליחה וקבלת מסגרות מעל loopback device ב-Windows, נשתמש בשני מחשבים.

שלב ראשון – שרת עצמאי

- על השרת להאזין לשאילתות DNS נכנסות בפורט 53, ולענות עליהן.
- השרת צריך לתמוך רק בסוגי הרשומות A ו-PTR[28].
- על השרת לשמור קובץ TXT שיכיל את מסד הנתונים שלו. בכל רשומה יהיה רשום סוג הרשומה, הערכים וה-TTL.
- כאשר לקוח פונה לשרת בבקשת DNS, אם הוא פונה עבור רשומה שקיימת במסד הנתונים של השרת, על השרת לענות לו תשובה תקינה בהתאם למסד הנתונים.
- אם הלקוח פנה בבקשה לרשומה שלא קיימת במסד הנתונים של השרת, על השרת להגיב בתשובת DNS עם השגיאה "no such name" (באמצעות שדה ה-flags ב-DNS).

שלב שני – שרת חברתי

בשלב הקודם השרת שלכם פעל לבדו. אם הוא לא הכיר שם דומיין מסוים – הוא לא הצליח לתת שירות טוב ללקוח. כידוע, שרתים שיודעים לפעול בצורה חברתית מסוגלים לתת שירות טוב יותר. שפרו את השרת שכתבתם בשלב הקודם:

- במידה שהשרת לא מכיר שם דומיין שעליו הוא נשאל, הוא ישלח את השאילתה לשרת DNS אחר.
- במידה ששרת ה-DNS האחר ידע לענות על השאילתה, הוא יחזיר את התשובה התקינה של שרת ה-DNS אל הלקוח.
- במידה ששרת ה-DNS האחר לא ידע לענות על השאילתה, השרת שלכם יחזיר הודעת שגיאה "no such name".

שלב שלישי – שרת עם מטמון

הוסיפו לשרת שלכם יכולות מטמון (Caching).

- במידה שהשרת נשאל על שם שהוא לא הכיר, והשיג את פרטי השם הזה משרת אחר, הוא יוסיף את המידע שהוא גילה אל מסד הנתונים שלו.
- בפעם הבאה שהשרת ישאל על השם הזה, הוא יוכל לענות בעצמו ולא יזדקק לשרת נוסף.

---

[28] ל-Scapy יש Bug ידוע עם חבילות PTR. כדי להתגבר עליו, תוכלו לגשת לקובץ "dns.py" בתיקייה "layers", ולהחליף את השורה:

```
elif pkt.type in [2,3,4,5]: # NS, MD, MF, CNAME
```

בשורה הבאה:

```
elif pkt.type in [2,3,4,5,12]: # NS, MD, MF, CNAME, PTR
```

פרק 6

שכבת התעבורה

עד כה התעסקנו באפליקציות. הצלחנו לשלוח מידע מתוכנה שנמצאה במחשב אחד לתוכנה במחשב אחר. אבל איך כל זה קרה? איך עובד הקסם הזה של העברת מידע בין מחשב אחד למחשב אחר? בפרק זה נתחיל להפיג את הקסם, ולהסביר לעומק איך הדברים עובדים.

במהלך הפרק הקרוב נלמד מהם **פורטים (ports)**, נכיר כלים ומושגים חדשים ונלמד על הפרוטוקולים UDP ו-TCP. נכתוב תוכנה להעברת מידע סודי, ונצליח לגלות אילו שירותים פתוחים במחשב מרוחק. על מנת לעשות זאת, עלינו להבין את שכבת התעבורה.

מה התפקיד של שכבת התעבורה?

שכבת התעבורה אחראית להעביר מידע מתוכנית (תהליך) לתוכנית (תהליך) מרוחקת. כחלק מכך, יש לה שתי מטרות עיקריות:

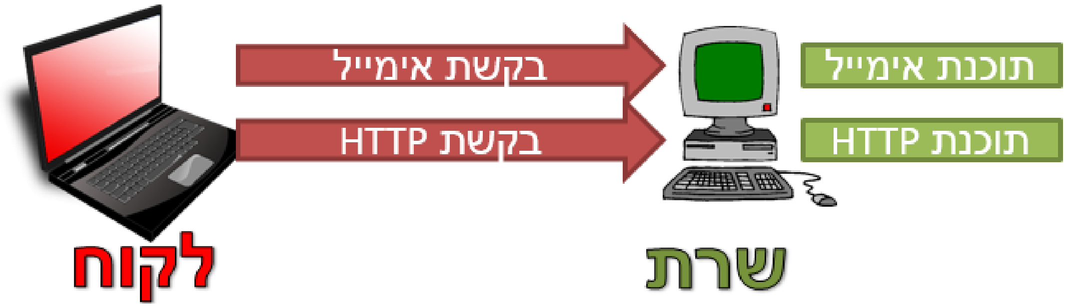
- ריבוב מספר אפליקציות עבור אותה הישות – כלומר היכולת לתקשר עם ישות רשת אחת (אל מול אותה כתובת IP בודדת) ולהשתמש בכמה שירותים שונים של הישות, כך שהישות תדע להבדיל איזה זרם מידע שייך לאיזה שירות שהיא מספקת. מטרה זו קיימת **תמיד** בשכבת התעבורה.
- העברה אמינה של מידע. זוהי מטרה אופציונלית, ומכאן שהיא לא קיימת בכל המימושים של שכבת התעבורה (כלומר, לא בכל הפרוטוקולים של שכבת התעבורה).

ריבוב אפליקציות – פורטים

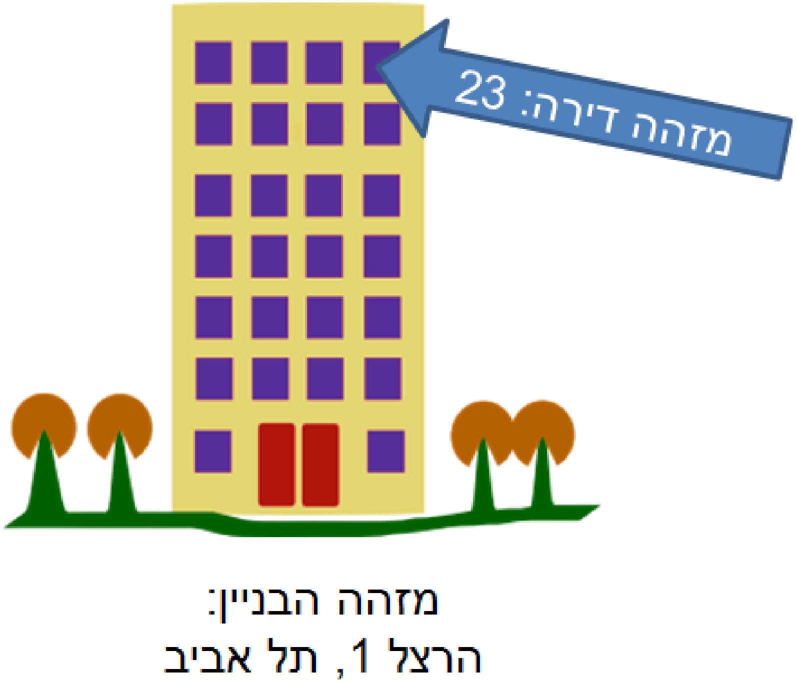
נאמר ויש לנו חיבור בין שרת ללקוח. כעת, הלקוח שולח בקשת אימייל לשרת מעל חיבור זה:

הדבר הגיוני בהנחה שהשרת מריץ שירות של אימייל. עם זאת, ייתכן שהלקוח ישלח יותר מבקשה אחת לשרת. למשל, ייתכן שהשרת מריץ גם שירות של שרת אימייל וגם שירות של שרת Web (למשל – HTTP עליו למדנו בפרק שכבת האפליקציה). מה יקרה אם הלקוח ישלח אל השרת גם בקשה של אימייל, וגם בקשת HTTP?

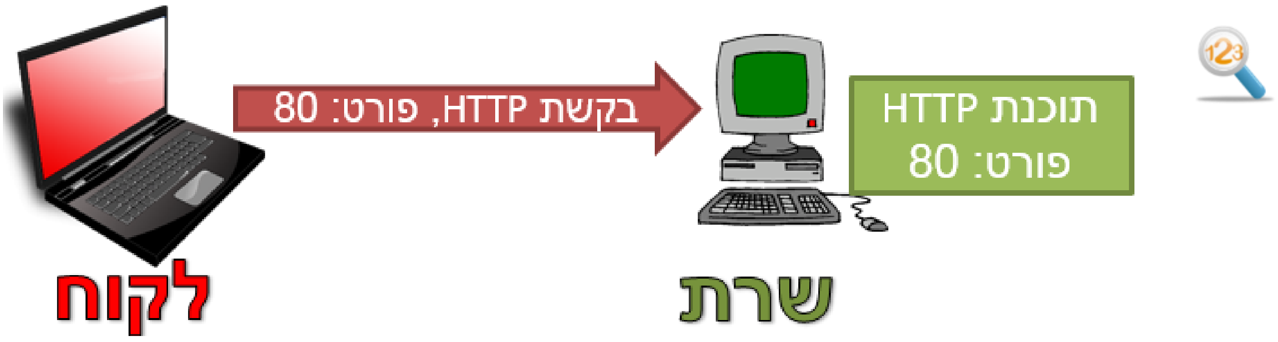
כעת, על השרת להבין לאיזה שירות שלו נשלחה הבקשה. במקרה זה, תהיה אצל השרת תוכנה שתטפל בבקשות מלקוחות הקשורות באימייל, ותוכנה שתטפל בבקשות HTTP. על השרת להצליח להפריד ביניהן, כדי להפנות את הבקשה לתוכנה המתאימה:



לשם כך, יש לנו צורך במזהה התוכנה. לא מספיק שהלקוח יודע לפנות אל השרת (להזכירכם, מזהה השרת באינטרנט הוא כתובת IP), הוא צריך גם לספק מזהה של התוכנה הספציפית אליה הוא פונה. בפרק תכנות ב- Sockets / כתובות של Socket, דימינו זאת לשליחת מכתב דואר בין שתי משפחות הגרות בשכונה של בתים רבי קומות. ציינו כי מזהה הרכיב (במקרה הזה – השרת) הינו מזהה הבניין של המשפחה – למשל "רחוב הרצל בעיר תל אביב, בית מספר 1". מזהה התוכנה (במקרה זה – תוכנת האימייל או תוכנת ה-HTTP) הוא מזהה הדירה הספציפית בבניין, למשל "דירה 23".



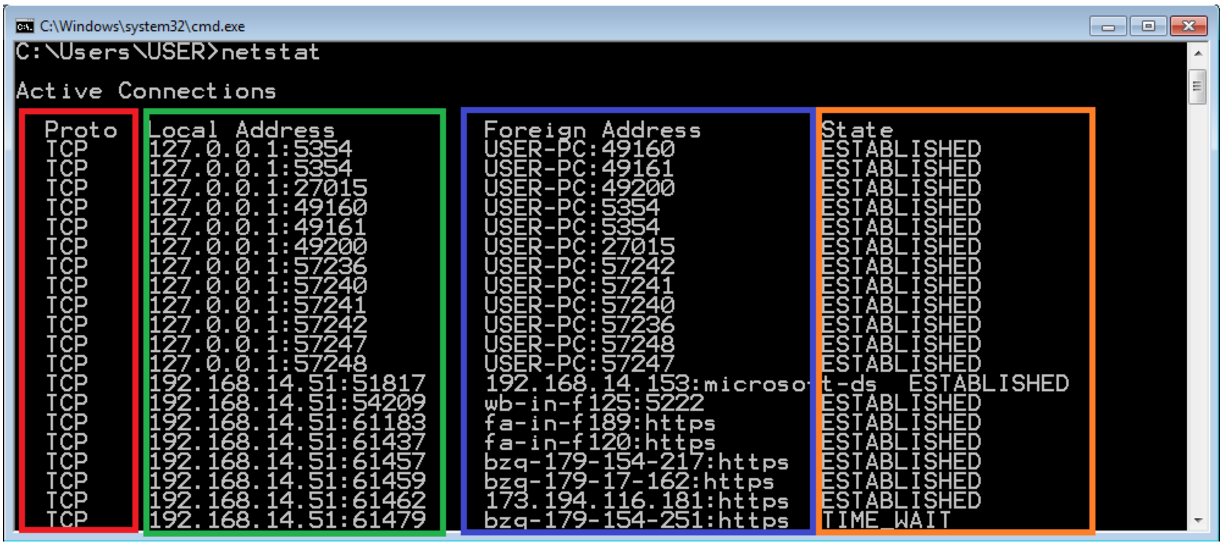
בעולם הרשת, מזהה הבניין הוא כתובת IP, ומזהה הדירה נקרא **פורט (Port)**. באמצעות פנייה לפורט מסוים בבקשה, השרת יכול לדעת לאיזו תוכנה אנו פונים. כך לדוגמה, באם נשלח הודעה לפורט מספר 80 (באמצעות פרוטוקול TCP, עליו נדבר בהמשך הפרק), השרת צפוי להבין שאנו פונים לתוכנת ה-HTTP ולא לתוכנה המייל, מכיוון שתוכנת ה-HTTP **מאזינה** על פורט 80:



תרגיל 6.1 מודרך – אילו פורטים פתוחים במחשב שלי?

המונח "פורט פתוח" מתייחס לפורט שתוכנה כלשהי מאזינה עליו. כלומר, אם יפנו אל הפורט הזה, תהיה תוכנה שמוכנה לקבל חיבור. אם יש לנו שרת שמריץ תוכנת HTTP שמאזינה על פורט 80, ואין תוכנות נוספות שמאזינות על פורטים נוספים, אז פורט 80 ייקרא "פתוח" בעוד פורט 81 למשל ייקרא "סגור".

כעת נלמד כיצד לגלות אילו פורטים פתוחים במחשב שלנו. לשם כך, פתחו את ה-Command Line והריצו את הפקודה **netstat**:



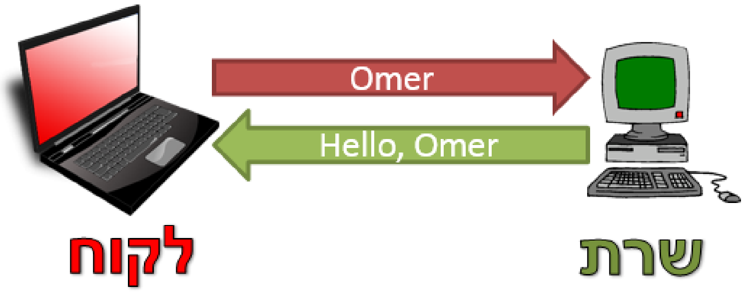
❶ ❷ ❸ ❹

נבחן את הפלט של הפקודה **netstat**:

- **באדום** (1) – אנו רואים את הפרוטוקול שעליו המחשב מבצע האזנה. בשכבת התעבורה ישנם שני פרוטוקולים נפוצים עליהם נלמד בהמשך הפרק, והם TCP ו-UDP.
- **בירוק** (2) – הכתובת המקומית עליה המחשב מאזין. הכתובת כתובה בפורמט של "IP:Port". כך לדוגמה בשורה הראשונה, כתובת ה-IP הינה 127.0.0.1, והפורט הינו 5354. התעלמו מכתובות ה-IP בשלב זה, נלמד להכיר אותן בהמשך הספר.
- **בכחול** (3) – הכתובת הרחוקה אליה המחשב מחובר. במידה שאנחנו לא רק מחכים לחיבור, אלא חיבור כבר קיים, **netstat** יודע להציג גם את הכתובת המרוחקת של החיבור. כך למשל, החיבור הראשון הינו מפורט 5354 במחשב שלנו, אל פורט 49160 במחשב בשם USER-PC (שהוא למעשה המחשב ממנו רצה הפקודה, מכיוון שבמקרה זה מדובר בתקשורת מקומית על המחשב).
- **בכתום** (4) **–** אנו רואים את מצב החיבור. נלמד על משמעות מידע זה בהמשך הפרק.

נזכיר שרצינו לדעת אילו פורטים פתוחים במחשב שלנו. מכאן שהמידע שמעניין אותנו נמצא בטור ה**ירוק** (2).

כעת ננסה למצוא את החיבור שלנו כשנריץ את השרת שכתבנו בפרק תכנות ב-Sockets. הריצו את השרת הראשון שכתבנו בפרק תכנות ב- Sockets / תרגיל 2.3 מודרך – השרת הראשון שלי, זה שמקבל שם מהלקוח ומחזיר לו תשובה בהתאם:



להזכירכם, השרת בתרגיל האזין על פורט 8820. הריצו את השרת:

```
C:\Windows\system32\cmd.exe - server.py
C:\Users\USER>cd \Cyber

C:\Cyber>server.py
```

אפשרו לשרת להמשיך לרוץ. כעת, הריצו שוב את הכלי **netstat**. אינכם צפויים לראות את ההאזנה. דבר זה נובע מכך שבאופן ברירת מחדל, **netstat** מציג רק חיבורים קיימים. כלומר, כל עוד אף לקוח לא התחבר לשרת שהרצתם, לא תראו שהמחשב שלכם מאזין על הפורט הרלוונטי. בכדי לגרום ל-**netstat** להציג בכל זאת את החיבור שלנו, נשתמש בדגל a-[29], כלומר נריץ את הפקודה בצורה הבאה:

netstat -a

כעת אם נביט בפלט, נוכל לראות את ההאזנה שאנו מבצעים:

```
C:\Windows\system32\cmd.exe
C:\Users\USER>netstat -a

Active Connections

  Proto  Local Address          Foreign Address        State
  TCP    0.0.0.0:22             USER-PC:0              LISTENING
  TCP    0.0.0.0:135            USER-PC:0              LISTENING
  TCP    0.0.0.0:445            USER-PC:0              LISTENING
  TCP    0.0.0.0:5357           USER-PC:0              LISTENING
  TCP    0.0.0.0:49152          USER-PC:0              LISTENING
  TCP    0.0.0.0:49153          USER-PC:0              LISTENING
  TCP    0.0.0.0:49154          USER-PC:0              LISTENING
  TCP    0.0.0.0:49155          USER-PC:0              LISTENING
  TCP    0.0.0.0:49157          USER-PC:0              LISTENING
  TCP    0.0.0.0:49162          USER-PC:0              LISTENING
  TCP    0.0.0.0:49165          USER-PC:0              LISTENING
  TCP    127.0.0.1:2559         USER-PC:0              LISTENING
  TCP    127.0.0.1:5354         USER-PC:0              LISTENING
  TCP    127.0.0.1:5354         USER-PC:49160          ESTABLISHED
  TCP    127.0.0.1:5354         USER-PC:49161          ESTABLISHED
  TCP    127.0.0.1:27015        USER-PC:0              LISTENING
  TCP    127.0.0.1:27015        USER-PC:49200          ESTABLISHED
  TCP    127.0.0.1:49160        USER-PC:5354           ESTABLISHED
  TCP    127.0.0.1:49161        USER-PC:5354           ESTABLISHED
  TCP    127.0.0.1:49200        USER-PC:27015          ESTABLISHED
```

למעשה, ניתן לראות את ההאזנה כבר בשורה הראשונה!

שימו לב שכעת ה-State הינו LISTENING. מכך אנו למדים שהמשמעות של LISTENING היא שהמחשב מחכה ליצירת חיבור. שורה שה-State שלה הוא ESTABLISHED, מתארת חיבור רץ וקיים.

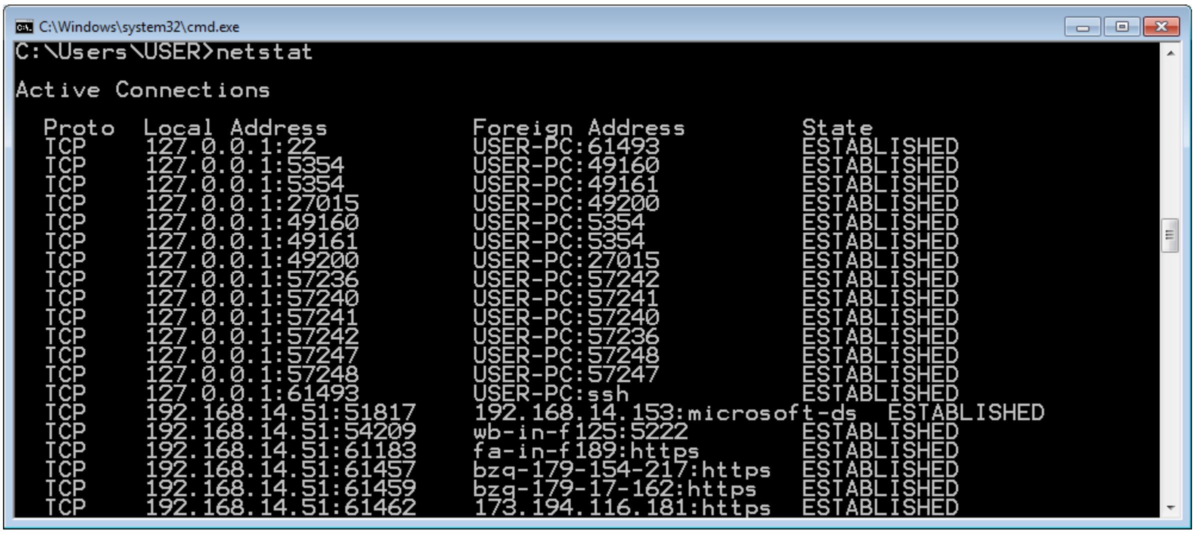
[29] דגל (באנגלית flag) בהקשר הזה הינו פרמטר לפקודה. כך למשל, הפרמטר "a-" לפקודה netstat אשר מציין לפקודה להראות את כל החיבורים.

בואו נבחן זאת. פתחו את Python, וכתבו לקוח קטן אשר מתקשר עם השרת אותו יצרתם, כפי שלמדנו בפרק תכנות ב-Sockets. אל תנתקו את החיבור בסופו ואל תסגרו את אובייקט ה-**socket**, שכן אנו מנסים לשמור על החיבור פתוח. להלן דוגמה לקוד כזה:

```python
import socket
my_socket = socket.socket()
my_socket.connect(('127.0.0.1', 22))

print('I am connected')
input()
```

הערה: ההוראה ()input תמנע מן הסקריפט לסיים את הריצה כאשר תריצו אותו, שכן היא גורמת לסקריפט לחכות לקלט מהמשתמש.

הריצו את הקוד. כעת הריצו שוב את הפקודה **netstat**:



הפעם אין צורך בדגל a-. מכיוון שהחיבור קיים (במצב ESTABLISHED), הרי ש-**netstat** מציג לנו אותו גם ללא שימוש בדגל זה. בדוגמה לעיל, השורה הרלוונטית היא השורה הראשונה.

מי מחליט על מספרי הפורטים?

בפועל, פורט הינו מספר בין 0 ל-65,535. על מנת שתוכנה אחת תוכל להתחבר לתוכנה מרוחקת, עליה לדעת את הפורט שבו התוכנה המרוחקת מאזינה.

לשם כך, ישנם **פורטים מוכרים (Well known ports)**. אלו הם הפורטים מ-0 ועד 1023, והם הוקצו בידי IANA (Internet Assigned Number Authority)[30]. כך למשל ידוע שהפורט 80 משויך לפרוטוקול HTTP. ישנם פורטים נוספים אשר הוקצו בידי IANA ולא נמצאים בטווח 0-1023. במקרה אחר, מפתחי אפליקציות פשוט צריכים להסכים על הפורט בו הם משתמשים. כך למשל, בשרת הדים שכתבנו בפרק תכנות ב- Sockets / תרגיל 2.5 – מימוש שרת הדים, החלטנו להאזין על פורט 1729. במקרה זה, כל לקוח שירצה להשתמש בשרת שלנו, יצטרך לדעת שאנו משתמשים במספר הפורט הזה בכדי להצליח לגשת לשרת.

העברה אמינה של מידע

עד כה דיברנו על אחת המטרות של שכבת התעבורה, והיא ריבוב תקשורת של כמה תוכנות. מטרה נוספת של שכבת התעבורה הינה סיפוק העברת מידע בצורה אמינה.

הרשת בה משתמשת שכבת התעבורה בכדי להעביר מידע עשויה להיות לא אמינה. כלומר, חבילות מידע יכולות "ללכת לאיבוד" בדרך ולא להגיע ליעדן, או אולי להגיע בסדר הלא נכון (חבילה מספר 2 תגיע לפני חבילה מספר 1). שכבת האפליקציה לא רוצה להתעסק בכך. היא רוצה לבקש משכבת התעבורה להעביר מידע מתוכנה אחת לתוכנה שנייה, ולא לדאוג למקרה שהחבילה לא תגיע. לשם כך, שכבת התעבורה צריכה לספק העברה אמינה של מידע מצד לצד.

עם זאת, לא תמיד נרצה ששכבת התעבורה תספק העברה אמינה של המידע. לכן, מטרה זה היא אופציונלית בלבד – ובחלק מהמימושים של פרוטוקולי שכבת התעבורה אין הבטחה שהמידע יגיע ושיגיע בסדר הנכון. בהמשך הפרק נכיר פרוטוקולים שונים של שכבת התעבורה, וכן נבין מדוע לעיתים נעדיף להשתמש בפרוטוקול שמבטיח אמינות, ובמקרים אחרים נעדיף פרוטוקול שלא מבטיח אמינות.

מיקום שכבת התעבורה במודל השכבות

שכבת התעבורה הינה השכבה הרביעית במודל חמש השכבות.

מה השירותים ששכבת התעבורה מספקת לשכבה שמעליה?

עבור השכבה החמישית, שכבת האפליקציה, היא מאפשרת:

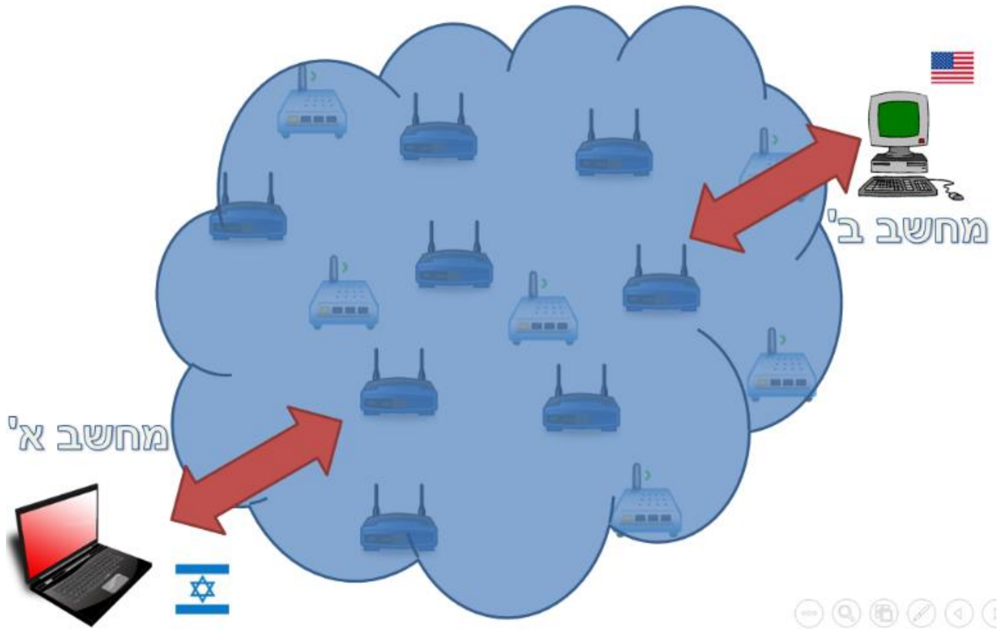
- לשלוח ולקבל מידע מתוכנה (תהליך) מרוחקת.
- במידה שהחיבור אמין – היא מאפשרת ליצור חיבור בין תוכנות שונות, וכן לסגור את החיבור.

[30] IANA (דף הבית: https://www.iana.org) הוא ארגון שאחראי על ניהול והקצאה יחודית של מספרים באינטרנט.

מכאן שעבור שכבת האפליקציה, שכבת התעבורה מאפשרת להעביר מידע מהתהליך שלה אל הצד השני של הרשת.

מה השירותים ששכבת התעבורה מקבלת מן השכבה שמתחתיה?

שכבת הרשת, השכבה השלישית, מספקת לשכבת התעבורה מודל של "ענן", שבו חבילות מידע מגיעות מצד אחד לצד שני. שכבת התעבורה אינה מודעת כלל למבנה הרשת המתואר, ולמעשה מבחינתה יש פשוט "רשת כלשהי" שמחברת בין מחשב א' למחשב ב'. "תמונת הרשת", מבחינתה, נראית כך:



שימו לב ששכבת הרשת אינה מודעת לפורטים. לכן, בשכבת הרשת העברת חבילת מידע מתבצעת מישות לישות (לדוגמה – בין מחשב למחשב), ובשכבת התעבורה, היא מתבצעת מתוכנה אחת לתוכנה אחרת (כלומר – מפורט מסוים אל פורט אחר).

פרוטוקולים מבוססי קישור ולא מבוססי קישור

בשכבת התעבורה, פרוטוקולים יכולים להיות מבוססי קישור (Connection Oriented) או לא מבוססי קישורי (Connection Less).

פרוטוקולים מבוססי קישור

ניתן להמשיל פרוטוקולים מבוססי קישור למערכת הטלפוניה. כדי לתקשר עם מישהו באמצעות הטלפון, עלינו להרים את מכשיר הטלפון, לחייג את המספר שלו, לדבר ואז לנתק את השיחה. לא ניתן פשוט לדבר אל מכשיר הטלפון, ולצפות שאדם בצד השני יקבל את המסר שלנו, אם כלל לא חייגנו אליו. באופן דומה, על מנת לתקשר עם מישהו באמצעות פרוטוקול מבוסס קישור, יש ראשית "להקים" את הקישור, לאחר מכן להשתמש

בקישור שהוקם ולבסוף לנתק את הקישור. מבחינת המשתמש, הוא מתייחס לקישור כמו לשפופרת הטלפון: הוא מזין מידע (במקרה שלנו – רצף של בתים) לקצה אחד, והמשתמש השני יקבל את המידע בצד השני.

דוגמה לפרוטוקול מבוסס קישור היא **פרוטוקול TCP** (שבקרוב נכיר לעומק), או בשמו המלא – Transmission Control Protocol. בתור מפתחי שכבת האפליקציה, כשאנו משתמשים ב-TCP בכדי להעביר מידע, איננו יכולים פשוט לשלוח חבילה אל תוכנה מרוחקת. ראשית עלינו ליצור קישור עם התוכנה המרוחקת, ועתה כל חבילה שנשלח תהיה חלק מאותו קישור.

פרוטוקולים מבוססי קישור מבטיחים אמינות בשליחת המידע. כלומר, הם מבטיחים שכל המידע שנשלח יגיע אל המקבל, וכן שהוא יגיע בסדר שבו הוא נשלח. עם זאת, לפרוטוקולים מבוססים קישור יש **תקורה (Overhead)** גבוהה יחסית. כלומר, יש מידע רב שנשלח ברשת בנוסף על המידע שרצינו להעביר. באם נרצה להעביר את המסר "שלום לכם" באמצעות פרוטוקול מבוסס קישור, עלינו להרים את הקישור לפני שליחת ההודעה, לסיים את הקישור בסיום, ולהשתמש במנגנונים שונים כדי להבטיח שהמסר אכן הגיע אל היעד. פעולות אלו לוקחות זמן ומשאבים, ולכן העברת המסר "שלום לכם" תהיה איטית יותר מאשר שליחת המסר מבלי הרמת הקישור.

תקורה קיימת גם במקומות אחרים בחיים. למשל, על מנת ללמוד שיעור שמתרחש בבית הספר, עליכם לקום מהמיטה, להתלבש, לצאת מהבית, ולהגיע אל בית הספר. במידה שהתמזל מזלכם, אתם יכולים להגיע ברגל. אם אתם גרים במרחק מסוים, ייתכן שעליכם להגיע אל תחנת האוטובוס, להמתין עד שיגיע האוטובוס ולנסוע באמצעותו אל בית הספר. כל זאת הינה תקורה של התהליך – מטרתכם היא אמנם ללמוד בשיעור בבית הספר, אך עליכם לעבור תהליך על מנת לעשות זאת. במקרה זה, ניתן היה למשל להנמיך את התקורה אם הייתם בוחרים לישון בבית הספר, ובכך הייתה נמנעת התקורה של תהליך ההגעה. עם זאת, כפי שוודאי מובן לכם, לפעמים עדיף לשלם את מחיר התקורה על מנת לקבל את היתרונות שהיא מציעה (שינה בבית, או העברה אמינה של מידע מעל הרשת).

פרוטוקולים שאינם מבוססי קישור

ניתן להמשיל פרוטוקולים שאינם מבוססי קישור לרשת הדואר. כל מכתב שאנו שולחים באמצעות הדואר כולל את כתובת היעד שלו, וכל מכתב עומד בזכות עצמו: הוא עובר ברשת הדואר מבלי קשר למכתבים אחרים שנשלחים. ברוב המקרים, אם נשלח שני מכתבים מכתובת אחת לכתובת שנייה, המכתב הראשון שנשלח יהיה זה שיגיע ראשון. עם זאת, אין לכך הבטחה, ולעיתים המכתב השני שנשלח יגיע קודם לכן.

דוגמה לפרוטוקול שאינו מבוסס קישור היא **פרוטוקול UDP** (שבקרוב נכיר לעומק), או בשמו המלא – User Datagram Protocol. כשאנו, בתור מפתחי שכבת האפליקציה, משתמשים ב-UDP בכדי לשלוח חבילה, אין הבטחה שהחבילה תגיע ליעדה. כמו כן, אין הבטחה שהחבילות יגיעו בסדר הנכון. אי לכך, אין גם צורך

בהרמה וסגירה של קישור. באם מתכנת בשכבת האפליקציה רוצה לשלוח חבילה מעל פרוטוקול UDP, הוא פשוט שולח את החבילה.

מתי נעדיף פרוטוקול מבוסס קישור ומתי פרוטוקול שלא מבוסס קישור?

לפרוטוקולים מבוססי קישור, כמו TCP, יתרונות רבים. הם מבטיחים הגעה של המידע בצורה אמינה ובסדר הנכון. אי לכך, נבחר להשתמש בהם במקרים רבים. לדוגמה, כאשר אנו מורידים קובץ מהאינטרנט, הגיוני שנעשה זאת מעל TCP: לא נרצה שחלק מהקובץ יהיה חסר, שכן אז לא נוכל לפתוח אותו. כמו כן לא נרצה שחלקים מהקובץ יגיעו בסדר לא נכון, ואז הקובץ לא יהיה תקין.

עם זאת, לא תמיד נרצה להשתמש בפרוטוקול מבוסס קישור כגון TCP. כמו שלמדנו קודם, ל-TCP יש תקורה גבוהה יחסית: יש צורך בהקמה וסגירה של קישור, יש צורך לוודא שהמידע הגיע ליעד והגיע בסדר הנכון… למעשה, שימוש ב-TCP גורר יותר זמן ומשאבים מאשר שימוש בפרוטוקול שאינו מבוסס קישור כגון UDP. לעיתים, העברה מהירה של המידע תהיה חשובה לנו הרבה יותר מאשר העברה אמינה של המידע.

בואו נבחן יחד את המקרים הבאים:

מקרה מבחן: תוכנה להעברת קבצים גדולים

מה דעתכם – האם בתוכנה להעברת קבצים גדולים בין מחשבים נעדיף להשתמש ב-UDP או ב-TCP?
התשובה במקרה זה היא TCP. כמו שאמרנו קודם, במקרה של העברת קובץ – נרצה שכל המידע על הקובץ יגיע, ושיגיע בסדר הנכון. אחרת, ייתכן שלא נוכל לפתוח את הקובץ בכלל. במקרה זה נעדיף "לשלם" את המחיר של הרמת וסגירת קישור, ווידוא הגעת המידע וכל ה-Overhead המשתמע משימוש ב-TCP – על מנת שהמידע יגיע באופן אמין.

מקרה מבחן: פרוטוקול DNS

היזכרו בפרוטוקול DNS עליו למדנו ב[פרק שכבת האפליקציה](). מה דעתכם – האם בשימוש ב-DNS נעדיף להשתמש ב-UDP או ב-TCP?
התשובה במקרה זה היא UDP[31]. הסיבה לכך היא ש-DNS הוא פרוטוקול מסוג שאילתה-תשובה. הלקוח שולח לשרת שאלה (למשל: "מי זה www.google.com?"), שזו חבילה אחת בלבד, ומקבל עליה התשובה. באם לא הגיעה תשובה, הלקוח יכול לשלוח את השאילתה שוב. במקרה זה, לא משתלם להרים קישור TCP שלם.

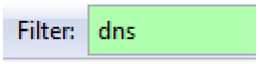
---

[31] יש גם מימושים של DNS מעל TCP, אך השימוש הנרחב הוא מעל פרוטוקול UDP.

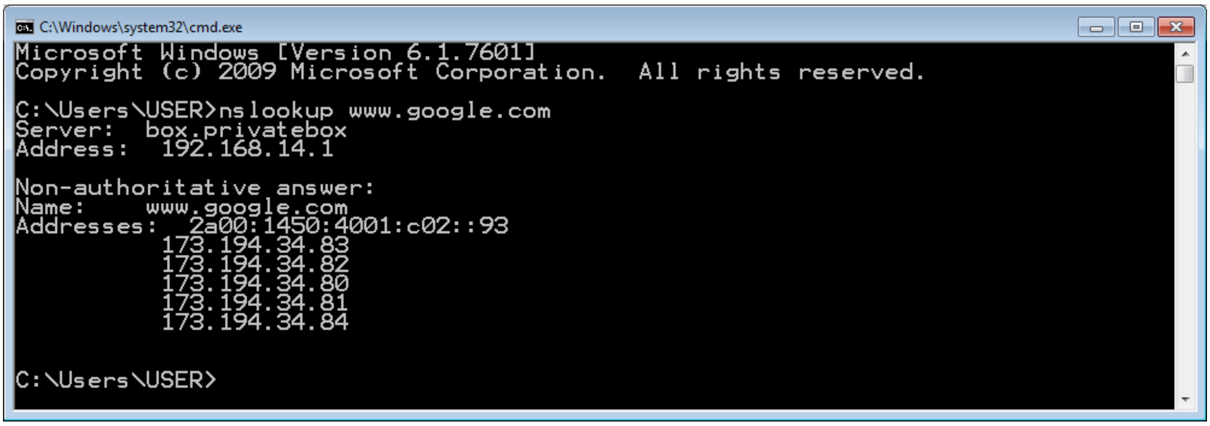
תרגיל 6.2 מודרך – מעל איזה פרוטוקול שכבת התעבורה עובר DNS?

ננסה לאמת את ההנחה שלנו – האם באמת פרוטוקול DNS עובר מעל UDP?

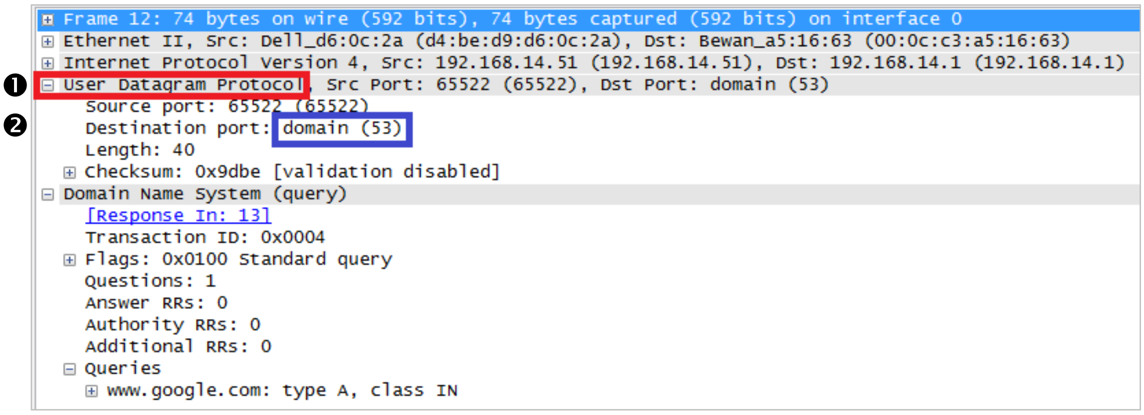
הריצו את Wireshark ופתחו הסנפה. השתמשו במסנן "dns":



כעת, פתחו את ה-Command Line והשתמשו בכלי **nslookup** בכדי לשלוח שאילתה על הדומיין www.google.com:



מצאו את השאילתה הרלוונטית והסתכלו על החבילה:



כפי שניתן לראות, הפרוטוקול בו מתבצע שימוש הוא פרוטוקול UDP (מסומן ב**אדום**, 1). פורט היעד אליו מתבצעת הפנייה הינו פורט 53 (מסומן ב**כחול**, 2), והוא הפורט המשויך לפרוטוקול DNS.

מקרה מבחן: תוכנה לשיתוף תמונות

מה דעתכם – האם בתוכנה להעברת תמונות רבות בין מחשבים נעדיף להשתמש ב-UDP או ב-TCP?

גם במקרה זה האמינות חשובה לנו יותר מהמהירות, ולכן נשתמש ב-TCP. מכיוון שנרצה שכל התמונות יגיעו באופן תקין ונוכל לצפות בהן, עלינו לשלם את המחיר של שימוש בפרוטוקול מבוסס קישור.

מקרה מבחן: Skype

מה דעתכם – האם בתוכנה לביצוע שיחות Voice over IP כגון Skype נעדיף להשתמש ב-UDP או ב-TCP?

במקרה זה בולט מאוד הצורך במהירות – אנו רוצים שהקול שלנו יגיע מצד לצד באופן כמה שיותר מהיר. גם כאן, אין הפסד גדול באם חלק מהחבילות הלכו לאיבוד בדרך. לכן במקרה זה נעדיף להשתמש בפרוטוקול UDP.

תרגיל 6.3 – מקרה מבחן: שרת HTTP

בפרק שכבת האפליקציה למדנו על פרוטוקול HTTP. חשבו בעצמכם – האם נעדיף במקרה של שרת HTTP להשתמש ב-UDP או ב-TCP?

ודאו את תשובתכם. השתמשו ב-Wireshark ובדפדפן בכדי לגלוש לשרת HTTP, ומצאו האם פרוטוקול שכבת התעבורה בו משתמש השרת הוא באמת הפרוטוקול בו חשבתם שהוא ישתמש.

שאלת חשיבה: מדוע צריך שכבת תעבורה לא אמינה מעל שכבת רשת לא אמינה?

נסו לחשוב על כך: אם שכבת הרשת שלנו אינה אמינה ולא מבטיחה העברה של מידע מצד לצד, מדוע להשתמש בכלל בשכבת תעבורה לא אמינה? מדוע להשתמש בפרוטוקול UDP ולא לשלוח חבילות ישר מעל שכבת הרשת?

לשימוש בפרוטוקול לא אמין של שכבת התעבורה (כדוגמת UDP) מעל שכבת רשת לא אמינה, יש שתי סיבות עיקריות. ראשית, השימוש בפורטים. כפי שהסברנו קודם לכן בפרק, השימוש בפורטים הוא הכרח בכדי לדעת לאיזו תוכנה אנו פונים בשרת המרוחק. UDP מאפשר לנו את השימוש בפורטים.

בנוסף על כן, שימוש של שכבת האפליקציה בשכבת הרשת "ישבור" את מודל השכבות: איננו רוצים שמתכנת של שכבת האפליקציה יכיר את שכבת הרשת. דבר זה יגרום למפתח של שכבת האפליקציה להעמיק בסוגיות הקשורות לשכבת הרשת, ולכתוב מימוש שונה עבור כל פרוטוקול בשכבה זו. מבחינת מפתח של שכבת האפליקציה, הוא צריך להכיר רק את שכבת התעבורה והשירותים שהיא נותנת לו. בכך, שכבת התעבורה "מעלימה" את שכבת הרשת משכבת האפליקציה, בין אם היא מספקת אמינות ובין אם לא.

UDP – User Datagram Protocol

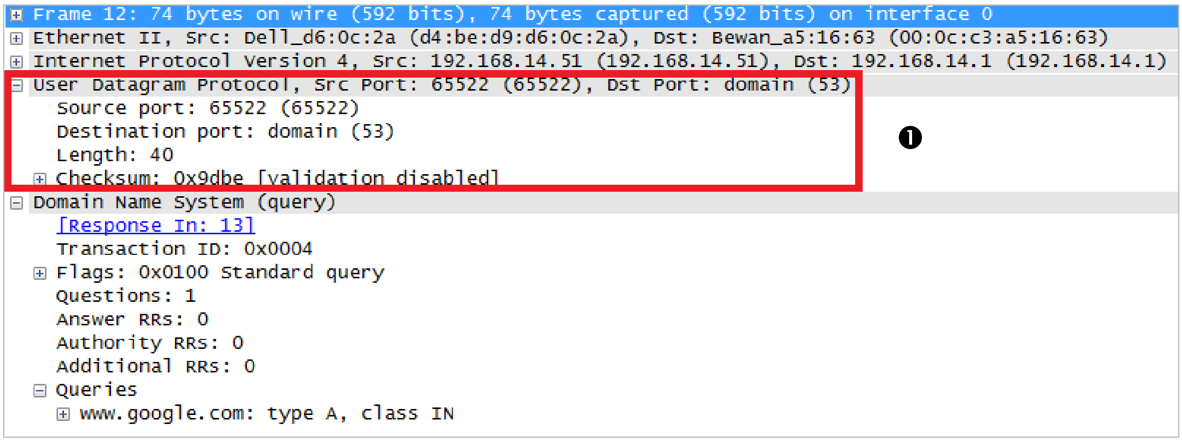
עכשיו כשהבנו לעומק את מטרותיה של שכבת התעבורה, כמו גם את ההבדלים בין פרוטוקולים מבוססי קישור לפרוטוקולים לא מבוססי קישור, הגיע הזמן להכיר את אחד הפרוטוקולים הנפוצים ביותר בשכבה זו – פרוטוקול UDP.

כאמור, פרוטוקול UDP אינו מבוסס קישור. כחלק מכך, UDP לא מבטיח הגעה של המידע כלל והגעה בסדר הנכון בפרט. הדבר דומה לשליחת מכתב בדואר רגיל (שאינו רשום): אם ברצוני לשלוח מכתב למישהו, עליי לשים אותו במעטפה ולשלשל אותו לתיבה. אין לי צורך להודיע אל הנמען שהוא צפוי לקבל ממני את ההודעה, וכן אין הבטחה של רשות הדואר שהמכתב יגיע מהר, או שיגיע בכלל. ייתכן שהמכתב יאבד בדרך.

תרגיל 6.4 מודרך – התבוננות בפרוטוקול UDP

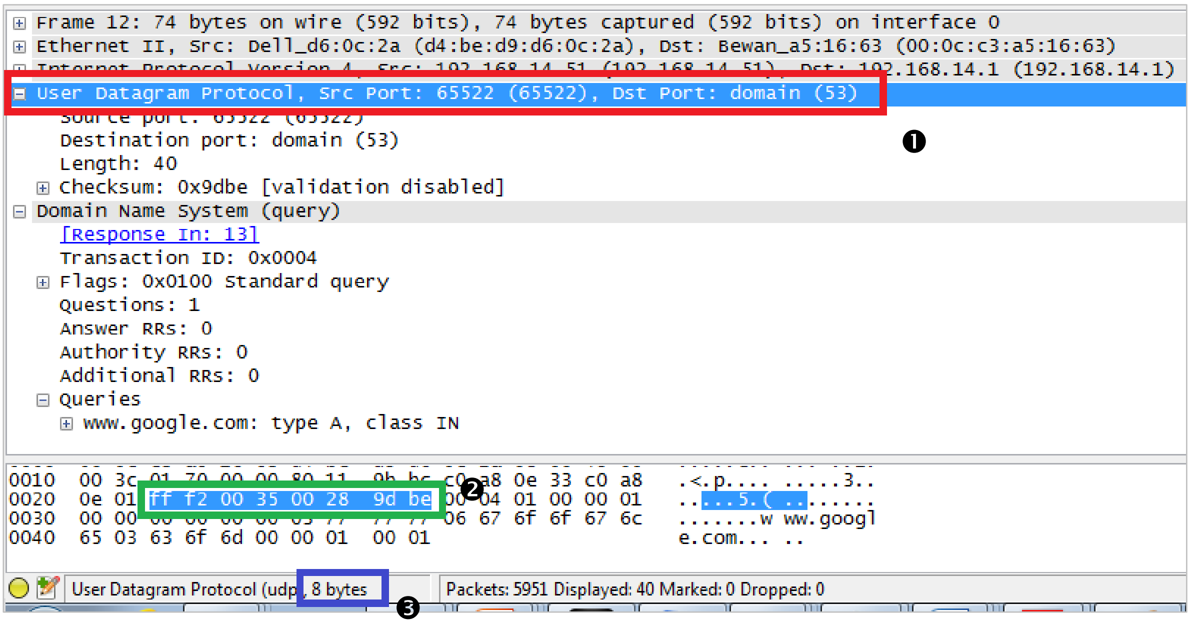
הריצו את Wireshark. הסניפו עם המסנן "udp". חכו עד שחבילות UDP תופענה על המסך. לחלופין, תוכלו לשלוח שאילתת DNS, שנשלחת מעל UDP כפי שלמדנו קודם לכן.

בחרו באחת החבילות. הסתכלו על ה-Header של החבילה:


```
⊞ Frame 12: 74 bytes on wire (592 bits), 74 bytes captured (592 bits) on interface 0
⊞ Ethernet II, Src: Dell_d6:0c:2a (d4:be:d9:d6:0c:2a), Dst: Bewan_a5:16:63 (00:0c:c3:a5:16:63)
⊞ Internet Protocol Version 4, Src: 192.168.14.51 (192.168.14.51), Dst: 192.168.14.1 (192.168.14.1)
⊟ User Datagram Protocol, Src Port: 65522 (65522), Dst Port: domain (53)
    Source port: 65522 (65522)
    Destination port: domain (53)
    Length: 40
  ⊞ Checksum: 0x9dbe [validation disabled]
⊟ Domain Name System (query)
    [Response In: 13]
    Transaction ID: 0x0004
  ⊞ Flags: 0x0100 Standard query
    Questions: 1
    Answer RRs: 0
    Authority RRs: 0
    Additional RRs: 0
  ⊟ Queries
    ⊞ www.google.com: type A, class IN
```
❶

מה גודל ה-Header של חבילת UDP?

בכדי לענות על שאלה זו, נשתמש בעזרתו של Wireshark, שיודע לספור עבורנו בתים. נלחץ עם העכבר על שורת ה-UDP (מסומן ב**אדום**, 1):



כעת Wireshark יסמן לנו את השורה גם היכן שלחצנו, וגם בתצוגה התחתונה שמראה את הבתים שנשלחו (מסומן בירוק, 2). בנוסף, הוא יכתוב לנו למטה את כמות הבתים שסימנו (מסומן בכחול, 3). מכאן שהגודל של Header של חבילת UDP הוא שמונה בתים.

נסו לענות בעצמכם על השאלה הבאה בטרם תמשיכו את הקריאה:

אילו שדות יש ב-Header של חבילת UDP? מה התפקיד של כל שדה?

שני השדות הראשונים קלים להבנה:

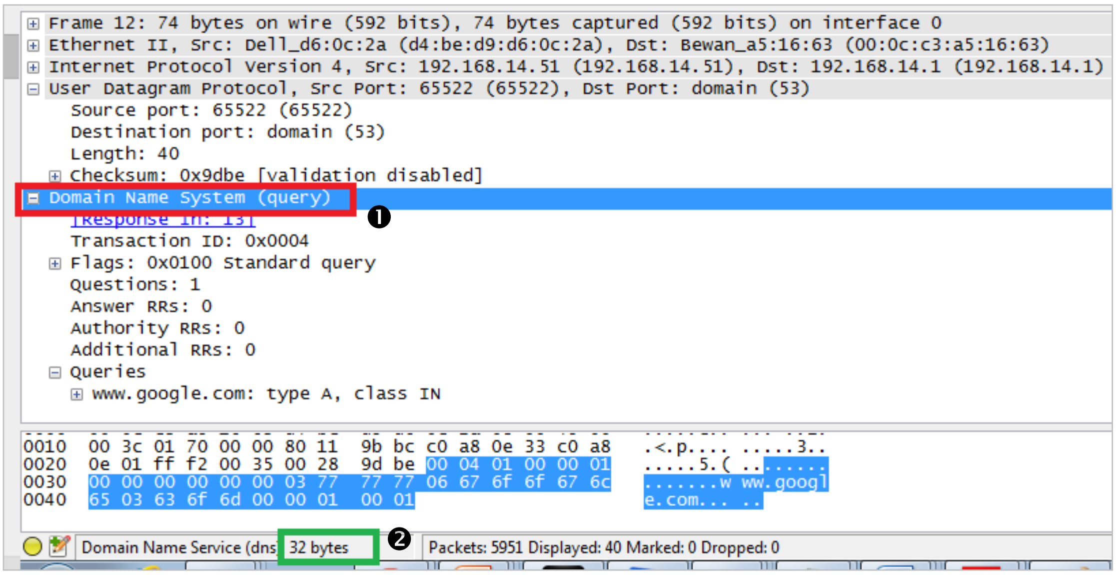
- Source Port (פורט מקור) – הפורט של התוכנה ששלחה את החבילה. במקרה זה, זהו הפורט של התוכנה ששלחה את שאילתת ה-DNS ומחכה לקבל תשובה. שרת ה-DNS צפוי להחזיר את התשובה שלו אל הפורט הזה.
- Destination Port (פורט יעד) – הפורט של התוכנה שצפויה לקבל את החבילה. במקרה זה, זהו הפורט של שירות ה-DNS.

השדה הבא הינו שדה האורך (Length).

מה מציין שדה האורך ב-UDP?

האם הוא מציין את אורך המידע של חבילת ה-UDP (במקרה זה, ה-DNS)? האם את אורך ה-Header? האם את האורך הכולל של ה-Header והמידע?

בדומה לדרך בה גילינו את אורך ה-Header של החבילה, נשתמש בספירת הבתים של Wireshark בכדי לגלות את גודל שכבת ה-DNS בחבילה זו:



לאחר שנלחץ על השורה של Domain Name System, היא תסומן בידי Wireshark (מסומן ב**אדום**, 1 בתמונה לעיל). כעת, Wireshark יראה לנו גם את גודל השכבה – 32 בתים (מסומן ב**ירוק**, 2).

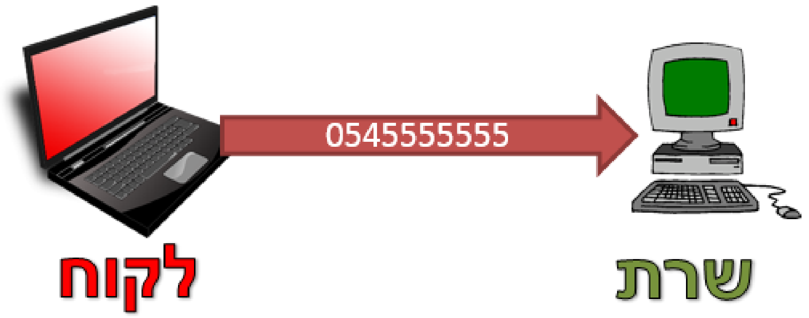
מכיון שגילינו קודם לכן שגודל ה-Header הוא 8 בתים, ועכשיו גילינו שגודל המידע במקרה הזה הוא 32 בתים, אנו לומדים ששדה האורך ב-Header של UDP מתאר את גודל ה-Header והמידע גם יחד.

בכדי להבין את משמעות השדה הבא, נצטרך לענות על השאלה:

מה זה Checksum?

עד כה ציינו שבעיות ברשת יכולות לגרום לחבילה לא להגיע כלל, או לרצף של חבילות להגיע ברצף הלא נכון. אך בעיות ברשת יכולות גם לגרום לשגיאות בחבילה עצמה – כלומר שהחבילה תגיע עם תוכן שונה מהתוכן שנשלח במקור.

לדוגמה, נביט בפרוטוקול שנועד לשלוח מספרי טלפון נייד ממחשב אחד למחשב אחר. בפרוטוקול זה, בכל חבילה, נשלחות 10 ספרות של מספר טלפון אחד. כך למשל, חבילה לדוגמה יכולה להיראות כך:



הבעיה היא, שייתכן שהחבילה השתנתה בדרך בגלל תקלה כלשהי. כך למשל, ייתכן שהשרת יקבל את החבילה בצורה הבאה:

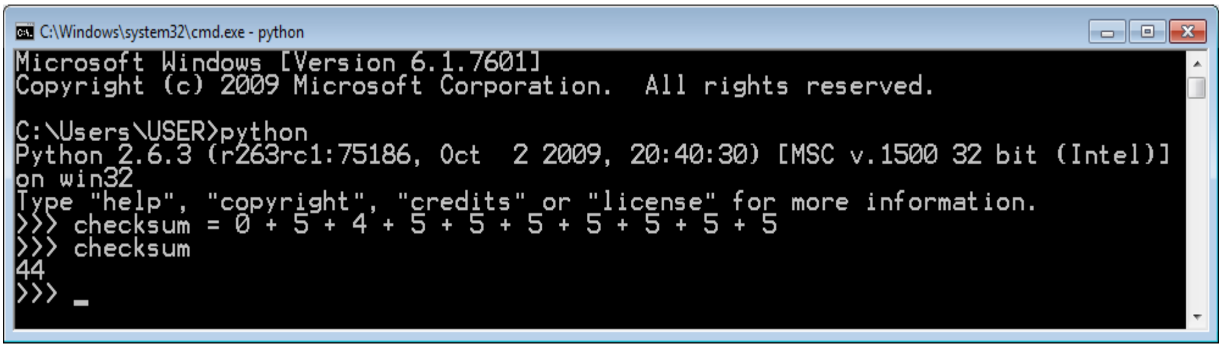
שימו לב, הספרה הראשונה השתנתה, ועכשיו היא כבר לא 0 אלא 6.

במקרה זה, נרצה שהשרת יידע שאירעה שגיאה, ולא יתייחס לחבילה התקולה. דרך אחת לעשות זאת, היא להשתמש ב-Checksum. הרעיון הוא כזה: נבצע פעולה כלשהי על המידע שאנו רוצים לשלוח ונשמור את התוצאה. בצד השני (במקרה זה, בצד השרת) החישוב יתבצע שוב, ויושווה לתוצאה שנשלחה. אם התוצאה שונה, הרי שיש בעיה.

נמשיך עם הדוגמה של מספר הטלפון הנייד. נאמר ובחרנו בפונקציית ה-Checksum הבאה: חיבור כל הספרות של מספר הטלפון. כלומר, עבור מספר הטלפון 054-5555555 שראינו קודם, יתבצע החישוב הבא:

0 + 5 + 4 + 5 + 5 + 5 + 5 + 5 + 5 + 5

נוכל לעשות זאת באמצעות פייתון ולראות את התוצאה:



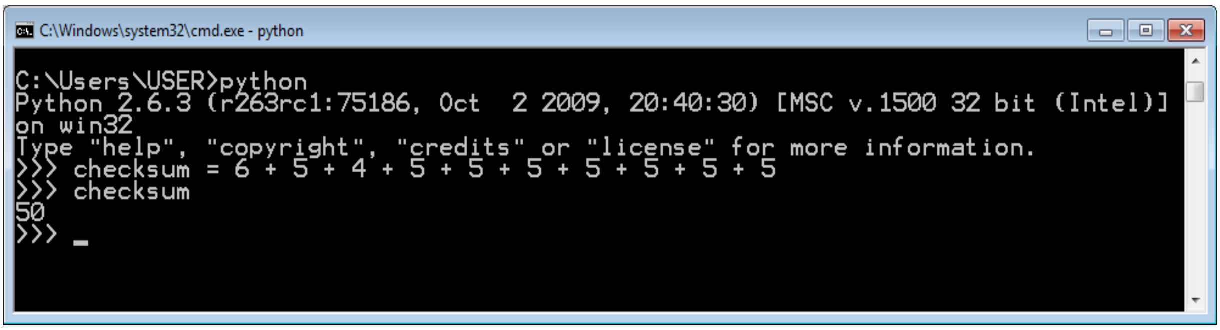
כעת, השולח ישלח לא רק את המידע שהוא רצה לשלוח (כלומר את מספר הטלפון), אלא גם את התוצאה של ה-Checksum. בדוגמה זו, תישלח החבילה הבאה:

עכשיו, במידה שתקרה אותה השגיאה שהתרחשה קודם לכן, השרת יקבל את ההודעה הבאה:

כעת השרת ינסה לבצע את החישוב של ה-Checksum על המידע עצמו:

6 + 5 + 4 + 5 + 5 + 5 + 5 + 5 + 5 + 5

שוב, נוכל להשתמש בפייתון:



```
C:\Windows\system32\cmd.exe - python
C:\Users\USER>python
Python 2.6.3 (r263rc1:75186, Oct  2 2009, 20:40:30) [MSC v.1500 32 bit (Intel)]
on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> checksum = 6 + 5 + 4 + 5 + 5 + 5 + 5 + 5 + 5 + 5
>>> checksum
50
>>> _
```

התוצאה יצאה 50, אך ה-Checksum שהלקוח שלח היה 44.[32] אי לכך, יש שגיאה בחבילה – והיא צריכה להיזרק.

[32] אלא אם כן, הייתה שגיאה בשדה ה-Checksum עצמו. גם במקרה זה, החבילה צפויה להיזרק.

באמצעות שדה נוסף בן 2 ספרות, הצלחנו לוודא שהמידע ששלחנו ב-10 הספרות הקודמות הגיע בצורה תקינה. עם זאת, הפונקציה שלנו אינה מושלמת, מן הסתם. אם, בדוגמה הקודמת, השרת היה מקבל את המספר 0635555555, התוצאה של ה-Checksum הייתה עדיין 44, וזהו לא המספר אותו הלקוח התכוון לשלוח. בספר זה לא נסביר את הפונקציה שבה משתמשים כדי לחשב את ה-Checksum בפרוטוקול UDP, אך חשוב שנבין את המשמעות של השדה הזה ושהוא נועד למציאת שגיאות.

אם אתם מעוניינים לראות דוגמה נוספת ל-Checksum, אתם מוזמנים לקרוא על ספרת ביקורת במספר הזהות בישראל, בכתובת: http://goo.gl/CvYtgt. לכל אזרח בישראל יש מספר זהות בעל תשע ספרות. למעשה, שמונה הספרות השמאליות הן מספר הזהות עצמו, והספרה הימנית ביותר היא ספרת הביקורת – תפקידה לוודא שאין שגיאה בכתיבה של שמונה הספרות שלפניה.
אגב, אין חובה להשתמש ב-Checksum בפרוטוקול UDP. אם הלקוח לא מעוניין להשתמש ב-Checksum, ניתן לשלוח 0 בשדה של ה-Checksum.

נסכם את מה שלמדנו על שדות ה-Header של UDP:

- Source Port (פורט מקור) – הפורט של התוכנה ששלחה את החבילה.
- Destination Port (פורט יעד) – הפורט של התוכנה שצפויה לקבל את החבילה.
- Length (אורך) – אורך החבילה (כולל Header ומידע).
- Checksum – חישוב כדי לוודא שהחבילה הגיעה באופן תקין.

Socket של UDP

עכשיו שלמדנו על פרוטוקול UDP, הגיע הזמן להשתמש בקוד ששולח הודעות UDP.

תרגיל 6.5 מודרך – לקוח UDP ראשון

כעת נכתוב את לקוח ה-UDP הראשון שלנו. הלקוח יהיה דומה מאוד ללקוח הראשון שכתבנו בפרק תכנות ב- Sockets /תרגיל 2.1 מודרך – הלקוח הראשון שלי, אלא שהוא יהיה מבוסס על פרוטוקול UDP ולא TCP כפי שעשינו לפני כן.

הדבר הראשון שעלינו לעשות הוא לייבא את המודול של **socket** לפייתון:

```
import socket
```

כעת, עלינו ליצור אובייקט מסוג **socket**. נקרא לאובייקט זה בשם my_socket:

```
my_socket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
```

כאן אנו נתקלים בהבדל הראשון בין socket מבוסס TCP ל-socket מבוסס UDP. להזכירכם, כאשר כתבנו את הלקוח מבוסס TCP, השתמשנו בשורה הבאה:

```
my_socket = socket.socket()
```

מכיוון שלא סיפקנו פרמטרים ל-()socket, פייתון הניח שאנו משתמשים בפרמטרים ברירת המחדל, שהם:

```
my_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
```

נתעלם כרגע מהפרמטר הראשון (socket.AF_INET) ונתרכז בפרמטר השני, שיכול לקבל בין השאר את הערכים הבאים:

- SOCK_STREAM – הכוונה היא לשימוש בחיבור מבוסס קישור. בפועל, השימוש הוא בפרוטוקול TCP.
- SOCK_DGRAM – הכוונה היא לשימוש שחיבור שאינו מבוסס קישור. בפועל, השימוש הוא בפרוטוקול UDP.

מכאן שהלקוח שכתבנו בעבר השתמש, מבלי שציינו זאת באופן מראש, בפרוטוקול TCP. כעת, מכיוון שאנו מציינים את הפרמטר socket.SOCK_DGRAM, הוא ישתמש בפרוטוקול UDP:

```
my_socket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
```

להזכירכם, כאשר כתבנו את הלקוח הקודם שלנו, השתמשנו בשלב זה במתודה **connect**. מתודה זה יצרה חיבור TCP אמין וקבוע בין הלקוח והשרת. מכיוון שאנו כותבים באמצעות UDP, אין צורך במתודה זו, ונוכל ישר לשלוח את המידע שלנו באמצעות המתודה **sendto**:

```
my_socket.sendto(‘Omer’.encode(), (‘127.0.0.1, 8821))
```

כפי שניתן לראות, המתודה **sendto** קיבלה את המידע שברצוננו לשלוח ('Omer') וכן את ה-tuple שמתאר את התוכנה המרוחקת ומכיל כתובת IP ומספר פורט. בשורה זו שלחנו את המחרוזת 'Omer' אל התוכנה שמאזינה לפורט UDP 8821 ב-Local Host (127.0.0.1).

על מנת לקבל מידע, עלינו להשתמש במתודה **recvfrom**. שימו לב, שמכיוון שלא נוצר קישור בינינו לבין השרת המרוחק, ייתכן גם שנקבל מידע מישות אחרת. לכן, **recvfrom** גם מאפשרת לנו לדעת ממי קיבלנו את המידע שקיבלנו:

```
(data, remote_address) = my_socket.recvfrom(1024)
```

נוכל כמובן להדפיס את המידע שקיבלנו:

```
print('The server sent: ' + data.decode())
```

ולבסוף, "נסגור" את אובייקט ה-**socket** שיצרנו בכדי לחסוך במשאבים:

```
my_socket.close()
```

להלן כלל הקוד שכתבנו:

```python
import socket

my_socket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
my_socket.sendto('Omer'.encode(), ('127.0.0.1', 8821))
(data, remote_address) = my_socket.recvfrom(1024)
print('The server sent: ' + data.decode())
my_socket.close()
```

תרגיל 6.6 – הרצת הלקוח מול שרת מוכן

כעת תחברו את הלקוח לשרת. בתרגיל זה השרת כבר מומש עבורכם. השרת משכפל כל מידע שתשלחו לו, ושולח אותו אליכם בחזרה, כמו הד, בתוספת "Hello". כך למשל, אם תכתבו אל השרת את המידע: "Omer" (שימו לב – הכוונה היא למחרוזת), הוא יענה: "Hello, Omer":

הורידו את השרת מהכתובת:

https://data.cyber.org.il/networks/udp_server.pyc

שמרו את הקובץ למיקום הבא:

C:\networks\work\udp_server.pyc

על מנת להריץ את השרת, היכנסו אל ה-Command Line, והריצו את שורת הפקודה:

python C:\networks\work\udp_server.pyc

השרת מאזין על הפורט 8821.

תרגיל 6.7 – השוואת זמנים בשרת הדים

כעת, נשדרג את הלקוח. עליכם לחשב כמה זמן לקח מאז ששלחתם את ההודעה אל השרת, ועד שהתקבלה תשובה (רמז: השתמשו במודול **time** של Python). הדפיסו למסך את הזמן הזה.

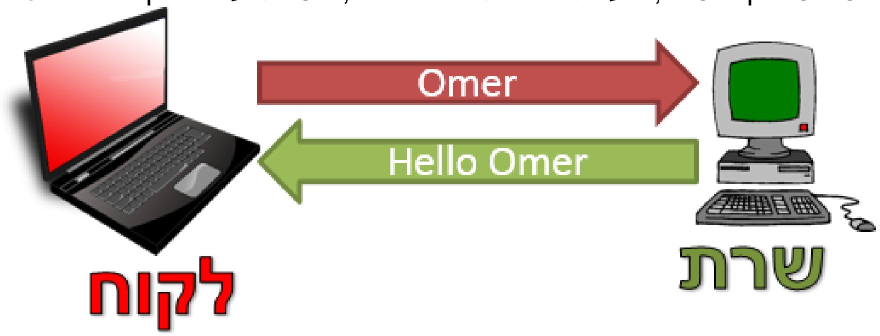
לאחר מכן, השתמשו בקוד שכתבתם בפרק תכנות ב- Sockets /לקוח לשרת הדים, בו השתמשנו, כזכור, ב- TCP. הוסיפו גם ללקוח זה את היכולת למדוד זמן מהרגע שבו נשלחה ההודעה אל השרת, לבין התשובה.

כעת, הריצו את הלקוחות ובדקו את הזמנים. האם יש הפרש בין הזמן שלקח לתשובה להגיע במימוש ה-UDP לבין הזמן שלקח לתשובה להגיע במימוש ה-TCP?

שימו לב: עליכם להריץ את הלקוחות אל מול שרת שנמצא במחשב מרוחק, ולא אל מול שרת שנמצא במחשב שלכם.

תרגיל 6.8 מודרך – שרת UDP ראשון

מוקדם יותר, יצרנו לקוח ששולח לשרת את שמו, לדוגמה: "Omer". כעת, נגרום לשרת לקבל את השם שהלקוח שלח, ולענות לו בהתאם. לדוגמה, השרת יענה במקרה זה: "Hello, Omer":



גם ב-UDP, הדרך לכתיבת שרת דומה מאוד לכתיבה של לקוח. גם הפעם, הדבר הראשון שעלינו לעשות הוא לייבא את המודול של **socket** לפייתון:

```
import socket
```

כעת, עלינו ליצור אובייקט מסוג **socket**. שוב, עלינו להגדיר שמדובר בחיבור UDP. נקרא לאובייקט זה בשם server_socket:

```
server_socket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
```

בשלב הבא, עלינו לבצע קישור של אובייקט ה-**socket** שיצרנו לכתובת מקומית. לשם כך נשתמש במתודה **bind**. המתודה זהה למקרה של שימוש ב-TCP. נשתמש בה, לדוגמה, כך:

```
server_socket.bind((‘0.0.0.0’, 8821))
```

בצורה זו יצרנו קישור בין כל מי שמנסה להתחבר אל הרכיב שלנו לפורט מספר 8821 – אל האובייקט server_socket.

הפעם, בניגוד לשרת ה-TCP שמימשנו בעבר, אין צורך להשתמש במתודה **listen**, וגם לא במתודה **accept**. למעשה, אנו מוכנים לקבל מידע:

```
(client_name, client_address) = server_socket.recvfrom(1024)
```

מכיוון שלא הקמנו קישור, המתודה **recvfrom** מחזירה לנו לא רק את המידע שהלקוח שלח (אותו שמרנו אל המשתנה client_name), אלא גם את הכתובת של הלקוח (אותו שמרנו במשתנה client_address). כתובת זו תשמש אותנו כשנרצה לשלוח מידע חזרה אל הלקוח:

```
data = client_name.decode()
response = "Hello " + data
server_socket.sendto(response.encode(), client_address)
```

המימוש זהה למעשה לקבלת ושליחת מידע בצד הלקוח, ומשתמש במתודות **sendto** ו-**recvfrom** אשר פגשנו קודם לכן. שימו לב שבניגוד לתקשורת TCP, לא נוצר לנו אובייקט **socket** חדש עבור כל לקוח, שכן לא הרמנו קישור עם הלקוח.

כעת נוכל לסגור את אובייקט ה-**socket**:

```
server_socket.close()
```

להלן כלל הקוד של השרת שיצרנו:

```python
import socket

server_socket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
server_socket.bind(("0.0.0.0", 8821))
(client_name, client_address) = server_socket.recvfrom(1024)
data = client_name.decode()
response = "Hello " + data
server_socket.sendto(response.encode(), client_address)

server_socket.close()
```

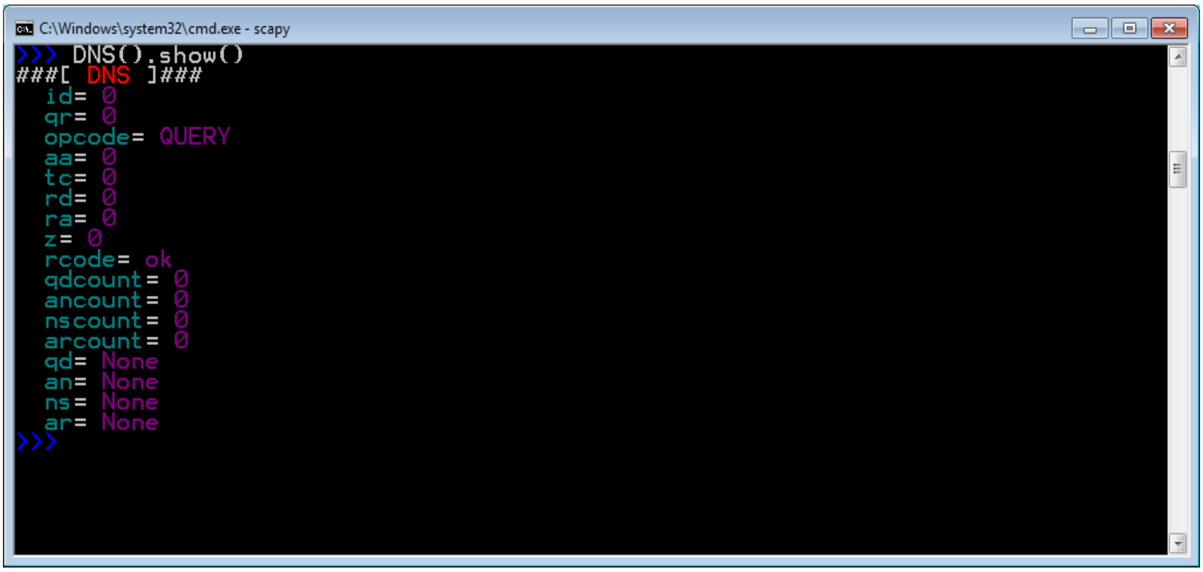
UDP ב-Scapy

כעת נלמד כיצד לשלוח חבילות UDP באמצעות Scapy.

תרגיל 6.10 מודרך – שליחת שאילתת DNS באמצעות Scapy

בתרגיל זה נשלח בעצמנו שאלת DNS באמצעות Scapy. ראשית, פתחו את Scapy. כעת, נתחיל מלבנות חבילה של DNS. נסתכל על מבנה החבילה:

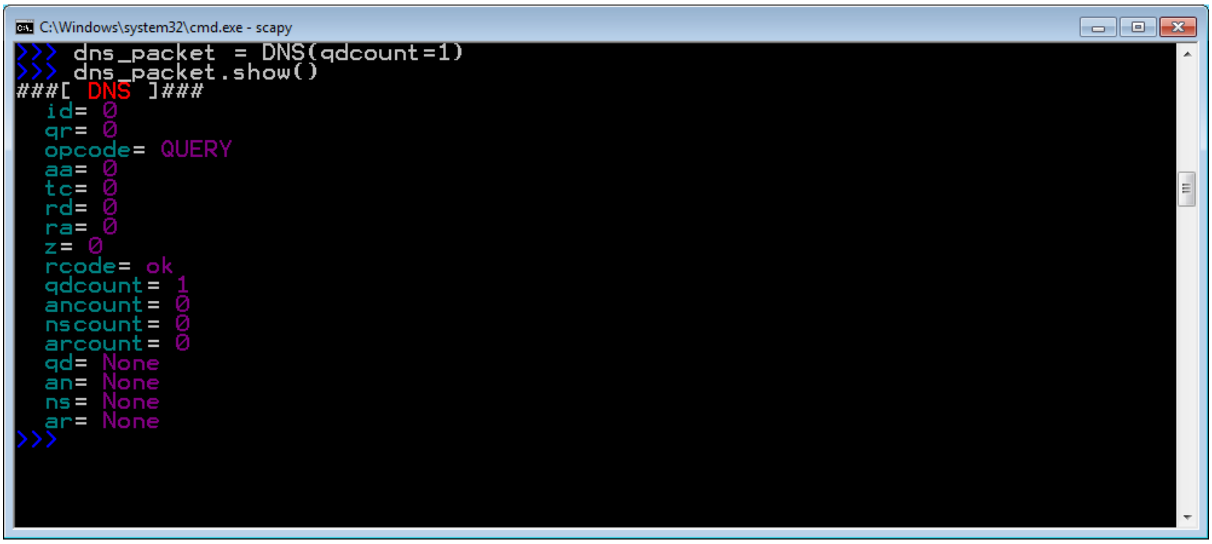
>>> DNS().show()



על מנת לבנות חבילת שאילתה ב-DNS, עלינו לציין כמה שאילתות אנו שולחים. שדה זה נקרא 'qdcount'. לכן, נייצר את חבילת ה-DNS כאשר בשדה זה ישנו הערך 1, המציין שאנו שולחים שאילתה אחת:

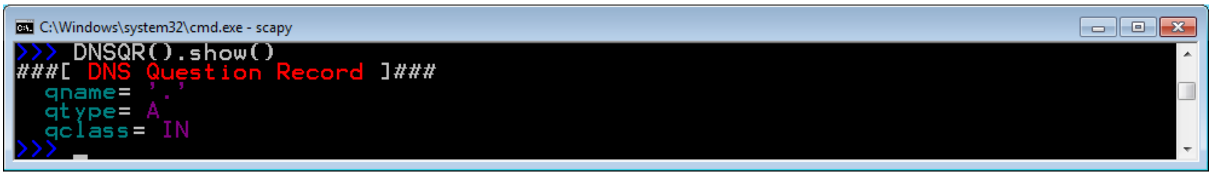
>>> dns_packet = DNS(qdcount = 1)

>>> dns_packet.show()



כעת עלינו לבנות את השאילתה. ראשית נסתכל על הדרך שבה Scapy מציג שאילתה:

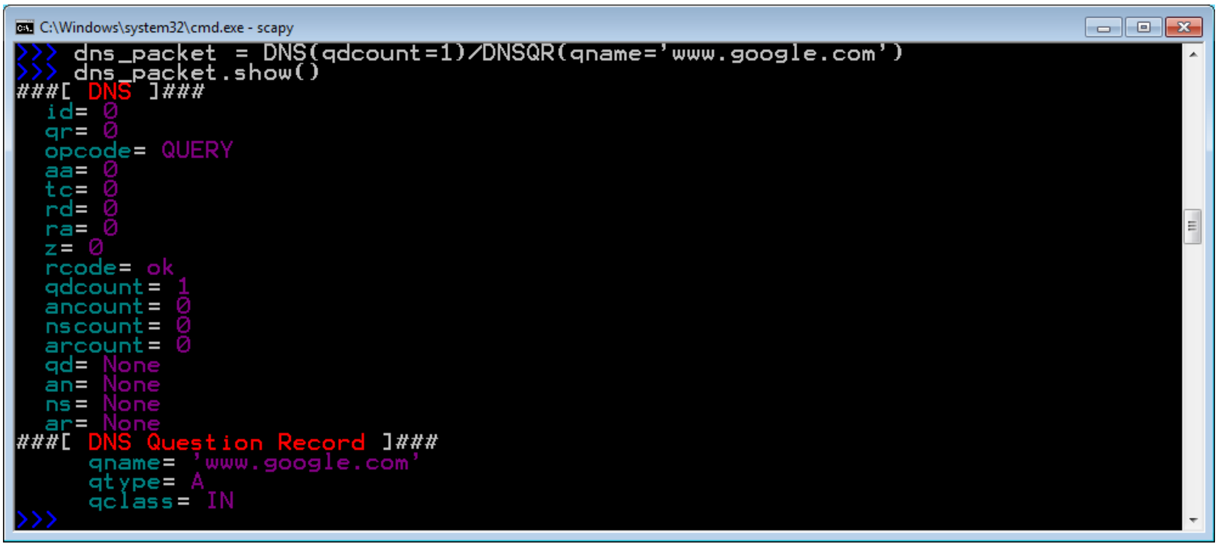
>>> DNSQR().show()



כפי שניתן לראות, Scapy מניח בעצמו שהשאילתה היא מסוג A, כלומר מיפוי של שם דומיין לכתובת IP. מכאן שעלינו לשנות רק את שם הדומיין, שהוא בשדה qname:

>>> dns_packet = DNS(qdcount=1)/DNSQR(qname='www.google.com')

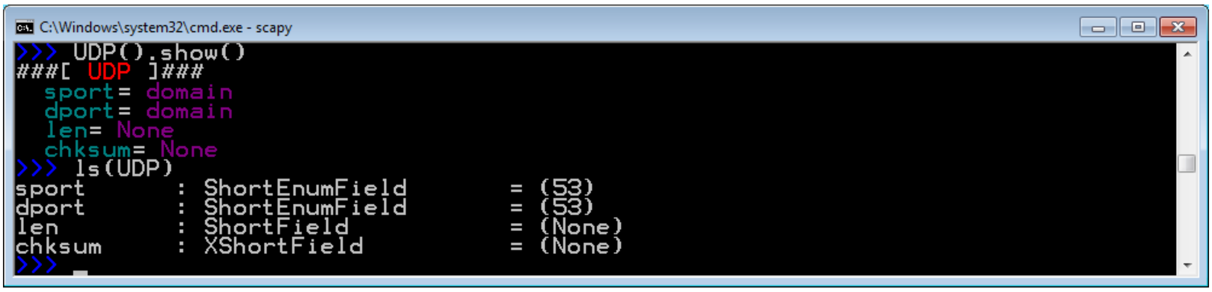
>>> dns_packet.show()



כעת ברשותנו חבילה שמורכבת משכבת DNS בלבד. על מנת לשלוח אותה, נצטרך להרכיב גם את השכבות התחתונות. נתחיל מלהרכיב את שכבת ה-UDP. על מנת לעשות זאת, נבחר להשתמש ב-53 כפורט יעד (מכיוון שזה הפורט המשויך ל-DNS), ונבחר בפורט מקור כרצוננו, לדוגמה: 24601. ראשית, נסתכל על הדרך בה Scapy קורא לשדות השונים של UDP. נוכל לעשות זאת באמצעות המתודה **show** על חבילת UDP כלשהי, או באמצעות הפקודה **ls**:

>>> UDP.show()

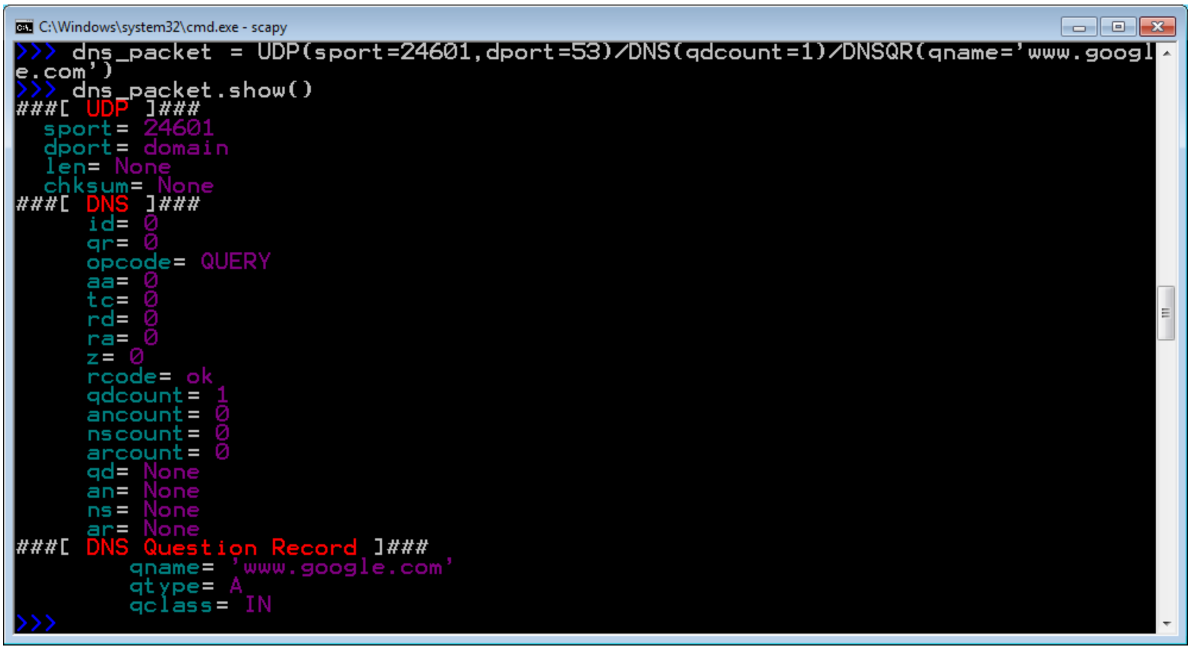
>>> ls(UDP)



עכשיו ניצור את החבילה:

>>> dns_packet = UDP(sport=24601, dport=53)/DNS(qdcount=1)/DNSQR(qname='www.google.com')

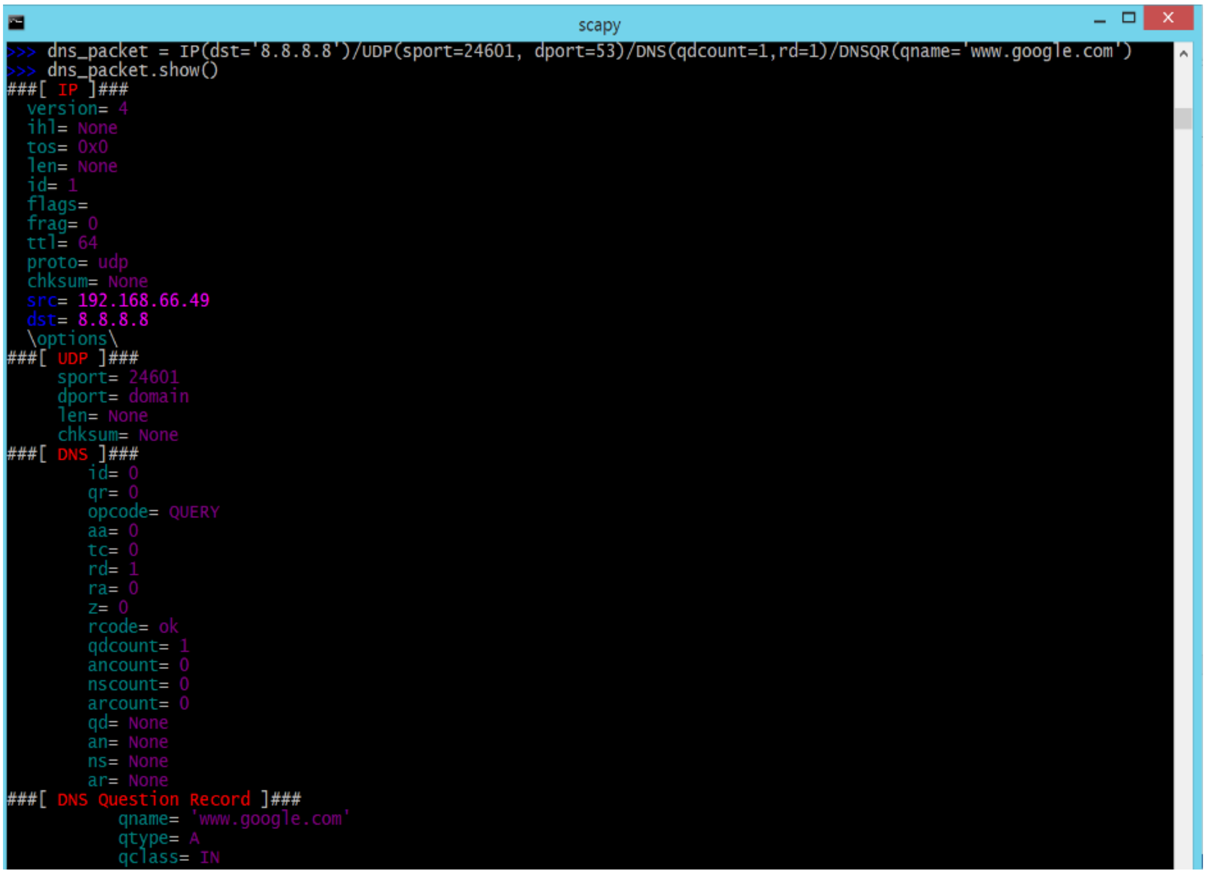
>>> dns_packet.show()



שימו לב – לא הצבנו אף ערך בשדות האורך (שנקרא על ידי Scapy בשם **len**) וה-Checksum (שנקרא על ידי Scapy בשם **chksum**). אל דאגה, ערכים אלו יתמלאו באופן אוטומטי כאשר החבילה תישלח!

כעת עלינו להחליט לאיזו כתובת IP לשלוח את החבילה. לצורך התרגיל, נשלח לכתובת "8.8.8.8", שמשמשת שרת DNS באינטרנט. נבנה את החבילה המלאה:

```
>>> dns_packet = IP(dst='8.8.8.8')/UDP(sport=24601,
                    dport=53)/DNS(qdcount=1,rd=1)/DNSQR(qname='www.google.com')
>>> dns_packet.show()
```

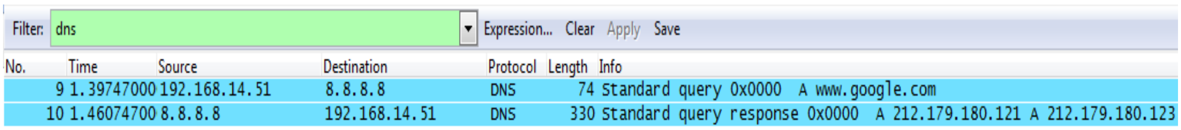


```
>>> dns_packet = IP(dst='8.8.8.8')/UDP(sport=24601, dport=53)/DNS(qdcount=1,rd=1)/DNSQR(qname='www.google.com')
>>> dns_packet.show()
###[ IP ]###
  version= 4
  ihl= None
  tos= 0x0
  len= None
  id= 1
  flags=
  frag= 0
  ttl= 64
  proto= udp
  chksum= None
  src= 192.168.66.49
  dst= 8.8.8.8
  \options\
###[ UDP ]###
     sport= 24601
     dport= domain
     len= None
     chksum= None
###[ DNS ]###
        id= 0
        qr= 0
        opcode= QUERY
        aa= 0
        tc= 0
        rd= 1
        ra= 0
        z= 0
        rcode= ok
        qdcount= 1
        ancount= 0
        nscount= 0
        arcount= 0
        qd= None
        an= None
        ns= None
        ar= None
###[ DNS Question Record ]###
           qname= 'www.google.com'
           qtype= A
           qclass= IN
```

בטרם נשלח את החבילה, פתחו את Wireshark והריצו הסנפה עם המסנן dns. כעת, שלחו את החבילה:

>>> send(dns_packet)

אתם צפויים לראות את חבילת השאילתה, כמו גם התשובה שהגיעה מהשרת:



Filter: dns — Expression... Clear Apply Save

No.	Time	Source	Destination	Protocol	Length	Info
9	1.39747000	192.168.14.51	8.8.8.8	DNS	74	Standard query 0x0000 A www.google.com
10	1.46074700	8.8.8.8	192.168.14.51	DNS	330	Standard query response 0x0000 A 212.179.180.121 A 212.179.180.123

תרגיל 6.11 מודרך – קבלת תשובה לשאילתת DNS באמצעות Scapy

אז הצלחנו לשלוח שאילתה של DNS לשרת המרוחק, וגם ראינו ב-Wireshark שההודעה נשלחה כמו שצריך ואף התקבלה תשובה. אך עכשיו נרצה להצליח לקבל את התשובה באמצעות Scapy. יש מספר דרכים לעשות זאת, ובשלב זה נלמד דרך אחת שהיא שימוש בפונקציה **sr1** המשמשת לשליחת חבילה אחת וקבלת תשובה עליה.

השתמשו באותה חבילת השאילתה שיצרנו קודם לכן, ושלחו אותה. אך הפעם, במקום להשתמש ב-**send**, השתמשו בפונקציה **sr1**, ושמרו את ערך החזרה שלה. פונקציה זו תשלח את החבילה, ואז תסניף את הרשת (כמו הפקודה **sniff**), ותשמור את התשובה לחבילה שנשלחה. עשו זאת כך:

>>> response_packet = sr1(dns_packet)

```
C:\Windows\system32\cmd.exe - scapy
>>> response_packet = sr1(dns_packet)
Begin emission:
Finished to send 1 packets.
...*
Received 4 packets, got 1 answers, remaining 0 packets
>>> _
```

שימו לב לשורות התחתונות. Scapy הציג, תוך כדי ריצה, שלוש נקודות (.) ואז כוכבית (*). כל נקודה כזו היא פקטה ש-Scapy הסניף שלא הייתה קשורה לפקטה ששלחנו (כלומר לא תשובה לשאילתת ה-DNS ששלחנו קודם לכן). הכוכבית היא פקטה שכן קשורה (כלומר פקטת התשובה לשאילתה ששלחנו). לסיום Scapy מסכם ואומר זאת במילים – "קיבלתי 4 פקטות, 1 מהן הייתה פקטת תשובה. יש 0 פקטות שעדיין מחכות לתשובה". Scapy מציין שיש 0 פקטות שמחכות לתשובה מכיוון שהפעם שלחנו חבילת שאלה אחת בלבד. יש פונקציות אחרות המאפשרות לשלוח יותר משאלה אחת בכל פעם.

כעת, נוכל להסתכל על התשובה של שרת ה-DNS:

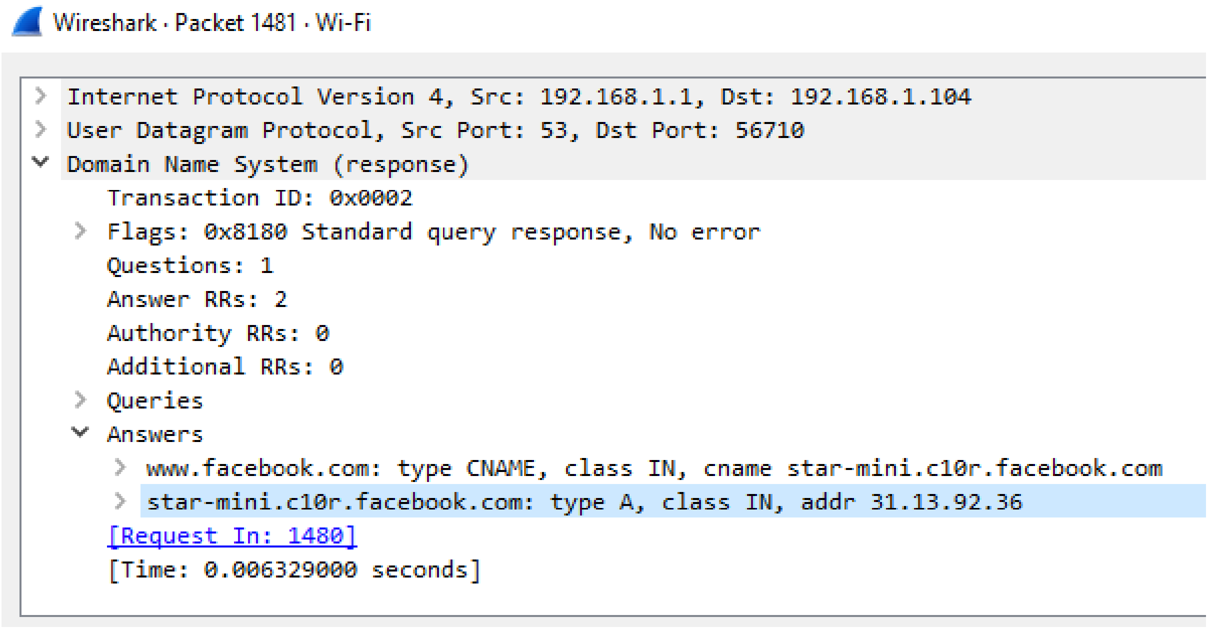
>>> response_packet.show()

```
C:\Windows\system32\cmd.exe - scapy
>>> response_packet.show()
###[ IP ]###
  version= 4L
  ihl= 5L
  tos= 0xb8
  len= 316
  id= 23035
  flags=
  frag= 0L
  ttl= 48
  proto= udp
  chksum= 0x5013
  src= 8.8.8.8
  dst= 192.168.14.51
  \options\
###[ UDP ]###
     sport= domain
     dport= 24601
     len= 296
     chksum= 0xb38
###[ DNS ]###
        id= 0
        qr= 1L
        opcode= QUERY
        aa= 0L
        tc= 0L
        rd= 0L
        ra= 1L
        z= 0L
        rcode= ok
        qdcount= 1
        ancount= 16
        nscount= 0
        arcount= 0
        \qd\
         |###[ DNS Question Record ]###
         |  qname= 'www.google.com.'
         |  qtype= A
         |  qclass= IN
        \an\
         |###[ DNS Resource Record ]###
         |  rrname= 'www.google.com.'
         |  type= A
         |  rclass= IN
         |  ttl= 290
         |  rdlen= 4
         |  rdata= '212.179.180.95'
```

תרגיל 6.12 – תשאול שרת DNS באמצעות Scapy

עד כה יצרנו ביחד שאילתת DNS, שלחנו אותה אל השרת וקיבלנו את התשובה. כעת, כתבו סקריפט אשר מקבל מהמשתמש את הדומיין שעליו הוא רוצה לשאול, ומדפיס את כתובת ה-IP הרלוונטית. לדוגמה, אם המשתמש יזין את הכתובת "www.google.com", על הסקריפט להדפיס את כתובת ה-IP הרלוונטית (למשל – "212.179.180.95").

שימו לב: לעיתים תוחזר יותר מאשר תשובת DNS אחת. לדוגמה, תשאול של facebook מחזיר שתי תשובות:



התשובה הראשונה היא מסוג CNAME והיא אינה כוללת כתובת IP, אלא את השם הקנוני (המוכר ברשת הפנימית) של www.facebook.com. התעלמו מתשובות מסוג CNAME. התשובה השניה, מסוג A, היא שכוללת את כתובת ה-IP המבוקשת.

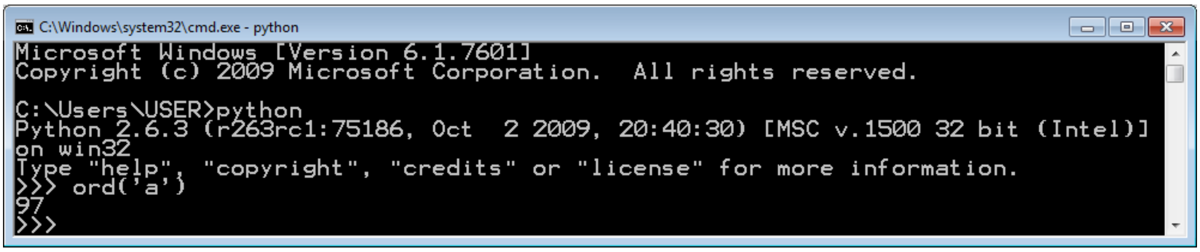
טיפ: בידקו כיצד נקרא השדה שמחזיק את התשובות. השדה הזה הוא למעשה רשימה, ניתן להגיע לכל איבר ברשימה באמצעות אינדקס, בדיוק באותו אופן שבו ניגשים לאיבר של רשימה בפייתון.

תרגיל 6.13 – תקשורת סודית מעל מספרי פורט

בתרגיל זה עליכם לסייע לשני תלמידים, יואב ומאור, לתקשר בצורה סודית מעל הרשת. מטרת התלמידים היא להצליח להעביר מסרים מאחד לשני, מבלי שאף אדם יוכל לקרוא אותם, גם אם הוא יכול להסניף את התעבורה ביניהם.

התלמידים החליטו על הפתרון הבא: על מנת להעביר אות ביניהם, הם ישלחו הודעה ריקה למספר פורט שמסמל אותה, כשהסימול הוא לפי קידוד ASCII (לקריאה נוספת – http://en.wikipedia.org/wiki/ASCII).

לדוגמה, נאמר שיואב רוצה לשלוח למאור את האות 'a'. לשם כך, עליו ראשית להבין מה הערך ה-ASCII שלה. בכדי לעשות זאת, הוא יכול להשתמש בפונקציה **ord** של פייתון:



```
C:\Windows\system32\cmd.exe - python
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation.  All rights reserved.

C:\Users\USER>python
Python 2.6.3 (r263rc1:75186, Oct  2 2009, 20:40:30) [MSC v.1500 32 bit (Intel)]
on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> ord('a')
97
>>>
```

כעת כשיואב גילה שערך ה-ASCII של האות 'a' הוא 97, הוא ישלח הודעת UDP ריקה לפורט 97 של מאור. במידה שיואב ירצה להעביר למאור את ההודעה "Hello", יהיה עליו לשלוח הודעה ריקה לפורט 72 (הערך של התו 'H'), לאחר מכן לשלוח הודעה לפורט 101 (הערך של התו 'e'), שתי הודעות ריקות לפורט 108 (הערך של התו 'l') ולבסוף הודעה ריקה לפורט 111 (הערך של התו 'o').

בתרגיל זה עליכם לממש את הסקריפטים בהם ישתמשו יואב ומאור בכדי להעביר מסרים זה לזה:

- כתבו סקריפט בשם secret_message_client.py. הסקריפט יבקש מהמשתמש להקליד הודעה, ולאחר מכן ישלח אותה אל השרת באופן סודי, כפי שתואר למעלה. את כתובת ה-IP של השרת אתם יכולים לכלול בקוד שלכם באופן קבוע ולא לבקש אותה מהמשתמש. השתמשו ב-Scapy בכדי לשלוח את החבילות.
- כתבו סקריפט בשם secret_message_server.py. הסקריפט ידפיס למסך מידע שהוא הבין כתוצאה משליחה של הסקריפט secret_message_client.py. השתמשו ב-Scapy בכדי להסניף ולקבל את החבילות.

שימו לב שעל מנת לבדוק תרגיל זה, עליכם להשתמש בשני מחשבים שונים – אחד ללקוח ואחד לשרת.

בונוס: כפי שלמדתם, פרוטוקול UDP אינו מבוסס קישור, ולכן ייתכן שחלק מהמידע ששלחתם מהלקוח לשרת לא יגיע, או לחלופין יגיע בסדר הלא נכון. חשבו כיצד ניתן להתגבר על בעיות אלו, וממשו פתרון אמין יותר.

TCP – Transmission Control Protocol

TCP הינו פרוטוקול שכבת התעבורה הנפוץ ביותר באינטרנט לחיבורים מבוססי קישור. כשאנו, בתור מפתחי שכבת האפליקציה, משתמשים ב-TCP בכדי להעביר מידע, איננו יכולים פשוט לשלוח חבילה אל תוכנה מרוחקת. ראשית עלינו ליצור קישור עם התוכנה המרוחקת, ועתה כל חבילה שנשלח תהיה חלק מאותו קישור. דבר זה דומה לשיחת טלפון: על מנת לדבר עם אדם אחר, איני יכול פשוט להגיד את ההודעה שלי (למשל: "ניפגש היום בשעה חמש ליד בית הספר"). עליי ראשית לחייג את המספר שלו, לשמוע צליל חיוג, ולחכות עד שירים את הטלפון ובכך יוקם בינינו קישור.

TCP תוכנן ועוצב לרוץ מעל שכבת רשת שאינה אמינה. כלומר, ההנחה הבסיסית היא שבשכבת הרשת חבילות יכולות ללכת לאיבוד או להגיע שלא בסדר הנכון. בתור פרוטוקול מבוסס קישור, TCP מבטיח לשכבת האפליקציה שהמידע יגיע אל היעד בסדר הנכון.

כיצד ניתן לוודא שהמידע מגיע אל היעד? כיצד ניתן לוודא שהוא מגיע בסדר הנכון?

על מנת לעשות זאת, TCP מנצל את העובדה שהוא פרוטוקול מבוסס קישור. מכיוון שכל החבילות (שנקראות בשכבת התעבורה בשם **סגמנטים**[33]) הן חלק מקישור, אנו יכולים לבצע דברים רבים.

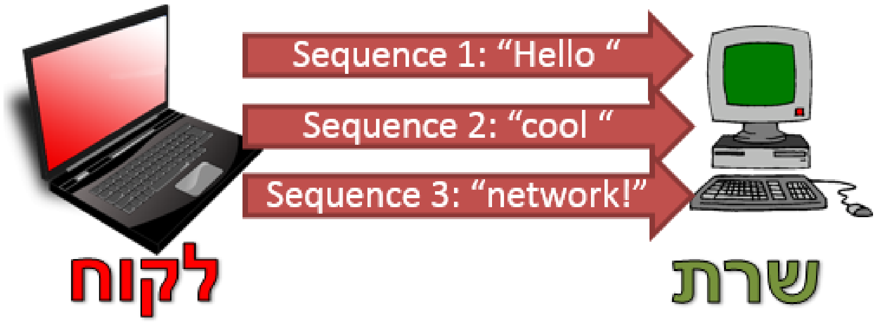
ראשית, אנו יכולים לתת מספר סידורי לחבילות שלנו. נאמר שבשכבת האפליקציה רצינו לשלוח את המידע "Hello cool network!". בשכבת התעבורה, נאמר שהמידע חולק לחבילות בצורה הבאה:

- חבילה מספר אחת – "Hello "
- חבילה מספר שתיים – "cool "
- חבילה מספר שלוש – "network!"

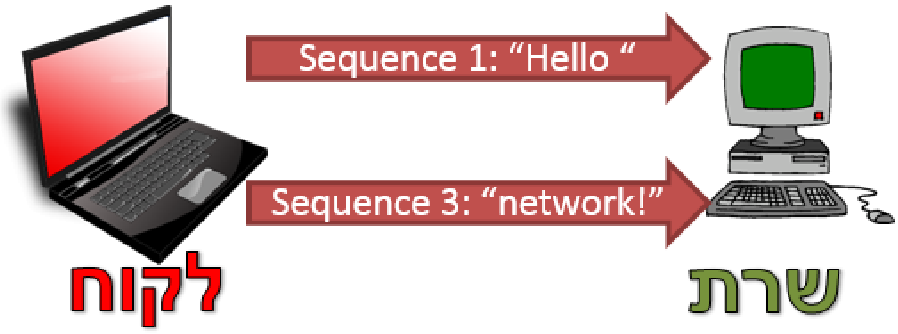
---

[33] גושי מידע בשכבת התעבורה נקראים "סגמנטים". עם זאת, כל סגמנט הוא למעשה גם חבילה של השכבה השלישית (שמכילה בתוכה את השכבה הרביעית, בהתאם למודל השכבות). על כן, ניתן לומר שכל סגמנט הוא גם פקטה (מונח זה שייך לשכבת הרשת, השכבה השלישית) וניתן לקרוא לו כך.

כעת נוכל לשלוח את החבילות כשלצידן יש מספר סידורי (Sequence Number):



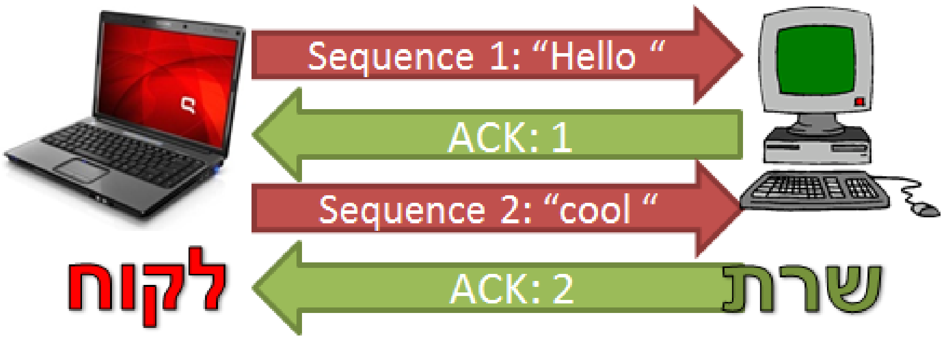
כעת, נסתכל על צד השרת. ניזכר כי ברשת ייתכן שחבילות מסוימות "נופלות" ולא מגיעות ליעדן. כך למשל, ייתכן שחבילה מספר שתיים "נפלה" בדרך, והשרת רואה מהלקוח רק שתי חבילות:



כעת, השרת יכול להבין שחסרה לו חבילה מספר שתיים! הוא יכול לעשות זאת מכיוון שהוא יודע שבחיבור הנוכחי בינו לבין הלקוח, הוא קיבל את חבילה מספר אחת וחבילה מספר שלוש, ולכן הוא אמור היה לקבל גם את חבילה מספר שתיים.

ניתן להשתמש במספרי החבילות בכדי לוודא שחבילה אכן הגיעה ליעדה. כך למשל, ניתן להחליט שעל כל חבילה שהגיעה, השרת שולח אישור ללקוח. חבילה כזו נקראת בדרך כלל **ACK** (קיצור של Acknowledgement), ומשמעותה – "קיבלתי את החבילה שלך".

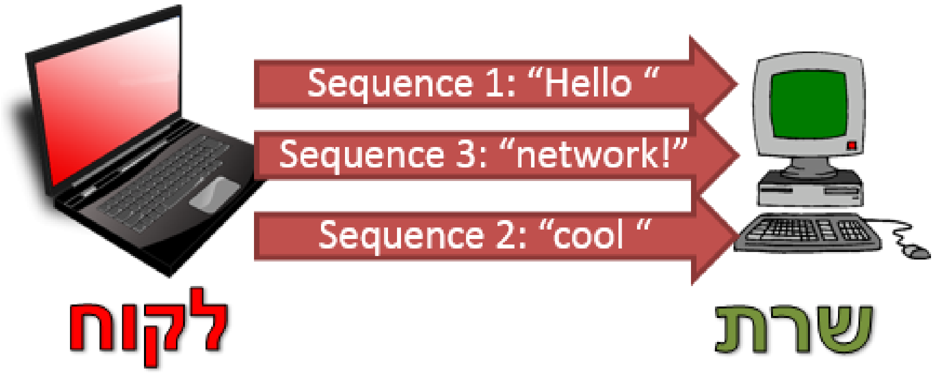
הלקוח יצפה לקבל ACK על כל חבילה אותה הוא שולח לשרת:



בצד הלקוח, אם לא התקבלה חבילת ACK מהשרת לאחר זמן מסוים, כנראה שהחבילה שהוא שלח "נפלה בדרך". במקרה כזה, החבילה תישלח שוב:

כך הצלחנו להבטיח שהחבילות ששלחנו באמת הגיעו ליעדן!

השימוש במספר סידורי לכל חבילה מאפשר לנו להתמודד עם בעיות נוספות. בגלל שהרשת לא אמינה, ייתכן שהחבילות יגיעו לשרת בסדר לא נכון:



במקרה זה, חבילה מספר שלוש הגיעה לפני חבילה מספר שתיים. עם זאת, מכיוון שהשרת רואה את המספר הסידורי של כל חבילה, הוא יכול לסדר אותן מחדש בסדר הנכון. שכבת האפליקציה לא תדע בכלל שהחבילות הגיעו במקור בסדר שונה מזה שאליו הלקוח התכוון.

השימוש ב-ACKים ובמספרים סידוריים מבטיח לנו אמינות: המידע ששלחנו יגיע, וגם יתקבל בסדר הנכון. לשם כך היינו צריכים להרים קישור, ולשלוח את החבילות כחלק מהקישור. בסיום הקישור, נרצה לסגור אותו.

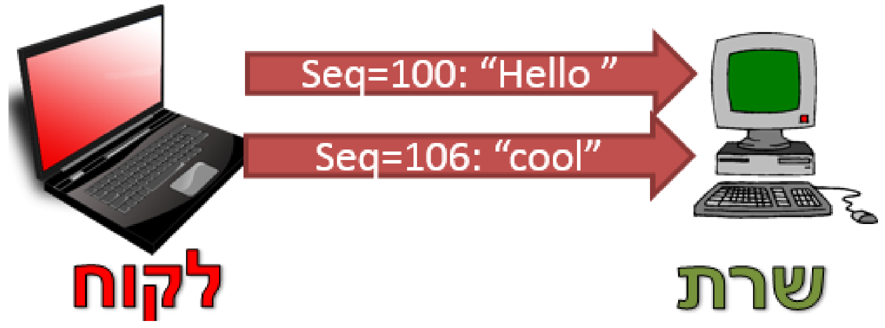
איך TCP משתמש ב-Sequence Numbers?

למדנו את חשיבותם של מספרים סידוריים ו-ACKים. כעת נסביר את השימוש של קונספטים אלו בפרוטוקול TCP.

פרוטוקול TCP לא נותן מספר סידורי לכל חבילה, אלא לכל בית (byte). כזכור, אנו מעבירים מצד לצד רצף של בתים. לכל אחד מהבתים ברצף יש מספר סידורי משלו. בכל חבילה שנשלח, יהיה המספר הסידורי שמציין את הבית הנוכחי בחבילה. כך למשל בדוגמה הבאה:

הערה: "Seq" משומש כקיצור ל-"Sequence Number".

התו "H" הוא הבית בעל המספר הסידורי 100 בתקשורת בין הלקוח לשרת. "e" הוא בעל המספר 101, ה-"l" הראשונה היא מספר 102, ה-"l" השנייה היא מספר 103, "o" הוא מספר 104 והרווח שנמצא לאחריו הוא הבית בעל המספר הסידורי 105. מכיוון שהבית האחרון שנשלח היה הבית בעל המספר הסידורי 105 (שימו לב שגם הרווח שלאחר ה-Hello נספר בתור בית!), הבית הבא יהיה בעל המספר 106 לכן, המשך התקשורת ייראה כך:



החבילה השנייה התחילה עם המספר הסידורי 106. המשמעות של כך היא שהבית הראשון שבה, כלומר התו "c", הוא בעל המספר הסידורי 106. ה-"o" שלאחריו הוא בעל המספר 107, וכן הלאה.

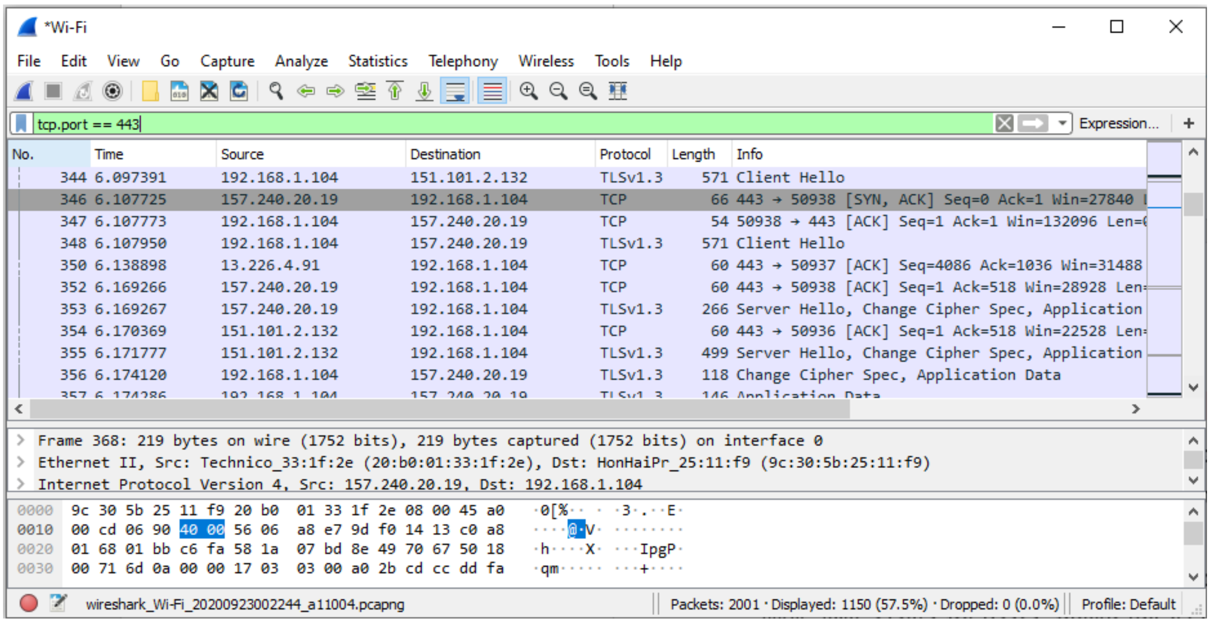
שימו לב שתקשורת TCP היא למעשה שני Streamים של מידע: רצף בתים לכל צד. התקשורת שבין הלקוח לשרת מהווה רצף בתים בפני עצמו, וה-Sequence Number בכל מקטע מתייחס לרצף בין הלקוח לשרת בלבד, ולא לרצף שנשלח מהשרת אל הלקוח.

תרגיל 6.14 מודרך – צפייה ב-Sequence Numbers של TCP

פתחו את Wireshark והריצו הסנפה. פתחו דפדפן, וגלשו אל הכתובת: http://www.themarker.com. עצרו את ההסנפה, והשתמשו בפילטר הבא:

tcp.port == 443

מהו הפילטר הזה? כיום כמעט כל אתרי האינטרנט משתמשים בפרוטוקול HTTPS, כאשר ה-S מציין "Secured". שימוש בפילטר http איתו עבדנו עד עכשיו לא ייתן תוצאות. הזכרנו את העובדה שיש Well Known Ports, מספר הפורט 443 משויך ל-HTTPS. נשתמש במידע הזה כדי לפלטר את כל הפקטות שהינן HTTPS:



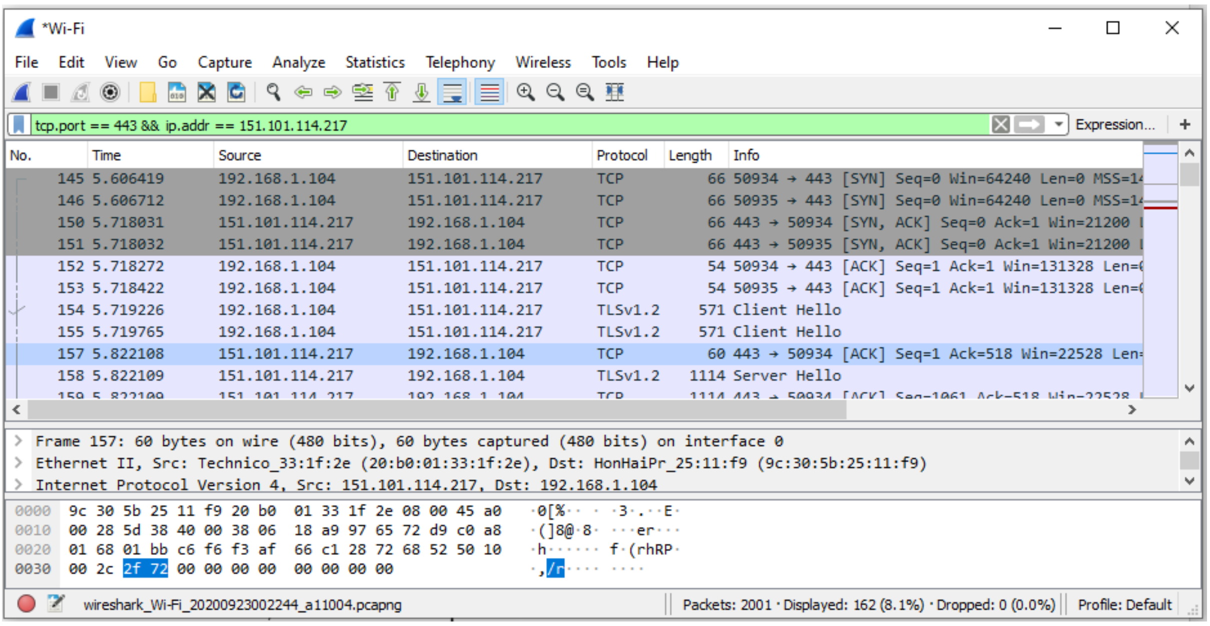
כיוון שבריב שהמחשב שלכם פועל כרגע מול מספרי שרתי HTTPS, נרצה להתמקד בתעבורה מול השרת של themarker.com. לכן נמצא את כתובת ה-ip של השרת (אנחנו כבר מנוסים בכך, למדנו לבצע שאילתות DNS) ונוסיף את התנאי לפילטר:

tcp.port == 443 && ip.addr == 151.101.114.217

דגשים:

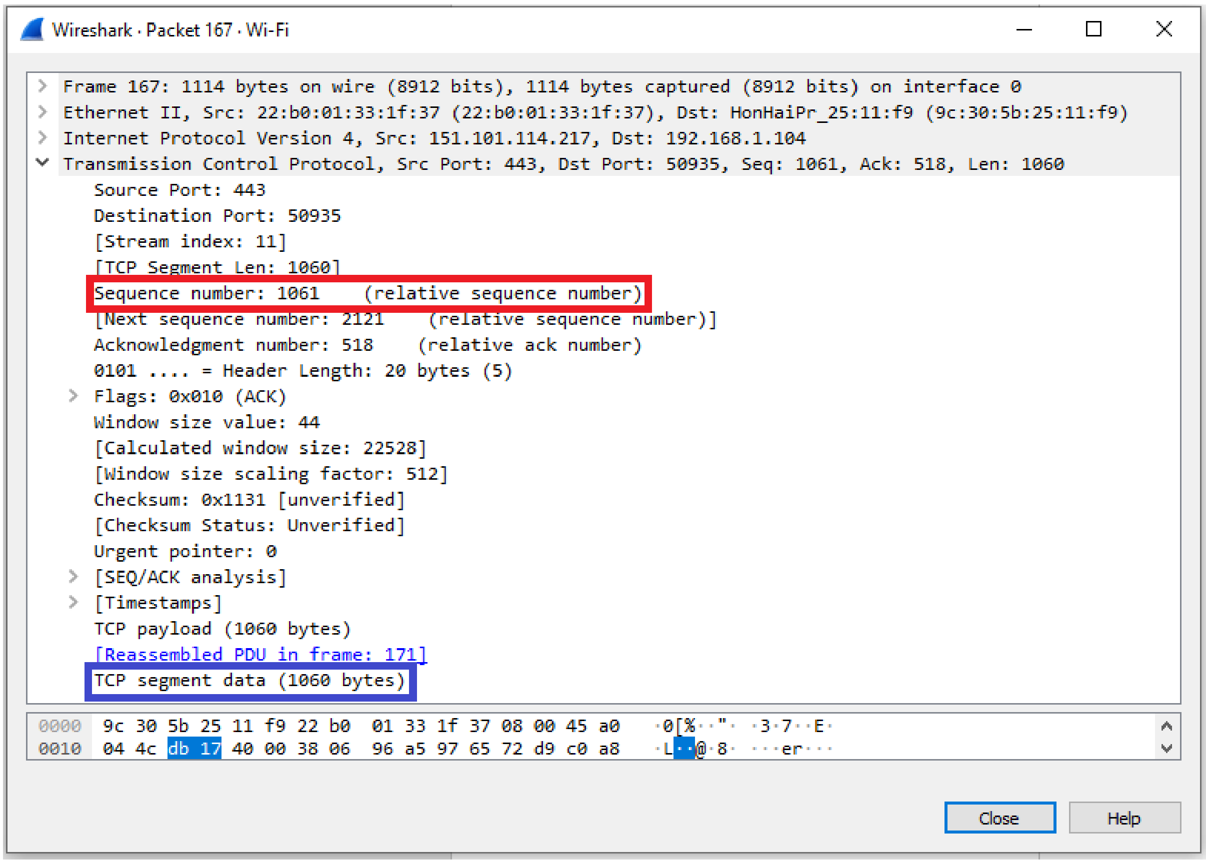
- כתובת ה-ip עשויה כמובן להשתנות עם הזמן, בידקו בעצמכם לפני ביצוע הפילטור
- הסימן && משמעותו תנאי "וגם"

- אנחנו מחפשים כל פקטה שיוצאת או נכנסת מכתובת ה-ip המבוקשת, לכן התנאי הוא ip.addr ולא ip.src או ip.dst



כעת Wireshark מציג בפנינו את התקשורת בינינו לבין השרת של themarker.

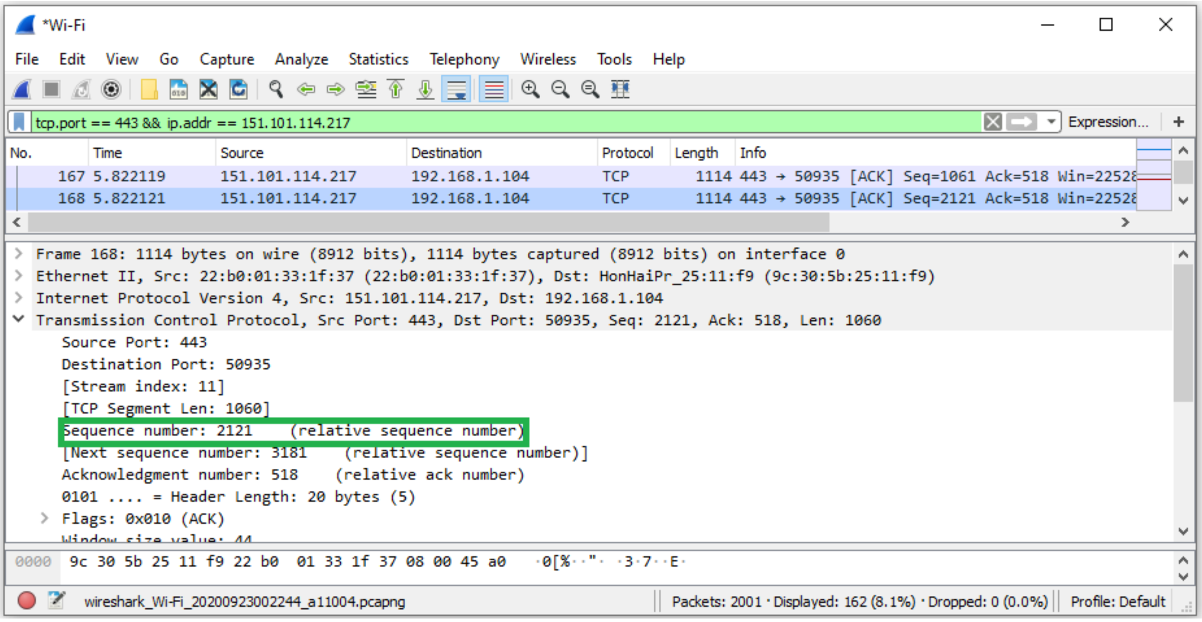
בחרו באחת החבילות שהשרת שלח אליכם, עדיף חבילה שאחריה נשלחה מייד עוד חבילה מהשרת:



באדום (1) ניתן לראות את ה-Sequence Number הנוכחי, כלומר מהו המספר של הבית הראשון בסגמנט זה, והוא **1061** בדוגמה שלנו. ב**כחול** (2) ניתן לראות את גודל המידע של הסגמנט הנוכחי – **1060** בתים. באמצעות שני נתונים אלו, המספר הסידורי של הבית הנוכחי, וכמות הבתים שנשלחים בסגמנט הנוכחי – נוכל לחשב את המספר הסידורי של הסגמנט הבא! נעשה זאת יחד:

1061 + **1060** = **2121**

גם Wireshark מציין בפנינו שזה יהיה המספר הסידורי הבא (תחת הסעיף [Next sequence number]). נמשיך לוודא זאת בעצמנו. נבחר את החבילה הבאה ונראה מה המספר הסידורי שלה:



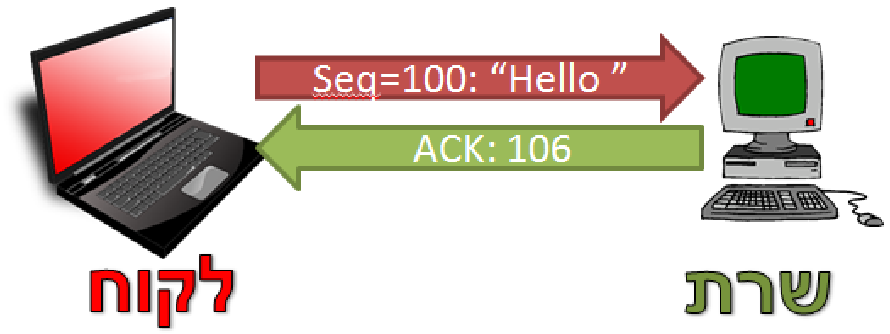
במסגרת ה**ירוקה** אנו רואים שהמספר הסידורי הוא אכן **2121**, כמו שחישבנו קודם לכן.

נסו לעשות את החישוב הזה גם על חבילות נוספות.

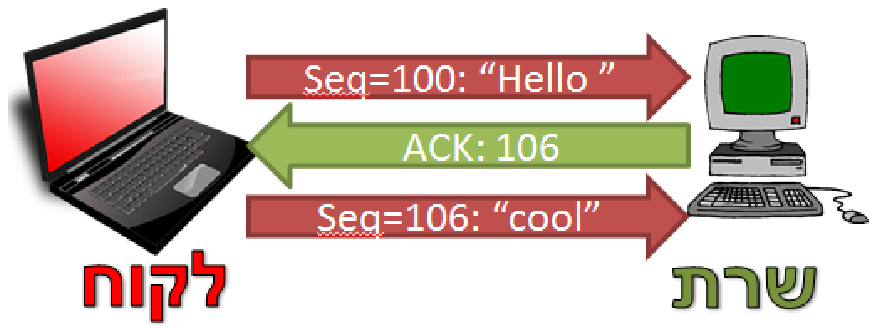
איך TCP משתמש ב-Acknowledgement Numbers?

מכיוון שהמספרים הסידוריים של TCP מתייחסים לבתים (bytes) ברצף המידע, כך גם מספר ה-ACK. מספר ה-ACK ב-TCP מציין את המספר הסידורי של הבית הבא שמצופה להתקבל. כך למשל, בדוגמה הקודמת שלנו:

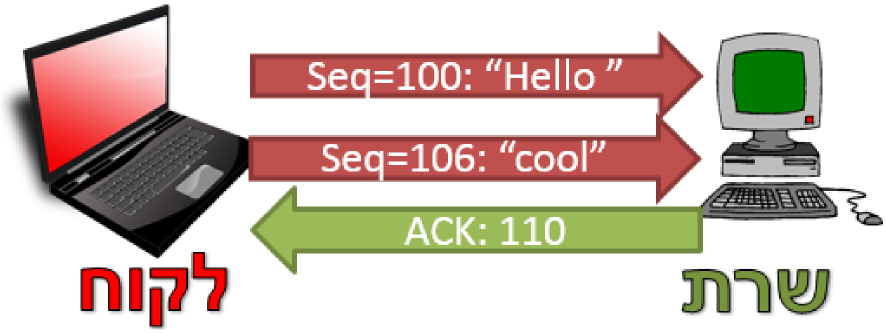
ציינו שהבית הבא שאמור להישלח מהלקוח יהיה בעל המספר הסידורי 106. אי לכך, ה-ACK אמור להכיל את הערך 106:



בצורה זו קל מאוד לבצע מעקב אחרי התקשורת. מכיוון שה-ACK מכיל את המספר הסידורי הבא, הרי שזה יהיה המספר הסידורי שיישלח בחבילת המידע הבאה. כך בדוגמה זו, רצף החבילות יראה כדלקמן:



בנוסף, כאשר נשלח ACK ב-TCP, הכוונה היא שכל המידע שהגיע עד לבית שמצוין ב-ACK הגיע באופן תקין. כך לדוגמה, במקרה לעיל השרת יכול היה לא לשלוח ACK עבור החבילה שכללה את המידע "Hello", אלא רק לאחר קבלת החבילה שכללה את המידע "cool". במקרה זה, ערך ה-ACK צריך להיות המספר הסידורי הבא – והוא יהיה 110 (שכן הוא כולל את ערך הבית הראשון בחבילה השנייה, שהוא 106, ובנוסף גודל החבילה – שהוא 4 בתים):

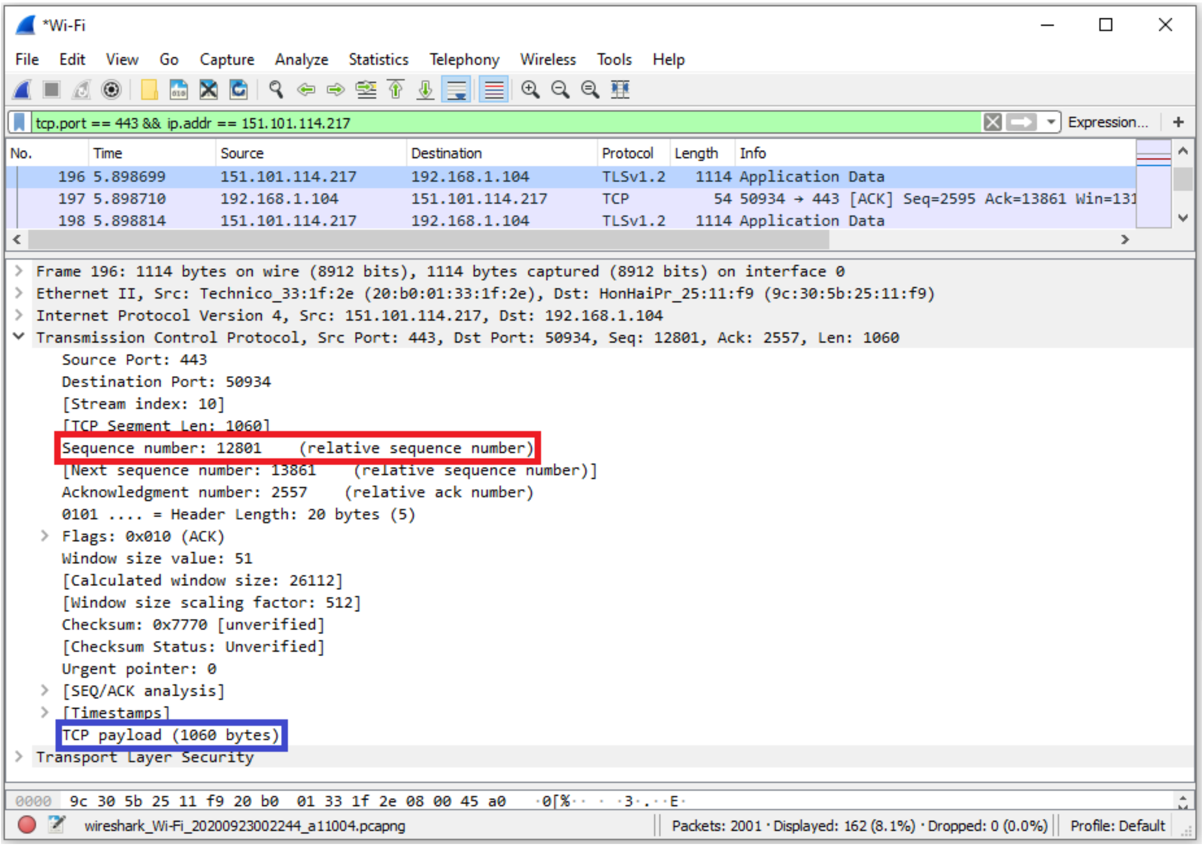


במקרה זה, הלקוח מבין ששתי החבילות, הן זאת שמכילה את המידע "Hello", והן זאת שמכילה את המידע "cool", הגיעו כמו שצריך. זאת מכיוון שכשהשרת שלח ACK עם הערך 110, הוא למעשה אמר: "קיבלתי את כל הבתים עד הבית ה-110 בהצלחה".

לאחר שליחת החבילות שלו, הלקוח מחכה זמן מסוים לקבלת ה-ACK. אם ה-ACK לא הגיע עד לתום הזמן הזה, הוא שולח אותן מחדש.

תרגיל 6.15 מודרך – צפייה Acknowledgement Numbers של TCP

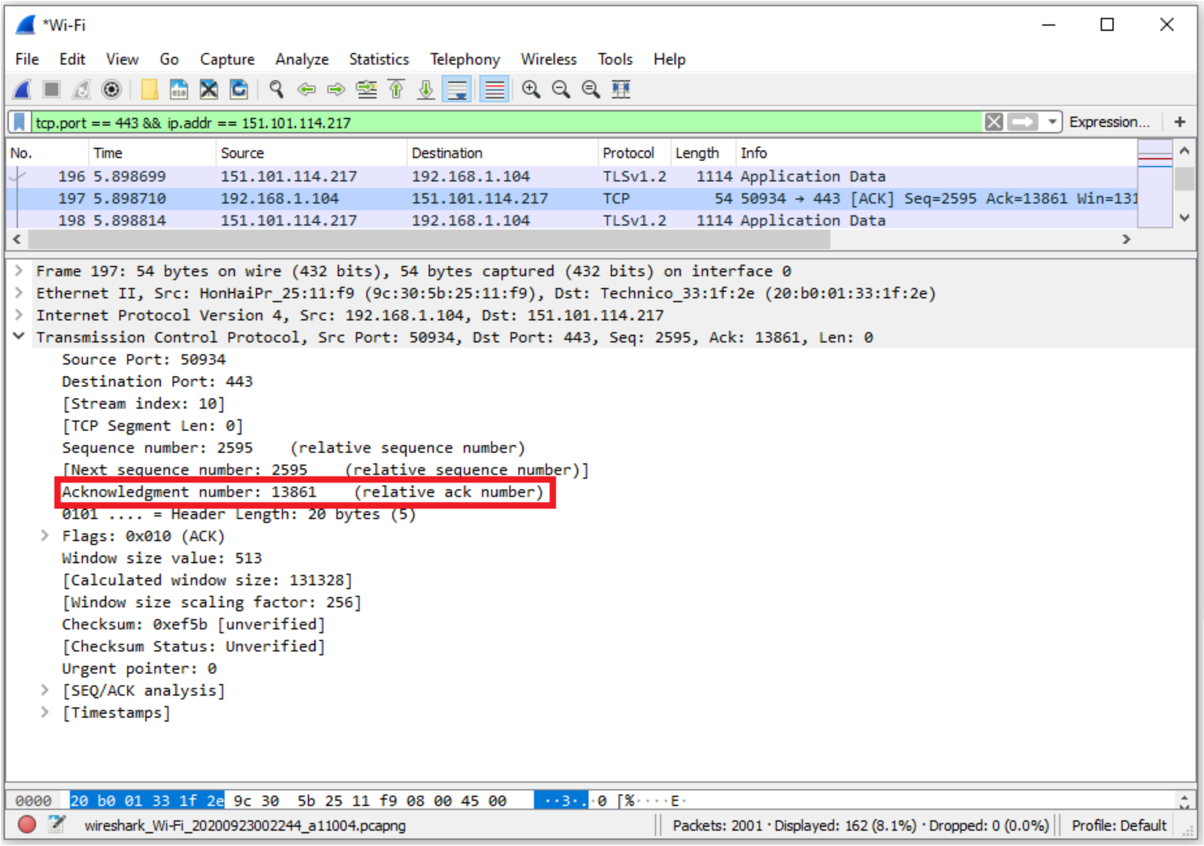
חפשו חבילה כלשהי, שמיד אחריה יש תשובה של הצד השני:



נחשב את המספר הסידורי של החבילה הבאה, כפי שלמדנו לעשות קודם לכן:

12801 + 1060 = 13861

מכאן שהמזהה הסידורי של הבית הבא אמור להיות 13861. כפי שלמדנו זה עתה, זה צפוי להיות גם הערך של חבילת ה-ACK. בואו נבחן זאת בחבילת ה-ACK הרלוונטית (שהיא החבילה הבאה, לכן ההדרכה בתחילת התרגיל היתה למצוא חבילה שמיד אחריה יש תשובה של הצד השני):



כמו שניתן לראות, אכן התקבל הערך הצפוי.

מענה ב-ACK יחיד למספר חבילות

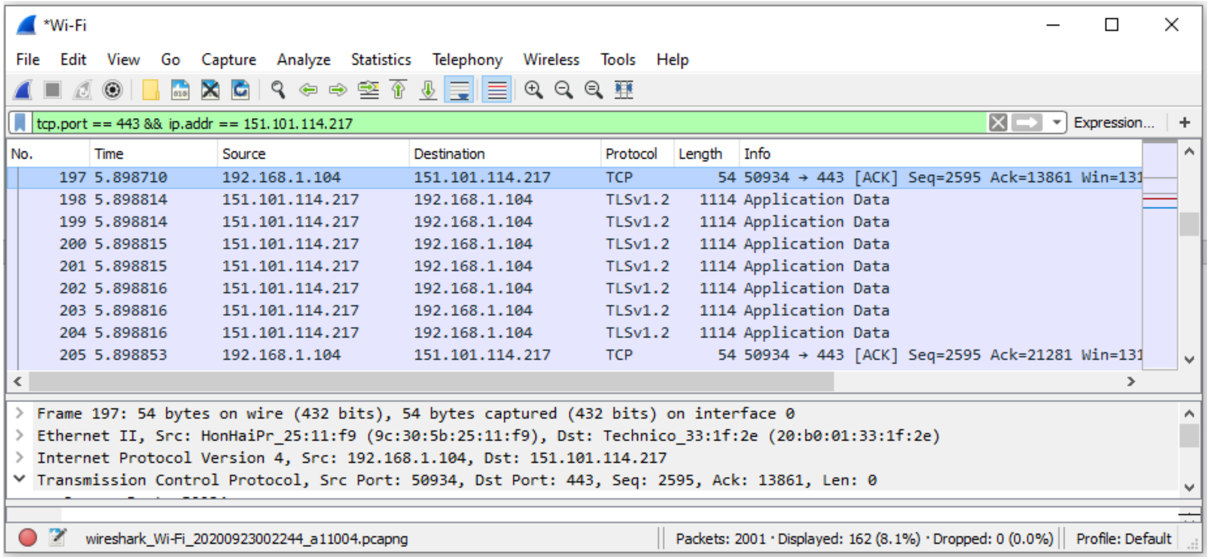
שימו לב לכך שלא בהכרח נשלחת חבילת ACK עבור כל חבילת מידע.

נניח שצד א שלח 4 חבילות, כל אחת בגודל 1000 בתים:

- חבילה עם מספר Seq = 0
- חבילה עם מספר Seq = 1000
- חבילה עם מספר Seq = 2000
- חבילה עם מספר Seq = 3000

צד ב יכול לענות ב-ACK יחיד, שערכו 4000. במקרה זה צד א יבין שכל החבילות שנשלחו על ידו לצד ב הגיעו ליעדן בצורה תקינה וכי צד ב מצפה לכך שהחבילה הבאה תתחיל ב- Seq = 4000.

ההסנפה הבאה ממחישה את התהליך:



החבילה שבראש המסך, מספר 197, היא חבילת ACK = 13861. הלקוח שולח את חבילת ה-ACK לשרת.

לאחר מכן מגיעות מספר חבילות מהשרת, הלקוח אינו עונה על כל אחת מהן בנפרד.

במקום זאת, החבילה שבתחתית המסך, מספר 205, היא חבילת ACK = 21281. כלומר ב-ACK יחיד הלקוח ענה על כל החבילות מאז ה-ACK הקודם.

נוודא זאת באמצעות חישוב קצר. ההפרש בין ה-ACKים של הלקוח הוא:

21281 – 13861 = 7420

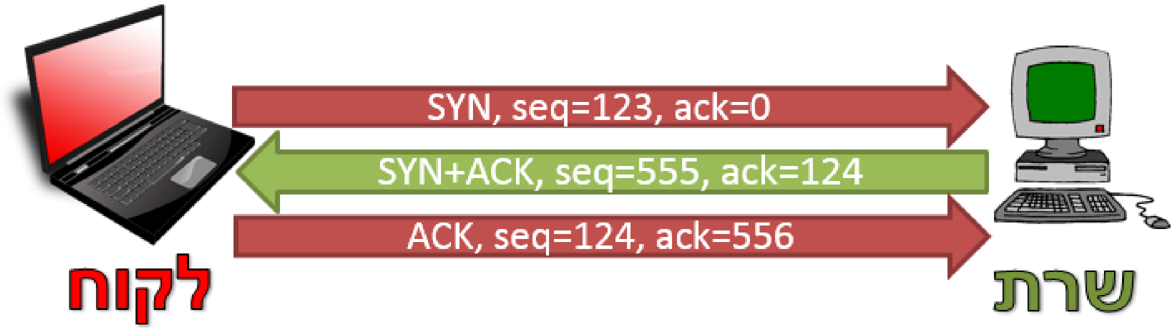
ראינו כי החבילות הקודמות שהגיעו מהשרת היו באורך של 1060 בתים של מידע. ישנן 7 חבילות שהגיעו מהשרת, והמכפלה היא בדיוק מספר הבתים שהוא ההפרש בין ה-ACKים.

נסו עתה לחשב בעצמכם את ערכי ה-ACK שצפויים להתקבל עבור חבילות נוספות שנקלטו בהסנפה שלכם, ומצאו אותם. חפשו ACKים שעונים על יותר מאשר חבילה אחת.

הקמת קישור ב-TCP

כשדיברנו על פרוטוקולים מבוססי קישור, חזרנו על כך שיש צורך להקים את הקישור בין הצדדים לפני שלב העברת המידע ביניהם. באמצעות הקמת הקישור, אנו מודיעים לצד השני שאנו מתחילים מולו בתקשורת ושעלינו להיות מוכן לכך. בנוסף, לעיתים יש לתאם פרמטרים בין שני הצדדים בכדי שהקישור יעבוד בצורה יעילה יותר.

באופן כללי, הקמת קישור ב-TCP נקראת **Three Way Handshake (לחיצת יד משולשת)**, ונראית כך:



כפי שניתן לראות, במהלך הרמת הקישור נשלחות שלוש חבילות. ישנו שימוש בשדות ה- Sequence Number וה-Acknowledgement Number של כל חבילה בכדי להצליח להרים את הקישור. כעת, נבין את התפקיד של כל חבילה ואת האופן בו מחושבים הערכים בשדות האלו.

חבילה ראשונה – SYN

בשלב הראשון, הלקוח שולח לשרת חבילה שמטרתה להתחיל את הקמת הקישור. באופן זה, הלקוח מציין: "אני רוצה להקים קישור מולך". בכל חבילה של TCP יש כמה דגלים שניתן לציין, כחלק מה-Header[34]. בחבילה זו, הדגל SYN דלוק. משמעות הדגל SYN היא תחילת תקשורת. ה-Sequence Number של חבילה זו הינו ה-Sequence Number ההתחלתי של הלקוח עבור הקישור הזה עם השרת, ונקרא בשם ISN (**Initial Sequence Number**).

כיצד נבחר ה-Initial Sequence Number?

ניתן היה להסכים שה-ISN, אותו מספר התחלתי עבור הקישור, יהיה תמיד ערך קבוע – כגון 0. דבר זה יכול להקל מאוד על הבנת התקשורת. למשל, הבית עם המספר הסידורי 0 יהיה תמיד הבית הראשון בתקשורת, הבית עם המספר הסידורי 1 יהיה הבית השני בתקשורת וכך הלאה.

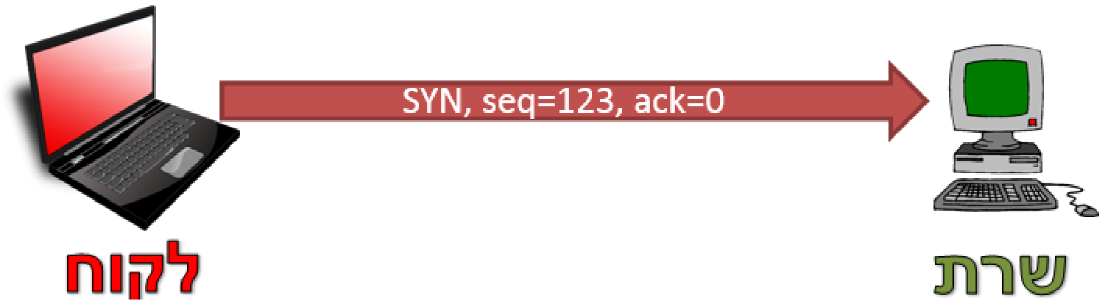
עם זאת, ה-ISN נבחר באופן רנדומלי. הסיבה העיקרית לכך היא למנוע התנגשויות של חיבורים. נדמיין לעצמנו מצב שבו כל החיבורים היו מתחילים עם המזהה 0. נאמר שהלקוח שלח לשרת חבילה עם המספר הסידורי 100. במידה שהקישור בין הלקוח לשרת נפל (למשל, מכיוון שהייתה שגיאה אצל הלקוח או אצל השרת), יקום חיבור חדש אף הוא עם המזהה 0. אז עשויה להגיע חבילה מהחיבור הקודם ליעדה, והשרת יחשוב אותה לחבילה מהחיבור החדש. אי לכך, על מנת למנוע מקרים כאלו, נבחר בכל פעם מספר באופן רנדומלי.

---

[34] במובן זה, דגל הינו ביט שמציין אפשרות מסוימת. הסבר על כל הדגלים קיים בנספח א' - TCP Header.

בדוגמה שלנו, המספר הסידורי שנבחר הינו 123. דגל ה-ACK של החבילה הראשונה כבוי, שהרי לא ניתן ACK על אף חבילה קודמת.

בשלב זה, התקשורת נראית כך:

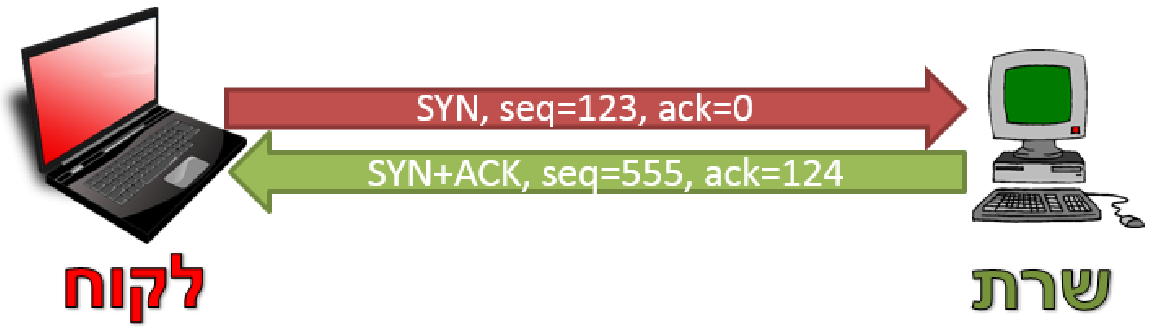


חבילה שנייה – SYN + ACK

בשלב השני, בהנחה שהשרת הסכים להקים את הקישור, הוא עונה בחבילה בה דלוקים שני הדגלים: SYN ו-ACK. הדגל SYN דלוק מכיוון שזו חבילה שמודיעה על הקמה של קישור. הדגל ACK דלוק מכיוון שהשרת מודיע ללקוח שהוא קיבל את החבילה הקודמת שהוא שלח, שהיא חבילת ה-SYN.

ה-Sequence של החבילה של השרת יהיה ה-ISN של התקשורת בינו לבין הלקוח. כלומר, יתאר את המספר הסידורי ההתחלתי של הבתים שנשלחו מהשרת אל הלקוח. נדגיש שוב שתקשורת TCP היא למעשה שני Streamים של מידע: רצף בתים לכל צד. המספר הסידורי של התקשורת של הלקוח (שמתחיל ב-123) מציין את המספר הסידורי של הבתים בין הלקוח לשרת, והמספר הסידורי שהשרת ישלח בשלב הזה יתאר את המספר ההתחלתי של הבתים בינו לבין הלקוח. גם השרת יגריל את ה-ISN באופן רנדומלי, מהסיבות שתוארו קודם לכן. בדוגמה שלנו, המספר שנבחר הוא 555.

בנוסף, על השרת לציין את מספר ה-ACK כדי להודיע ללקוח שהוא קיבל את החבילה שלו. כפי שהסברנו קודם, ה-ACK מציין את המספר הסידורי של הבית הבא שצפוי להגיע. במקרה של חבילת SYN, החבילה נספרת בגודל של בית אחד (על אף שלא נשלח שום מידע). כלומר, ערך ה-ACK יהיה המספר הסידורי של החבילה שהלקוח שלח (בדוגמה שלנו, 123) ועוד 1 עבור ה-SYN. מכאן שערך ה-ACK יהיה 124. כך נראית התקשורת בשלב זה:

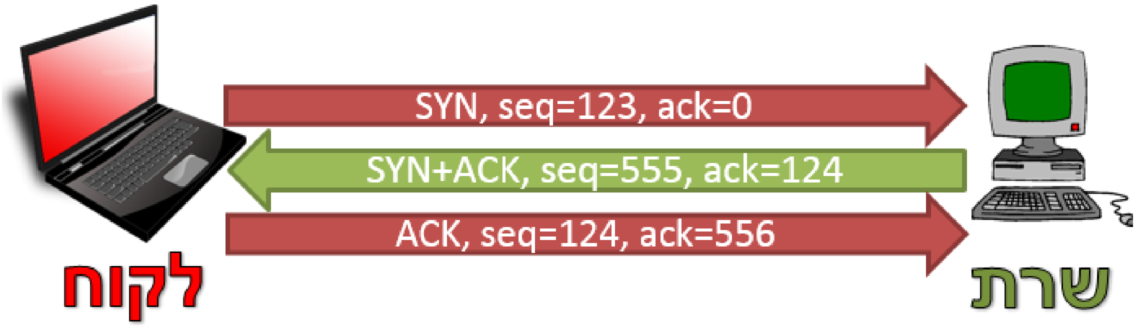


חבילה שלישית – ACK

על מנת שהקישור יוקם בהצלחה, על השרת לדעת שהחבילה הקודמת שהוא שלח, חבילת ה-SYN+ACK, הגיעה אל הלקוח בהצלחה. באם החבילה אכן הצליחה להגיע, גם הלקוח וגם השרת יודעים שהקישור קם, הסכימו להתחיל אותו, וכן מסונכרנים על המספרים הסידוריים הראשוניים (כלומר ה-ISN) אחד של השני.

על מנת לעשות זאת, הלקוח שולח חבילה כשדגל ה-ACK דלוק, ומספר ה-ACK מציין את הבית הבא שהוא מצפה לקבל מהשרת. הבית הבא מחושב על ידי שימוש במספר הסידורי שהשרת שלח (במקרה שלנו – 555) ועוד 1 עבור הבית של SYN. מכאן שבדוגמה שלנו, הערך יהיה 556.

שימו לב שהדגל SYN כבוי, שכן זו כבר לא החבילה הראשונה שנשלחת מהלקוח לשרת בקישור הנוכחי. כמובן שהלקוח צריך גם לכלול את המזהה הסידורי של הבית שהוא שולח, כמו בכל חבילה של TCP. הערך הזה הינו הערך שהיה ב-ACK של החבילה שהתקבלה מהשרת, שחושב באמצעות לקיחת המספר הסידורי הראשוני (123) והוספת 1 עבור ה-SYN. מכאן שהמספר הסידורי הוא 124:



בשלב זה הוקם קישור בין הלקוח לשרת, ועכשיו ניתן לשלוח מעליו חבילות מידע!

תרגיל 6.16 מודרך – צפייה ב-Three Way Handshake של TCP

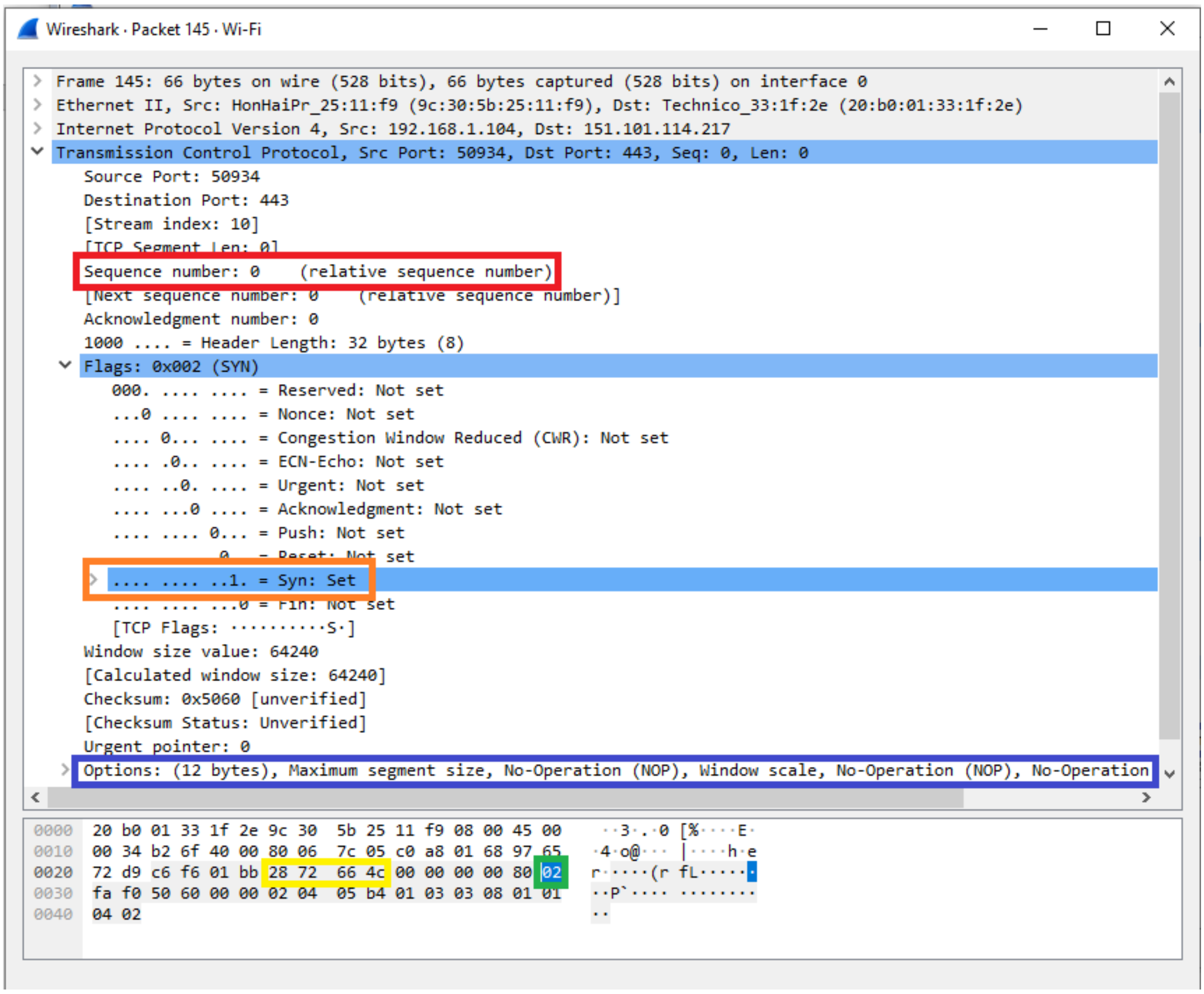
פתחו את Wireshark והריצו הסנפה. גלשו שוב אל האתר www.themarker.com , בזמן שמסנן התצוגה שלכם הוא

tcp.port == 443 && ip.addr == 151.101.114.217

לעיתים הדפדפן יוצר יותר מזרם מידע אחד מול השרת, מה שעלול להפריע מעט למחקר שלנו. לכן, בעודכם עומדים על הפקטה הראשונה הקליקו קליק ימני ואז

Follow -> TCP Stream

כעת נתמקד יחד בחבילה הראשונה של הקישור:



כפי שניתן לראות בכתום, הדגל הדלוק בחבילה הוא דגל SYN, שמציין הקמת חיבור. על מנת לראות מה הערך של שדה ה-Flags כאשר דגל ה-SYN דולק (וכל יתר הדגלים כבויים) נוכל להקליק עליו עם העכבר ונצפה בערך שמופיע למטה (בירוק) בחלון שמציג את בתי המידע בצורתם הגולמית כפי שהתקבלו, בלי פרשנות של Wireshark.

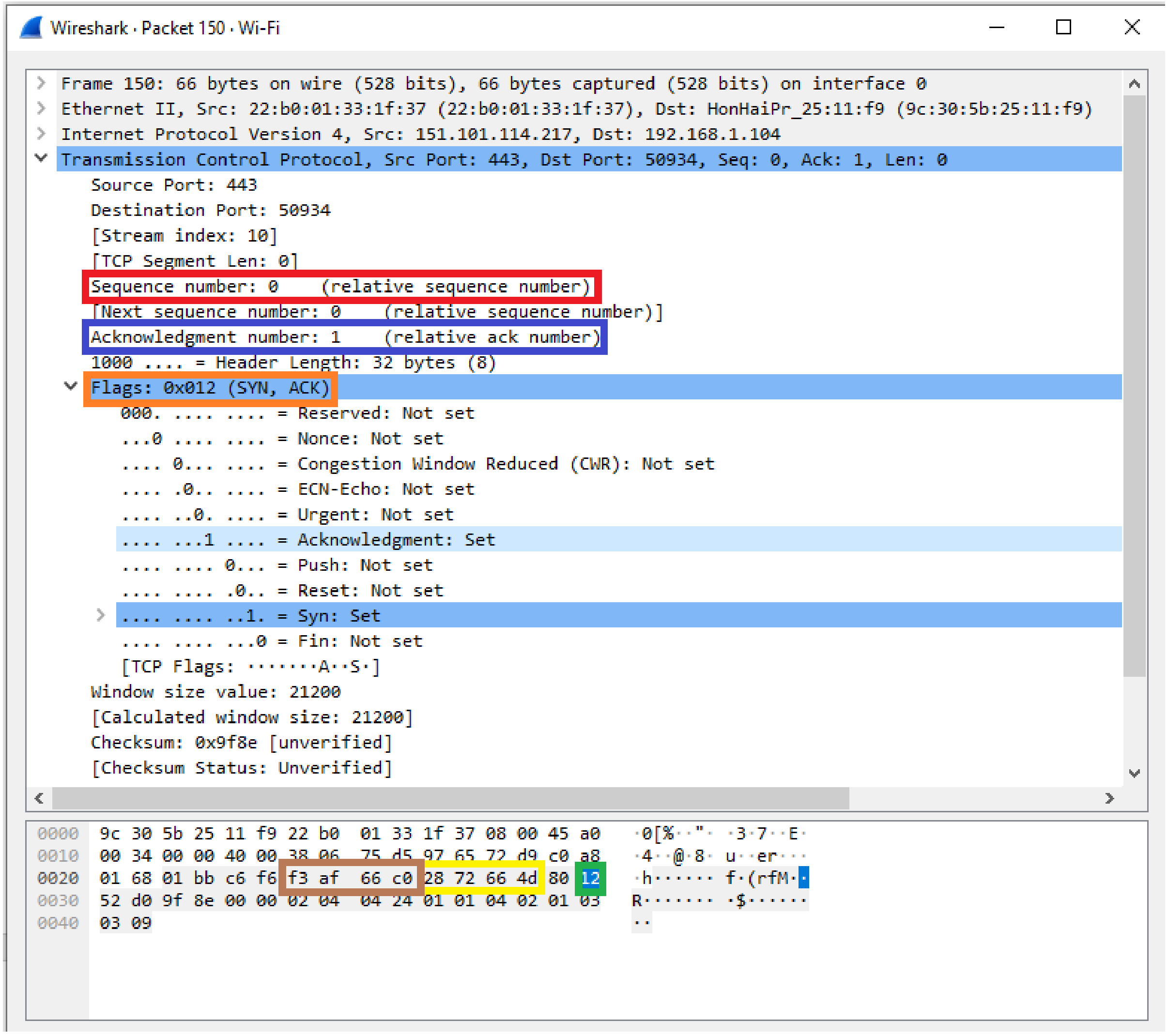
ה-Sequence Number צבוע באדום, ומקבל את הערך 0. הקליקו עליו ומיצאו את הערך האמיתי של ה-Sequence Number! בדוגמה זו, הערך מופיע במסגרת צהובה, הערך הינו 0x2872664C.

נכתוב באופן מסודר את הערכים האמיתיים של ה-Seq וה-ACK בכל אחת מהפקטות. צרו לעצמכם טבלה כזו והשלימו אותה עם הערכים שיתקבלו אצלכם:

הפקטה	ערך Seq אמיתי	ערך ACK אמיתי
SYN	0x2872664C	0
SYN-ACK		
ACK		

בנוסף, ניתן לראות בכחול את שדה ה-Options. אלו הן אפשרויות שישפיעו על כל המשך הקישור, ויש לציין אותן כבר בשלב הקמת הקישור. לא נעמיק עליהן בשלב זה. עד כאן חבילת ה-SYN.

נעבור לחבילה הבאה, חבילת ה-SYN-ACK:



כפי שניתן לראות בכתום, הדגלים הדולקים בחבילה הם אלו של SYN ו-ACK.

למטה בירוק אפשר לראות כי הערך של שני הדגלים יחד הוא 0x12 (כלומר 18). נסו לחשוב- מהו ערכו של דגל ה-ACK לבד? יש לכם את הכלים להסיק זאת כבר בשלב זה.

באדום, ניתן לראות כי גם הפעם Wireshark מציין כי ערך ה-ISN הוא 0, באופן יחסי. אם נלחץ על שדה זה, Wireshark יציג לנו את הערך האמיתי, למטה. הערך האמיתי צבוע בחום והינו 0xF3AF66C0. מיצאו את הערך האמיתי כפי שמופיע אצלכם ורישמו אותו בצד.

בכחול, אנו יכולים לראות את ערך ה-ACK. מכיוון ש-Wireshark מציג ערכים יחסיים, הוא מציג לנו כי הערך הוא 1 (ערך הגדול ב-1 מה-ISN, שצוין כ-0). אם נלחץ על שדה זה, נראה שהערך האמיתי הינו הערך שנמצא בצהוב למטה, במקרה זה 0x2872664D. שימו לב כי הערך הזה גדול ב-1 מערך ה-Seq שנשלח אלינו בחבילה הקודמת, חבילת ה-SYN. זאת מכיוון שכפי שאמרנו, האורך של חבילת SYN הינו 1, לכן הבית הבא שמצפים לו הוא ה-Seq של חבילת ה-SYN פלוס 1.

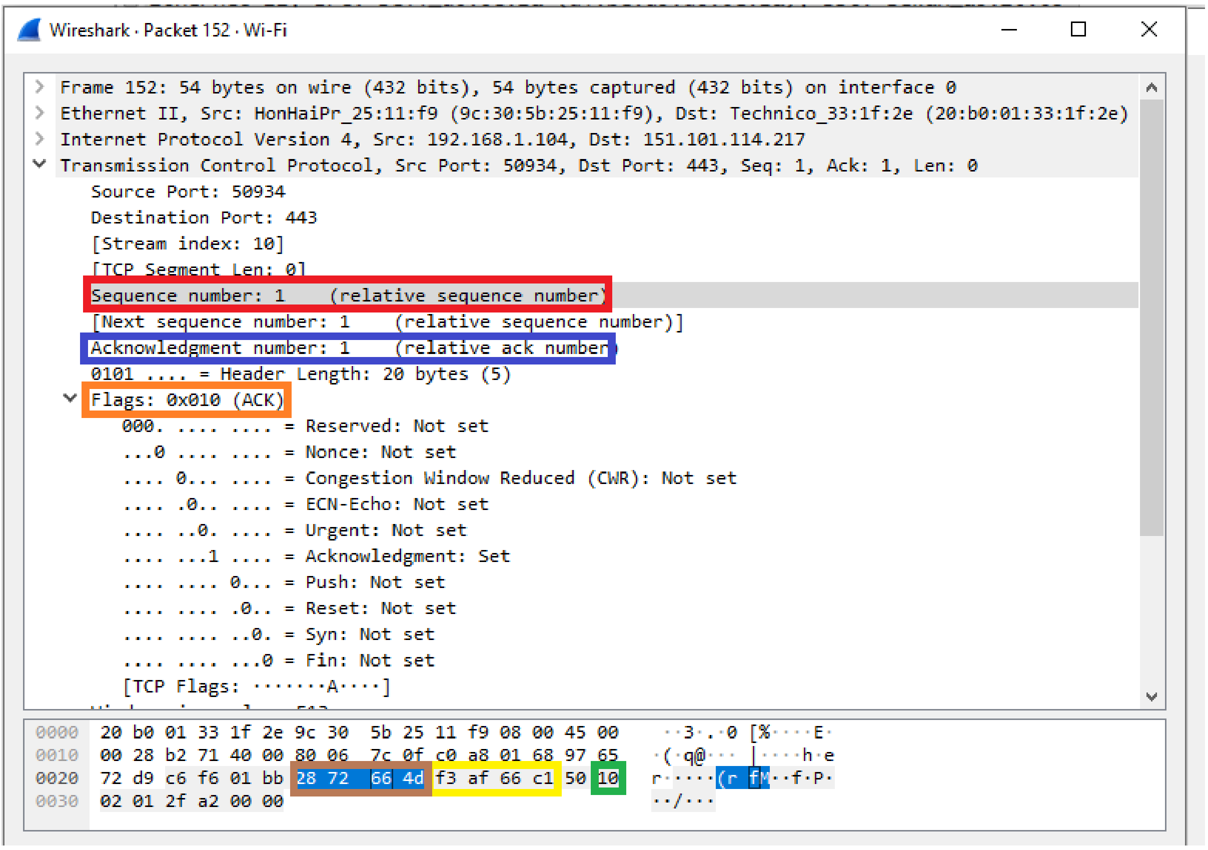
בידקו בעצמכם את ערך ה-ACK שבחבילה שלכם, השוו אותו לערך ה-SYN של החבילה הקודמת וודאו שאכן זה מה שקבלתם.

נעדכן את הטבלה שלנו עם הערכים המתאימים:

הפקטה	ערך Seq אמיתי	ערך ACK אמיתי
SYN	0x2872664C	0
SYN-ACK	0xF3AF66C0	0x2872664D
ACK		

עד כאן חבילת ה- SYN-ACK.

לסיום נסתכל בחבילת ה-ACK שמסיימת את הרמת הקישור:



כפי שניתן לראות ב**כתום**, הדגל הדלוק הוא אכן דגל ה-ACK, ואף לא דגל אחר.

למטה, ב**ירוק**, ניתן לראות שערכו של דגל ה-ACK הוא 0x10. האם צדקתם כשחשבתם שזה יהיה הערך? החישוב הוא כזה: בפקטה הראשונה גילינו שערכו של דגל ה-SYN הוא 0x2. בפקטה השניה מצאנו שסכום הערכים של SYN ועוד ACK הוא 0x12. לכן ההפרש הוא ערכו של דגל ה-ACK.

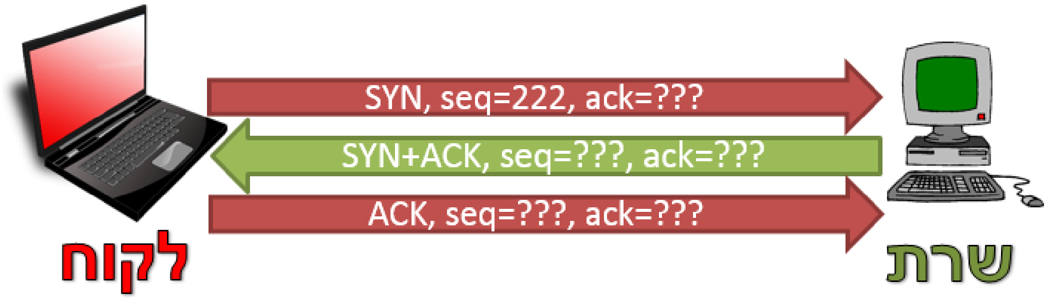
ב**אדום** ניתן לראות את המספר הסידורי של הבית הנוכחי. כפי שכבר ציינו, Wireshark מציג אותו באופן יחסי כ-1, ולחיצה עליו תראה לנו ב**חום** כי הערך האמיתי הינו 0x2872664D בבסיס הקסדצימלי.

בצהוב (4) ניתן לראות את ערך ה-ACK, שהוא באופן יחסי 1, ולחיצה עליו תגלה לנו כי הערך הוא 0xF3AF66C1 בבסיס הקסדצימלי, כצפוי.

הפקטה	ערך Seq אמיתי	ערך ACK אמיתי
SYN	0x2872664C	0
SYN-ACK	0xF3AF66C0	0x2872664D
ACK	0x2872664D	0xF3AF66C1

תרגיל 6.17 – חישובי Three Way Handshake

בשרטוט שלפנינו מוצגת לחיצת יד משולשת, אך חלק מהערכים בה חסרים ובמקומם מופיעים שלושה סימני שאלה:



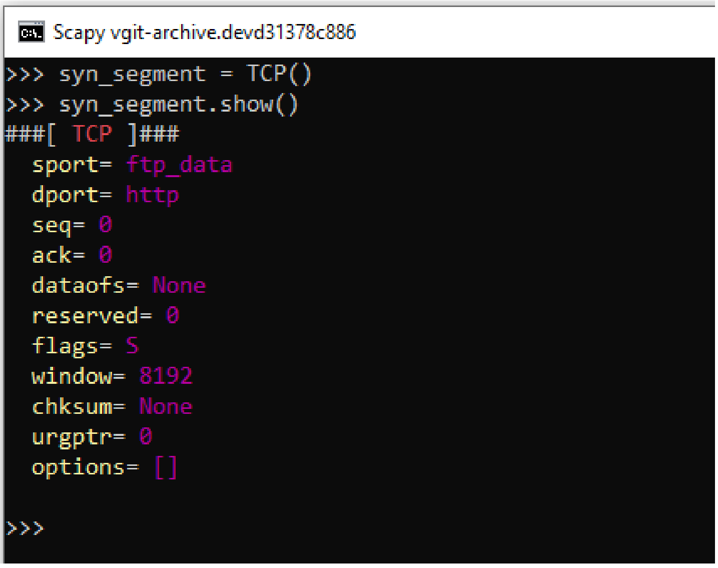
השלימו את סימני השאלה האלו בערכים משלכם. באם ערך מסוים צריך להיות רנדומלי, בחרו ערך כלשהו.

תרגיל 6.18 מודרך – מימוש Three Way Handshake

בתרגיל זה תממשו Three Way Handshake באמצעות Scapy. נתחיל מכך שנפתח הסנפה ב-Wireshark.

כעת, נעלה את Scapy וניצור חבילת TCP:

```
>>> syn_segment = TCP()
>>> syn_segment.show()
```

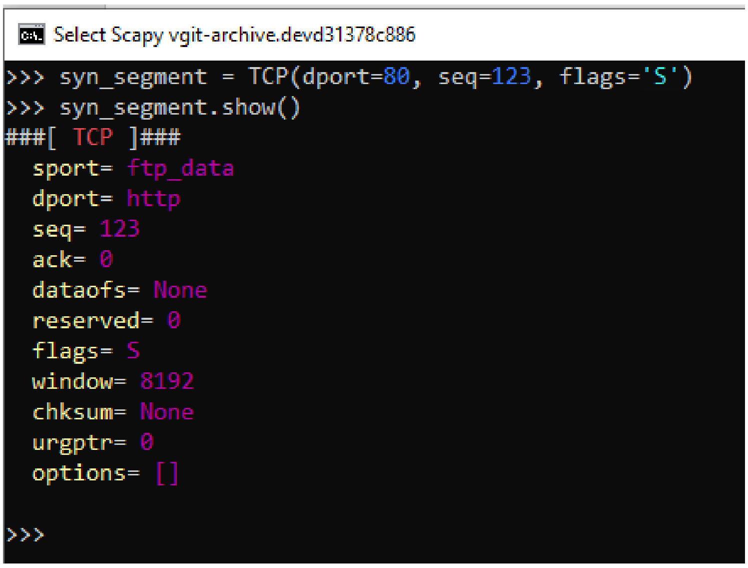

Scapy vgit-archive.devd31378c886
```
>>> syn_segment = TCP()
>>> syn_segment.show()
###[ TCP ]###
  sport= ftp_data
  dport= http
  seq= 0
  ack= 0
  dataofs= None
  reserved= 0
  flags= S
  window= 8192
  chksum= None
  urgptr= 0
  options= []

>>>
```

למעשה, ניתן לראות ש-Scapy יצר עבורנו באופן ברירת מחדל חבילה מסוג SYN, על ידי כך שרשום:

flags = S

ערך ה-Sequence Number הוא כרגע 0. פורט היעד הוא פורט 80, ו-Scapy מציג אותו עבורנו כ-HTTP. עם זאת, נרצה לייצר בעצמנו את החבילה ולשלוט בפרמטרים האלו. ולכן, ניצור בעצמנו סגמנט TCP עם הדגל SYN, כשפורט היעד הוא פורט 80, וניתן בערך ה-Sequence Number את המספר 123:

```
>>> syn_segment = TCP(dport=80, seq=123, flags='S')
>>> syn_segment.show()
```


Select Scapy vgit-archive.devd31378c886
```
>>> syn_segment = TCP(dport=80, seq=123, flags='S')
>>> syn_segment.show()
###[ TCP ]###
  sport= ftp_data
  dport= http
  seq= 123
  ack= 0
  dataofs= None
  reserved= 0
  flags= S
  window= 8192
  chksum= None
  urgptr= 0
  options= []

>>>
```

כעת נרצה "להעמיס" את שכבת ה-TCP שיצרנו מעל שכבה של IP, בכדי שנוכל לשלוח אותה. ננסה לשלוח את החבילה אל google, ולכן ניצור אותה בצורה הבאה:

```
>>> syn_packet = IP(dst='www.google.com')/syn_segment
```

>>> syn_packet.show()

```
Select Scapy vgit-archive.devd31378c886
>>> syn_packet = IP(dst='www.google.com')/syn_segment
>>> syn_packet.show()
###[ IP ]###
  version= 4
  ihl= None
  tos= 0x0
  len= None
  id= 1
  flags=
  frag= 0
  ttl= 64
  proto= tcp
  chksum= None
  src= 192.168.1.104
  dst= Net('www.google.com')
  \options\
###[ TCP ]###
     sport= ftp_data
     dport= http
     seq= 123
     ack= 0
     dataofs= None
     reserved= 0
     flags= S
     window= 8192
     chksum= None
     urgptr= 0
     options= []

>>>
```

נשלח את החבילה בזמן שאנו מסניפים:

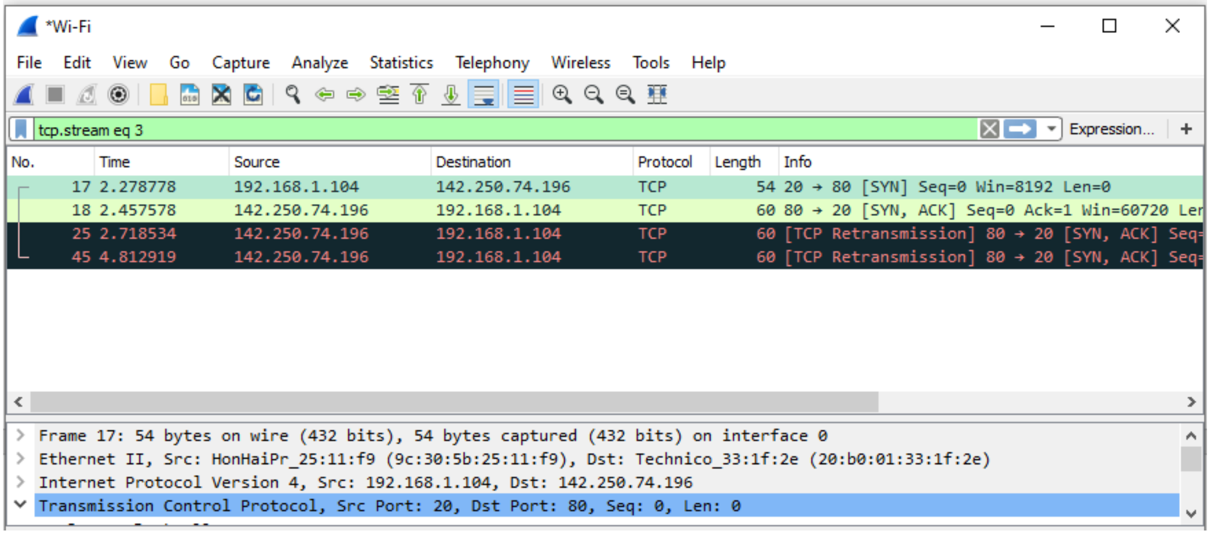
>>> send(my_packet)

נמצא את החבילה בהסנפה. אם יש לכם תעבורה רבה, אפשר לפלטר כפי שלמדנו את כל החבילות שיש להן דגל SYN דולק:

tcp.flags.syn == 1

לאחר שמצאנו את החבילה של ה-SYN ששלחנו, נעקוב אחרי כל התעבורה באמצעות הקלקה ימנית על החבילה ואז:

Follow -> TCP Stream



כפי שניתן לראות, חבילת ה-SYN שלנו הגיעה אל Google. השרת של Google, בתורו, ניסה להשלים את לחיצת היד המשולשת ושלח אלינו חבילת תשובה – SYN + ACK. עם זאת, לא הגבנו על חבילה זו. השרת של Google, שהניח שאולי החבילה לא הגיעה אלינו, ניסה לשלוח אותה אלינו פעמיים נוספות. לאחר מכן, הוא התייאש. החיבור לא קם.

כעת נשלח שוב את החבילה באמצעות Scapy, אך הפעם נשתמש בפקודה **sr1** שפגשנו קודם לכן, בכדי שנוכל לשמור את חבילת התשובה של Google:

```
>>> syn_ack_packet = sr1(syn_packet)
>>> syn_ack_packet.show()
```

```
Scapy vgit-archive.devd31378c886
>>> syn_ack_packet = sr1(syn_packet)
Begin emission:
Finished sending 1 packets.
...*
Received 4 packets, got 1 answers, remaining 0 packets
>>> syn_ack_packet.show()
###[ IP ]###
  version= 4
  ihl= 5
  tos= 0xa0
  len= 44
  id= 13048
  flags=
  frag= 0
  ttl= 121
  proto= tcp
  chksum= 0x87a6
  src= 172.217.23.164
  dst= 192.168.1.104
  \options\
###[ TCP ]###
     sport= http
     dport= ftp_data
     seq= 3173284599
     ack= 124
     dataofs= 6
     reserved= 0
     flags= SA
     window= 60720
     chksum= 0xebab
     urgptr= 0
     options= [('MSS', 1380)]
###[ Padding ]###
        load= '\x00\x00'
```

ניתן לראות שערך ה-ACK הינו אכן 124, כמו שציפינו והסברנו קודם לכן בפרק זה. בנוסף, ערך ה-Sequence הוא ערך רנדומלי שהוגרל בשרת של Google. בשדה הדגלים, הערך הוא "SA", כלומר SYN (שמיוצג על ידי "S") ו-ACK (שמיוצג על ידי "A").

בעיה נפוצה:

ביצוע הפעולות שהדגמנו כעת עלול להסתיים בהודעת שגיאה של Scapy:

```
    raise ConnectionResetError(*exc.args)

Exception [WinError 995] The I/O operation has been aborted because of either
 a thread exit or an application request
Press ENTER to continue...
>>>
```

הסיבה להודעת השגיאה היא שמערכת ההפעלה שלנו מזהה כי קבלנו הודעת SYN ACK אך לא מוכר לה שיש תהליך כלשהו במחשב שלנו ששלח בקשת SYN (חבילת ה-SYN שיצרנו נוצרה באופן מלאכותי, ואינו

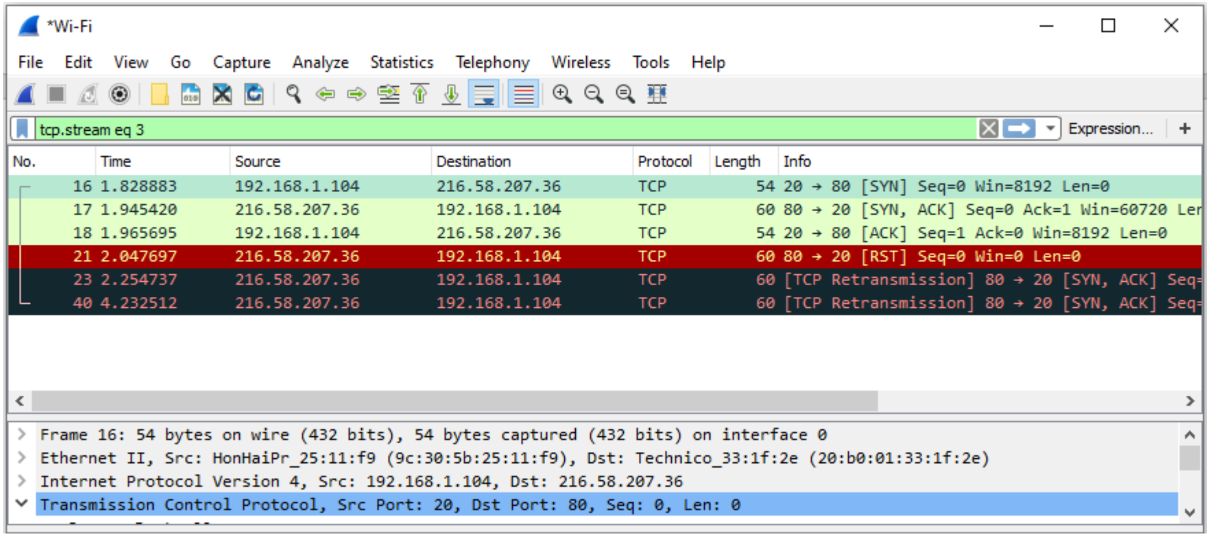
שייכת להתליך מסויים במחשב שלנו שמנסה לתקשר עם שרת חיצוני). כתוצאה מכך מבחינת מערכת ההפעלה קיבלנו כרגע חבילת SYN ACK חשודה, שלא באה כתגובה לפניה שלנו לשרת. לכן מערכת ההפעלה מבצעת Reset ומעלה Exception שנקרא ConnectionResetError. עם זאת, עדיין הצלחנו לקבל את פקטת ה-SYN ACK.

כעת, כשבידכם חבילת ה-SYN+ACK, **המשיכו בעצמכם**. השתמשו בחבילה הזו בכדי ליצור את חבילת ה-ACK שתשלים את החיבור, ושלחו אותה אל Google.
מומלץ להשתמש בשלד התוכנית הבאה, להשלים את הקוד במקום בו נמצא סימן השאלה ולהריץ אותה ב-Pycharm:

```python
from scapy.all import *

syn_segment = TCP(dport=80, seq=123, flags='S')
syn_packet = IP(dst='www.google.com')/syn_segment
syn_ack_packet = sr1(syn_packet)
ack_segment = ?
ack_packet = IP(dst='www.google.com')/ack_segment
send(ack_packet)
```

אם תצליחו, תקבלו בהסנפה את החבילות הבאות:

- חבילת SYN
- חבילת SYN ACK
- חבילת ACK
- חבילת Reset (צבועה באדום) שמערכת ההפעלה יצרה ושלחה לשרת
- חבילות SYN ACK שהשרת שלח לנו לפני שחבילת ה-Reset ששלחנו הספיקה להגיע אליו

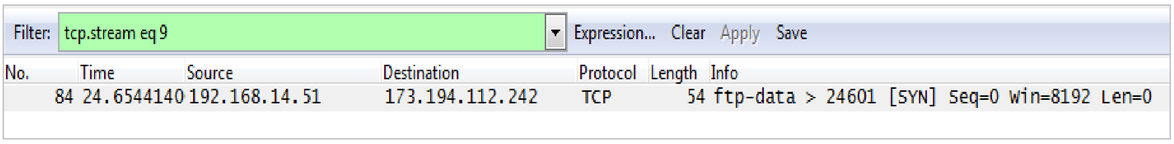


תרגיל 6.19 – איזה שירותים פתוחים?

באמצעות ביצוע Three Way Handshake, אנו יכולים לגלות אילו שירותים פתוחים אצל מחשב מרוחק. כיצד?

קודם לכן, כאשר שלחנו חבילת SYN אל פורט 80 של Google, קיבלנו בתשובה חבילת SYN+ACK. מכך למדנו שפורט 80 "פתוח" אצל Google, כלומר יש אצלו תוכנה שמאזינה על פורט 80. מכיוון שאנו יודעים שעל פורט 80 מאזינה בדרך כלל תוכנה שנותנת שירות HTTP, גילינו שכרגע שירות ה-HTTP "פתוח" אצל Google וניתן לגשת אליו.

מה יקרה אם נשלח חבילת SYN לפורט "סגור", כלומר לפורט שאף תוכנה לא מאזינה עליו? בואו ננסה זאת. נשלח חבילת SYN לשרת של Google, אך לפורט 24601:



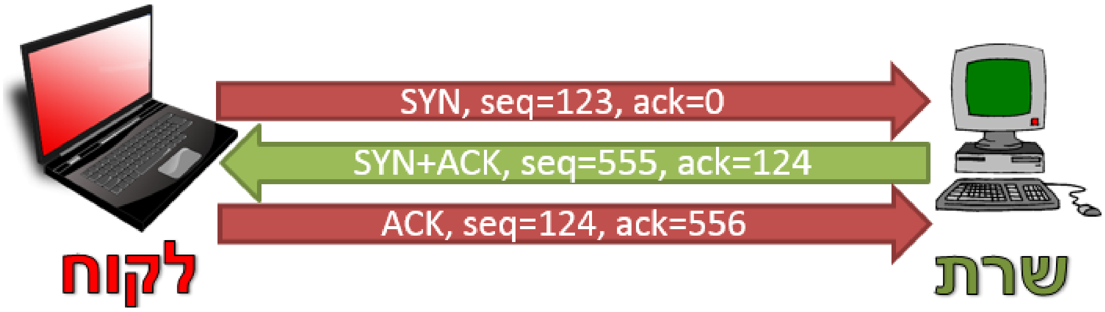
השרת של Google כלל לא נתן תשובה של SYN+ACK! במקרים מסוימים, שרתים מרוחקים יענו חבילה כשדגל ה-RST דולק, ומשמעותו שהשרת לא מוכן להרים את הקישור.

השתמשו בהתנהגות זאת כדי לכתוב סקריפט אשר מקבל מהמשתמש כתובת IP, ומדפיס למסך אילו פורטים פתוחים במחשב המרוחק, בטווח הפורטים 20-1024. מכיוון שהסקריפט עתיד לשלוח תעבורה רבה, **אל תבדקו אותו** על שרתים באינטרנט, אלא רק על מחשבים נוספים בביתכם או בכיתתכם.

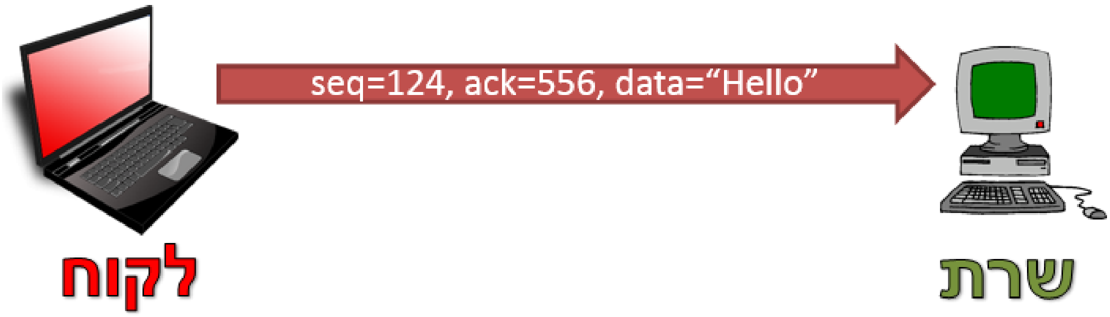
העברת מידע על גבי קישור שנוצר

הבנו כיצד מרימים קישור ב-TCP. לאחר ביצוע ה-Three Way Handshake, קיים קישור בין שני הצדדים. כעת, כל סגמנט שמגיע משויך בידי TCP לקישור מסוים. בנוסף, TCP יודע לסדר את החבילות שהוא מקבל לפי סדר השליחה שלהן ולקבל מחדש חבילות שהגיעו בצורה משובשת, או לא הגיעו כלל, זאת על ידי שימוש ב-Sequence Numbers וב-Acknowledgement Numbers.

החישוב של שדות ה-Sequence Number וה-Acknowledgement Number לאורך הקישור, ממשיך באותו האופן שבו הוא בוצע במהלך ה-Three Way Handshake. כך לדוגמה, נאמר שהלקוח והשרת הרימו קישור בצורה הבאה:



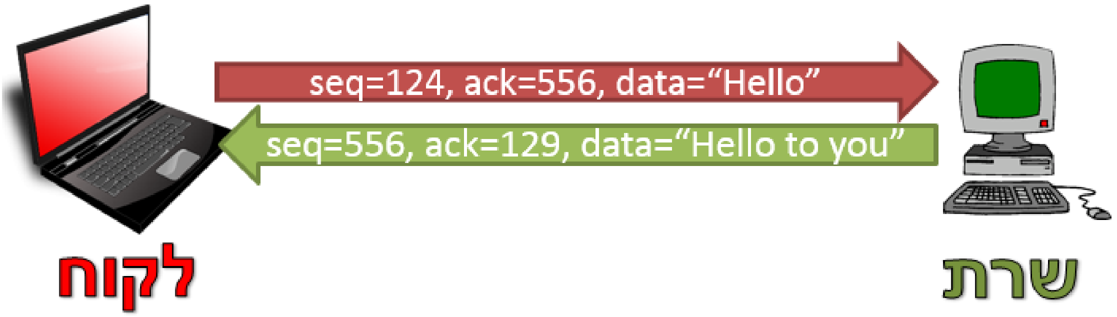
כעת, הלקוח רוצה לשלוח הודעה נוספת, ובה המידע: "Hello". כיצד ייראו המזהים?
ובכן, שדה ה-Sequence Number יהיה 124, שכן זה הבית הבא שנשלח. בהודעה האחרונה שמוצגת בציור לעיל, הלקוח שלח ACK, אך הוא לא שלח כלל מידע. שדה ה-ACK יישאר בעל הערך 556, שכן לא נשלח מידע חדש מהשרת:



כעת נאמר שהשרת רוצה להשיב ב-ACK, אך גם לכלול את ההודעה: "Hello to you". מה יהיו הערכים בהודעה?

שדה ה-Sequence Number יכיל את הערך 556, מכיוון שזה הבית הבא שהשרת צפוי לשלוח. שימו לב שניתן לדעת זאת מכיוון שזה ערך ה-ACK בחבילה האחרונה של הלקוח.

בשביל לחשב את הערך של שדה ה-ACK, עלינו לקחת את ה-Sequence Number האחרון של הלקוח (והוא 124), ולחבר אליו את אורך ההודעה שהוא שלח (האורך הוא 5, מכיוון שזה האורך של המחרוזת "Hello"). אי לכך, ערך ה-ACK יהיה 129:

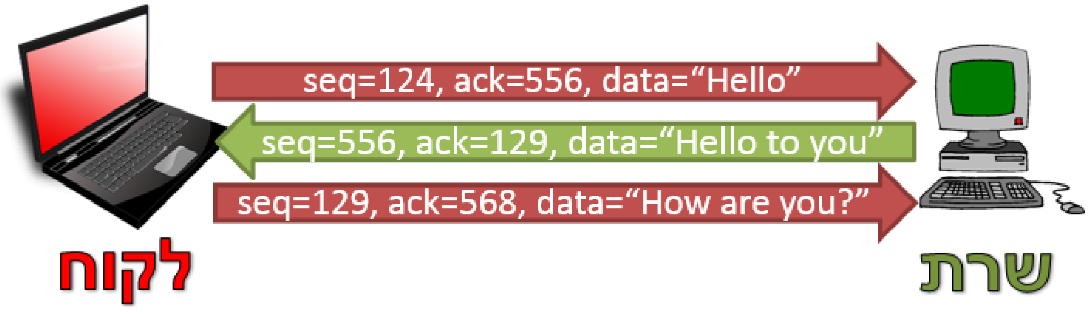


כעת הלקוח רוצה לשלוח לשרת ACK על החבילה שהתקבלה, אך להוסיף מסר – "How are you?".

נסו לחשב בעצמכם: מה יהיה ערך ה-Sequence Number? מה יהיה ערך שדה ה-ACK?

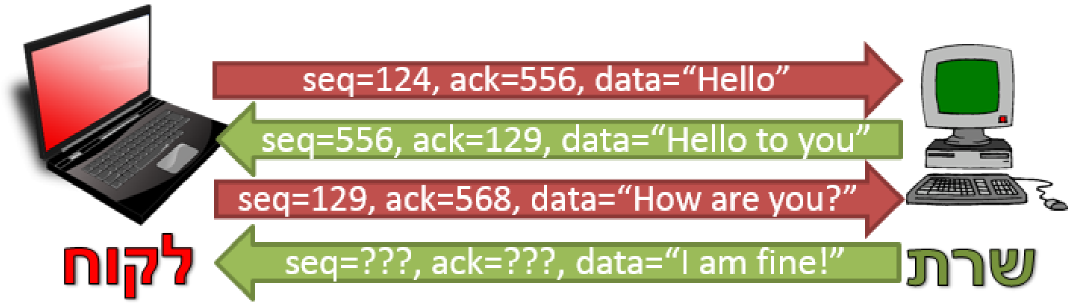
ראשית נחשב את שדה ה-Sequence Number של ההודעה. הערך יהיה זהה לערך ה-ACK של החבילה הקודמת: כלומר 129.

כעת נחשב את שדה ה-ACK. שדה ה-ACK מחושב על פי שימוש ב-Sequence Number של השרת (שהיה 556), ועוד האורך של המידע שהוא שלח. האורך של המחרוזת "Hello to you" הוא 12 בתים, ולכן הערך יהיה 568 (12 + 556):



תרגיל 6.20 – חישובי ערכים בשיחת TCP

בשרטוט שלפניכם מוצג המשך השיחה לעיל, אך חלק מהערכים בה חסרים ובמקומם מופיעים שלושה סימני שאלה:



השלימו את סימני השאלה האלו בערכים המתאימים.

תפקידים ושיפורים נוספים של TCP

דיברנו על TCP לא מעט, ועם זאת – נגענו רק בחלק קטן מהתפקידים של TCP ודרכי המימוש שלהם.

TCP אחראי גם לכך שהמידע יעבור מצד לצד בצורה יעילה עד כמה שניתן. לשם כך, למשל, TCP לא תמיד מחכה ל-ACK על מנת לשלוח את החבילה הבאה – אלא שולח מספר חבילות יחד. עם זאת, TCP לא מעוניין לשלוח יותר מידע ממה שהמחשב שמקבל יהיה מוכן לאחסן. אי לכך, על הצדדים להסכים על גודל מסוים של כל מקטע[35]. כמו כן, TCP מנסה למנוע היווצרות של עומס על הרשת כולה. באם TCP מזהה שיש עומס על הרשת, הוא פועל בכדי לשלוח פחות מידע ולאפשר לרשת "להתאושש".

ל-TCP יש גם שיפורים שנוספו לאורך הזמן וקיימים בחלק מהמימושים שלו. לדוגמה, בחלק מהמקרים, לא צריך לחכות לכך שלא יגיע ACK בכדי לשלוח חבילה מחדש. לעיתים, צד אחד של החיבור יכול להבין שחסרה לו חבילה שלא הגיעה, ולהודיע על כך לשולח באמצעות חבילות מיוחדות בשם NAK.

על תפקידים אלו ועוד לא נרחיב במסגרת ספר זה, אך אתם מוזמנים להרחיב אופקיכם ולקרוא עליהם באינטרנט, ובסעיף צעדים להמשך בפרק זה.

---

[35] גודל המידע שהמחשב המקבל מוכן לקבל נקרא גודל החלון. הוא דינאמי, ונשלח בכל מקטע TCP בשדה Window Size. תהליך ניהול החלון הוא מורכב, ותוכלו להרחיב את הידע שלכם אודותיו בסעיף צעדים להמשך של פרק זה.

שכבת התעבורה – סיכום

בפרק זה סקרנו את השכבה הרביעית במודל חמש השכבות, היא שכבת התעבורה. התחלנו מלמידה על מטרות השכבה, וכחלק מכך הסברנו את המושג **פורט**. הבנו את מיקום השכבה במודל חמש השכבות, הכרנו את הכלי **netstat** ואף תרגלנו את השימוש בו. לאחר מכן הכרנו את המושגים **חיבור מבוסס קישור וחיבור שאינו מבוסס קישור**. למדנו על ההבדלים ביניהם, ומתי נעדיף להשתמש בכל אחד.

מאוחר יותר העמקנו בפרוטוקול **UDP**. באמצעות Wireshark למדנו להכיר את השדות השונים של הפרוטוקול. לאחר מכן למדנו איך לכתוב לקוח ושרת שמשתמשים בפרוטוקול UDP באמצעות Sockets. כמו כן, למדנו לשלוח ולקבל פקטות UDP באמצעות Scapy. כתבנו מספר שרתים, שלחנו שאילתת DNS וכן העברנו מידע סודי באמצעות שימוש במספרי פורטים.

לאחר מכן למדנו על פרוטוקול **TCP**. הבנו כיצד TCP משתמש ב-Sequence Numbers וב-ACKים בכדי לשמור על אמינות. למדנו כיצד מרימים קישור ב-TCP באמצעות Three Way Handshake, צפינו ב-Wireshark כיצד נראות חבילות של TCP, בעת הקמת קישור ובכלל. בהמשך מימשנו בעצמנו Three Way Handshake באמצעות Scapy, והשתמשנו ביכולת זו בכדי לגלות אילו שירותים פתוחים במחשב מרוחק. אז העמקנו בדרך בה נראית תקשורת לאחר שחיבור קם, והבנו כיצד מחושבים ערכים של שדות שונים בחבילות TCP. לסיום, הזכרנו בקצרה תפקידים של TCP עליהם לא העמקנו בפרק זה.

במהלך הפרק, הכרנו לעומק מספר בעיות אפשריות ברשת, ודרכים להתמודד עימן. להלן טבלה המסכמת את האתגרים ומנגנוני ההתמודדות:

בעיה	מנגנון התמודדות	האם קיים ב-UDP?	האם קיים ב-TCP?
חבילה לא מגיעה ליעדה	שליחת Acknowledgement על מידע שנשלח	לא	כן
חבילה מגיעה ליעדה, אך בסדר לא נכון מתוך רצף חבילות	כלילת מספר סידורי עבור כל מידע שנשלח	לא	כן
חבילה מגיעה ליעדה, אך באופן לא תקין	שימוש ב-Checksum	כן	כן

בפרק הבא, נמשיך ללמוד על מודל השכבות ונלמד להכיר את שכבת הרשת. נדבר על האתגרים הניצבים בפני שכבה זו, וכיצד היא מתקשרת לשכבת התעבורה עליה למדנו עתה.

שכבת התעבורה – צעדים להמשך

כפי שהבנתם מסעיף תפקידים ושיפורים נוספים של TCP בפרק זה, יש עוד דברים רבים ללמוד אודות TCP בפרט, ושכבת התעבורה בכלל.

אלו מכם שמעוניינים להעמיק את הידע שלהם בשכבת התעבורה, מוזמנים לבצע את הצעדים הבאים:

קריאה נוספת

בספר המצוין Computer Networks (מהדורה חמישית) מאת Andrew S. Tanenbaum ו- David J. Wetherall, הפרק השישי מתייחס במלואו לשכבת התעבורה. באופן ספציפי, מומלץ לקרוא את הסעיפים:

- 6.5.4 – מספק מידע על ה-TCP Header, ויכול להשלים את המידע שניתן בנספח א' של פרק זה.
- 6.5.6 – מתאר על תהליך סיום תקשורת TCP, אשר לא סקרנו במהלך פרק זה.
- 6.5.8 – ניהול החלון של TCP.
- 6.5.10 – ניהול עומסים של TCP.

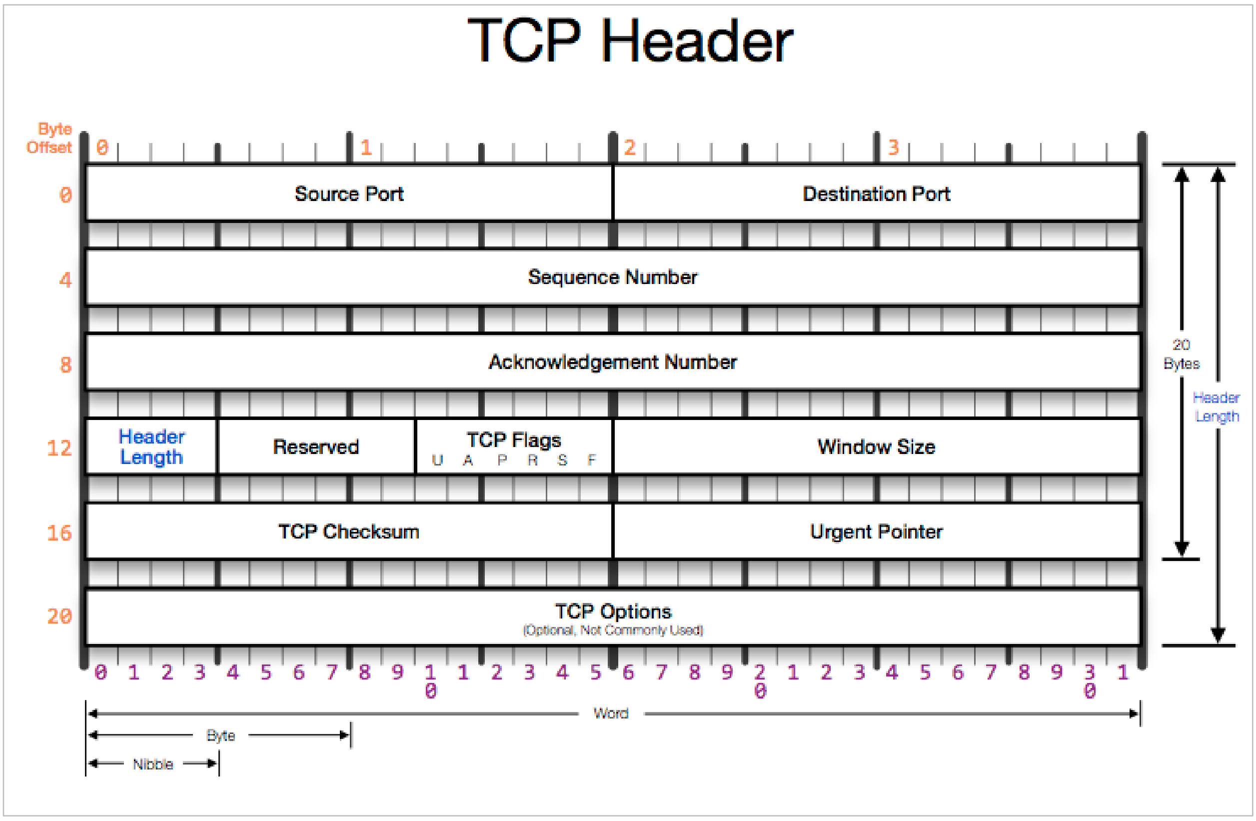
בספר Computer Networking: A Top-Down Approach (מהדורה שישית) מאת James F. Kurose, הפרק השלישי מוקדש כולו לשכבת התעבורה. באופן ספציפי, מומלץ לקרוא את הסעיפים:

- 3.3.2 – למתעניינים בדרך בה מחושב ה-Checksum של UDP.
- 3.4 – שיטות ועקרונות במימוש פרוטוקול אמין.
- 3.6 – עקרונות ושיטות לניהול עומסים.
- 3.7 – ניהול עומסים של TCP.

כמו כן, ניתן להרחיב את אופקיכם בפרק על TCP ו-UDP מתוך The TCP/IP Guide, אותו ניתן למצוא בכתובת: http://goo.gl/GC69q4.

נספח א' – TCP Header

נספח זה נועד כדי לתאר את כלל השדות של ה-Header של TCP. לא חיוני להבין את כל השדות עד הסוף. מידע נוסף ניתן למצוא בכתובת: http://goo.gl/CyVbvX.



- Source Port – פורט המקור.
- Destination Port – פורט היעד.
- Sequence Number – מספר סידורי עוקב, המזהה את המקטע בתוך זרם המידע הכולל. כל Sequence Number מתאר בית אחד ברצף המידע.
- Acknowledgment Number – מתאר את הבית הבא שצפוי להתקבל.
- Header Length – מתאר את אורך ה-Header של החבילה, מכיוון שהוא עשוי להשתנות מחבילה לחבילה, מכיוון שישנם שדות של TCP Options אותם נתאר בהמשך. הערך של השדה מתאר את הגודל ביחידות מידה של 32 ביטים. כך למשל, עבור חבילה בגודל 20 בתים, הערך כאן יהיה 5 (מכיוון שמדובר ב-160 ביטים, שהם 5 פעמים 32 ביטים). הערך 5 (שמתאר 20 בתים) הוא הערך הנפוץ ביותר עבור שדה זה.
- Reserved – ביטים ששמורים עבור שימוש עתידי.

- Flags – הדגלים
  - C – דגל CWR – קיצור של Congestion Window Reduced – מתייחס להקטנת Congestion Window וקשור לטיפול בעומסים. לא נרחיב על נושא זה בספר זה.
  - E – דגל ECN-ECHO – קיצור של Explicit Congestion Notification – מודיע לשולח שבקשת Congestion Experienced התקבלה. כחלק משיפורים חדשים יותר של TCP. לא נרחיב על נושא זה בספר זה.
  - U – דגל URG – קיצור של Urgent. הסגמנט מכיל מידע דחוף, והשדה Urgent Pointer מצביע על מידע דחוף זה.
  - A – דגל ACK – קיצור של Acknowledgement – אישור על קבלת מקטע קודם. מאשר את המספר הסידורי שמופיע בשדה Acknowledgement Number.
  - P – דגל PSH – מבקש ממערכת ההפעלה להעביר את המידע ללא דיחוי. לא נרחיב על נושא זה בספר זה.
  - S – דגל SYN – בקשה להקמת קישור.
  - F – דגל FIN – בקשה לסגירת קישור קיים.
- Window Size – גודל חלון השליחה של TCP. מציין כמה מידע השולח מוכן לקבל ברגע זה.
- Checksum – מוודא תקינות המידע בתוך הסגמנט. ה-Checksum מתבצע גם על המידע עצמו, ולא רק על ה-Header.
- Urgent Pointer – מצביע למיקום המידע הדחוף בסגמנט, שהדגל URG מורה על קיומו.
- Options – אפשרויות נוספות.
  - דוגמה לאופציה: ניתן לכלול Maximum Segment Size – גודל המקטע המקסימלי המותר לשליחה.

פרק 7

שכבת הרשת

בתחילת מסענו אל רשת האינטרנט עשינו שימוש בכלי tracert כדי לדעת מה המסלול בינינו לבין שרת כלשהו. המסלול, כפי שראינו, הוא שכבת הרשת. בפרק זה נענה על שאלות רבות ומרתקות – מהי תפקידה של שכבת הרשת? מהו פרוטוקול IP? מה היא כתובת IP? מה זה נתב? בסיום הפרק נלמד לכתוב כלי tracert בעצמנו.

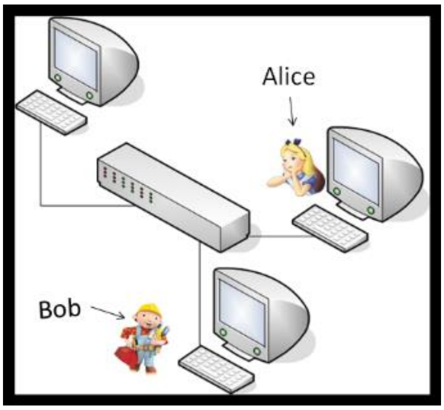
נתחיל מהשאלה הראשונה:

מה תפקידה של שכבת הרשת?

ראשית עלינו להבין מה מהותה של השכבה השלישית במודל השכבות, הלוא היא שכבת הרשת.

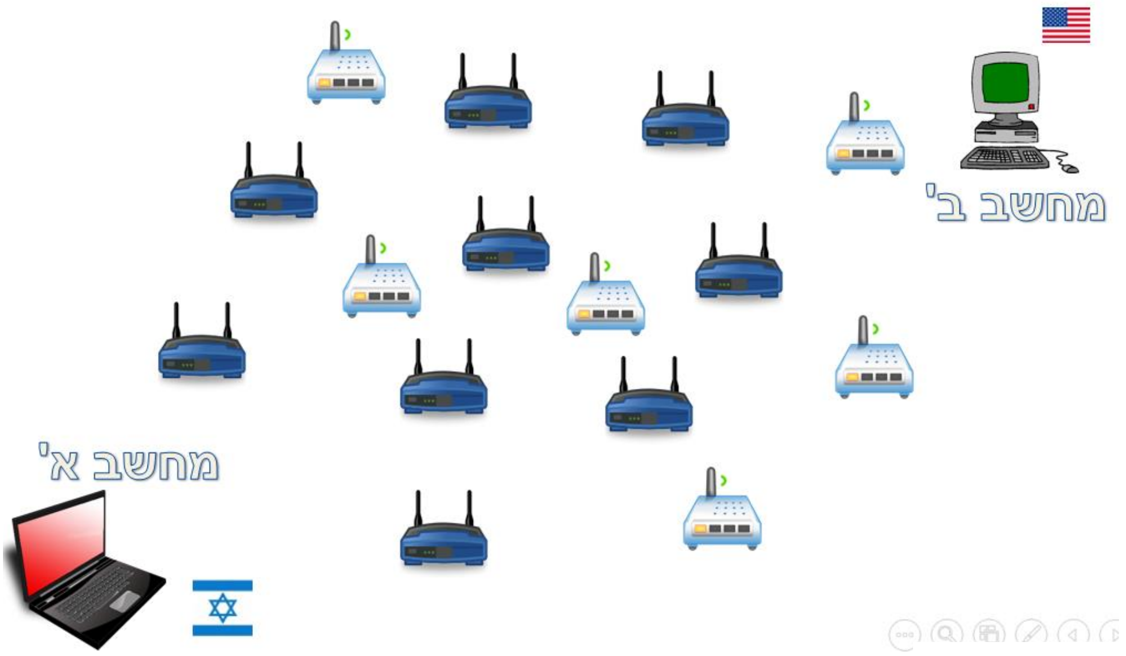
בפרק הקודם למדנו על שכבת התעבורה, והבנו שהיא מאפשרת לנו לשלוח הודעה ממחשב אחד למחשב אחר. כמו שלמדנו, שכבת התעבורה עשויה לדאוג לכך שלא יהיה כישלון בהעברת המידע בין הצדדים. אם כך, מדוע צריך את השכבה השלישית? **חשבו על כך ונסו לענות על שאלה זו בעצמכם.**

כדי לענות על השאלה הזו, נתחיל מלחשוב על רשתות ישנות, שהיו קיימות לפני כ-30 שנים. עצמו את העיניים, דמיינו עולם ללא טלפונים ניידים וללא רשת האינטרנט שאתם מכירים כיום. רשת סטנדרטית בתקופה הזו הייתה יכולה לכלול כמה מחשבים בודדים, שחוברו באמצעות כבלים וקופסה קטנה.



בעולם כזה, בו רשתות כוללות מספר מצומצם של מחשבים בלבד, קל יחסית לגרום למידע לעבור מצד לצד (בדוגמה שבציור – מהמחשב של Bob אל המחשב של Alice). אך רשתות אלו אינן דומות כלל וכלל לרשתות שאנו מכירים היום! המחשב שלכם, שנמצא כאן בישראל, כמו גם הסמארטפון שלכם, נמצאים באותה רשת

כמו שרת Google שנמצא בארצות הברית, והיא רשת האינטרנט. על מנת להעביר מידע בין מחשבים[36] ברשת האינטרנט, עלינו לעבור בדרך בהרבה רכיבים שונים.הביטו בשרטוט הרשת הבא:



בשרטוט זה ניתן לראות את מחשב א', שנמצא בישראל, ומחשב ב', שנמצא בארצות הברית. כעת, נניח שהמשתמש במחשב א' רוצה לשלוח הודעה אל מחשב ב'.

שכבת התעבורה, עליה למדנו בפרק הקודם, מניחה כי אפשר להעביר חבילת מידע בודדת ממחשב א' למחשב ב'. שכבת הרשת היא האחראית לתהליך זה.

מטרת שכבת הרשת היא להעביר חבילות מידע מישות[37] אחת אל ישות אחרת.

נסתכל שוב בשרטוט לעיל. ישות אחת (מחשב א') מעוניינת לשלוח מידע לישות שנייה (מחשב ב').

המידע הזה מחולק בתורו לחבילות מידע. כשאנו מתייחסים ל"**חבילות מידע**" בשכבת הרשת, אנו קוראים להן בשם **Packets, ובעברית – חבילות (או "פקטות")**. מכיוון שרוב המידע שעובר באינטרנט מועבר באמצעות השכבה השלישית, אנו מכנים לרוב כל מידע כזה בשם "פקטה", כפי שראיתם לאורך הספר.

בדוגמה לעיל, שכבת הרשת אחראית להעביר את חבילות המידע בין מחשב א' לבין מחשב ב'.

---

[36] המושג "מחשבים" משומש כאן מטעמי נוחות. למעשה, הדבר נכון עבור כל ישות רשת – כמו נתבים או שרתים. שימו לב שכאשר אנו מתייחסים ל"ישות" ברשת, אנו מתייחסים לכרטיס רשת מסוים. כך למשל, מחשב עם שני כרטיסי רשת מייצג למעשה שתי "ישויות רשת" שונות.

[37] למשל ממחשב אחד לאחר או ממחשב לשרת.

שימו לב שכל נקודת קצה בפני עצמה אינה מכירה את מבנה הרשת הכולל. כלומר, מחשב א' (המקור) לא יודע אילו רכיבים נמצאים בינו לבין מחשב ב' (היעד). הוא "מבקש" משכבת הרשת לשלוח את החבילה עבורו, ובאחריות שכבת הרשת להבין את מבנה הרשת.

אילו אתגרים עומדים בפני שכבת הרשת?

שימו לב שבפני שכבת הרשת עומדת משימה לא קלה בכלל. חבילה שמגיעה ממחשב א' למחשב ב' עוברת בדרך קשה ומפותלת. בין השאר, שכבת הרשת עשויה להתמודד עם:

1. חומרות שונות – ייתכן שבדרך בין מחשב א' למחשב ב' יהיו רכיבים שונים לחלוטין, כשאחד מהם הוא שרת עצום ואחד מהם הוא קופסה קטנה.
2. תקנים שונים – ייתכן שהתקשורת בין מחשב א' למחשב ב' תעבור בלוויין בחלל, לאחר מכן באמצעות כבל רשת סטנדרטי[38], לאחר מכן באמצעות WiFi ובחזרה.

מיקום שכבת הרשת במודל השכבות

שכבת הרשת היא השכבה השלישית במודל חמש השכבות.

מה השירותים ששכבת הרשת מספקת לשכבה שמעליה?

שכבת הרשת מספקת לשכבת התעבורה, השכבה הרביעית במודל השכבות, מודל של "ענן", שבו חבילות מידע מגיעות מצד אחד לצד שני. שכבת התעבורה אינה מודעת כלל למבנה הרשת המתואר, ולמעשה מבחינתה יש פשוט "רשת כלשהי" שמחברת בין מחשב א' למחשב ב'. "תמונת הרשת", מבחינתה, נראית כך:

בצורה זו, שכבת הרשת מצליחה "להעלים" את הרשת מבחינת שכבת התעבורה! שכבת התעבורה מבקשת "שלחי לי חבילה מכאן לכאן" (למשל ממחשב א' למחשב ב'), ושכבת הרשת דואגת לכל התהליך משם ואילך.

---

[38] הכוונה היא לכבל Ethernet, עליו נלמד בפרק שכבת הקו.

מה השירותים ששכבת הרשת מקבלת מהשכבה שמתחתיה?

שכבת הקו מספקת לשכבת הרשת ממשק להעברת מידע בין שתי ישויות המחוברות זו לזו באופן ישיר. באופן זה, שכבת הרשת לא צריכה לדאוג לסוגיות הקשורות לחיבור בין שתי תחנות. את שכבת הרשת לא מעניין אם הישויות מחוברות בכבל, בלוויין, או באמצעות יוני דואר. היא רק אחראית להבין מה המסלול האופטימלי. כמו ש-Waze רק אומרת לרכב באיזו דרך לעבור, ולא מסבירה לנהג שהוא צריך לתדלק, ללחוץ על הגז או לאותת. בזה יטפל הנהג, או במקרה שלנו – שכבת הקו. על כל זאת, נלמד לעומק בפרק הבא.

מסלולים ברשת

תרגיל 7.1 מודרך – מי נמצא בדרך שלי?

עכשיו נסו בעצמכם – כיצד תגלו מה הדרך בה חבילת מידע עוברת מן המחשב שלכם אל Facebook מבלי להשתמש באתר חיצוני?

לשם כך נוכל להיעזר בכלי אשר פגשנו קודם לכן בספר, בשם **tracert**.

פתחו את שורת הפקודה (CMD). הנכם כבר יודעים כיצד לעשות זאת.

כעת, הקלידו את הפקודה: **tracert -d www.facebook.com**

אתם צפויים לראות פלט הדומה לכך:

```
C:\WINDOWS\system32\cmd.exe
Microsoft Windows [Version 10.0.18362.1082](c) 2019 Microsoft Corporation. All rights reserved.

C:\Users\barak>tracert www.google.com

Tracing route to www.google.com [172.217.18.4]
over a maximum of 30 hops:

  1     1 ms     1 ms     1 ms  OpenWrt.lan [192.168.1.1]
  2     5 ms     2 ms     1 ms  176-230-251-233.orange.net.il [176.230.251.233]
  3     *        *        *     Request timed out.
  4     *        *        *     Request timed out.
  5     *        *        *     Request timed out.
  6     *        *        *     Request timed out.
  7     3 ms     3 ms     3 ms  82.102.132.78
  8    70 ms    93 ms    99 ms  EDGE-FRA-01-ae3-42.ip4.012.net.il [80.179.166.50]
  9   205 ms    70 ms    62 ms  72.14.216.121
 10    85 ms   100 ms   102 ms  216.239.59.17
 11   125 ms   203 ms   305 ms  74.125.37.125
 12   108 ms   100 ms    60 ms  fra02s19-in-f4.1e100.net [172.217.18.4]

Trace complete.

C:\Users\barak>
```

כל שורה כאן מייצגת תחנה נוספת בדרך בין המחשב שלכם לבין Facebook. מכאן שהחבילה שנשלחה אל www.facebook.com הגיעה ראשית אל 192.168.1.1, לאחר מכן אל 176.230.251.233 וכך הלאה. ישנם

מספר רכיבי רשת בדרך אשר לא הגיבו, לכן הם מסומנים בכוכבית. אין זה אומר שרכיבי הרשת הללו לא תקינים, אלא רק שהם לא מחזירים לנו אינפורמציה שמאפשרת לנו לדעת שהחבילה שלנו עברה דרכם.

שימו לב: הפלט במחשבכם יהיה שונה מהפלט של הפקודה שבדוגמה. בהמשך נבין ממה נובעים שינויים אלו.

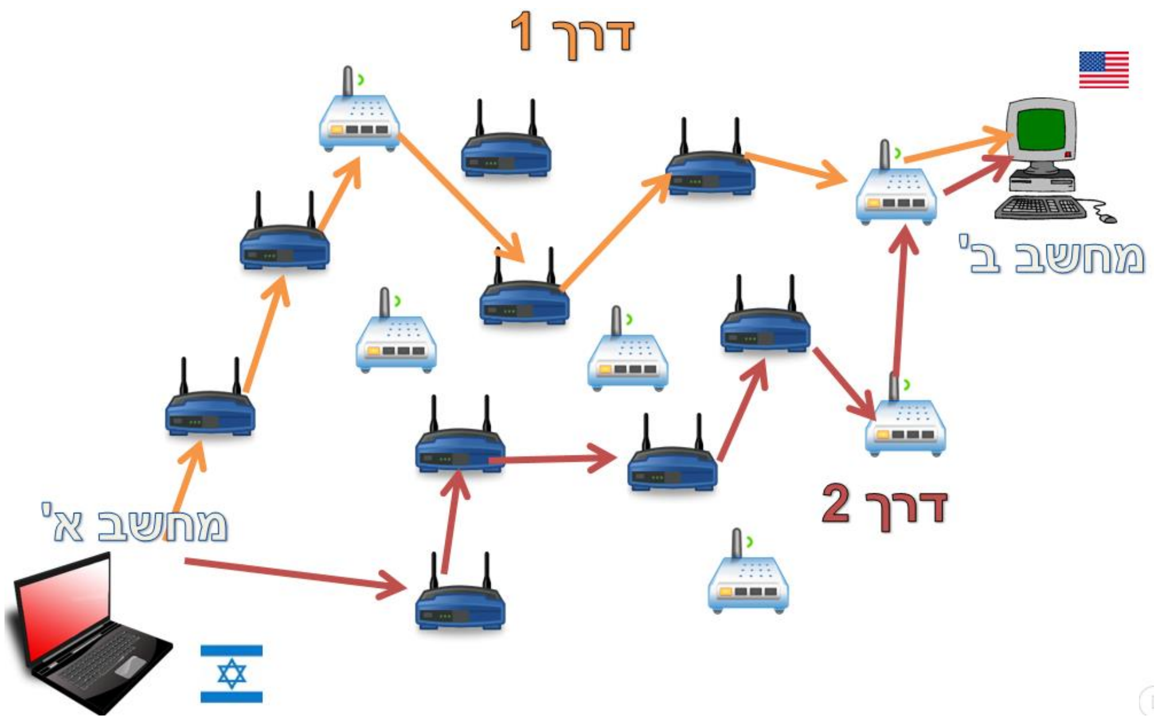
כעת, נסו בעצמכם להשלים את הטבלה הבאה לפי הפלט של הפקודה **tracert** במחשבכם:

<u>תחנות בדרך בין המחשב שלכם אל Facebook</u>

מספר תחנה	כתובת
1	
2	

מתרגיל זה אנו לומדים על מטרה נוספת חשובה של שכבת הרשת: **ניתוב (Routing)**. הכוונה היא החלטה על הדרך שבה יש להגיע מנקודה א' לנקודה ב'. הדבר דומה למציאת דרך נסיעה ברכב, למשל כמו שעושה האפליקציה Waze. על האפליקציה להבין מה הדרך שבה על הרכב (או במקרה שלנו – חבילת המידע) לעבור כדי להגיע מהמקור אל היעד.

נביט שוב בשרטוט הרשת הקודם שלנו:



כאן מוצגות שתי דרכים שונות בהן יכולה לעבור חבילת מידע ממחשב א' למחשב ב': דרך 1, המסומנת ב**כתום**, ודרך 2, המסומנת ב**אדום**. אם נחזור לעולם המושגים של Waze, החיצים מסמנים למעשה כבישים בהם הרכב יכול לעבור. כמובן שניתן לבחור בהרבה דרכים אחרות ואין מניעה לכך. על שכבת הרשת להחליט באיזה דרך להעביר כל חבילת מידע שהגיעה אליה, והיא יכולה לבחור בכל דרך שתרצה.

מה צריכה שכבת הרשת לדעת בכדי להחליט כיצד לנתב?

נסו לחשוב על כך בעצמכם בטרם תמשיכו בקריאה.

ראשית, על שכבת הרשת להכיר את מבנה הרשת. אם שכבת הרשת תדע על כל הרכיבים שנמצאים בציור, היא תוכל להחליט על דרך מלאה אותה יש לעבור כדי להגיע ממחשב א' למחשב ב'. קל להקביל זאת למציאת דרך נסיעה ברכב: על מנת ש-Waze תוכל לדעת מה הדרך הטובה ביותר להגיע מתל אביב לירושלים, עליה להכיר את כל הכבישים במדינה.

שנית, על שכבת הרשת לדעת האם קיים בכלל חיבור בין המקור ליעד. אם כל הכבישים בין תל אביב לירושלים חסומים כרגע, עדיף ש-Waze יגיד לנו להישאר בבית. כך גם צריכה שכבת הרשת לעשות: אם לא קיים כרגע אף חיבור בין מחשב א' למחשב ב', עליה להודיע למחשב א' שהיא אינה מסוגלת להעביר את חבילת המידע שלו.

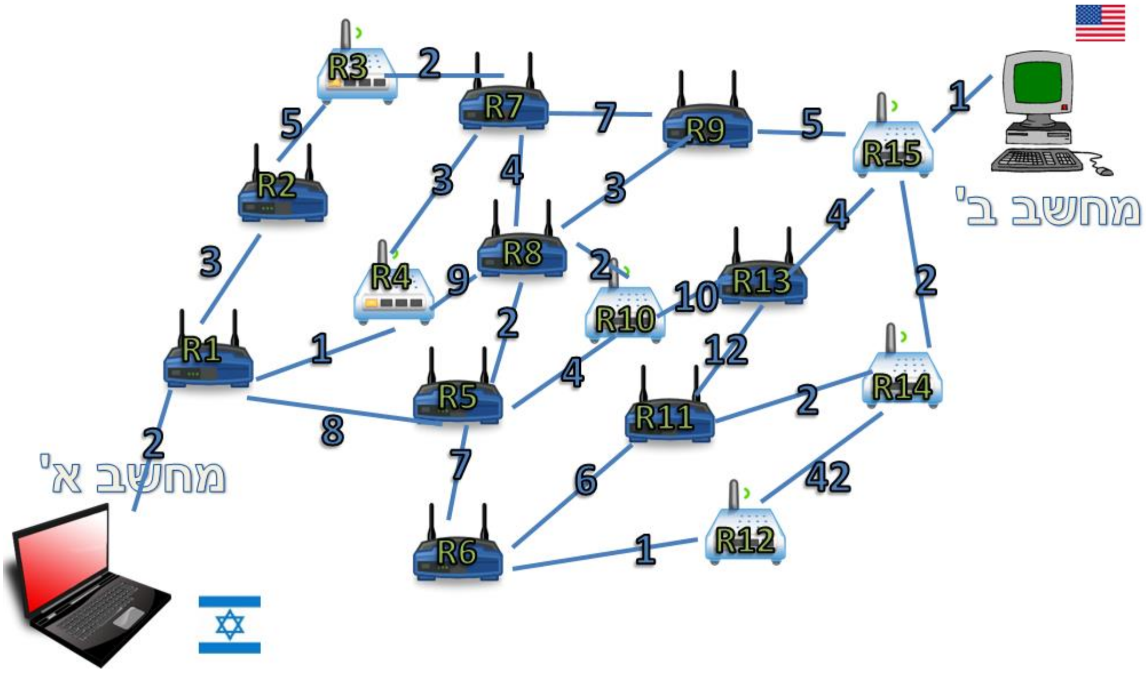
שלישית, שכבת הרשת צריכה להבין מהי הדרך הכי מהירה. שוב, בדומה ל-Waze, המטרה של השכבה היא לאפשר לחבילת המידע להגיע בדרך המהירה ביותר.

רביעית, באחריות שכבת הרשת להבין אילו דרכים אסורות. כפי שייתכן שכביש מסוים חסום כרגע, או שאי אפשר לעבור בו בגלל סכנת מפולת, כך גם בעולם הרשת – ישנם נתיבים אשר לא ניתן לעבור בהם.

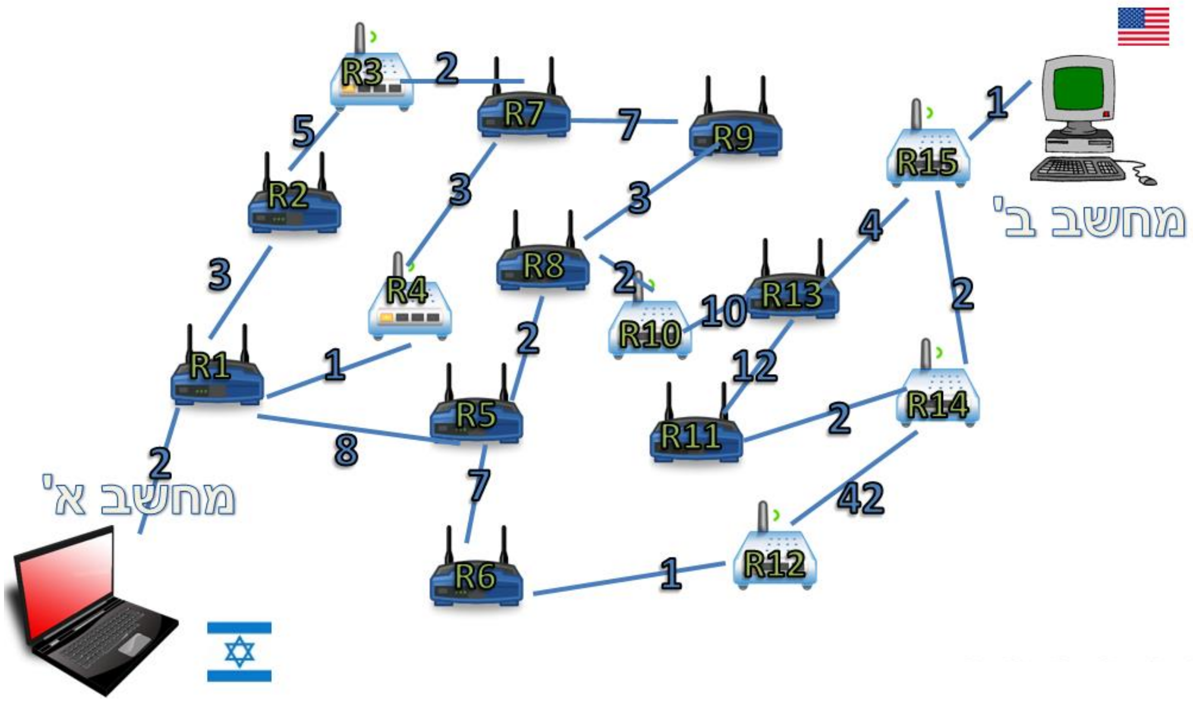
עשה זאת בעצמך – תכנון מסלול ניתוב

לפניכם שוב שרטוט הרשת הקודם. הפעם, לכל רכיב בדרך יש אות ומספר שמזהים אותו (לדוגמה: R1), ולכל קישור בינו לבין רכיב אחר יש "עלות". "עלות" זו יכולה להיקבע על פי גורמים שונים – למשל מרחק פיזי, איכות הקו, סוג הקישור (קווי, אלחוטי) ועוד. כך למשל, שליחת חבילה ממחשב א' אל R1 "עולה" 2, שליחת חבילה מ-R1 אל R2 "עולה" 3, ולכן שליחת חבילה ממחשב א' אל R2 דרך R1 "עולה" 5.

מצאו את הדרך ה"זולה" ביותר להגעה ממחשב א' אל מחשב ב'. לאיזו עלות הגעתם? באיזה רכיבים עברתם בדרך?



תקלות שונות גרמו לכך שחלק מהקישורים בין הרכיבים כבר לא פעילים. הסתכלו בתמונת הרשת החדשה, ומצאו שוב את הדרך ה"זולה" ביותר להגעה ממחשב א' למחשב ב':



האם הדרך השתנתה?

כך הבנו את אחת הסיבות ששכבת הרשת משנה את החלטת הניתוב עבור כל פקטה (חבילה). בכל פעם גם מצב הרשת משתנה, ולכן עלינו לנהוג בהתאם. בהקבלה לנסיעה ברכב, זכרו כי Waze עשויה לבחור עבורנו דרך שונה להגיע מביתנו לבית הספר – שכן עכשיו יש עומסי תנועה בדרך מסוימת, ודרך אחרת חסומה בשל עבודות בכביש.

פרוטוקול IP

הבנו מדוע צריך את שכבת הרשת, ולפחות חלק מהתפקידים שלה. הבנו שבאחריותה להעביר חבילות מידע מישות אחת ברשת לישות אחרת, וכן הבנו שהיא אחראית על ניתוב החבילות.

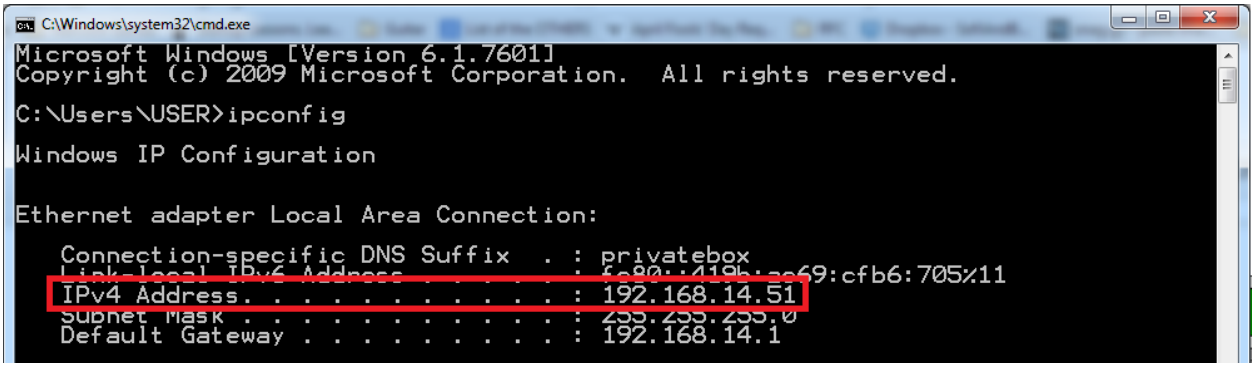
עכשיו הגיע הזמן לראות קצת איך הדברים קורים במציאות. לשם כך, נכיר את IP (Internet Protocol), הפרוטוקול של האינטרנט.

כתובות IP

מה כתובת ה-IP שלי?

ראשית נגלה מה כתובת ה-IP שלנו. לשם כך, היכנסו ל-Command Line (CMD), והקישו את הפקודה: **ipconfig**.

הפלט שלכם יהיה דומה לפלט הבא:



כתובת ה-IP של המחשב ממנו הורצה הפקודה מסומנת במלבן ה**אדום**, והיא הכתובת 192.168.14.51. מהי כתובת ה-IP שלכם?

מה זו כתובת IP?

דיברנו במהלך הספר לא מעט על כתובות IP. כעת אנו יכולים לציין כי כתובות IP הן כתובות של שכבת הרשת – שכבה זו משתמשת בהן כדי לדעת מה הכתובת של הישויות השונות ברשת.[39]
חשוב להבין שכתובת IP היא כתובת **לוגית** בלבד, בניגוד לכתובת פיזית. כלומר, זו לא כתובת ש"צרובה" על כרטיס הרשת, אלא עשויה להשתנות עם הזמן ולהתחלף.

כתובת IP, כפי שכבר ראינו, מיוצגת באמצעות ארבעה בתים (bytes)[40], ולכן תיראה כצירוף של ארבעה מספרים, שכל אחד נע בין הערכים 0 ו-255. להלן דוגמה לכתובת IP: 192.168.2.5

חשבו: האם הכתובת הבאה הינה כתובת IP תקינה?

192.168.275.2

התשובה היא – לא. בגלל ש-275 הינו מספר גדול מדי (גדול יותר מ-255, ולכן אינו נכנס בגודל של בית אחד), הוא לא יכול להוות חלק מכתובת ה-IP.

[39] יכולות להיות כתובות אחרות, במידה שלא משתמשים ב-IP אלא בפרוטוקול אחר. לצורך הנוחות, נניח לאורך הפרק שהשימוש הוא תמיד בפרוטוקול IP בתור הפרוטוקול של שכבת הרשת.

[40] הדבר נכון כמובן רק עבור כתובת IP בגרסה 4 (IPv4), ולא עבור גרסאות אחרות (למשל IPv6). מטעמי נוחות, במהלך הספר, נתייחס לפרוטוקול IPv4 בשם "IP".

כמה כתובות IP אפשריות קיימות?

מכיוון שכתובת IP מיוצגת על ידי ארבעה בתים (bytes), היא למעשה מיוצגת על ידי 32 ביטים (bits), שכל אחד מהם יכול להיות 0 או 1. אי לכך, ישנן 2^{32} אפשרויות לכתובות IP שונות.

כתובת IP מחולקת לשני חלקים:

- **מזהה רשת (Network ID)** – לאיזו רשת שייכת כתובת ה-IP הזו? לדוגמה: האם היא חלק מהרשת של בית הספר?
- **מזהה ישות (Host ID)** – לאיזה כרטיס הרשת שייכת הכתובת הזו, בתוך הרשת[41]? למשל – האם היא שייכת לתלמיד משה או לתלמיד אהרון?

ניקח לדוגמה את הכתובת הבאה: 200.100.0.1 המשויכת למחשב מסוים. נקרא למחשב זה "מחשביל"ה".
נאמר שהגדרנו את שני הבתים הראשונים (200.100 המסומנים ב**אדום**) בתור מזהה הרשת, ושני הבתים הבאים (0.1 המסומנים ב**כחול**) בתור מזהה הישות. מכאן שכל כתובת שתתחיל ב-200.100 תתאר ישות שנמצאת באותה הרשת, וכל כתובת אחרת – לא.

עבור כל אחת מכתובות ה-IP הבאות, כתבו האם היא נמצאת באותה הרשת של מחשביל"ה, או שמא ברשת נפרדת:

1. 200.100.0.2
2. 200.100.2.0
3. 100.200.0.5
4. 200.100.200.100
5. 1.2.3.4
6. 200.200.100.100
7. 1.0.200.100

פתרון מודרך – בדקו את עצמכם

לאחר שכתבתם את תשובותיכם בסעיף הקודם, בואו נביט ביחד בכתובות ונסמן את מזהה הרשת:

1. 200.100.0.2 – מזהה הרשת זהה למזהה של מחשביל"ה, ולכן מדובר בכתובת שנמצאת באותה הרשת.

[41] להזכירכם, ישות יכולה להיות מחשב, נתב, שרת או רכיב אחר. באופן ספציפי, מזהה הישות מתייחס לכרטיס רשת מסויים באותו מחשב, נתב, שרת או רכיב.

2. 200.100.2.0 – מזהה הרשת זהה למזהה של מחשבילי"ה, ולכן מדובר בכתובת שנמצאת באותה הרשת.
3. 100.200.0.5 – מזהה הרשת שונה, ולכן מדובר בכתובת שלא נמצאת באותה הרשת! שימו לב כי אין זה משנה אם החליפו את הסדר או עשו כל דבר אחר. כל עוד מזהה הרשת לא זהה לחלוטין (בדוגמה זו 200.100) – מדובר ברשת אחרת.
4. 200.100.200.100 – מזהה הרשת זהה למזהה של מחשבילי"ה, ולכן מדובר בכתובת שנמצאת באותה הרשת.
5. 1.2.3.4 – מזהה הרשת שונה, ולכן מדובר בכתובת שלא נמצאת באותה הרשת!
6. 200.200.100.100 – מזהה הרשת שונה, ולכן מדובר בכתובת שלא נמצאת באותה הרשת!
7. 1.0.200.100 – מזהה הרשת שונה, ולכן מדובר בכתובת שלא נמצאת באותה הרשת!

כמו שראינו, אם אנו יודעים מהו מזהה הרשת, אנו יכולים לדעת אילו ישויות (מחשבים, נתבים, שרתים ועוד) נמצאות באותה הרשת. לכל ישות יהיה **מזהה ישות** שונה. כך למשל, במקרה של המחשב מחשבילי"ה, כתובתו הייתה כזכור: 200.100.0.1, ומזהה הרשת שלו היה 200.100. מכאן שמזהה הישות שלו ברשת הינו: 0.1. נאמר ויש ברשת של מחשבילי"ה ישות נוספת, למשל המדפסת של מחשבילי"ה, הנקראת כצפוי מדפסתל"ה. מזהה הישות של מדפסתל"ה צריך להיות שונה מזה של מחשבילי"ה, והוא יוכל להיות למשל: 0.2. כך תהיה כתובתה המלאה: 200.100.0.2.

- הכתובת של מחשבילי"ה היא: 200.100.0.1
- הכתובת של מדפסתל"ה היא: 200.100.0.2

בגלל שלמחשבילי"ה ולמדפסתל"ה יש את אותו מזהה הרשת (200.100), אנחנו יודעים שהם נמצאים באותה הרשת. עם זאת, מכיוון שלכל אחד מהם יש מזהה ישות שונה, אנו יכולים לפנות אליהם בנפרד ולדעת האם המידע שאנו שולחים מיועד למחשבילי"ה או למדפסתל"ה.

בדוגמה לעיל ראינו מזהה רשת בגודל של שני בתים. שימו לב כי מזהי רשת יכולים להיות בגודל משתנה. לדוגמה, יכולנו גם להגדיר את כתובתו של מחשבילי"ה כך:

200.100.0.1

כלומר, הבית הראשון (200) מייצג את מזהה הרשת, ושלושת הבתים האחרים (100.0.1) את מזהה הישות. במקרה שמזהה הרשת הוגדר כך, הרי שכל כתובת IP שמתחילה בערך 200 מייצגת כתובת באותה הרשת. כך למשל, הכתובת הבאה: 200.50.2.3, נמצאת בכתובת של מחשבילי"ה. זאת בניגוד להגדרה הקודמת של כתובתו של מחשבילי"ה, שבה כתובת הייתה צריכה להתחיל ברצף הבתים 200.100 בכדי להיות חלק מן הרשת.

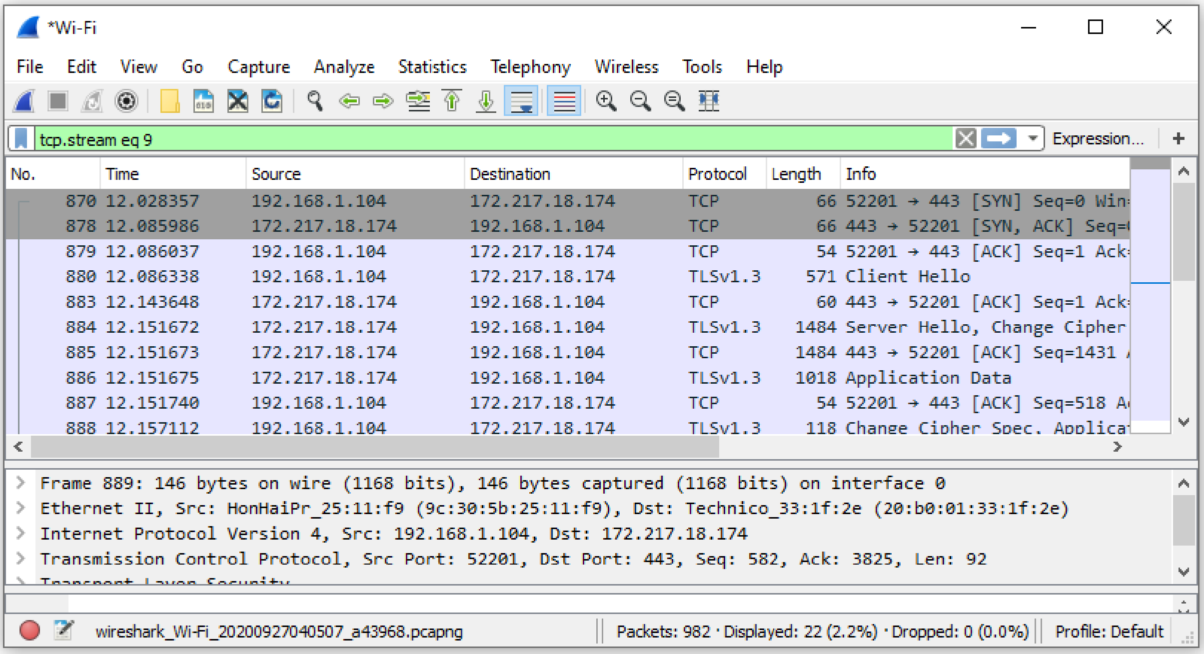
תרגיל 7.2 מודרך – מציאת כתובת ה-IP באמצעות הסנפה

מוקדם יותר בפרק, מצאתם את כתובת ה-IP שלכם באמצעות הפקודה ipconfig. על מנת לוודא שהכתובת שמצאתם היא הכתובת הנכונה, נשלח בקשה לאתר חיצוני (למשל ל-Google), ונסניף אותה באמצעות Wireshark. כאן נוכל כבר להסתכל לראשונה על חבילת IP.

בואו נעשה זאת יחדיו. פתחו את Wireshark והתחילו להסניף. השתמשו במסנן התצוגה (display filter) הבא: "tcp.port == 443".

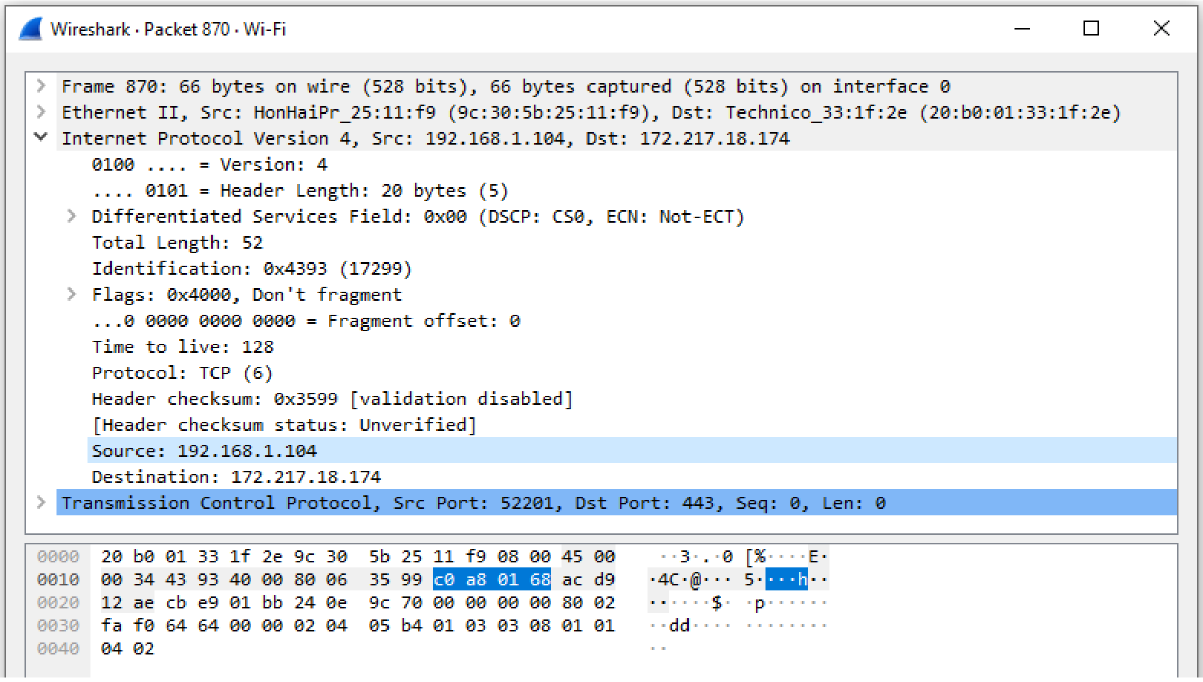
לאחר מכן, פתחו את הדפדפן האהוב עליכם, וגלשו אל הכתובת www.google.com. כעת, הפסיקו את ההסנפה. מיצאו את החבילה הראשונה שנשלחת אל גוגל (באופן ידני או באמצעות פילטור של פקטות SYN כפי שלמדנו) ולבסוף בצעו Follow TCP Stream.

החלון שלכם אמור להיראות בערך כך:



כבר עתה, ניתן לראות בפקטה את כתובת המקור תחת העמודה Source. שימו לב כי אכן מדובר בכתובת אותה מצאתם קודם לכן, באמצעות **ipconfig**.

כעת נסתכל גם בפקטה עצמה. הסתכלו בשדות השונים ב-Wireshark, ופתחו את שכבת הרשת, היכן שכתוב Internet Protocol:



לא נסתכל על כל השדות עכשיו, אך נשים לב שבשדה ה-Source (כתובת המקור) אכן מצוינת כתובת ה-IP שראינו קודם.

כתובת היעד (Destination) של החבילה היא כמובן כתובת ה-IP של www.google.com. כך אנו רואים שבאמת הגלישה בוצעה מהמחשב שלנו (המקור) אל Google (היעד).

מה חסר לנו?

אז גילינו את כתובת ה-IP שלנו. עם זאת, משהו עדיין חסר. בהינתן החומר שלמדנו בינתיים, נסו לחשוב איזה פרט מידע חסר לנו לפני שתמשיכו לקרוא את השורה הבאה.

ובכן, כעת ברשותנו כתובת ה-IP המלאה שלנו. עם זאת, כפי שלמדנו, הכתובת מחולקת למזהה רשת ולמזהה ישות (במקרה זה – מזהה המחשב שלנו בתוך הרשת). כיצד נדע מהם המזהים?

לדוגמה, כיצד נדע האם המחשב בעל הכתובת 192.168.0.5 נמצא איתנו באותה הרשת? כלומר – עלינו לדעת, מתוך כתובת ה-IP שלנו, מהו מזהה הרשת ומהו מזהה הישות. מכאן שעלינו להבין אילו מהביטים מגדירים את מזהה הרשת, ואילו מהביטים מגדירים את מזהה הישות.

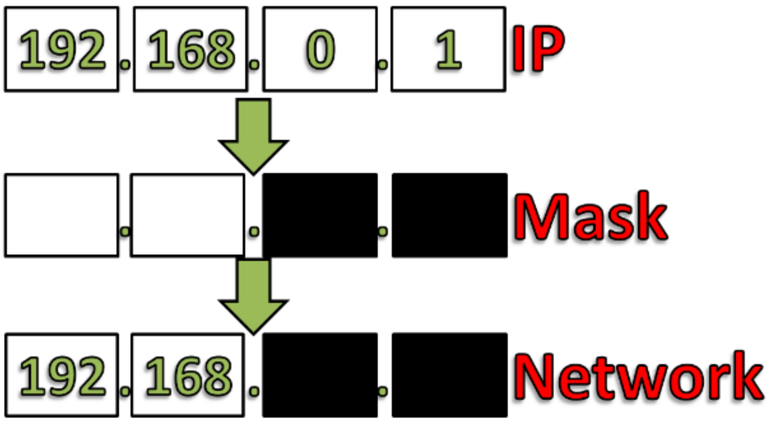
מהו מזהה הרשת שלי? מהו מזהה הישות?

כדי לענות על שאלה זו, עלינו ללמוד מונח חדש בשם **Subnet Mask (מסיכת רשת)**. עבור כל כתובת IP, עלינו לדעת מהי ה-Subnet Mask שלה, על מנת לדעת מהו מזהה הרשת. ה-Subnet Mask מגדיר כמה ביטים (bits) מתוך כתובת ה-IP מייצגים את מזהה הרשת.

נשתמש בדוגמה. הביטו בכתובת הבאה: 192.168.0.1. נאמר שמסיכת הרשת שלה מוגדרת כ-16 ביטים (או שני בתים[42]). מכאן ש-192.168 הינו מזהה הרשת, ו-0.1 הינו מזהה הישות. מקרה זה ניתן להציג בדרכים שונות:

192.168.0.1/16 – הוספת "16/" בסוף הכתובת מציינת שה-Subnet Mask מכיל 16 ביטים, כלומר שזהו מזהה הרשת הרלוונטי.

דרך נוספת היא לציין שהכתובת היא 192.168.0.1, וה-Subnet Mask הינו 255.255.0.0 (16 הביטים הראשונים דולקים, ולכן הם מזהה הרשת. 16 הביטים הבאים כבויים, ולכן הם מזהה הישות).בדוגמה הזו, המסכה "נראית" ככה:



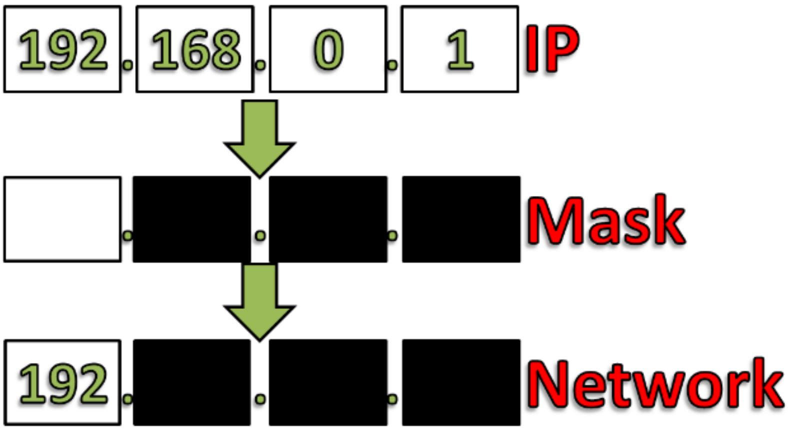
כלומר, המסכה גורמת לנו להבין שרק 16 הביטים (שני הבתים) הראשונים הם הרלוונטים עבור מזהה הרשת, ובכך "מעלימה" את 16 הביטים (שני הבתים) הנותרים.

בואו נראה דוגמאות נוספות:

עבור הכתובת 192.168.0.1/8, ה"מסכה" נראית ככה:

---

[42] אם אינכם מרגישים עדיין בטוחים במונחים "ביטים" ו-"בתים" אל תדאגו, הביטחון נרכש עם הזמן. עם זאת, קראו לאט וודאו כי אתם מבינים את הכוונה בדוגמאות שניתנות לפניכם.



כלומר, המסכה גורמת לנו להבין שרק 8 הביטים הראשונים (כלומר הבית הראשון) הם הרלוונטים עבור מזהה הרשת, ובכך "מעלימה" את 24 הביטים (שלושת הבתים) הנותרים.

חשבו על השאלות הבאות עבור כתובת ומסכה אלו (192.168.0.1/8):

- האם הכתובת 192.168.0.2 נמצאת באותה הרשת? התשובה היא כן – הרי יש לה את אותו מזהה הרשת (192).
- האם הכתובת 192.5.0.2 נמצאת באותה הרשת? התשובה היא כן – הרי יש לה את אותו מזהה הרשת (192). שימו לב שדבר זה נכון כיוון שמזהה הרשת כולל רק 8 ביטים, כלומר את 192, ולא מתחשב למעשה בבית השני, המכיל את הערך 168.
- האם הכתובת 100.200.0.1 נמצאת באותה הרשת? התשובה היא לא – כיוון שמזהה הרשת שונה (לא 192).

את אותה הכתובת עם מסיכת הרשת ניתן היה גם להציג כך: הכתובת 192.168.0.1, המסכה: 255.0.0.0.

שימו לב: עולה לנו כאן נקודה מעניינת. מחשב בעל כתובת ה-IP הבאה: 192.160.0.1, הינו חלק מאותה הרשת של המחשב 192.168.0.2/8, אך לא חלק מאותה הרשת של המחשב 192.168.0.2/16. מכאן שעל מנת לדעת אילו ישויות נמצאות באותה הרשת, עלינו להבין גם את ה- Subnet Mask, ולא רק את כתובת ה-IP.

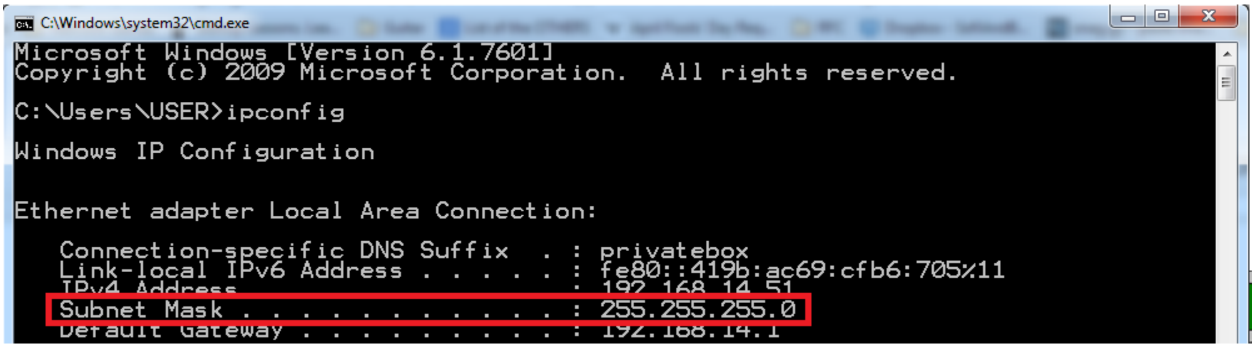
הערה: מזהה הרשת מוגדר באמצעות מספר ביטים (bits) ולא בתים (bytes), ולכן מסכת רשת יכולה להיות מוגדרת לא רק כמספר שמגדיר בתים שלמים (8, 16, 24), אלא גם באמצעות מספר ביטים בודד (למשל 9 או 23). לא נתעכב על נקודה זו בספר זה.

נחזור לשאלה שהצגנו קודם:

מהו מזהה הרשת שלי? מהו מזהה הישות?

נסו לענות על כך בעצמכם.

נסתכל כעת בדוגמה שהצגנו קודם, באמצעות הפקודה **ipconfig**:


```
C:\Windows\system32\cmd.exe
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation.  All rights reserved.

C:\Users\USER>ipconfig

Windows IP Configuration


Ethernet adapter Local Area Connection:

   Connection-specific DNS Suffix  . : privatebox
   Link-local IPv6 Address . . . . . : fe80::419b:ac69:cfb6:705%11
   IPv4 Address. . . . . . . . . . . : 192.168.14.51
   Subnet Mask . . . . . . . . . . . : 255.255.255.0
   Default Gateway . . . . . . . . . : 192.168.14.1
```

אנו רואים שמסכת הרשת שלנו היא 255.255.255.0, כלומר שמזהה הרשת מוגדר באמצעות 24 ביטים (או 3 בתים).

נחזור לשאלה נוספת ששאלנו מוקדם יותר: האם מחשב בעל הכתובת 192.168.0.5 נמצא איתנו באותה הרשת?

התשובה היא – לא. זאת מכיוון שמזהה הרשת שלנו מוגדר באמצעות 24 ביטים, והוא למעשה 192.168.14. הכתובת שהצגנו אינה מכילה את מזהה רשת זה, ולכן המחשב בעל הכתובת הזו אינו נמצא באותה הרשת כמו המחשב שעליו הרצנו את הפקודה **ipconfig**.

מצאו את כתובת ה-IP שלכם ואת ה-Subnet Mask שלכם. כתבו כתובת IP אחת שנמצאת אתכם באותה הרשת, וכתובת IP אחת שלא נמצאת אתכם באותה הרשת.

כעת הסניפו את הרשת שלכם במשך חמש דקות. הסתכלו בקובץ ההסנפה, ומצאו את כתובות ה-IP השונות שבו. אילו כתובות נמצאות ברשת שלכם? אילו כתובות לא?

כתובות IP מיוחדות

ישנן כמה כתובות IP שמוגדרות כ"כתובות מיוחדות", ושווה להכיר אותן. כרגע, נכיר רק חלק מהן:

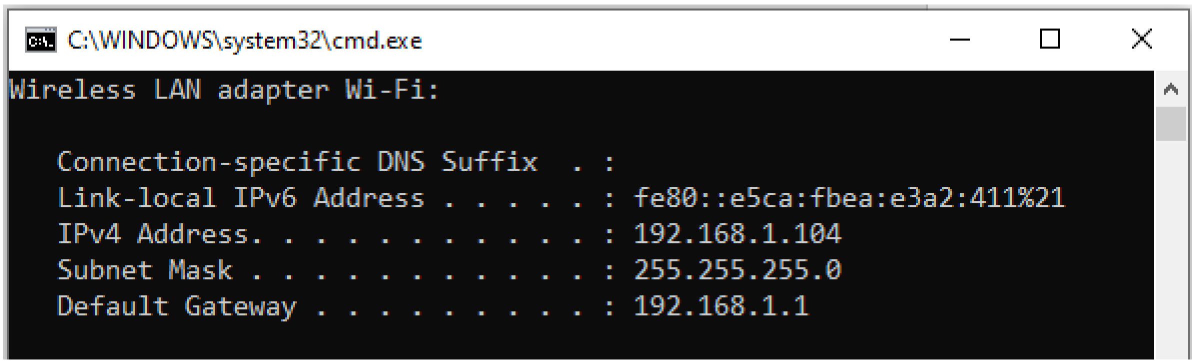
- 255.255.255.255 – כתובת זו היא כתובת Broadcast. הכוונה היא שחבילה שנשלחת לכתובת זו מיועדת לכלל הישויות ברשת. לדוגמה: ברשת בה יש ארבעה מחשבים: של דני, דנה, אורית ואורי, אם דני שולח חבילה לכתובת היעד "255.255.255.255", היא תגיע לדנה, אורית ואורי – כלומר לכל שאר המחשבים ברשת.
- 127.0.0.0/8 – כתובות "Loopback", הנקראות גם "Local Host". כתובות אלו מציינות למעשה שהחבילה לא צריכה לעזוב את כרטיס הרשת, אלא "להישאר במחשב" (בפועל – נשלחת לכרטיס הרשת הווירטואלי של מערכת ההפעלה). הכתובת המוכרת ביותר הנמצאת בטווח זה היא הכתובת

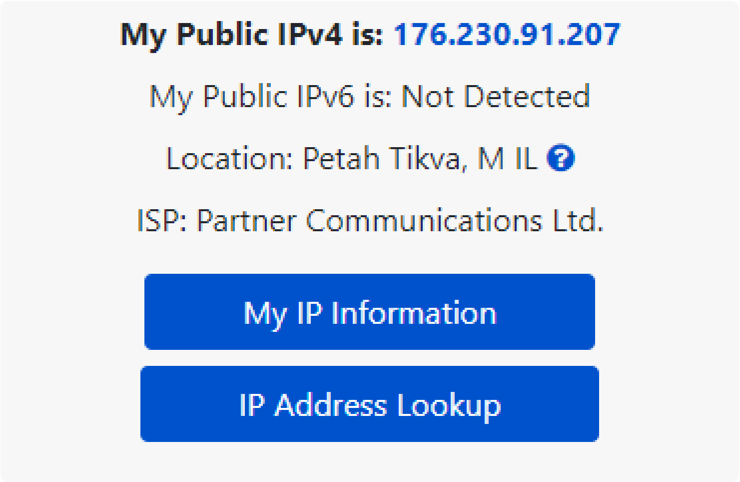
127.0.0.1, אך מכיוון שמזהה הרשת הינו בגודל 8 ביטים (או בית אחד), לכתובת 127.5.6.7 (לדוגמה) יש את אותה המשמעות.

כאמור, ישנן כתובות מיוחדות נוספות עליהן לא נרחיב בשלב זה.

כתובות פרטיות ו-NAT

בואו נבצע ניסוי קטן. בצעו ipconfig ובידקו מהי כתובת ה-IP שלכם. כעת הכנסו לאתר https://www.whatismyip.com/ ושם תמצאו גם כן את כתובת ה-IP שלכם. הפלא ופלא, הכתובות אינן זהות... שימו לב לתוצאות שהתקבלו מהמחשב של כותב שורות אלה:





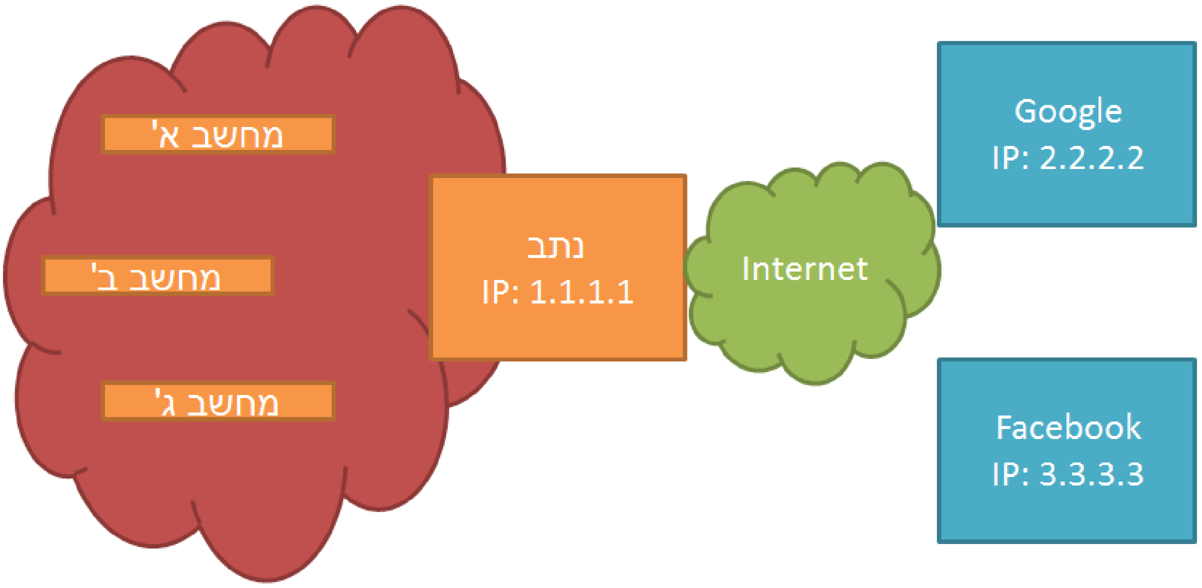
איך יכול להיות שמצד אחד כתובת ה-IP של המחשב היא 192.168.1.104 ומצד שני כתובת ה-IP היא 176.230.91.207? כדי לענות על השאלה הזו נתחיל מהבסיס.

החל מסוף שנות ה-80' – נוצרה בעולם בעיה אמיתית ומוחשית – נגמרו כתובות ה-IP. מסיבות שונות, נוצר מצב שבו למרות ש-IPv4 מספק כמעט 4.3 מיליארד[43] כתובות שונות, התבצעה הקצאה לא יעילה שלהן ולא נותרו כתובות IP שניתן היה להקצות לרכיבי רשת חדשים שהזדקקו להן. בתחילת שנות

[43] מכיוון שבארבעה בתים (bytes) יש 32 ביטים (bits), שכל אחד מהם יכול להיות 0 או 1. אי לכך, מספר האפשרויות הינו 2^{32}.

ה-90', כשהאינטרנט זכה לגדילה מהירה מאוד, המחסור בכתובות ה-IP החל לפגוע בספקיות אינטרנט שפשוט לא יכלו להקצות כתובות IP ללקוחות שלהן[44].

נוצר צורך למצוא פתרון מהיר לבעיה. לפתרון הזה, קוראים **NAT (Network Address Translation)**. על מנת להסביר את הרעיון הכללי, נביט בתמונת הרשת הבאה:



משמאל לפנינו נמצאת "הרשת האדומה", ובה שלושה מחשבים. הרשת מחוברת, באמצעות הנתב בעל כתובת ה-IP של 1.1.1.1, אל האינטרנט. בנוסף, יש שרתים של Google ו-Facebook אליהם ירצו המחשבים ברשת לגשת.

עד אשר החל השימוש ב-NAT, היה צורך לספק כתובת IP יחודית לכל אחת מהישויות. כלומר שמחשב א', מחשב ב' ומחשב ג', יזכו כל אחד לכתובת IP אמיתית וייחודית בעולם. דבר זה חיובי מהרבה בחינות, אך במציאות בה אין כבר כתובות IP לתת – הדבר לא ייתכן. אי לכך, נוצר הרעיון של NAT. לפי רעיון זה, כל הישויות בתוך הרשת – מחשב א', מחשב ב' ומחשב ג', יקבלו **כתובות פרטיות** – כלומר כתובות שיזהו אותן בתוך הרשת בלבד, ולא בעולם החיצוני. כתובות אלו אינן ניתנות לניתוב – כלומר, נתב באינטרנט שרואה חבילה שמיועדת לכתובת שכזו עתיד "לזרוק" אותה.

לשם כך, הוגדרו שלושה טווחים של כתובות פרטיות:

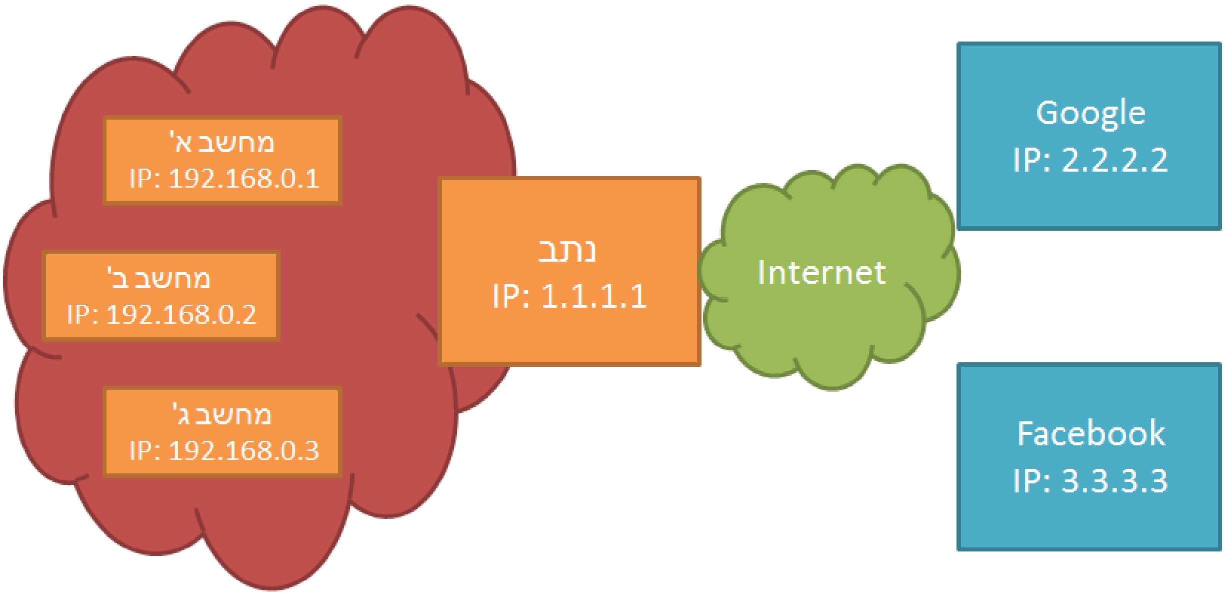
- 10.0.0.0/8 – בטווח זה יש 16,777,216 כתובות.
- 172.16.0.0/12 – בטווח זה יש 1,048,576 כתובות.

---

[44] לקריאה נוספת על תופעה זו – קראו בעמוד: http://en.wikipedia.org/wiki/IPv4_address_exhaustion.

- 192.168.0.0/16 – בטווח זה יש 65,536 כתובות.

לצורך הדוגמה, הרשת שלנו עכשיו תיראה כך:



בצורה זו, הרשת האדומה "בזבזה" רק כתובת IP אחת – זו של הנתב שלה, ולא ארבע כתובות כמו שהיא הייתה צורכת לפני השימוש ב-NAT.

עם זאת, כיצד תצליח הרשת לעבוד? כיצד יצליח עכשיו לפנות אל Google מחשב א', שהינו בעל כתובת פרטית שאסור לנתב? חמור מכך – כיצד Google יצליח להחזיר תשובה אל כתובת IP פרטית שאסור לנתב, וכן שייכת להרבה רכיבים שונים ברחבי העולם?

באופן כללי, התהליך יעבוד כך:

בשלב הראשון, מחשב א' ישלח הודעה ממנו (כתובת המקור: 192.168.0.1) אל Google (כתובת היעד: 2.2.2.2). את החבילה הוא ישלח אל הנתב.

בשלב השני, הנתב[45] יקבל את החבילה, **ויחליף בה את כתובת המקור לכתובת שלו**. כלומר, החבילה עכשיו תישלח מכתובת המקור 1.1.1.1, אל כתובת היעד 2.2.2.2.

בשלב השלישי, השרת של Google יקבל את החבילה. שימו לב: השרת של Google כלל לא מודע לכתובת ה-IP של מחשב א', או לעובדה שישנה ישות רשת מאחורי הנתב. הוא מודע אך ורק לישות הרשת בעלת

---

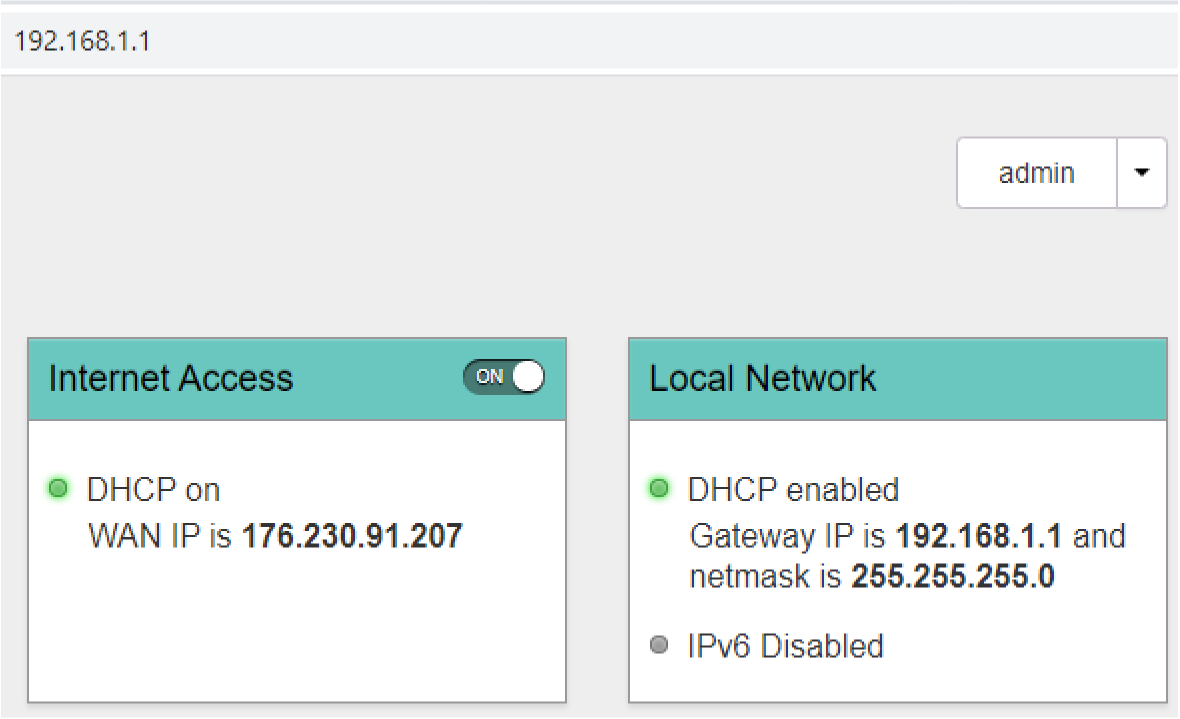
[45] מימוש ה-NAT לא חייב להתבצע אצל הנתב של הרשת. עם זאת, בפועל, ברוב המקרים הנתב הוא אכן זה שמממש את ה-NAT.

הכתובת 1.1.1.1, וזהו הנתב של הרשת. כעת, Google יעבד את הבקשה של מחשב א', וישיב עליה – אל הנתב. כאמור, מבחינתו של Google, הוא קיבל את הבקשה באופן ישיר מן הנתב.
בשלב הרביעי, הנתב יקבל את התשובה מהשרת של Google. החבילה תכיל את כתובת המקור של השרת של Google (כלומר 2.2.2.2), וכתובת היעד של הנתב (1.1.1.1). כעת, הנתב יבין שמדובר בחבילה המיועדת למעשה אל מחשב א'. אי לכך, הוא יבצע **החלפה של כתובת היעד**. כלומר, הוא ישנה את כתובת היעד מ-1.1.1.1 ל-192.168.0.1, ויעביר את החבילה למחשב א'.

בשלב החמישי, מחשב א' יקבל את הודעת התשובה מ-Google. מבחינת מחשב א', החבילה הגיעה מכתובת ה-IP של Google (שהיא 2.2.2.2), היישר אל הכתובת שלו (192.168.0.1), ולכן מבחינתו מדובר בתהליך "רגיל" לכל דבר.

בצורה זו מצליחות ישויות רשת מתוך הרשת האדומה שלנו לתקשר עם רכיבים מחוץ לאינטרנט, על אף שאין להם כתובת IP יחודית. עם זאת, ישנה סוגיה לא פתורה – כיצד, בשלב הרביעי שהצגנו, יודע הנתב שהחבילה שהגיעה מ-Google מיועדת למעשה למחשב א' ולא למחשב ב'? כיצד הוא יידע לעשות זאת במידה שמחשב א' ומחשב ב' פנו שניהם, באותו הזמן בדיוק, אל Google? על מימושים שונים של NAT לא נתעכב בספר זה, אך אתם מוזמנים להרחיב על כך בעמוד:
http://en.wikipedia.org/wiki/Network_address_translation

לאחר שהבנו את מנגנון פעולת ה-NAT, נחזור לדוגמה ממנה התחלנו ונברר מניין מגיעה כתובת ה-IP שנראתה באתר whatismyip.
כפי שראינו, המקום אותו יש לבדוק הוא כתובת ה-IP ממנה הראוטר הביתי שלנו גולש אל הרשת. אין לנו יכולת לראות את כתובת ה-IP הזו ברשת הפנימית הביתית שלנו, מכיוון שברשת הביתית הראוטר שלנו מוכר עם כתובת IP פנימית. כדי לבדוק מה כתובת ה-IP שאיתה הראוטר יוצא החוצה אל רשת האינטרנט, נזדקק להכנס אל תוך הראוטר.
על גבי הראוטר שלכם ישנה מדבקה עם כתובת IP וסיסמה. הזינו את כתובת ה-IP אל הדפדפן שלכם והכניסו את הסיסמה כאשר תתבקשו לעשות כך. תגיעו אל תוכנת הניהול של הראוטר הביתי שלכם.
להן דוגמה לתוכנת ניהול כזו, כמובן שתוכנת הניהול משתנה לפי הדגמים השונים של הראוטרים ולכן המסך שלכם צפוי להראות שונה, אך הרעיון הוא זהה.



קיבלנו מידע הן על הרשת המקומית והן על החיבור לאינטרנט.

ברשת המקומית, נכתב לנו כי כתובת ה-IP של ה-Gateway היא 192.168.1.1. זוהי גם כתובת ה-IP אותה הזנו בדפדפן שלנו כדי להגיע לממשק הניהול.

ברשת האינטרנט, כלומר ביציאה אל ספקית התקשורת שלנו, הראוטר משתמש בכתובת ה-IP שונה: 176.230.91.207. אמנם, זוהי בדיוק הכתובת שמוצגת לנו באתר whatismyip.

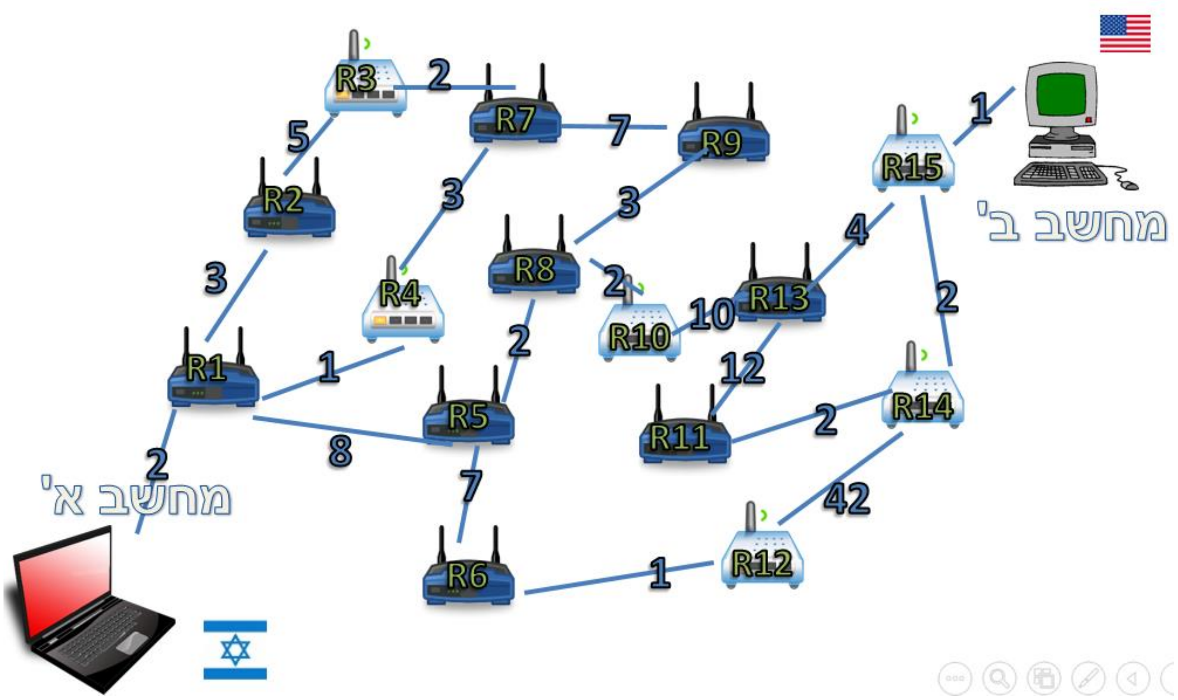
ניתוב

עכשיו כשאנו יודעים כיצד בנויה כתובת IP, הגיע הזמן להתרכז בתהליך הניתוב. הסברנו קודם, שמשמעות הניתוב היא ההחלטה על הדרך שבה חבילה תעבור בין שתי נקודות. אם נשתמש שוב בהקבלה לעולם הרכבים, הרי שתהליך הניתוב הוא ההחלטה על הדרך בה הרכב צריך לנסוע כדי להגיע מנקודת המוצא אל היעד. הרכיבים שמבצעים את רוב מלאכת הניתוב בעולם רשתות המחשבים, נקראים **נתבים**.

נתב (Router)

הנתב (Router) הוא רכיב רשתי בשכבה שלישית. מטרתו היא לקשר בין מחשבים ורשתות ברמת ה-IP.

ניזכר בתרשימי הרשת שהצגנו קודם:



כל רכיב בדרך כאן הינו למעשה נתב, ומכאן גם נובע הייצוג שלו (R1 הוא למעשה קיצור עבור "Router 1"). על הנתב לקבל כל חבילה, ולהחליט איך לנתב אותה הלאה בדרך הטובה ביותר.

ציינו שנתב הוא רכיב של שכבת הרשת. מה המשמעות של כך? לכל נתב יש כתובת IP משלו[46]. מעבר לכך, הנתב "מבין" את שכבת הרשת – הוא יודע מה היא כתובת IP, מכיר את מבנה חבילת ה-IP, קורא את ה-Header (תחילית) של החבילה ומקבל החלטות בהתאם.

---

[46] ישנם גם נתבים בפרוטוקולים שאינם פרוטוקולי IP. עם זאת, מטעם הנוחות, נניח במהלך הספר שכלל הנתבים הם נתבי IP.

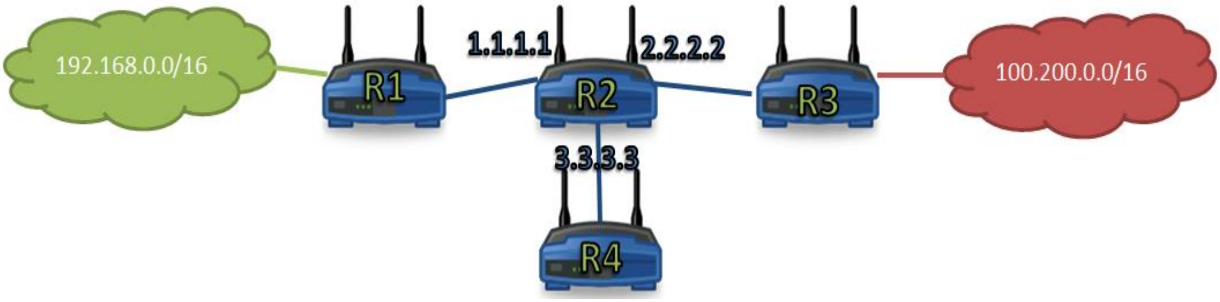
למעשה, כאשר ביצענו **tracert** קודם לכן וקיבלנו את הדרך שאותה עברה חבילה מהמחשב שלנו ואל www.facebook.com, קיבלנו את **רשימת הנתבים** אצלם עברה החבילה בדרך.
בקרוב נבין כיצד ניתן ליצור את רשימה זו, כלומר – איך **tracert** פועל.

כיצד נתב מחליט לאן לנתב?

נסתכל בתמונה לעיל, ונניח כי נתב R1 קיבל את החבילה של מחשב א' שנשלחה אל מחשב ב'. כיצד יודע הנתב R1 אם להעביר את החבילה אל הנתב R2, אל R4 או אל R5?

על מנת להחליט כיצד לנתב את החבילה, לכל נתב יש **טבלת ניתוב (Routing Table)** משלו. טבלה זו מתארת לאן יש להעביר כל חבילה שמגיעה אל הנתב. ברוב המקרים, הטבלה היא דינאמית – כלומר, היא יכולה להשתנות בהתאם למצב הרשת. היזכרו בתרגיל שביצענו קודם לכן בפרק זה, בו מצאנו את הדרך ה"זולה" ביותר להגיע מנקודה אחת ברשת לנקודה אחרת. עקב מספר תקלות – "מצב הרשת" השתנה, שכן חלק מהחיבורים לא היו תקינים עוד. בעקבות כך, הנתבים בדרך היו צריכים לשנות את דעתם על הרשת ולהחליט על דרך חדשה לנתב.

נביט לדוגמה בתמונת הרשת הבאה (הערה: מדובר בתמונת רשת חלקית בלבד):



בתמונת הרשת הזו ישנם ארבעה נתבים. הנתב R1 מחובר באופן ישיר לרשת בעלת המזהה 192.168.0.0/16. הנתב R3 מחובר באופן ישיר לרשת בעלת המזהה 100.200.0.0/16. הנתב R2 מחובר באופן ישיר לנתבים R1 ו-R3, וגם לנתב נוסף בשם R4.

שימו לב שנתב R2 נמצא למעשה בשלוש רשתות שונות: הרשת שלו ושל R3, הרשת שלו ושל R1, והרשת שלו ושל R4. עבור כל אחת מהרשתות האלה, ל-R2 יש כתובת IP אחרת.

- כתובת ה-IP של R2 ברשת שלו ושל R1 הינה: 1.1.1.1.
- כתובת ה-IP של R2 ברשת שלו ושל R3 הינה: 2.2.2.2.
- כתובת ה-IP של R2 ברשת שלו ושל R4 הינה: 3.3.3.3.

כל אחד מהנתבים עשוי להיות מחובר גם לנתבים נוספים, אך נתעלם מכך לצורך ההסבר.

אל הנתב R2 מגיעה חבילה, כשכתובת היעד שלה היא: 100.200.5.8. כיצד יידע הנתב R2 אל מי להעביר את החבילה? האם ל-R1, ל-R3 או ל-R4?

על מנת לענות על שאלה זו, נסתכל בטבלת הניתוב של נתב R2:

ממשק (Interface)	מסכת רשת (Network Mask)	יעד Network) (Destination	מספר שורה
3.3.3.3	0.0.0.0	0.0.0.0	1
1.1.1.1	255.255.0.0	192.168.0.0	2
2.2.2.2	255.255.0.0	100.200.0.0	3

על הנתב להסתכל בטבלת הניתוב, ולראות לאיזה מ**רשומות הניתוב** (שורות בטבלה) שלו היא מתאימה. את החיפוש שלו מבצע הנתב מלמטה למעלה.

נבחן יחד את הפעולה של הנתב. ראשית, הוא מסתכל ברשומה התחתונה ביותר – רשומה מספר שלוש, המייצגת את הרשת 100.200.0.0 עם המסכה 255.255.0.0. כעת הוא שואל את עצמו:
"האם הכתובת 100.200.5.8 שייכת לרשת הזו?"

התשובה היא, כפי שלמדנו, כן. הרי מזהה הרשת הינו 100.200, והכתובת 100.200.5.8 אכן תואמת מזהה זה.
אי לכך, החבילה מתאימה לחוק המצוין בשורה 3, ולכן הנתב **יעביר (forward)** את החבילה אל הממשק[47] 2.2.2.2, כלומר אל R3. בתורו, R3 יעביר את החבילה הלאה, עד שזו תגיע אל הישות בעלת הכתובת 100.200.5.8.

נראה דוגמה נוספת. כעת הגיעה אל הנתב חבילה עם כתובת היעד 192.168.6.6. הנתב יבצע את הפעולות הבאות: בתור התחלה, הוא ינסה לבדוק האם הכתובת מתאימה לרשומת הניתוב האחרונה שיש לו, כלומר לרשומה מספר שלוש. לשם כך הוא ישאל: "האם הכתובת 192.168.5.8 שייכת לרשת הזו?"

---

[47] המילה "ממשק" מתארת כרטיס רשת. לנתב יש מספר כרטיסי רשת, וכל אחד מהם מתואר באמצעות כתובת IP אחרת. בדוגמה זו, המשמעות של "ממשק 2.2.2.2", היא כרטיס הרשת בעל הכתובת 2.2.2.2, כלומר הכרטיס המחבר את הנתב R2 אל הנתב R3.

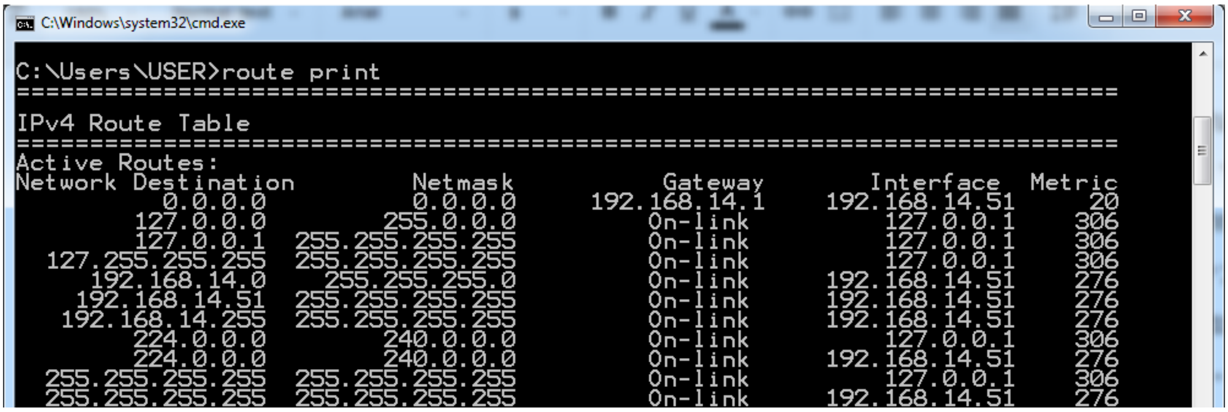
התשובה היא לא, זאת מכיוון שמזהה הרשת הינו 100.200, והכתובת 192.168.5.8 אינה עונה על מזהה זה.

כעת, הנתב ימשיך לרשומה הבאה, רשומה מספר שתיים, אשר מתארת את הרשת 192.168.0.0 עם המסכה 255.255.0.0. הנתב שוב ישאל: "האם הכתובת 192.168.5.8 שייכת לרשת הזו?"
התשובה היא כן, שכן מזהה הרשת הינו 192.168, והכתובת 192.168.5.8 אכן עונה על מזהה זה.
אי לכך, החבילה מתאימה לחוק זה, והנתב יעביר את החבילה על הממשק 1.1.1.1, כלומר אל R1. בתורו, R1 יעביר את החבילה הלאה, עד שזו תגיע אל הישות בעלת הכתובת 192.168.5.8.

נראה דוגמה שלישית. כעת הגיעה אל הנתב חבילה עם כתובת היעד 5.5.5.5. הנתב ינסה לבדוק האם הכתובת מתאימה לרשומת הניתוב האחרונה שיש לו, כלומר לרשומה מספר שלוש, המתארת את הרשת 100.200.0.0/16. בשלב זה אנו כבר מבינים שהכתובת לא נמצאת ברשת המתוארת ברשימה זו, ולכן הנתב יעבור אל הרשומה הבאה, המתארת את הרשת 192.168.0.0/16. גם כאן, הכתובת 5.5.5.5 אינה חלק מהרשת, ולכן הנתב יעבור אל הרשומה האחרונה בטבלה.
כעת, הנתב שואל את עצמו: "האם הכתובת 5.5.5.5 היא חלק מהרשת 0.0.0.0/0?"
התשובה לשאלה זו היא – כן. למעשה, החוק שמתאר את הרשת 0.0.0.0 עם המסכה 0.0.0.0 הוא חוק גנרי, וכל כתובת IP תואמת אליו. היות שמסכת הרשת היא בגודל 0 ביטים, המשמעות היא שכל כתובת IP שהיא – תהיה חלק מן הרשת הזו. אי לכך, הנתב יעביר את החבילה אל הממשק 3.3.3.3 – כלומר אל R4, שבתורו ימשיך את הטיפול בחבילה.
מכאן אנו למדים למעשה, שכל חבילה אשר הנתב לא מעביר אל הנתבים R1 או R3, הוא צפוי להעביר אותה אל R4.

מהי טבלת הניתוב שלי?

גם למחשבים, בדומה לנתבים, יש טבלת ניתוב. על מנת לראות את טבלת הניתוב שלכם, היכנסו כעת ל-Command Line, והקישו את הפקודה **route print**:



```
C:\Windows\system32\cmd.exe

C:\Users\USER>route print
===========================================================================
IPv4 Route Table
===========================================================================
Active Routes:
Network Destination        Netmask          Gateway       Interface  Metric
          0.0.0.0          0.0.0.0     192.168.14.1    192.168.14.51     20
        127.0.0.0        255.0.0.0         On-link         127.0.0.1    306
        127.0.0.1  255.255.255.255         On-link         127.0.0.1    306
  127.255.255.255  255.255.255.255         On-link         127.0.0.1    306
     192.168.14.0    255.255.255.0         On-link     192.168.14.51    276
    192.168.14.51  255.255.255.255         On-link     192.168.14.51    276
   192.168.14.255  255.255.255.255         On-link     192.168.14.51    276
        224.0.0.0        240.0.0.0         On-link         127.0.0.1    306
        224.0.0.0        240.0.0.0         On-link     192.168.14.51    276
  255.255.255.255  255.255.255.255         On-link         127.0.0.1    306
  255.255.255.255  255.255.255.255         On-link     192.168.14.51    276
```

לעת עתה, נתעלם מהעמודות "Gateway" ו-"Metric", ונתעמק בעמודות "Network Destination", "Netmask" ו-"Interface".

ראוי לציין שניתן לראות שני ממשקים (Interfaces) למחשב זה:

- הממשק בעל הכתובת "192.168.14.51". זוכרים שמצאנו כתובת זו בתחילת הפרק? זהו כרטיס הרשת שלנו.
- הממשק בעל הכתובת "127.0.0.1". זוכרים שדיברנו על הכתובת הזו קודם תחת סעיף כתובות מיוחדות? חבילות שנשלחות לממשק זה לא באמת יגיעו לכרטיס הרשת, אלא "יישארו במחשב".

שימו לב לרשומה הראשונה בטבלה: אותה רשומה שעליה תתאמת כל כתובת IP, זו שמתארת את הרשת 0.0.0.0/0:

Network Destination	Netmask	Gateway	Interface	Metric
0.0.0.0	0.0.0.0	192.168.14.1	192.168.14.51	20

רשומה זו מתארת את ה-**Default Gateway** של המחשב – כלומר מי הנתב המשויך אל המחשב. כל חבילה שלא התאמתה על חוק ספציפי בטבלת הניתוב, תישלח אל ה-Default Gateway. מכאן שכל חבילה שתישלח אל הממשק "192.168.14.51", תישלח למעשה אל הנתב 192.168.14.1.

תרגיל 7.3 – ניתוב על פי טבלת ניתוב

על פי טבלת הניתוב שלעיל, ענו על השאלות הבאות:

1. לאן תישלח חבילה עם כתובת היעד "255.255.255.255"?
2. לאן תישלח חבילה עם כתובת היעד "192.168.14.51"?
3. לאן תישלח חבילה עם כתובת היעד "127.0.0.1"?
4. לאן תישלח חבילה עם כתובת היעד "127.5.5.6"?
5. לאן תישלח חבילה עם כתובת היעד "1.2.3.4"?

ICMP

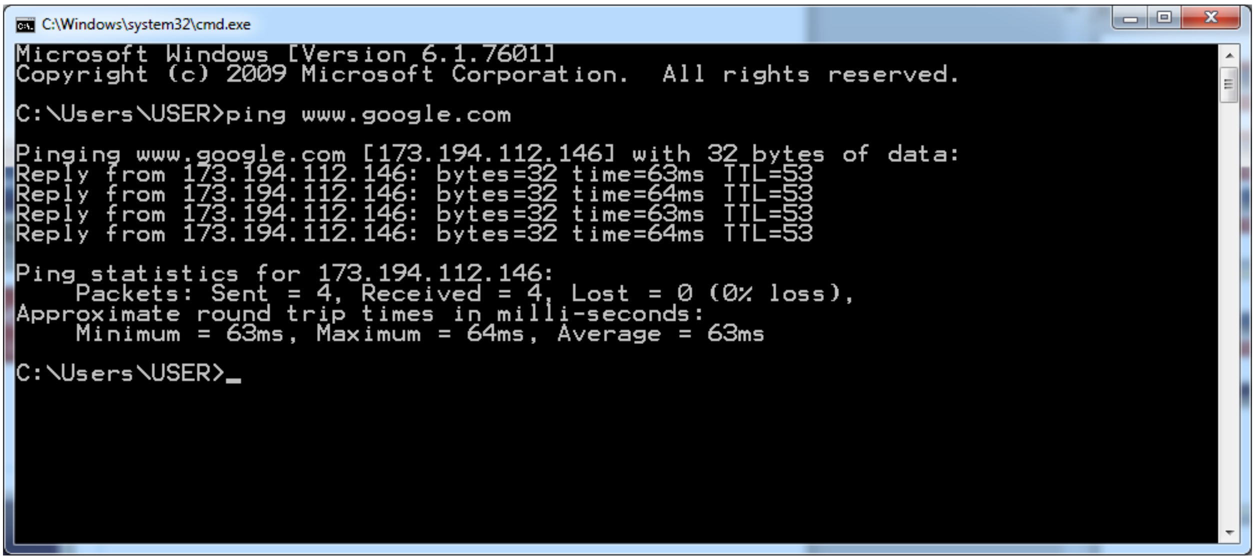
כעת נלמד על פרוטוקול נוסף – ICMP, ששמו המלא הוא: Internet Control Message Protocol. פרוטוקול זה נועד לעזור לטכנאים ולנו (אנשים המעוניינים להבין לעומק את דרך הפעולה של רשתות מחשבים) למצוא תקלות ברשת ולהבין את מצב הרשת.

הכלי המוכר ביותר המשתמש בפרוטוקול ICMP הוא הכלי **ping**, אשר פגשנו מספר פעמים לאורך הספר. כלי זה נועד להבין האם ישות מסוימת "למעלה" – כלומר דולקת, עובדת ומגיבה לשליחת הודעות אליה. לחלופין, הוא יכול לוודא שיש תקשורת מהמחשב שעליו אנו מריצים את הפקודה אל הרשת אליה הוא מחובר.

תרגיל 7.4 מודרך – בדיקת קישוריות ל-Google

נסו זאת בעצמכם – בצעו ping ל-"www.google.com", וודאו שאתם מקבלים תשובה. פעולה זו יכולה לעזור לנו לוודא האם המחשב שלנו מחובר לאינטרנט – מכיוון שלא סביר ש-Google "נפל", אנו יכולים לשלוח אליו ping ולקוות לתשובה. באם לא קיבלנו תשובה, כנראה שיש בעיה כלשהי אצלנו.

המסך שלכם אמור להיראות פחות או יותר כך:



כפי שניתן לראות, **ping** מציג את כתובת ה-IP של "www.google.com", שולח כברירת מחדל ארבע הודעות ומספר לנו כמה מהן הגיעו ליעדן ונענו, ובאיזו מהירות.

איך Ping עובד?

מה, לא הסנפתם כשהרצתם קודם לכן את **ping**? ובכן, אתם אכן עדיין חדשים בעולם הרשתות.

פתחו עתה את Wireshark, הסניפו והריצו שוב את פקודת ה-**ping** שהרצתם קודם. אל תשכחו להשתמש ב-filter כדי לסנן את הפקטות הלא רלוונטיות:

Filter: icmp   Expression... Clear Apply Save

No.	Time	Source	Destination	Protocol	Length	Info
19	4.90664200	192.168.14.51	173.194.113.148	ICMP	74	Echo (ping) request id=0x0001, seq=131/33536, ttl=128
20	4.98088400	173.194.113.148	192.168.14.51	ICMP	74	Echo (ping) reply id=0x0001, seq=131/33536, ttl=51
21	5.90732300	192.168.14.51	173.194.113.148	ICMP	74	Echo (ping) request id=0x0001, seq=132/33792, ttl=128
22	5.98162000	173.194.113.148	192.168.14.51	ICMP	74	Echo (ping) reply id=0x0001, seq=132/33792, ttl=51
24	6.90834300	192.168.14.51	173.194.113.148	ICMP	74	Echo (ping) request id=0x0001, seq=133/34048, ttl=128
25	6.98264600	173.194.113.148	192.168.14.51	ICMP	74	Echo (ping) reply id=0x0001, seq=133/34048, ttl=51
29	7.91034000	192.168.14.51	173.194.113.148	ICMP	74	Echo (ping) request id=0x0001, seq=134/34304, ttl=128
30	7.98515900	173.194.113.148	192.168.14.51	ICMP	74	Echo (ping) reply id=0x0001, seq=134/34304, ttl=51

כעת, נסתכל באחת החבילות שנשלחו. שימו לב לבחור בחבילת בקשה (request):

```
⊞ Frame 18: 74 bytes on wire (592 bits), 74 bytes captured (592 bits) on interface 0
⊞ Ethernet II, Src: Dell_d6:0c:2a (d4:be:d9:d6:0c:2a), Dst: Bewan_a5:16:63 (00:0c:c3:a5:16:63)
⊞ Internet Protocol Version 4, Src: 192.168.14.51 (192.168.14.51), Dst: 173.194.113.146 (173.194.113.146)
⊟ Internet Control Message Protocol
    Type: 8 (Echo (ping) request)
    Code: 0
    Checksum: 0x4cd1 [correct]
    Identifier (BE): 1 (0x0001)
    Identifier (LE): 256 (0x0100)
    Sequence number (BE): 138 (0x008a)
    Sequence number (LE): 35328 (0x8a00)
    [Response In: 19]
  ⊟ Data (32 bytes)
      Data: 6162636465666768696a6b6c6d6e6f7071727374757677616...
      [Length: 32]

0000  00 0c c3 a5 16 63 d4 be  d9 d6 0c 2a 08 00 45 00   .....c.. ...*..E.
0010  00 3c 01 b4 00 00 80 01  4a dd c0 a8 0e 33 ad c2   .<...... J....3..
0020  71 92 08 00 4c d1 00 01  00 8a 61 62 63 64 65 66   q...L... ..abcdef
0030  67 68 69 6a 6b 6c 6d 6e  6f 70 71 72 73 74 75 76   ghijklmn opqrstuv
0040  77 61 62 63 64 65 66 67  68 69                     wabcdefg hi
```

נשים לב למודל השכבות:

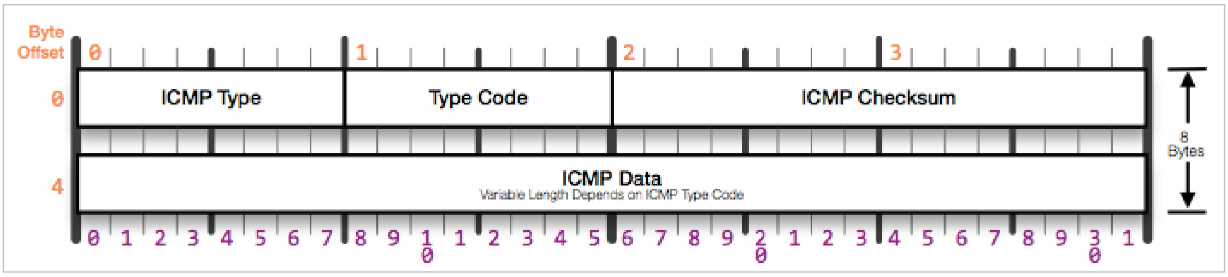
- בשכבה השנייה, ניתן לראות את השימוש בפרוטוקול Ethernet[48]. דבר זה מלמד כי כרטיס הרשת ממנו נשלחה החבילה הוא כרטיס מסוג Ethernet[49].
- בשכבה השלישית, ישנו שימוש בפרוטוקול IP. כתובת המקור היא הכתובת של המחשב ששלח את ה-ping, וכתובת היעד היא הכתובת של Google.
- לאחר מכן, בתוך המידע של השכבה השלישית, ישנו פרוטוקול ICMP.

כעת נבחן את מבנה ה-Header של חבילת ICMP:

---

48 על השכבה השנייה בכלל, ופרוטוקול Ethernet בפרט, נרחיב בפרק הבא.

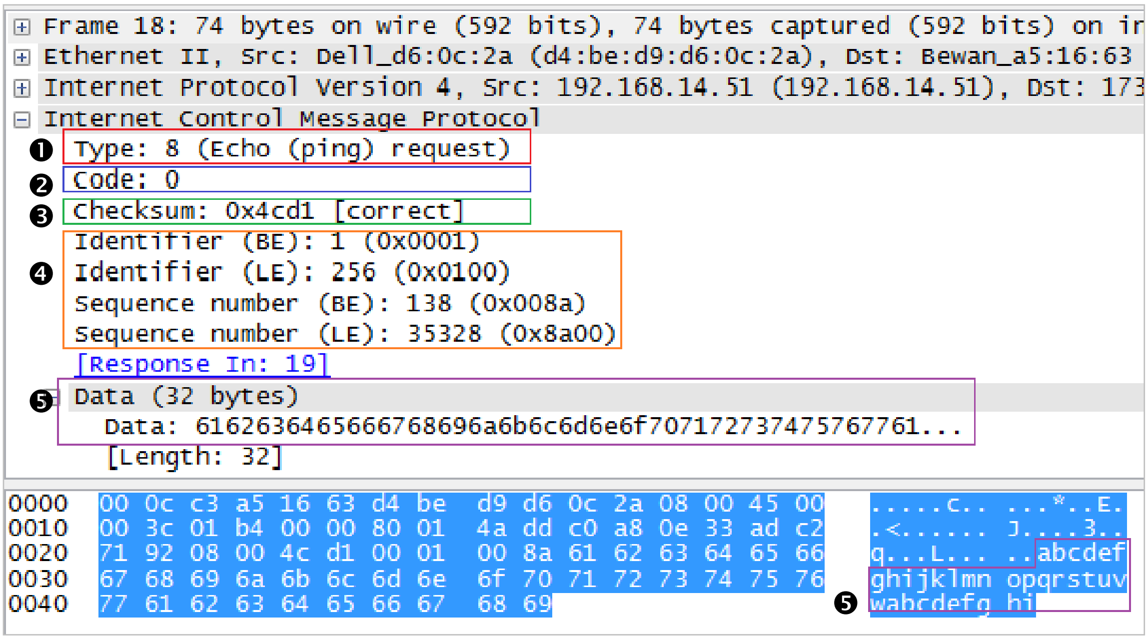
49 ייתכן שבמחשב שלכם תראו שכבה שנייה אחרת, בהתאם לכרטיס הרשת שלכם. למשל, ייתכן שתראו כי מדובר בשכבה שנייה של WiFi, באם אתם מחוברים באמצעות כרטיס רשת WiFi.



ה-Header של חבילת ICMP הוא לעולם בגודל קבוע של 8 בתים:

- בית 0[50] – Type: כולל את סוג החבילה. ישנם סוגים שונים של חבילות ICMP. בקרוב נראה דוגמה.
- בית 1 – Code: זה למעשה תת-סוג של ה-Type שהוגדר בבית הקודם. שוב, בקרוב נראה דוגמה.
- בתים 2-3 – Checksum: ערך שמחושב על שדות ה-Header והמידע של ה-ICMP. נועד כדי לוודא שאין שגיאות בחבילה[51].
- בתים 4-7 – כל השאר: ערך הזה יהיה שונה בהתאם לסוג החבילה, שהוגדר בידי הבתים Type ו-Code.

נחזור לחבילה ששלחנו ל-Google ונביט במזהים:



- ב**אדום** (1) – אנו רואים את הבית הראשון, הלוא הוא ה-Type, סוג החבילה. מדובר בסוג 8, וכמו ש-Wireshark מועיל בטובו להגיד לנו – מדובר ב-Echo Request. זאת ועוד, Wireshark מגדיל

---

[50] שימו לב לכך שהבית הראשון הוא תמיד בית באינדקס 0. הבית השני הוא בית באינדקס 1, וכך הלאה.

[51] להזכירכם, למדנו על Checksum בפרק שכבת התעבורה/ מה זה Checksum?.

לעשות ואף מתאר בסוגריים כי זו חבילה שקשורה ל-ping. השימוש ב-ping הוא כה נפוץ, עד שחבילות מסוג 8 (שהוגדרו במקור כ-"Echo Request") נקראות לעיתים "Ping Request".

- **בכחול** (2) – אנו רואים את הבית השני, ה-Code. עבור Type מסוג 8 (חבילת Echo Request), שדה ה-Code תמיד מכיל את הערך 0.
- **בירוק** (3) – הבתים השלישי והרביעי, המציינים את ה-Checksum. זהו אותו חישוב שמתבצע במחשב ששולח את ההודעה, כמו גם במחשב שמקבל את ההודעה. אם התוצאה היא אותה התוצאה – אין שגיאה בחבילה.
- **בכתום** (4) – אלו מזהים יחודיים להודעת Echo Request. הלקוח, אשר שולח את בקשת ה-Ping, יכול להשתמש במזהים אלו כדי לזהות את חבילת הבקשה שלו כשהוא שולח מספר חבילות. כלומר, הוא יכול לציין כאן את הערך "1", ובחבילת הבקשה הבאה את הערך "2". כשהשרת יענה, הוא יגיב לכל החבילה עם המספר שלה. מבולבלים? אל דאגה, המשיכו לעקוב אחר הדוגמה.
- **בסגול** (5) – זהו ה"מידע" של חבילת ה-Echo Request. בשימוש ב-Windows, כפי שאתם יכולים לראות, נשלח פשוט כל ה-ABC האנגלי עד האות 'w', ואז שוב מהאות 'a' ועד לאות 'i'.

שאלת מחשבה: אילו מהערכים זהים בין החבילה שבדוגמה לחבילה ששלחתם במחשב שלכם? מדוע דווקא הערכים האלו נותרו קבועים, בעוד אחרים השתנו?

מחשב היעד (במקרה הזה, השרת של Google) קיבל את החבילה ששלחנו. כעת, הוא מסתכל עליה, ומבין שמדובר בחבילת ICMP. לאחר מכן הוא מסתכל בשדה ה-Type, ומבין שמדובר בבקשת Echo Request (או Ping). אי לכך, הוא מחליט שעליו לענות. כעת נבחן את החבילה הבאה, היא חבילת התשובה:

```
⊞ Frame 19: 74 bytes on wire (592 bits), 74 bytes captured (592 bits) on in
⊞ Ethernet II, Src: Bewan_a5:16:63 (00:0c:c3:a5:16:63), Dst: Dell_d6:0c:2a
⊞ Internet Protocol Version 4, Src: 173.194.113.146 (173.194.113.146), Dst:
⊟ Internet Control Message Protocol
 ❶   Type: 0 (Echo (ping) reply)
 ❷   Code: 0
 ❸   Checksum: 0x54d1 [correct]
     Identifier (BE): 1 (0x0001)
     Identifier (LE): 256 (0x0100)
 ❹   Sequence number (BE): 138 (0x008a)
     Sequence number (LE): 35328 (0x8a00)
     [Response To: 18]
     [Response Time: 74.398 ms]
   ⊟ Data (32 bytes)
 ❺     Data: 6162636465666768696a6b6c6d6e6f7071727374757677761...

0000  d4 be d9 d6 0c 2a 00 0c  c3 a5 16 63 08 00 45 b8   .....*.. ...c..E.
0010  00 3c 5c d3 00 00 33 01  3c 06 ad c2 71 92 c0 a8   .<\...3. <...q...
0020  0e 33 00 00 54 d1 00 01  00 8a 61 62 63 64 65 66   .3..T... ..abcdef
0030  67 68 69 6a 6b 6c 6d 6e  6f 70 71 72 73 74 75 76   ghijklmn opqrstuv
0040  77 61 62 63 64 65 66 67  68 69                  ❺  wabcdefg hi
```

מבנה השכבות נותר, מן הסתם, זהה ולכן לא נתעכב עליו. נעבור על השדות הרלוונטים ל-ICMP:

- **באדום** (1) – אנו רואים את הבית הראשון, הלוא הוא ה-Type, סוג החבילה. מדובר בסוג 0, כאשר Wireshark מציין בפנינו שמדובר ב- Echo Reply. גם הפעם, Wireshark לא עוצר כאן ומוסיף בסוגריים כי זו חבילה שקשורה ל-ping.
- **בכחול** (2) – אנו רואים את הבית השני, ה-Code. גם עבור חבילות מסוג Echo Reply, שדה ה-Code תמיד מכיל את הערך 0.
- **בירוק** (3) – הבתים השלישי והרביעי, המציינים את ה-Checksum.
- **בכתום** (4) – כאן Google העתיק את המזהים שנשלחו בחבילת הבקשה. הסתכלו בחבילה הקודמת, ושימו לב כי אכן מדובר באותם ערכים בדיוק! לאחר מכן, הסתכלו בבקשת ה-Ping הבאה, כלומר בחבילת השאלה הבאה. תראו שהמזהים האלו שונים. מצאו את חבילת התשובה שלה, כלומר חבילת תשובה שבה מזהים אלו זהים למזהים שבבקשה.
- **בסגול** (5) – זהו ה"מידע" של חבילת ה-Echo Reply. השרת (במקרה שלנו, Google) חייב להעתיק את אותו המידע שניתן בחבילת הבקשה. כפי שניתן לראות, זהו אכן אותו המידע.

שימו לב: לקחנו פקודה מוכרת (**ping**), והשתמשנו ב-Wireshark על מנת להבין איך היא באמת עובדת. ניתן לראות כאן שימוש נוסף בכלי Wireshark, שמאפשר לנו להבין איך הדברים עובדים מאחורי הקלעים!

תרגיל 7.5 מודרך – Ping – עשה זאת בעצמך

כתבו, באמצעות Scapy, סקריפט ששולח חבילת Echo Request אל "www.google.com", ומקבל את התשובה. הסניפו את התעבורה תוך כדי.[52]

לאחר שניסיתם לבצע את התרגיל לבדכם, נעשה זאת יחדיו.

ראשית, טענו את Scapy. כעת, נתחיל מלבנות את פקטת השאלה, שלב אחר שלב.

בתור התחלה, ניצור פקטה עם שכבת IP, כשמעליה יש ICMP. נעשה זאת כך:

```
>>> request_packet = IP()/ICMP()
```

[52] כפי ששמתם לב לאורך הספר, **הסנפה** היא פעולה אשר אנו מבצעים באופן שוטף ועוזרת לנו במספר תחומים – תוך כדי כתיבת קוד, במהלך הרצת כלי קיים (כגון Ping), ובמקרים נוספים. זכרו להשתמש בכלי עוצמתי זה!

כעת, נביט בחבילה שנוצרה:

```
C:\Windows\system32\cmd.exe - scapy
>>>
>>> request_packet = IP()/ICMP()
>>> request_packet.show()
###[ IP ]###
  version= 4
  ihl= None
  tos= 0x0
  len= None
  id= 1
  flags=
  frag= 0
  ttl= 64
  proto= icmp
  chksum= None
  src= 192.168.14.51
  dst= 127.0.0.1
  \options\
###[ ICMP ]###
     type= echo-request
     code= 0
     chksum= None
     id= 0x0
     seq= 0x0
>>> _
```

ניתן לראות ש-Scapy יצר עבורנו חבילה עם ערכים שהוא מנחש שיסייעו לנו. כך למשל, כתובת המקור בחבילת ה-IP מכילה את כתובת ה-IP שלנו, במקרה הזה – "192.168.14.51". עם זאת, כתובת היעד אינה הכתובת של "www.google.com". נסו לשנות זאת, ובדקו שהחבילה אכן השתנתה. ניתן לעשות זאת כך:

```
C:\Windows\system32\cmd.exe - scapy
     id= 0x0
     seq= 0x0
>>> request_packet[IP].dst="www.google.com"
>>> request_packet.show()
###[ IP ]###
  version= 4
  ihl= None
  tos= 0x0
  len= None
  id= 1
  flags=
  frag= 0
  ttl= 64
  proto= icmp
  chksum= None
  src= 192.168.14.51
  dst= Net('www.google.com')
  \options\
###[ ICMP ]###
     type= echo-request
     code= 0
     chksum= None
     id= 0x0
     seq= 0x0
>>>
```

שימו לב – לא נתנו ל-Scapy כתובת IP אמיתית, אלא את שם ה-DNS של "www.google.com". מעבר לכך, Scapy גם שמר אצלו את הכתובת בתור Net('www.google.com'), ולא בתור כתובת IP! בדרך זו, Scapy מקל עלינו ומבצע את תרגום כתובת ה-DNS לכתובת IP עבורנו.

לחלופין, יכולנו כמובן להשתמש בכתובת IP, ולכתוב לדוגמה:

>>> request_packet[IP].dst="173.194.113.178"

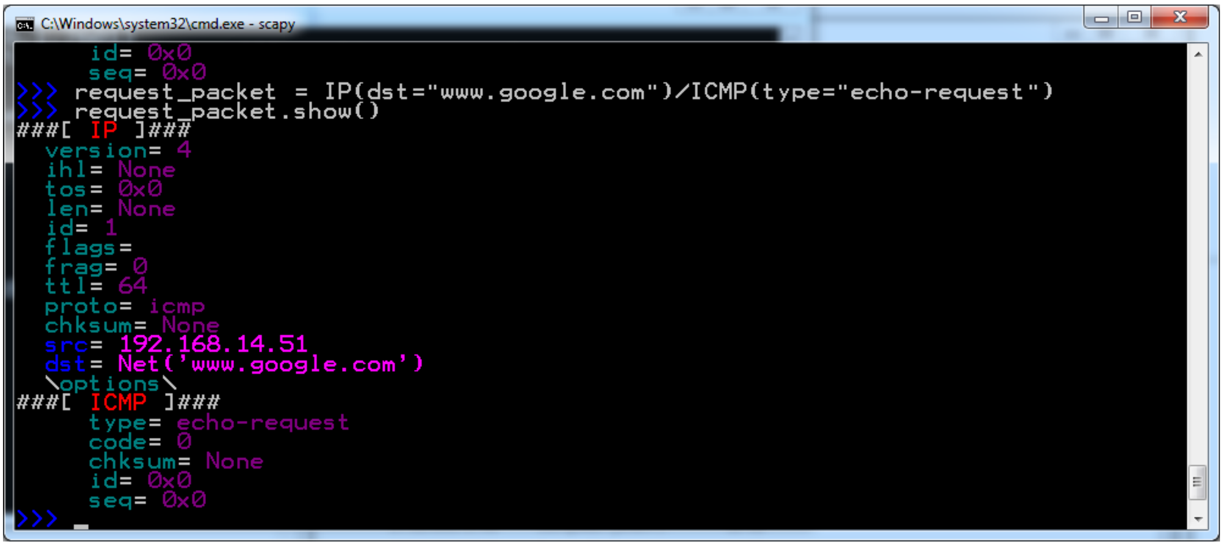
כעת יש לנו חבילת IP, שכתובת המקור שלה היא המחשב שלנו, וכתובת היעד שלה היא "www.google.com". החבילה כוללת גם Header של ICMP. אם נבחן את השדות, נראה כי ה-Type מכיל "echo-request". מכאן ש-Scapy מנחש שאם אנו מבקשים ליצור חבילת ICMP, אנו ככל הנראה רוצים חבילה מסוג Echo Request, שהיא כפי שלמדנו שאלת Ping.

שדה ה-Code מכיל את הערך 0, כפי שחבילת Echo Request צריכה להיראות. לאחר מכן, שדה ה-Checksum (שנקרא בפי Scapy בשם "chksum") הינו ריק (מטיפוס None). דבר זה קורה מכיוון שה-Checksum מחושב בזמן שליחת הפקטה, ולא בזמן היצירה שלה. שאר השדות מכילים את הערך 0x0, אך לא באמת מעניינים אותנו כרגע.

על מנת ליצור את החבילה הזו בשורה אחת ולוודא שאכן נוצרת חבילה מסוג Echo Request, יכולנו להשתמש בשורה הבאה:

>>> request_packet = IP(dst="www.google.com")/ICMP(type="echo-request")

נוודא זאת:

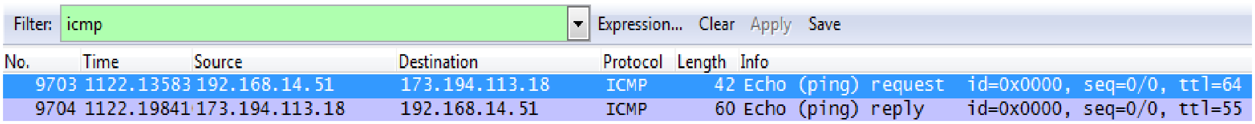


כמה נוח ויפה להשתמש ב-Scapy! שימו לב שהשורה הזו מסתמכת על כך ש-Scapy משתמש בכתובת ה-IP שלנו בתור כתובת מקור, וב-0 בתור שדה ה-Code של ICMP.

היות שאין צורך לשנות דברים נוספים בחבילה, ניתן לשלוח אותה:

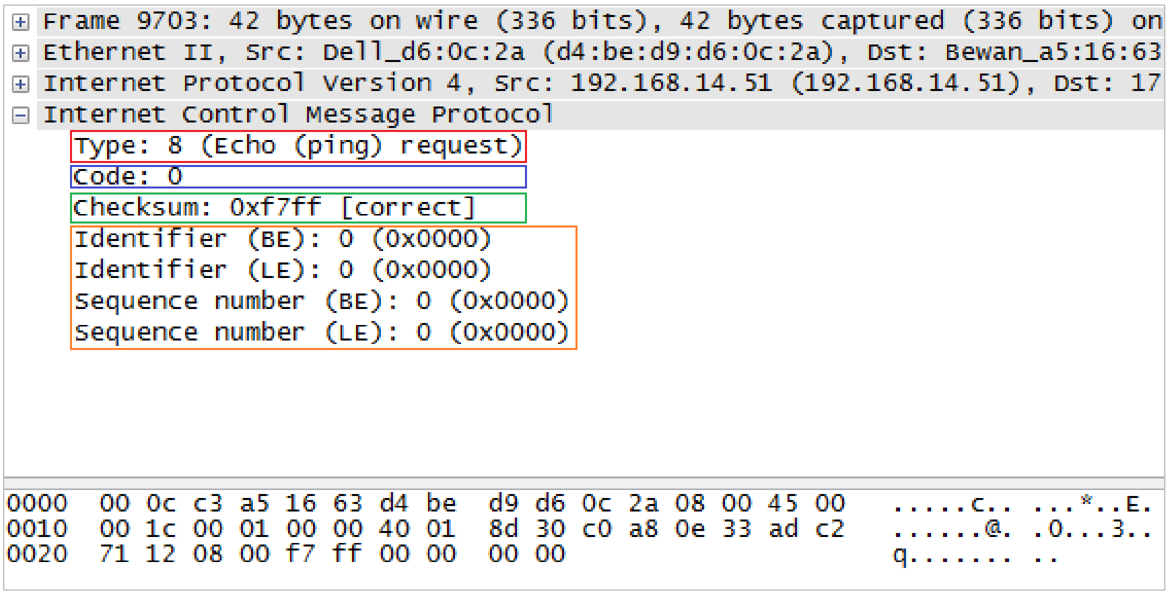
>>> send(request_packet)

הסתכלו ב-Wireshark. אתם צפויים לראות שתי חבילות:



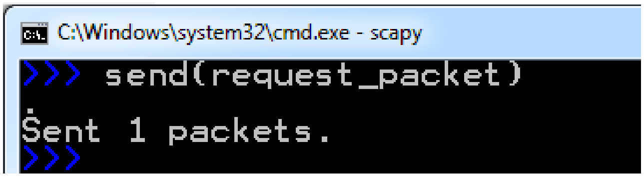
החבילה הראשונה היא חבילת השאלה שלנו. החבילה השנייה היא חבילת התשובה של Google, שאכן ענה לנו!

אם נבחן את שדות הבקשה שלנו, נראה שהם באמת השדות שיצרנו (מלבד ל-Checksum שחושב לאחר מכן):



כעת הסתכלו בחבילת התשובה. ציינו קודם לכן שהשרת (במקרה הזה "www.google.com") צריך להשתמש באותם מזהים שנשלחו בבקשה. במקרה שלנו, התשובה צריכה לכלול את הערך 0 בכל השדות: Identifier (BE), Identifier (LE), Sequence Number (BE), Sequence Number(LE).

הצלחנו לראות את בקשת התשובה! אך עשינו זאת רק ב-Wireshark, ו-Scapy נותר אדיש למדי נוכח התשובה של "www.google.com":

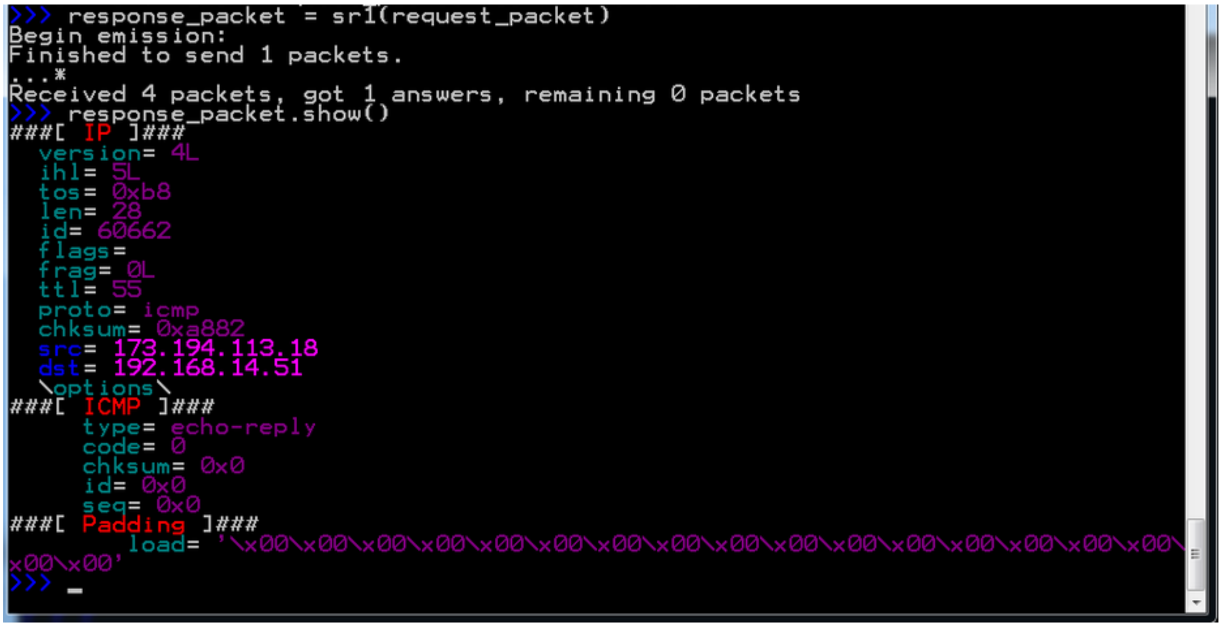


כעת, נשתמש בפונקציה שמאפשרת גם לקבל תשובה: **sr1**. את הפונקציה הזו הכרנו לראשונה ב[פרק שכבת התעבורה/ תרגיל 6.11 מודרך – קבלת תשובה לשאילתת DNS באמצעותScapy]().

נשתמש בפונקציה זו בצורה הבאה:

```
>>> response_packet = sr1(request_packet)
```

כעת Scapy ישלח את החבילה ויחכה לתשובה. כאשר התשובה תחזור, היא תישמר במשתנה response_packet. בדקו זאת על ידי הסתכלות בתשובה:

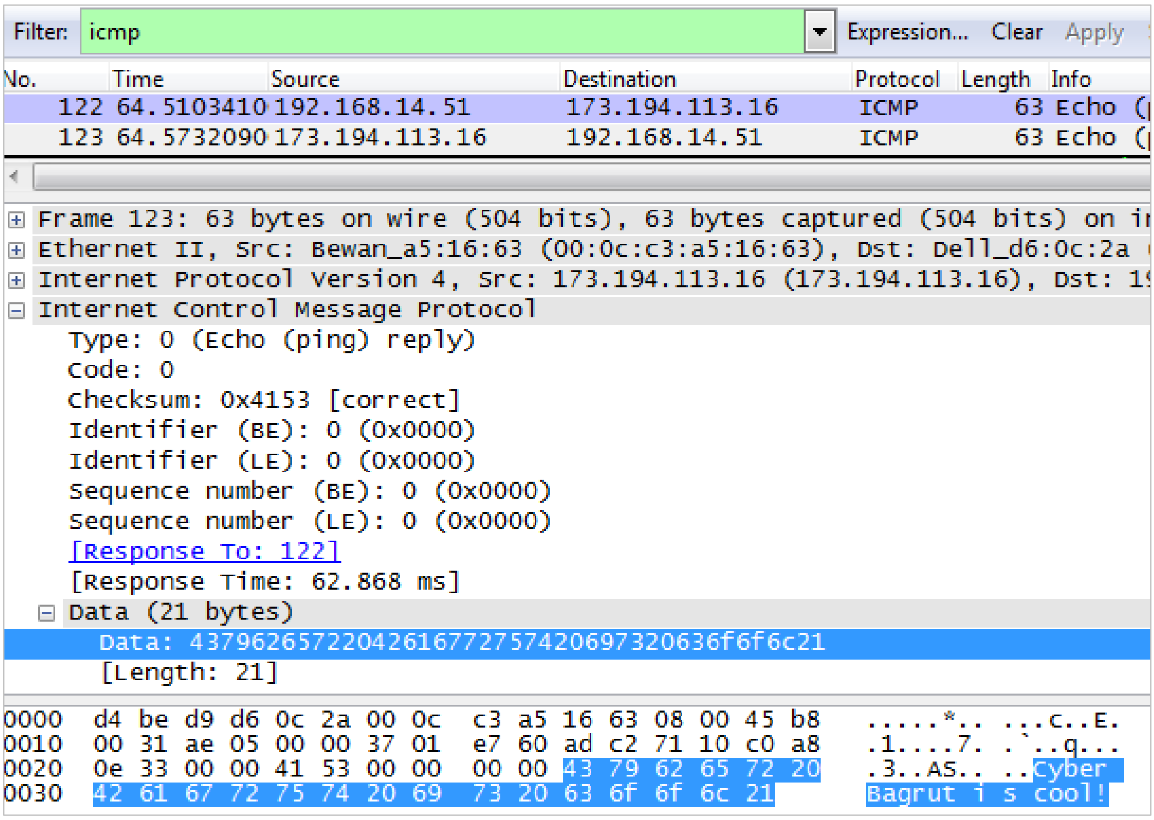


עתה הגיעה העת לבחון הנחות שהיו לנו קודם לכן. טענו שהשרת ישכפל כל מידע שנשלח לו בחבילת התשובה. נסו לשלוח אל השרת חבילת Echo Reuqest, עם המידע "Cyber Bagrut is cool!". אם תצליחו, אתם צפויים לקבל את המידע הזה מהשרת. נוכל לעשות זאת בצורה הבאה:

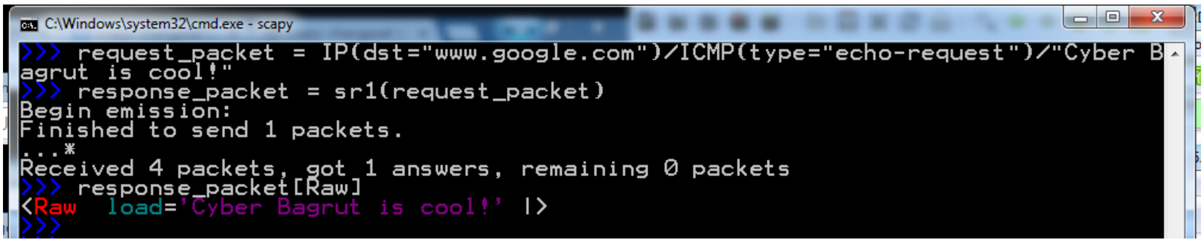
>>> request_packet = IP(dst="www.google.com")/ICMP(type="echo-request")/"Cyber Bagrut is cool!”

>>> response_packet = sr1(request_packet)

נסתכל ב-Wireshark ונראה ש-"www.google.com" אכן הגיב לנו וחזר על המידע שנתנו לו:



נוכל לוודא זאת גם באמצעות Scapy:



תרגיל 7.6 – תרגילי Ping

השתמשו ב-Scapy ובידע שלכם בכדי לממש סקריפטים אשר יבצעו משימות שונות:

1. שלחו הודעת Ping ל-"www.facebook.com". השתמשו ב-Identifier מסוים, וודאו שהשרת החזיר לכם תשובה בעלת אותו ה-Identifier.
2. שלחו שתי הודעות Ping ל-"www.facebook.com", ולכל אחת תנו Identifier אחר. שימו לב שאתם מקבלים את התשובות ומבינים איזו תשובה שייכת לאיזו שאלה.

3. כתבו סקריפט אשר ניתן להריץ משורת הפקודה. על הסקריפט לקבל כפרמטר כתובת של שרת, לשלוח אליו 4 חבילות Echo Request, ולכתוב למסך כמה תשובות הגיעו בהצלחה.

לדוגמה, שימוש בסקריפט ייראה כך:

my_ping.py 2.3.4.5

תשובה לדוגמה תהיה:

Sending 4 packets to 2.3.4.5.

Received 3 reply packets.

רמז: קראו את התיעוד של הפונקציה sr1, והבינו אילו פרמטרים נוספים היא יכולה לקבל.

4. שפרו את הסקריפט הקודם, כך שיקבל מספר שונה של פקטות לשלוח (לאו דווקא 4). כמו כן, וודאו שהסקריפט שולח את כל הפקטות, ורק אז מחכה לתשובה.

לדוגמה, שימוש בסקריפט ייראה כך:

my_ping 2.3.4.5 3

תשובה לדוגמה תהיה:

Sending 3 packets to 2.3.4.5

Received 2 reply packets.

רמז: קראו על הפונקציה sr.

איך Traceroute עובד?

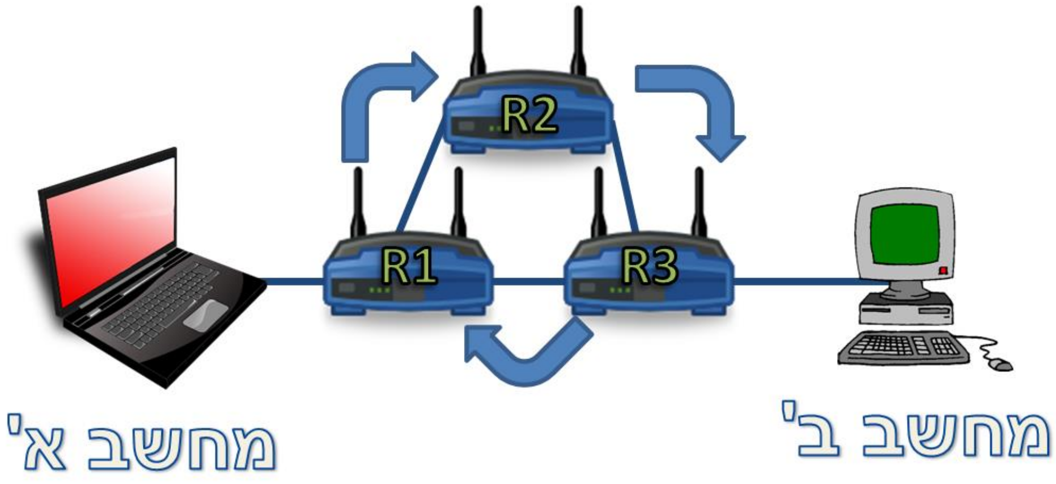
ראינו בתחילת הפרק את הכלי Traceroute אשר מאפשר לנו להבין את הדרך שחבילה עוברת בין שתי נקודות קצה. כעת, נלמד להבין איך Traceroute עובד ונממש את הפונקציונליות של הכלי – בעצמנו.

ראשית, נלמד על אחד השדות ב-Header של IP, שנקרא **Time To Live**, או בקיצור – **TTL**. השדה באורך בית אחד, כלומר שהערך שניתן בו יכול להיות בין 0 ל-255. על מנת להבין את השדה, נתחיל קודם להסביר את הצורך בו.

הסתכלו בתמונת הרשת הבאה:

נאמר שמחשב א' שולח חבילה למחשב ב'. החבילה תגיע ראשית אל R1. כעת, הנתב R1 יסתכל בטבלת הניתוב שלו, ויחליט שהדרך הטובה ביותר להעביר את החבילה למחשב ב' היא דרך הנתב R2. לכן החבילה תעבור אל R2. בתורו, הנתב R2 יבחן את החבילה, ויגיע למסקנה שהדרך הטובה ביותר להעביר אותה למחשב ב' היא דרך הנתב R3, ולכן יעביר אותה אליו. כעת, החבילה תגיע אל R3, שיבחן אותה ויגיע למסקנה שהדרך הטובה ביותר להעביר את החבילה למחשב ב', היא דרך R1. הנתב R3, יעביר את החבילה לנתב R1, שכעת יגיע למסקנה שהדרך הטובה ביותר להעביר את החבילה למחשב ב' היא דרך הנתב R2…

כפי שבוודאי הבחנתם, החבילה "לכודה" כרגע בין הנתבים R1, R2 ו-R3, ותמשיך להיות מועברת בתוך ה"לולאה" שנוצרה:



כעת חשבו – איזה נזק נגרם כתוצאה מכך שהחבילה הזו נשארת "תקועה" ולא תגיע אל יעדה?

כמובן, מחשב ב' לא יקבל את החבילה שיועדה לו. אך מעבר לכך, החבילה הזו יוצרת עומס על הרשת. שלושת הנתבים – R1, R2 ו-R3, ממשיכים לעבד אותה בכל פעם מחדש. הם מקדישים לה זמן ומשאבים שהיו יכולים להיות מוקדשים לחבילות אחרות. כך גם הקישורים בהם החבילה מועברת (הקישור בין R1 ל-R2, הקישור בין R2 ל-R3 והקישור בין R3 ל-R1) עמוסים יותר, כיוון שהחבילה הזו ממשיכה לעבור בהם. קיום של חבילות "לכודות" כאלו ברשת משפיע על הביצועים של הרשת ופוגע בהם!

אי לכך, עלינו למנוע מקרים כאלה. שיטה אחת לעשות זאת היא להשתמש במנגנון ה-TTL, שקובע למעשה כמה פעמים החבילה יכולה להיות מועברת הלאה. נשתמש בדוגמה. נאמר שערך ה-TTL הראשוני של החבילה אותה שלח מחשב א', הוא 4. את ערך TTL זה קבע מחשב א'. נבחן את המסלול של החבילה, וכן את ערך השדה TTL:

- החבילה נשלחה ממחשב א' ומגיעה אל הנתב R1. הנתב בוחן את ערך ה-TTL, ורואה שהוא 4. אי לכך, הוא מוריד ממנו 1, ומעביר אותו הלאה, אל נתב R2. כלומר, הוא משנה את השדה ל-TTL=3, שכן הוא כבר טיפל בחבילה.
- החבילה מגיעה אל הנתב R2. הוא בוחן את ערך ה-TTL, ורואה שהוא 3. הוא מוריד ממנו 1, ומעביר אותו אל הנתב R3. כלומר, הוא משנה את השדה ל-TTL=2, ומעביר אותו אל הנתב R3.
- כעת הנתב R3 קיבל את החבילה. הוא בוחן את ערך ה-TTL, ורואה שהוא 2. במידה שהוא יעביר אותו אל מחשב ב' – תהליך השליחה הסתיים, החבילה הגיעה ליעדה והכל בסדר. אך מכיוון שלא כך המצב, הנתב מוריד את ערך ה-TTL, ומעביר את החבילה לנתב R1, כשערך ה-TTL=1.
- כעת הנתב R1 מסתכל על החבילה. הוא בוחן את ערך ה-TTL, ורואה שהוא שווה ל-1. הוא מחסיר 1 מערך זה, ומגיע למצב שבו TTL=0. אי לכך, הנתב מבין שאסור לו להעביר את החבילה הלאה – והוא משמיט את החבילה!

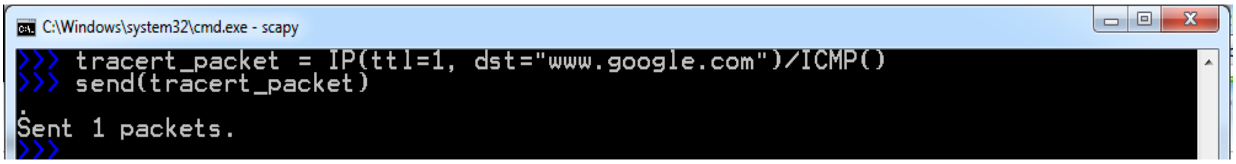
כך למעשה מנגנון ה-TTL מנע מהחבילה להישאר "תקועה" לנצח. עם זאת, במקרה כזה, מן הראוי להודיע למחשב א' שהחבילה שלו לא הגיעה ליעדה. לשם כך, נוצרה הודעת ICMP בשם Time-to-live exceeded. כאשר הנתב R1 מבין שעליו להשמיט את החבילה היות שערך ה-TTL שלה נמוך מדי, הוא ישלח אל מחשב א' הודעה מסוג Time-to-live exceeded, כדי לידע אותו על כך.

שימו לב לשימוש שנוכל לעשות בשדה זה: נניח כי אין בעיות ניתוב וחבילה יכולה להגיע ממחשב א' למחשב ב', כשהיא עוברת בדרך בנתב R1, לאחר מכן בנתב R2, משם היא מועברת לנתב R3 שלבסוף מועברת אל מחשב ב'. אם נשלח ממחשב א' למחשב ב' חבילת IP עם ערך TTL=1, הרי שהיא תגיע לנתב R1 שיגיד לנו שהוא לא יכול להעביר את החבילה. אם נשלח ממחשב א' למחשב ב' חבילת IP עם ערך TTL=2, היא צפויה להגיע לנתב R2 שיגיד לנו שהוא לא יכול להעביר את החבילה הלאה. כך נוכל לגלות את כל הרכיבים בדרך ממחשב א' למחשב ב'!

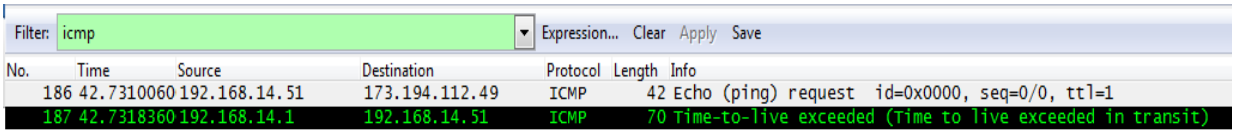
תרגיל 7.7 מודרך – Traceroute – עשה זאת בעצמך

בשלב זה נעשה שימוש בשדה TTL של חבילת IP, כדי לגלות את הרכיבים שנמצאים בין המחשב שלנו לבין "www.google.com". נסו לעשות זאת בעצמכם, ולמצוא את הרכיב הקרוב ביותר אליכם.

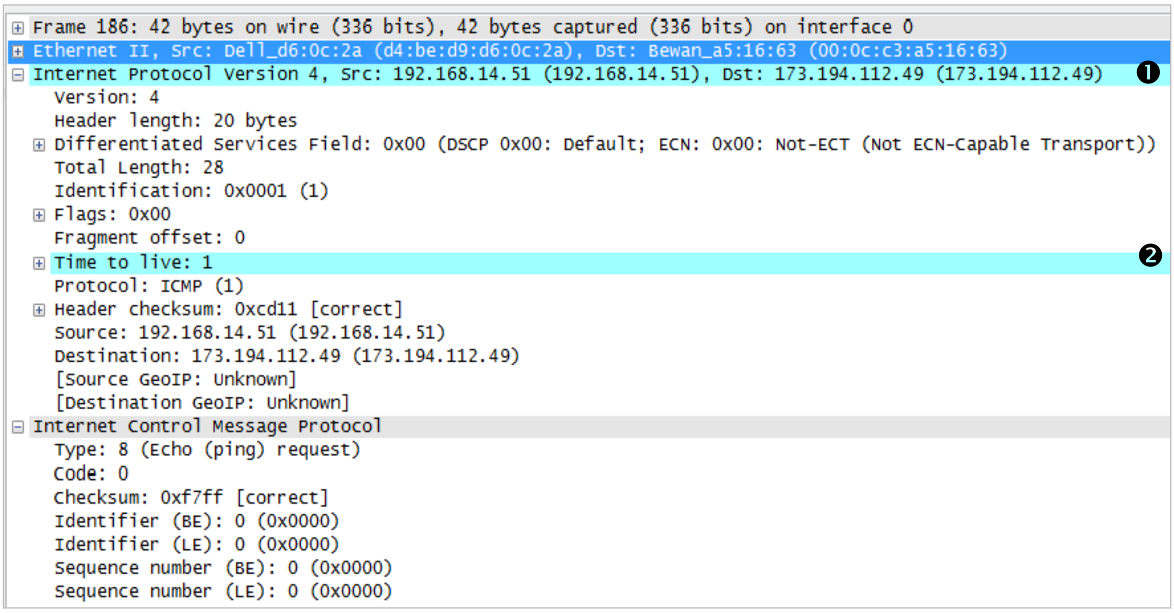
כעת נעשה זאת יחד. ראשית, פתחו את Wireshark והריצו הסנפה. השתמשו ב-Scapy, וצרו חבילת IP כאשר בשדה ה-TTL נמצא הערך 1. על מנת להידמות למימוש של הכלי **tracert**, החבילה תכיל הודעת ICMP Echo Request (כלומר הודעת Ping). עם זאת, אין בכך הכרח. בנו את החבילה ושלחו אותה אל "www.google.com":



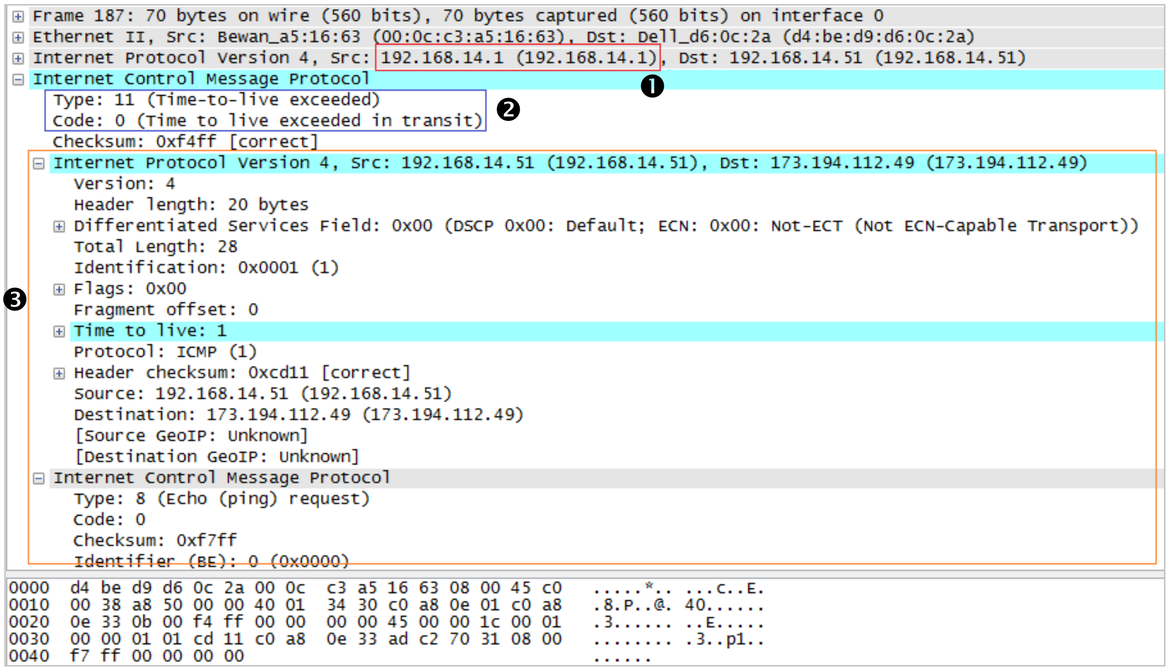
הסתכלו בהסנפה. אתם צפויים לראות שתי חבילות – הראשונה, החבילה ששלחתם. השנייה, החבילה שהתקבלה מהרכיב הראשון ביניכם לבין www.google.com:



נביט בחבילה שאנחנו שלחנו. שימו לב לשדה ה-Time to live המסומן ב**תכלת** (1,2):



כעת הסתכלו בחבילת התשובה:



נבחין בדגשים הבאים:

- ב**אדום** (1) – מסומנת כתובת השולח של החבילה. זהו למעשה הנתב שקיבל את חבילת ה- Echo Request שאנחנו שלחנו, הבין שהוא לא יכול להעביר אותה הלאה, ושלח את חבילת ה-Time-to--live exceeded. במילים אחרות, זוהי הכתובת של הנתב הראשון בינינו לבין "www.google.com"!
- ב**כחול** (2) – אלו השדות המאפיינים חבילה מסוג Time-to-live exceeded, כאשר שדה ה- Time to live קיבל את הערך 0. ניתן לראות שתחת שדה ה-Type של ICMP יש את הערך 11, ותחת השדה Code יש את הערך 0.

- **בכתום** (3) – זוהי למעשה העתקה של החבילה המקורית, כלומר החבילה שאנחנו שלחנו אל "www.google.com". הנתב בכתובת 192.168.14.1 העתיק אותה ושלח לנו אותה לאחר ה-Header של ICMP.

ובכן, הצלחנו למצוא את הכתובת של הנתב הראשון! עם זאת, לא עשינו זאת באופן תכנותי. נסו לשפר את הקוד שלכם כך שימצא את הכתובת של הנתב הראשון באופן תכנותי. ניתן לעשות זאת כך:

```
C:\Windows\system32\cmd.exe - scapy
>>> tracert_packet = IP(ttl=1, dst="www.google.com")/ICMP()
>>> tracert_response = sr1(tracert_packet)/ICMP()
Begin emission:
Finished to send 1 packets.
....*
Received 5 packets, got 1 answers, remaining 0 packets
>>> print 'The first router is: ' + tracert_response[IP].src
The first router is: 192.168.14.1
>>>
```

הכתובת שמצאנו היא כמובן של הנתב המחובר אל המחשב שלנו, והוא מהווה את ה-**Default Gateway** שלנו. בכדי לוודא זאת, נוכל להריץ את פקודת **ipconfig** אותה הכרנו קודם לכן:

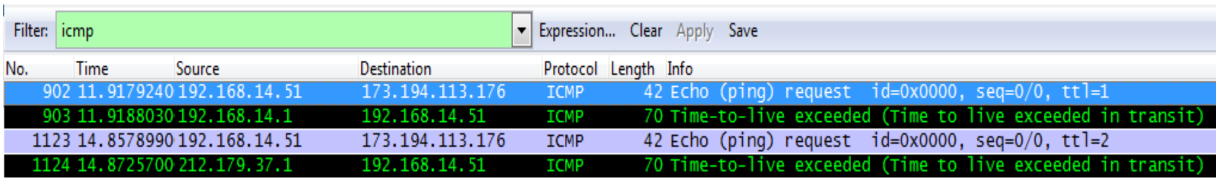
```
C:\Windows\system32\cmd.exe
C:\Users\USER>ipconfig

Windows IP Configuration


Ethernet adapter Local Area Connection:

   Connection-specific DNS Suffix  . : privatebox
   Link-local IPv6 Address . . . . . : fe80::419b:ac69:cfb6:705%11
   IPv4 Address. . . . . . . . . . . : 192.168.14.51
   Subnet Mask . . . . . . . . . . . : 255.255.255.0
   Default Gateway . . . . . . . . . : 192.168.14.1

Tunnel adapter Teredo Tunneling Pseudo-Interface:

   Connection-specific DNS Suffix  . :
   IPv6 Address. . . . . . . . . . . : 2001:0:9d38:90d7:1095:2a27:3f57:f1cc
   Link-local IPv6 Address . . . . . : fe80::1095:2a27:3f57:f1cc%12
   Default Gateway . . . . . . . . . : ::

Tunnel adapter isatap.privatebox:

   Media State . . . . . . . . . . . : Media disconnected
   Connection-specific DNS Suffix  . : privatebox
```

נסו בעצמכם למצוא את הכתובת של הנתב השני באותה השיטה.

נעשה זאת יחד:

```
C:\Windows\system32\cmd.exe - scapy
>>> tracert_packet = IP(ttl=2, dst="www.google.com")/ICMP()
>>> tracert_response = sr1(tracert_packet)/ICMP()
Begin emission:
Finished to send 1 packets.
.*
Received 2 packets, got 1 answers, remaining 0 packets
>>> print 'The second router is: ' + tracert_response[IP].src
The second router is: 212.179.37.1
>>>
```

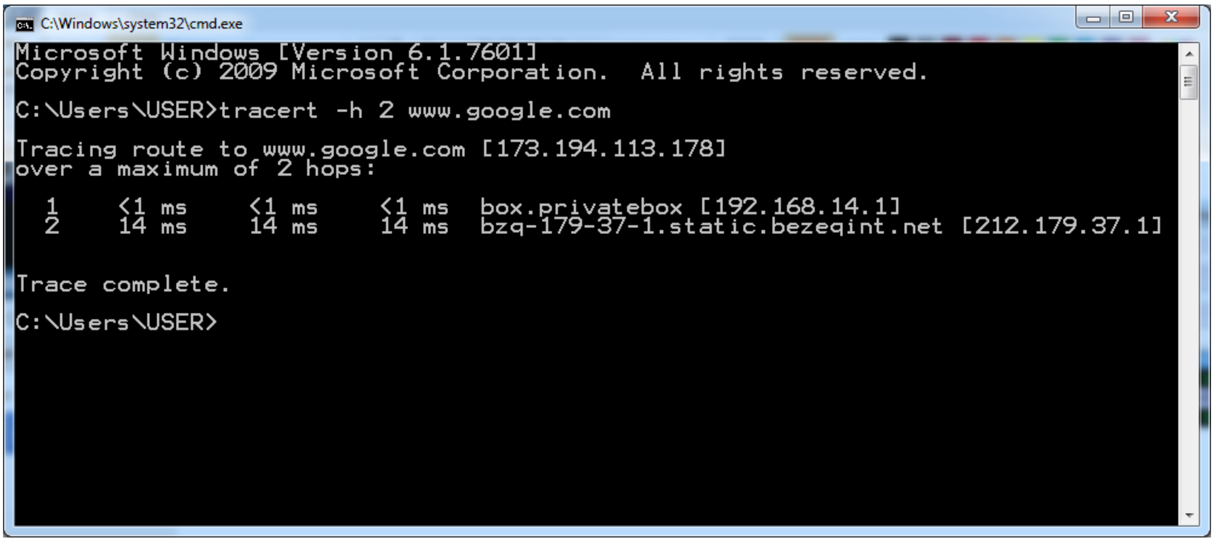
כעת נביט גם בהסנפה:



ההסנפה כוללת ארבע פקטות:

1. פקטת ה-Echo request ששלחנו אל "www.google.com", כאשר שדה ה-TTL מכיל את הערך 1.
2. תשובת Time-to-live exceeded מהנתב הראשון בדרך מהמחשב שלנו אל "www.google.com".
3. פקטת ה-Echo request ששלחנו אל "www.google.com", כאשר שדה ה-TTL מכיל את הערך 2.
4. תשובת Time-to-live exceeded מהנתב השני בדרך מהמחשב שלנו אל "www.google.com".

מכאן שרק הסתכלות בהסנפה יכולה להראות לנו את רשימת הנתבים בינינו לבין "www.google.com". כעת, נשתמש בכלי **tracert** בכדי למצוא את שני הנתבים הראשונים בדרך בינינו לבין "www.google.com", ונבדוק כי התשובה זהה[53]:



התשובה אכן כוללת את הנתבים שמצאנו בעצמנו. מעבר לכך, היא כוללת מידע נוסף אותו לא מדדנו. אם נביט בהסנפה, נראה שהחבילות ש-**tracert** שולח דומות מאוד לחבילות ששלחנו בעצמנו.

תרגיל 7.8 – עכשיו תורכם – תרגילי Traceroute

53 כפי שכבר למדנו, שכבת הרשת יכולה לבחור לשנות את הניתוב עבור כל חבילה וחבילה, ולכן ייתכן שהתשובה לא תהיה זהה. עם זאת, בדרך כלל הנתבים הראשונים בדרך כן יישארו זהים, מסיבות שלא נפרט אותן כרגע.

1. כתבו סקריפט אשר מדפיס את כל התחנות ביניכם לבין "www.google.com". על הסקריפט להדפיס את כתובת ה-IP של התחנה בלבד.
דגשים:
   - כאשר הגעתם לכתובת ה-IP של "www.google.com", הפסיקו את פעולת הסקריפט.
   - אל תעלו את ערך שדה ה-TTL באופן ידני כמו שעשינו עד כה. השתמשו בלולאה.
2. שפרו את הסקריפט שלכם. מדדו את הזמן שלוקח לכל נתב להגיב להודעה ששלחתם לו (כלומר הזמן שלקח מהרגע ששלחתם את חבילת השאלה, ועד אשר קיבלתם הודעה מהנתב), בדומה לכלי **tracert**. הדפיסו גם את מידע זה למסך.
3. שפרו את הסקריפט כך שהמשתמש יעביר בשורת הפקודה את הכתובת אליה הוא רוצה לבצע traceroute. שימוש לדוגמה בסקריפט ייראה כך:

my_traceroute.py www.google.com

הערה: על מנת לבצע traceroute באמצעות Scapy, כמו גם על מנת לבצע הרבה דברים אחרים, יש דרכים נוספות, אלגנטיות יותר מאלו שמוצגות בספר. עם זאת, מטרת הספר היא ללמד ולהבין את הדרך שבה הדברים עובדים, ולא ללמד שיטות 'מגניבות' להשתמש ב-Scapy. אתם מוזמנים להרחיב את הידע שלכם ב-Scapy על ידי קריאת ה-Tutorial שלו, שנמצא בכתובת:

http://www.secdev.org/projects/scapy/doc/usage.html

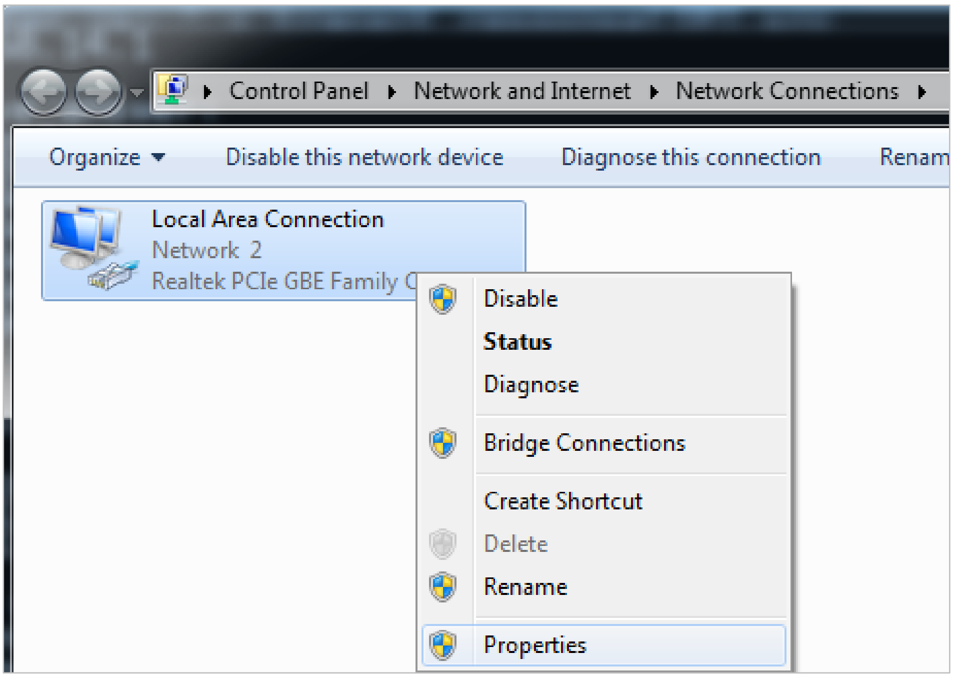
DHCP

למדנו בינתיים על שכבת הרשת בכלל, ועל פרוטוקולי IP ו-ICMP בפרט. מונח שחזר הרבה במהלך הפרק הוא "כתובת IP" – אותה כתובת לוגית בשכבת הרשת המשמשת כל ישות ברשת בכדי להזדהות. אך שאלה גדולה עדיין נותרה בלא תשובה:

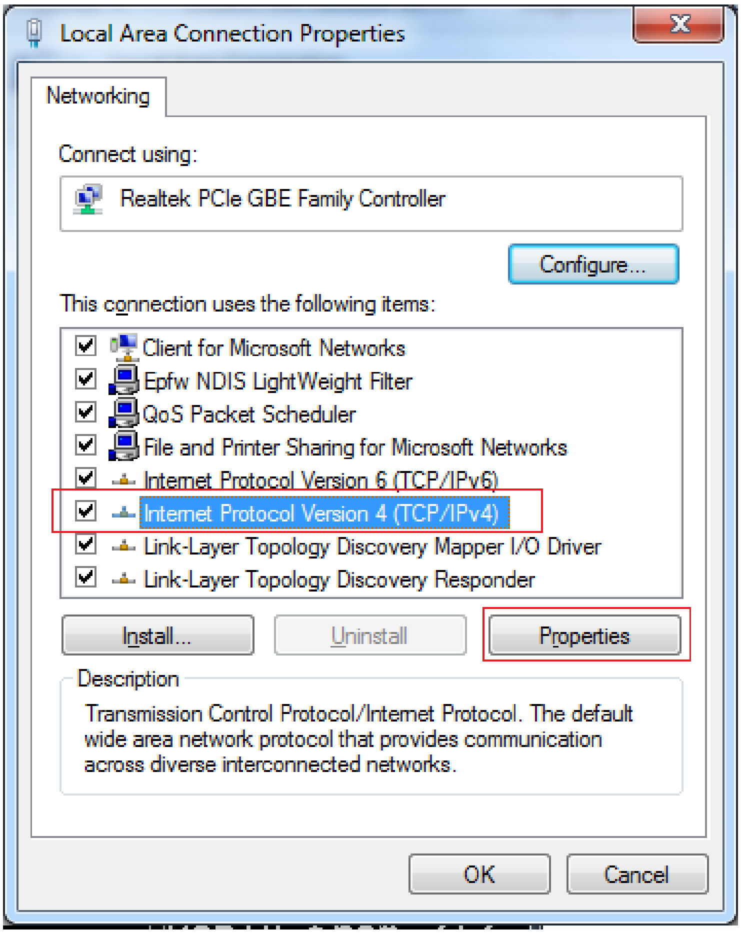
איך רכיב מקבל כתובת IP?

לצורך הדוגמה, כיצד המחשב שלי יודע מה כתובת ה-IP שלו?

יש מספר דרכים לקבל כתובת IP, ולא נדון בכולן. הפשוטה שבהן נקראת **הקצאה סטטית** של כתובת IP. על מנת לבחור כתובת IP בצורה סטטית, היכנסו ללוח הבקרה (**Control Panel**) ובחרו בניהול קישורי רשת (**Network Connections**). תוכלו לעשות זאת גם על ידי הקשה על **WinKey+R**, והקשת **ncpa.cpl**. לאחר מכן, לחצו על החיבור שלכם באמצעות המקש הימני של העכבר, ובחרו ב**מאפיינים** (או **Properties**):

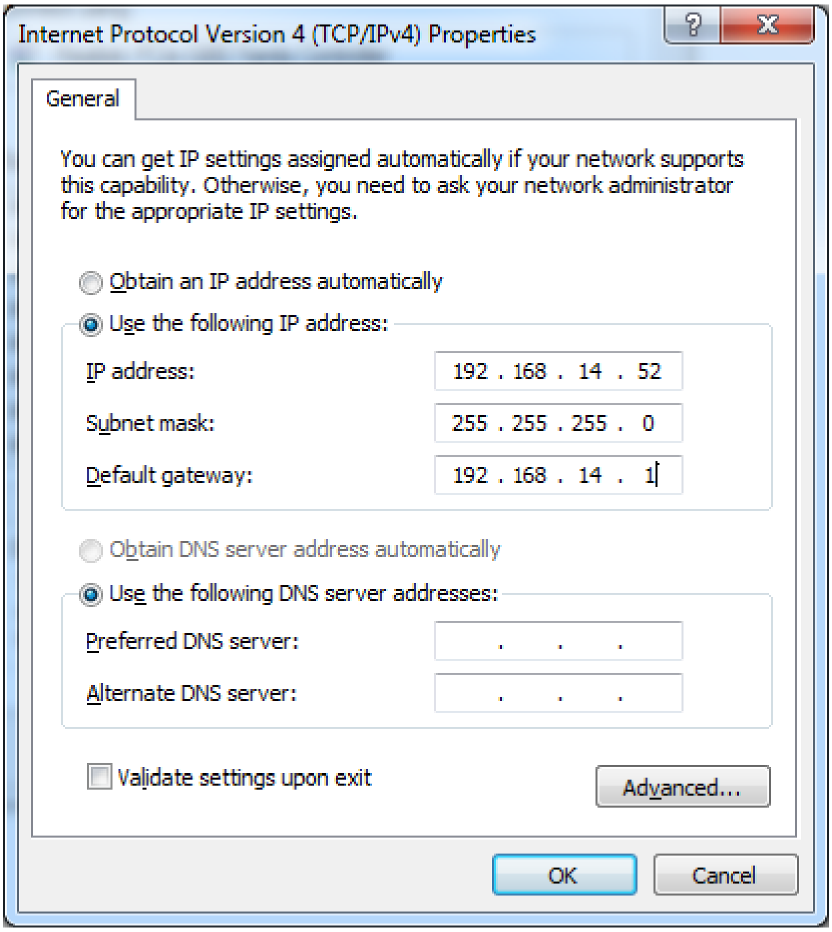


מתוך החלון שנפתח, בחרו ב-**Internet Protocol Version 4 (TCP/IPv4)**, ולחצו שוב על **מאפיינים** (או **Properties**):



כעת תוכלו לבחור באפשרות **השתמש בכתובת ה-IP הבאה** (או **Use the following IP address**), ולאחר מכן תוכלו לבחור כתובת IP, Subnet Mask וכן Default Gateway כרצונכם.

לדוגמה:



כדי שאפשרות זו תעבוד, עליכם לבחור בכתובת שלא קיימת כבר ברשת. כמו כן, שימו לב להשתמש באותן הגדרות Subnet Mask ו-Default Gateway שהיו לכם קודם[54].

דרך נוספת לקבל כתובת IP היא הדרך הדינאמית, שמתבצעת בדרך כלל באמצעות הפרוטוקול **DHCP (Dynamic Host Configuration Protocol)**.

כשאנו מחברים מחשב חדש לרשת, המחשב אינו יודע את כתובת ה-IP שלו, ולכן עליו לברר אותה. הדבר הראשון שהוא יעשה לשם כך, הוא שליחת הודעה בשם **DHCP Discover**.

בבקשה זו, המחשב למעשה פונה לעולם ומבקש: "שלום, אין לי כתובת IP. האם תוכלו לעזור לי? האם יש כאן שרת DHCP שיכול להקצות לי כתובת IP?"

---

[54] ניתן להגדיר כתובת IP בצורה זו רק עבור כתובת פרטית שנמצאת בשימוש רק בתוך הרשת שלכם, ולא עבור כתובות חיצוניות. כלומר, אם מדובר בכתובת שצריכה להיות מוקצית על ידי ספק האינטרנט – תהליך שינוי הכתובת מסובך בהרבה. לפרטים נוספים קראו את הנספח על כתובות פרטיות ו-NAT.

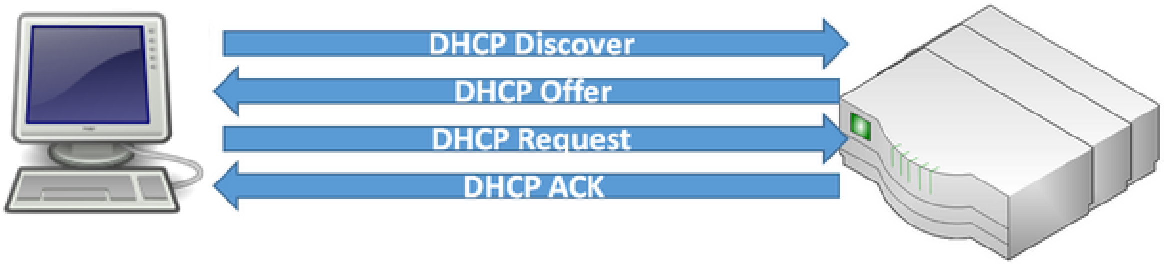
בקשה זו נשלחת כמובן ב-Broadcast, כלומר לכל הישויות ברשת. המטרה היא ששרת DHCP שיראה את הבקשה יוכל לענות אליה, בעוד ישויות אחרות יתעלמו מבקשה זו.

כעת, נאמר וישנם שני שרתי DHCP ברשת. שניהם יוכלו לענות למחשב המבקש כתובת IP עם הצעה. להודעה זו קוראים **DHCP Offer**.
בהצעה, השרת כותב למחשב איזו כתובת IP הוא יכול לקבל, וכן פרטים נוספים – כגון משך הזמן אותו הוא מבטיח להקצות את כתובת ה-IP הזו למחשב המבקש ולא לאף ישות אחרת ברשת. הודעה זו נשלחת אף היא לכתובת Broadcast – שכן אין עדיין כתובת IP למחשב שאמור לקבל את הבקשה.

המחשב קיבל שתי הצעות שונות לכתובות IP. עליו לבחור באחת מהן, ואז לשלוח בקשה לקבל באמת את הכתובת הזו. הודעה זו נקראת **DHCP Request**.
בחבילת הבקשה, הלקוח מציין מי שרת ה-DHCP ממנו הוא רוצה לקבל את הכתובת, כמו גם את הפרטים אשר השרת הציע לו. גם הודעה זו נשלחת ב-Broadcast, וזאת על מנת ששאר השרתים ידעו שהמחשב בחר בשרת הספציפי הזה, ולא ישמרו עבורו את הכתובת. לדוגמה, אם המחשב קיבל הצעה משרת DHCP_A והצעה נוספת מהשרת DHCP_B, והוא בחר בהצעה של שרת DHCP_A, כאשר הוא שולח את הודעת ה- **DHCP Request** שלו לכולם, גם השרת DHCP_B יראה אותה ויבין שמחשב א' לא בחר בו.

לבסוף, השרת צריך להחזיר למחשב אישור סופי שהוא אכן הופך להיות שרת ה-DHCP שלו. להודעה זו קוראים בשם **DHCP ACK**.
בהודעה זו, השרת חוזר על הפרטים שהוא נותן ללקוח. החל מעכשיו, הלקוח יכול להשתמש בכתובת ה-IP שניתנה לו באמצעות שרת ה-DHCP.

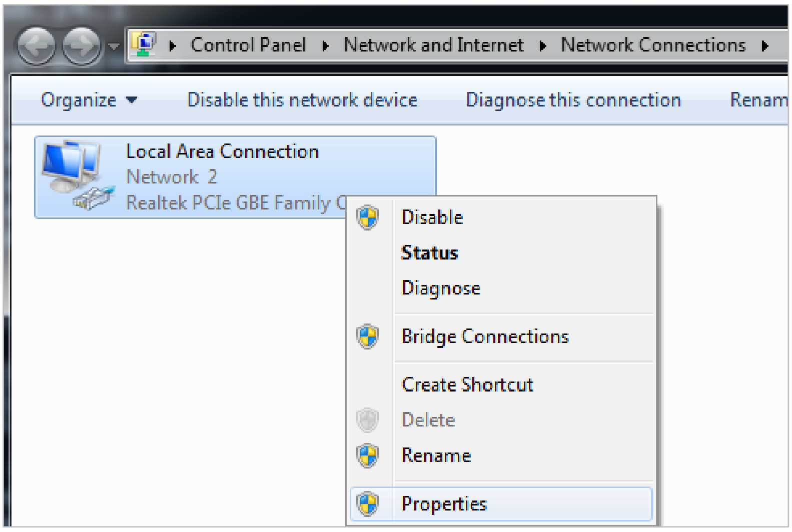
לסיכום, התהליך נראה כך:



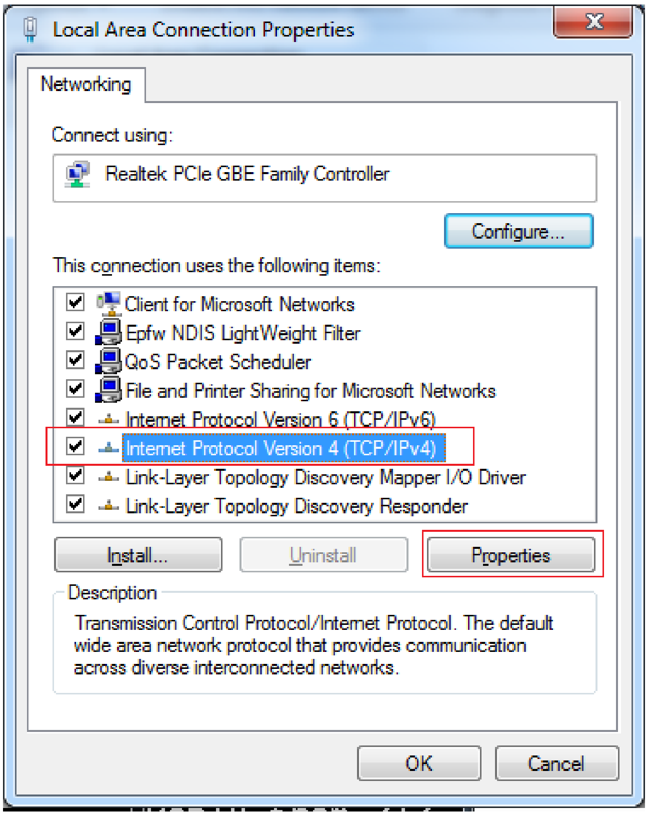
לשימוש ב-DHCP יתרונות רבים. בין השאר, הוא מאפשר לחסוך בכתובות IP בכך שהוא מקצה כתובות ללקוחות רק כשהם זקוקים להן, ולא כל הזמן. בנוסף, באמצעות DHCP ניתן לתת פרטי קונפיגורציה נוספים ללקוחות מלבד לכתובות IP (למשל – מי שרת ה-DNS שישמש את הלקוח). על כך לא נרחיב בספר זה.

תרגיל 7.9 מודרך – קבלת IP באמצעות DHCP

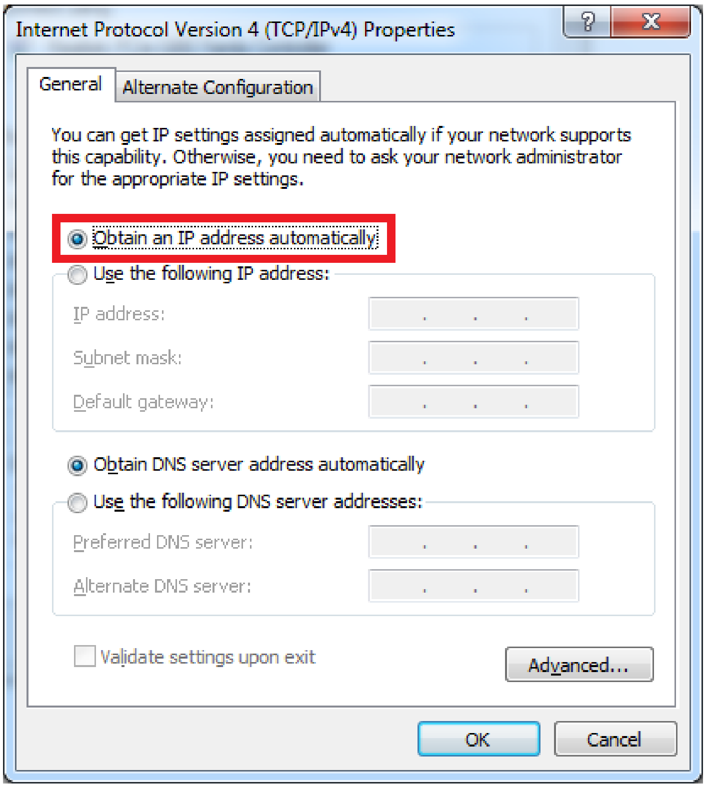
כעת נקבל בעצמנו כתובת IP משרת ה-DHCP שלנו. ראשית, עלינו לוודא שכרטיס הרשת שלנו מקבל כתובת IP בצורה דינאמית ולא סטטית. על מנת לעשות זאת, היכנסו ללוח הבקרה (**Control Panel**) ובחרו בניהול קישורי רשת (**Network Connections**). תוכלו לעשות זאת גם על ידי הקשה על **WinKey+R**, והקשת **ncpa.cpl**. לאחר מכן, לחצו על החיבור שלכם באמצעות המקש הימני של העכבר, ובחרו ב**מאפיינים** (או **Properties**):



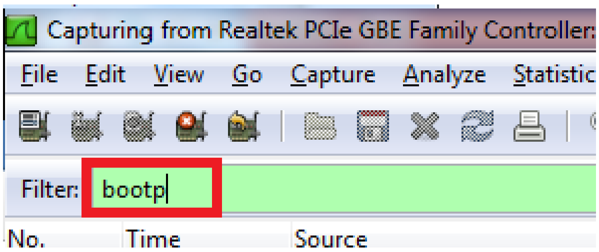
מתוך החלון שנפתח, בחרו ב-**Internet Protocol Version 4 (TCP/IPv4)**, ולחצו שוב על **מאפיינים** (או **Properties**):



כעת, ודאו כי האפשרות המסומנת הינה האפשרות **Obtain an IP address automatically**:

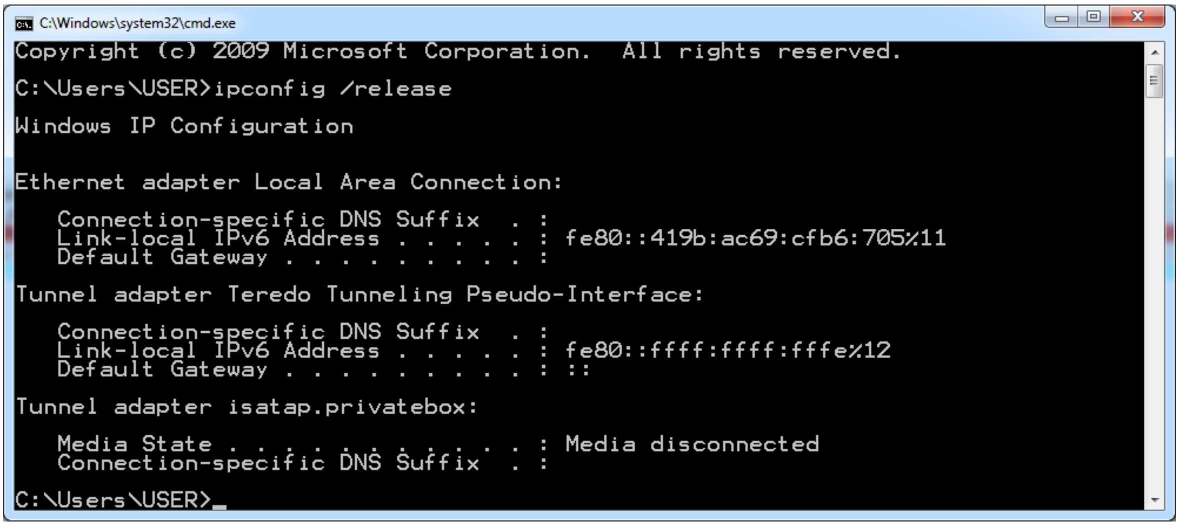


כעת הריצו את Wireshark, והשתמשו במסנן התצוגה (display filter) הבא: bootp[55].

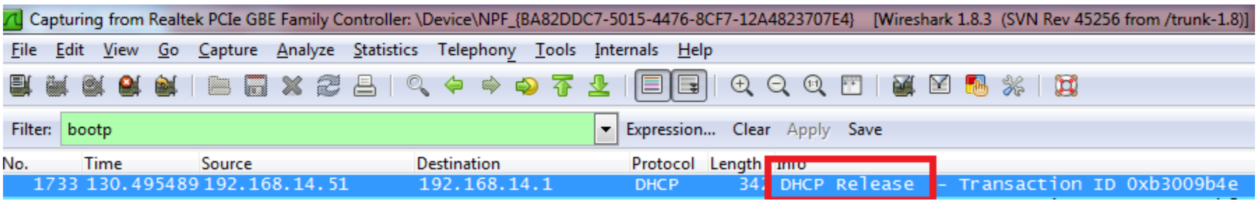


[55] DHCP הינו למעשה הרחבה של פרוטוקול ישן יותר בשם BOOTP, ו-Wireshark לא מכיר את מסנן התצוגה "dhcp". על ההבדלים בין BOOTP ל-DHCP לא נתעכב בספר זה, אך אתם מוזמנים לחפש על כך באינטרנט.

כעת היכנסו ל-Command Line, והריצו את הפקודה הבאה **ipconfig /release**:

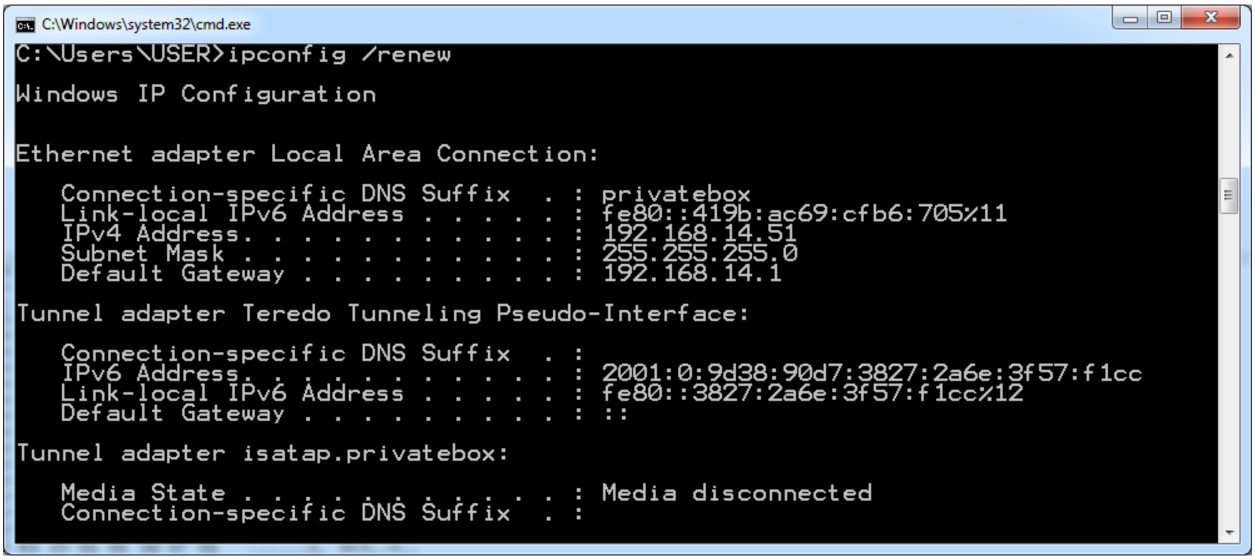


כעת אין למחשבכם כתובת IP. אם נסתכל בהסנפה, נגלה שלמעשה המחשב שלח הודעה לשרת ה-DHCP שמציין כי הוא מבקש להתנתק ממנו:

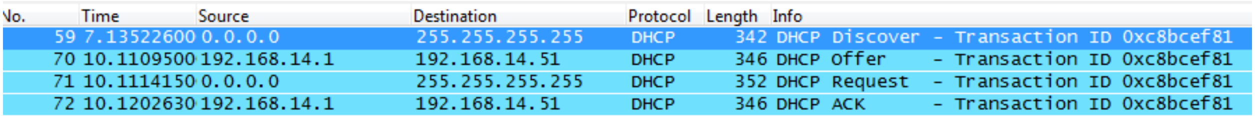


לא נתעכב על הודעה זו.

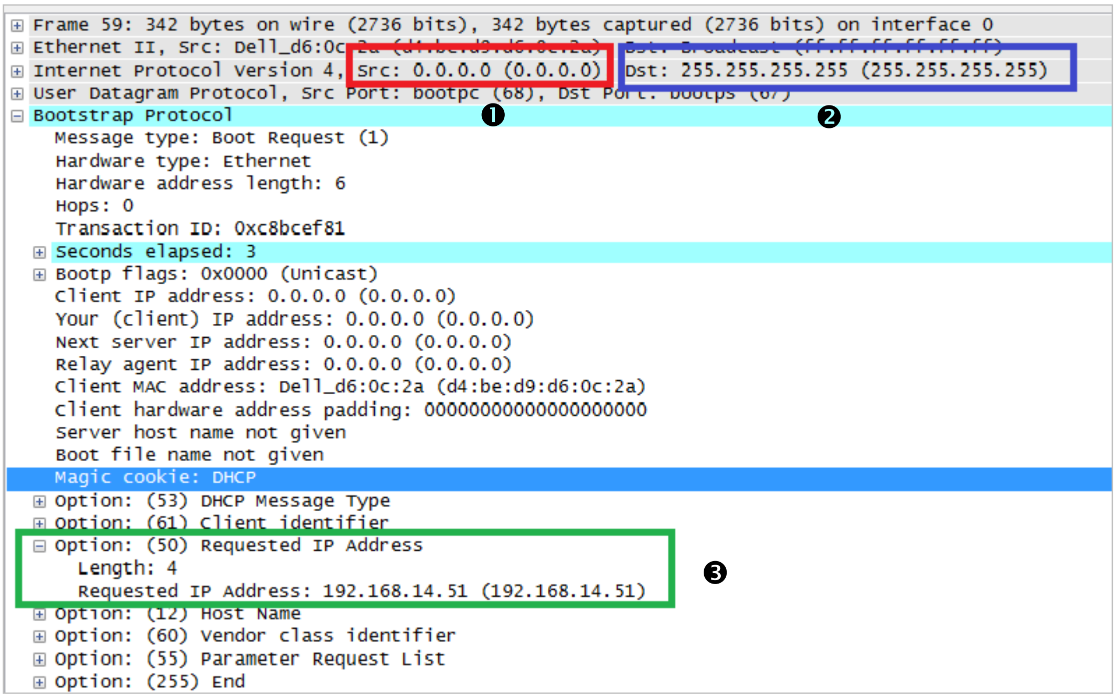
כעת, נסו לקבל כתובת IP חדשה. לשם כך, הריצו את הפקודה: **ipconfig /renew**.



כעת הביטו בהסנפה ומצאו את החבילות הרלוונטיות:



נעבור בקצרה על החבילות. הסתכלו על החבילה הראשונה, חבילת ה-**DHCP Discover**. להזכירכם, זוהי חבילה שהמחשב שולח כדי למצוא שרתי DHCP ולבקש מהם לתת לו הצעה עם כתובת IP.



נתחיל ממודל השכבות:

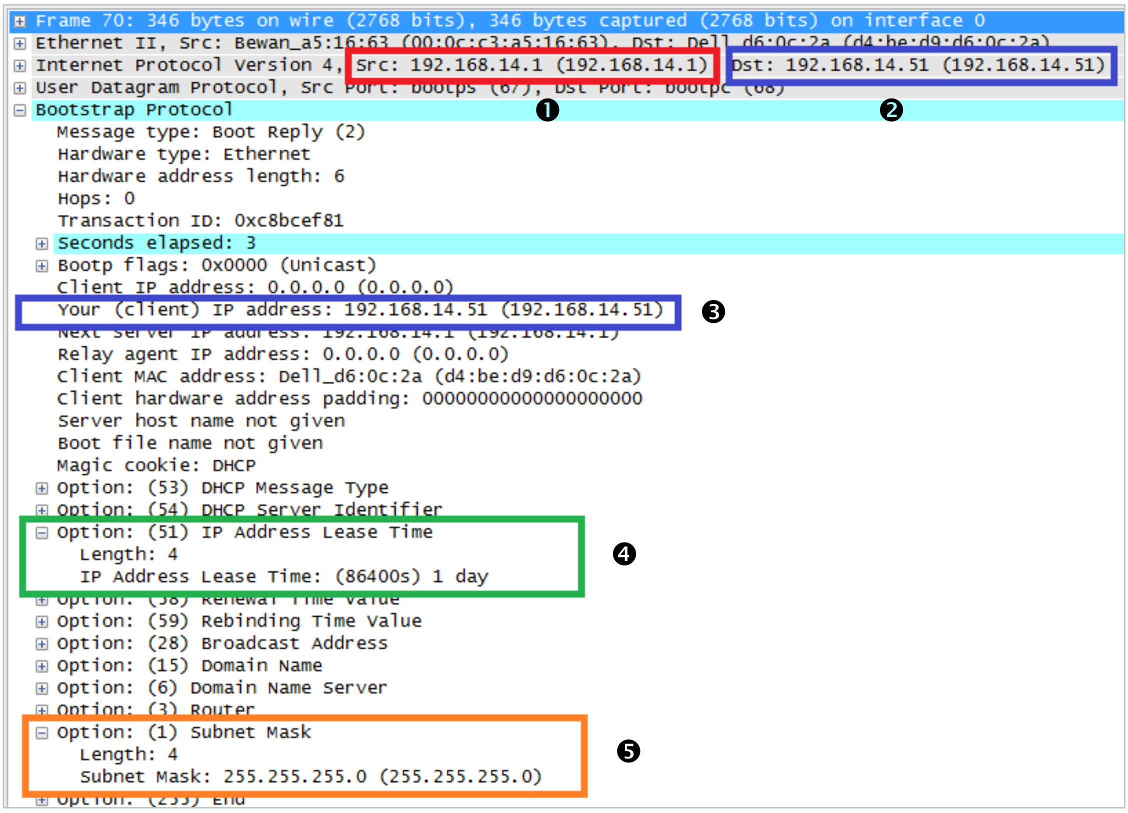
- בשכבה השנייה, ניתן לראות את השימוש בפרוטוקול Ethernet[56]. דבר זה מלמד כי כרטיס הרשת ממנו נשלחה החבילה הוא כרטיס מסוג Ethernet.
- בשכבה השלישית, יש שימוש בפרוטוקול IP. שימו לב שכתובת המקור המסומנת ב**אדום** (1) היא הכתובת: 0.0.0.0, שמציינת כי החבילה נשלחה מ"המחשב שלי". היות שלמחשב שלנו עדיין אין כתובת IP, הוא משתמש בכתובת הזו. כתובת היעד המסומנת ב**כחול** (2) היא הכתובת: 255.255.255.255, המציינת כי החבילה נשלחת ב-Broadcast, כלומר לכלל הישויות ברשת, וזאת מכיוון שהמחשב מנסה להגיע לכלל שרתי ה-DHCP, והוא אינו יודע את הכתובות שלהם.

[56] כמו שציינו קודם לכן – על השכבה השנייה בכלל, ופרוטוקול Ethernet בפרט, נרחיב בפרק הבא.

- בשכבה הרביעית, יש שימוש בפרוטוקול UDP, עליו למדנו בפרק הקודם. ישנן מספר סיבות להעדפת פרוטוקול זה על פני TCP במקרה של DHCP. ראשית, פרוטוקול DHCP הוא פרוטוקול מסוג "בקשה-תשובה", כלומר: אם בקשה אחת "תלך לאיבוד", ניתן פשוט לשלוח בקשה נוספת. עם זאת, הסיבה המהותית יותר, היא שאנו מבקשים לשלוח הודעות ב-Broadcast, דבר אשר לא אפשרי בפרוטוקול TCP, כפי שלמדנו בפרק שכבת התעבורה.
- בשכבה החמישית, יש שימוש בפרוטוקול DHCP, אשר Wireshark מזהה עבורנו כפרוטוקול BOOTP (או בשמו המלא – Bootstrap Protocol).

לא נתעכב על כלל השדות של פרוטוקול DHCP, אך נשים לב לנקודה מעניינת: אחד השדות אשר הלקוח שולח, ומסומן בירוק (3), הוא השדה **Requested IP Address**. היות שלא מדובר במחשב חדש ברשת, המחשב זוכר את כתובת ה-IP שהייתה לו קודם לכן, ומבקש לקבל אותה שוב.

החבילה הבאה הינה חבילת ההצעה של השרת: **DHCP Offer**.



מודל השכבות זהה ולכן לא נתעכב עליו. עם זאת, שימו לב לכתובות ה-IP בשימוש. כתובת המקור של החבילה, המסומנת באדום (1), היא כתובת ה-IP של שרת ה-DHCP השולח את ההצעה. בכחול (2), נמצאת כתובת ה-IP אשר הלקוח ביקש. דבר זה אפשרי רק כאשר הלקוח מציין במפורש את הכתובת שאותה הוא רוצה, כמו שראינו בשלב הקודם. במקרים אחרים, כתובת זו תהיה "0.0.0.0".

גם בשכבת ה-DHCP, נמצאת ב**כחול** (3) כתובת ה-IP אשר המחשב ביקש. כאן היא נמצאת תחת השדה **Your (client) IP address**, כלומר – זו הכתובת שהשרת מציע ללקוח.
כפי שניתן לראות, השרת שולח פרטים נוספים רבים. חשוב בשלב זה לשים לב לשדה ה-**Lease Time**, אשר מסומן בצבע **ירוק** (4). המשמעות שלו היא הזמן שבו מובטח ללקוח שכתובת ה-IP המוצעת מוקצה עבורו ולא עבור אף אחד אחר. כלומר, לאחר מעבר הזמן הזה (בדוגמה שלנו – יום אחד), באם הלקוח לא ביצע חידוש של ה-**Lease Time**, אסור לו להמשיך ולהשתמש בכתובת ה-IP הזו, שכן ייתכן שהיא מוקצה לישות אחרת.

ב**כתום** (5) ניתן לראות ששרת ה-DHCP מספק גם את ה-**Subnet Mask**. דבר זה הגיוני בהתאם למה שלמדנו במהלך הפרק – על המחשב לדעת לא רק מה כתובת ה-IP שלו, אלא גם מה מזהה הרשת שלו, ולשם כך עליו להכיר את ה-Subnet Mask שלו. כעת אנו מבינים שהמחשב יודע אותה באמצעות הודעת ה-DHCP.

לאחר מכן נשלחת הודעת ה-**DHCP Request**, המציינת בפני שרת ה-DHCP הזה (וגם בפני שרתים נוספים, אם יש כאלו), שהלקוח בחר בו ומעוניין להשתמש בהצעה שלו. היות שהודעה זו חוזרת באופן כמעט מלא על הודעות קודמות, לא נתעכב עליה.

לבסוף, נשלחת הודעת ה-**DHCP ACK**, המציינת שהשרת קיבל את הבקשה של הלקוח, וכי הוא רשאי להשתמש בכתובת ה-IP שהוקצתה עבורו. היות שהודעה זו חוזרת באופן כמעט מלא על הודעות קודמות, לא נתעכב עליה.

בתום תהליך זה, המחשב שלנו יודע מה כתובת ה-IP שלו ומה ה-Subnet Mask הרלוונטית. כמו כן הוא גילה פרטים חשובים נוספים על הרשת, כגון ה-Default Gateway שלו, שרת ה-DNS בו עליו להשתמש ועוד. עכשיו, לאחר שהושלם תהליך ה-DHCP, הוא יכול להשתמש בכרטיס הרשת שלו ולצאת לתקשר עם העולם.

שכבת הרשת – סיכום

במהלך פרק זה למדנו להכיר את שכבת הרשת. התחלנו מלהבין לעומק את תפקידה במודל השכבות, וכיצד היא משתלבת בשכבות עליהן למדנו עד כה. למדנו על האתגרים בפניה עומדת שכבת הרשת, ועל דרכים בהן היא מתמודדת איתם.

הכרנו את **פרוטוקול IP**, הפרוטוקול של האינטרנט, והתעמקנו ב**כתובות IP** והמבנה שלהן. לאחר מכן הכרנו את מונח ה**ניתוב**, כמו גם את הרכיב **נתב**. במהלך הפרק השתמשנו בכלים שונים כגון **ping**, **tracert**, **ipconfig** ו-**route**.

הכרנו גם פרוטוקולים נוספים, כגון **פרוטוקול ICMP**. באמצעות למידת פרוטוקול זה הבנו כיצד ממומשות הפקודות Ping ו-Traceroute ולמדנו לממש אותן בעצמנו. לבסוף, למדנו על דרכים לקבל כתובות IP, והרחבנו בעיקר על **פרוטוקול DHCP**.

בפרק הבא, נרד שכבה נוספת במודל השכבות ונלמד להכיר את **שכבת הקו**. נדבר על האתגרים הניצבים בפני שכבה זו, וכיצד היא מתקשרת לשכבת הרשת עליה למדנו עתה.

שכבת הרשת – צעדים להמשך

על אף שלמדנו רבות על שכבת הרשת, נותרו נושאים רבים בהם לא נגענו. לא הסברנו כמעט בכלל כיצד מתבצעות החלטות הניתוב – כלומר איך הנתבים בונים את טבלאות הניתוב שלהם. לא הרחבנו על שיטות להתמודד עם עומסים ברשת, ולא נגענו בבקרת איכות (Quality Of Service). לא סיפרנו על אבטחה ב-IP (באמצעות IPSec), ולא דיברנו על אפשרויות נוספות של השכבה.

אלו מכם שמעוניינים להעמיק את הידע שלהם בשכבת הרשת, מוזמנים לבצע את הצעדים הבאים:

קריאה נוספת

בספר המצוין Computer Networks (מהדורה חמישית) מאת Andrew S. Tanenbaum ו- David J. Wetherall, הפרק החמישי מתייחס במלואו לשכבת הרשת. באופן ספציפי, מומלץ לקרוא את הסעיפים:

- 5.2.1-5.2.6 – ניתובים ואלגוריתמי ניתוב.
- 5.3 – בקרת עומסים.
- 5.4.1-5.4.2 – בקרת איכות.
- 5.6.6-5.7.7 – על אלגוריתמי הניתוב שבשימוש באינטרנט.

בספר Computer Networking: A Top-Down Approach (מהדורה שישית) מאת James F. Kurose, הפרק הרביעי מוקדש כולו לשכבת הרשת. באופן ספציפי, מומלץ לקרוא את הסעיפים:

- 4.3 – התהליך שקורה בנתב.
- 4.5 – ניתובים ואלגוריתמי ניתוב.
- 4.6 – על אלגוריתמי הניתוב שבשימוש באינטרנט.
- 4.7 – ניתוב Broadcast ו-Multicast.

כמו כן, ניתן להרחיב את אופקיכם בפרק על IP מתוך The TCP/IP Guide, אותו ניתן למצוא בכתובת: http://goo.gl/iggBmk.

תרגיל 7.10 – פרגמנטציה של IP (אתגר)

בתרגיל זה תממשו תקשורת מעל פרגמנטציה של IP (על פרגמנטציה תוכלו לקרוא בנספח א' של פרק זה).

השתמשו במודול **socket** של פייתון כדי לכתוב שרת פשוט המאזין על פורט 55555 לחבילות UDP נכנסות. על הסקריפט להדפיס למסך כל חבילת מידע שהוא מקבל.

כעת כתבו סקריפט המקבל מהמשתמש מסר שעליו לשלוח (כלומר, מחרוזת – למשל: "hello world"), ומספר פרגמנטים. עליכם לשלוח את המסר שהלקוח שלח אל שרת ה-UDP שכתבתם קודם לכן, ולחלק אותו באמצעות פרגמנטציה של IP למספר הפרגמנטים שהלקוח ביקש. כך למשל, אם הלקוח ביקש לשלוח את המסר "hello world" בשלושה חלקים, תוכלו לשלוח אותו כך:

- חלק ראשון – "hel"
- חלק שני – "lo worl"
- חלק שלישי – "d"

כמובן שתוכלו לשנות את מספר התווים שנשלחים בכל חלק.

ודאו כי השרת מצליח להדפיס נכונה את ההודעה ששלחתם לו. כמו כן, הסניפו באמצעות Wireshark וודאו שאתם שולחים מספר נכון של פרגמנטים. שימו לב שעל מנת לבדוק תרגיל זה, עליכם להשתמש בשני מחשבים שונים – אחד ללקוח ואחד לשרת[57].

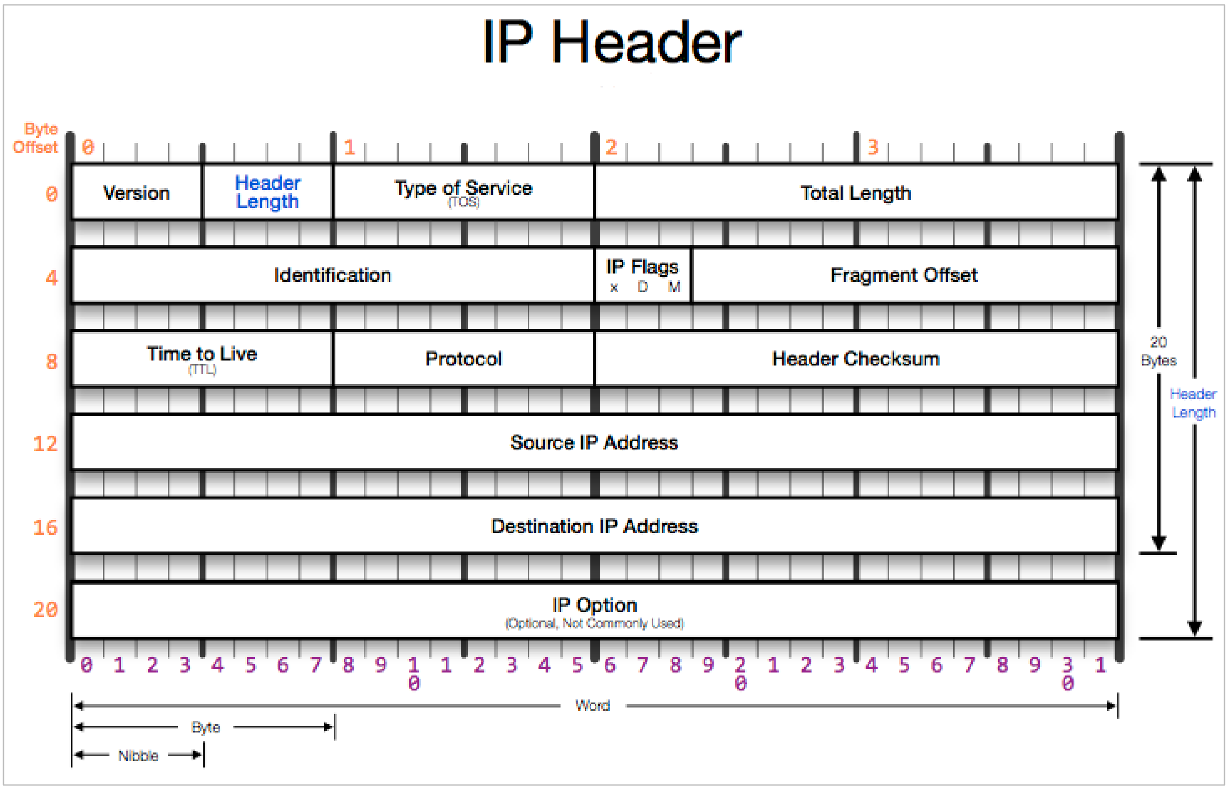
<u>דגשים</u>

- יש לבנות ולשלוח את החבילות באמצעות Scapy.
- ה-Header של UDP צריך להופיע רק פעם אחת, אין צורך לחזור עליו בכל פרגמנט מחדש.

[57] באופן תאורטי, יכולנו לעשות זאת מעל loopback device – כלומר מעל הכתובת "127.0.0.1", המוכרת לנו מתרגילים קודמים. עם זאת, עקב Bug של Scapy בשליחה וקבלת מסגרות מעל loopback device ב-Windows, נשתמש בשני מחשבים.

נספח א' – IP Header

נספח זה נועד כדי לתאר את כלל השדות של ה-Header של IPv4. לא חיוני להבין את כל השדות עד הסוף. מידע נוסף ניתן למצוא בכתובת: http://en.wikipedia.org/wiki/IPv4_header#Header.



- **Version** – שדה זה מתאר את גרסת ה-IP. לדוגמה, עבור פקטה של IPv4, השדה יכיל את הערך 4. עבור פקטה של IPv6, השדה יכיל את הערך 6.
- **Header Length** – מתאר את אורך ה-Header של החבילה, היות שהוא עשוי להשתנות מחבילה לחבילה, מכיוון שישנם שדות של IP Options אותם נתאר בהמשך. הערך של השדה מתאר את הגודל ביחידות מידה של 32 ביטים. כך למשל, עבור חבילה בגודל 20 בתים, הערך כאן יהיה 5 (מכיוון שמדובר ב-160 ביטים, שהם 5 פעמים 32 ביטים). הערך 5 (שמתאר 20 בתים) הוא הערך הנפוץ ביותר עבור שדה זה. שדה זה דומה מאוד לשדה ה-Header Length של פרוטוקול TCP, עליו למדנו בפרק שכבת התעבורה.
- **Type of Service (TOS)** – לשדה זה היו משמעויות שונות לאורך השנים והוא גם הוגדר מחדש. במקור, השדה איפשר לציין את העדיפות של חבילה מסוימת על פני חבילות אחרות, זאת כדי לבקש מהנתבים להעביר חבילות בעלות עדיפות גבוהה לפני חבילות בעלות עדיפות נמוכה יותר. בפועל, לא היה שימוש נרחב בשדה זה.

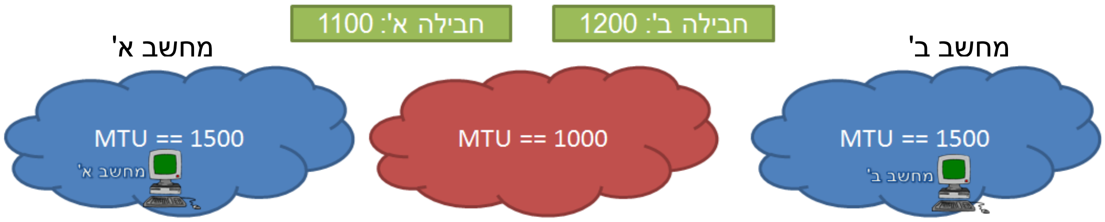
- **Total Length** – שדה זה מציין את הגודל, בבתים, של כלל החבילה בשכבת ה-IP (כולל ה-Header והמידע).

שימו לב – הגודל המינימלי של חבילת IP הוא 20 בתים, שכן זהו הגודל המינימלי של התחילית.

הגודל המקסימלי הוא 65,535 בתים – שכן השדה Total Length הוא באורך של 16 ביטים.

בטרם נמשיך לשדות הבאים, עלינו להסביר שני מונחים חדשים: **פרגמנטציה (Fragmentation)** ו-**MTU.** רשתות שונות יכולות לטפל בגודל שונה של חבילות. כך למשל, ייתכן שרשת אחת משתמשת בפרוטוקול Ethernet של השכבה השנייה – שכבת הקו, ולכן תוכלו להעביר חבילות עד גודל של 1,500 בתים, ולא יותר מכך. אי לכך, נאמר שה-**MTU (Maximum Transmission Unit)** של רשת זו הינו 1,500 בתים.

נביט בתמונת הרשת הבאה:

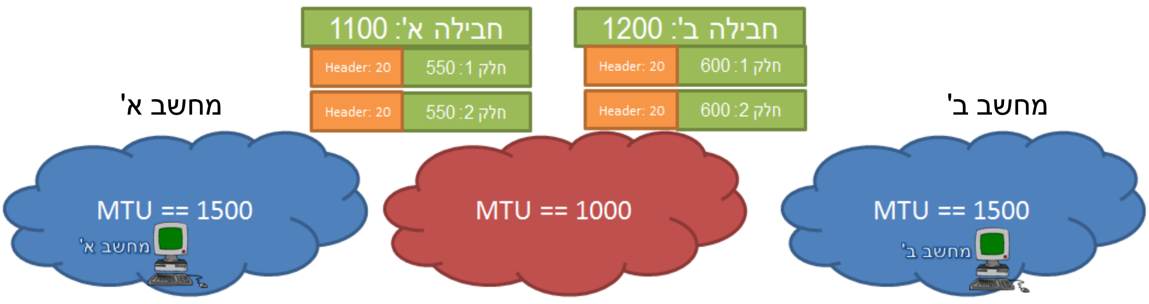


כאן מחשב א' נמצא ברשת שה-MTU שלה הוא 1,500 בתים – כלומר, ברשת ניתן להעביר חבילות עד גודל של 1,500 בתים ולא יותר מכך. מחשב א' רוצה להעביר למחשב ב' שתי חבילות: האחת בגודל 1,100 בתים, והשנייה בגודל 1,200 בתים.

מחשב ב' נמצא אף הוא ברשת שה-MTU שלה הוא 1,500 בתים.

עם זאת, על מנת להגיע ממחשב א' למחשב ב', החבילות צריכות לעבור ברשת שה-MTU שלה הוא 1,000 בתים בלבד (הרשת האדומה, האמצעית). מכאן שלא חבילה א', ולא חבילה ב' יכולות לעבור בה. אם חבילה שכזו תגיע לרשת האדומה, היא צפויה להיזרק.

על מנת לפתור את סוגיה זו, ניתן לבצע **פרגמנטציה** של החבילות: נחלק אותן לשני חלקים (**פרגמנטים**), שכל אחד מהם בגודל של 1,000 בתים או פחות. נביט בתמונה הבאה:



כאן חבילה א' התחלקה לשני חלקים, שכל אחד מהם בגודל של 550 בתים של מידע, ועוד 20 בתים של Header.
חבילה ב' התחלקה אף היא לשני חלקים, שכל אחד מהם בגודל של 600 בתים של מידע, ועוד 20 בתים של Header.

כעת יש ארבעה חלקי חבילות, שאף אחד מהם לא עובר את גודל ה-MTU המותר ברשת האדומה (באמצע) . לכן, מחשב א'[58] יכול לשלוח את כולן מבלי להיתקל בבעיית ה-MTU. כאשר החלקים יגיעו למחשב ב' ברשת הכחולה, יהיה עליו להבין איזה חלקים היו שייכים לאיזו חבילה, לחבר אותם מחדש ולקבל את החבילות שמחשב א' ניסה לשלוח אליו במקור.
ניתן לקרוא מידע נוסף על פרגמנטציה בכתובת: http://en.wikipedia.org/wiki/IP_fragmentation.

כעת נמשיך להסביר על השדות השונים של IP Header:

- **Identification** – שדה זה משומש במקרה של פרגמנטציה. בדוגמה לעיל, מחשב ב' צריך לדעת להבדיל בין חלק 1 של חבילה א' לבין חלק 1 של חבילה ב', כדי לדעת להרכיב נכון את החלקים. אי לכך, גם לחלק 1 וגם לחלק 2 של חבילה א' יהיה את אותו המזהה בשדה ה-Identification (לדוגמה – המזהה 100), בעוד לחלק 1 וחלק 2 של חבילה ב' יהיה מזהה אחר (לדוגמה – המזהה 200).
- **Flags** – דגלים שונים לשימוש. מוגדרים שלושה דגלים:
  - Reserved – ביט זה נשמר תמיד על הערך 0.
  - Don't Fragment (DF) – אסור לבצע פרגמנטציה לחבילה שנשלחת עם הדגל הזה דולק[59]. בדוגמה שלעיל, במידה שחבילה א' נשלחה עם הדגל DF דולק, לאף נתב בדרך אסור לפצל אותה ולבצע פרגמנטציה. אם כך, החבילה לא תוכל להישלח למחשב ב', ותשלח על כך הודעת שגיאה.
  - More Fragments (MF) – דגל זה משמש במקרה של פרגמנטציה. במידה שנשלחת חבילה מפוצלת, בכל fragment שאינו ה-fragment האחרון בחבילה, הביט הזה יהיה דולק. בדוגמה לעיל, בחלק 1 של חבילה א' הדגל יהיה דולק (מכיוון שיש גם את חלק 2). בחלק 2 של חבילה א' הדגל יהיה כבוי (כיוון שהוא החלק האחרון של החבילה). באופן דומה, בחלק 1 של חבילה ב' הדגל יהיה דולק, ובחלק 2 של חבילה ב' הדגל יהיה כבוי.
- **Time To Live (TTL)** – נועד כדי למנוע מחבילות להסתובב לנצח ברחבי הרשת. הרחבנו על שדה זה תחת ההסבר על פרוטוקול ICMP בפרק זה.

---

[58] בפועל, בחלק גדול מהמקרים, יהיה זה אחד הנתבים בדרך שיבצע את הפרגמנטציה ולא מחשב הקצה.
[59] המשמעות של "דגל דולק" היא שערך הביט הוא 1. "דגל כבוי" משמעותו שערך הביט הוא 0.

- **Protocol** – שדה זה מתאר מהו הפרוטוקול שנמצא מעל שכבת ה-IP. לדוגמה, הערך "6" מציין שהשכבה שמעל לשכבת ה-IP היא שכבת TCP. הערך "17" מציין שמדובר בשכבת UDP.
- **Header Checksum** – נועד לוודא את תקינות ה-Header של החבילה (שימו לב שווידוא התקינות מתבצע על ה-Header בלבד, ולא על המידע של החבילה). כאשר חבילה מגיעה אל ישות כלשהי ברשת, היא מחשבת את ערך ה-Checksum של ה-Header ומשווה אותו לערך שמצוי בשדה ה-Checksum. במידה שהערכים לא זהים, יש לזרוק את החבילה. להזכירכם, למדנו על Checksum בפרק שכבת התעבורה/ מה זה Checksum ?.
- **Source Address** – כתובת המקור של החבילה, כלומר כתובת ה-IP של שולח החבילה.
- **Destination Address** – כתובת היעד של החבילה, כלומר כתובת ה-IP של היעד הסופי.
- **Options** – שדה זה מאפשר ל-Header להיות גמיש ולכלול בתוכו אפשרויות נוספות. השימוש בו נדיר למדי.

נספח ב' – IPv6

כפי שלמדנו בפרק זה כאשר הסברנו את נושא ה-NAT, בסוף שנות ה-80' נוצרה בעיה אמיתית ומוחשית – נגמרו כתובות ה-IP. הדרך התשתיתית להתמודד עם בעיה זו, היא ליצור גירסה חדשה של פרוטוקול IP, כזו שתתמוך בהרבה יותר כתובות מאשר 2^{32} הכתובות של IPv4. בנוסף, היות שפרוטוקול IP היה בשימוש כבר זמן מה, הוסקו מסקנות לגבי השימושים שלו ברשת האינטרנט, וניתן להפיק מהן לקחים וליצור גירסה טובה יותר של הפרוטוקול. לשם כך, עלתה בשנת 1995 ההצעה הראשונה לגירסה 6 של פרוטוקול IP, הידועה בשם IPv6.

בנספח זה נתאר רק חלק מהמידע הרלוונטי ל-IPv6, ונתמקד בשינויים העיקריים בינו לבין IPv4.

כתובות IPv6

ההבדל הראשון הוא, כמובן, בכתובות. כתובת של IPv4 הייתה, כזכור, באורך של ארבעה בתים (bytes), שהם 32 ביטים (bits), ומכאן 2^{32} האפשרויות השונות לכתובות של IPv4. ב-IPv6 הוחלט שכל כתובת תהיה באורך של 16 בתים (bytes), כלומר 128 ביטים (bits). אי לכך, יש 2^{128} אפשרויות לכתובות IPv6. זהו מספר עצום של כתובות שלא אמור להיגמר גם כאשר לכל מקרר, מצנם ומדיח תהיה כתובת IP משלו.

כתובות IPv6 מחולקות לסוגים: ישנן כתובות Unicast השונות מכתובות Multicast. בנספח זה נתאר כתובות Unicast בלבד. עבור כתובות אלו, 64 הביטים (bits) העליונים מציינים את **מזהה הרשת**, בעוד 64 הביטים התחתונים מציינים את **מזהה הישות**.

הכתובות מוצגות באמצעות שמונה "קבוצות" של ארבע ספרות הקסדצימליות, כאשר כל "קבוצה" מייצגת למעשה שני בתים (bytes). ה"קבוצות" מופרדות באמצעות התו נקודתיים (:). להלן דוגמה של כתובת IPv6:

2340:0000:0000:000A:0000:0000:0000:0001

מכיוון שכתובת שכזו היא ארוכה מאוד, ישנם חוקים המאפשרים להציג את הכתובת בצורה קצרה יותר, בייחוד כאשר יש שימוש באפסים. למשל, בכל קבוצה ניתן להשמיט את האפסים הראשונים. כך למשל, הקבוצה 0010, יכולה להיות מוצגת כ-10 בלבד. באמצעות חוקים אלו, ניתן להציג את הכתובת לעיל גם כך:

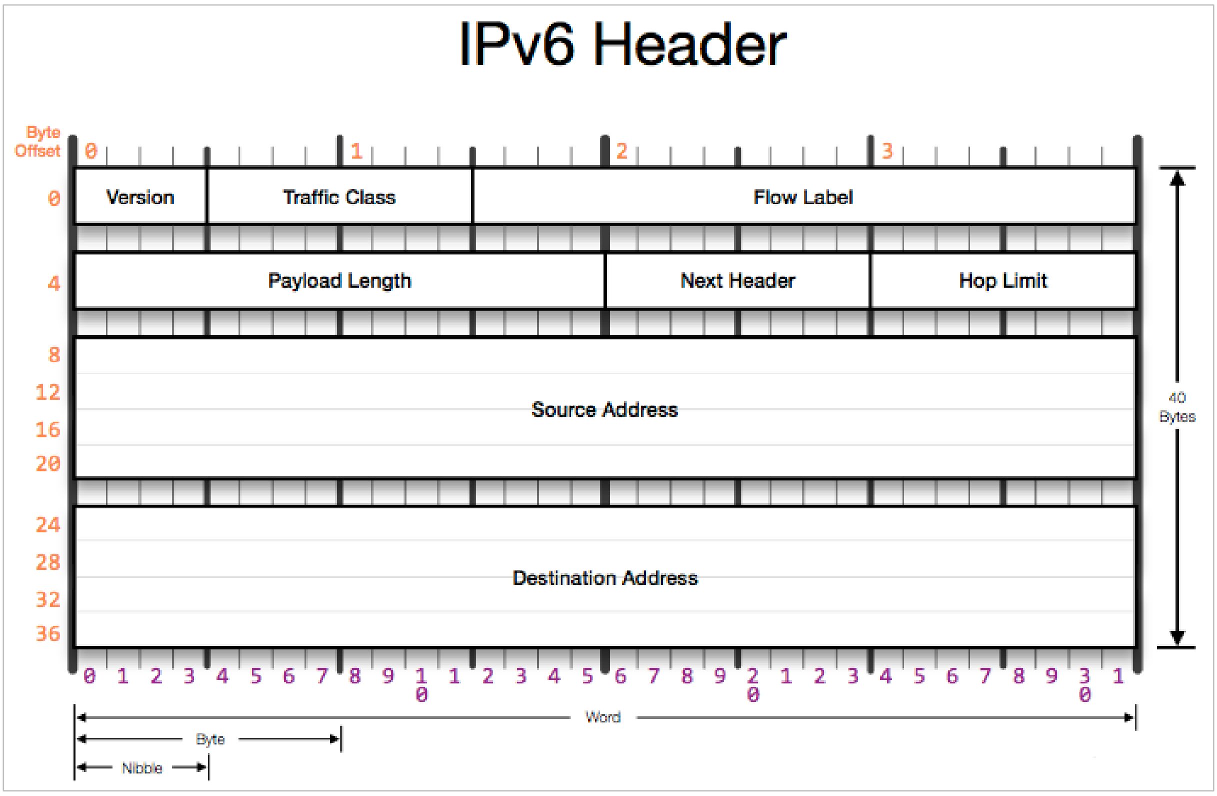
2340:0:0:A:0:0:0:1

זאת ועוד, קבוצות שכוללות רק את הספרה אפס, שמופיעות ברצף זו אחר זו, יכולות להיות מוצגות באמצעות שני תווי נקודתיים (::). עם זאת, ניתן להשתמש בשני תווי נקודתיים רק פעם אחת בכתובת. כך למשל, את הכתובת לעיל ניתן להציג גם בצורה הבאה:

2340::A:0:0:0:1

IPv6 Header

כך נראה ה-Header של חבילת IPv6:



השדות הם:

- **Version** – שדה זה מתאר את גרסת פרוטוקול IP. במקרה של IPv4, ערך שדה זה יהיה 4. במקרה זה, מכיוון שמדובר ב-IPv6, הערך יהיה 6.
- **Traffic Class** – שדה זה דומה לשדה ה-Type Of Service ב-IPv4. השדה מאפשר לציין את העדיפות של חבילה מסוימת על פני חבילות אחרות – בכדי לבקש מהנתבים להעביר חבילות בעלות עדיפות גבוהה לפני חבילות בעלות עדיפות נמוכה יותר.
- **Flow Label** – שדה זה נועד כדי לאפשר לאפיין חבילות השייכות לאותו "flow". הכוונה היא לחבילות מאותו זרם מידע. על אף ש-IP הינו פרוטוקול שלא מבוסס קישור, יש כאן דרך לשייך חבילה בודדת לקישור מלא. כך למשל, ניתן לתת את אותו ערך בשדה ה-Flow Label לכל החבילות שקשורות לשיחת VoIP מסוימת, או לתקשורת בין דפדפן לבין אתר. במידה שהחבילה לא מקושרת לאף "flow", בשדה זה יהיה הערך 0. הרעיון הוא שכל הנתבים בדרך יטפלו בכל החבילות שקשורות לאותו ה-"flow" באותה הדרך. כך, כל החבילות שקשורות לאותו קישור, ינותבו באותו האופן.
- **Payload Length** – אורך ה-Payload של החבילה. האורך כאן מדבר רק על ה-Payload, ולא כולל את ה-Header כמו ב-IPv4. הסיבה לכך היא שאורך ה-Header של IPv6 הוא קבוע.

- **Next Header** – מתאר איזה Header מגיע אחרי ה-IPv6 Header. כך למשל, ערך של 6 מזהה שה-Header הבא הוא של TCP, בעוד הערך 17 מזהה שה-Header הבא הינו Header של UDP.
- **Hop Limit** – זהה במשמעות לשדה ה-TTL ב-IPv4. ההבדל הוא רק בשם. העובדה היא ששדה ה-TTL לא היה קשור לזמן, אלא למספר הקפיצות (hops) שחבילה יכולה לעבור. לכן, Hop Limit מהווה שם מתאים יותר מאשר Time To Live.
- **Source Address** – כתובת המקור של החבילה, כלומר – כתובת ה-IPv6 של שולח החבילה.
- **Destination Address** – כתובת היעד של החבילה, כלומר – כתובת ה-IPv6 של היעד הסופי של החבילה.

הבדל משמעותי אחד בין ה-Header של IPv4 לבין ה-Header של IPv6, הוא שאורך ה-Header של IPv6 הינו קבוע ועומד תמיד על ארבעים בתים (bytes). זאת בניגוד ל-IPv4 Header, שכולל כזכור שדה של Options, שעשוי להשפיע על האורך שלו.

מעבר לכך, שימו לב שאין יותר שדה Checksum כמו שהיה ב-IPv4. הסיבה לכך היא ששנים של ניסיון הוכיחו שבדרך כלל חבילת IPv4 רצה מעל שכבה שנייה שכוללת Checksum (כגון Ethernet), ומעלה נמצאת שכבה רביעית שגם כוללת Checksum (כגון UDP או TCP, עליהם נלמד בפרק הבא). אי לכך, ה-Checksum מחושב בכל כרטיס רשת בדרך, והן על ידי מכשירי הקצה. מכאן שאין צורך לחשב את ה-Checksum גם בכל נתב ונתב. פעולת חישוב ה-Checksum הינה יקרה ומעמיסה על הנתב. אי לכך, נתבים העובדים עם IPv6 יכולים להשקיע את זמנם בניתוב של פקטות, ולא בחישוב של Checksum.

ל-IPv6 יתרונות רבים נוספים על IPv4, ביניהם תמיכה נוחה בכתובות Multicast, אפשרות לישויות לתת לעצמן כתובות IP מבלי צורך ב-DHCP (תהליך הנקרא SLAAC) ועוד. על אלו לא נרחיב בנספח זה, אך אתם מוזמנים להרחיב אופקיכם.

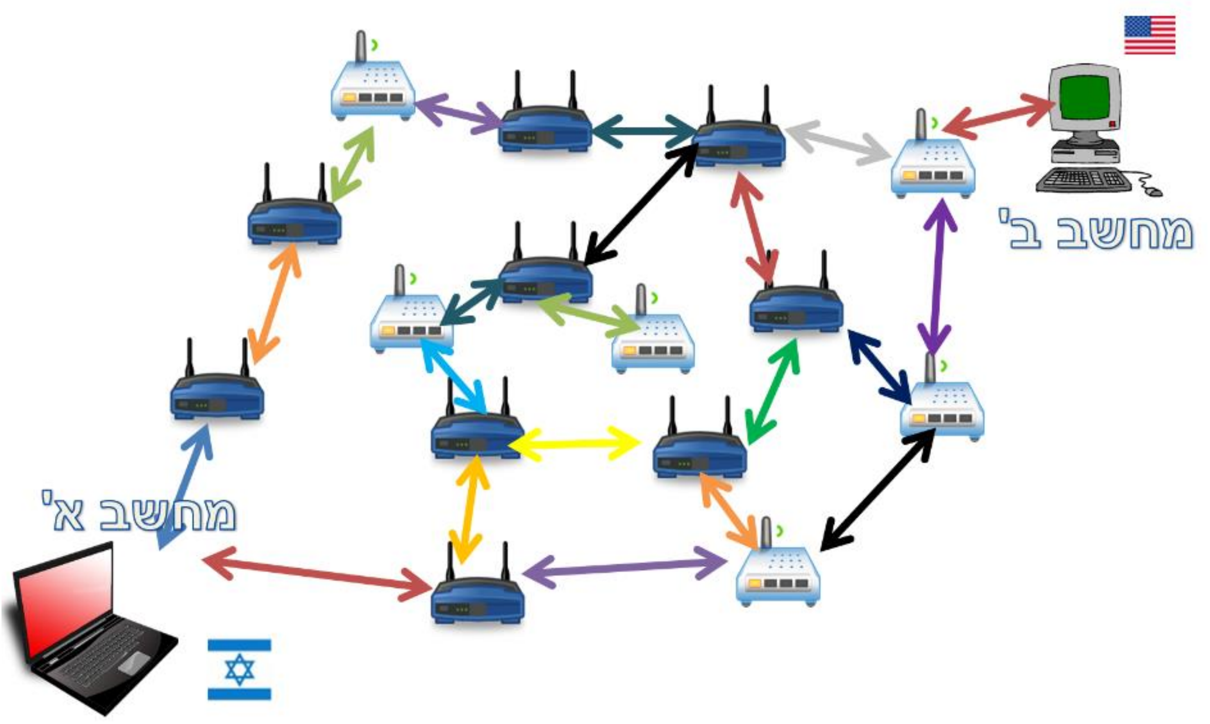
פרק 8

שכבת הקו

בפרק הקודם למדנו על שכבת הרשת, וכעת אנו מבינים שחבילות מידע שעוברות בין שתי נקודות קצה עוברות בדרך כלל בין מספר רכיבים בדרך (למשל נתבים). לכל אורך הפרק הקודם, הנחנו שניתן להעביר חבילה בין ישות אחת לישות אחרת כשאלו צמודות זו לזו. עם זאת, פעולה זו אינה כה פשוטה. במהלך הפרק הקרוב נבין את מטרתה של שכבת הקו, נלמד על פרוטוקול Ethernet, כמו גם פרוטוקול ARP, ונבין את האתגרים עימה מתמודדת השכבה.

מה התפקיד של שכבת הקו?

נביט בתמונה המלווה אותנו לאורך הספר:



בפרק הקודם ראינו ששכבת הרשת אחראית להעביר חבילה בין מחשב א' למחשב ב'. כמו כן, הבנו שהיא אחראית על המסלול שבה החבילה תעבור. השכבה השנייה אחראית על התקשורת בין כל שתי ישויות הקשורות זו לזו באופן ישיר. באיור לעיל, כל חץ צבעוני מייצג תקשורת בשכבה השנייה בין שתי ישויות – כל עוד הן מחוברות זו לזו באופן ישיר, הטיפול של העברת הודעה ביניהן שייך לשכבה זו. בשכבת הקו אין הבנה

של הדרך המלאה שהחבילה עוברת מהמקור אל היעד כמו בשכבת הרשת, אלא רק בין ישויות סמוכות – כלומר כל חץ צבעוני באיור לעיל בלבד.

מטרת שכבת הקו היא להעביר מידע בין שתי ישויות המחוברות זו לזו באופן ישיר

המשמעות של חיבור ישיר היא שמידע יכול לעבור בין הישויות מבלי לעבור בישות אחרת בדרך. חיבור ישיר יכול להיות בצורות שונות. ייתכן שמדובר בחיבור קווי – משתמשים בכבל פיזי כדי לחבר ישות אחת לאחרת. למשל, נאמר שמחשב א' מחובר לאחד הנתבים הקרובים אליו בכבל. חיבור אחר יכול להיות חיבור אלחוטי – לדוגמה, ייתכן שהנתב של מחשב א' מחובר לנתב הבא באמצעות WiFi. ייתכן אפילו והחיבור יהיה באמצעות יוני דואר – למשל, ייתכן שהתקשורת בין מחשב ב' לבין הנתב הקרוב אליו מתבצעת באמצעות יוני דואר. מקרים אלו שונים אחד מהשני מאוד, והשכבה השנייה צריכה לדאוג לכך שחבילות המידע יצליחו לעבור מישות לישות בצורה אמינה.

השכבה צריכה להתמודד עם תקלות שיכולות להיות בקו, עליהן נפרט בהמשך.

שכבת הקו מספקת לשכבת הרשת ממשק להעברת מידע בין שתי ישויות המחוברות זו לזו באופן ישיר

באופן זה, שכבת הרשת לא צריכה לדאוג לסוגיות הקשורות לחיבור בין שתי תחנות. את שכבת הרשת לא מעניין אם הישויות מחוברות בכבל, בלוויין, ב-Wifi, או באמצעות יוני דואר. היא רק אחראית להבין מה המסלול האופטימלי. כמו ש-Waze רק אומרת לרכב באיזו דרך לעבור, ולא מסבירה לנהג שהוא צריך לתדלק, ללחוץ על הגז, לאותת ולעצור ברמזור או להולך רגל. בזה יטפל הנהג, או במקרה שלנו – שכבת הקו.

איפה ממומשת שכבת הקו?

המימוש של שכבת הקו "נמצא" בכל ישות ברשת – וספציפית, בכרטיס הרשת של הישות. כך למשל כרטיס Ethernet יממש את פרוטוקול Ethernet, וכרטיס WiFi יממש את פרוטוקול WiFi. כך כרטיסי רשת שונים מתקשרים זה עם זה. עם זאת, כרטיס רשת Ethernet לא יכול לתקשר ישירות עם כרטיס רשת של WiFi.

פרוטוקול Ethernet

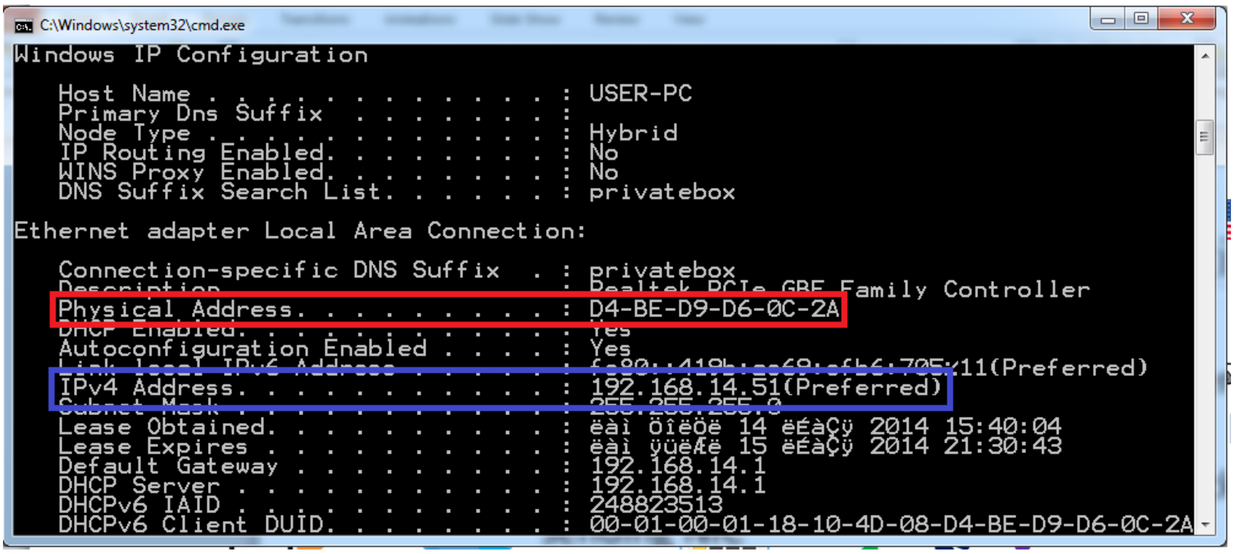
בפרק זה נתמקד בפרוטוקול Ethernet, בו משתמשים כרטיסי רשת מסוג Ethernet. כשאנו מדברים על כרטיס Ethernet, הכוונה היא לכרטיסי רשת המתחבר באופן קווי, עם כבל שנראה כך[60]:

מהי הכתובת שלי בשכבת הקו?

לצורך הסבר זה נניח שכרטיס הרשת שלכם הוא מסוג Ethernet. אם הוא לא, אנא עבדו על מחשב שיש לו כרטיס רשת כזה.

על מנת לתקשר זה עם זה, כרטיסי הרשת צריכים שיהיו להם מזהים – או כתובות, בהם הם יוכלו להשתמש. דבר זה נחוץ מכיוון שבחלק מהמקרים בשכבת הקו, שתי הישויות שמנסות לתקשר מחוברות באופן ישיר לא רק אחת לשנייה, אלא גם לישויות נוספות. לדוגמה, חשבו על רשת WiFi – כלל הישויות המחוברות לרשת יכולות לתקשר זו עם זו באופן ישיר, כלומר בלי לעבור באף תחנה אחרת בדרך. באם ישות מסוימת רוצה לפנות אל ישות אחרת, היא תצטרך לפנות אל הכתובת שלה. כתובות בשכבה השנייה נקראות **כתובות MAC** (באנגלית – **MAC Addresses**, קיצור של Media Access Control Addresses).

היכנסו שוב ל-Command Line, והקישו את הפקודה הבאה: **ipconfig /all**. שימו לב שהשתמשנו בפרמטר **/all**, שכן אחרת הפקודה ipconfig לא מציגה את כתובת השכבה השנייה.



[60] בפרק השכבה הפיזית / הרשת המשרדית, נלמד יותר על כבל זה.

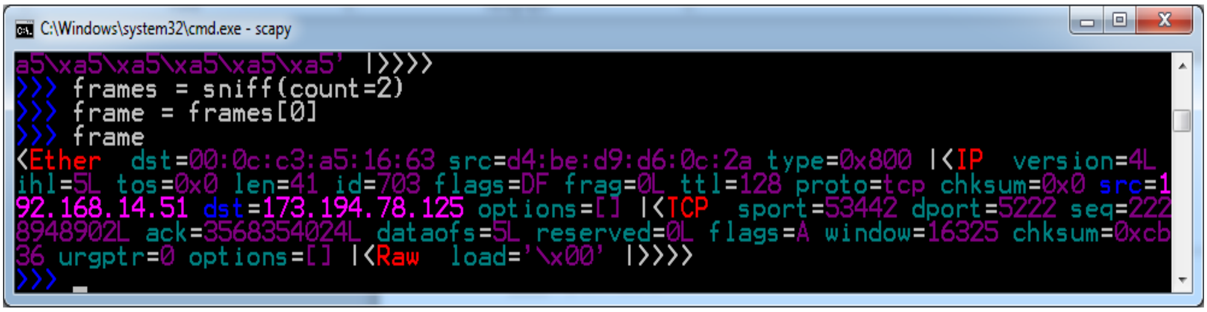
ניתן לראות שהמידע מוצג עבור הכרטיס עם כתובת ה-IP שמצאנו בפרק הקודם, 192.168.14.51, כפי שמסומן **בכחול** (המסגרת התחתונה). מעט למעלה יותר, יש שדה בשם Physical Address. זוהי כתובת ה-MAC, המסומנת **באדום** (המסגרת העליונה). שימו לב שכמו שהפקודה מציגה לנו, אכן מדובר **בכתובת פיזית** – כתובת זו "צרובה" על כרטיס הרשת עצמו, ולא אמורה להשתנות[61]. כמו כן, היא אמורה להיות ייחודית – כלומר, לא אמור להיות עוד כרטיס רשת בעולם בעל אותה הכתובת בדיוק. בהמשך הפרק נסביר כיצד מורכבת כתובת MAC.

מהי כתובת ה-MAC שלכם? מצאו אותה כעת.

תרגיל 8.1 מודרך – הסנפת כתובות ה-MAC ברשת המקומית

בתרגיל זה נבצע הסנפה, ובמהלכה נדפיס את כל כתובות ה-MAC של הישויות שפנו אל כתובת ה-MAC שלנו. לשם כך, נשתמש בכלי **Scapy**. נפתח את Scapy, ונבצע הסנפה פשוטה של שתי חבילות מידע. בשלב זה נכיר מונח נוסף בשם **מסגרת (Frame)**. בעוד חבילת מידע בשכבת הרשת נקראה חבילה או פקטה (Packet), בשכבת הקו רצף המידע שמועבר נקרא מסגרת. לאחר מכן, נסתכל על אחת המסגרות שהוסנפו:

```
>>> frames = sniff(count=2)
>>> frame = frames[0]
>>> frame
```



כפי שניתן לראות, בשכבת ה-Ethernet נמצאות כתובת היעד של המסגרת (destination address, שמופיעה בשם "dst"), וכתובת המקור של המסגרת (source address, שמופיעה בשם "src").

[61] בחלק מהמימושים של השכבה השנייה יש אפשרות להשתמש בכתובת שאינה צרובה על הכרטיס.

כפי שלמדנו בפרקים הקודמים, ניתן להשתמש בפרמטר **lfilter** של הפונקציה **sniff** של Scapy כדי לסנן מסגרות (או חבילות) המתאימות לתנאי שלנו. כדי לסנן מסגרות שפונות לכתובת ה-MAC של הכרטיס שלנו בלבד, עלינו לכתוב פונקציה שתחזיר True אם שדה כתובת היעד של המסגרת תואם את כתובת ה-MAC שלנו. לשם כך, נגדיר קודם את כתובת ה-MAC אותה מצאנו קודם לכן באמצעות ipconfig, בצורה הבאה:

```
>>> MY_MAC = 'd4:be:d9:d6:0c:2a'
```

שימו לב שהכתובת שלכם תהיה שונה מהכתובת שמופיעה בדוגמה. כמו כן, על אף ש-**ipconfig** הציג את הכתובת כאשר כל בית (byte) מופרד באמצעות התו מקף ('-'), אנו משתמשים בפורמט של Scapy בו כל בית מופרד באמצעות התו נקודותיים (':'). בנוסף, אנו כותבים באותיות קטנות ('a') כפי שעושה Scapy, ולא באותיות גדולות ('A') כפי שעושה **ipconfig**.

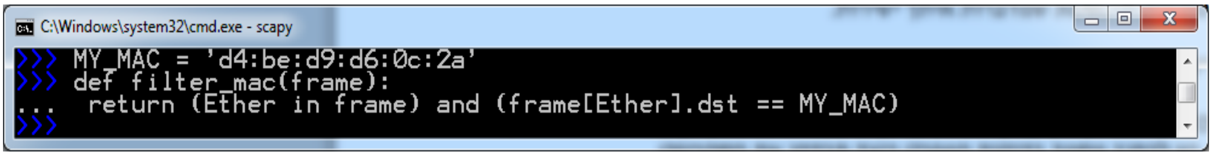
כעת נוכל לכתוב את הפונקציה שלנו:

```
>>> def filter_mac(frame):
        return (Ether in frame) and (frame[Ether].dst == MY_MAC)
```

בשלב הראשון, וידאנו שמדובר במסגרת Ethernet. לאחר מכן, הפונקציה מחזירה True במידה שכתובת היעד של המסגרת היא הכתובת של כרטיס הרשת שלנו. אם לא, היא מחזירה False.

לשם הבהרה, ניתן היה לרשום גם את הפונקציה בצורה הבאה:

```
>>> def filter_mac(frame):
...     if Ether not in frame:
...             return False
…       if frame[Ether].dst == MY_MAC:
…               return True
…       else:
…               return False
```

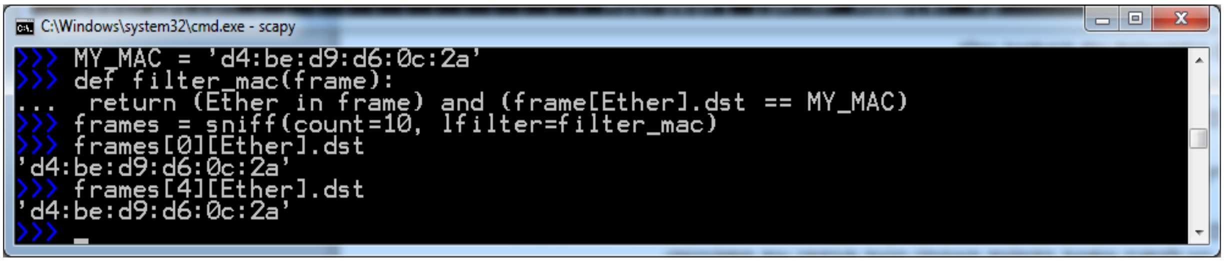


כעת נוכל להסניף רק מסגרות שיועדו אל כתובת ה-MAC שלנו באמצעות פונקציית הפילטר. נעשה זאת כך:

```
>>> frames = sniff(count=10, lfilter=filter_mac)
```

נוודא זאת על ידי התבוננות בכתובת היעד של שתי מסגרות:

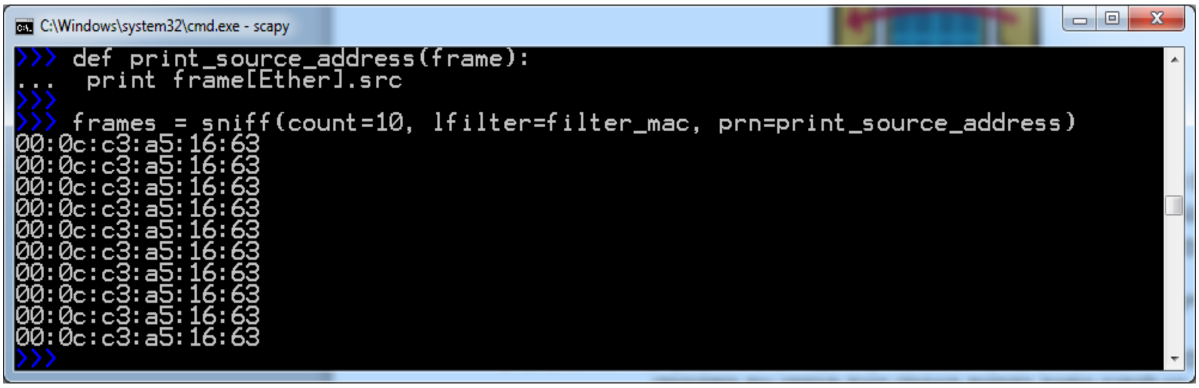
```
>>> frames[0][Ether].dst
'd4:be:d9:d6:0c:2a'
>>> frames[4][Ether].dst
'd4:be:d9:d6:0c:2a'
```



כעת ברצוננו להדפיס את כתובות ה-MAC של הישויות הפונות אלינו. לשם כך – נשתמש בפרמטר **prn** של הפונקציה **sniff**, אשר גם אותו הכרנו בפרקים קודמים בספר. נגדיר את הפונקציה שתטפל בכל מסגרת, כך שתודפס למסך כתובת המקור של המסגרת:

```
>>> def print_source_address(frame):
        print frame[Ether].src
```

נבצע את ההסנפה:



כפי שראיתם, בדוגמה זו כל הכתובות היו שייכות לאותה הישות (עליה נלמד בהמשך). במצבו הנוכחי, הסקריפט שכתבנו לא כל כך מועיל.

תרגיל 8.2 – מציאת כתובות MAC שפונות אל כרטיס הרשת שלי

שפרו את הסקריפט שכתבנו עד כה. גרמו לכך שהסקריפט "יזכור" האם הוא הדפיס כתובת מסוימת למסך, ואם כן – לא ידפיס אותה שוב. אפשרו לסקריפט לרוץ במשך חמש דקות (רמז – קראו על פרמטרים נוספים לפונקציה **sniff**).

על מנת לגרום לסקריפט להדפיס כתובות נוספות, נסו לפנות אליו מישויות נוספות ברשת שלכם, למשל ממחשב אחר שקיים ברשת. תוכלו להיעזר לשם כך בפקודה **ping** עליה למדנו ב[פרק שכבת הרשת/ איך Ping עובד?](#).

איך בנויה כתובת Ethernet?

עד כה בפרק הספקנו לראות מספר כתובות MAC של Ethernet. כפי שוודאי הבחנתם, כתובות Ethernet בנויות משישה בתים (bytes). ניתן לייצג את הכתובות בדרכים שונות. ניתן, כפי שעושה הפקודה **ipconfig**, לייצג את הכתובות באמצעות הפרדה עם התו מקף ('-') בין הבתים השונים, לדוגמה:

D4-BE-D9-D6-0C-2A

ניתן גם, כפי שעושה Scapy, לייצג את הכתובת באמצעות הפרדה עם התו נקודתיים (':') בין הבתים, למשל:

D4:BE:D9:D6:0C:2A

ניתן גם לבצע הפרדה רק בין צמדים של בתים, למשל:

D4BE:D9D6:0C2A

יש דרכים נוספות לייצג את הכתובות, אך בכל מקרה חשוב להבין שמדובר ברצף של שישה בתים.

עם זאת, לא לכל הבתים יש את אותה המשמעות. באופן כללי, כתובת Ethernet מחולקת לשני חלקים:

- **מזהה יצרן (Vendor ID)** – מזהה מי החברה שייצרה את כרטיס הרשת.
- **מזהה ישות (Host ID)** – מזהה של כרטיס הרשת הספציפי.

שלושת הבתים העליונים (הראשונים) שייכים למזהה היצרן, בעוד שלושת הבתים התחתונים שייכים למזהה הישות. כך למשל, בכתובת שהצגנו קודם לכן:

D4:BE:D9:D6:0C:2A

שלושת הבתים העליונים (הראשונים המסומנים ב**אדום**) הם מזהה היצרן, ושלושת הבתים התחתונים (האחרונים המסומנים ב**כחול**) שייכים לכרטיס הרשת הספציפי.

כך, אם נסתכל בכתובת הבאה:

D4:BE:D9:11:22:33

נוכל לדעת ששני כרטיסי הרשת יוצרו בידי אותו יצרן, שכן מזהה היצרן שלהם (3 הראשונים המסומנים ב**אדום**) – זהה. מזהי היצרנים ידועים, ולכן ניתן לדעת בקלות לאיזה יצרן שייכת כתובת מסוימת. לדוגמה, ניכנסו לאתר http://www.coffer.com/mac_find ונזין לתוכו את אחת משתי כתובות ה-MAC לעיל, שמתחילות במזהה היצרן D4:BE:D9. האתר יספר לנו שמדובר בכתובת השייכת ליצרנית Dell.

שימו לב שליצרניות שונות עשוי להיות יותר ממזהה יצרן אחד. כך למשל, גם מזהה היצרן E0:DB:55 שייך ל-Dell, גם המזהה A4:BA:DB, וגם רבים נוספים.

תרגיל 8.3 – מציאת היצרנית של כרטיס הרשת שלי

באמצעות האתר שהוצג לעיל, כמו גם פקודת **ipconfig**, מצאו מי היצרנית של כרטיס הרשת שלכם.

תרגיל 8.4 – מציאת יצרניות של כרטיסי רשת מתוך הסנפה

הורידו את קובץ ההסנפה Layer2_1.pcap מהכתובת:

http://data.cyber.org.il/networks/c07/Layer2_1.pcap

היעזרו בהסנפה בכדי למצוא את שתי כתובות ה-MAC שנמצאות בה, ולאחר מכן גלשו לאתר שהוצג לעיל ומצאו את היצרנים של כתובות ה-MAC המופיעות בקובץ. הוסיפו אותם לטבלה הבאה:

כתובת MAC	יצרן

כתובת Broadcast

הכתובת FF:FF:FF:FF:FF:FF הינה כתובת Ethernet מיוחדת. כתובת זו היא כתובת Broadcast – כלומר כל הישויות ברשת. שליחת מסגרת עם כתובת היעד FF:FF:FF:FF:FF:FF משמעותה שליחת המסגרת לכל הישויות שנמצאות איתנו ברשת.[62]

תרגיל 8.5 – כתובות בהסנפה

הורידו את קובץ ההסנפה Layer2_Broadcast.pcap מהכתובת:

http://data.cyber.org.il/networks/c07/Layer2_Broadcast.pcap

ענו על השאלות הבאות:

1. כמה מסגרות נשלחו אל כתובת Broadcast ברמת ה-Ethernet? מה המספר הסידורי של מסגרות אלו בהסנפה?
2. איזה מסנן תצוגה (display filter) יש לתת ל-Wireshark כדי לסנן רק את המסגרות שנשלחות לכתובת Broadcast ברמת ה-Ethernet?

[62] כתובת Broadcast שייכת למעשה לקבוצת כל הישויות ברשת. על כתובות Ethernet של קבוצות נלמד בנספח א' של פרק זה.

תרגיל 8.6 – כתובות Ethernet

בתרגיל זה תכתבו בפייתון סקריפט אשר מבקש מהמשתמש כתובת MAC ומדפיס עליה מידע.

1. קבלו מהמשתמש כתובת MAC. על הכתובת להיות בפורמט AA:BB:CC:DD:EE:FF (הפרדה של כל בית באמצעות התו נקודותיים). האותיות יכולות להיכתב כאותיות קטנות ('a') או גדולות ('A'). לאחר קבלת הכתובת, הדפיסו "Valid" אם הכתובת הינה כתובת Ethernet תקינה, ו-"Invalid" אם הכתובת אינה תקינה.

   בחנו את עצמכם עם הכנסת הקלטים הבאים:

   - 11:22:33:44:55:66 (כתובת תקינה)
   - FF:FF:FF:FF:FF:FF (כתובת תקינה)
   - AB:12:cd:34:31:21 (כתובת תקינה)
   - 11:22:33:44:55:66:77 (כתובת שאינה תקינה)
   - 11-22-33-44-55-66 (כתובת שאינה תקינה עבור סקריפט זה)
   - 11:22:33:44:55 (כתובת שאינה תקינה)
   - H:22:33:44:55:661 (כתובת שאינה תקינה)

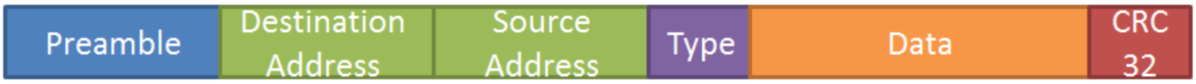
   רמז: היעזרו במתודה **split**.

   לחלופין, ניתן לקרוא על regular expressions (https://docs.python.org/2/library/re.html) ולהשתמש בהם.

2. אם הכתובת תקינה, הדפיסו את מזהה היצרן. אין צורך להדפיס את שם היצרן, אלא רק את המזהה (למשל: 11:22:33).

מבנה מסגרת Ethernet

מסגרת Ethernet נראית כך:



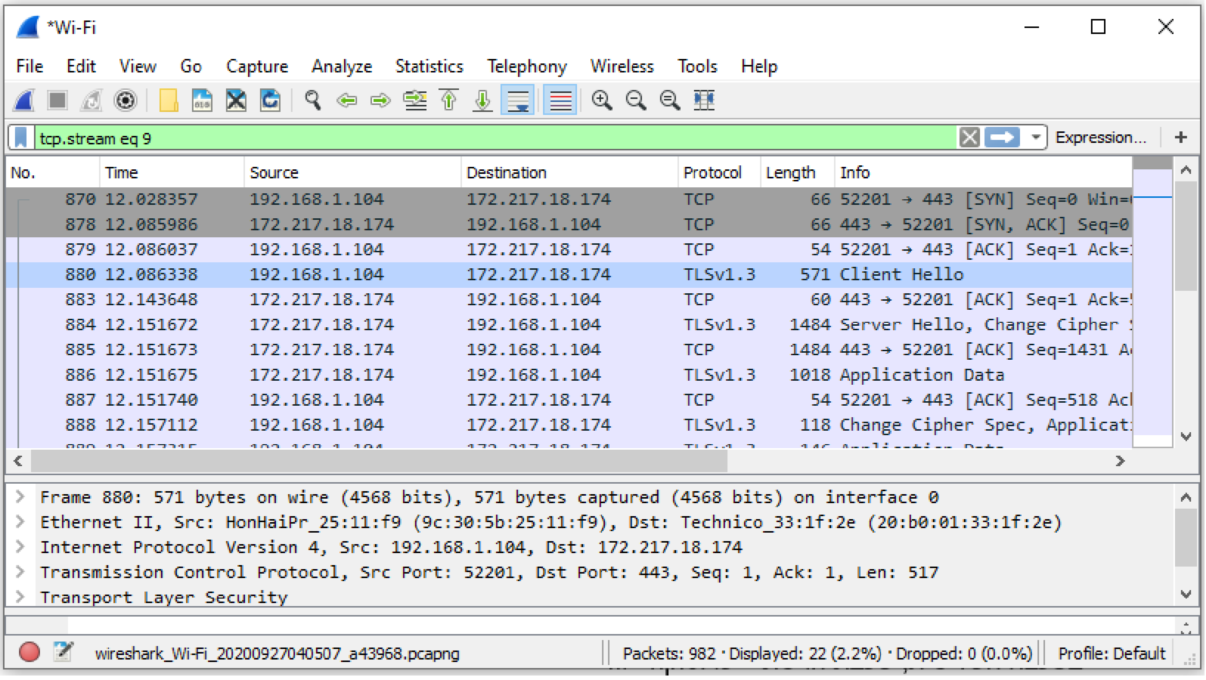
- **Preamble** – רצף קבוע מראש של שמונה בתים (bytes) שנועדו לסנכרן את שני הצדדים על כך שמתחילה מסגרת חדשה. שימו לב – לא רואים שדה זה ב-Wireshark.
- **Destination Address** – כתובת היעד של המסגרת. שדה זה מכיל שישה בתים (bytes), בפורמט עליו למדנו קודם לכן.
- **Source Address** – כתובת המקור של המסגרת. שדה זה מכיל שישה בתים (bytes), בפורמט עליו למדנו קודם לכן.

- **Type** – סוג המסגרת. שדה זה מכיל שני בתים (bytes) שמעידים על סוג ה-Data. כך למשל, מסגרת שבה שדה זה מכיל את הערך 0x800 הינה מסגרת מסוג IP. כך כרטיס הרשת יודע להפנות את המידע של המסגרת (במקרה הזה – חבילת IP) אל הגורם שיודע לטפל במידע הזה.[63]
- **Data** – המידע עצמו של החבילה. על המידע להיות באורך של 64 בתים (bytes) לפחות. אם המידע קצר יותר, נוסף רצף של אפסים (00) בסוף המידע.
- **CRC32** – מנגנון Checksum לגילוי שגיאות. על משמעות של Checksum למדנו במסגרת [פרק שכבת התעבורה/ מה זה Checksum ?](). בפרוטוקול Ethernet, אורך ה-Checksum הוא 32 ביטים (bits), שהם ארבעה בתים (bytes). גם שדה זה לא נראה ב-Wireshark, שכן הווידוא שלו מתרחש אצל כרטיס הרשת עוד לפני ש-Wireshark "רואה" את המסגרת.

תרגיל 8.7 מודרך – התבוננות בבקשת HTTPS

פתחו את Wireshark והתחילו הסנפה עם מסנן התצוגה "tcp.port == 443". במקביל, פתחו את הדפדפן שלכם וגלשו אל [www.google.com]().

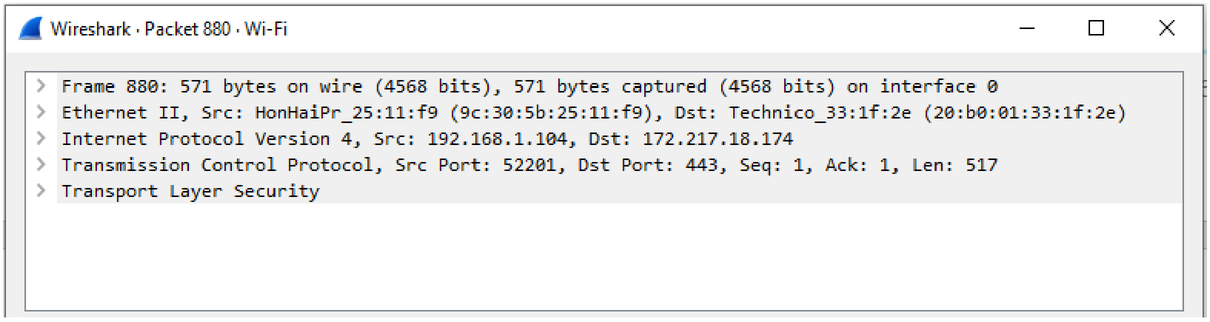
מצאו את החבילות הרלוונטיות בהסנפה. באופן ספציפי, מצאו את חבילת ה-Client Hello, שמגיעה מיד לאחר ה-Three Way Handshake:



הקליקו על החבילה כדי לפתוח אותה, ושימו לב למודל השכבות:

---

[63] במקרים מסוימים, שדה זה גם יכול להעיד על האורך הכולל של המסגרת. עם זאת, בספר זה נתעלם מאפשרות זו. אתם מוזמנים לקרוא עליה באינטרנט.

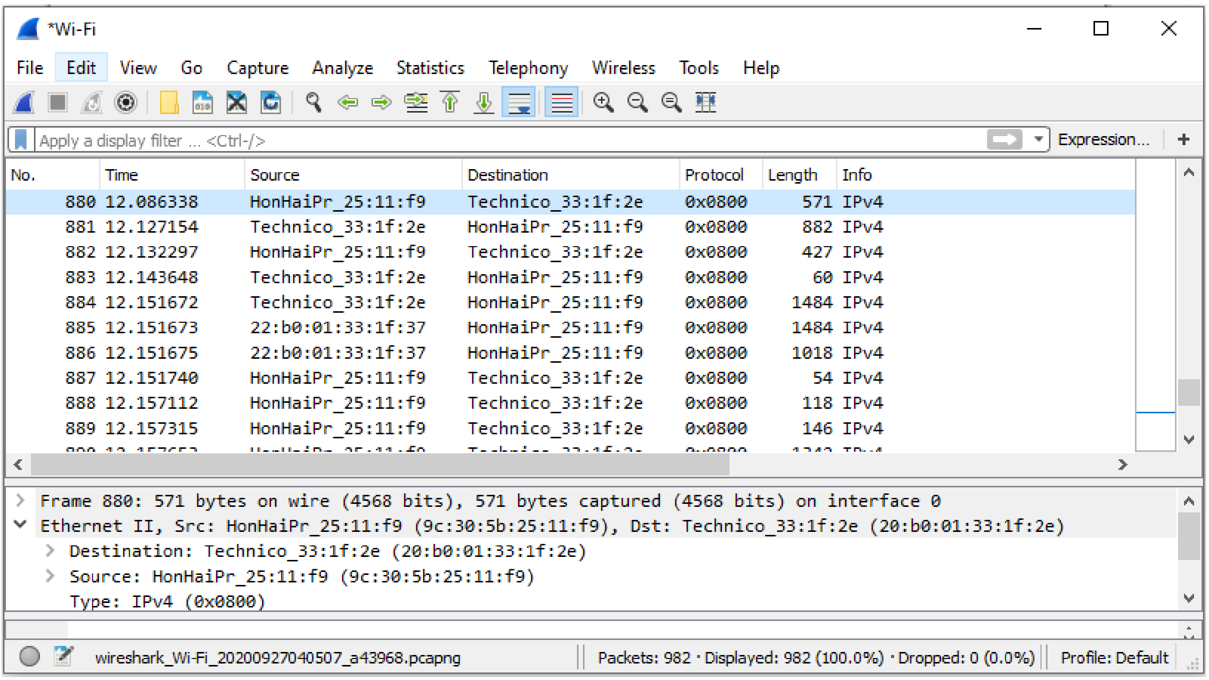


שימו לב למודל השכבות:

- בשכבה השנייה, שכבת הקו – פרוטוקול Ethernet.
- בשכבה השלישית, שכבת הרשת – פרוטוקול IP.
- בשכבה הרביעית, שכבת התעבורה – פרוטוקול TCP.
- בשכבה החמישית, שכבת האפליקציה – פרוטוקול TLS מצפין את תעבורת פרוטוקול ה-HTTP וכך מתקבל HTTPS.

היזכרו במושג ה**כימוס (Encapsulation)** אותו הכרנו בפרק Wireshark ומודל חמש השכבות/ כיצד בנויה פקטה?. מסגרת ה-Ethernet שלנו בשכבת הקו מכילה "בתוכה" את שכבת הרשת שעושה שימוש בפרוטוקול IP. שכבת הרשת מכילה "בתוכה" את שכבת התעבורה שעושה שימוש בפרוטוקול TCP. שכבת התעבורה מכילה "בתוכה" את שכבת האפליקציה שעושה שימוש בפרוטוקול TLS.

רישמו לעצמכם בצד את מספר החבילה של ה-Client Hello (במקרה זה 880). היכנסו ב-Wireshark לתפריט Analyze בסרגל הכלים, ובחרו באפשרות Enabled Protocols.
כעת, הורידו את הסימון מהפרוטוקול IPv4. לחצו על OK. הסירו את מסנן התצוגה "tcp.port", שכן Wireshark כבר אינו מכיר אותו. כעת Wireshark יציג בפנינו רק את השדות של שכבת ה-Ethernet, וכל השאר ייראה כ-Data, ממש כשם שכרטיס הרשת שלנו רואה את המסגרת. באמצעות מספר החבילה אותו רשמתם לפני כן, תוכלו לאתר אותה בקלות:



הסתכלו על השדות השונים ב-Header של Ethernet:

- **כתובת היעד** – האם זוהי הכתובת של Google? התשובה היא **לא!** היזכרו בפרק שכבת הרשת, בו דיברנו על כך שבדרך כלל ישנם רכיבים רבים שמקשרים בין ישויות קצה ברשת. בהנחה שאינכם מחוברים באופן ישיר אל Google באמצעות כבל (או דרך אחרת), אתם עוברים בדרך בישות נוספת. הישות הקרובה ביותר אליכם היא **הנתב שלכם**, וכתובת ה-MAC הזו היא הכתובת שלו. כמו כן, שימו לב ש-Wireshark יודע להגיד לנו שהיצרנית של הנתב הינה Bewan.
שימו לב: לנתב יש יותר מכתובת MAC אחת, שכן יש לו יותר מכרטיס רשת אחד. כתובת ה-MAC שמוצגת בהסנפה היא הכתובת של כרטיס הרשת של הנתב המחובר אל הרשת הביתית שלכם, ולא של כרטיס הרשת של הנתב שמחובר אל האינטרנט.
- **כתובת המקור** – כתובת זו צפויה להיות הכתובת של כרטיס הרשת שלכם. עבור תעבורה שיוצאת מהמחשב שלכם, כתובת המקור תהיה זהה לכתובת שמצאתם באמצעות הפקודה **ipconfig /all**. כמו כן, Wireshark יודע להגיד לנו שהיצרנית של כרטיס הרשת, במקרה זה, הינה HonHaiPr, הידועה גם בשם FoxConn, אחת מיצרניות ציוד המחשוב הגדולות בעולם.
- **סוג** – הערך 0x800 מצביע על כך שהמסגרת היא מסוג IP. כך כרטיס הרשת יידע להפנות את כל מה שנמצא בשדה ה-Data אל הישות שמטפלת בחבילות IP (במקרה שלנו – מערכת ההפעלה).
- שדה ה-Data כאן מכיל את כל השכבות שנמצאות "מעל" ל-Ethernet, החל משכבת ה-IP, דרך שכבת ה-TCP ועד שכבת ה-HTTP.
- שימו לב ששדות ה-Preamble וה-Checksum אינם מופיעים בהסנפה, כפי שציינו קודם לכן.

שימו לב: כאן רואים באופן יפה את ההבדל בין השכבה השנייה לשכבה השלישית. בחבילת שמוצגת לעיל, כתובת המקור בשכבה השלישית, שכבת הרשת, היא כתובת ה-IP של המחשב

שלנו, וכתובת היעד היא כתובת ה-IP של השרת של Google. שכבת הרשת מציגה את כל המסלול – מהיכן החבילה נשלחה ומה היעד הסופי שלה. עם זאת, בשכבה השנייה, שכבת הקו, כתובת המקור היא כתובת כרטיס הרשת של המחשב שלנו וכתובת היעד היא הכתובת של הנתב הקרוב, שכן שכבת הקו מדברת על קשר בין ישויות שמחוברות באופן ישיר בלבד. לכן, בעוד בשכבת הרשת נראה את הכתובת ההתחלתית והסופית של החבילה, בשכבת הקו אנו נראה כל שלב בדרך.

כפי שלמדנו בפרק שכבת הרשת/ ניתוב, בשלב הבא החבילה תועבר מהנתב הקרוב למחשב שלנו אל הנתב הבא בדרך. בשלב זה, הנתב הקרוב למחשב שלנו יצור את המסגרת בשכבת ה-Ethernet, כך שכתובת המקור של המסגרת תהיה הכתובת של כרטיס הרשת שלו שמחובר לאינטרנט, וכתובת היעד תהיה הכתובת של הנתב הבא. כך, שכבת הקו מתארת נכונה את ה-Hop הנוכחי: מעבר בין הנתב שקרוב אלינו אל הנתב הבא אחריו. עם זאת, בשכבת הרשת עדיין יישמרו הנתונים על נקודות הקצה של החבילה – המועברת מהמחשב שלנו ואל Google.

תרגיל 8.8 – התבוננות בתשובת HTTPS

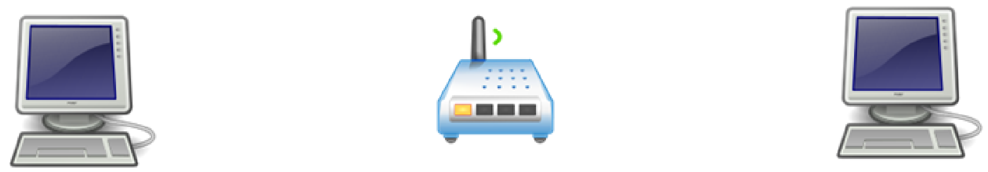
כעת הסתכלו בתשובת ה-HTTPS שחזרה בתגובה ל-Client Hello. ענו על השאלות הבאות:

1. למי שייכת כתובת היעד של המסגרת? כיצד תוכלו לוודא זאת?
2. למי שייכת כתובת המקור של המסגרת? כיצד תוכלו לוודא זאת?
3. מהו סוג המסגרת?

בסיום התרגיל, אל תשכחו לחזור לתפריט Analyze > Enabled Protocols ב-Wireshark ולהחזיר את הסימון לפרוטוקול IPv4.

תרגיל 8.9 – כיצד תיראה החבילה שלי?

עיינו בתרשים הרשת הבא:



מחשב א'	נתב	מחשב ב'
כתובת MAC: 64:20:0c:34:12:34 כתובת IP: 1.1.1.1	כתובת MAC ראשונה: 22:22:22:22:22:22 כתובת IP ראשונה: 1.1.1.2 כתובת MAC שנייה: 33:33:33:33:33:33 כתובת IP שנייה: 3.3.3.2	כתובת MAC: 11:22:33:34:44:53 כתובת IP: 3.3.3.3

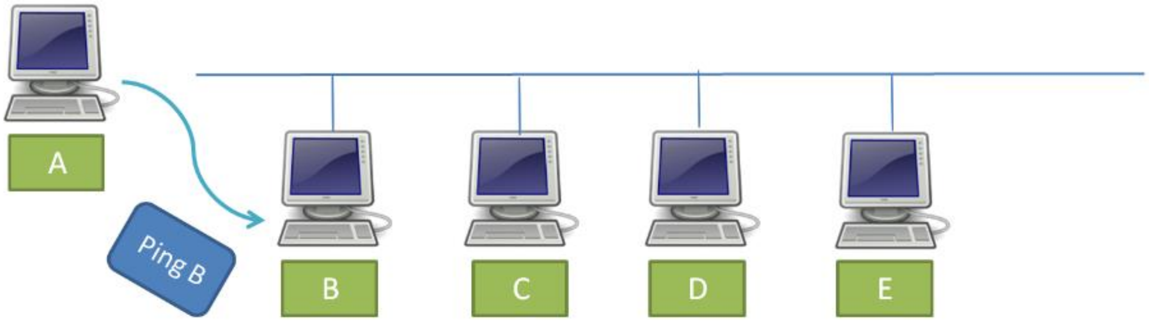
מחשב א' רוצה לשלוח פקטה למחשב ב'. מחשב א' מחובר למחשב ב' דרך הנתב.

השלימו את השדות של הפקטה אותה ישלח מחשב א':

כתובת IP מקור	1.1.1.1
כתובת MAC מקור	c:34:12:3464:20:0
כתובת IP יעד	
כתובת MAC יעד	

פרוטוקול Address Resolution Protocol – ARP

עד כה תיארנו את פרוטוקול Ethernet וכיצד הוא עובד. אך עדיין, משהו חסר. הביטו בשרטוט הרשת הבא:



כלל המחשבים כאן נמצאים על תווך משותף. כלומר, מסגרת הנשלחת לכתובת Broadcast תגיע אל כולם, ואין צורך בישות נוספת (כגון נתב) כדי להעביר הודעות ממחשב אחד למחשב אחר. במקרה לפנינו, המחשב שנקרא A רוצה לשלוח הודעת ping (כלומר Echo-Request, כמו שלמדנו בפרק שכבת הרשת) אל המחשב B. המחשב A יודע את כתובת ה-IP של מחשב B, למשל, באמצעות שימוש בפרוטוקול DNS. אך דבר זה אינו מספיק – על מחשב A לדעת גם את כתובת ה-MAC של כרטיס הרשת של מחשב B!

מדוע הדבר כך? מדוע לא מספיקה כתובת ה-IP?

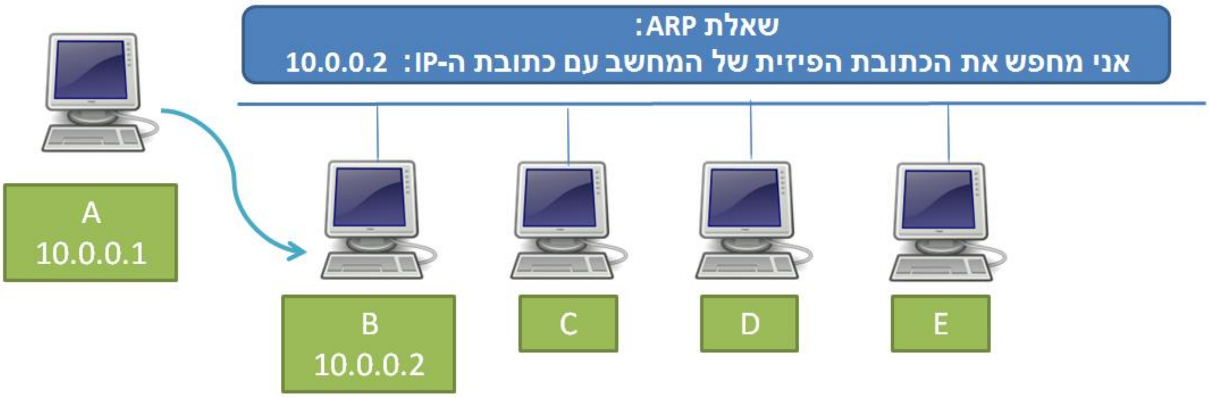
ראשית, היזכרו במודל השכבות. חבילת ה-Ping שתשלח ממחשב A צפויה להיות בנויה משכבת ה-Ethernet בשכבה השנייה, מעליה שכבת IP ולבסוף שכבת ICMP. על מנת לבנות את מסגרת ה-Ethernet, על המחשב A לדעת מה כתובת היעד של המסגרת, כלומר מה הכתובת של כרטיס הרשת של B.

שנית, על כרטיס הרשת של המחשב B להבין שהמסגרת מיועדת אליו. את זאת הוא עושה באמצעות הסתכלות בשדה כתובת היעד של המסגרת. **כרטיס הרשת אינו מבין כתובות IP**, הוא כרטיס מסוג Ethernet ומכיר כתובת Ethernet בלבד. על כן, הכרטיס צריך לראות כתובת MAC.

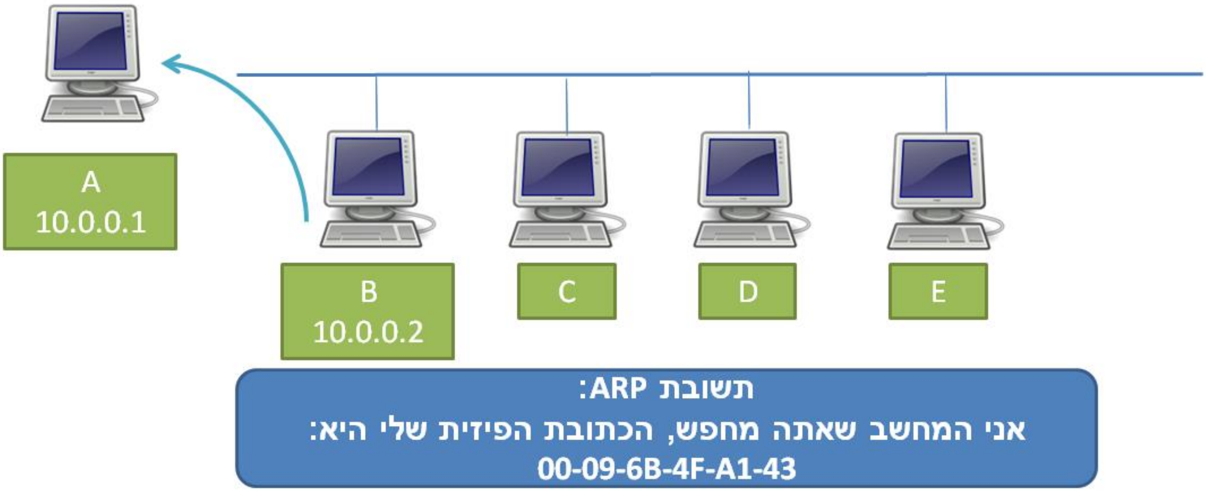
אי לכך, על המחשב A להבין מה כתובת ה-MAC של המחשב B. לשם כך נועד פרוטוקול **ARP** (או בשמו המלא – **Address Resolution Protocol**). פרוטוקול זה ממפה בין כתובות לוגיות של שכבת הרשת לכתובות פיזיות של שכבת הקו.

במקרה שלנו, המחשב A מעוניין למפות בין כתובת ה-IP הידועה לו של מחשב B, לבין כתובת ה-MAC של כרטיס הרשת של המחשב B. בשלב הראשון, המחשב A ישלח שאלה לכל הרשת (כלומר – לכתובת Broadcast) שמשמעותה: למי יש את הכתובת הפיזית של המחשב עם כתובת ה-IP של B?

נאמר שכתובת ה-IP של מחשב B הינה 10.0.0.2, והכתובת של המחשב A הינה 10.0.0.1:



בשלב זה, כלל המחשבים מקבלים את ההודעה. מי שצפוי לענות להודעה זו הוא המחשב B, אשר יודע את הכתובת הפיזית שלו.



כך מחשב A מגלה את כתובת ה-MAC של כרטיס הרשת של מחשב B. כעת, יש ברשותו את כל המידע שהוא צריך כדי לשלוח את חבילת ה-Ping:

- כתובת ה-IP שלו עצמו (מחשב A) – שכן הוא יודע מה הכתובת שלו, למשל באמצעות DHCP.

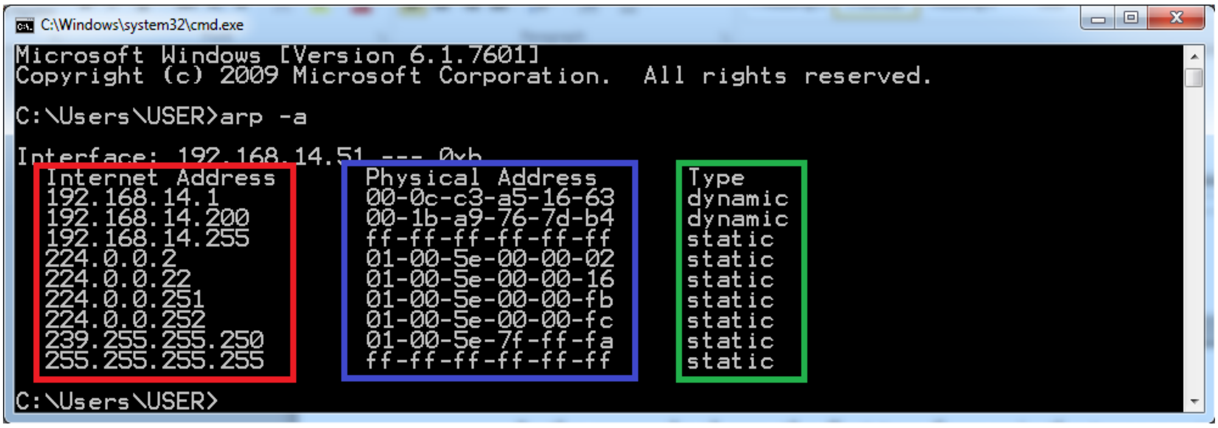
- כתובת ה-MAC של עצמו (מחשב A) – שכן הוא יודע מה הכתובת שלו, שהרי היא צרובה על הכרטיס.
- כתובת ה-IP של מחשב B – שהוא גילה, למשל, באמצעות DNS.
- כתובת ה-MAC של מחשב B – שהוא גילה באמצעות פרוטוקול ARP.

מטמון (Cache) של ARP

גם עבור פרוטוקול ARP מערכת ההפעלה שלנו שומרת מטמון (Cache), במטרה לא לשאול שוב ושוב את אותה שאלת ARP. כפי שכבר הבנו, המחשב שלנו זקוק לתקשר עם הנתב הקרוב אליו באופן תדיר. על מנת לאפשר את התקשורת עם הנתב, עליו לדעת מה כתובת ה-MAC שלו. לא הגיוני שלפני כל חבילה שהמחשב יעביר לנתב הוא יבצע שאילתת ARP ויחכה לתשובה – תהליך זה יקח זמן רב מדי. לכן, סביר שכתובת ה-MAC של הנתב תישמר במטמון.

על מנת להביט במטמון, היכנסו לשורת הפקודה והריצו את הפקודה:

arp -a



הפקודה מציגה לנו טבלה עם שלוש עמודות:

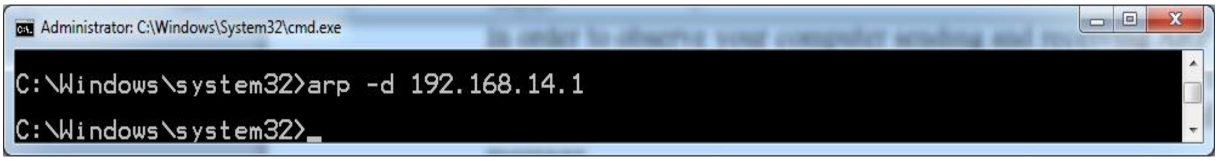
- **באדום (שמאלית) –** כתובת IP.
- **בכחול (אמצעית) –** כתובת MAC המשויכת לאותה כתובת IP.
- **בירוק (ימנית) –** סוג הרשומה – האם היא דינאמית (כלומר הושגה באמצעות פרוטוקול ARP) או סטאטית (הוכנסה באופן ידני ולא משתנה).

על מנת שהתרגיל הבא יעבוד, עליכם לרוקן את המטמון. לשם כך, הריצו את הפקודה:

arp -d <ip_address>

לדוגמה:

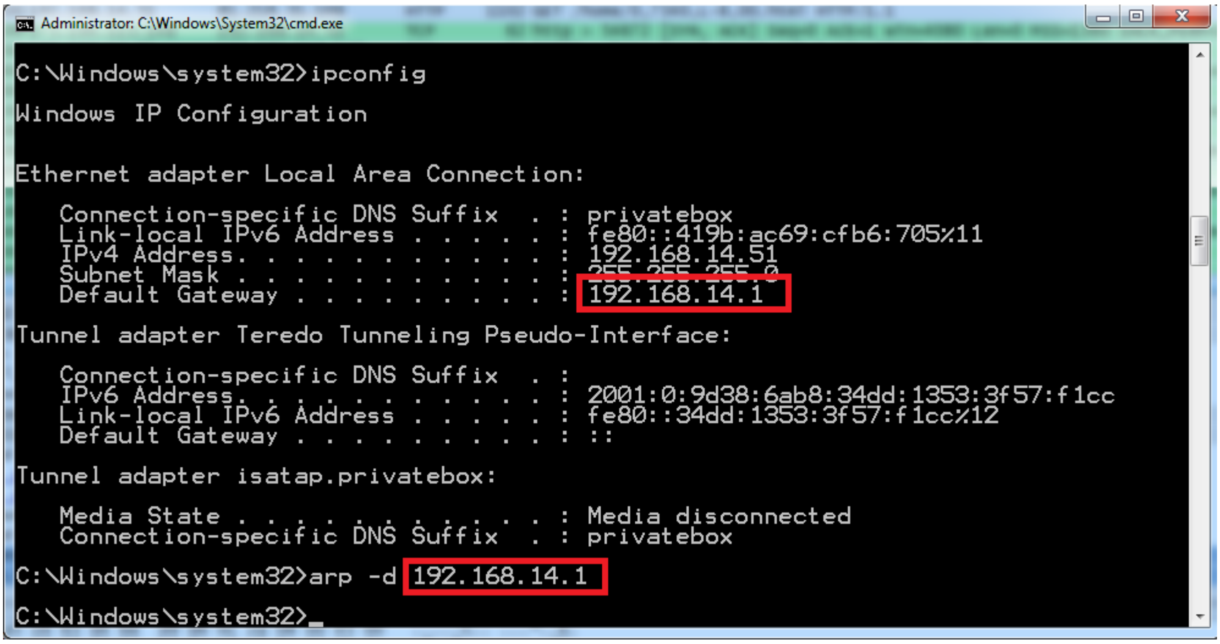
arp -d 192.168.4.1



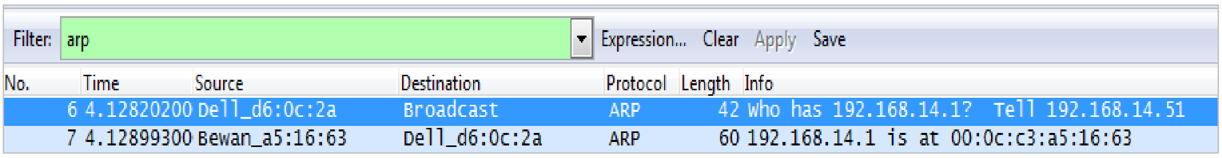
שימו לב שאת הפקודה יש להריץ בהרשאות גבוהות. לכן, הריצו את שורת הפקודה בהרשאות Administrator.

תרגיל 8.10 מודרך – התבוננות בשאילתת ARP

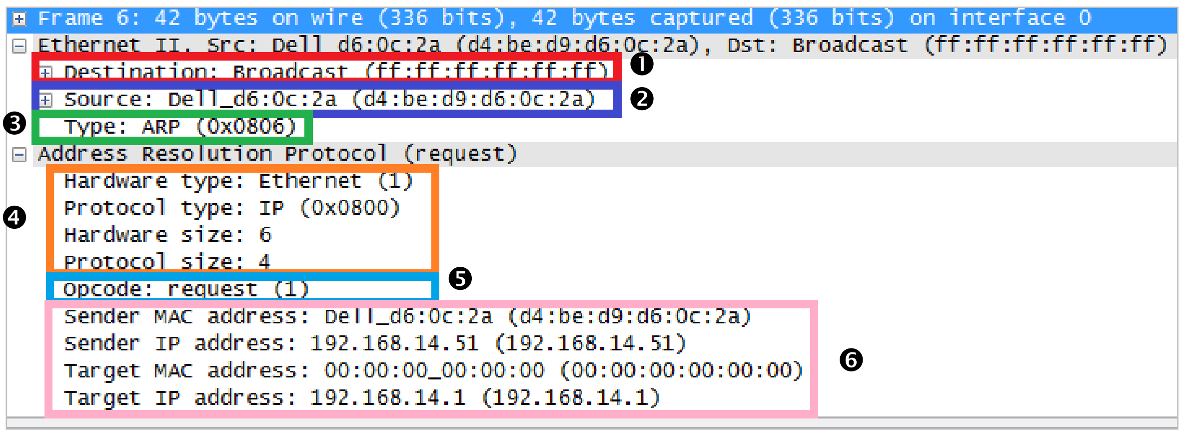
פתחו את Wireshark והריצו הסנפה. כמו כן, מחקו מה-ARP Cache שלכם את הרשומה שקשורה לנתב שלכם (ה-Default Gateway). להזכירכם, כדי לגלות את כתובת ה-IP של הנתב, ניתן להשתמש בפקודה **ipconfig**:



גלשו אל https://www.themarker.com. כפי שכבר הבנו, על מנת לתקשר עם TheMarker, המחשב יצטרך לגשת אל הנתב. תוכלו להשתמש במסנן התצוגה "arp" בכדי לסנן את החבילות הרלוונטיות:



כעת נביט במסגרת השאילתה:



נתחיל מלהביט בשדות של שכבת ה-Ethernet:

- באדום (1) – כתובת היעד של המסגרת. הכתובת היא כתובת Broadcast, כלומר ff:ff:ff:ff:ff:ff.
- בכחול (2) – כתובת המקור של המסגרת. זוהי הכתובת של כרטיס הרשת שלנו.
- בירוק (3) – סוג המסגרת. מדובר במסגרת מסוג ARP.

נביט בשדות של שכבת ה-ARP:

- בכתום (4) – נתונים המתארים כי המיפוי הוא מכתובת IP לכתובת Ethernet. שדות אלו נחוצים כיוון ש-ARP יכול למפות גם מכתובות לוגיות אחרות לכתובות פיזיות אחרות.
- בתכלת (5) – קוד חבילת ה-ARP. הקוד הינו 1, והוא מציין שאילתה (Request).
- בוורוד (6) – השדות שקשורים לכתובות:
  - כתובת ה-MAC של הישות השולחת, כלומר של המחשב שלנו ששלח את השאילתה.
  - כתובת ה-IP של הישות השולחת, כלומר של המחשב שלנו ששלח את השאילתה.
  - כתובת ה-MAC המבוקשת. במקרה זה, הכתובת היא אפסים מכיוון שעליה אנו שואלים – איננו יודעים מהי כתובת ה-MAC של היעד (הנתב שלנו).
  - כתובת ה-IP של היעד, כלומר כתובת ה-IP של הנתב.

תרגיל 8.11 – התבוננות בתשובת ARP

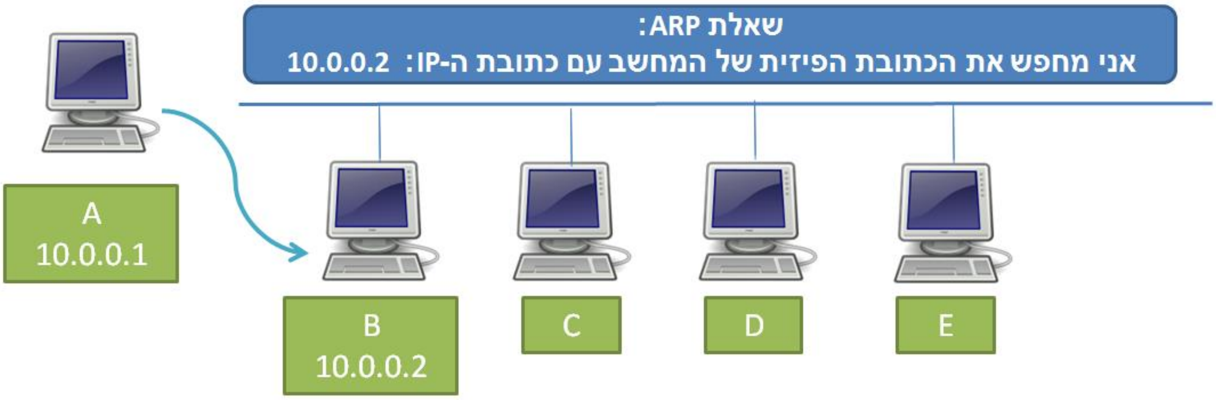
כעת, מצאו את מסגרת התשובה. ענו על השאלות הבאות:

1. בשכבת ה-Ethernet, מהי כתובת המקור של המסגרת? מי שלח אותה?
2. בשכבת ה-Ethernet, מהי כתובת היעד של המסגרת? האם היא נשלחת אל כולם (Broadcast) או רק לישות מסוימת?
3. מהו הערך של שדה ה-opcode בתשובה?
4. איפה במסגרת ה-ARP מופיעה התשובה לשאילתה שנשלחה, כלומר הכתובת הפיזית של הנתב?

5. התבוננו בתשובת ה-ARP. מה היא הכתובת הפיזית של הנתב שלכם?
6. הציגו את ה-ARP Cache של מחשבכם. האם הכתובת הפיזית שמצאתם במסגרת ה-ARP הינה הכתובת הפיזית שמופיעה ב-Cache?

למי נשלחת שאלת ה-ARP?

היות שמסגרת ARP הינה בשכבה שנייה בלבד, ולא בשכבה שלישית, היא לא מועברת הלאה על ידי נתבים. מכאן שבדוגמה הבאה:



אם מחשב A מעוניין לשלוח שאלת ARP למחשב B, עליהם להיות בחיבור ישיר זה עם זה כדי שהשאלה תגיע ליעדה. כדי להבין אם מחשב B מחובר אליו בחיבור ישיר, מחשב A בודק האם מחשב B נמצא איתו **באותו ה-Subnet**. דבר זה אמנם מוזר, שכן Subnet הוא מונח של שכבת הרשת, וכמו שלמדנו – כתובת IP היא כתובת לוגית שלא מלמדת בוודאות על מיקום פיזי של כרטיס הרשת. אף על פי כן, זו הבדיקה המתבצעת – מחשב A בודק אם כתובת ה-IP של מחשב B נמצאת איתו באותו ה-Subnet. במקרה זה, הכתובת של מחשב A הינה: 10.0.0.1, הכתובת של מחשב B הינה: 10.0.0.2, ונניח כי מסכת הרשת היא: 255.0.0.0. כפי שלמדנו בפרק שכבת הרשת/ מהו מזהה הרשת שלי? ומהו מזהה הישות?, משמעותה של מסכת רשת זו היא שהבית הראשון שייך למזהה הרשת, ועל-כן המחשבים נמצאים באותו ה-Subnet. ולכן במקרה זה, המחשב A אכן שולח שאלת ARP עבור כתובת ה-IP של B.

אך מה היה קורה לו B לא היה נמצא באותו ה-Subnet כמו מחשב A? מה היה קורה לו מחשב A היה מעוניין לשלוח הודעה אל Google, שכתובת ה-IP שלו הינה, לדוגמה, 3.3.3.3? במקרה זה, מכיוון שהכתובת 3.3.3.3 אינה ב-Subnet של מחשב A, החבילה צריכה להישלח אל ה-**Default Gateway** של מחשב A, אותו נתב המיועד לטפל בחבילות היוצאות מהרשת, כפי שלמדנו בפרק שכבת הרשת/ מהי טבלת הניתוב שלי?. לכן, במידה שרשומת ה-ARP הרלוונטית לא קיימת במטמון של מחשב A, תישלח הודעת ARP עבור כתובת ה-IP של ה-Default Gateway.

לסיכום, כאשר מחשב מעוניין לשלוח חבילה אל כתובת IP כלשהי, מתבצעת בדיקה האם כתובת ה-IP של היעד הינה באותו ה-Subnet של המחשב השולח:

- אם כתובת היעד נמצאת ב-Subnet של המחשב השולח, הרי שניתן לשלוח שאלת ARP עבור כתובת ה-IP של היעד , ואז לשלוח את החבילה ישירות אל כתובת ה-MAC המוחזרת בתשובה.
- אם כתובת היעד לא נמצאת ב-Subnet של המחשב השולח, הרי שלא ניתן לשלוח שאלת ARP עבור כתובת ה-IP של היעד. במקרה זה, נשלחת שאלת ARP לגילוי כתובת ה-MAC של ה- Default Gateway, ואז החבילה מועברת אליו להמשך טיפול.

כמובן שבשני המקרים לא נשלחת שאלת ARP אם המידע הרלוונטי נמצא כבר במטמון.

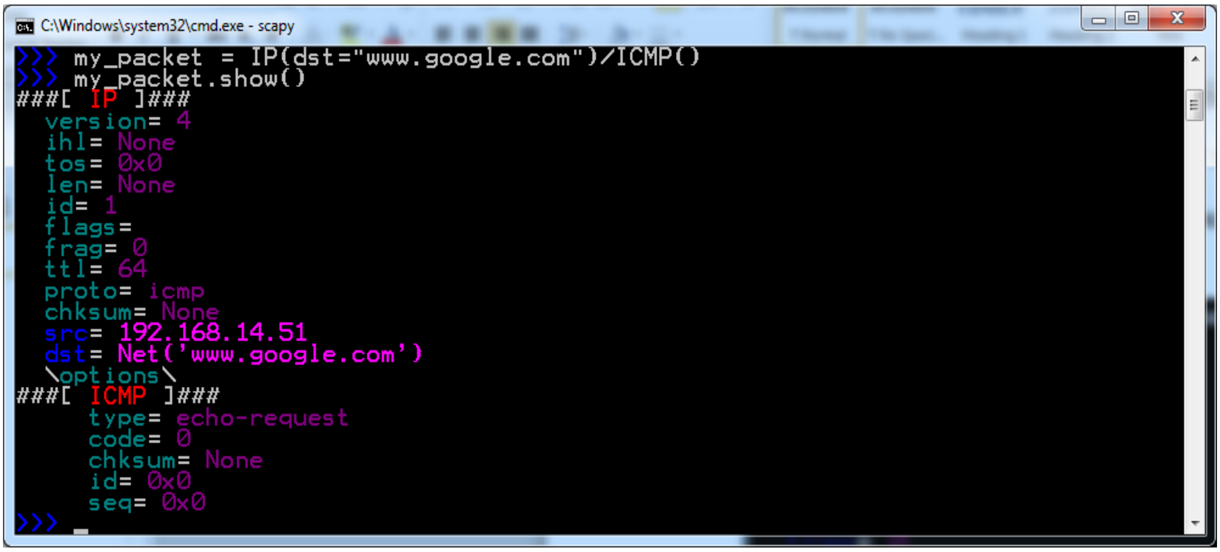
שליחת מסגרות בשכבה שנייה באמצעות Scapy

כשלמדנו על Scapy, למדנו לשלוח חבילות בשכבה השלישית באמצעות הפונקציה **send**. למדנו, למשל, ליצור פקטה שמתחילה בשכבה זו, כגון פקטת ICMP Ping:

>>> my_packet = IP(dst = "www.google.com") / ICMP()

שימו לב ש-Scapy יוצר באופן ברירת מחדל את שכבת ה-ICMP כבקשת Ping (כלומר Echo Request).

ניתן לוודא זאת:



כעת נוכל לשלוח את הפקטה:

```
>>> send(my_packet)
.
Sent 1 packets.
```

עכשיו, כשאנו יודעים יותר על רמת הקו, העובדה ש-Scapy הצליח לשלוח את החבילה אמורה לגרום לנו להרמת גבה. כיצד Scapy עשה זאת? איך הוא הצליח לשלוח פקטה בשכבה שלישית בלבד?

אז כידוע לכם – אין באמת קסמים ברשתות, ו-Scapy הוא לא קוסם. Scapy הבין לבד שאני מעוניין שהוא ישלח את הפקטה מעל Ethernet, ובנה שכבה זו בעצמו בכדי לשלוח את הפקטה (כשכתובת המקור היא הכתובת של כרטיס הרשת שלנו, כתובת היעד היא של הנתב, והסוג הוא IP).

אך מה אם היינו רוצים לשלוח את הפקטה דווקא מכרטיס רשת מסוים? מה אם היינו רוצים שהיא תשלח מעל WiFi ולא מעל Ethernet? או מה אם היינו רוצים לשנות את השדות של שכבת ה-Ethernet באופן ספציפי? לשם כך, Scapy מציע לנו לשלוח את הפקטה בשכבה שנייה, זאת באמצעות הפונקציה **sendp**. בואו ננסה:

```
>>> my_layer2_packet = Ether()/IP(dst="www.google.com")/ICMP()
>>> sendp(my_layer2_packet)
.
Sent 1 packets.
```

שימו לב לא להתבלבל בין הפונקציות **send** ו-**sendp**. שליחת פקטה שלא מכילה שכבה שנייה באמצעות **sendp** תגרום, מן הסתם, לשליחה של פקטה לא תקינה. כך גם שימוש ב-**send** לשליחת פקטה שמכילה כבר שכבת Ethernet, או כל פרוטוקול שכבה שנייה אחר.

תרגיל 8.12 – שליחה עם שליטה בשדות ה-Ethernet

השתמשו בכתובת ה-IP של ה-Default Gateway שלכם שמצאתם קודם לכן. השתמשו בפקודה **ping** כדי לשלוח אליו בקשת Echo Request. הסתכלו ב-Wireshark על השדות של הבקשה. באופן צפוי, בשכבת ה-Ethernet, כתובת היעד צפויה להיות כתובת ה-MAC של הנתב.

כעת, נסו לעשות דבר אחר. שלחו בקשת ICMP Echo Request אל ה-IP של הנתב, אך בשכבת ה-Ethernet, השתמשו בכתובת היעד FF:FF:FF:FF:FF:FF.
האם הנתב ענה לשאילתה שלכם? מדוע?

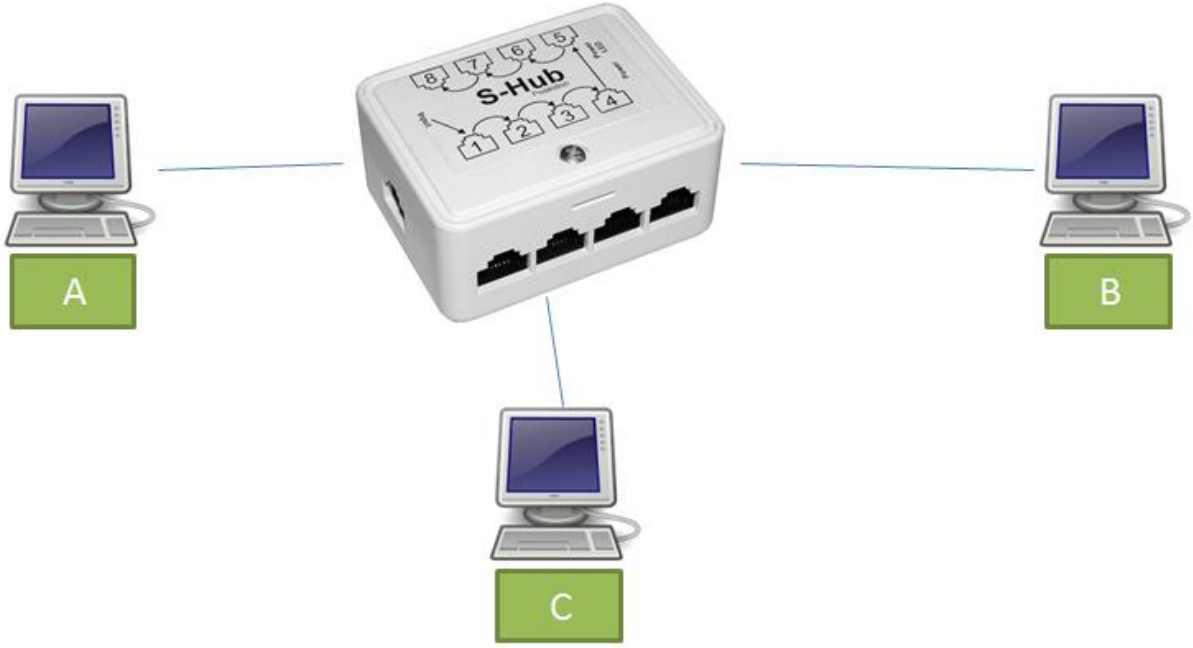
רכיבי רשת בשכבת הקו ובשכבה הפיזית

אז למדנו רבות על שכבת הקו, והבנו איך היא פועלת. למדנו על פרוטוקול Ethernet, וכן על פרוטוקול ARP שמאפשר לנו למפות בין כתובות לוגיות לכתובות פיזיות. אך איך כל הדבר הזה עובד? איך, למעשה, ישויות שונות (כמו מחשבים) מחוברות באותה שכבה שנייה? האם כל המחשבים ברשת מחוברים על אותו הכבל? לשם כך, עלינו להכיר רכיבי רשת שמחברים את הישויות השונות זו לזו (בהנחה שמדובר בתווך קווי ולא אלחוטי).

Hub (רכזת)

Hub (ובעברית – רכזת) הינו למעשה רכיב של השכבה הפיזית, כלומר השכבה הראשונה במודל השכבות. ה-Hub הוא "קופסה" שאליה מתחברים מספר כבלי רשת. הוא לא יודע איך נראות מסגרות Ethernet, לא מבין מה זה כתובת MAC ולא יודע לחשב Checksum. הוא רק מחבר כמה ישויות זו לזו.

כל "כניסה" ב-Hub אליה ניתן לחבר כבל רשת נקראת **פורט** (באנגלית – **Port**)[64]. כאשר מחשב שמחובר ל-Hub שולח מסגרת, ה-Hub מעתיק את המסגרת ושולח אותה לכל הפורטים שלו, מלבד לזה שממנו המסגרת נשלחה. כך למשל, בדוגמה הבאה:



המחשבים A, B ו-C מחוברים זה לזה באמצעות Hub. במקרה שהמחשב A ישלח מסגרת אל B, המסגרת תגיע הן אל המחשב B והן אל המחשב C. אם המחשב A ישלח הודעה אל המחשב C, המסגרת גם תגיע הן אל המחשב B והן אל המחשב C. אם המחשב B ישלח מסגרת המיועדת אל המחשב A, היא תגיע הן למחשב A והן למחשב C, וכך הלאה.

במקרה שהמחשב A שלח מסגרת אל המחשב B, המסגרת תגיע כאמור הן אל המחשב B והן אל המחשב C. בשלב זה, כרטיס הרשת של כל מחשב צריך להבין האם המסגרת מיועדת אליו, על פי כתובת היעד של המסגרת בשכבה השנייה (למשל – כתובת היעד של Ethernet עליה למדנו). אם המסגרת מיועדת אל כרטיס הרשת הרלוונטי (בדוגמה שלנו – מחשב B), הוא ידאג לשלוח את המידע למי שצריך לטפל בו (למשל – מערכת ההפעלה). אם לא (בדוגמה שלנו – מחשב C), המסגרת נזרקת.

השימוש ב-Hub מאפשר אמנם לחבר מספר מחשבים זה לזה, אך יש בו בעיות רבות. ראשית, העובדה שכל המסגרות מגיעות לכלל המחשבים עשויה לפגוע בפרטיות של המשתמש, שכן כרטיס רשת שלא אמור לראות את המסגרת מקבל אותה. שנית, העובדה שהמסגרות מגיעות תמיד לכלל הישויות מעמיסה בצורה

[64] על אף שזהו אותו השם, שימו לב לא להתבלבל בין פורטים פיזיים לבין הפורטים עליהם למדנו בפרק שכבת התעבורה.

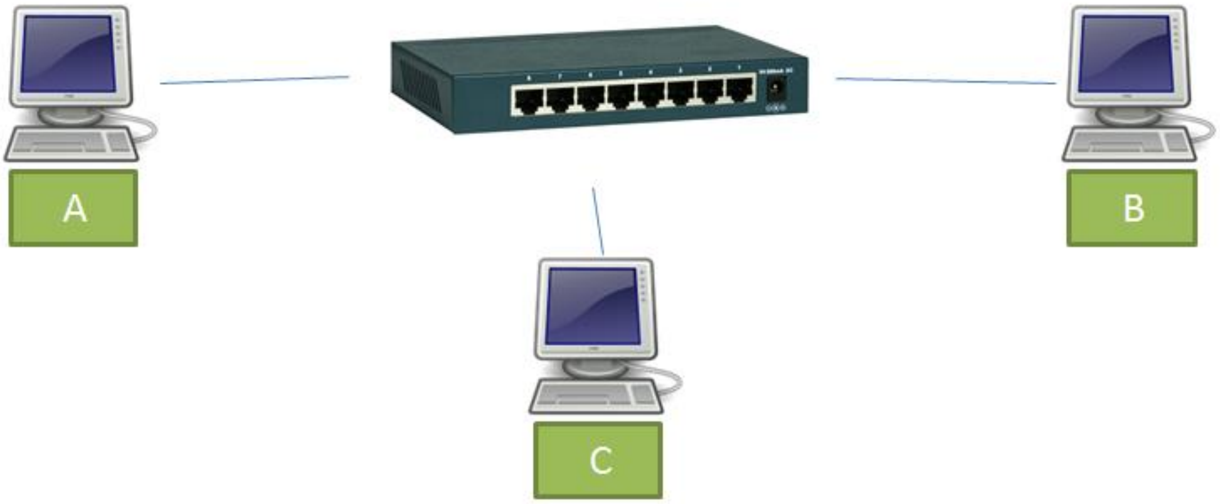
משמעותית על הרשת. היא מעמיסה הן על החיבורים (שבהם יכולה להישלח מסגרת אחת בלבד בכל פעם), והן על כרטיסי הרשת של כל ישות שצריכים לטפל בכל מסגרת. שלישית, השימוש ב-Hub לא מונע התנגשויות, בעיה עליה נלמד בהמשך הפרק.

מכל הסיבות האלו, השימוש ב-Hub אינו מספיק טוב. על מנת להתגבר עליהן, הומצא ה-**Switch**.

Switch (מתג)

בניגוד ל-Hub, עליו למדנו קודם, ה-Switch (בעברית – מתג) הוא כבר רכיב שכבה שנייה לכל דבר. ה-Switch מכיר פרוטוקולים של שכבת הקו (לדוגמה – פרוטוקול Ethernet) ומכיר כתובות MAC. חלק מה-Switchים גם יודעים לחשב Checksum ו"לזרוק" מסגרות עם Checksum שגוי.

מבחינה חיצונית, Switch ו-Hub הם די דומים: שניהם נראים כמו קופסה עם כמה פורטים אליהם ניתן לחבר כבל רשת. עם זאת, הפונקציונליות שלהם שונה מאוד. לאחר שה-Switch למד את הרשת, הוא מעביר מסגרת מהפורט בה הוא קיבל אותה אל הפורט הרלוונטי בלבד. בדוגמה הבאה:



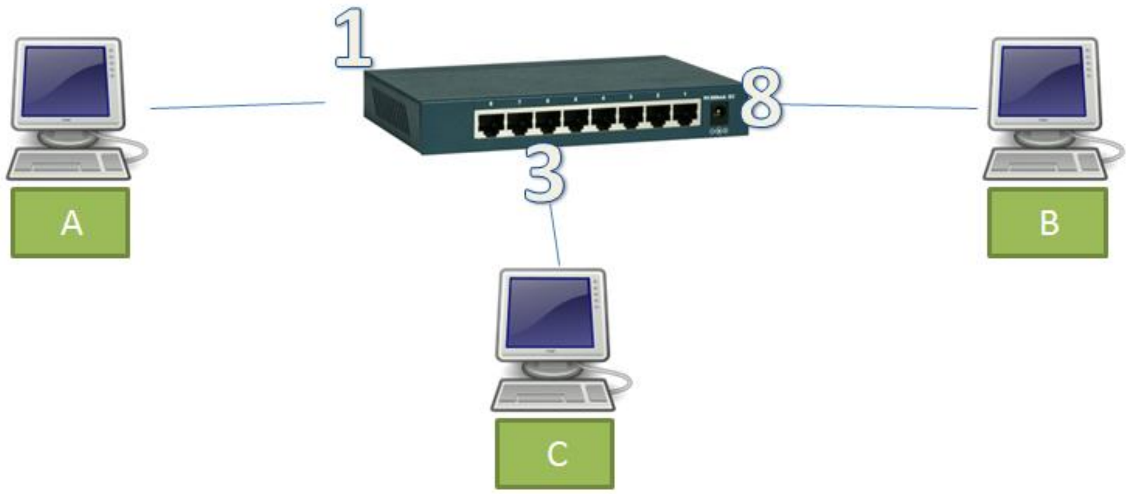
המחשבים A, B ו-C מחוברים ל-Switch. אם המחשב A שלח מסגרת למחשב B, המסגרת תגיע אל המחשב B בלבד – ולא תגיע למחשב C או תוחזר אל המחשב A. באותו אופן, אם המחשב B שלח מסגרת למחשב C, המסגרת תגיע אליו בלבד ולא תגיע אל המחשב A (וכמובן לא תוחזר אל המחשב B).

אי לכך, ל-Switch יש יתרונות רבים על פני ה-Hub. היות שכל מסגרת מגיעה רק אל היעד שלה, אין פגיעה בפרטיות המשתמשים. בנוסף, הוא חוסך את העומס הרב שהיה נוצר לו היינו משתמשים ב-Hub. כמו-כן, Switch מסייע במניעה של התנגשויות, עליהן נלמד בהמשך.

כיצד Switch פועל?

הבנו שה-Switch יודע להעביר כל מסגרת אל הפורט המיועד אליה בלבד. אך כיצד הוא עושה זאת? איך הוא יודע היכן כל ישות רשת נמצאת?

ובכן, ל-Switch יש טבלה שעליו למלא בזמן ריצה. הטבלה תמפה בין כתובת MAC לבין הפורט הפיזי הרלוונטי. לכל פורט פיזי יש מספר המאפשר לזהות אותו. לדוגמה, ב-Switch של הרשת שהצגנו לעיל, יש שמונה פורטים, שממוספרים מפורט 1 ועד פורט 8. אם נביט שוב בדוגמה לעיל, אך הפעם נסתכל גם על מספרי הפורטים:



נראה שפורט מספר 1 מקושר למחשב A, פורט מספר 3 מקושר למחשב C, ופורט מספר 8 מקושר למחשב B. אי לכך, על ה-Switch לבנות אצלו טבלה שתיראה בסופו של דבר כך:

Port	MAC Address
1	A
3	C
8	B

כמובן שה-Switch לא יודע שלמחשב קוראים A, אלא הוא שומר את כתובת ה-MAC של כרטיס הרשת שלו (לדוגמה: D4:BE:D9:D6:0C:2A). לצורך נוחות הקריאה, נשאיר בטבלה את שם המחשב ולא את כתובת ה-MAC של כרטיס הרשת.

נאמר וה-Switch הינו Switch חדש ברשת, כלומר הוא עדיין לא הספיק להכיר אותה. בשלב זה, הטבלה שלו תהיה ריקה:

MAC Address	Port

כעת, המחשב A שולח מסגרת אל מחשב B. מה על ה-Switch לעשות? הרי הוא לא יודע לשייך את כתובת ה-MAC של כרטיס הרשת של המחשב B לשום פורט פיזי.
במקרה זה, מכיוון שה-Switch אינו יודע למי להעביר את המסגרת, הוא מתנהג בדומה ל-Hub ומעביר אותה לכל הפורטים מלבד לזה שאליו היא נשלחה. כלומר, בדוגמה הזו, הוא יעביר את המסגרת אל מחשב B ואל מחשב C, אך לא חזרה אל מחשב A.

אך בנוסף, ה-Switch למד משהו. הוא ראה את המסגרת שהגיעה ממחשב A מגיעה מפורט מספר 1. מעבר לכך, הוא יכול לקרוא את המסגרת, ולראות את כתובת ה-MAC שמצוינת ב**כתובת המקור** של החבילה. בשלב זה, ה-Switch למד שכתובת ה-MAC של כרטיס הרשת של מחשב A מחוברת אל פורט 1. כעת, הוא יכול לציין זאת בטבלה שלו:

MAC Address	Port
A	1

נאמר ועכשיו מחשב A שוב שולח מסגרת אל מחשב B. חשבו על כך – האם הפעם תהיה התנהגות שונה?
התשובה היא – לא. ה-Switch אמנם יודע איפה נמצאת כתובת ה-MAC של כרטיס הרשת של מחשב A, אך הוא אינו יודע איפה נמצאת הכתובת של כרטיס הרשת של מחשב B. אי לכך, הוא נאלץ שוב לשלוח את המסגרת לכל הפורטים מלבד לפורט המקור, כלומר למחשב B ולמחשב C.

לאחר זמן מה, מחשב B שולח מסגרת אל מחשב A. הפעם, התהליך הוא שונה. ה-Switch מסתכל בטבלה, ורואה שהוא מכיר את כתובת ה-MAC אליה המסגרת נשלחת, והיא מקושרת לפורט מספר 1. אי לכך, ה-Switch מעביר את המסגרת רק אל פורט 1, ולא לאף פורט אחר. בנוסף, הוא מסתכל במסגרת ורואה שבשדה כתובת המקור נמצאת כתובת ה-MAC של כרטיס הרשת של מחשב B.

היות שהמסגרת נשלחה מפורט מספר 8, הוא יכול להוסיף את מידע זה לטבלה:

MAC Address	Port
A	1
B	8

במצב זה, כל מסגרת שתשלח אל מחשב A (בין אם ממחשב C ובין אם ממחשב B) תגיע אל פורט מספר 1 ופורט זה בלבד. כל מסגרת שתשלח אל המחשב B (בין אם ממחשב A ובין אם ממחשב C) תגיע אל פורט מספר 8 ופורט זה בלבד. עם זאת, מסגרות שתישלחנה אל מחשב C, יועברו לכל הפורטים מלבד לזה שממנו הן נשלחו, שכן ה-Switch עדיין לא מכיר את כתובת ה-MAC הרלוונטית. מצב זה ישתנה כאשר מחשב C ישלח מסגרת, ואז ה-Switch יוכל ללמוד על הכתובת של כרטיס הרשת שלו, ולהשלים את הטבלה:

MAC Address	Port
A	1
C	3
B	8

שימו לב – המחשבים לא מודעים לכך שהם מחוברים ל-Hub, ל-Switch או לכל רכיב אחר בשכבת הקו. בניגוד ל-Router, עליו למדנו בפרק שכבת הרשת/ נתב (Router), שהמחשב צריך להכיר בכדי להצליח להפנות אליו חבילות, המחשב מחובר ישירות ל-Hub או ל-Switch ולא מודע לקיומו.

תרגיל 8.13 – פעולת Hub

הביטו בשרטוט הרשת שלפניכם:

כאן שלושה מחשבים ושרת (Server) מחוברים זה לזה באמצעות Hub. מחשב A מחובר לפורט מספר 1, מחשב C מחובר לפורט מספר 2, השרת מחובר לפורט מספר 3 ומחשב B מחובר לפורט מספר 4.

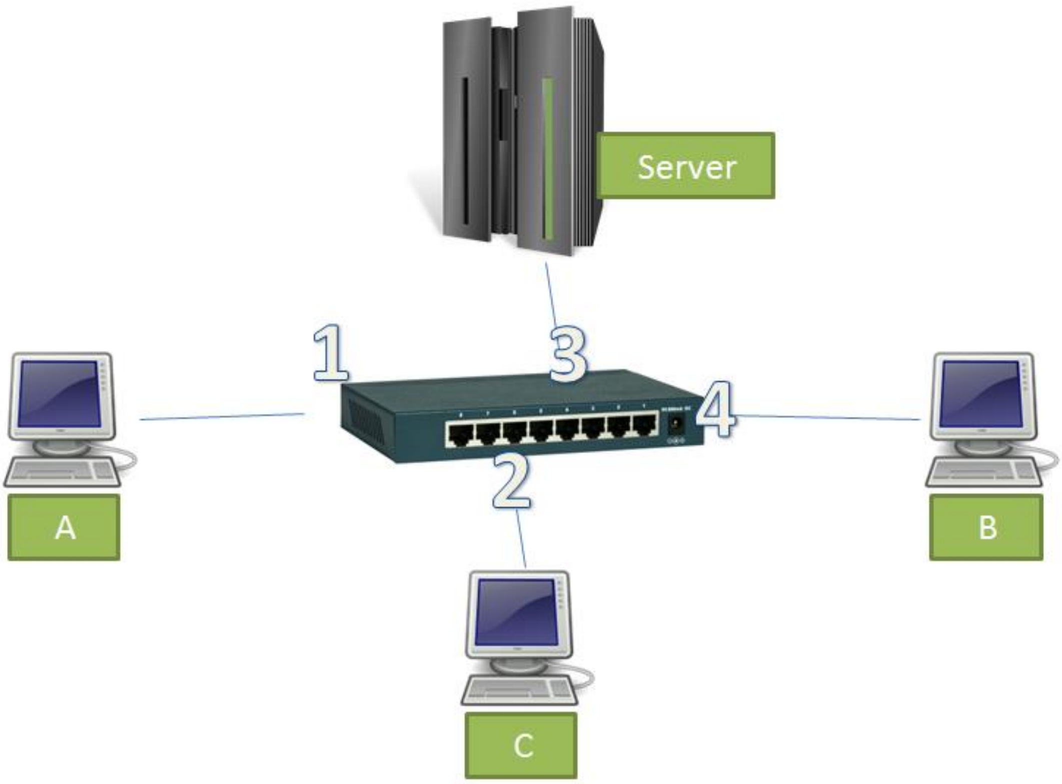
הניחו שה-Hub הינו Hub חדש ברשת. כמו כן, כל שאלה מסתמכת על השאלות הקודמות (בשאלה מספר 3 ניתן להניח שהמסגרת משאלה 2 כבר נשלחה).

ענו על השאלות הבאות:

1. המחשב A שולח מסגרת אל המחשב B. לאילו פורטים ישלח ה-Hub את המסגרת?
2. המחשב B שולח מסגרת אל השרת. לאילו פורטים ישלח ה-Hub את המסגרת?
3. המחשב C שולח מסגרת אל כולם (מסגרת Broadcast). לאילו פורטים ישלח ה-Hub את המסגרת?
4. השרת שולח מסגרת אל המחשב A. לאילו פורטים ישלח ה-Hub את המסגרת?

תרגיל 8.14 – פעולת Switch

לפניכם שרטוט רשת הזהה לשרטוט שהוצג בתרגיל הקודם, אך הפעם – הישויות השונות מחוברות באמצעות Switch, ולא באמצעות Hub:



מחשב A מחובר לפורט מספר 1 של ה-Switch, מחשב C מחובר לפורט מספר 2, השרת מחובר לפורט מספר 3 ומחשב B מחובר לפורט מספר 4.

הניחו שה-Switch הינו Switch חדש ברשת. כמו כן, כל שאלה מסתמכת על השאלות הקודמות (בשאלה מספר 3 ניתן להניח שהמסגרת משאלה 2 כבר נשלחה).

כעת, ענו על השאלות הבאות:

1. המחשב A שולח מסגרת אל המחשב B. לאילו פורטים ישלח ה-Switch את המסגרת?
2. המחשב B שולח מסגרת אל השרת. לאילו פורטים ישלח ה-Switch את המסגרת?
3. המחשב C שולח מסגרת אל כולם (מסגרת Broadcast). לאילו פורטים ישלח ה-Switch את המסגרת?
4. השרת שולח מסגרת אל המחשב A. לאילו פורטים ישלח ה-Switch את המסגרת?

שכבת הקו – סיכום

בפרק זה הכרנו את השכבה השנייה במודל חמש השכבות, שכבת הקו. בתחילת הפרק למדנו על תפקידיה של השכבה וכיצד היא משתלבת במודל השכבות. לאחר מכן, הכרנו את **פרוטוקול Ethernet**. למדנו על **כתובות MAC**, איך הן בנויות, וכיצד למצוא את כתובת ה-MAC של כרטיס הרשת שלנו. למדנו איך נראית מסגרת Ethernet, והתבוננו בשדות של פרוטוקול זה בעת ביצוע פניית HTTP.

לאחר מכן למדנו על **פרוטוקול ARP**, הבנו את הצורך בו וכן את דרך הפעולה שלו. בהמשך למדנו כיצד ניתן להשתמש ב-Scapy בכדי לשלוח מסגרות בשכבה שנייה ולהשפיע על שדות בשכבה זו, וכן תרגלנו נושא זה. לאחר מכן למדנו על רכיבי רשת – הכרנו **Hub**, שהוא רכיב של השכבה הראשונה, ו-**Switch**, שהוא רכיב של השכבה השנייה, ולמדנו על ההבדלים ביניהם. לסיום, הכרנו את סוגיית ההתנגשויות והתפקיד של שכבת הקו בהתמודדות עם סוגיה זו.

בפרקים הבאים, נלמד על השכבה הפיזית ובכך נסיים את היכרותנו עם מודל השכבות. כמו כן, נמשיך ללמוד על נושאים מתקדמים בתחום רשתות המחשבים.

נספח א' – התנגשויות

עד כה הסברנו את מטרתה של שכבת הקו, הכרנו פרוטוקול לדוגמה של שכבה זו וראינו רכיבי רשת המאפשרים לחבר ישויות שונות. עם זאת, לא התמודדנו עדיין עם בעיה שעל שכבת הקו לטפל בה – בעיית ההתנגשויות.

כדי להסביר את הבעיה, נדמיין מקרה המוכר לנו שאינו קשור לעולם המחשבים. נאמר ואנו נמצאים בעיצומו של דיון סוער של מועצת הביטחון של האו"ם. בדיון ישנם נציגים מחמש עשרה המדינות החברות במועצה, וכל אחד מהנציגים מעוניין להביע את עמדתו. במקרה שכלל הנציגים ידברו במקביל, כלומר באותו הזמן ממש, אף אחד לא יוכל להבין את דברי הנציגים האחרים. אי לכך, יש למצוא שיטה שתבטיח שרק אדם אחד יוכל לדבר בכל זמן נתון, על מנת שכולם יצליחו להבין אותו.

סוגיה זו קיימת גם ברשתות מחשבים הפועלות מעל ערוץ משותף. במקרה כזה, ישנם מספר משתתפים (ישויות רשת) שרוצים לשדר בו זמנית על אותו הערוץ. במידה שכמה ישויות ישדרו יחד, תיווצר **התנגשות (Collision)**, והמידע לא יגיע בצורה מסודרת.

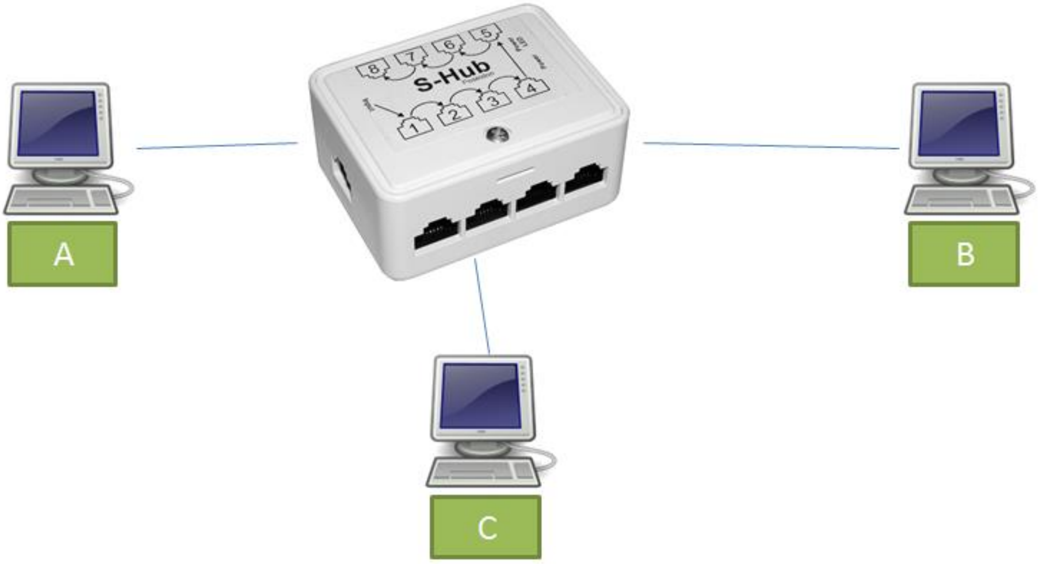
ערוץ משותף – הגדרה

ערוץ משותף הוא קו תקשורת המחבר בין מספר ישויות ומאפשר להן להחליף מידע. המידע בערוץ מופץ ב-Broadcast – כלומר, כאשר ישות משדרת, המידע מגיע לכל הישויות. בנוסף, כל ישות יכולה להאזין לערוץ בזמן שהיא משדרת – וכך לגלות האם המידע שהיא שידרה הועבר בהצלחה.

דוגמה לרשת המחוברת בערוץ משותף היא רשת בה הישויות השונות מחוברות ב-Hub. כפי שלמדנו קודם לכן, במידה שברשת ישנם מספר מחשבים שמחוברים ב-Hub, כל מסגרת שתשלח תגיע לכל המחשבים.

מהי התנגשות?

כפי שציינו קודם לכן, כאשר שתי ישויות (או יותר) משדרות בערוץ המשותף בו זמנית, נוצר מצב של **התנגשות (Collision)**. במקרה זה, המידע שנשלח יגיע באופן משובש – כלומר, המידע שיגיע הוא לא המידע שהישות התכוונה לשלוח. בדוגמה הבאה:



אם המחשבים A ו-B ישדרו מסגרת באותו הזמן, עלולה להיווצר התנגשות. אם נראה למשל את הקישור בין המחשב C לבין ה-Hub, נראה שבאותו הזמן אמורה להישלח עליו המסגרת שהגיעה ממחשב A, כמו גם המסגרת שהגיעה ממחשב B.[65] במקרה זה תהיה התנגשות, והמידע שיגיע למחשב C יהיה משובש – הוא לא יהיה זהה למידע שנשלח ממחשב A ולא לזה שנשלח ממחשב B. במקרה זה, על מנת להצליח להעביר את המידע שמחשבים A ו-B רצו לשלוח, יש לשלוח את המסגרות מחדש.

זמן שבו מתרחשת התנגשות נחשב ל"זמן מת" – מכיוון שלא עובר בקו שום מידע משמעותי, ויש לשדר מחדש כל מסגרת שתשלח – אין סיבה להשתמש עוד בערוץ המשותף ולשלוח עוד מסגרות. ברור למדי כי זוהי תופעה שלילית, שכן אנו מבזבזים זמן בו לא נשלח שום מידע על הערוץ.

מה תפקידה של שכבת הקו בנוגע להתנגשויות?

על שכבת הקו להגיע לנצילות מירבית של הקו – כלומר, לצמצם את הזמנים המתים עד כמה שניתן.

[65] ישנם Hubים חכמים שיודעים להמנע ממקרים כאלה ובכך למנוע התנגשויות, אך בפרק זה נתעלם ממקרים אלו.

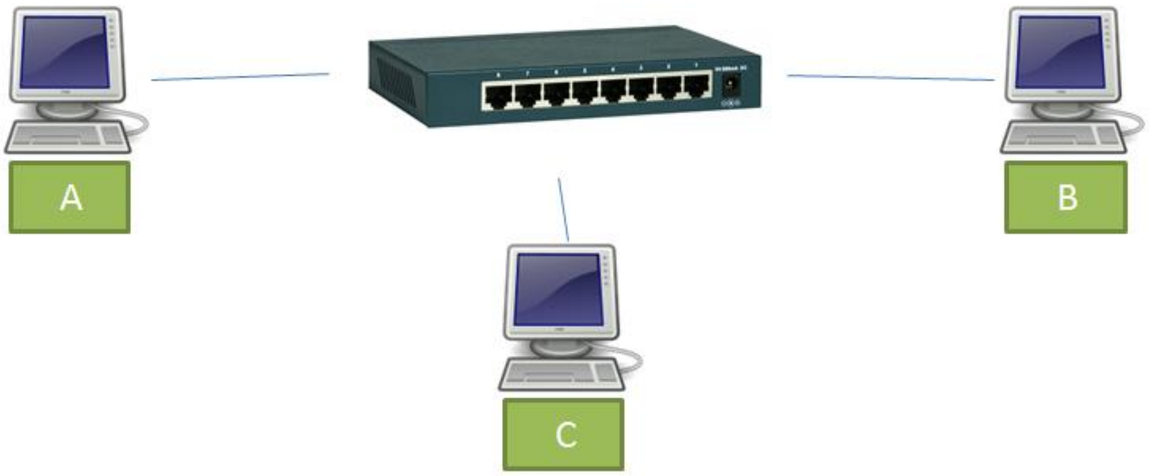
מניעת התנגשויות

יש דרכים רבות למנוע התנגשויות, והשיטות התקדמו עם הזמן. אחד הפרוטוקולים הראשונים שניסו להתמודד עם סוגיה זו נקרא ALOHA. לפי פרוטוקול זה, כאשר ישות מסוימת רוצה לשדר מסגרת, היא מתחילה לשדר אותה מיד. בזמן שהתחנה משדרת, היא מאזינה לערוץ ובודקת האם השידור הועבר באופן תקין. בזמן שידור המסגרת, היא בודקת האם המידע שהיא קלטה מהערוץ זהה למידע שאותה היא שידרה. אם המידע זהה – הכל בסדר. אך אם המידע שונה – הייתה התנגשות.

במקרה של התנגשות, הישות ממתינה פרק זמן אקראי, ולאחריו מנסה לשדר את המסגרת שוב. חשוב שהישות תמתין פרק זמן אקראי, שכן אחרת גם הישות השנייה שניסתה לשדר וגרמה להתנגשות, הייתה מחכה אותו זמן כמוה, והייתה נוצרת התנגשות נוספת. חשבו למשל על חמישה אנשים הנמצאים בחדר חשוך לחלוטין. עליהם לנסות ולדבר, אך לא להתחיל לדבר יחד. אם שני אנשים (למשל: נגה ואופיר) מתחילים לדבר באותו הזמן ממש, מתרחשת התנגשות. באם לאחר ההתנגשות, נגה ואופיר יחכו חמש דקות בדיוק, ויתחילו לדבר מחדש – תיווצר שוב התנגשות. לכן, כל אחד מהם ימתין זמן רנדומלי. למשל, נגה תמתין דקה ואז תנסה לדבר, בעוד אופיר ימתין שלוש דקות בטרם יתחיל להשמיע את קולו. כך, השניים צפויים להצליח להעביר את המסר שלהם מבלי שתיווצר התנגשות.

זוהי דוגמה אחת בלבד לניסיון להתמודד עם התנגשויות על ערוץ משותף. בפרק זה לא נסקור דרכים נוספות, אך קוראים סקרנים מוזמנים להרחיב את הידע שלהם בסעיף צעדים להמשך של פרק זה.

במקום להתמודד עם התנגשויות כשהן מתרחשות, ניתן גם למנוע התנגשויות מראש. דרך משמעותית מאוד לעשות זאת ברשתות Ethernet נוצרה כאשר הומצא ה-Switch. בדוגמה הבאה:



אם מחשב A משדר מסגרת למחשב C, וגם מחשב B משדר מסגרת למחשב C, ה-Switch יודע לשלוח את המסגרת רק כאשר הערוץ פנוי. כלומר, הוא ישלח קודם את אחת המסגרות (למשל – זו ששלח המחשב A), ורק לאחר מכן את המסגרת השנייה (למשל – זו ששלח המחשב B). כך, ה-Switch מצליח למנוע התנגשויות מראש, בלא טיפול מורכב.

הפתרון של שימוש ב-Switch אפשרי במקרים מסוימים (כמו רשתות Ethernet), אך לא תמיד. למשל, ברשת WiFi בה החיבור עובר באוויר, כל המסגרות מגיעות לכל הישויות שנמצאות בטווח הקליטה. במקרים כאלו, יש להשתמש בפתרונות אחרים.

שכבת הקו – צעדים להמשך

על אף שלמדנו רבות על שכבת הקו, נותרו נושאים רבים בהם לא העמקנו. מניעת התנגשויות מהווה נושא מרתק, ובפרק זה נגענו רק בקצה המזלג במשמעות שלו ובדרכים שונות להשיג אותו. כמו כן, התמקדנו בפרוטוקול Ethernet וכמעט לא הזכרנו מימושים נוספים, כגון WiFi או Bluetooth. לא שאלנו את עצמנו כיצד השכבה השנייה מצליחה לבצע מסגור – הפרדה של רצף המידע למסגרות שונות, כיצד היא יודעת מתי מסגרת מתחילה ומתי היא נגמרת. בנוסף, לא הסברנו על המושג Virtual LANs.

אלו מכם שמעוניינים להעמיק את הידע שלהם בשכבת הקו, מוזמנים לבצע את הצעדים הבאים:

קריאה נוספת

בספר המצוין Computer Networks (מהדורה חמישית) מאת Andrew S. Tanenbaum ו- David J. Wetherall, הפרקים השלישי והרביעי מתייחסים במלואם לשכבת הקו. באופן ספציפי, מומלץ לקרוא את הסעיפים:

- 3.1.2 – מסגור.
- 4.2 – התמודדות עם התנגשויות בערוץ משותף.
- 4.3.2 – מבנה מסגרת Ethernet. באופן ספציפי, בפרק זה לא הרחבנו על המקרה בו שדה ה-Type מעיד על אורך המסגרת. כמו-כן, לא הזכרנו מדוע אורך המסגרת חייב להיות 64 בתים או יותר. התשובות לשאלות אלו נמצאות כאן.
- 4.4.1, 4.4.3, 4.4.4 – רשתות אלחוטיות.
- 4.6 – Bluetooth.
- 4.8.5 – Virtual LANs.

בספר Computer Networking: A Top-Down Approach (מהדורה שישית) מאת James F. Kurose, הפרק החמישי מוקדש כולו לשכבת הקו. כמו כן, הפרק השישי מוקדש לרשתות אלחוטיות וסלולריות. באופן ספציפי, מומלץ לקרוא את הסעיפים:

- 5.3 – פרוטוקולים בגישה לערוץ משותף.
- 5.4.4 – Virtual LANs.
- 6.3 – רשתות אלחוטיות.

תרגיל 8.15 – זיהוי מרחוק של מחשב מסניף (אתגר)

בפרק Wireshark ומודל השכבות/ Capture Options, למדנו על האפשרות של הסנפה ב- Promiscuous Mode. אז, הסברנו שהמשמעות של אפשרות זו היא להכניס את כרטיס הרשת ל"מצב פרוץ", מה שיגרום לכך שנראה בהסנפה את כל המסגרות שרואה כרטיס הרשת, גם כאלו שלא מיועדות אליו. כעת אנו מסוגלים להבין את משמעות ה-Promiscuous Mode בצורה טובה יותר. במצב זה, נראה בהסנפה גם מסגרות שהגיעו אל כרטיס הרשת שלנו מבלי שכתובת ה-MAC בשדה כתובת היעד של המסגרת תהיה הכתובת של כרטיס הרשת שלנו, או כתובת של קבוצה בה הוא חבר[66] (למשל מסגרת המיועדת ל-Broadcast, כלומר לכל הישויות ברשת). ייתכן שמסגרות כאלו יגיעו אל כרטיס הרשת שלנו, למשל, מכיוון שברשת שלנו המחשבים מחוברים באמצעות Hub. ייתכן גם שנראה מסגרות כאילו מכיוון שה-Switch עדיין לא למד להכיר את הכתובות השונות.

האם נוכל לזהות מרחוק מחשבים ברשת שכרגע מסניפים? המסמך הבא: http://goo.gl/2LZwqP מתאר שיטה לזיהוי מרחוק של כרטיסי רשת במצב Promiscuous Mode. מכיוון שבדרך כלל מי שמסניף אכן משתמש באפשרות זו, נוכל להשתמש בשיטה המתוארת כדי לזהות מחשבים שמסניפים ברשת.

קראו את המאמר, והבינו את השיטה המוצעת לזיהוי כרטיסי רשת שנמצאים ב-Promiscuous Mode. לאחר מכן, כתבו סקריפט באמצעות Scapy שמממש את השיטה הזו, ומדפיס את כתובת ה-MAC של כל כרטיס רשת שהוא מזהה ככרטיס שנמצא במצב Promiscuous Mode. הריצו את הסקריפט ברשת שלכם מבלי שאף מחשב מסניף, וודאו כי הסקריפט לא מתריע על כך שיש מחשבים מסניפים. לאחר מכן, הפעילו הסנפה באחד המחשבים ברשת והריצו את הסקריפט בשנית. ודאו כי הפעם הסקריפט מתריע על המחשב המסניף.

[66] על המשמעות של כתובות Ethernet של קבוצות, תוכלו לקרוא בנספח א' של פרק זה.

נספח א' – כתובות Ethernet של קבוצות

כתובת Ethernet יכולה להיות שייכת לישות אחת (Unicast) או למספר ישויות (Multicast). כתובות השייכות למספר ישויות מתארות למעשה כתובת של קבוצה – למשל קבוצת כל הנתבים ברשת, או קבוצת כל הישויות שמריצות תוכנה מסוימת. כאשר כרטיס רשת Ethernet מסתכל על מסגרת Ethernet, הדבר הראשון שהוא רואה הוא כתובת היעד של המסגרת. אם כתובת היעד שייכת אליו – כרטיס הרשת "מעלה" את המידע בחבילה להמשך טיפול (לדוגמה, אם מדובר מחבילה מסוג IP – הוא מעלה את המידע למי שמטפל בחבילות IP, למשל מערכת ההפעלה). יש שני מקרים בהם כתובת היעד שייכת לכרטיס הרשת:

- הכתובת היא של כרטיס הרשת עצמו. כלומר, כתובת Unicast.
- הכתובת היא של קבוצה אליה כרטיס הרשת שייך. כלומר, כתובת Multicast.

כדי לדעת האם כתובת מסוימת היא כתובת Multicast או Unicast, עלינו להסתכל על ביט (bit) מסוים. הביט הזה נמצא בבית (byte) העליון של הכתובת. לדוגמה, נסתכל בכתובת הבאה:

02:03:04:05:06:07

כעת, נסתכל רק על הבית העליון, כלומר 02. נמיר את הבית הזה לפורמט הבינארי (כלומר, בתצוגת ביטים) שלו[67]:

00000010

כעת, עלינו להסתכל על הביט התחתון ביותר של הכתובת (מסומן ב**אדום**, הספרה האחרונה מימין):

00000010

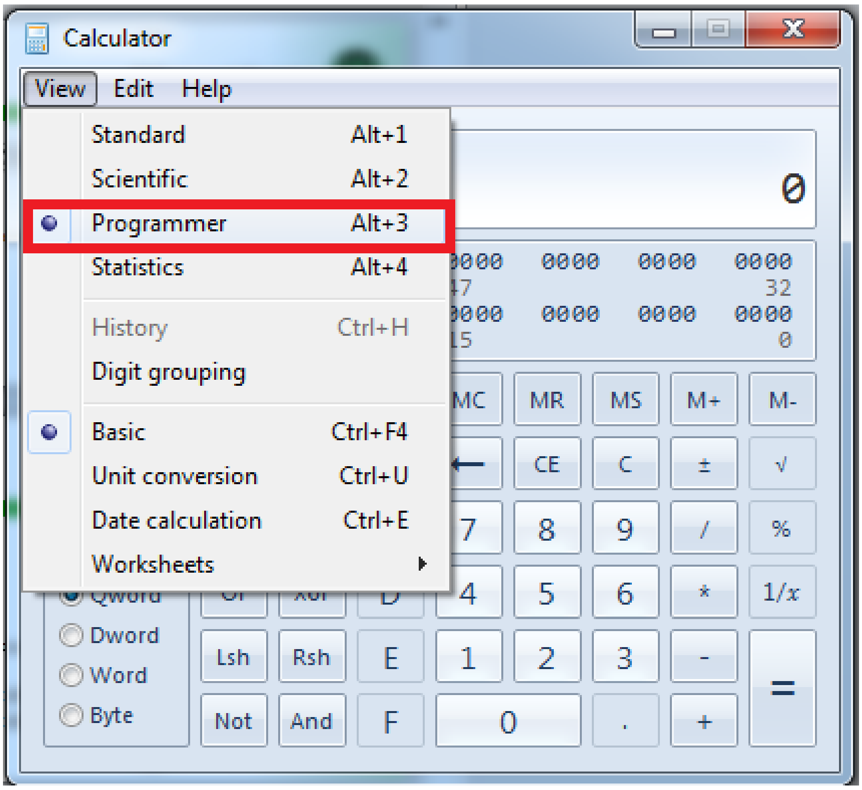
מכיוון שביט זה כבוי (הערך שלו הוא 0), מדובר בכתובת Unicast.

נסתכל על כתובת כרטיס הרשת שלנו שהצגנו קודם לכן:

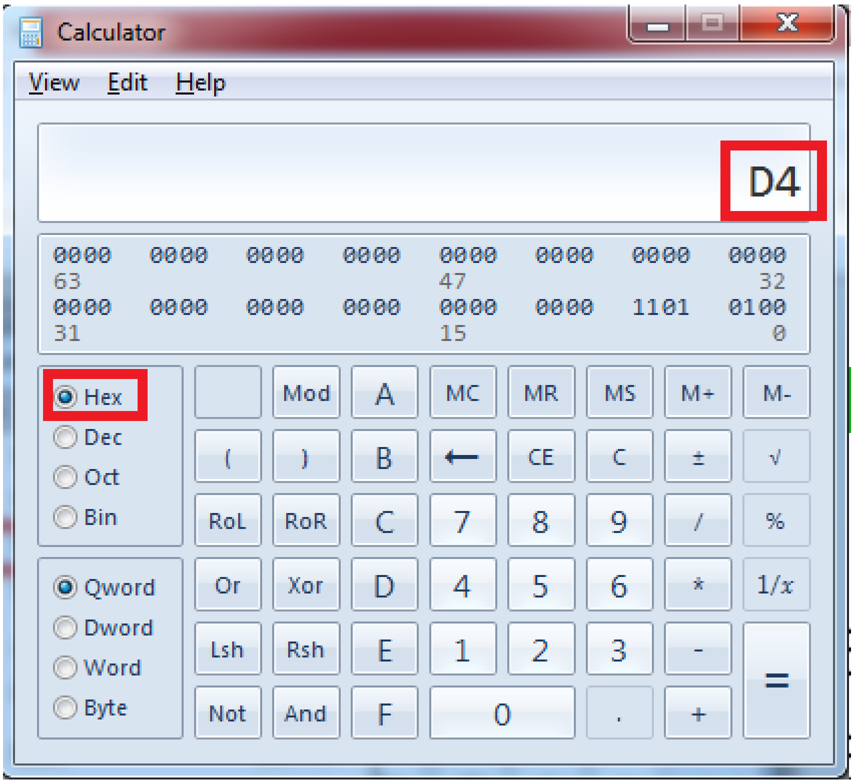
D4:BE:D9:D6:0C:2A

כדי להבין האם כתובת זו היא Unicast או Multicast, נסתכל בבית העליון ביותר, שהוא הבית D4. כעת, על מנת להמיר אותו לפורמט בינארי, ניעזר במחשבון של Windows. היכנסו למחשבון, ובתפריט בחרו באפשרות View > Programmer:

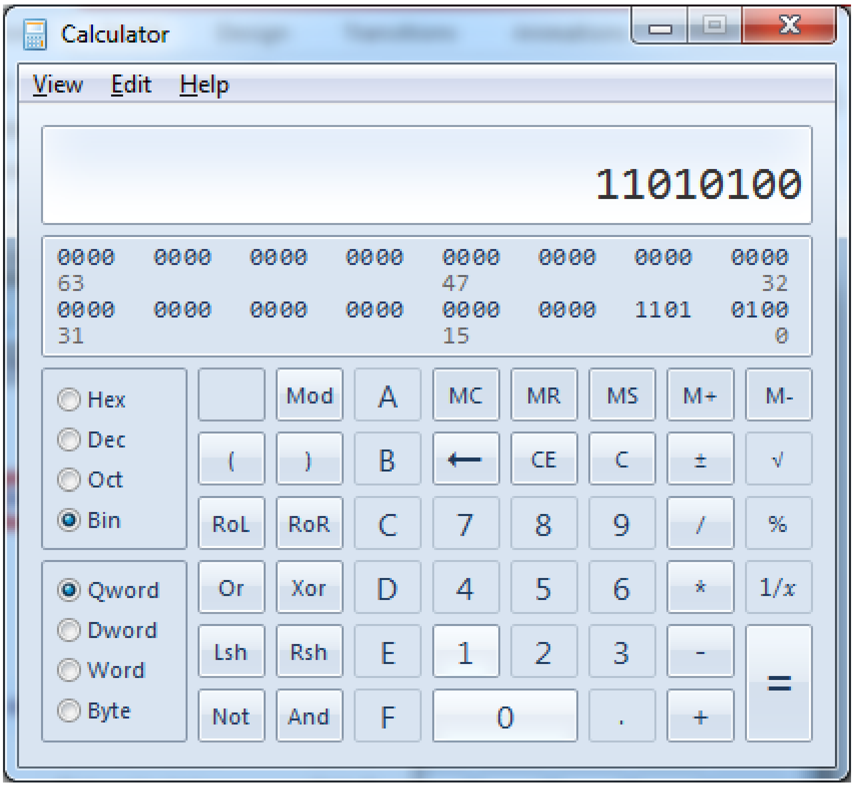
[67] אם אינכם מרגישים עדיין בטוחים במונחים "ביטים" ו-"בתים" או בהמרות בין הפורמטים השונים – אל תדאגו, הביטחון נרכש עם הזמן. עם זאת, קראו לאט וודאו כי אתם מבינים את הכוונה בדוגמאות שניתנות לפניכם.



כעת, בחרו בפורמט הקסדצימלי (Hex), והקישו את הבית הרלוונטי – D4:



כעת, בחרו בפורמט בינארי (Bin). המחשבון יעשה עבורכם את ההמרה:



אם כן, הערך הבינארי של הבית הוא:

11010100

הביט התחתון (מסומן ב**אדום**, הספרה האחרונה מימין) כבוי, ומכאן שהכתובת הינה כתובת Unicast. הדבר הגיוני, מכיוון שמדובר בכתובת של כרטיס רשת, וכתובת כזו היא תמיד מסוג Unicast.

בצעו את התהליך הזה גם על כתובת כרטיס הרשת שלכם, וודאו כי הכתובת היא מסוג Unicast.

כעת נבחן כתובת Ethernet נוספת:

03:04:05:06:07:08

על מנת להבין אם הכתובת היא Unicast או Multicast, נסתכל בבית העליון – 03. נבצע המרה לבסיס בינארי:

00000011

הביט התחתון (מסומן ב**אדום**, הספרה האחרונה מימין) דולק – ולכן מדובר בכתובת של קבוצה, כלומר כתובת Multicast.

כתובת Multicast נוספת הינה הכתובת FF:FF:FF:FF:FF:FF. כתובת זו היא כתובת Broadcast – כלומר הקבוצה אליה שייכות כל הישויות ברשת. שליחת מסגרת עם כתובת היעד FF:FF:FF:FF:FF:FF משמעותה שליחת המסגרת לכל הישויות שנמצאות איתנו ברשת.

תרגיל 8.16 – זיהוי כתובות Multicast

השתמשו בסקריפט שכתבתם בתרגיל 8.6 – כתובות Ethernet, אשר מבקש מהמשתמש כתובת MAC ומדפיס עליה מידע. ערכו את הסקריפט כך שידפיס גם האם הכתובת הינה Unicast או Multicast.

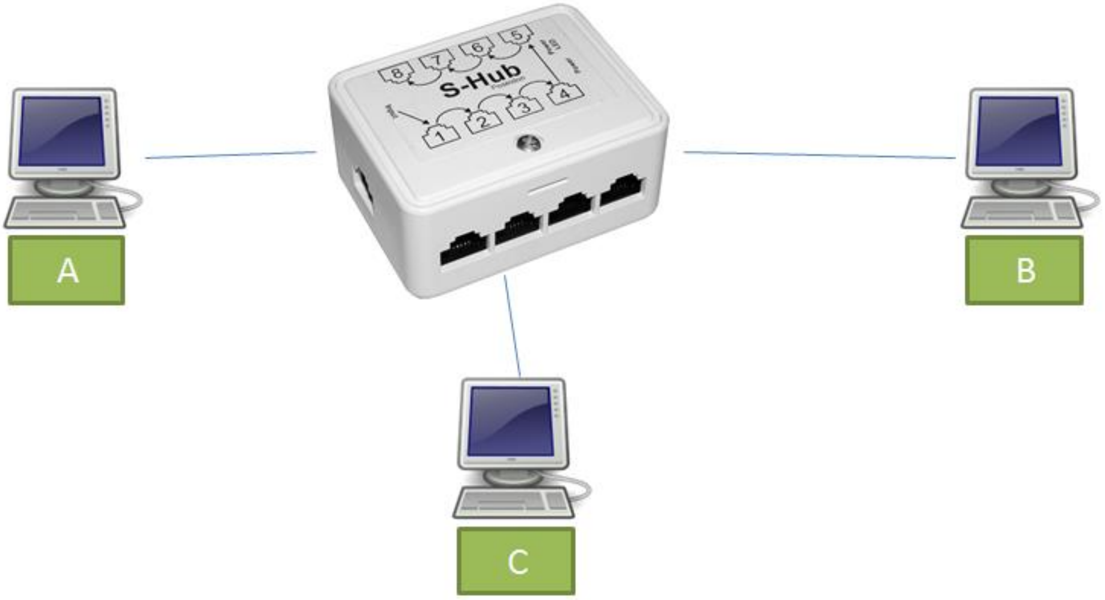
פרק 9

רכיבי רשת

בפרקים הקודמים הכרנו מספר רכיבי רשת. פרק זה נועד כדי לעשות סדר ברכיבים עליהם למדנו.

Hub (רכזת)

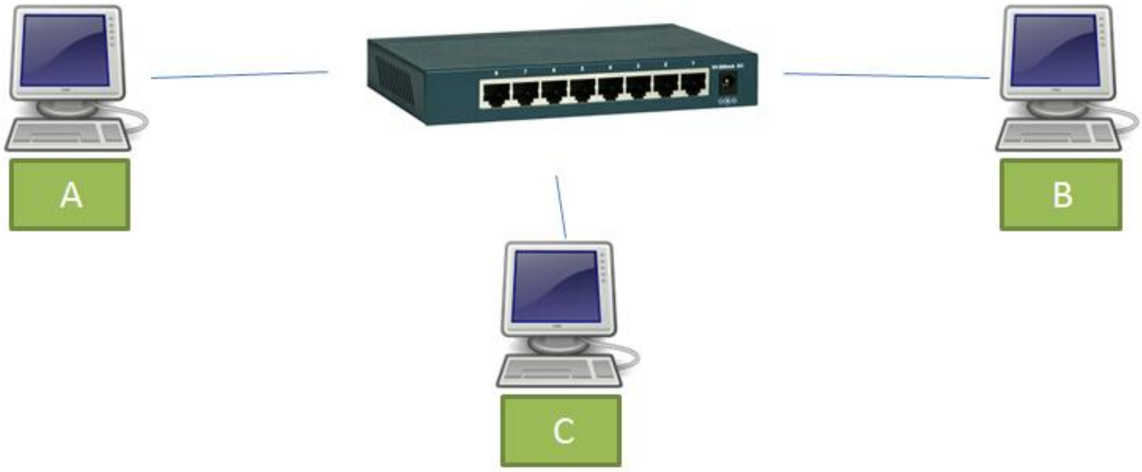
ה-Hub הינו רכיב של השכבה הפיזית, השכבה הראשונה. הוא נועד כדי לחבר כמה ישויות רשת יחד. ה-Hub אינו מכיר כתובות Ethernet או IP, מבחינתו הוא רק מעביר זרם חשמלי מפורט אחד אל פורטים אחרים. כאשר מחשב שמחובר ל-Hub שולח מסגרת, ה-Hub מעתיק את המסגרת ושולח אותה לכל הפורטים שלו, מלבד לזה שממנו המסגרת נשלחה. כך למשל, בדוגמה הבאה:



המחשבים A, B ו-C מחוברים זה לזה באמצעות Hub. אם המחשב A ישלח מסגרת אל B, המסגרת תגיע הן אל המחשב B והן אל המחשב C. במקרה שהמחשב A ישלח הודעה אל המחשב C, המסגרת גם תגיע הן אל המחשב B והן אל המחשב C. אם המחשב B ישלח מסגרת המיועדת אל המחשב A, היא תגיע הן למחשב A והן למחשב C, וכך הלאה.

Switch (מתג)

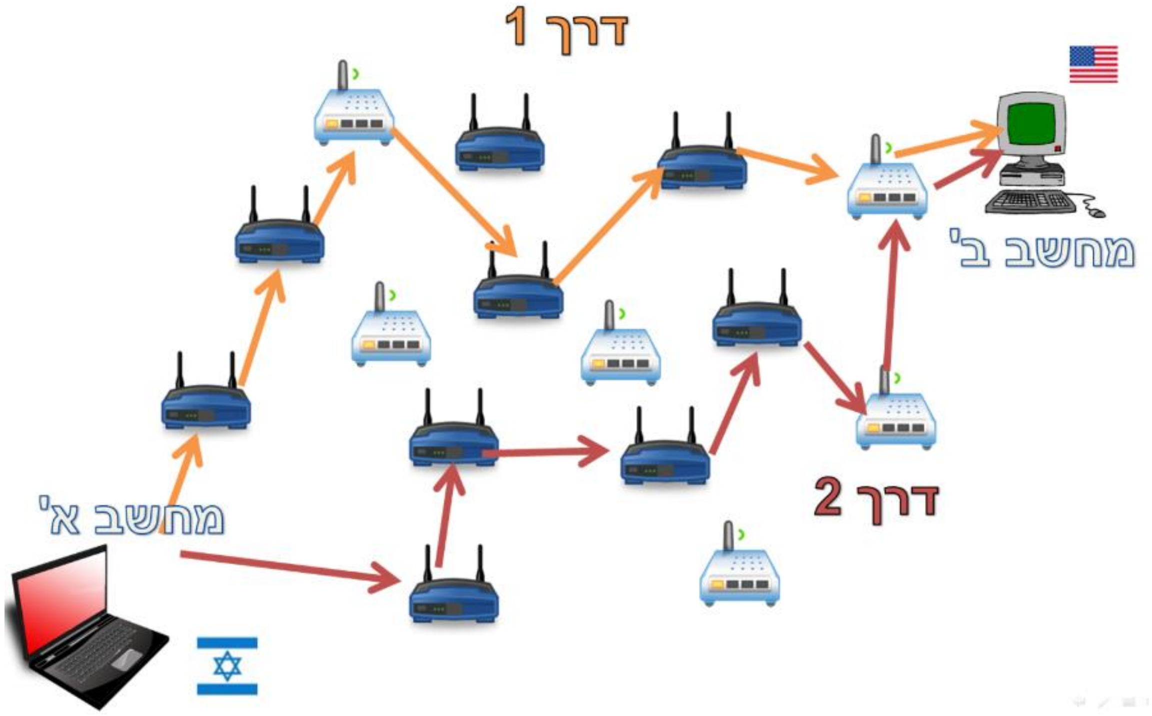
ה-Switch הינו רכיב של שכבת הקו, השכבה השנייה. אי לכך, ה-Switch מכיר כתובות MAC, מבין את המבנה של מסגרות בשכבה שנייה (למשל מסגרות Ethernet), ויודע לחשב Checksum. לאחר שה-Switch למד את הרשת, הוא מעביר מסגרת מהפורט בה הוא קיבל אותה אל הפורט הרלוונטי בלבד. בדוגמה הבאה:



המחשבים A, B ו-C מחוברים ל-Switch. אם המחשב A שלח מסגרת למחשב B, המסגרת תגיע אל המחשב B בלבד – ולא תגיע למחשב C או תוחזר אל המחשב A. באותו אופן, אם המחשב B שלח מסגרת למחשב C, המסגרת תגיע אליו בלבד ולא תגיע אל המחשב A (וכמובן לא תוחזר אל המחשב B).

Router (נתב)

הנתב (Router) הינו רכיב של שכבת הרשת, השכבה השלישית. הנתב מכיר כתובות IP, מבין את המבנה של חבילות IP ופועל על פיהן. על הנתב להבין מבנה של כתובות IP וכן מסכות רשת (Subnet Masks), ולהחליט על מסלול הניתוב הטוב ביותר עבור כל חבילה שנשלחת.



את החלטות הניתוב מבצעים הנתבים באמצעות טבלאות ניתוב דינאמיות. הטבלאות מתעדכנות באמצעות הודעות שהנתבים שולחים אחד לשני, באם יש שינוי ברשת (למשל – נתב אחד שקרס או עומס באזור מסוים ברשת).

טבלת סיכום

שם	שכבה	כתובות שמכיר	מטרה ודרך פעולה
Hub (רכזת)	פיזית (1)	לא מכיר כתובות	יוצר קשר בין מספר ישויות. מעביר כל מידע לכלל הישויות המחוברות אליו.
Switch (מתג)	קו (2)	MAC Addresses	יוצר קשר בין מספר ישויות ברמת הקו. מעביר כל מסגרת רק למי שהמסגרת מיועדת אליו, באמצעות טבלה הממפה כתובת MAC לפורט פיזי.
Router (נתב)	רשת (3)	IP Addresses	מקשר בין רשתות ומחשבים ברמת ה-IP. מחליט על הדרך הטובה ביותר לנתב חבילה ממקור אל יעד. לרוב פועל בעזרת טבלאות ניתוב דינאמיות.

פרק 10

השכבה הפיזית (העשרה)

מבוא

עד כה למדנו, שבמודל 5 השכבות יש תפקידים שונים לכל שכבה. הכרנו את שכבת הקו, שתפקידה לאפשר לשני מחשבים לדבר אחד עם השני באופן ישיר. השכבה השלישית, שכבת הרשת, תפקידה לאפשר לשני מחשבים לדבר אחד עם השני בין רשתות שונות, קרי דרך רכיבי תקשורת מתווכים. שכבה נוספת מאפשרת לשני מחשבים, ללא תלות במרחק בין אחד לשני, להעביר מספר ערוצי מידע ביניהם. ביחד, שכבות אלו מרכיבות את רשתות התקשורת שאנו כל כך רגילים לשימוש היומיומי בהן. למרות שנשמע כאילו כיסינו כבר הכל, מגלישה ל-Google ועד העברת מסגרות בין שני מחשבים מחוברים באופן ישיר, עדיין חסרה לנו שכבה בודדת אחת – השכבה שתפקידה להגדיר את המילים (או האותיות, אם תרצו) הבסיסיות בהן מחשבים משתמשים בשביל לדבר אחד עם השני: **הסיבית** (באנגלית – **bit**, ולעיתים גם בעברית – **ביט**).
כל ארבע השכבות אשר נשענות על השכבה הפיזית, מניחות שאפשר להעביר סיביות, או ביטים (bits), בין שני רכיבי מחשוב.

סיבית (Bit, ביט) – קיצור של המושג ספרה בינארית, ספרה שיכולה להחזיק אחד משני ערכים: 0 או 1. סיבית היא תרגום של המושג הלועזי bit, שהוא קיצור לביטוי binary digit. בספר זה אנו משתמשים במושג הלועזי, אך נאיית אותו בעברית: ביט או ביטים.

העברה של ביט בין שני רכיבי מחשוב אמנם נשמעת כמו פעולה די בסיסית, הרי בסך הכל מדובר בשתי אופציות (0 או 1). תופתעו לגלות שמאחורי פעולה זו מתחבאת שכבה שלמה, השכבה הפיזית. מגוון האפשרויות להעביר ביטים בין מחשבים הינו עצום, ולכן יש מגוון רחב של טכנולוגיות המממשות את השכבה הזו.

מטרת השכבה הפיזית היא להעביר ביט יחיד בין רכיבי מחשוב.

אפשר להתבונן בשיטות שונות להעברת ביט יחיד דרך הטכנולוגיות המלוות אותנו באמצעי התקשורת בחיינו:

- המחשב הביתי שגולש דרך מודם ה-ADSL.
- ממיר שידורי הכבלים והלוויין.
- הטלפון הסלולרי.
- הדיבורית האלחוטית באוטו.
- מכשיר ה-GPS.
- אפילו מנורות הרחוב, שבאופן מסתורי נדלקות בתזמון מדויק על פי השעה בה שוקעת וזורחת השמש.

כל אחת מהטכנולוגיות הללו מסתמכת על שיטה אחרת שתפקידה להגדיר כיצד להעביר ביט או אוסף ביטים על גבי תווכים שונים: באוויר, בכבלי מתכת ובכבלי זכוכית.

תווך (Medium) – ברשתות תקשורת, תווך התקשורת הוא החומר, או האמצעי הפיזי, המשמש להעברת המידע.

לא ניכנס לכל מימושי השכבה הפיזית בפרק זה, אך נשים לב שרוב המימושים מתחלקים באופן גס על פני שלושה תווכים:

- כבלי מתכת (לרוב נחושת).
- אוויר (״אלחוטי״).
- סיבים אופטיים (שהם בעצם זכוכית או פלסטיק).

בפרק זה, ננסה להבין כיצד אפשר להעביר מידע בחומרים הנ״ל, ונפרט בקצרה את התכונות השונות של כל שיטת תקשורת. בנוסף, ננתח לעומק מספר דוגמאות מהחיים, באמצעותן נשפוך אור על שיטות התקשורת בחיינו ועל מאפייניהן החשובים. נתחיל ממבט אל העבר בו נוסדו שיטות שונות להעברת מידע, ולאחר מכן נתבונן באמצעי התקשורת הפיזית שקיימים בכל בית ממוצע בישראל. לסיום, נבחן אתגרי תקשורת גדולים יותר, כמו רשת בין יבשתית של חברת היי-טק גדולה.

עמודי התווך של התקשורת

כפי שציינו קודם, התווך הינו החומר המוחשי בו אנו עושים שימוש על מנת להעביר מידע (למשל, כבל נחושת או האוויר). כל תווך מכתיב אופן שונה להעברת ביטים, ומגבלות שונות על קצב העברת הביטים והמרחק אליו ניתן להעבירם.

העברת מידע באוויר

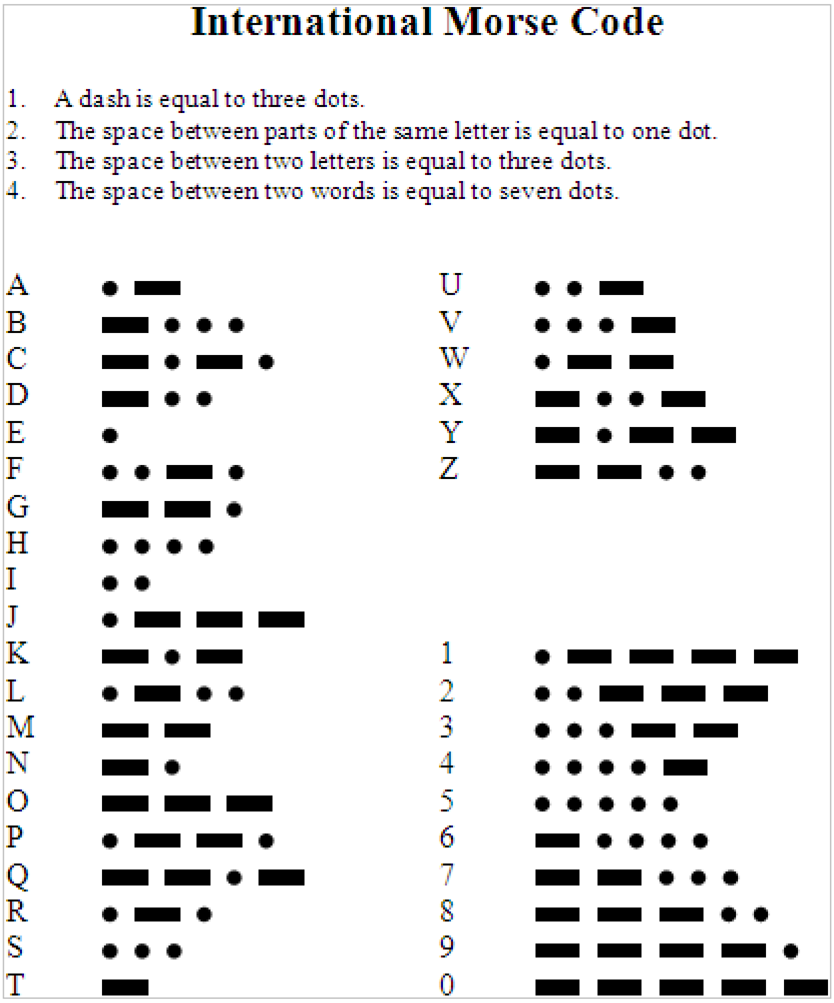
האם אי פעם שאלתם את עצמכם את השאלה הבאה:

איך מתקשר השלט הרחוק עם הטלוויזיה?

השלט הרחוק משדר אותות לטלוויזיה באור אינפרה אדום, אבל איך זה באמת עובד? ראשית, נציין כי הרעיון של שימוש באור על מנת לתקשר למרחקים נהגה עוד בימי החשמונאים. אבותינו השתמשו באש ובעשן כדי לאותת ולהעביר הודעות שונות בין גבעות מרוחקות. זה אמנם נשמע מוזר, אבל השימוש באש ובעשן היווה אמצעי לקידוד ביטים. כשיש אור (או עשן), נתייחס אליו כאילו הוא מייצג את הספרה 1, וכשאין אור (או אין עשן) נתייחס לחושך כמייצג את הספרה 0. בסוף המאה ה-18, פותחו מכשירים שנועדו להעביר אור למרחקים, ובאמצעות קידוד מוסכם מראש העבירו הודעות. דוגמה נפוצה מאוד של קידוד פשוט שמאפשר העברת הודעות באמצעות איתותי אור היא קוד מורס.

קידוד (Coding או Encoding) – תהליך בו מידע מתורגם לאותות מוסכמים (למשל: אור או חושך). כל שיטת מימוש של השכבה הפיזית מגדירה אופן בו מתרגמים את הספרות 0 ו-1 לסימנים מוסכמים על גבי התווך.

קוד מורס שהזכרנו קודם לכן, מהווה אחת משיטות הקידוד הוותיקות בעולמנו, ושימושים נרחבים ומגוונים נעשו בו בעבר ובהווה. קוד מורס מגדיר לכל אות באלף-בית האנגלי קוד המורכב משני סימנים: קו ונקודה. הטבלה הבאה מראה כיצד מתרגמים כל אות באלף-בית לרצף של קווים ונקודות.



קל להבין כיצד נוכל לתרגם בין קו ונקודה ל-0 ו-1, ולהחליף קידוד זה בקידוד בינארי. אבל אם נעמיק ונחשוב על קידוד מורס והאופן בו משתמשים בו, נבין שיש לנו מאפיין אחד חסר.

לו הייתם חיים במאה ה-19, וברשותכם פנס המאפשר לכם להעביר איתותי אור למרחקים, כיצד הייתם מבדילים בין מקטע אור שמייצג נקודה לבין מקטע אור שמייצג קו?

המאפיין החסר הוא הגדרה של זמן, או יותר נכון תזמון. ההבדל בין קו לנקודה הוא האורך, או משך הזמן של הבהוב האור. אפשר לומר שהמאפיין הוא מעט ״סמוי״, מאחר שכל אדם יבחין בהפרש המשכים בין הבהובי האור, ויסיק בעצמו שהקצר הינו נקודה והארוך הינו קו. אך יש מקום להגדרה יותר מדויקת כדי לאפשר לאנשים שונים לקודד מידע בקצב שונה (אם נקצר את משכי האור של קו ונקודה, נוכל להעביר יותר אותיות

בשנייה אחת). כמו כן, הגדרה מדויקת של משכי הקו והנקודה מאפשרת להפריד בין אותיות ובין מילים בקלות רבה יותר. הסתכלו שוב על הטבלה וחשבו: כיצד נוכל להבדיל בין האות Q, לבין רצף האותיות MA?[68]

נחזור אל השלט הרחוק. כעת קל מאוד לדמיין כיצד השלט הרחוק מתקשר עם הטלוויזיה באמצעות אור (כן, אינפרה אדום הוא גם אור, רק שאינו נראה על ידי העין האנושית). בכל לחיצה על כפתור בשלט הרחוק, השלט משדר רצף הבהובים באורכים משתנים, בצורה דומה מאוד לקידוד מורס, אשר הטלוויזיה קולטת ומפענחת. במקום לקודד אותיות, אפשר לקודד כפתורים כגון "העלה ווליום", "הורד ערוץ" ו-"כבה".

גלים נושאי מידע

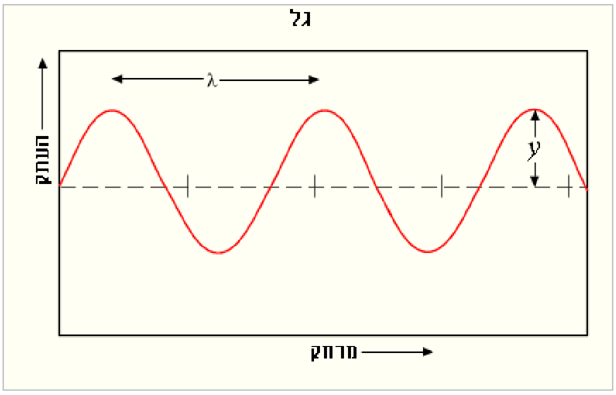
פריצת הדרך הבאה בניצול האוויר כתווך תקשורת התרחשה כמעט מאה שנים לאחר שהומצא קוד מורס, בסוף המאה ה-19, כאשר חוקרים הצליחו להעביר קוד מורס באמצעות גלים אלקטרומגנטיים למרחק של שישה קילומטרים. כדי להבין קצת מהם גלים אלקטרומגנטיים, עלינו להבין ראשית מהו גל.

גל (Wave) – התפשטות (או התקדמות) של הפרעה מחזורית בתווך במרחב. גל יכול לנוע בחומר (כמו גלים במים), אך גם באוויר (כמו גל קול) ואף בוואקום (כמו גלים אלקטרומגנטיים, שיכולים לנוע בוואקום ובתווכים רבים אחרים).

בתמונה לעיל ניתן לראות גלים המתקדמים במים. הם כמובן לא חלקים ומושלמים כמו פונקציית הסינוס שבשרטוט הבא, אבל נוכל לדמיין שאם היינו מסתכלים על גובה המים מהצד, היינו רואים גל שמזכיר פונקציית סינוס. לכל גל יש מספר מאפיינים מאוד חשובים שמתארים אותו.

[68] אם תרצו להרחיב את היכרותכם עם קוד מורס והשימושים הרבים לו, תוכלו לקרוא עליו כאן:
http://en.wikipedia.org/wiki/Morse_code

מאפיינים אלו ניתן לראות בשרטוט:



- כל גל הוא תופעה מחזורית (כמו פונקציית סינוס).
- **אורך הגל (wave length)** הוא המרחק שהגל עובר כשהוא מבצע מחזור אחד. אורך זה מסומן בשרטוט ב-λ (האות היוונית למדה).
- **זמן המחזור** הוא משך הזמן שלוקח לגל לבצע מחזור אחד.
- **תדירות הגל (Frequency)** היא כמות המחזורים שהגל מבצע בשנייה אחת. תדירות הגל נקראת גם "תדר" הגל.
- **משרעת (Amplitude)** הגל היא ה"גובה" המקסימלי אליו מגיע הגל (בשרטוט מסומן כ"העתק", ולעיתים נקרא גם **אמפליטודה** בעברית).

דפוס התנועה של גלים מופיע בחומרים ותווכים שונים בטבע: תנודות על פני מים (כמו הגלים בים), שינוי גלי בלחץ האוויר (הידוע גם כגל קול), ושינוי גלי בשדה החשמלי והמגנטי (הנקרא לרוב הגל האלקטרומגנטי).

הגל האלקטרומגנטי (Electromagnetic Wave) – הוא סוג של גל שנע בתווכים שונים במרחב (אוויר, מים, זכוכית, ריק/ואקום ועוד) באמצעות שינוי של השדות החשמליים והמגנטיים. האור שמגיע מהשמש ושאותו אנו רואים הוא גל אלקטרומגנטי שהתדר שלו נמצא בטווח שהעין רואה (אורך הגל הרלוונטי נע בין 400 ל-800 ננומטר[69] בקירוב). עוצמת האור שווה לאמפליטודה של הגל האלקטרומגנטי, שמעידה על חוזק התנודה בשדות החשמליים והמגנטיים. הגלים האלקטרומגנטיים הם תופעה מיוחדת מאוד בטבע, ותוכלו לקרוא עוד עליהם בוויקיפדיה[70].

בחזרה להעברת קוד מורס על גבי גלים אלקטרומגנטיים, כיצד אפשר לקודד נקודה וקו באמצעות גל אלקטרומגנטי?

הפתרון הטריוויאלי, בהינתן שיש לנו מחולל גלים בעוצמה (אמפליטודה) ובתדר לבחירתנו, הוא לקודד 1 ו- 0 באמצעות שידור או אי-שידור של הגל. כאשר נרצה לקודד נקודה, נשדר גל למשך שנייה. כאשר נרצה לקודד קו, נשדר גל למשך שלוש שניות. בין כל קו לנקודה נפסיק את השידור. זה אכן פתרון פשוט, אבל במציאות לא משתמשים בו, והוא רחוק מלהיות יעיל (זכרו, היום כל טלפון נייד מקודד 100 מגה סיביות בשנייה, שזה שווה

[69] ננומטר אחד שווה ל-10^{-9} מטר.

[70] קרינה אלקטרומגנטית http://he.wikipedia.org/wiki/

למאה מיליון 0ים ו-1ים בשנייה!). הסיבה בגינה פתרון זה אינו ישים נעוצה באופי ההתפשטות של גלים במרחב. דמיינו עצמכם בבריכה, קופצים מעלה ומטה ומייצרים גלים. אם תעצרו לשנייה ואז תמשיכו, האם הגלים בבריכה יעצרו גם הם? מובן שלא! לא רק שיש להם קצב התקדמות משלהם, הם גם מוחזרים מדפנות הבריכה וממשיכים לנוע הלוך ושוב למשך זמן רב. כך גם גלים אלקטרומגנטיים במרחב.

אם כך, מה השיטה האמיתית באמצעותה אפשר להעביר מידע באמצעות גלים אלקטרומגנטיים?

יש שתי שיטות שהתפתחו כבר בראשית השימוש בגלים אלקטרומגנטיים: הראשונה נשענת על שינוי האמפילטודה, והשנייה נשענת על שינוי התדר. אם זה לא נשמע לכם מוכר, חשבו שוב על תחנות הרדיו האהובות עליכם – רובן משתמשות בשיטת קידוד מבוססת שינויי תדר – FM – Frequency Modulation.

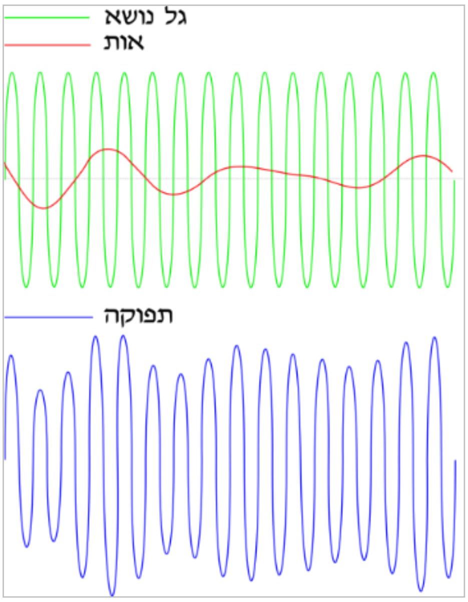
Modulation (אפנון) היא העברה של מידע על גבי גל נושא. הרעיון באפנון הוא להרכיב גל של מידע (כגון גל קול של מוסיקה) על גבי גל "נושא". גל נושא הוא גל "חלק" (גל שמאוד קרוב לפונקציית סינוס) בתדר גבוה יותר מגל המידע. לשימוש בגל נושא יש סיבות רבות, אך הן מורכבות מכדי להסבירן בפרק זה.

כעת נוכל להבין מה הן שתי שיטות האפנון שהתפתחו עם ראשית השימוש בגלים אלקטרומגנטיים:

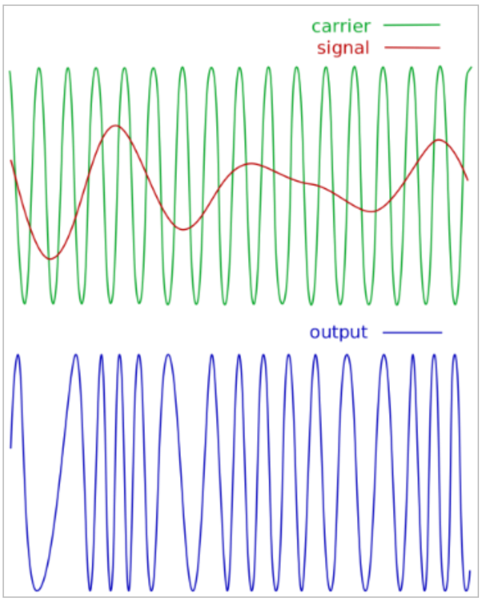
- **אפנון מבוסס אמפליטודה – Amplitude Modulation** – בשיטת אפנון זו, "מרכיבים" גל של מידע בתדר נמוך על גבי גל "נושא" בתדר גבוה וקבוע. בשיטה זו, <u>האמפליטודה</u> של הגל הנושא (בתדר הקבוע) תשתנה לפי האמפליטודה של גל המידע.
- **אפנון מבוסס תדר – Frequency Modulation** – בשיטת אפנון זו, משנים את <u>התדר</u> של הגל הנושא על פי האמפליטודה של גל המידע.

הביטו בשני התרשימים הבאים. בירוק מסומן הגל הנושא, גל בתדר גבוה וקבוע. באדום מסומן גל המידע (״האות״). בכחול, תוצאות האפנון של גל המידע בגל הנושא. איזה תרשים מראה אפנון AM ואיזה FM?

תרשים א׳



תרשים ב׳



תרשים א׳ מראה Amplitude Modulation, מאחר שהאמפליטודה של הגל הכחול משתנה (״גובה״ הגלים) בעוד התדר (המרחק בין כל פסגה של גל) נשאר קבוע.

תרשים ב׳ מראה Frequency Modulation, היות שהתדר של הגל הכחול משתנה, בעוד האמפליטודה שלו נשארת קבועה.

השימוש בגלים אלקטרומגנטיים לצורך העברת מידע התפתח מאוד מאז סוף המאה ה-19, בה היה ניתן להעביר מספר בודד של ביטים בשנייה. היום ניתן להעביר מאות מיליוני ביטים בשנייה (100Mbps) וכל זאת על-ידי מכשיר סלולרי קטנטן. לפני שנבין את שאר ההתפתחויות, נלמד גם על תקשורת חוטית.

כבלי נחושת כתווך תקשורת

המאה ה-19 הייתה מרגשת מאוד מבחינת השכבה הפיזית. בתחילת המאה ה-19 הומצא הטלגרף, מכשיר פשוט מאוד שמקודד ביטים באמצעות זרם חשמלי, דרך חוט נחושת. משך הזרם (ארוך או קצר) הפריד בין 0 ל-1, וקצב התקשורת היה תלוי ביכולת של בני האדם לייצר ולפענח את רצפי הביטים. בשלב זה אתם כבר אמורים לנחש שקידוד המידע נעשה באמצעות קוד מורס.

העברת מידע באמצעות זרמים חשמליים על חוטי נחושת הניח את היסוד לתעשייה ענקית של תקשורת, שבאופן משעשע התפתחה תחת ענף הדואר. רשויות הדואר בעולם פרשו חוטי נחושת בין ערים ומדינות, ואף על קרקעית הים. אזרחים מן המניין היו מגיעים אל הדואר כדי לשלוח ולקבל מברקים שעברו באופן מיידי באמצעות זרמים חשמליים שקדדו בקוד מורס אותיות ומילים על גבי כבלי נחושת. בסוף המאה ה-19 הומצא הטלפון, וסחף אחריו גל פיתוחים טכנולוגיים שנגיע אליהם בהמשך הפרק.

הרשת בבית

הבית הממוצע בישראל הוא בית מודרני מאוד מבחינה תקשורתית. זה לא מפתיע, בהתחשב בצורך האנושי ההכרחי לתקשר עם הקרובים והאהובים עלינו. מדענים, מהנדסים ויזמים רבים זיהו זאת כהזדמנות עסקית לאורך ההיסטוריה, וכך הומצאו ופותחו אמצעי תקשורת רבים.

חשבו במשך דקה ונסו למנות את כל אמצעי התקשורת שיש לכם בבית.

בואו ננסה למפות את אמצעי התקשורת השונים, ולהצמיד לכל אחד את התווך בו הוא עושה שימוש:

אמצעי	תווך	אמצעי	תווך
רדיו	אוויר	טלפון	כבלי נחושת
טלוויזיה בכבלים	כבלי נחושת	פקס	כבלי נחושת
טלוויזיה בלוויין	אוויר	מודם ADSL או כבלים	כבלי נחושת
טלפון סלולרי	אוויר	נתב אלחוטי	אוויר

קו הטלפון הביתי, או למה צריך פילטר למודם ADSL?

אחרי שלמדנו קצת על סוגי תווך שונים, ועל גלים אלקטרומגנטיים, הגיע הזמן שנתמקד באמצעי התקשורת שבאמת עשה את פריצת הדרך של כל הזמנים – הטלפון. הטלפון משתמש בכבלי נחושת וזרמים חשמליים בשביל להעביר קול אנושי. אם זה לא מספיק, כשהאינטרנט הגיע הביתה באמצע שנות ה-90, הוא הגיע גם על גבי אותו כבל נחושת שנשא את קו הטלפון. מדהים, לא?!. איך כבל[71] נחושת בעובי של מילימטר בודד יכול להעביר לא רק קול אנושי, אלא גם מוסיקה וסרטים באיכות HD?

נחקור את השאלה הזו דרך הפילטר הקטן שנמצא בשקע הטלפון של משתמשי ה-ADSL. על אף שפירוש המילה פילטר הוא מסנן, הפילטר מפצל את השקע לשני שקעים: אחד עבור הטלפון הרגיל, ושני עבור מודם ה-ADSL.

פילטר למודם ADSL

איך חתיכת פלסטיק מפצלת כבל נחושת אחד לשניים? ואם היא מפצלת, למה קוראים לה פילטר (מסנן)? מה יקרה אם נחבר טלפון ללא פילטר לשקע אחד, ומודם ADSL לשקע שני?

התשובה הפשוטה לשאלה האחרונה היא – כשנחבר טלפון ומודם ללא פילטר, ברגע הראשון לא יקרה כלום. אך ברגע שנרים את הטלפון לתחילת שיחה, ניצור התנגשויות והפרעות בין שני ערוצי המידע שעוברים על כבל הנחושת הדקיק: ערוץ הקול וערוץ ה-data. כדי להבין מדוע זה קורה, נדרש קודם כל להבין קצת ברשת הטלפוניה הביתית.

איך עובד טלפון?

טלפון הוא דוגמה מצוינת לפרק שלנו, מכיוון שהוא אפליקציה לשיחה אנושית שמשתמשת ישירות בשכבה הפיזית, ללא אף שכבה מתווכת.

מכשיר הטלפון ממיר גל קול לזרם חשמלי ולהיפך, באמצעות מיקרופון[72] ורמקול. התהליך כולל את השלבים הבאים:

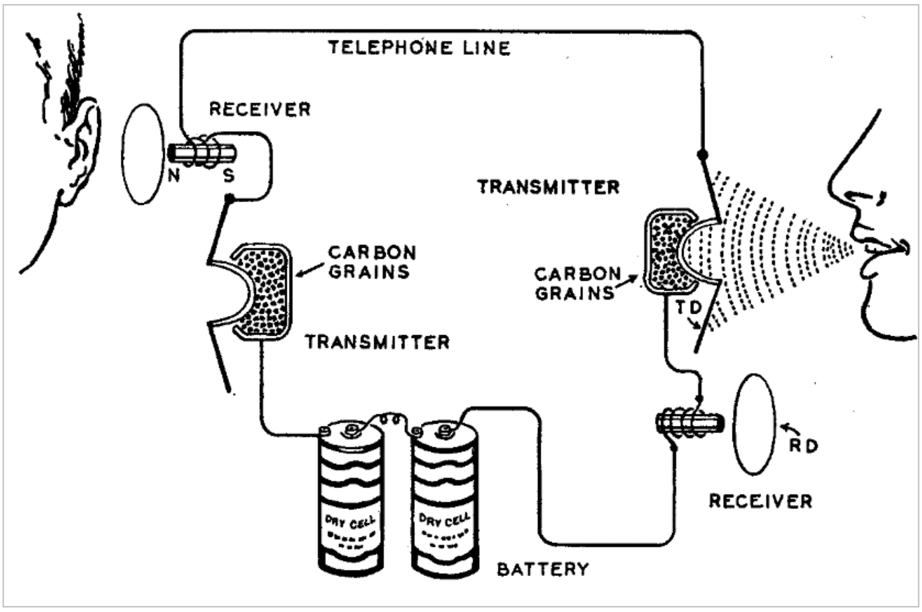
- כאשר אנו מדברים אל תוך שפופרת הטלפון, אנו מייצרים גל קול שנע באוויר.

---

[71] לדייקנים שבינינו, כבלי הנחושת תמיד באים בזוגות. קצת כמו חיבורי '+' ו-'-'.

[72] הידעת? מיקרופון בטלפון עשוי מגרגרי פחם: http://en.wikipedia.org/wiki/Carbon_microphone.

- גל הקול פוגע ומרעיד את משטח המיקרופון בשפופרת, בעוצמה ובתדירות משתנות, על פי טון הדיבור ותוכנו.
- המיקרופון בשפופרת הטלפון מייצר זרם חשמלי שעוצמתו ותדירותו תואמות את העוצמה והתדירות של גל הקול.[73]
- הזרם החשמלי מועבר לטלפון בצד השני, דרך מעגל חשמלי פשוט.
- בטלפון השני, עוצמת הזרם מתורגמת באמצעות רמקול[74] חזרה לגלי קול אותם ניתן לשמוע.



על מנת שהטלפון יעבוד, נדרש לסגור מעגל חשמלי בין שני מכשירי הטלפון, ולכן כבלי הטלפון הם כבלי זוגות, כבלים שהם זוג כבלי נחושת דקים, מלופפים[75] אחד סביב השני. כאשר מרימים את הטלפון בשני צדי הכבל, נסגר המעגל שמאפשר העברת זרם חשמלי שמקודד את גל הקול. בשרטוט לעיל ניתן לראות אדם מדבר אל מיקרופון, קולו מתורגם לזרם חשמלי שעובר אל רמקול המשחזר את גל הקול באוזנו של השומע. כמובן, כל המעגל דורש חשמל ולכן יש בו גם סוללות.

חשוב להבין – הטלפון פועל בשכבה הפיזית בלבד:

- הזרם החשמלי מעביר את המידע – עוצמת גל הקול ללא המרה למידע אחר (כגון 0 ו-1).
- קצב העברת התקשורת תלוי בדוברים, כמו גם סנכרון הדיבור (״הלו?״, ״היי!״, ״ביי!״).
- הטלפון הינו דו כיווני – שני הדוברים יכולים לדבר בו זמנית.

---

[73] אם נצייר גרף של לחץ האוויר במיקרופון כפונקציה של הזמן, ונשווה אותו לגרף של עוצמת הזרם בכבל הנחושת כפונקציה של הזמן, נקבל גרף שנראה מאוד דומה.

[74] רמקול הוא בעצם מגנט עטוף בסליל ומחובר למשטח.

[75] מטרת הליפוף היא לבטל השפעות של השראה אלקטרו-מגנטית. קראו עוד על כך: http://goo.gl/tc1Tae.

איך עובד מודם?

מודם (Modem) – קיצור (באנגלית) של Modulator & Demodulator – מכשיר שמאפנן ומשחזר ביטים על גבי ערוץ תקשורת[76].

המודם הוא הצעד הראשון בתהליך מהפכת המידע שמתחוללת בשני העשורים האחרונים. עד המודם, אמצעי התקשורת העבירו בעיקר גלי קול (למשל טלפון או רדיו). בצורה ״גלית״ שכזו, קשה מאוד להעביר מידע שאינו נראה כמו גל. המודם איפשר להעביר יחידת מידע הרבה יותר בסיסית: הביט. העברת ביטים למרחקים, בקצב ובמהירות גבוהים, פתחה מגוון רחב של אפשרויות להעברת מידע מכל סוג, כל עוד אפשר לקודד אותו בביטים. כנראה שכל פיסת מידע שתוכלו לחשוב עליה אפשר לקודד בביטים, אבל עניננו כעת אינו בקידוד אלא באופן בו הביטים עוברים ממודם אחד אל מודם אחר.

המודמים הראשונים העבירו ביטים בקצב מאוד איטי על גבי כבלי נחושת. תכונה חשובה של כבלי הזוגות היא שהם מעבירים בצורה טובה תדרים של קול אנושי למרחקים ארוכים. כדאי שנזכר שה**תדר** של גל הוא כמות המחזורים שהגל מבצע בשנייה. תדר נמדד ב-Herz (או בקיצור Hz), כאשר 1Hz הוא תדר של פעם בשנייה. התדר של גלי קול אנושיים נע בין 50Hz ל-20KHz [77].

המודם מנצל תכונה זו של כבלי הזוגות, ומשתמש בתדרי זרם חשמלי כדי לקודד ביטים. המודם הראשון קודד 0 ו-1 על פי הטבלה הבאה:

ביט	תדר צד א׳	תדר צד ב׳
0	1070Hz	2070Hz
1	1270Hz	2270Hz

מאחר שהמודם לא מדבר עם עצמו, וכדי לאפשר תקשורת דו כיוונית, המודם בצד השני קודד 0 ו-1 באמצעות תדרים שונים.

---

[76] כדאי להפריד בין מודם לראוטר הביתי. הראוטר הביתי משלב בתוכו שלושה או ארבעה רכיבים: מודם שמתקשר על גבי קו טלפון/כבלים, מתג (Switch) שמאפשר חיבור קווי של מחשבים ב-LAN, (לעיתים) Access Point שמאפשר חיבור אל-חוטי של מחשבים ב-LAN, ונתב (Router) שתפקידו לנתב את המידע בין המודם ל-Switch ול- Access Point.

[77] או 20,000 HZ, ה-K הוא קיצור ל-Kilo.

לו הייתם מאזינים לקו הטלפון, הייתם שומעים רחשים בתדרים האלו, אך מאחר וקצב התקשורת של המודמים הראשונים היה 300 ביטים לשנייה (BPS – Bits Per Second), שזה קצב מהיר מאוד עבור האוזן האנושית (אך איטי מאוד למחשב), הייתם שומעים[78] רחש קבוע.

איך מודמים התקדמו למהירויות של כ-100Mbps המקובלות בביתנו כיום? זה שיפור של פי מיליון!

השיפור בקצבי התקשורת נעוץ במספר דרכים לניצול יעיל יותר של חוט הנחושת:

- הגדלת כמות התדרים שכל צד יודע לשדר בהם, ובכך להכפיל את כמות הביטים שניתן לשדר. ראה לדוגמה את הטבלה הבאה:

ביט	תדר צד א׳	תדר צד ב׳
00	1070Hz	2070Hz
01	1170Hz	2170Hz
10	1270Hz	2270Hz
11	1370Hz	2370Hz

- הגברת קצב שידור הביטים, דהיינו קצב ההחלפה בין תדרים. במקום 300 חילופים בשנייה ל-300bps, אפשר לבצע 600 חילופי תדר בשנייה, ולהגדיל את קצב השידור ל-600bps.
- ביטול ההד הוא שיטה נוספת לניצול יעיל יותר של כבל הנחושת. לרוב אנחנו לא חושבים על כך, אבל כשאנו מדברים בטלפון, אנחנו שומעים את עצמנו (או את ההד שלנו). אם לא היינו שומעים את עצמנו, הייתה לנו תחושה כאילו הקו מנותק. תכונה זו של קו הטלפון מפריעה למודם מאחר שהביטים שהוא שולח מתערבבים עם הביטים שהוא מקבל. בלי ביטול ההד, שני המודמים בשיחה נדרשים להשתמש בתדרים שונים (כפי שראינו בטבלאות לעיל). כאשר המודם מבודד את התדרים שהוא שולח מהתדרים שהוא מקבל (באמצעות ביטול ה"הד"), שני הצדדים יכולים להשתמש באותם תדרים, ובכך מגדילים את סך כל כמות התדרים שאפשר להשתמש בהם.
- סינון רעשים ותיקון שגיאות – רעשים בקו מייצרים שגיאות תקשורת (שגיאה היא כאשר צד אחד שולח ביט 0, והצד השני מקבל ביט 1, או להיפך). ללא תיקון שגיאות, עם כל שגיאה נדרש להעביר את כל המסר מחדש. סינון רעשים ותיקון שגיאות נעשה באמצעות שיטות מתקדמות אשר מסתמכות על מתמטיקה מורכבת ועיבוד אותות מתקדם.

[78] הקשיבו כיצד נשמעו המודמים הראשונים בקישור הבא: https://www.youtube.com/watch?v=3l2Q5-15Mic

המודם הקלאסי המהיר ביותר הגיע למהירות של 56Kbps, אז איך מודם ה-ADSL המודרני מהיר פי 10,000 (100Mbps)?

הזכרנו תחילה שכבלי זוגות מעבירים תדרי קול אנושי (50Hz עד 20KHz) למרחקים ארוכים. קפיצת המדרגה האמיתית נעשתה כאשר חברות הטלפוניה החליטו ״לקצר טווחים״, ולקרב את המרכזיה אל בתי הלקוחות. קרבה זו אפשרה למודמים לנצל תדרים גבוהים בהרבה מ-20KHz, שמאפשרים קצבי העברת מידע גבוהים מאוד, וזאת מבלי לסבול מחסרון האורך המגביל של כבלי הזוגות. כך התפתח מודם ה-ADSL, שהגעתו לכל רחבי המדינה נעשתה בהדרגתיות עקב תהליך התקנת המרכזיות[79] החדשות קרוב לבתים.

אנלוגי, דיגיטלי ומה שביניהם

למדנו עד כה על הטלפון ועל המודם – שני אמצעים טכנולוגיים שמשמשים אותנו עד היום.

למדנו שהטלפון מעביר באופן ישיר את גל הקול לגל של זרם חשמלי, ולכן הוא מכשיר אנלוגי.
למדנו שהמודם לוקח מידע בינארי, ומקודד אותו באמצעות תדרים קבועים עבור 0 ועבור 1. לכן המודם הוא מכשיר דיגיטלי.

בקידוד **אנלוגי**, טווח ערכי המספרים שאפשר להעביר הוא רציף. שעון אנלוגי הוא שעון אשר מודד את הזמן באמצעות קפיץ מתוח שמשתחרר בזמן מדוד ועקבי. שחרור המתח בקפיץ מתבצע בזמן באופן רציף, ומה שמפריד בין שנייה לשנייה הן השנתות על השעון, דרכן עובר המחוג.

בקידוד **דיגיטלי**, טווח ערכי המספרים שאפשר להעביר הוא בדיד וסופי. שעון דיגיטלי מתבסס על קריסטל שמשחרר זרם בפרקי זמן קבועים ומדודים. השעון הדיגיטלי סופר כפולות של הזרמים הבדידים, ולכן מקדם את השעה בפרקי זמן קבועים ובדידים (לדוגמה נאנו שנייה).

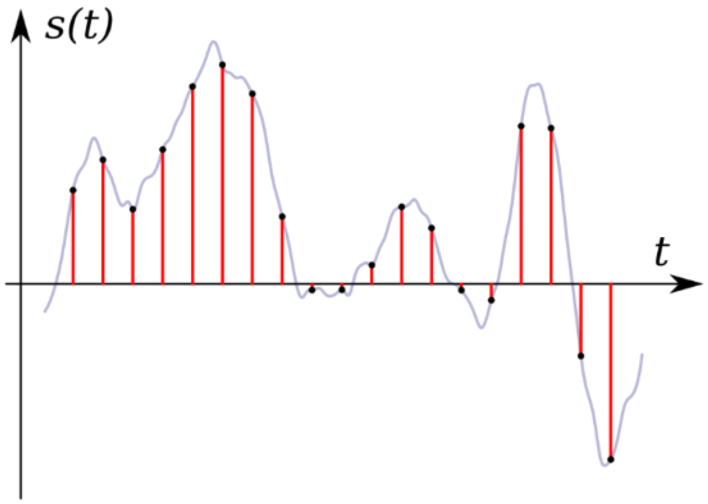
דוגמה שממחישה היטב את ההבדל בין אנלוגי לדיגיטלי היא ההבדל בין תקליטים לדיסקים.

בתקליט, גל הקול שמייצג את המוסיקה, נחרט על התקליט באותה צורה בה הוא מופיע במציאות. אילו היינו לוקחים את התקליט, ומתבוננים מקרוב בחריטה המעגלית שעליו, היינו מקבלים תרשים של גל הקול. המחט בפטיפון עוברת מעל החריטות בצורת גל הקול שבתקליט, ומשחזרת את הצליל המקורי. אפשר לדמיין

---

[79] מרכזיות אלו נקראות מרכזיות DSLAM –
http://en.wikipedia.org/wiki/Digital_subscriber_line_access_multiplexer

שהתמונה הבאה מייצגת תרשים של גל הקול (הקו הסגול) כפי שהוא עבר באוויר. החריטה בתקליט (אם נתבונן בה "מהצד") תראה בדיוק אותו דבר[80].



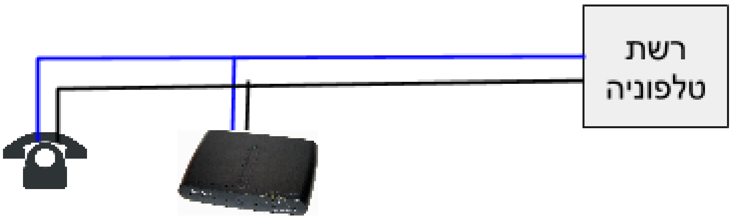
לעומת זאת, *בדיסק* (CD) המוסיקה מקודדת בביטים. משמעות הדבר היא שאחרי הקלטת המוסיקה, נדרש לקודד את גל הקול למספרים, להמיר אותם לייצוג בינארי, ולאחר מכן לצרוב את הביטים על הדיסק. קידוד מוסיקה לביטים הינו נושא נרחב, אך אם נתבונן שוב בתרשים שלעיל, נוכל להבין במעט כיצד זה קורה. מטרתנו היא להמיר את הקו הסגול, שמתאר את גל הקול, לרצף מספרים שניתן יהיה לרשום בביטים. כדי לעשות זאת, נבחר רצף נקודות בהן נדגום (או נמדוד) את עוצמת הגל ונרשום לכל דגימה את עוצמת הגל שנמדדה. אפשר לראות תהליך זה בתרשים לעיל, לפי הנקודות השחורות שהן הדגימות, והקווים האדומים שמראים את העוצמה הנמדדת בכל דגימה. לאחר שרשמנו את כל המדידות ברצף ביטים על גבי CD, נוכל לשחזר את הגל המקורי בפעולה ש"תחבר את הנקודות" מחדש. כמובן שנדגום את גל הקול בקצב יותר גבוה (קרי, ככל שנמדוד את הגל בנקודות יותר צפופות), יהיה יותר קל לשחזר את הגל המקורי.

הסיבה שקובץ קול (או וידאו) בקידוד 192kbps הוא באיכות טובה יותר מאשר קובץ בקידוד 128kbps, נעוצה ביכולת להעביר יותר נקודות של גל הקול, ובכך לחדד את הרכבת גל הקול המקורי בעת ניגון הקובץ.

מה קורה כשמחברים טלפון לשקע אחד ומודם ADSL לשקע נוסף?

80 ניתן לקרוא עוד על תקליטים ופטפונים בקישור: http://en.wikipedia.org/wiki/Gramophone_record

כל שקעי הטלפון המחוברים לאותו קו בעצם מחוברים לאותו מעגל חשמלי. אפשר לראות זאת בשרטוט הבא:



כאשר שני המכשירים מחוברים לאותו מעגל, הזרמים החשמליים שהם מייצרים מגיעים לשניהם, ולכן יש סיכוי שהם יפריעו אחד לשני.

עכשיו אפשר להבין למה צריך את הפילטר בשקע הטלפון ומה בדיוק הוא מסנן!

תפקידו של הפילטר הוא להפריד ולסנן תדרים.

- הטלפון והמודם מחוברים שניהם לאותו קו, כלומר לאותו מעגל חשמלי.
- הטלפון יכול לקבל ולהעביר רק תדרים שהאוזן האנושית שומעת ושהקול האנושי מייצר (מ-50Hz ועד 20KHz).
- מודם ה-ADSL מקבל ומעביר רק תדרים גבוהים יותר.
- הפילטר משתמש ברכיבים אלקטרוניים על מנת לסנן עבור הטלפון רק גלים שהם בתווך הנשמע לאוזן האנושית.
- הפילטר מסנן עבור מודם ה-ADSL רק גלים בתווך התדרים שלו.

הפילטר נדרש כדי למנוע הפרעות של המכשירים אחד לשני. כל עוד יש הפרדה מלאה בתדרים, אין בעיה. אם מחברים טלפון לשקע ללא פילטר, הטלפון יכול להכניס לקו תדרים גבוהים יותר מאשר 20KHz (לא נדע, כי האוזן לא תשמע אותם), ותדרים אלו יפריעו לסנכרון העדין בין מודם ה-ADSL לבין המרכזיה.

סיכום ביניים

עד כה למדנו על עמודי התווך ההיסטוריים של השכבה הפיזית:

- תקשורת מבוססת אור (מדורות, מורס באמצעות פנסים, שלט רחוק באינפרה אדום).
- תקשורת מבוססת גלים אלקטרומגנטיים באוויר.
- תקשורת מבוססת זרמים חשמליים בכבלי נחושת.

כמו כן, למדנו מה הוא גל ומה הן תכונותיו הייחודיות (אורך הגל, זמן המחזור, התדירות והמשרעת).

בנוסף, הכרנו מספר מושגים חשובים:

- סיבית / ביט.
- תווך.
- קידוד.
- אפנון.
- אנלוגי ודיגיטלי.

כדי להבין את המושגים הנ״ל, חקרנו מספר אמצעי תקשורת הקיימים כמעט בכל בית ועמדנו על האופן בו הם פועלים. הטבלה הבאה מסכמת את הרכיבים הללו:

אמצעי תקשורת	תווך	קידוד	פרוטוקול
רדיו	גלים אלקטרומגנטיים	אנלוגי	AM, FM
טלוויזיה	גלים אלקטרומגנטיים כבל קואקסיאלי (coax)	אנלוגי או דיגיטלי	DVB-T, RF RF, DOCSIS
טלפון	כבל זוגות גלים אלקטרומגנטיים	אנלוגי דיגיטלי	אין DECT
פקס	כבל זוגות	דיגיטלי	V.34
מודם ADSL	כבל זוגות	דיגיטלי	ADSL
נקודת גישה אלחוטית	גלים אלקטרומגנטיים	דיגיטלי	802.11

כעת, נמשיך ונלמד על מימושים ושימושים של השכבה הפיזית בחיי היום יום – החל מהרשת המשרדית, דרך משרדים גדולים וכלה בספקיות תקשורת.

הרשת המשרדית

עד כה חקרנו את מגוון רשתות התקשורת הנמצאות בביתנו. למרות שהן רבות, הן אינן מייצגות את מרבית פתרונות התקשורת הקיימים בעולם. כדי להבין היכן עוברת מרבית מתקשורת הנתונים העולמית, עלינו להסתכל במקום בו רוב בני אדם מבלים חלק ניכר מזמנם: המשרד. המשרד מהווה סביבת מידע שונה מאוד מהרשת הביתית:

- הרשת המשרדית לרוב משרתת כמות גדולה יותר של אנשים:
  - הרשת פרושה על שטח יותר גדול.
  - כמויות המידע וקצבי התקשורת גבוהים יותר.
- במשרד יש צורך בשירותי אינטראנט (Intranet – אינטרנט פנימי), בנוסף לשירותי אינטרנט.
- המידע הנמצא ברשת המשרדית לרוב רגיש יותר (סודות עסקיים, כספים, וכו׳) מהמידע ברשת הביתית.

כדי להבין מעט מפתרונות התקשורת המסחריים ננסה ללמוד על חיבורי הרשת במשרדים בגודל שונה.

משרד קטן

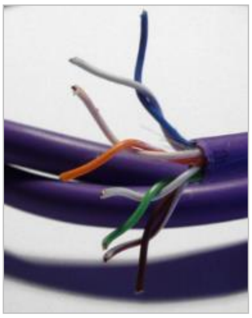
בואו נדמיין שאנו מנהלים חברה לפיתוח יחסי ציבור באינטרנט. בחברה חמישה עשר עובדים שמשקיעים את מרבית זמנם בפעילות תקשורתית ברשתות חברתיות. החברה מספקת לכל עובד שולחן כתיבה, מחשב שולחני, מסך גדול ומקלדת. כמו כן, בחברה יש שרת קבצים המאחסן תיקיית קבצים משותפת לכל החברה, ובה מסמכי אסטרטגיה ותיעוד של כל פרויקטי יחסי הציבור.

מה הדרך הנכונה לחבר את כל מחשבי החברה לרשת האינטרנט?

17 מחשבים (שרת + 15 עובדים + מחשב מנכ״ל) שכולם נמצאים במשרד אחד מהווים מקרה די פשוט עבור רשת מקומית מבוססת switch. ניזכר ש-Switch הוא רכיב שמחבר מחשבים בשכבת הקו, והוא מעביר מסגרות מידע על פי כתובת MAC. על שכבת הקו ועל פרוטוקול Ethernet הרחבנו בפרק הרלוונטי. בפרק זה נתמקד בשכבה הפיזית בה עושה שימוש כמעט כל רשת מקומית בימינו.

השכבה הפיזית שמתחת לפרוטוקול Ethernet היא כבל הרשת הסטנדרטי. תשמחו לדעת שכבל זה נקרא כבל CAT5, החיבור בקצה שלו נקרא RJ-45, והתקן שמגדיר את השימוש בכבל נקרא Base-T10. הרבה מושגים לכבל כה פשוט – כעת נגדיר אותם בצורה מסודרת:

כבל CAT5 – כבל שמאגד בתוכו 4 כבלי זוגות, כל זוג בצבע שונה. אפשר לראות את כבלי הזוגות המלופפים סביב עצמם בתמונה הבאה. אם לדייק, ישנם מספר סוגי כבלים כאלו: CAT3, CAT5, CAT5e, וגם CAT6. מבחוץ כולם נראים אותו הדבר, אך מבפנים הם נבדלים באיכות בידוד ההפרעות החשמליות. איכות הבידוד משפיעה על קצב העברת הביטים. כשאתם הולכים לחנות לקנות כבל רשת, חשוב שתרכשו כבל שמתאים למהירות הגלישה שלכם[81]. הכתבה הבאה מאת רן בר זיק, עושה סדר בנושא הכבל המתאים לכם: https://www.haaretz.co.il/captain/tutorial/.premium-1.9156625



כבל CAT5

חיבור RJ-45 – השקע והתקע של כבלי הרשת הסטנדרטיים, כולל צורתם וסידור הכבלים הפנימיים לפי צבע, מוגדר בתקן שנקרא RJ – Registered Jack [82]. עבור כבלי רשת Ethernet, התקן הוא RJ-45 [83], אך ישנם תקנים דומים גם עבור כבלים אחרים כגון כבל הטלפון (RJ-11).

חיבור RJ-45

בטרם נדבר על התקן Base-T10, נכיר מונח נוסף:

---

81 ניתן לקרוא עוד על כבלי רשת בקישור: http://en.wikipedia.org/wiki/Category_5_cable

82 ניתן לקרוא עוד על תקן RJ בקישור: http://en.wikipedia.org/wiki/Registered_jack

83 יש לציין שחיבור זה נקרא גם חיבור P8C8, ובמקומות אלו בהם כך הוא נקרא, הכוונה היא לאותו סוג חיבור כמו RJ-45.

Duplex – דופלקס – מאפיין של מערכות תקשורת דו כיווניות בין שתי נקודות. מערכת שהיא Half Duplex מאפשרת לשני צדדים לתקשר אחד עם השני באופן דו כיווני אך לא סימולטני. דוגמה למערכת Half Duplex היא ווקי-טוקי (מכשיר קשר אלחוטי), בו רק צד אחד יכול לדבר בזמן שהצד השני מקשיב. כששני הצדדים מנסים לדבר, אף אחד לא שומע את השני. מערכת שהיא Full Duplex מאפשרת לשני צדדים לתקשר אחד עם השני באופן מלא וסימולטני, כלומר שני הצדדים יכולים לדבר באותו הזמן. הטלפון הוא דוגמה למערכת Full Duplex, מאחר שהיא מאפשרת לשני דוברים לדבר בו זמנית וגם לשמוע אחד את השני.

תקן Base-T10 – תקן זה מגדיר כיצד משתמשים בכבלי CAT5 וחיבורי RJ-45 כדי להעביר ביט בודד על גבי הכבל. ראשית, התקן מגדיר שנדרשים רק שני כבלי זוגות (ארבעה כבלי נחושת בסך הכל), והוא מגדיר גם בדיוק באילו מכבלי הזוגות להשתמש (ראו תמונה). התקן גם מגדיר כיצד להעביר ביט בודד (תדרים[84] וקידוד[85]), כיצד קובעים את קצב התקשורת, והאם התקשורת היא Half Duplex או Full Duplex. תקן Base-T10 הוא רק אחד ממשפחת תקנים הנקראים Ethernet Over Twisted Pair (Twisted Pair הינו המונח האנגלי לכבל זוגות).

Pin	Pair	Color	telephone	10BASE-T
1	3	white/green		TX+
2	3	green		TX−
3	2	white/orange		RX+
4	1	blue	ring	
5	1	white/blue	tip	
6	2	orange		RX−
7	4	white/brown		
8	4	brown		

טבלה[86] המפרטת את השימוש של כל תת-כבל בתוך כבל CAT5

(TX – Transmit, RX – Receive)

[84] ניתן להרחיב על התדרים בהם נעשה שימוש בכבלי רשת: http://en.wikipedia.org/wiki/Baseband

[85] כדי להבין קידוד ביטים בכבלי רשת, קראו עוד על קוד מנצ׳סטר: http://en.wikipedia.org/wiki/Manchester_code

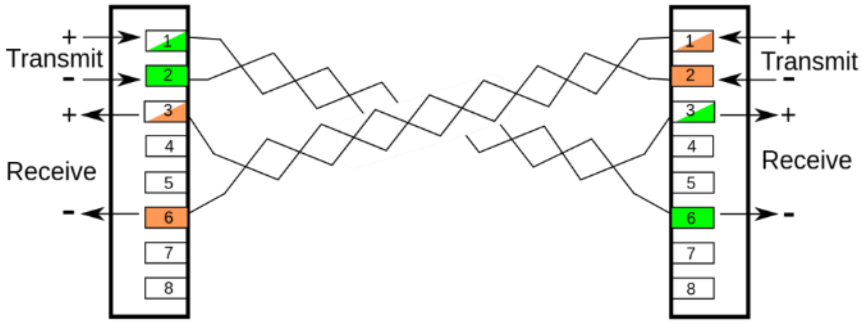
[86] התרשים המקורי: http://en.wikipedia.org/wiki/Ethernet_physical_layer

ייתכן ששמעתם פעם את המושג "כבל מוצלב", כאשר ניסיתם לחבר שני מחשבים ישירות אחד לשני עם כבל רשת. בכרטיסי רשת ישנים יותר, לו היינו מחברים שני מחשבים עם כבל רשת רגיל, הם לא היו מצליחים לתקשר.

התבוננו בטבלת פירוט תתי הכבלים בכבל CAT5. האם תוכלו לחשוב על סיבה מדוע חיבור ישיר של שני מחשבים לא יעבוד?

המושג "כבל מוצלב" מרמז על התשובה. שימו לב שהכבל הירוק מסומן כקבל "שידור" (TX – Transmit), והכבל הכתום מסמן "קליטה" (RX – Receive). אם נחבר שני מחשבים לכבל אחד, הם ינסו לשדר אותות על אותו כבל, ובכבל הקליטה לא יעבור אף מידע. כאשר מחברים כבל בין מחשב ל-Switch, ה-Switch קורא מידע מהכבל הירוק וכותב מידע אל הכבל הכתום. מכאן אפשר לנחש ש"כבל מוצלב" פשוט מצליב בין הכבל הירוק לכתום: בצד אחד של הכבל, החיבור יהיה כפי שהוא בטבלה, ובצד השני, הכבל הירוק יחובר לפינים של הכבל הכתום (פינים 3 ו-6) ולהיפך. יש לציין שכיום רוב כרטיסי הרשת תומכים בזיהוי אוטומטי של כבלי השליחה והקבלה, ולכן כמעט ואין יותר צורך בכבלים מוצלבים.

כבל רשת מוצלב (Ethernet Crossover Cable) – הינו כבל רשת בו כבלי הזוגות של השליחה וקבלה הוצלבו, דבר המאפשר לחבר שני מחשבים ישירות אחד לשני, ללא Switch ביניהם. ניתן לראות את ההצלבה בין הכבלים בשרטוט שלהלן:



משרד גדול

משרד יכול להתנהל עם אוסף Switchים ועם מספר Routerים כל עוד המרחק הפיזי בין רכיבי תקשורת קטן, ואפשר למשוך כבל רשת בין כל שני רכיבים. אבל מה אפשר לעשות כאשר החברה שלנו גדלה ומתפרשת על פני מספר בניינים? או במקרה יותר מורכב, החברה שלנו היא בעצם מפעל שפרוש על שטח די גדול, ובתוכו נדרש להעביר תקשורת מקצה אחד לקצה השני? חברה פרטית, שאינה חברת תקשורת כמו בזק, או הוט, אינה יכולה להרשות לעצמה למשוך כבלי תקשורת למרחקים ארוכים כי מדובר בתהליך יקר וממושך. במקרים כאלו, נדרש פתרון אחר שמחזיר אותנו לתקשורת האלחוטית.

בחצי הראשון של פרק זה פגשנו בתקשורת אלחוטית בשלט הרחוק. בנספח א' של פרק זה, אתם מוזמנים להרחיב על הרשת האלחוטית הביתית. טכנולוגיות אלו של תקשורת אלחוטית סובלות ממרחק תקשורת מוגבל ומקצב יחסית נמוך. כדי לחבר שני בניינים בהם עשרות או מאות מחשבים, נדרש פתרון תקשורת שיעבור מרחק של מאות מטרים ועד קילומטרים בודדים, ויעביר קצבים גבוהים של מידע (כאלו שעומדים בקצבי רשת Gigabit Ethernet). תקשורת מיקרוגל היא הפתרון שעונה לצרכים האלו.

תקשורת מיקרוגל (Microwave Transmission) – העברת מידע באמצעות גלים אלקטרומגנטיים בטווח אורכי גל שניתן למדוד בסנטימטרים. כפי שלמדנו קודם, אורך הגל ותדירותו קשורים ביחס הפוך, ולכן ניתן להסיק שגלי מיקרוגל הם גלים בטווח התדרים בין 1GHz ל-30GHz.

למי שאינו מבין מספיק בפיזיקה או הנדסת חשמל, גלי מיקרוגל יכולים להישמע כמו טווח תדרים מאוד שרירותי: למה דווקא גלים בתדר בין 1GHz ל-30GHz עונים לנו על הבעיה? יש לכך כמה סיבות מאוד מדויקות:

- בגלל אורך הגל (קטן אך לא קטן מדי), קל לבנות אנטנות כיווניות – אנטנות שמרכזות את הגלים האלקטרומגנטיים בכיוון אחד (ראו תמונה לעיל).
- גלי מיקרוגל עוברים את האטמוספירה ללא הפרעות משמעותיות.
- התדירות של גלי המיקרוגל מאפשרת לאפנן עליהם ביטים בקצב גבוה.

תכונות אלו של הגלים האלטרומגנטיים מאפשרות לבנות ציוד שידור וקליטה מאוד יעיל. האנטנות הכיווניות מאפשרות לחסוך אנרגיה מצד אחד, ומצד שני מונעות הפרעות בין ערוץ תקשורת מיקרוגל אחד לשני. חסכון האנרגיה מתאפשר בגלל ריכוז אלומת הגלים האלקטרומגנטיים. באנלוגיה לאור נראה (שכזכור, הוא גם גל אלקטרומגנטי), פנס ממוקד מאיר למרחק גדול יותר מאשר נורה "עגולה". באותה אנלוגיה, אפשר גם להבין מדוע ערוצי תקשורת מיקרוגל לא מפריעים אחד לשני: שני פנסים ממוקדים יכולים להאיר שתי אלומות באותו חדר מבלי שהאלומות יפריעו אחת לשנייה, ולעומתם האור משתי נורות יתערבב ויהיה קשה להפריד בין

מקורות האור. החסרון של גלי מיקרוגל הוא שנדרש קו ישיר ונקי ממחסומים בין האנטנה המשדרת לאנטנה הקולטת[87].

לסיכום, לגלי מיקרוגל יש תכונות מיוחדות המסייעות בהעברת מידע בקצבים גבוהים ולמרחקים ארוכים (כל עוד הם "ישרים" וללא מחסומים). שיטת אפנון הביטים בגלי מיקרוגל דומה לשיטה שלמדנו בראשית הפרק, המתבססת על שינוי תדר.

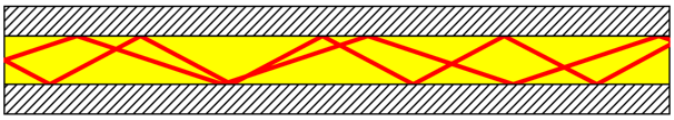
בתמונה לעיל מצולמת אנטנת "תוף", המשמשת לתקשורת מיקרוגל. אפשר למצוא אותה לרוב על עמודי אנטנות סלולריות, ולא בגלל שהיא מתקשרת עם מכשירים סלולריים, אלא בגלל שהיא מאפשרת תקשורת בין אנטנות לבין מרכזי התקשורת של חברות הסלולר.

ספק תקשורת

ראינו שמשרד קטן עד בינוני יכול להסתפק באוסף Switchים ו-Routerים, ושחברה יותר גדולה יכולה להיעזר בתקשורת מיקרוגל על מנת לחבר משרדים או אתרים מרוחקים. עם זאת, התעלמנו עד כה מחלק גדול מאוד בשרשרת התקשורת שבה אנו משתמשים כל יום: ספקי התקשורת. ספקי התקשורת כוללים גם את ספקי התשתית (בזק, הוט, חברות הסלולר) וגם את ספקי השירות (ISPים[88] למיניהם). חברות אלו מעבירות כמויות בלתי נתפסות של ביטים בכל שנייה, ולמרחקים בינלאומיים ארוכים מאוד. הפתרון היחיד להעברת תקשורת בקצבים גבוהים מאוד למרחקים ארוכים מאוד הינו הסיב האופטי.

סיב אופטי (Optical Fiber) – הינו סיב עשוי זכוכית או פלסטיק, המאפשר העברת אור בתוכו למרחקים ארוכים עם אובדן מינימלי של עוצמה.

הייחוד של הסיב האופטי, כפי שניתן לראות בתמונה, נעוץ בכך שהאור "כלוא" בתוך הסיב, ולא יכול להתפזר ולאבד מעוצמתו. אור שנכנס בקצה אחד של הסיב האופטי, נע לאורכו ומוחזר פנימה מדפנות הסיב (כמו ממראה).



[87] אם תחשבו לרגע, גלי רדיו רגילים עוברים מרחקים ארוכים בהרבה, עוקפים הרים וגבעות ונכנסים לבית דרך הקירות. אין זה כך בתדרי המיקרוגל.

[88] ISP – Internet Service Provider.

ישנם יתרונות עצומים לסיב האופטי:

- נדרשת כמות מעטה מאוד של אנרגיה בשביל להעביר אור דרך הסיב: הבהוב קל של נורת LED בצד אחד מאפשר להעביר את האור למרחק של עשרות קילומטרים (לשם השוואה, דמיינו כמה רחוק אתם יכולים להאיר עם פנס ה-LED החזק ביותר שלכם).
- מידע לא נכנס ולא יוצא מהסיב – אין הפרעות בתקשורת ולכן ניתן להגיע לקצבים גבוהים מאוד של תקשורת. לשם השוואה, קצבי המידע אליהם ניתן להגיע באמצעות סיבים אופטיים מגיעים ל-100Terabit/s, למרחק של מעל 100 ק״מ. בקצב זה אפשר להעביר 12.5 הארד-דיסקים של 1TB בשנייה!

אפנון ביט על גבי גל אור בתוך סיב אופטי נעשה בצורה פשוטה, שדומה יותר להבהוב – אור חזק מייצג 1, אור חלש מייצג 0. כמו כן, ישנן מספר דרכים יצירתיות להגברת קצב שידור הביטים. דרך אחת היא כמובן להגביר את קצב ההבהוב, ולהעביר באופן ישיר יותר ביטים בשנייה. דרך אחרת היא להעביר מידע במספר ״צבעים״ במקביל. ״צבעים״ שונים של אור הם בעצם תדרים שונים של גל האור. אם נשתמש במספר LEDים בצבעים שונים ובמספר גלאי אור (שרגישים לצבע יחיד), נוכל להכפיל את קצב שידור המידע. דרך שלישית היא פשוט להעביר מספר סיבים אופטיים ביחד: אם כבר מושכים כבל למרחק, אפשר לקבץ עשרות סיבים אופטיים ביחד ולפרוש אותם בפעם אחת.

שאלה נוספת שיכולה לעלות לגבי סיבים אופטיים למרחקים ארוכים היא כיצד מושכים סיב למרחק גדול מ-100 ק״מ? התשובה לכך פשוטה, והיא דומה כמעט לכל אמצעי התקשורת: ממסרים. **ממסר (Relay)** הוא רכיב שמקבל אות תקשורתי, מגביר אותו ומשדר אותו הלאה. ממסר אור מקבל את האותות מסיב אופטי אחד, ומייצר אותם מחדש בסיב האופטי השני. למקרה שתהיתם, מניחים ממסרים גם על קרקעית הים, בשביל למשוך כבלים תת-ימיים.

לסיכום, סיבים אופטיים הם תווך יעיל ביותר להעברת מידע, והם אחראים להעביר את רוב תקשורת הנתונים בעולם. אמנם השתמשנו בספקי תקשורת כצרכנים עיקריים של סיבים אופטיים, אבל חשוב לציין שסיבים אופטיים נמצאים בשימוש נרחב גם ברשתות משרדיות בינוניות וגדולות, על מנת לחבר בין נתבים שמעבירים קצבים גדולים של מידע.

השכבה הפיזית – סיכום

בפרק זה למדנו על הדרכים השונות בהן אפשר להעביר את יחידת המידע הבסיסית: הביט.

למדנו על תקשורת קווית (טלגרף, טלפון, Ethernet, סיבים אופטיים) וגם על תקשורת אלחוטית (שלט רחוק ותקשורת מיקרוגל), ועל האופן בו סוגי התקשורת השונים משתלבים בחיינו. כמו כן, למדנו מושגים בסיסיים שמאפשרים לנו להבין את המאפיינים של שיטות התקשורת השונות ולהשוות ביניהן.

כשמדובר בתקשורת קווית או אלחוטית, מדובר ב**תווך התקשורת,** החומר על גביו ניתן להעביר ביטים (נחושת, אוויר, אור ועוד). האופן בו מגדירים את האות המסמן 0 ואת האות המסמן 1 נקרא **קידוד.** לתווכים שונים ושיטות קידוד שונות מוכתב **קצב** אחר. הקצב של ערוץ תקשורת נקבע על פי הכמות המקסימלית של ביטים שניתן להעביר על גבי הערוץ בשנייה אחת. אך לא רק הקצב משתנה משיטת תקשורת אחת לשנייה, אלא גם **מרחק השידור:** כמה רחוק אפשר להעביר את הביטים. מאפיין נוסף של ערוצי תקשורת הוא אופן **הסנכרון:** האם שני הצדדים המתקשרים יכולים לשדר בו זמנית, ואם כן אז איך דואגים שביטים יועברו בשני הכיוונים מבלי להתנגש. כמובן שהשאיפה לערוץ תקשורת ללא התנגשויות כמעט ובלתי ניתנת להשגה, ולכן ערוצי תקשורת נדרשים גם ל**תיקון שגיאות,** שיטות המסייעות בגילוי ותיקון ביטים שהשתנו או התנגשו. בעזרת המושגים והדוגמאות שעברנו עליהן בפרק זה, קיימים בידיכם הכלים להבין קצת יותר טוב את אמצעי התקשורת הסובבים את חיינו.

במהלך הפרק, ניתחנו מספר דוגמאות מהחיים. התחלנו במבט אל העבר בו נוסדו שיטות שונות להעברת מידע, ולאחר מכן התבוננו באמצעי התקשורת הפיזית שקיימים בכל בית ממוצע בישראל. עברנו דרך הטלגרף והטלפון, השווינו בין פטיפון לדיסק, ודיברנו על רשתות במשרד קטן וגדול ואף על ספקיות אינטרנט.

בפרק זה הבנו שהשכבה הפיזית מאוד שונה משאר השכבות בכך שיישומיה מגוונים מאוד וקיימים באמצעים רבים. אפשר לחשוב על השכבה כממשק המחבר בין שכבות התקשורת המופשטות וה״נקיות״ לבין אמצעי התקשורת הקיימים בעולם.

על אף שעברנו על כל חמש השכבות, עיסוקינו ברשתות עדיין רחוק מלהסתיים. בפרקים הבאים נחבר את הכלים והמידע שרכשנו בפרקים האחרונים לכדי הבנה טובה יותר של הדרך בה עובדת האינטרנט, וכן נכיר נושאים מתקדמים.

נספח א' – הרשת האלחוטית

אנחנו שוחים כל יום בכמות בלתי נתפסת של ביטים שזורמים באוויר בצורת גלים אלקטרומגנטיים. רשת ה-WiFi היא רק אחת מהן, ואליה מצטרפות הרשת הסלולרית, שידורי הטלוויזיה (האנלוגיים והדיגיטליים – עידן+[89]), שידורי הלוויין, שלטי טלוויזיה, דיבוריות Bluetooth ועוד. בכל רשת אלחוטית, ישנם מספר מאפיינים שמפרידים אותה מהשאר:

- טווח התדרים בו הרשת יכולה לשדר.
- עוצמת האות המשודר וכיוונו במרחב.
- קצב התקשורת.
- שיטת הסנכרון והתזמון בין רכיבי הרשת.

הרשת האלחוטית הביתית, בשמה הנפוץ ה-WiFi, מתבססת על פרוטוקול 802.11 שהוגדר ע״י מכון ה-IEEE [90]. פרוטוקול זה מכיל תת פרוטוקולים רבים: 802.11a, 802.11b, 802.11g, 802.11n ועוד. תת הפרוטוקולים שונים זה מזה במאפיינים הנ״ל, אך כולם משמשים לאותה המטרה – חיבור אלחוטי בין מספר מחשבים שהמרחק ביניהם לא עולה על כ-15 מטרים.
הרשתות האלחוטיות משדרות בתדר 2.4GHz (תת-פרוטוקול b ו-g) או 5GHz (תת-פרוטוקול a). תדר של 5GHz מאפשר קצב שידור גבוה יותר, בעוד תדר 2.4GHz מאפשר מרחק שידור גבוה יותר[91].

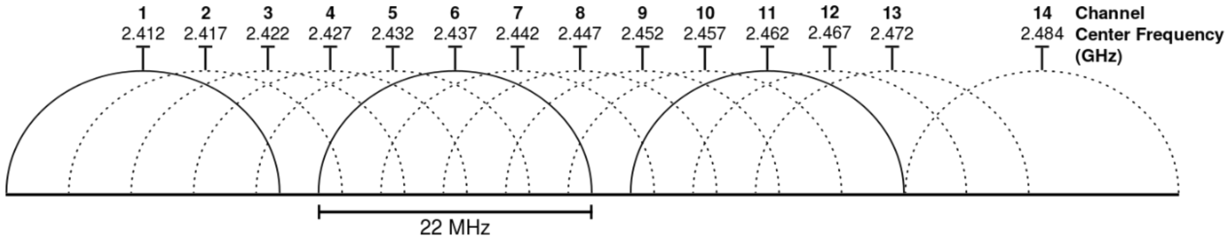
רשת ה-WiFi אינה משתמשת בתדר המדויק, אלא בתדר קרוב לתדר שצוין לעיל. התדר המדויק מוגדר לפי ה-Channel (ערוץ) בה הרשת האלחוטית עושה שימוש. לרשתות האלחוטיות הוגדרו 14 ערוצים. רשתות המשדרות בתדר 2.4GHz, בעצם משדרות בטווח התדרים בין 2.4GHz לבין 2.5GHz, ומתוכו כל רשת אלחוטית עושה שימוש בערוץ אחד אשר מכיל טווח תדרים של 22MHz. לדוגמה, רשת אלחוטית 802.11g, בערוץ מספר 6, משדרת בתדרים 2.426GHz עד 2.448GHz. כמו כן, בין הערוצים השונים יש חפיפה, ולכן רשתות אלחוטיות המשדרות בערוצים קרובים עלולות להפריע אחת לשנייה. התרשים הבא מראה את הערוצים של רשת WiFi בתדר 2.4GHz. ניתן לראות בתרשים שלכל ערוץ יש תדר מרכזי, וקשת המייצגת את טווח התדרים בהם הערוץ עושה שימוש.

---

[89] עידן+: שידורי טלוויזיה אלחוטיים דיגיטליים. ניתן להרחיב בקישור הבא: http://goo.gl/xtwlcA.

[90] מכון ה-IEEE הוא מכון בינלאומי שתפקידו לכתוב ולתחזק תקנים המאפשרים לתעשייה לייצר מוצרים שיעבדו אחד עם השני בתיאום. פגשנו גם בפרק שכבת הקו את ארגון ה-IEEE, אשר הגדיר את תקן ה-Ethernet, שמספרו 802.3. ניתן להרחיב: http://en.wikipedia.org/wiki/Institute_of_Electrical_and_Electronics_Engineers.

[91] כאשר התדר נמוך יותר, הגל מבצע פחות מחזורים בשנייה, ולכן אורך הגל גדול יותר. ככל שאורך הגל גדול יותר, כך טובה יותר יכולתו לעבור מכשולים כגון קירות.



מגבלה נוספת על קצב השידור ברשתות אלחוטיות נובעת משימוש בתדר שידור יחיד. מגבלה זו מחייבת את הרשת האלחוטית לפעול במצב **Half Duplex**. במצב זה, רק צד אחד יכול לשדר מידע, בעוד כל שאר רכיבי התקשורת נדרשים להאזין. משמעות המגבלה היא שככל שיש יותר מחשבים מחוברים ברשת אלחוטית בודדת, הקצב שלה יקטן ויתחלק בין כל המחשבים.

פרק 11

איך הכל מתחבר, ואיך עובד האינטרנט?

בפרק הראשון של הספר, התחלנו לשאול – איך עובד האינטרנט?
ניסינו לעשות זאת על ידי התמקדות בשאלה הבאה:

מה קורה כשאנו גולשים ל-Facebook?

בפרק הראשון, התחלנו לענות על השאלה הזו במושגים כלליים מאוד. מאז, עברנו כברת דרך ארוכה. למדנו לתכנת באמצעות Sockets, הכרנו את מודל חמש השכבות והתעמקנו בכל שכבה בו. רכשנו כלים כמו Wireshark ו-Scapy, והכרנו רכיבי רשת שונים. עכשיו, מצוידים בכל הידע הזה, נוכל לשאול מחדש את השאלה ששאלנו ולנסות להבין – איך כל מה שלמדנו מתחבר יחד?

בפרק זה ננסה לענות על כך ביתר פירוט, ונחבר דברים שכבר למדנו לכדי סיפור שלם – איך המחשב שלי מצליח לגלוש באינטרנט? לשם כך נשאל הרבה שאלות. **נסו לחשוב על התשובות, ובדקו האם אתם מצליחים לספר את הסיפור בעצמכם**.

הסיפור שלנו מתחיל עם המחשב שלנו:

מה המחשב שלנו צריך לעשות בכדי להצליח לתקשר עם האינטרנט?

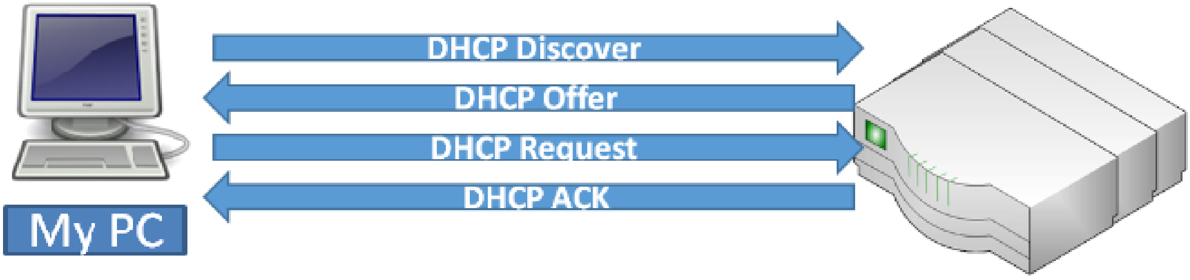
בתור התחלה, המחשב שלנו יצטרך לדעת כל מיני פרטים. הוא צריך לדעת מה כתובת ה-IP שלו, כדי שיוכל לשלוח אחר כך חבילות נוספות. הוא צריך לדעת מה מסיכת הרשת שלו, כדי לדעת איזה מחשבים נמצאים איתו ב-Subnet ואילו לא.

איזה מידע יש למחשב שלנו על הרשת?

בשלב הזה, המחשב יודע רק את **כתובת ה-MAC** של כרטיס הרשת שלו. הוא יודע את הכתובת, כיוון שזו צרובה באופן פיזי על כרטיס הרשת.

איך המחשב שלנו משיג את כתובת ה-IP ושאר פרטי הרשת שלו?

כפי שלמדנו ב[פרק שכבת הרשת/ DHCP](), ישנן מספר דרכים לקבל את פרטי הרשת. הדרך הנפוצה כיום היא באמצעות **פרוטוקול DHCP**. באמצעות פרוטוקול זה, כרטיס הרשת שלנו שולח הודעת DHCP Discover. ההודעה נשלחת ב-Broadcast, כלומר שכל הישויות ברשת יקבלו אותה. שרת ה-DHCP רואה את הבקשה ומחזיר DHCP Offer, הודעה בה הוא כולל את פרטי הרשת המוצעים לכרטיס הרשת שלנו: כתובת ה-IP שלו, שרת ה-DNS הרלוונטי ועוד. מכיוון שברשת שלנו יש רק שרת DHCP אחד, לא תהיינה הצעות נוספות, והמחשב שלנו ישלח הודעת DHCP Request שתודיע לשרת ה-DHCP שהוא מבקש לקבל את ההצעה. לבסוף, שרת ה-DHCP ישלח הודעת DHCP ACK, שאחריה יוכל המחשב שלנו להתחיל להשתמש בכתובת ה-IP שניתנה לו.



מזל טוב! קיבלנו כתובת IP!

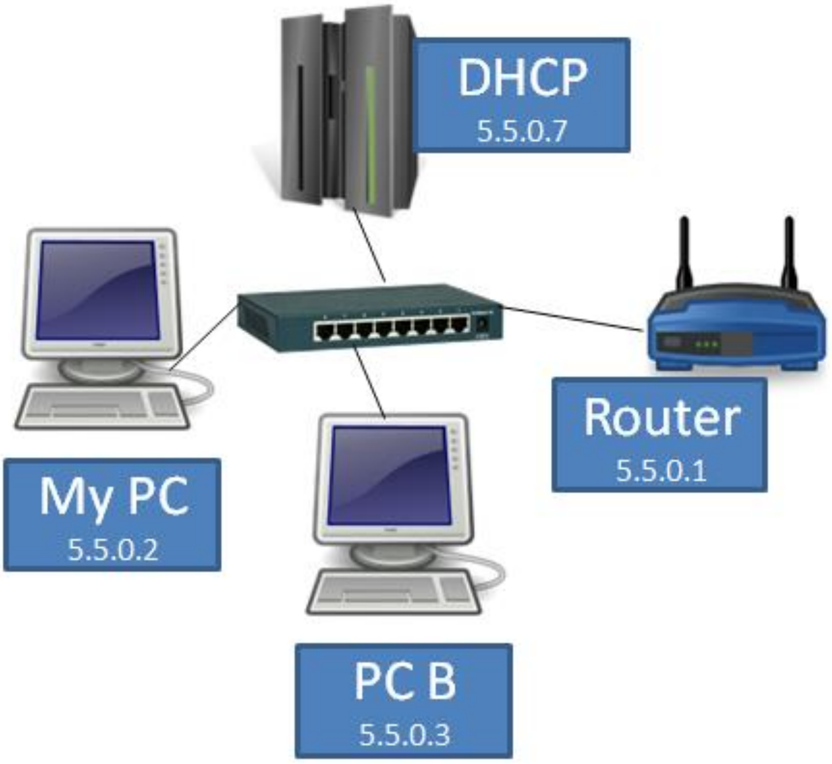
לצורך הדוגמה, נאמר שקיבלנו את הכתובת 5.5.0.2, כשמסכת הרשת היא 255.255.0.0[92].

בנוסף, הנתב שלנו עובר תהליך דומה על מנת להשיג כתובת IP משלו. את התהליך הזה הוא עובר מול שרת ה-DHCP של ספקית האינטרנט שלנו. נאמר שהנתב קיבל את הכתובת: 1.1.1.1, ומסיכת הרשת היא 255.255.0.0.

[92] כפי שלמדנו ב[פרק שכבת הרשת/ נספח ב' – כתובות פרטיות ו- NAT](), הכתובת תהיה לעיתים תכופות כתובת IP פרטית. על מקרה זה נרחיב ב[נספח א' של פרק זה – תקשורת מאחורי NAT]().

איך ההודעה הגיעה אל שרת ה-DHCP?

על מנת שההודעה תצליח להגיע אל שרת ה-DHCP, על הלקוח להיות איתו באותו ה- Broadcast Domain. כלומר, ההודעה צריכה להגיע מבלי לעבור באף נתב בדרך. זה הזמן להיזכר שבעצם, כפי שלמדנו בפרק שכבת הקו/ רכיבי רשת בשכבת הקו, המחשב שלנו מחובר אל **Switch (מתג)**, אליו מחוברים גם מחשבים נוספים ברשת (למשל המחשב בשם "PC B"), אותו שרת DHCP[93], וכמובן – ה**נתב (Router)** שלנו.



מה השלב הבא?

כעת, המחשב שלנו צריך לגלות מה היא כתובת ה-IP של www.facebook.com, על מנת שיוכל לשלוח אל השרת של Facebook בקשות. כפי שלמדנו בפרק שכבת האפליקציה, על המחשב שלנו להשתמש ב**פרוטוקול DNS**, ולתשאל את שרת ה-DNS שלו מהי כתובת ה-IP של Facebook.

[93] ברשתות ביתיות של משתמשים פרטיים, הנתב הביתי הוא בדרך כלל גם שרת ה-DHCP של הרשת. לצורך פשטות ההסבר, נניח במקרה זה שישנו שרת DHCP נפרד.

מהו שרת ה-DNS שלנו? כיצד המחשב יודע זאת?

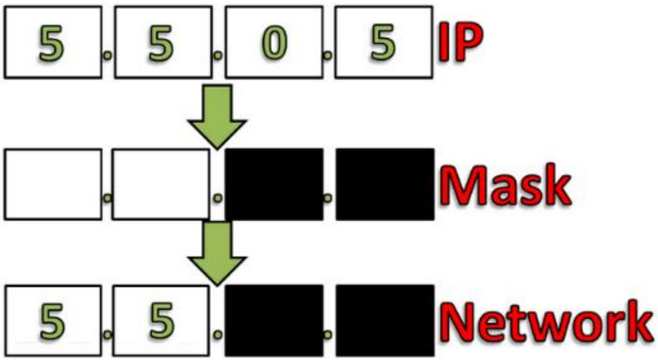
שרת ה-DNS שלנו הוא שרת ה-DNS של ספקית האינטרנט שלנו[94]. המחשב שלנו יודע זאת כיוון שהוא קיבל את כתובת ה-IP של שרת ה-DNS באמצעות תהליך ה-DHCP, בו הוא קיבל גם את כתובת ה-IP שלו. לצורך הדוגמה, נאמר שכתובת ה-IP של שרת ה-DNS הינה 2.2.2.2.

כיצד המחשב שלנו יודע לפנות אל שרת ה-DNS?

עכשיו כשהמחשב יודע שעליו לפנות אל שרת ה-DNS ולשלוח אליו שאילתת DNS, איך הוא יוכל לעשות זאת? האם הוא יפנה אל שרת ה-DNS ישירות? אם לא, אל מי הוא יעביר את החבילה?

בשלב זה, כפי שלמדנו בפרק שכבת הקו/ למי נשלחת שאלת ה- ARP?, המחשב בודק האם כתובת ה-IP של שרת ה-DNS נמצאת איתו באותו ה-Subnet.

כזכור, כתובת ה-IP של המחשב שלנו הינה: 5.5.0.2, ומסכת הרשת היא: 255.255.0.0. כפי שלמדנו בפרק שכבת הרשת/ מהו מזהה הרשת שלי ? מהו מזהה הישות?, משמעותה של מסכת רשת זו היא ששני הבתים הראשונים שייכים למזהה הרשת:



כתובת ה-IP של שרת ה-DNS, אותה מצאנו קודם, היא 2.2.2.2. מכיוון ששני הבתים הראשונים של כתובת זו הם 2.2 ולא 5.5, הרי ששרת ה-DNS לא נמצא באותו ה-Subnet של המחשב.

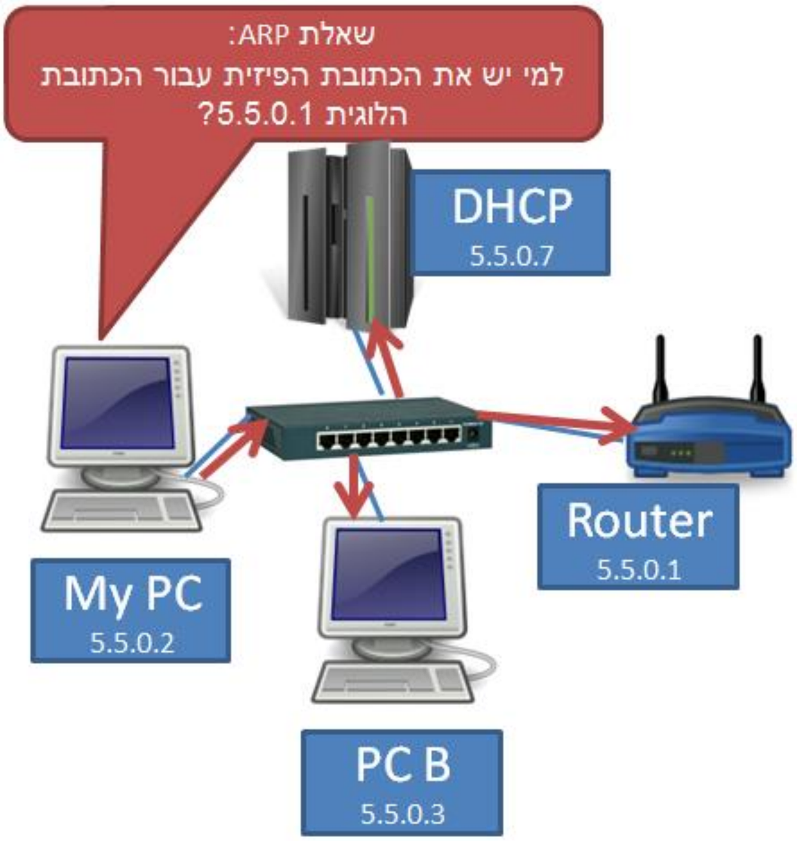
היות שהכתובת 2.2.2.2 לא נמצאת באותו ה-Subnet כמו זו של המחשב, מערכת ההפעלה מבינה שעליה לפנות אל ה-**Default Gateway**, אותו הנתב שיאפשר למידע לצאת מהרשת המקומית אל האינטרנט. גם את הכתובת של נתב זה מצאנו קודם לכן, בתהליך ה-DHCP.

---

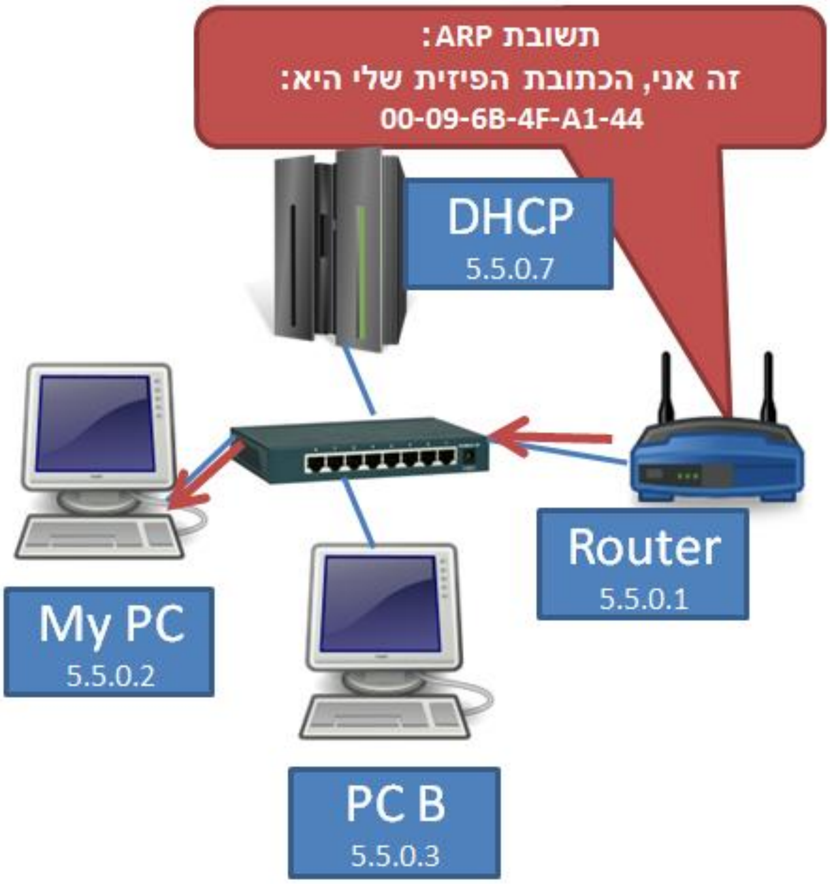
[94] במקרים מסוימים, הנתב ישמש כשרת ה-DNS. כלומר, המחשב ישלח שאילתות DNS אל הנתב, שבתורו יתשאל את השרת של הספקית.

איך נצליח לתקשר עם הנתב?

הבנו שאנחנו צריכים לשלוח את חבילת ה-DNS אל שרת ה-DNS, שכתובתו 2.2.2.2. אנו גם יודעים שעלינו להעביר ראשית את החבילה אל הנתב, שכתובתו 5.5.0.1. אך מידע זה אינו מספיק. מכיוון שכתובת IP היא כתובת לוגית, וכרטיס הרשת שלנו מכיר כתובות פיזיות בלבד – נצטרך לגלות את **הכתובת הפיזית** של הנתב. כפי שלמדנו בפרק שכבת הקו/ פרוטוקול ARP, תהליך זה מתבצע באמצעות **פרוטוקול ARP**. בהנחה שאין רשומה עבור הנתב ב-ARP Cache של המחשב שלנו, המחשב ישלח ב-Broadcast הודעה לכלל הרשת: "למי יש את הכתובת הפיזית עבור הכתובת הלוגית 5.5.0.1?" שאלה זו נקראת ARP Request:



בשלב זה, הנתב רואה את ה-ARP Request, ומגיב למחשב שלנו בתשובה שנקראת ARP Reply:



כעת למחשב יש את כל המידע הדרוש על מנת לשלוח חבילה אל שרת ה-DNS! אך בטרם נמשיך להתעסק בהודעה זו, ישנה שאלה קודמת עליה אנו צריכים לענות:

איך ה-Switch יודע להעביר את ההודעות?

הודעת ה-ARP Reply נשלחה מהנתב, הגיעה אל ה-**Switch (מתג)**, שידע להעביר אותה אך ורק אל המחשב שלנו. כיצד הוא עשה זאת?

כפי שלמדנו בפרק שכבת הקו/ כיצד Switch פועל?, ל-Switch יש טבלה אותה הוא ממלא בזמן ריצה. טבלה זו ממפה בין כתובת MAC לבין הפורט הפיזי הרלוונטי. כאשר ה-Switch חובר לרשת לראשונה, הטבלה הייתה ריקה:

MAC Address	Port

בפעם הראשונה בה המחשב שלנו שלח מסגרת כלשהי, למשל את חבילת ה-DHCP Discover, ה-Switch קרא את כתובת המקור של המסגרת, ושייך אותה לפורט הפיזי אליו מחובר המחשב:

Port	MAC Address
1	My PC

עכשיו, כאשר הנתב שלח את ההודעה, ה-Switch בדק בטבלה שלו, וראה שהיא מיועדת לכתובת ה-MAC של המחשב שלנו, ומכאן שהיא מיועדת לפורט הפיזי שלו. כך ידע ה-Switch למתג את ה-ARP Reply אך ורק אל המחשב שלנו.

כעת נוכל להמשיך עם החבילה שברצוננו לשלוח אל שרת ה-DNS.

מהן הכתובות בחבילה?

בטרם נתעכב על שכבת ה-DNS ודרך הפעולה שלה, עלינו להבין כיצד נראית חבילה שנשלחת אל שרת ה-DNS באופן כללי. ננסה להשלים את השדות הבאים של הפקטה:

כתובת MAC מקור	
כתובת MAC יעד	
כתובת IP מקור	
כתובת IP יעד	

בטרם תמשיכו בקריאה, נסו להשלים את הטבלה בעצמכם.

כתובות ה-MAC

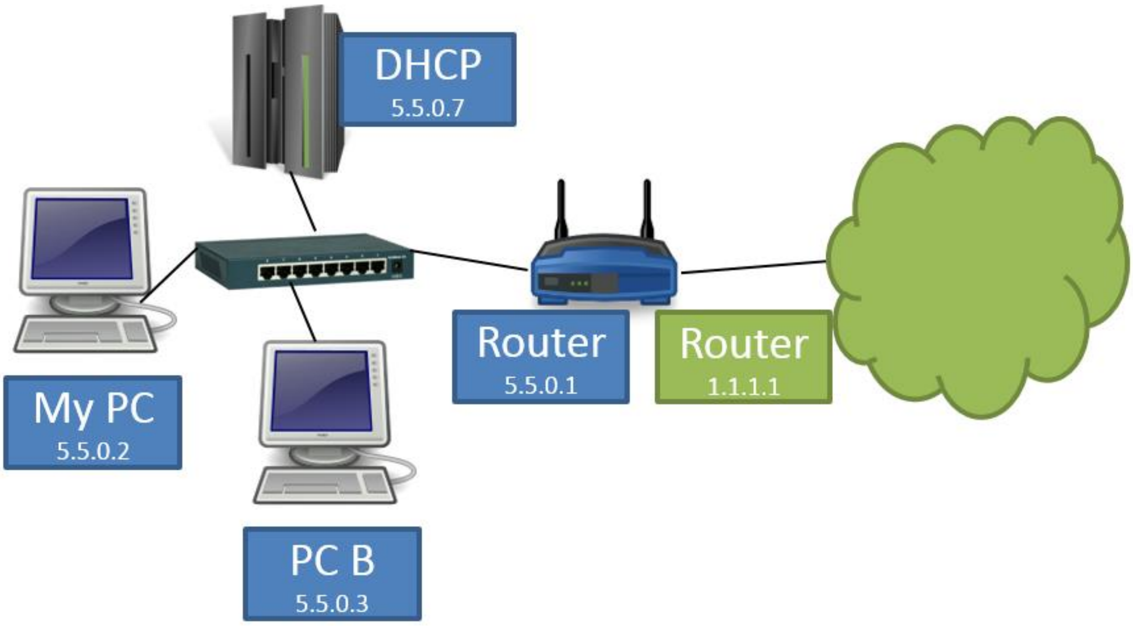
הכתובת הפיזית של המקור שייכת לכרטיס הרשת של המחשב שלנו – הרי הוא זה ששולח את החבילה. המחשב יודע את הכתובת הזו שכן היא שייכת לכרטיס הרשת שלו עצמו.

הכתובת הפיזית של היעד תהיה של הנתב, שכן הוא התחנה הקרובה בדרך אל היעד. כזכור, כתובות פיזיות שייכות לשכבת הקו ומציינות את התחנה הקרובה בכל פעם. את הכתובת הפיזית של הנתב השגנו קודם לכן באמצעות פרוטוקול ARP.

נמלא את השדות הרלוונטיים בטבלה:

כתובת MAC מקור	המחשב שלנו (כתובת ידועה)
כתובת MAC יעד	הנתב (הושגה באמצעות ARP)
כתובת IP מקור	
כתובת IP יעד	

שימו לב כי כתובת ה-MAC של הנתב משויכת לפורט הפיזי שמחובר אל ה-Switch ברשת שלנו, ולא לפורט פיזי אחר. לכל פורט פיזי של הנתב יש כתובת MAC משלו. אם נביט בשרטוט הרשת:



נראה כי לנתב יש שני פורטים פיזיים: הפורט הראשון מחבר אותו אל הרשת הביתית שלנו. עבור פורט זה, כתובת ה-IP היא: 192.168.0.1. לפורט זה יש כתובת MAC מסוימת. הפורט השני מחבר את הנתב אל הרשת של הספקית, המוצגת כענן ירוק. עבור פורט זה, כתובת ה-IP היא: 1.1.1.1. לפורט זה יש כתובת MAC שונה מהכתובת של הפורט הפיזי שמחבר את הנתב לרשת שלנו. בהודעה שהמחשב ישלח, כתובת ה-MAC בשדה כתובת היעד תהיה הכתובת של הפורט הפיזי המחובר לרשת שלנו.

כתובות ה-IP

כתובת ה-IP של המקור תהא אף היא של המחשב שלנו, זאת מכיוון שאנחנו שולחים את החבילה. את הכתובת הזו השגנו באמצעות תהליך ה-DHCP.
כתובת ה-IP של היעד תהיה הכתובת של שרת ה-DNS. זאת מכיוון שבשכבת הרשת, כתובת היעד מצביעה על היעד הסופי – אל מי החבילה אמורה להגיע בסופו של דבר. את הכתובת של שרת ה-DNS גילינו קודם לכן באמצעות תהליך ה-DHCP.
הטבלה המלאה תיראה כך:

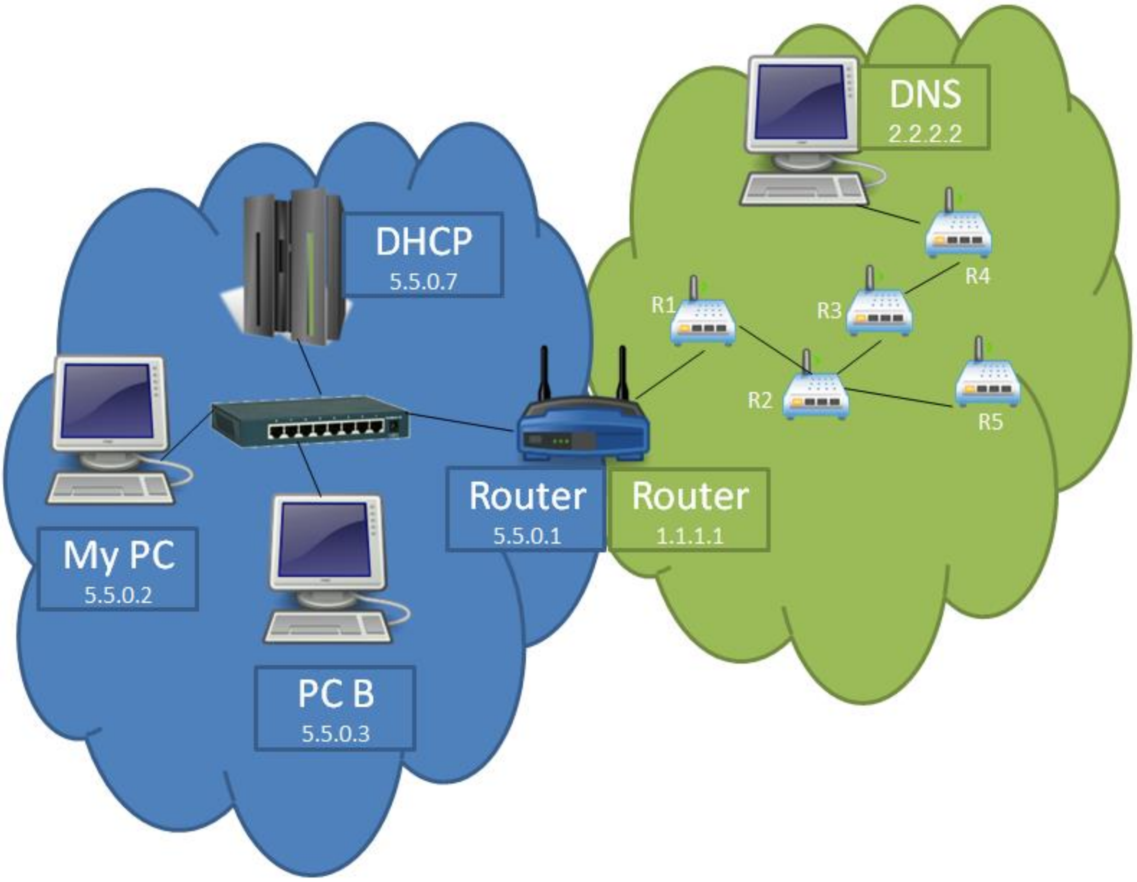
כתובת MAC מקור	המחשב שלנו (כתובת ידועה)
כתובת MAC יעד	הנתב שלנו (הושגה באמצעות ARP)
כתובת IP מקור	המחשב שלנו (הושגה באמצעות DHCP)
כתובת IP יעד	שרת ה-DNS (הושגה באמצעות DHCP)

נדגיש כי בטבלה זו נראה באופן ברור ההבדל בין השכבה השנייה לשכבה השלישית. בעוד שכבת הקו מציינת את הכתובות של ה-Hop הנוכחי, כלומר שלב אחד בדרך (ומיוצגת בידי כתובות ה-MAC בשתי השורות הראשונות של הטבלה), שכבת הרשת מציינת את המקור והיעד הסופיים של החבילה (ומוצגת בידי כתובות ה-IP בשתי השורות התחתונות של הטבלה).

מהן הכתובות בתחנה הבאה?

כאשר הנתב יקבל את החבילה, ויעביר אותה הלאה, כיצד תיראנה הכתובות?
נסתכל על תמונת הרשת שלנו, שהתרחבה מעט. כעת מופיע גם שרת ה-DNS, שהוא חלק מהרשת של ספקית האינטרנט שלנו.

בנוסף, מופיעים נתבים נוספים השייכים לספקית האינטרנט:



לצורך הבהרה, הרשת המקומית (LAN) שלנו סומנה בכחול, והרשת של הספקית סומנה בירוק. שימו לב שהנתב שלנו נמצא בשתי הרשתות, ויש לו שתי כתובות IP – אחת "פנימית", שהיא הכתובת 5.5.0.1, המשמשת אותו ברשת הביתית שלנו, והשנייה "חיצונית", שהיא הכתובת 1.1.1.1 ומשמשת אותו אל מול הספקית בפרט והאינטרנט בכלל.

נאמר שהנתב החליט להעביר את החבילה המיועדת אל שרת ה-DNS אל הנתב R1. אילו כתובות יהיו עכשיו בפקטה?

נסו למלא בעצמכם את הטבלה הבאה:

כתובת MAC מקור	
כתובת MAC יעד	
כתובת IP מקור	
כתובת IP יעד	

כתובות ה-MAC

כתובת ה-MAC של המקור תהיה עכשיו הכתובת של הנתב שלנו. זאת מכיוון שהוא זה ששולח את החבילה – כלומר כרטיס הרשת שלו הוא זה שמעביר את החבילה הלאה. שימו לב כי כתובת ה-MAC שייכת לפורט הפיזי של הנתב שנמצא ברשת של הספקית (הרשת הירוקה באיור לעיל), שהיא שונה מכתובת ה-MAC ששייכת לפורט הפיזי של הנתב ברשת המקומית שלנו.

כתובת ה-MAC של היעד תהיה של הנתב R1, באופן ספציפי זו של הפורט הפיזי שמחובר אל הנתב של הרשת הביתית שלנו (ולא הפורט הפיזי שמחובר אל הנתב R2). הכתובת תהיה של הנתב R1 מכיוון שהתחנה הבאה של החבילה היא הנתב R2, וכזכור השכבה השנייה אחראית לתקשורת בין תחנות הסמוכות זו לזו.
נמלא את השדות הרלוונטיים בטבלה:

כתובת MAC מקור	הנתב שלנו
כתובת MAC יעד	R1
כתובת IP מקור	
כתובת IP יעד	

כתובות ה-IP

כתובת המקור תהיה הכתובת של המחשב שלנו. זאת מכיוון שהוא שלח את ההודעה המקורית, ובשכבה השלישית אנו מציינים את המקור והיעד הסופיים של החבילה. מסיבה זו, כתובת IP היעד תהא זו של שרת ה-DNS.
הטבלה המלאה תיראה כך:

כתובת MAC מקור	הנתב שלנו
כתובת MAC יעד	R1
כתובת IP מקור	המחשב שלנו
כתובת IP יעד	שרת ה-DNS

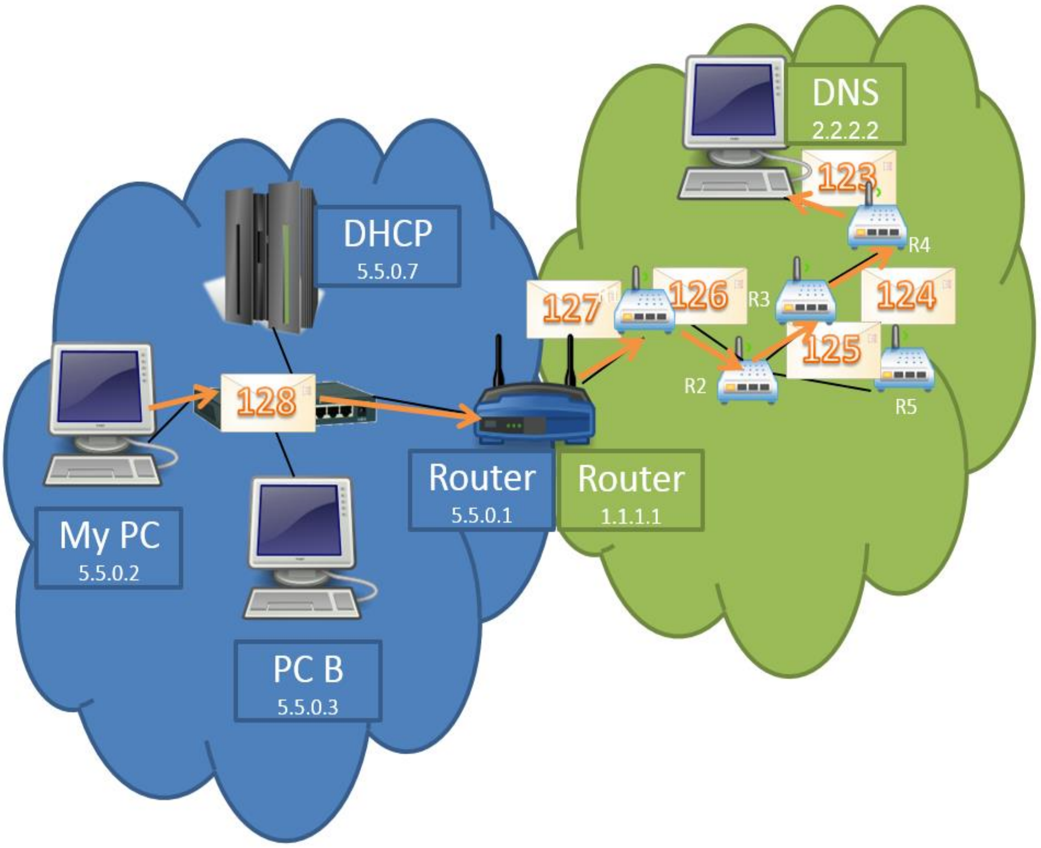
למעשה, כתובות ה-IP נותרו ללא שינוי! כך ניתן לראות שבעוד כתובות ה-MAC משתנות בכל Hop בדרך ומציינות מה השלב הנוכחי שהחבילה עוברת, כתובות ה-IP נשארות קבועות לאורך המשלוח ומציינות ממי החבילה הגיעה במקור ומה יעדה הסופי.

מה עושה כל נתב עם החבילה?

מה עוד עושה כל נתב בדרך, לפני שהוא מעביר הלאה את החבילה שהוא קיבל? מה יעשה R1 כאשר יקבל את החבילה? ומה יעשה R2?

כאשר הנתב מקבל את החבילה, עליו לחשב את ה-**Checksum** ולוודא שהיא תקינה. לאחר מכן, הוא מחסיר 1 מערך ה-(TTL) **Time To Live** של החבילה, כפי שלמדנו בפרק שכבת הרשת/ איך Traceroute עובד?. בעקבות החסרה זו, עליו לחשב את ערך ה-**Checksum** מחדש בטרם יעביר את החבילה הלאה.

נאמר שהמחשב שולח את החבילה עם ערך TTL התחלתי של 128. נעקוב אחר ערך ה-TTL לאורך המסלול (מוצג בתור המספר של החבילה):



כפי שניתן לראות, החבילה מגיעה אל שרת ה-DNS כשערך ה-TTL הוא 123, כיוון שהיא עוברת חמישה נתבים בדרך. שימו לב שהחבילה הגיעה אל הנתב הקרוב אל המחשב שלנו דרך ה-Switch, שלא שינה את ערך ה-TTL. רק רכיבי שכבת הרשת, כגון נתבים, משנים את ערך זה, ולא רכיבי שכבת הקו למיניהם.
כמו כן, הנתב צריך **לנתב** את הפקטה, כלומר להחליט מה המסלול שעליה לעבור. כפי שלמדנו בפרק שכבת הרשת/ ניתוב, הנתב מבין לאן עליו להעביר את החבילה באמצעות טבלאות ניתוב שנשמרות אצלו, ונבנות באופן דינאמי.

כיצד מוצא המחשב את כתובת ה-IP של Facebook?

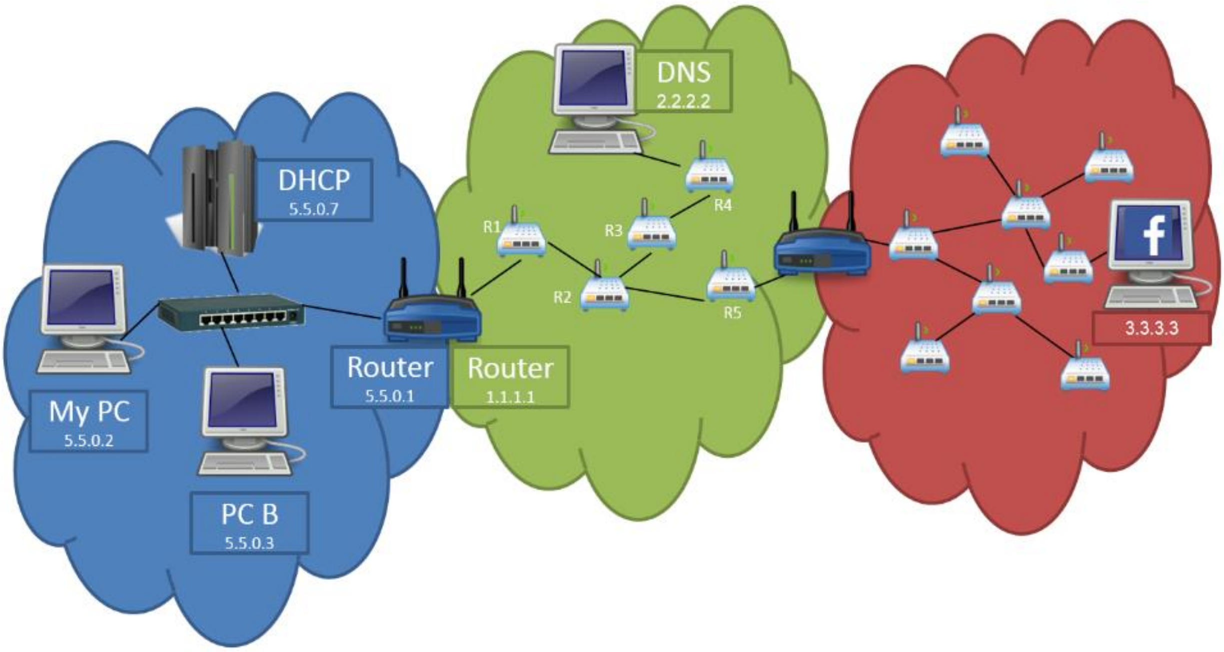
כעת, סוף סוף, לאחר שהבנו כיצד ניתן להעביר חבילה אל שרת ה-DNS, נוכל לחזור ולדון מה החבילה הזו כוללת. כפי שלמדנו בפרק שכבת האפליקציה/ התבוננות בשאילתת DNS, החבילה הנשלחת תהיה חבילת שאילתה (Query). כפי שלמדנו, שאילתות ותשובות DNS מורכבות מרשומות (שנקראות Resource Records). השאילתה שישלח המחשב שלנו צפויה להיות מסוג A, כלומר תרגום בין שם דומיין לכתובת IP. החבילה מכילה Transaction ID מסוים. לצורך הדוגמה, נאמר שה-Transaction ID הינו 1337. כאשר שרת ה-DNS יענה על השאילתה, גם התשובה תכלול את הערך 1337 בשדה ה- Transaction ID. כמו כן, חבילת התשובה תכלול הן את השאילתה המקורית ששלח המחשב שלנו, והן רשומת תשובה אחת או יותר.

מה השלב הבא?

לאחר שמצאנו את כתובת ה-IP של Facebook, הגיע הזמן לגשת אליו. מכיוון שבהמשך אנו עתידים להשתמש בפרוטוקול HTTP, עלינו ראשית **להרים קישור TCP** עם השרת של Facebook.

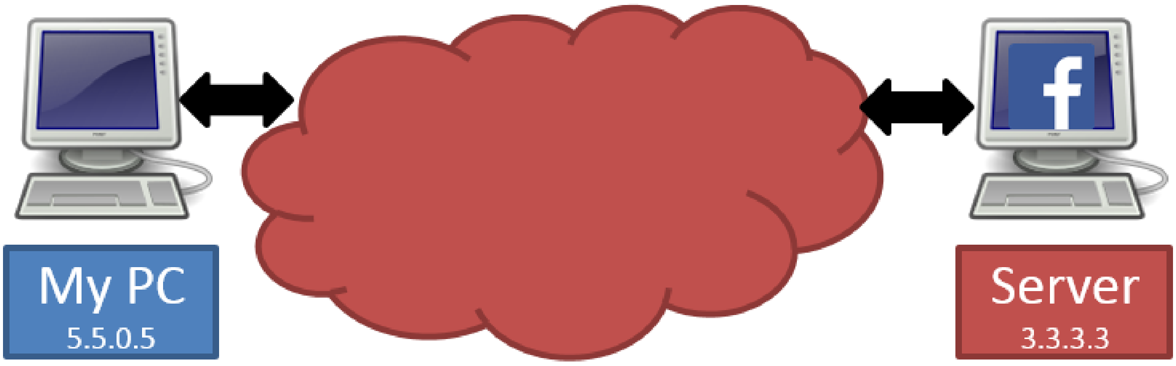
שימו לב שתמונת הרשת שלנו כבר גדלה מאוד, ועברנו מהשלב בו הסתכלנו על מחשב יחיד:

אל שלב בו אנו רואים את הרשת המקומית, הרשת של הספקית וכן האינטרנט:



שימו לב כי באיור האינטרנט אמנם נראה כמו ספקית יחידה (לטובת פשטות האיור), אך הוא מכיל הרבה ספקיות.

בשלב זה, תמונת הרשת כבר גדולה למדי, ומכילה את המחשב שלנו, רכיבי רשת כגון Switch ונתבים, וכן שרתים. עם זאת, כפי שלמדנו בפרק שכבת התעבורה/ מיקום שכבת התעבורה במודל השכבות, שכבת הרשת מספקת לשכבת התעבורה מודל של "ענן", ובכך מבטלת את הצורך שלה להכיר את מבנה הרשת. אי לכך, נוכל להציג את התמונה גם בצורה שבה שכבת התעבורה "רואה" אותה, כך:



באילו פורטים תתבצע התקשורת?

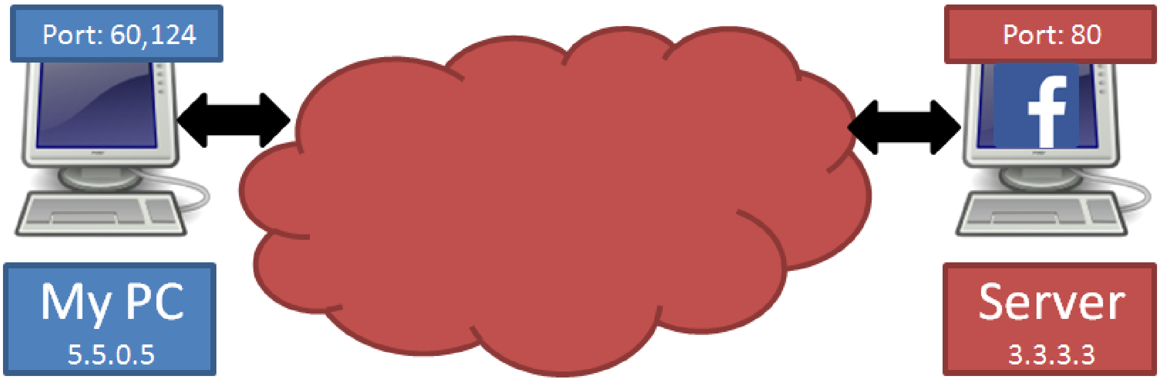
כפי שלמדנו בפרק שכבת התעבורה/ ריבוב אפליקציות – פורטים, תקשורת בשכבה זו בכלל, ובפרוטוקול TCP בפרט, מתבצעת בין מזהי כתובות IP **ופורטים**.

מה יהיה הפורט במחשב שלנו, ומה הפורט במחשב היעד?

הפורט שאליו תתבצע הפנייה, כלומר הפורט המאזין בשרת של Facebook, יהיה ככל הנראה פורט 80. פורט זה הינו הפורט המשמש בדרך כלל לפרוטוקול HTTP. **שימו לב:** אין הדבר אומר שלא ניתן לתקשר ב-HTTP מעל מספר פורט אחר, אך בדרך כלל שרתים יאזינו לבקשות HTTP בפורט זה.

הפורט ממנו תתבצע הפנייה, כלומר הפורט במחשב שלנו, יהיה מספר רנדומלי שתגריל מערכת ההפעלה. עם זאת, מספר זה לא יהיה רנדומלי לחלוטין, מכיוון שישנם מספרי פורטים השמורים לאפליקציות מסוימות. אי לכך, רוב מערכות ההפעלה מגרילות מספר פורט בטווח שבין 49,152 ובין 65,535. נאמר שהפורט שהוגרל הינו 60,124.

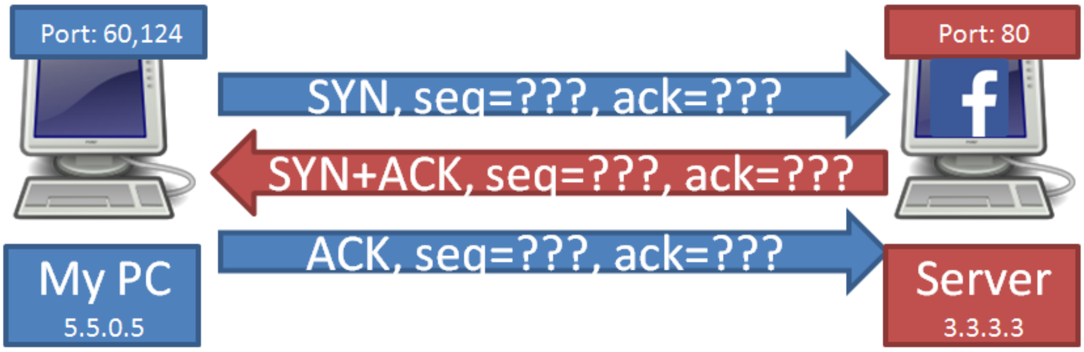
מכאן שהתקשורת תיראה כך:



כיצד נראית הרמת הקישור?

על מנת להרים קישור TCP בין המחשב שלנו לבין השרת המרוחק, נשתמש ב- **Three Way Handshake**, כפי שלמדנו בפרק שכבת התעבורה/ הרמת קישור ב- TCP. היות ששכבת התעבורה אינה מודעת למבנה הרשת, נסתמך על מודל ה"ענן" שמספקת שכבת הרשת ונתייחס לתקשורת כאילו היא מתבצעת באופן ישיר בין המחשב שלנו לבין השרת של Facebook.

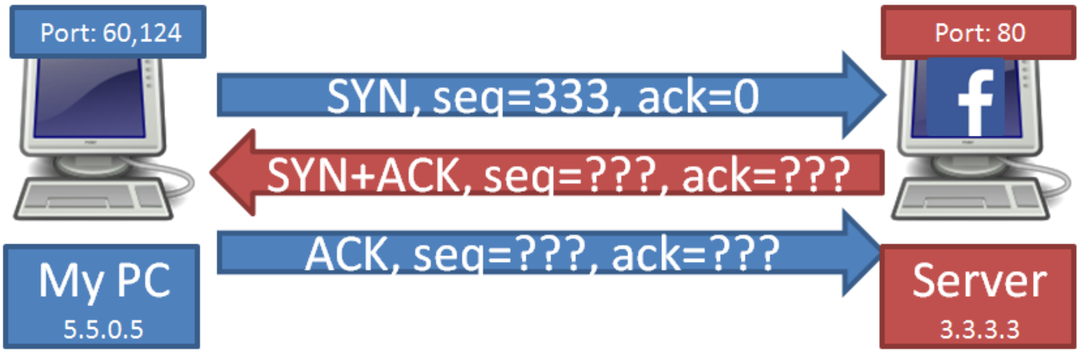
נסו להשלים בעצמכם את הערכים שבתרשים הבא, במקום סימני השאלה:



נתחיל מהחבילה הראשונה, היא חבילת ה-SYN. זו החבילה המציינת את תחילת הקישור, ולכן הדגל SYN בה דלוק. הדגל ACK כבוי, מכיוון שזו החבילה הראשונה בקישור, ולא מתבצע אישור על קבלב של מידע קודם.

ערך ה-Sequence Number של חבילה זו הינו ערך ה-Sequence Number ההתחלתי של הקישור, ועל כן הוא יהיה רנדומלי, כפי שלמדנו בפרק שכבת התעבורה/ חבילה ראשונה – SYN. לצורך הדוגמה, נאמר שהערך שהוגרל הוא 333.

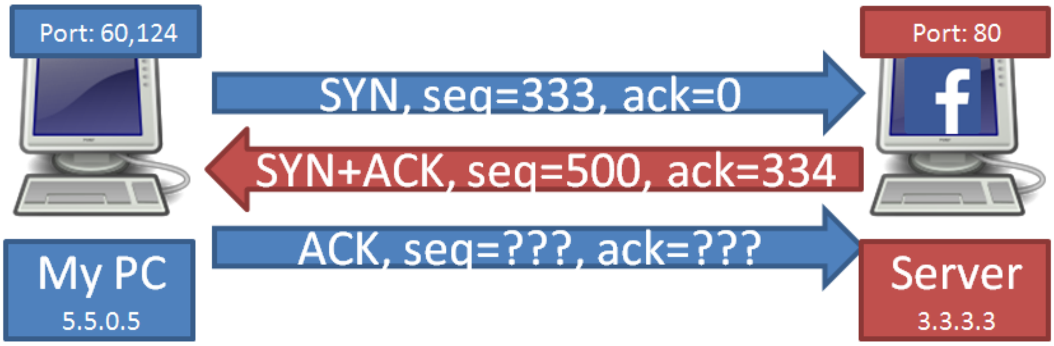
ערך ה-Acknowledgement Number של חבילה זו יהיה 0, זאת מכיוון שדגל ה-ACK כבוי. נוכל למלא את הערכים האלו בשרטוט:



כעת נעבור לחבילה השנייה. בחבילה זו, דגל ה-SYN דלוק היות שמדובר בחבילה הראשונה בצד של הקישור שבין השרת למחשב שלנו. דגל ה-ACK דלוק שכן יש לתת אישור על הגעת חבילת ה-SYN מהלקוח.

שדה ה-Sequence Number של חבילה זו יהיה אף הוא רנדומלי, מכיוון שהוא מציין את ערך ה- Sequence Number ההתחלתי של רצף המידע שעובר בין השרת לבין המחשב שלנו. לצורך הדוגמה, נאמר שהערך שהוגרל הוא 500.

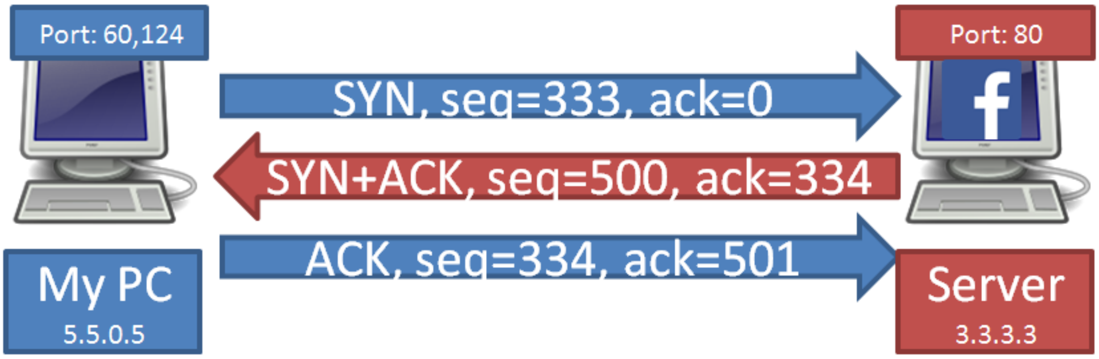
שדה ה-Acknowledgement Number אמור להעיד על כך שחבילת ה-SYN התקבלה. כפי שלמדנו בפרק שכבת התעבורה/ איך TCP משתמש ב- Acknowledgement Numbers, ערך שדה זה מחושב באמצעות ערך ה-Sequence Number של החבילה שהתקבלה (333), בנוסף לאורך המידע המועבר בה. היות שהחבילה שהתקבלה הינה חבילת SYN, גודל המידע שמחושב עבורה הוא גודל של בית (byte) אחד. אי לכך, הערך יהיה 333+1, כלומר 334.



נותרנו עם החבילה השלישית והאחרונה לתהליך הרמת הקישור. בחבילה זו, דגל ה-SYN כבוי, מכיוון שזו לא החבילה המעידה על יצירת הקישור. דגל ה-ACK דלוק, שכן יש לאשר את קבלת החבילה הקודמת שנשלחה מהשרת.

מה יהיה ערך ה-Sequence Number? כפי שלמדנו בפרק שכבת התעבורה, ערך ה-Sequence Number זהה לערך ה-Acknowledgement Number של החבילה הקודמת (בהנחה שלא נשלחו עוד חבילות בינתיים). אי לכך, ערך שדה זה יהיה 334.

כמו שציינו קודם לכן, שדה ה-Acknowledgement Number אמור להעיד על כך שהחבילה הקודמת התקבלה, ומחושב באמצעות ערך ה-Sequence Number של החבילה שהתקבלה (500), בנוסף לגודל המידע, שהוא בית (bytes) אחד במקרה של חבילת SYN. אי לכך, הערך יהיה 500+1, כלומר 501.



איך נראית בקשת ה-HTTP?

האתר של Facebook משתמש ב- HTTPS המודרני ולא ב-HTTP, אולם העקרונות הם דומים למדי, למעט העובדה ש-HTTPS הוא מוצפן. כיוון שבאמצעות Wireshark לא ניתן לצפות בתעבורת ה-HTTPS נמשיך את ההמחשה שלנו באמצעות HTTP, תוך שאנחנו זוכרים שבפועל הדברים עוברים הצפנה. הצלחנו להרים קישור TCP. עכשיו, באמצעותו, נוכל סוף סוף לבקש מ-Facebook לשלוח אלינו את העמוד הראשי שלו.

כפי שלמדנו בפרק שכבת האפליקציה/ פרוטוקול – HTTP בקשה ותגובה, כאשר דפדפן פונה לאתר כלשהו, הוא פונה באמצעות בקשת GET. מכיוון שהפנייה מתבצעת אל העמוד הראשי של Facebook, מבלי לבקש אף משאב ספציפי, היא תתבצע אל המשאב שנמצא בכתובת "/". כזכור מפרק שכבת האפליקציה/ מבנה פורמלי של בקשת HTTP, אחרי המילה GET תופיע המחרוזת "HTTP" והגירסה של הפרוטוקול, למשל: 1.0. לאחר מכן יופיעו ה-HTTP Headers, עליהם לא נעמיק בפרק זה.

איך נראית תשובת ה-HTTP?

בהנחה ו-Facebook מוכן להחזיר את העמוד הראשי שלו בלא עיכובים נוספים, הוא יענה בתשובת HTTP מסוג 200 (OK). כפי שראינו בפרק שכבת האפליקציה/ מבנה פורמלי של תשובת HTTP, התשובה מתחילה במחרוזת "HTTP" והגירסה של הפרוטוקול. מייד אחר כך, יופיע הקוד של התגובה (200) ואז הפירוש הטקסטואלי של הקוד (OK). לאחר מכן, יופיע כלל ה-Headers הרלוונטים, ולבסוף – המידע עצמו.

בתשובה זו הגענו לסוף התהליך, והדפדפן שלנו יכול סוף סוף לראות את העמוד הראשי של Facebook.

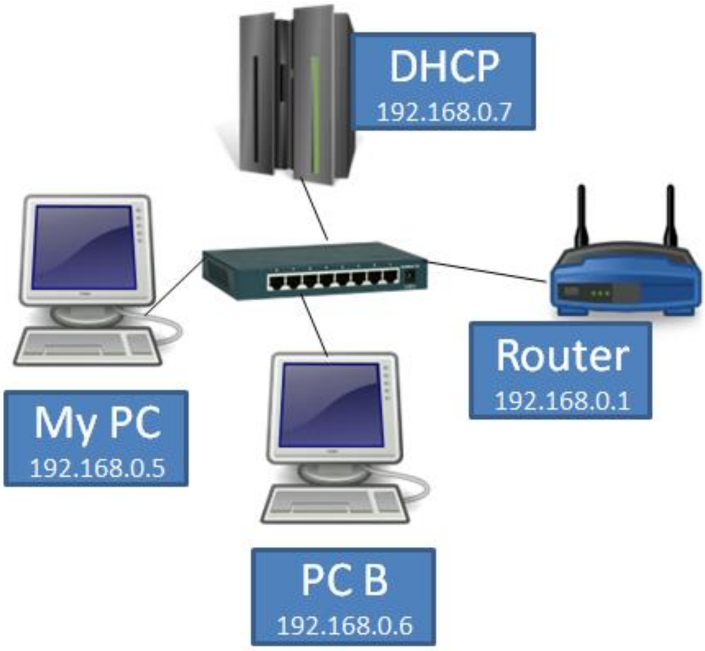
מה קורה כאשר המחשב שלנו נמצא מאחורי NAT?

בפרק זה, עברנו על היבטים רבים הנוגעים לתהליך שקורה כאשר אנו גולשים אל Facebook. עם זאת, על אף שהמקרה שהצגנו אפשרי, הוא לא המקרה הרווח ברשת האינטרנט, שכן הוא מתעלם מהשימוש בכתובות פרטיות ו-NAT, אותן למדנו להכיר בפרק שכבת הרשת. כעת נתאר את השינויים המתרחשים כאשר המחשב שלנו נמצא מאחורי NAT. שימו לב שנתמקד בהבדלים בלבד, ולא נחזור על התהליך כולו.

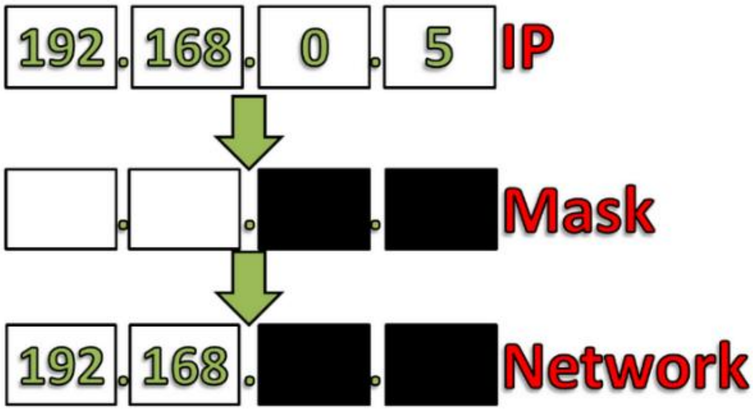
כפי שלמדנו בפרק שכבת הרשת, הכתובת שהמחשב שלנו מקבל משרת ה-DHCP שלו צפויה להיות **כתובת IP פרטית**, וזאת בכדי לחסוך בכתובות IP בעולם. במקרה שלנו, נאמר שקיבלנו את הכתובת: 192.168.0.5. מסיכת הרשת היא: 255.255.0.0.

בנוסף, על מנת לקבל **כתובת IP חיצונית**, הנתב שלנו עובר תהליך דומה. כאשר הנתב מתקשר עם שרת ה-DHCP של ספקית האינטרנט, הוא מקבל ממנה כתובת IP שאינה פרטית. נאמר שהנתב קיבל את הכתובת: 1.1.1.1, ומסיכת הרשת היא: 255.255.0.0.

תמונת הרשת עשויה להיראות כך:

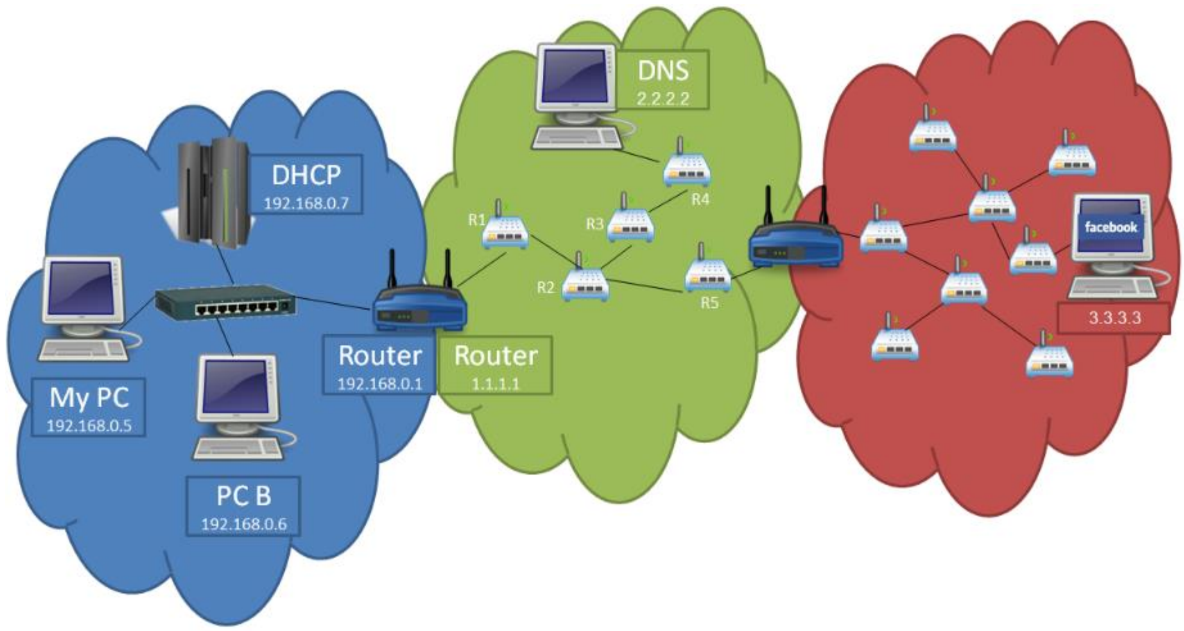


גם במקרה זה, כאשר המחשב ירצה לגשת אל שרת ה-DNS, עליו לבדוק האם השרת נמצא באותה הרשת כשלו. לשם כך, המחשב יבצע בדיקה על מסכת הרשת שלו:



כתובת ה-IP של שרת ה-DNS, אותה מצאנו קודם, היא 2.2.2.2. מכיוון ששני הבתים הראשונים של כתובת זו הם 2.2 ולא 192.168, הרי ששרת ה-DNS לא נמצא באותו ה-Subnet של המחשב.

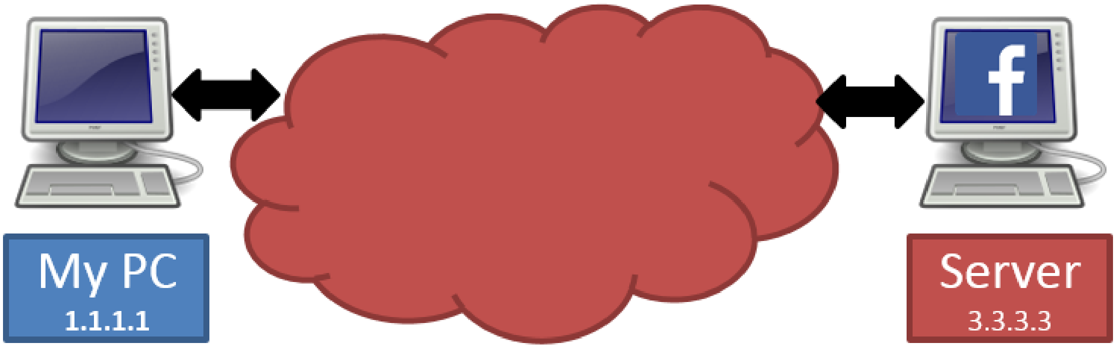
המשך התהליך דומה מאוד לתהליך ללא שימוש ב-NAT, ולכן לא נרחיב עליו כאן. עם זאת, עלינו להבין את ההבדל הנובע מהשימוש בכתובות פרטיות. תמונת הרשת שלנו נראית כך:



שימו לב שלנתב שלנו יש שתי כתובות IP – אחת "פנימית", שהיא הכתובת הפרטית 192.168.0.1, המשמשת אותו ברשת הביתית שלנו, והשנייה "חיצונית", שהיא הכתובת 1.1.1.1 המשמשת אותו אל מול הספקית בפרט והאינטרנט בכלל.

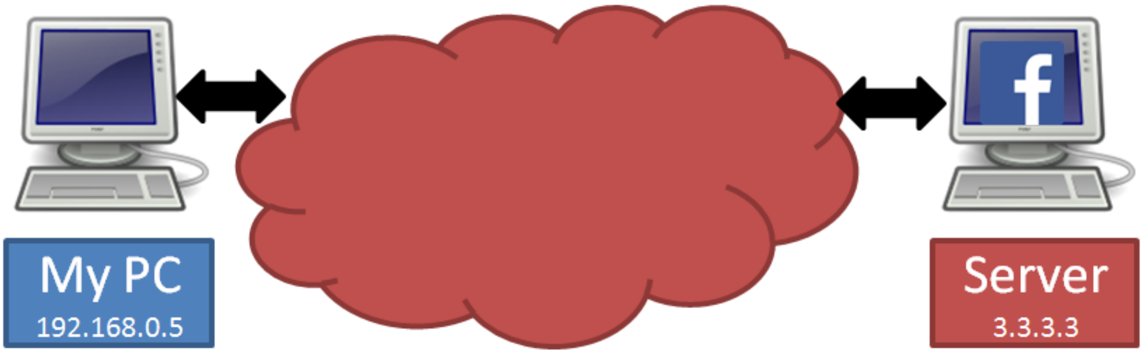
כאשר המחשב שלנו רוצה לתקשר עם Facebook, הוא לא יוכל לעשות זאת ישירות. תקשורת שכזו אינה אפשרית, מכיוון שאם Facebook ינסה לענות אל המחשב שלנו, הוא לא יוכל לשלוח אליו חבילה – הרי שהכתובת 192.168.0.5 הינה כתובת פרטית, והנתבים שבדרך לא יכולים לנתב אליה חבילות.

כאן נכנס לפעולה ה-NAT. כאשר המחשב שלנו ישלח את החבילה, הוא ישלח אותה אל השרת של Facebook באופן רגיל. כעת, כאשר הנתב יקבל את החבילה, הוא יחליף את כתובת המקור של החבילה לכתובת שלו, כאשר החבילה תגיע אל Facebook, היא תיראה כאילו היא הגיעה מהכתובת 1.1.1.1. נראה זאת בשרטוט הבא (שמוצג מנקודת המבט של השרת של Facebook):



החבילה מגיעה בשלב זה אל השרת של Facebook, שלא מודע כלל לכך שהיא נשלחה במקור מהכתובת 192.168.0.5. מבחינתו, החבילה הגיעה פשוט מישות שנמצאת בכתובת 1.1.1.1. לכן, כאשר Facebook עונה חבילת תשובה, הוא שולח אותה אל הכתובת 1.1.1.1. בשלב זה, כאשר הנתב מקבל את החבילה, הוא מחליף את כתובת היעד של החבילה, מהכתובת 1.1.1.1 אל הכתובת 192.168.0.5. לאחר החלפה זו, הוא מעביר אותה אל המחשב שלנו.

מכיוון שהנתב מחליף את הכתובות בטרם הוא מעביר את החבילה, המחשב שלנו כלל לא צריך להיות מודע לתהליך ה-NAT, והוא "שקוף" עבורו. למעשה, עבור המחשב שלנו, התהליך נראה כאילו לא היה NAT בכלל. נראה זאת בשרטוט הבא (שמוצג מנקודת המבט של המחשב שלנו):



בדרך זו, ה-NAT מאפשר חיסכון בכתובות IP מבלי לגרום לשינוי בצד של לקוח הקצה – המחשב שלנו במקרה הזה.

איננו מתעכבים בספר זה על דרכים שונות לממש NAT. עם זאת, אתם מוזמנים להרחיב את הידע שלכם בנושא בעמוד: http://en.wikipedia.org/wiki/Network_address_translation.

איך הכל מתחבר, ואיך עובד האינטרנט – סיכום

בפרק זה חזרנו לאותה השאלה ששאלנו בתחילת הספר – איך האינטרנט עובד? הפעם, מצוידים בכלים וידע שרכשנו לאורך הספר, יכולנו לענות על השאלה בצורה מעמיקה בהרבה משעשינו בפרק הראשון.

התחלנו מהמחשב הבודד שלנו, והצלחנו לקבל **כתובת IP** ואת שאר פרטי הרשת באמצעות פרוטוקול **DHCP**. על מנת למצוא את כתובת ה-IP של Facebook, הבנו שאנו צריכים לפנות לשרת ה-**DNS**. בכדי לעשות זאת, הסתכלנו על **מסכת הרשת (Subnet Mask)** שלנו והבנו ששרת ה-DNS לא נמצא איתנו באותה הרשת. על כן, פנינו אל **טבלת הניתוב** והבנו שעלינו להעביר את החבילה אל ה-**Default Gateway** שלנו.

על מנת לפנות אל הנתב, היינו צריכים לגלות את הכתובת הפיזית שלו, ולשם כך השתמשנו בפרוטוקול **ARP**. לאחר שנזכרנו בפעולה של פרוטוקול זה, חזרנו על הדרך בה ה-**Switch** פועל, וכיצד הוא יודע להעביר כל מסגרת רק אל הפורט הפיזי אליו היא מיועדת. לאחר מכן, הסתכלנו על הכתובות בשכבה השנייה ובשכבה השלישית שיהיו בחבילה שנשלחת אל שרת ה-DNS, וראינו איך הן משתנות לאורך המסלול. כמו כן, הזכרנו כיצד **נתב** מטפל בחבילה שמגיעה אליו.

כשהבנו את הדרך שעושה החבילה באינטרנט, חזרנו על דרך הפעולה של פרוטוקול **DNS**, באמצעותו מצאנו את כתובת ה-IP של Facebook. בשלב זה, היינו צריכים להקים **קישור TCP** עם Facebook, ולכן הזכרנו כיצד נבחרים מספרי ה**פורטים** בהם נעשה שימוש. הקמנו את הקישור באמצעות **Three Way Handshake**, וחזרנו על השדות **Sequence Number** ו-**Acknowledgement Number**. לבסוף, הצלחנו לגשת אל השרת של Facebook מעל קישור ה-TCP שהקמנו באמצעות פרוטוקול **HTTP**, שלחנו בקשה אל השרת וקיבלנו את התשובה.

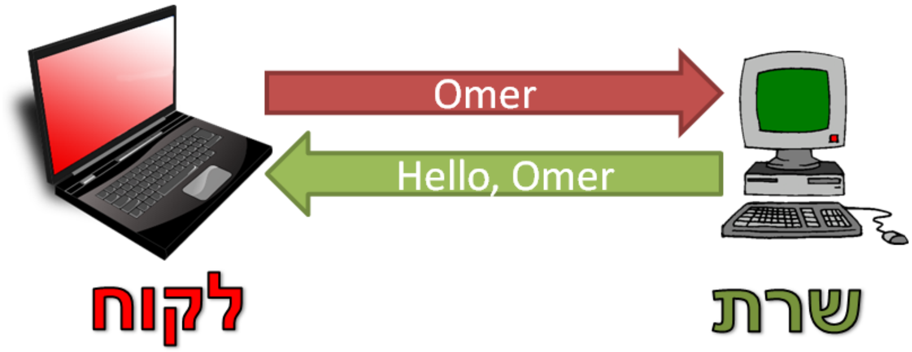
באופן זה נגענו בנושאים רבים במהלך הפרק, ועברנו דרך השכבות השונות של **מודל חמש השכבות**. בפרק זה הצלחנו לחבר יחדיו נושאים שונים שנסקרו לאורך כלל הפרקים הקודמים, ולראות כיצד הם פועלים יחד ברשת האינטרנט. עם זאת, החסרנו נגיעה מעמיקה בקונספטים רבים עליהם הרחבנו במהלך הספר. כמעט ולא נכנסנו לשדות ספציפיים של פרוטוקולים, והחסרנו התייחסות לנושאים חשובים בכל שכבה ולמבנה החבילות. אם תרצו, תוכלו לחזור אל הפרקים הרלוונטיים ולרענן את זכרונכם.

דרכנו עדיין לא הסתיימה. לאחר שלמדנו את מודל חמש השכבות, והרחבנו את היריעה על כל שכבה, ישנם נושאים מתקדמים נוספים עליהם נרחיב בהמשך הספר.

פרק 12

תכנות Sockets מתקדם: ריבוי משתמשים (הרחבה)

בפרק תכנות ב- Sockets למדנו מהו Socket, וכיצד לכתוב לקוח ושרת. כתבנו מספר לקוחות ומספר שרתים, כגון שרת שמקבל מהלקוח את שמו ומחזיר לו תשובה בהתאם (פרק תכנות ב- Sockets /תרגיל 2.3 מודרך – השרת הראשון שלי):



מה יקרה כשנרצה שהשרת שלנו ייתן שירות ליותר מלקוח אחד באותו הזמן? למשל, אם נרצה לפתח אפליקציית צ'אט שמתחברים אליה כמה לקוחות ומדברים ביניהם או לפתח משחק רשת שכמה שחקנים יכולים להשתתף בו במקביל? במילים אחרות, איך נגרום לשרת שלנו להיות שרת מרובה משתתפים?

במקרים מסוימים שרת UDP יכול לתפקד כשרת מרובה משתמשים. למשל, עבור שיחות וידאו מרובות משתתפים כמו זום, פרוטוקול UDP ייתן מענה טוב. אבל שימוש בפרוטוקול UDP לא מספיק טוב לנו לכל שרת מרובה משתתפים. תחשבו על זה: מה יקרה אם נפתח שרת להודעות (צ'אט) שמבוסס על טקסטים ויהיו לקוחות שלא יקבלו חלק מהטקסטים? מה אם נרצה שהלקוחות יעבירו קבצים לשרת? במקרים כאלו ועוד רבים העברה אמינה של מידע וניהול שיחה חשובים לנו מאוד, ולכן נרצה להשתמש בפרוטוקול TCP. וכאן מתעוררת בעיה באופן שבו עובדים סוקטים כפי שאנחנו מכירים אותם.

שרת מסוג TCP מחכה לחיבורים חדשים עד שהוא מקבל התחברות מלקוח חדש. בשלב זה הוא נקשר ללקוח שהתקשר איתו והוא מטפל רק בו עד שאחד הצדדים יחליט לסיים את השיחה. אבל מה יקרה אם השיחה תימשך דקות ארוכות, אפילו שעות, ובזמן הזה לקוחות אחרים ינסו להתחבר לשרת? במצב כזה יכול להיווצר "לחץ" ותור ארוך של לקוחות שמחכים להתחבר לשרת. בכל פעם השרת מדבר עם לקוח אחד ובדרךצורה זוכזאת אי אפשר לאפשר ללקוחות אחרים לא יכולים לתקשר עם השרת לתקשר אחד עם השני.,
כך שבאופן בו אנחנו מכירים סוקטים, אי אפשרלא ניתן לממש שרת מרובה משתמשים.

כדי לכתוב שרת מסוג TCP שיתמוך בכמה משתמשים בו-זמנית, עלינו להכיר טכניקה חדשה לניהול סוקטים בעזרת פונקציה הנקראת select.

שירות לכמה לקוחות במקביל

מדוע זו בעיה?

ובכן, מדוע אנו משקיעים פרק שלם בכדי לדון בסוגיה הזו? האם הפתרון לא כולל רק להוסיף לולאה לקוד?

הסוגיה הבעייתית הראשונה נובעת מהצורך ליצור חיבור מול לקוח חדש. להזכירכם, כאשר רצינו לקבל חיבור מלקוח חדש בעת שמימשנו שרת שמטפל בבקשה אחת בכל פעם, השתמשנו במתודה **accept** בצורה הבאה:

```
(client_socket, client_address) = server_socket.accept()
```

כפי שציינו בפרק תכנות ב-Sockets, המתודה **accept** הינה blocking – כלומר, הקוד "יקפא" ולא ימשיך לרוץ עד אשר יתקבל בשרת חיבור חדש. מכאן שלאחר שקיבלנו חיבור מלקוח ראשון, עלינו להחליט בין שתי אפשרויות:

- לטפל בבקשות שמגיעות מהלקוח הראשון.
- לאפשר ללקוח חדש להתחבר.

הסיבה לכך נעוצה בעובדה, שעל מנת לאפשר ללקוח חדש להתחבר, עלינו לקרוא שוב למתודה **accpet**, אשר עוצרת את ריצת התוכנית עד אשר יתחבר לקוח חדש.

סוגיה בעייתית נוספת קשורה לקריאת מידע מלקוח קיים. כאשר רצינו לקרוא מידע מלקוח בעת שמימשנו שרת שמטפל רק בבקשה אחת, השתמשנו במתודה **recv**:

```
client_name = client_socket.recv(1024)
```

נזכיר כי גם המתודה **recv** הינה blocking – ולא תאפשר את המשך ריצת התוכנית עד אשר נקבל מידע מהלקוח. מה יקרה במידה שננסה לקרוא מידע, אך הלקוח לא ישלח אלינו דבר? שוב, התוכנית תיתקע ולא נוכל לטפל בלקוחות נוספים. אי לכך, כאשר נקרא ל-**recv**, לא נוכל לאפשר ללקוח חדש להתחבר לשרת, או לחלופין – לקבל מידע מלקוח אחר המעוניין לכתוב אלינו.

הפתרון – Select

כדי שנוכל להבין את האופן שבו מתנהל שרת מרובה משתתפים, נשתמש בדוגמה מהחיים. דמיינו ברמן שעובד בפאב.

צילום מסך מתוך המשחק Beertender

לפאב שלנו יש כמה כסאות. מהם הדברים שלקוחות יכולים לעשות?

- לקוח חדש שנכנס לפאב יכול לעמוד ליד דלת הכניסה ולסמן לברמן. אם יש מקום פנוי הברמן יושיב אותו.
- לקוח שכבר יושב יכול לבקש מהברמן שירות.
- לקוח שסיים לשתות יכול לבקש חשבון ולצאת, לפנות את כסאו

מהי עבודתו של הברמן?

הברמן צריך לסרוק את הפאב כל הזמן ולהענות ללקוחות:

- לקוח חדש צריך מקום לשבת
- לקוח שמבקש להזמין צריך לקבל שירות
- לקוח שמבקש לעזוב צריך לקבל חשבון ויש צורך לפנות את המקום שלו לטובת הלקוח הבא שממתין

מבחינות רבות הפונקציה Select מבצעת פעולות דומות לפעולות של הברמן. הפונקציה מבצעת מעקב אחר הסוקטים שמבקשים להתחבר ומדווחת לנו ברגע שאחד מהסוקטים מוכן לקריאה. הפונקציה מוגדרת כך:

```
ready_to_read, ready_to_write, in_error  = select.select(read_list, write_list, error_list)
```

הפונקציה מקבלת שלוש רשימות ומחזירה שלוש רשימות. הפרמטרים שמקבלת הפונקציה הן הרשימות הבאות:

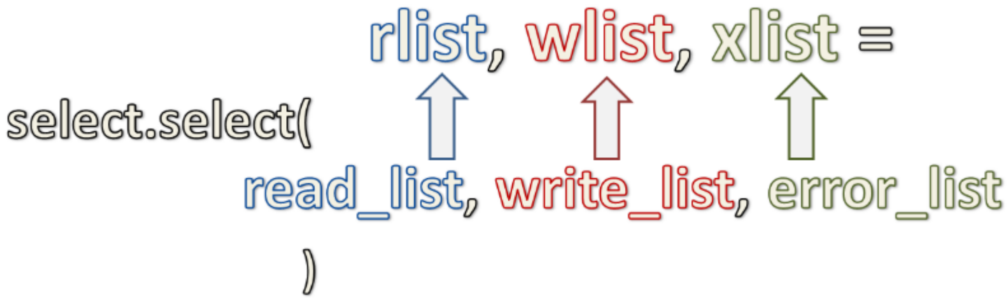
- רשימת ה-Sockets מהם אולי נרצה לקרוא.
- רשימת ה-Sockets אליהם אולי נרצה לכתוב.
- רשימת ה-Sockets עבורם נרצה לבדוק מקרים של שגיאות. לצורך הפשטות, נתעלם כרגע מרשימה זו.

כל אובייקט **socket** יכול להיכנס לתוך אחת או יותר מהרשימות הללו.

לאחר ש-**select** מסיימת את הריצה שלה, היא מחזירה שלוש רשימות:

- רשימת ה-Sockets מהם ניתן כרגע לקרוא (באמצעות **recv**).
- רשימת ה-Sockets אליהם ניתן כרגע לשלוח (באמצעות **send**).
- רשימת ה-Sockets שזרקו שגיאה כלשהי.

כל אחת מהרשימות הללו כוללת Sockets מתוך הרשימה התואמת שהעברנו:



נתחיל מהפרמטר הראשון, read_list. זוהי רשימת כל הסוקטים שאנחנו רוצים לסרוק. נשאלת השאלה: מהם בעצם הסוקטים שאנחנו רוצים לסרוק?

כפי שכבר הבנו, ישנם הלקוחות הקיימים (אלה שכבר יושבים בפאב). נגדיר רשימה בשם client_sockets ובהמשך נשמור בה את כל הסוקטים של הלקוחות הקיימים. בנוסף ישנם הסוקטים של הלקוחות שרוצים להצטרף, שהברמן עדיין לא הושיב. אלו מגיעים אלינו מסוקט השרת. לכן, החיבור של כל הסוקטים הללו לרשימה אחת יראה כך:

```
[server_socket] + client_sockets
```

אלו כל הסוקטים שאנחנו רוצים להעביר ב-read_list ולסרוק אותם.

קריאה לפונקציה select

למען הפשטות, נתמקד בפרמטר הראשון – read_list ונתעלם משני הפרמטרים האחרים, במקומם נעביר רשימה ריקה. הקריאה שלנו ל-select תיראה כך:

```
rlist, wlist, xlist = select.select([server_socket] + client_sockets,
                                                    [], [])
```

ברגע שהפונקציה תסתיים נתמקד בערך הראשון שחזר, הרשימה rlist. רשימה זו כוללת את כל הסוקטים שהשרת צריך לטפל בהם, כלומר אלו שסימנו לנו שיש בהם מידע. בתוכם יכולים להופיע סוקטים של לקוחות קיימים שקורה איתם משהו חדש, או לקוחות חדשים שרוצים להתחבר לשרת.

כדי לטפל בלקוחות האלה נוכל ליצור לולאה פשוטה שעוברת על רשימת הלקוחות, והיא תראה כך:

```
for current_socket in rlist:
    # Do something for every client
```

כך למשל, באם העברנו ברשימת ה-Sockets מהם אולי נרצה לקרוא (בשרטוט לעיל: read_list) את האובייקטים client_1_socket ו-client_2_socket, ייתכן שהפונקציה **select** תחזיר ברשימת ה-Sockets מהם ניתן כרגע לקרוא (בשרטוט לעיל: rlist) את האובייקט client_1_socket בלבד. דבר זה מציין כי אנו יכולים לבצע **recv** על האובייקט client_1_socket מבלי "לתקוע" את שאר ריצת התוכנית, אך לא על client_2_socket.

תרגיל 12.1 מודרך – השרת מרובה המשתתפים הראשון שלי

בתרגיל זה נממש שרת שמקבל חיבור מלקוח, קורא שם כלשהו מהלקוח, ומדפיס שם זה למסך. עם זאת, בניגוד לשרת שמימשנו בפרק תכנות ב-Sockets, השרת יוכל לטפל במספר לקוחות במקביל. מעבר לכך, כל לקוח יוכל לשלוח בכל פעם שם אחר. השרת ידפיס למסך כל שם שהוא מקבל:

שימו לב שהטיפול במקרה זה צריך להיות מקבילי – כלומר, השרת יוכל להשאיר את החיבור עם הלקוח הראשון פתוח בעודו מספק שירות ללקוח השני.

על מנת לעשות זאת, נתחיל בדרך דומה לזו שעשינו עד כה – ניצור אובייקט מסוג **socket**:

```python
import socket
import select

MAX_MSG_LENGTH = 1024
SERVER_PORT = 5555
SERVER_IP = "0.0.0.0"

server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
server_socket.bind((SERVER_IP, SERVER_PORT))
server_socket.listen()
```

שורת הקוד היחידה ששונה משרתים שתכננו עד כה היא ה- import select.

כעת נתחיל בשלבים שיהיו שונים.

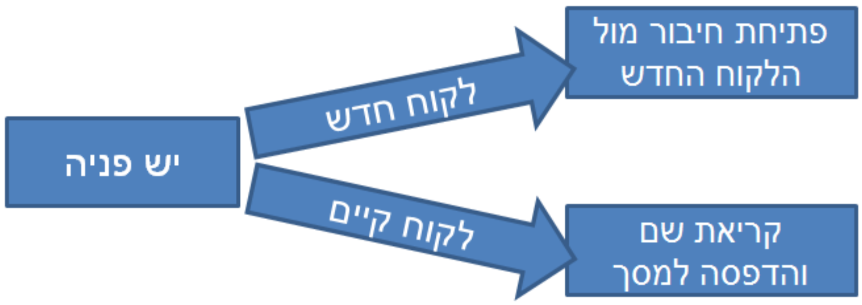
ניצור רשימה שתכיל את כל אובייקטי ה-**socket** של הלקוחות שיתחברו לשרת:

open_client_sockets = []

בשלב זה, עלינו לטפל בפניות המגיעות אל השרת. לשם כך, ניצור לולאה שבכל פעם:

- אם יש פנייה מלקוח חדש – תפתח מולו חיבור (באמצעות **accept**).
- אם יש פנייה מלקוח קיים – תקרא ממנו את שמו, ותדפיס אותו למסך (באמצעות **recv**).

כלומר, בכל ריצה של הלולאה, על הסקריפט לבצע את הלוגיקה הבאה עבור כל פנייה:



כזכור, כל לקוח יוכל לכתוב שם שונה בכל פעם, מבלי לסגור את החיבור:

הלולאה המתוארת לעיל תרוץ באופן תמידי, ולכן נכתוב:

while True:

בתוך לולאת ה-while, נקרא לפונקציה בצורה הבאה:

rlist, wlist, xlist = select.select([server_socket] + client_sockets, [], [])

כעת הקוד שלנו יראה כך:

```
import socket
import select

MAX_MSG_LENGTH = 1024
SERVER_PORT = 5555
SERVER_IP = "0.0.0.0"

print("Setting up server...")
server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
server_socket.bind((SERVER_IP, SERVER_PORT))
server_socket.listen()
print("Listening for clients...")
client_sockets = []

while True:
    rlist, wlist, xlist = select.select([server_socket] + cli
                                        ent_sockets, [], [])
```

השורה האחרונה היא שורת המפתח, ולכן נתעכב בכדי להבין אותה. כאמור, הפונקציה **select** מחזירה שלוש רשימות:

- רשימת ה-Sockets מהם ניתן כרגע לקרוא – תישמר למשתנה **rlist**.
- רשימת ה-Sockets אליהם ניתן כרגע לשלוח – תישמר למשתנה **wlist**.
- רשימת ה-Sockets שזרקו שגיאה כלשהי – תישמר למשתנה **xlist**.

הרשימות נבנות מתוך הקלט של הפונקציה. כך למשל, אל המשתנה **wlist** יישמרו Sockets אליהם ניתן כרגע לשלוח, מתוך רשימת ה-Sockets שניתנו לפונקציה **select** בתור הפרמטר השני. היות שהפרמטר השני והשלישי הם רשימה ריקה ([]), הרי ש-**wlist** ו-**xlist** תהיינה לעולם ריקות.

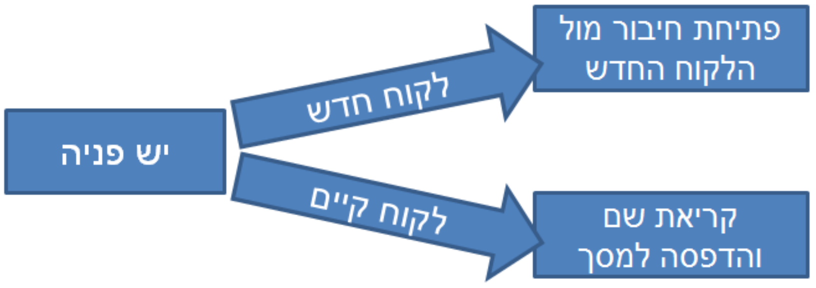
כעת נתמקד במשתנה **rlist**, אליו כאמור תישמר רשימת ה-Sockets מהם ניתן כרגע לקרוא. אנו שולחים אל הפונקציה **select** את ה-Sockets הבאים:

- ה-Socket של השרת המאזין. במידה שניתן לקרוא מ-Socket זה, המשמעות היא שיש פנייה של לקוח חדש. כלומר, במקרה זה, ניתן לקרוא ל-**accept** ולהקים חיבור עם הלקוח החדש.
- כלל ה-Sockets של הלקוחות. במידה שניתן לקרוא מ-Socket של לקוח, הרי שהוא שלח מידע. כלומר, במקרה זה, ניתן לקרוא ל-**recv** ולקבל מידע מלקוח קיים.

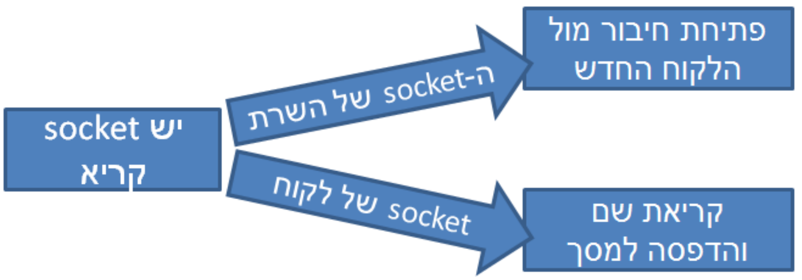
שימו לב כי על מנת לעשות זאת, השתמשנו בשרשור רשימות של פייתון:

[server_socket] + open_client_sockets

את הלוגיקה שציירנו קודם:



ניתן למעשה לרשום גם בצורה הבאה:



כלומר, עלינו להבין עבור כל Socket קריא, האם הוא ה-Socket של השרת או של לקוח קיים, ולפעול בהתאם. לשם כך, נעבור על כל ה-Sockets הקריאים:

```
for current_socket in rlist:
```

נבדוק אם ה-Socket הנוכחי הוא של השרת:

```
if current_socket is server_socket:
```

אם כן, הרי שיש להרים מולו חיבור:

```
connection, client_address = current_socket.accept()
print("New client joined!", client_address)
```

לנוחותנו גם הדפסנו הודעה על חיבור הלקוח החדש.

כמו כן, עלינו להוסיף אותו לרשימת הלקוחות, כדי שנוכל לקבל ממנו מידע בעתיד:

```
client_sockets.append(connection)
```

אם לא, בשלב זה רק נדפיס שקיבלנו מידע מלקוח קיים:

```
else:
       print("Data from existing client\n")
```

לולאת ה-while המלאה שלנו נראית בשלב זה כך:

```
while True:
    rlist, wlist, xlist = select.select([server_socket] + cli
                                        ent_sockets, [], [])
    for current_socket in rlist:
        if current_socket is server_socket:
            connection, client_address = current_socket.accept()
            print("New client joined!", client_address)
            client_sockets.append(connection)
        else:
            print("Data from existing client\n")
```

חשבו מה קורה כאשר השרת מופעל בפעם הראשונה. בתחילה, נקראת הפונקציה **select**, והיא ממשיכה לרוץ עד אשר מגיע לקוח חדש ומתחבר אל השרת. כאשר לקוח חדש יתחבר, הפונקציה **select** תחזיר אל המשתנה **rlist** רשימה שמכילה את ה-Socket של השרת (**server_socket**). בשלב זה, השרת יקים את החיבור מול הלקוח באמצעות המתודה **accept** ולאחר מכן יוסיף אותו אל הרשימה **open_client_sockets**.

בפעם הבאה שתרוץ לולאת ה-while, הפונקציה **select** תחזור כאשר הלקוח שלח מידע לשרת (בהנחה שלא התחבר בינתיים לקוח אחר). הפעם, היא תחזיר אל המשתנה **rlist** רשימה שמכילה את ה-Socket בין הלקוח לשרת, אותו Socket שהתווסף קודם לכן לרשימה **open_client_sockets**. בשלב זה, תודפס למסך ההודעה: "Data from existing client"

לאחר שהבנו את דרך הפעולה של הלולאה שלנו, הגיע הזמן לשפר אותה כך שהיא תטפל במידע שהגיע מהלקוח. בתור התחלה, עלינו לקרוא את המידע:

```
data = current_socket.recv(MAX_MSG_LENGTH).decode()
```

כפי שלמדנו בפרק תכנות ב- Sockets, יש להבין האם התקבלה מחרוזת ריקה ("") והחיבור נסגר:

```
if data == "":
    print("Connection closed", )
    client_sockets.remove(current_socket)
    current_socket.close()
```

אם התקבל מידע תקין, ניתן להדפיס אותו למסך:

```
else:
    print(data)
```

כך נראה בשלב זה כל הקוד שכתבנו:

```python
import socket
import select

MAX_MSG_LENGTH = 1024
SERVER_PORT = 5555
SERVER_IP = "0.0.0.0"

print("Setting up server...")
server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
server_socket.bind((SERVER_IP, SERVER_PORT))
server_socket.listen()
print("Listening for clients...")
client_sockets = []

while True:
    rlist, wlist, xlist = select.select([server_socket] + cli
                                         ent_sockets, [], [])
    for current_socket in rlist:
        if current_socket is server_socket:
            connection, client_address = current_socket.accept()
            print("New client joined!", client_address)
            client_sockets.append(connection)
        else:
            data = current_socket.recv(MAX_MSG_LENGTH).decode()
            if data == "":
                print("Connection closed", )
                client_sockets.remove(current_socket)
                current_socket.close()
            else:
                print(data)
```

תרגיל 12.2 – לקוח לשרת מרובה משתתפים

בתרגיל זה תכתבו לקוח על מנת לבדוק את השרת שכתבנו בתרגיל המודרך הקודם. כתבו לקוח אשר:

- מתחבר אל השרת שיצרתם.
- מבצע בלולאה אינסופית:
  - מקבל שם מהמשתמש (באמצעות **input**).
  - שולח את השם אל השרת.
  - אם השם הוא באורך אפס (כלומר המשתמש הקיש Enter בלי להזין שם כלשהו), הלקוח יסיים את ריצתו וייסגור את ה-Socket.

הפעילו את סקריפט הלקוח מספר פעמים במקביל, וכתבו אל השרת שמות שונים, בכל פעם מלקוח אחר. ודאו כי השרת מצליח להדפיס את ההודעות מהלקוחות שלכם.

כיוון ש- Pycharm אינו מאפשר להריץ את אותו הסקריפט יותר מפעם אחת, לכן נריץ את כל ה-clients שלנו דרך שורת הפקודה. פעלו לפי השלבים הבאים:

א. הפעילו את שורת הפקודה CMD

ב. ב-cmd הקלידו את הפקודה cd כדי להגיע אל התיקייה שבה נמצא הסקריפט של ה-client שלכם, לדוגמה cd c:\networks\work

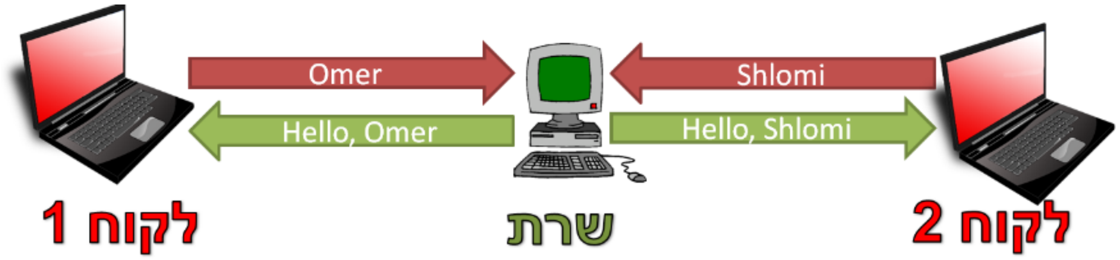
ג. הקלידו את הפקודה python client.py (בהנחה שלסקריפט הלקוח שלכם קוראים client)

ד כדי להריץ מספר רב של לקוחות, חזרו על שלבים ב עד ד כמה פעמים כרצונכם.

תרגיל 12.3 מודרך – שרת מרובה משתתפים עם תשובה ללקוחות

עד כה הצלחנו לקבל מידע ממספר לקוחות ממקביל. יכולת זו אמנם חשובה, אך אינה מספיקה – עלינו גם להצליח לתת שירות ללקוחות, כלומר לשלוח מידע אליהם.

בתרגיל זה נממש שרת שמקבל חיבור מלקוח, מקבל את שמו של הלקוח, ועונה לו בהתאם. עם זאת, בניגוד לשרת שמימשנו בפרק תכנות ב- Sockets /תרגיל 2.3 מודרך – השרת הראשון שלי, השרת יוכל לטפל במספר לקוחות במקביל.



שימו לב שהטיפול במקרה זה צריך להיות מקבילי – כלומר, השרת יוכל להשאיר את החיבור עם הלקוח הראשון פתוח בעודו מספק שירות ללקוח השני.

לצורך התרגיל, נסתמך על הקוד שכתבנו בתרגיל המודרך הקודם, בו ביצענו רק קריאה של נתונים מהלקוח. נשנה את הקוד באזור שטיפל בהודעה שהתקבלה מלקוח קיים. במימוש הקודם, הקוד גרם להדפסת ההודעה למסך (מסומן באדום ו**בהדגשה**):

```
if data == "":
    print("Connection closed", )
    client_sockets.remove(current_socket)
    current_socket.close()
```

```
else:
    print(data)
```

הפעם, נרצה לשלוח את המידע חזרה אל הלקוח. עם זאת, אנחנו לא יכולים פשוט להשתמש במתודה **send** בשלב זה.

מדוע לא ניתן פשוט לשלוח את המידע?

נסו לחשוב על כך בעצמכם בטרם תקראו את השורה הבאה.

התשובה נעוצה בכך שלא בטוח שאותו Socket שעכשיו קראתי ממנו את המידע מוכן לכך שנשלח לו את המידע. במקרה כזה, הפונקציה **send** עלולה להיתקע, ולא נוכל לתת שירות לשאר הלקוחות. כאן אנחנו רואים מקרה פשוט, בו השרת מגיב לכל לקוח בנפרד, אך בהמשך נראה מקרים מורכבים יותר בהם חשוב במיוחד לוודא שכל Socket אליו אנו מעוניינים לשלוח מידע יהיה במצב שמוכן לשליחה.

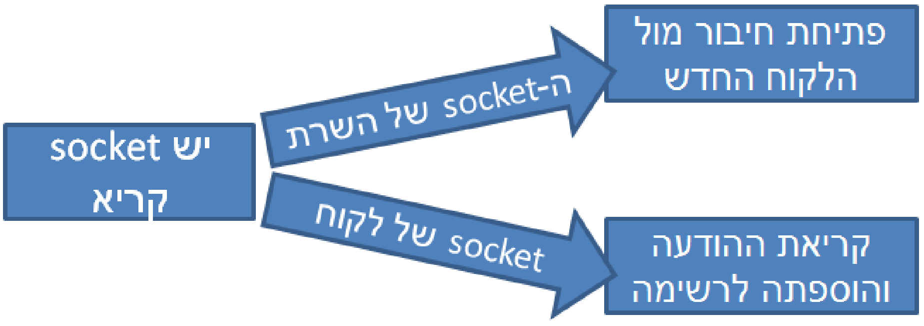
אי לכך, עלינו להוסיף את ההודעה לרשימה שתכיל את כל ההודעות שיש לשלוח, ולאחר מכן לשלוח אותה כשניתן יהיה לעשות זאת (מסומן ב**אדום** ו**בהדגשה**):

```
if data == "":
    print("Connection closed", )
    client_sockets.remove(current_socket)
    current_socket.close()
else:
    messages_to_send.append((current_socket, data))
```

שימו לב שהוספנו כאן לרשימה אובייקט מסוג tuple שמכיל את ה-Socket שאליו יש לשלוח את ההודעה, ואת תוכן ההודעה. מן הסתם, יהיה עלינו להגדיר את הרשימה לפני שנוסיף איבר לתוכה, ולכן נעשה זאת לפני לולאת ה-while :

```
messages_to_send = []
while True:
        (קוד הלולאה שתיארנו קודם) ...
```

זו הלוגיקה שמתארת את ההתנהגות הנוכחית של הלולאה:



בנוסף, בכל איטרציה של הלולאה, ננסה לשלוח את כל ההודעות שעדיין לא נשלחו. נעשה זאת באמצעות לולאה אשר תעבור על כל ההודעות שנמצאות ברשימת ההודעות המיועדות לשליחה ושלח אותן. שיום לב לכך שלא ניתן לשלוח אותן סתם כך, לפני כן צריך לוודא שהלקוח אכן מוכן לקבל את ההודעה. כזכור רשימת הלקוחות שמוכנים לקבל הודעות נמצאת במשתנה **wlist**:

```
for message in messages_to_send:
    current_socket, data = message
    if current_socket in wlist:
        current_socket.send(data.encode())
        messages_to_send.remove(message)
```

שימו לב ששורת הקוד האחרונה מסירה את ההודעה מרשימת ההודעות המיועדות לשליחה, כיוון שאם הגענו לשורת קוד זו סימן שההודעה נשלחה.

עם זאת, כאשר מימשנו את השרת שרק קרא את המידע מהלקוחות, אמרנו שהרשימה wlist תמיד תהיה ריקה, כיוון שהעברנו לפונקציה **select** רשימה ריקה בתור הפרמטר השני:

```
rlist, wlist, xlist = select.select([server_socket] + client_sockets,
                                    [], [])
```

נשנה זאת אפוא ונעביר לפונקציה **select** את רשימת כל הלקוחות, כדי שתחזיר לנו אל המשתנה **wlist** את רשימת הלקוחות שכרגע ניתן לשלוח אליהם מידע:

```
rlist, wlist, xlist = select.select([server_socket] + client_sockets,
                                    client_sockets, [])
```

תוספת לא הכרחית אך חביבה שנעניק לשרת היא הדפסה של רשימת כל ה-Socketים המחוברים אליו. לשם כך נכתוב פונקציה בשם print_client_sockets שתקבל בתור פרמטר את רשימת כל הלקוחות ותשתמש במתודה getpeername שקיימת לכל אובייקט מסוג Socket. מתודה זו מדפיסה את כתובת ה-IP והפורט של ה-Socket שביצע את ההתחברות:

```
def print_client_sockets(client_sockets):
    for c in client_sockets:
        print("\t", c.getpeername())
```

הקוד המלא של השרת נראה כך:

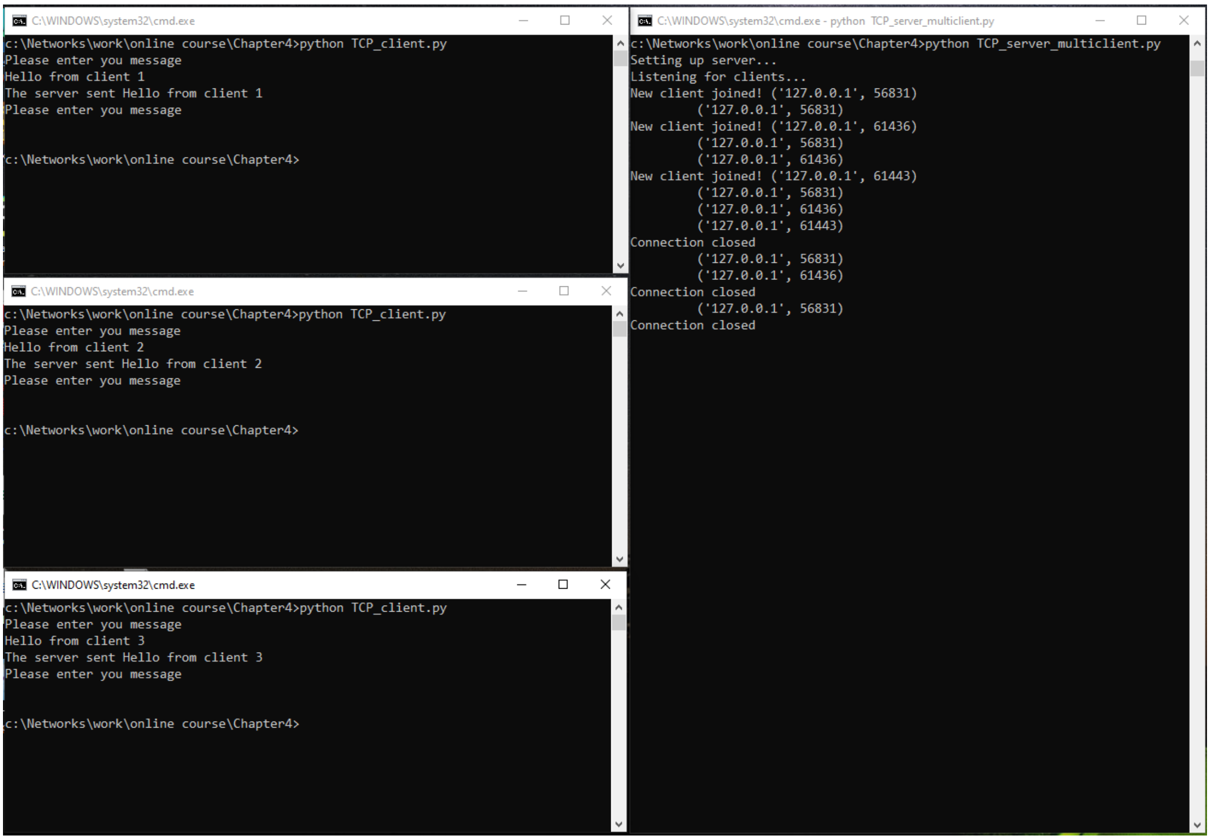
```
import socket
import select

MAX_MSG_LENGTH = 1024
SERVER_PORT = 5555
SERVER_IP = '0.0.0.0'


def print_client_sockets(client_sockets):
    for c in client_sockets:
        print("\t", c.getpeername())

print("Setting up server...")
server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
server_socket.bind((SERVER_IP, SERVER_PORT))
server_socket.listen()
print("Listening for clients...")
client_sockets = []
messages_to_send = []

while True:
    rlist, wlist, xlist = select.select([server_socket] + cli-
ent_sockets, client_sockets, [])
    for current_socket in rlist:
        if current_socket is server_socket:
            connection, client_address = current_socket.accept()
            print("New client joined!", client_address)
            client_sockets.append(connection)
            print_client_sockets(client_sockets)
        else:
            data = current_socket.recv(MAX_MSG_LENGTH).decode()
            if data == "":
                print("Connection closed", )
                client_sockets.remove(current_socket)
                current_socket.close()
                print_client_sockets(client_sockets)
            else:
                messages_to_send.append((current_socket, data))

    for message in messages_to_send:
        current_socket, data = message
        if current_socket in wlist:
            current_socket.send(data.encode())
            messages_to_send.remove(message)
```

תרגיל 12.4 – לקוח לשרת מרובה משתתפים שקורא מידע מהשרת

כעת עליכם לבדוק את הקוד של השרת הקודם שכתבנו. היעזרו בלקוח שכתבתם בתרגיל הלקוח שרק שולח מידע אל השרת (פרק תכנות Sockets מתקדם: ריבוי משתמשים/ תרגיל 12.2 – לקוח לשרת מרובה משתתפים). שפרו את הלקוח כך שגם יקרא מידע שנשלח אליו מהשרת וידפיס אותו למסך.

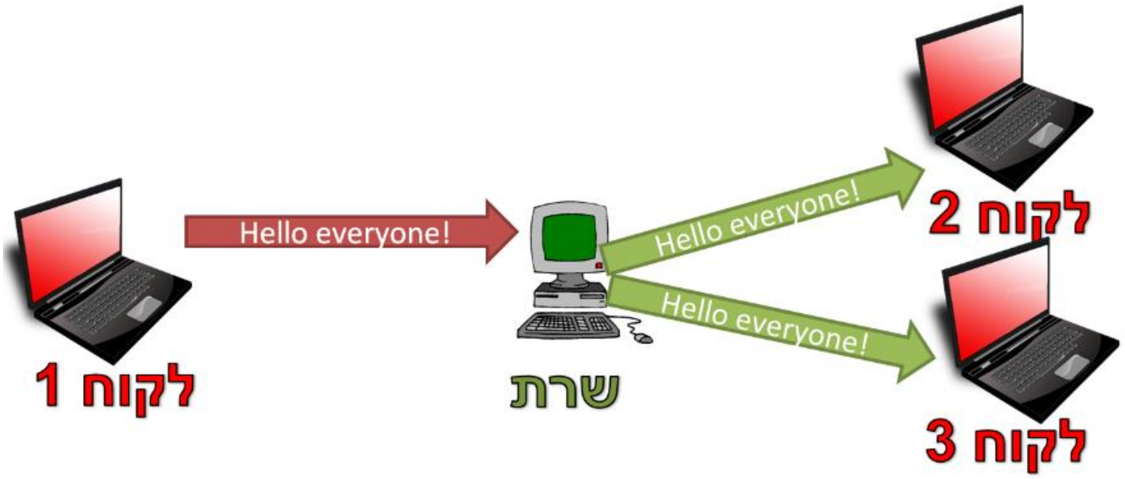
הפעילו את סקריפט הלקוח מספר פעמים במקביל, וכתבו אל השרת שמות שונים, בכל פעם מלקוח אחר. וודאו כי כל לקוח מקבל הודעה נכונה בהתאם לשם שהוא שלח לשרת.

דוגמת הרצה של השרת מול שלושה לקוחות עשויה להיראות כך:



תרגיל 12.5 – צ'אט מרובה משתתפים

בתרגיל זה תשתמשו בידע שרכשתם במהלך הפרק על מנת לממש צ'אט מרובה משתתפים. עליכם לממש **גם את השרת וגם את הלקוח**. כל לקוח יכול לכתוב לשרת איזה מידע שהוא רוצה. השרת ידאג לשלוח לשאר הלקוחות את ההודעה שלו (ולא אליו בחזרה). כל לקוח רואה את כל ההודעות של הלקוחות האחרים, אך לא יכול לדעת ממי הגיעה ההודעה:



<u>הנחיות לתרגיל</u>

יש להשתמש בתיקייה socket, ולא בספריות עזר כגון SocketServer.

<u>צד השרת</u>

השרת צריך לשלוח את ההודעה לכל הלקוחות **מלבד** לזה ששלח אותה אליו. מקרה זה שונה מהשרתים שמימשנו קודם, ומורכב יותר. בכל פעם שתנסו לשלוח את ההודעה, עליכם לזכור למי כבר הצלחתם לשלוח אותה, ולא לשלוח פעמיים את אותה ההודעה לאותו לקוח.

<u>צד הלקוח</u>

סקריפט הלקוח צריך להצליח גם לקרוא מידע מהמשתמש, במידה שהוא רוצה לשלוח הודעה לשאר המשתמשים, וגם לקרוא מידע מהשרת. פעולות אלו צריכות להתבצע במקביל, ולכן אסור להן להיות חוסמות (blocking).

בכל הפעמים בהן מימשנו לקוח, על מנת לקרוא מידע מהשרת, השתמשנו פשוט במתודה **recv**. עם זאת, המתודה היא blocking, ולכן לא תוכלו פשוט להשתמש בה. במקרה זה, יהיה עליכם להשתמש ב-**select** בדומה לצד השרת.

כמו כן, בכל הפעמים בהן מימשנו לקוח, קראנו מהמשתמש באמצעות הפונקציה **raw_input**. גם פונקציה זו היא blocking, ועל כן לא נשתמש בה. על מנת לקרוא באופן שאינו blocking, קראו על המודול **msvcrt** (הכלול בהתקנה הסטנדרטית של פייתון). באופן ספציפי, תוכלו להשתמש בפונקציות **kbhit** ו-**getch** של מודול זה.

תרגיל 12.6 – צ'אט מתקדם

בתרגיל הקודם כתבתם צ'אט מרובה משתתפים. הצ'אט אמנם אפשר למשתמשים לתקשר, אך לא בצורה נוחה. אף משתמש לא יכול היה לדעת מי המשתמש שכתב את ההודעה או מתי כל הודעה נשלחה. מעבר לכך, חסרו לצ'אט הרבה מהיכולות שיש לצ'אט אמיתי. בתרגיל זה, תכתבו צ'אט הרבה יותר מעניין.

את הצ'אט עליכם לכתוב באופן מדורג. כלומר, בכל פעם עבדו רק על סעיף אחד. רק לאחר שהשלמתם סעיף מסוים ובדקתם כי הצ'אט פועל בהתאם למצופה, עברו לסעיף הבא.
שימו לב לקרוא את תיאור הפרוטוקול שמופיע לאחר רשימת הסעיפים, ולהסתמך עליו לאורך התרגיל כולו.

סעיפים

1. כל הודעה תתחיל עם שם המשתמש שכתב אותה, לדוגמה:

talmid1: Hello everyone!

2. ליד כל הודעה תירשם השעה שבה היא נכתבה, לדוגמה:

08:02 talmid1: Hello, the lesson is about to start.

3. אם משתמש שולח את המחרוזת "quit" – על השרת לנתק אותו.
4. במקרה שמשתמש עזב את הצ'אט (השרת ניתק אותו, או הוא החליט לעזוב בעצמו), על השרת לכתוב לכולם שהוא יצא מהצ'אט. למשל:

09:45 talmid5 has left the chat!

5. אפשרו למשתמשים מסוימים להיות מנהלים. רשימת המנהלים תירשם באופן hard-coded אצל השרת (כלומר, בתוך רשימה קבועה בקוד). מנהל יכול להעיף משתמש אחר מהצ'אט.
כאשר מנהל מעיף משתמש, תשלח לכולם ההודעה:

09:47 talmid6 has been kicked from the chat!

6. ליד שם משתמש של מנהל יש להופיע הסימול '@'. לדוגמה:

10:05 @manager3: Woohoo, I am a manager.

7. דאגו לכך שמשתמשים לא יוכלו להשתמש בשם שמתחיל ב-'@', כדי לא לגרום לאחרים לחשוב שהם מנהלים למרות שהם לא.
8. אפשרו למנהלים למנות באופן דינאמי מנהלים נוספים. **שימו לב:** מי שאינו מנהל לא יכול למנות מנהלים.

9. תנו למנהלים את האפשרות להשתיק משתמש. הכוונה היא שהמשתמש יוכל לראות מה כתוב בצ'אט, אך לא יוכל לכתוב. אם הוא כותב, הוא יקבל תשובה – "You cannot speak here." עם זאת, משתמש מושתק יכול לעזוב את הצ'אט, ואז השרת עדיין יכתוב לכולם שהוא יצא מהצ'אט.
10. אפשרו שליחת הודעות פרטיות בין שני משתמשים (במקרה של הודעה פרטית, רק שני המשתמשים הרלוונטיים יראו את ההודעה, ולא אף אחד אחר). כל הודעה פרטית תתחיל בסימול "!", לדוגמה:

09:42 !talmid2: This is a private message.

11. אם לקוח שולח את המחרוזת "view-managers", תוחזר אליו רשימת כל המנהלים.

תיאור הפרוטוקול

צד הלקוח

על כל הודעה שהלקוח שולח להיות במבנה הבא:

שדה	סוג	דוגמה
אורך שם המשתמש	מספר	6
שם המשתמש	מחרוזת	talmid
פקודה לשימוש (פירוט בהמשך)	מספר	1
פרמטרים נוספים בהתאם לפקודה	משתנה	Hello world

הפקודות שניתן לשלוח הן:

מספר פקודה	משמעות	פרמטרים נוספים	דוגמה
1	הודעת צ'אט	אורך ההודעה, ההודעה	5, hello
2	מינוי מנהל	אורך שם המנהל, שם המנהל	7, manager
3	העפת משתמש	אורך שם המשתמש, שם המשתמש	7, talmid2
4	השתקת משתמש	אורך שם המשתמש, שם המשתמש	7, talmid3
5	הודעה פרטית בין משתמשים	אורך שם המשתמש שאליו רוצים לשלוח את ההודעה, שם המשתמש, אורך ההודעה, ההודעה	7, talmid4, 5, hello

דוגמה להודעת צ'אט:

4Omer118This is an example

הסבר:

- **באדום** (תו 1 משמאל) – אורך שם המשתמש של השולח. מכיוון ששם המשתמש הוא "Omer", האורך הינו 4.
- **בכחול** (תו 2-5 משמאל) – שם המשתמש של השולח. בדוגמה זו – "Omer".
- **בכתום** (תו 6 משמאל) – הפקודה לשימוש. במקרה זה מדובר בהודעת צ'אט, ועל כן הפקודה היא 1.
- **בירוק** (תו 7-8 משמאל) – אורך ההודעה. מכיוון שההודעה היא "This is an example", האורך הוא 18.
- **בסגול** (תו 9 משמאל עד סוף השורה) – ההודעה עצמה, במקרה הזה – "This is an example".

במקרה הזה, המשתמש Omer שלח את ההודעה: "This is an example".

שימו לב: על הגודל של השדה הראשון (אורך) להיות באורך מוגדר, למשל של ארבעה בתים. אחרת, כיצד נדע האם הודעה שמתחילה ב-"30" כוונתה לשם משתמש באורך 30, או שמא שם משתמש באורך 3 תווים, כשהתו הראשון הוא "0"?

דוגמה למינוי מנהל:

4Omer26Shlomi

הסבר:

- **באדום** (תו 1 משמאל) – אורך שם המשתמש של המנהל הממנה. מכיוון ששם המשתמש הוא "Omer", האורך הינו 4.
- **בכחול** (תו 2-5 משמאל) – שם המשתמש של המנהל הממנה. בדוגמה זו – "Omer".
- **בכתום** (תו 6 משמאל) – הפקודה לשימוש. במקרה זה מדובר במינוי מנהל, ועל כן הפקודה היא 2.
- **בירוק** (תו 7 משמאל) – אורך שם המשתמש שיש למנות למנהל. מכיוון ששם המשתמש הינו "Shlomi", האורך הוא 6.
- **בסגול** (תו 8 משמאל עד סוף השורה) – שם המשתמש שיש למנות למנהל. במקרה הזה – "Shlomi".

במקרה זה, המשתמש Omer מינה את המשתמש Shlomi להיות מנהל.

דוגמה להעפת משתמש:

4Omer33Avi

הסבר:

- באדום (תו 1 משמאל) – אורך שם המשתמש של המעיף. מכיוון ששם המשתמש הוא "Omer", האורך הינו 4.
- בכחול (תו 2-5 משמאל) – שם המשתמש של המעיף. בדוגמה זו – "Omer".
- בכתום (תו 6 משמאל) – הפקודה לשימוש. במקרה זה מדובר בהעפת משתמש, ועל כן הפקודה היא 3.
- בירוק (תו 7 משמאל) – אורך שם המשתמש שיש להעיף. מכיוון ששם המשתמש הינו "Avi", האורך הוא 3.
- בסגול (תו 8 משמאל עד סוף השורה) – שם המשתמש שיש להעיף. במקרה הזה – "Avi".

במקרה זה, המשתמש Omer העיף את המשתמש Avi מן הצ'אט.

דוגמה להשתקת משתמש:

4Omer43Avi

הסבר:

- באדום (תו 1 משמאל) – אורך שם המשתמש המשתיק. מכיוון ששם המשתמש הוא "Omer", האורך הינו 4.
- בכחול (תו 2-5 משמאל) – שם המשתמש של המשתיק. בדוגמה זו – "Omer".
- בכתום (תו 6 משמאל) – הפקודה לשימוש. במקרה זה מדובר בהשתקת משתמש, ועל כן הפקודה היא 4.
- בירוק (תו 7 משמאל) – אורך שם המשתמש שיש להשתיק. מכיוון ששם המשתמש הינו "Avi", האורך הוא 3.
- בסגול (תו 8 משמאל עד סוף השורה) – שם המשתמש שיש להשתיק. במקרה הזה – "Avi".

במקרה זה, המשתמש Omer השתיק את המשתמש Avi.

דוגמה לשליחת הודעה פרטית בין משתמשים:

4Omer56Shlomi23This message is private

הסבר:

- באדום (תו 1 משמאל) – אורך שם המשתמש של השולח. מכיוון ששם המשתמש הוא "Omer", האורך הינו 4.
- בכחול (תו 2-5 משמאל) – שם המשתמש של השולח. בדוגמה זו – "Omer".

- **בכתום** (תו 6 משמאל) – הפקודה לשימוש. במקרה זה מדובר בשליחת הודעה פרטית, ועל כן הפקודה היא 5.
- **בירוק** (תו 7 משמאל) – אורך שם המשתמש שאליו נשלחת ההודעה. מכיוון ששם המשתמש הינו "Shlomi", האורך הוא 6.
- **בסגול** (תו 8-13 משמאל) – שם המשתמש שאליו נשלחת ההודעה. במקרה הזה – "Shlomi".
- **בשחור** (תו 14-15 משמאל) – אורך ההודעה הפרטית. מכיוון שההודעה הינה: "This message is private", האורך הוא 23.
- **בחום** (תו 16 משמאל עד סוף השורה) – ההודעה הפרטית עצמה. במקרה זה, ההודעה היא: "This message is private".

במקרה זה, המשתמש Omer שלח למשתמש Shlomi את ההודעה הפרטית: "This message is private".

צד השרת

השרת מתקשר עם הלקוחות רק במחרוזות, כלומר הוא שולח מחרוזת שתוצג ללקוח. כל הודעה נשלחת בפורמט הבא:

שדה	סוג	דוגמה
אורך המחרוזת	מספר	11
המחרוזת	מחרוזת	Hello world

<u>דוגמה לשליחה הודעה מהשרת ללקוח:</u>

4409:47 talmid6 has been kicked from the chat!

הסבר:

- **באדום** (תו 1-2 משמאל) – אורך כל ההודעה שהשרת שולח. מכיוון שההודעה היא "09:47 talmid6 has been kicked from the chat!", האורך הינו 44.
- **בכחול** (תו 3 משמאל עד סוף השורה) – ההודעה עצמה שהשרת שלח.
בדוגמה זו – "09:47 talmid6 has been kicked from the chat!".

תכנות Sockets מתקדם – סיכום

בפרק זה למדנו כיצד לתכנת באמצעות Sockets שרת ולקוח היכולים לטפל בכמה בקשות במקביל. התחלנו בהבנת הצורך במימוש שירותים שמטפלים בכמה לקוחות, ולאחר מכן הבנו את האתגר שבמימוש שירותים שכאלה. לאחר מכן, הכרנו את **select**, הפונקציה שעזרה לנו להתגבר על אותם אתגרים.

מימשנו שרת שמקבל מידע מכמה לקוחות שונים במקביל, ומדפיס את המידע למסך. לאחר מכן, כתבנו לקוח שיבדוק את השרת הזה. בשלב זה רק הצלחנו לקרוא מידע מכמה לקוחות. בהמשך, מימשנו שרת שגם עונה לכל לקוח ולקוח בהתאם למידע שהוא שלח, וכתבנו לקוח שיבדוק את השרת הזה.

מאוחר יותר, כתבנו צ'אט שמאפשר לכמה לקוחות לתקשר זה עם זה באמצעות שרת אחד. במקרה זה, נתקלנו לראשונה בצורך לממש לקוח שיוכל גם לקרוא מידע מהמשתמש וגם לקבל מידע מהשרת במקביל. כמו כן, התמודדנו עם שליחת הודעה למספר רב של לקוחות, כאשר צריך בכל פעם לזכור איזה לקוח קיבל את המידע ואיזה עדיין לא. לסיום, שדרגנו את הצ'אט בשלל יכולות מעניינות: החל מהוספת שם המשתמש לכל הודעה, דרך מינוי מנהלים שיכולים להעיף או להשתיק משתמשים אחרים, ועד למימוש הודעות פרטיות.

בדרך זו רכשנו כלי משמעותי נוסף, שמאפשר לנו לתקשר עם מספר ישויות רשת במקביל מעל ממשק ה-Sockets.

פרק 13

מילון מושגים

פרק זה כולל מונחים בהם נעשה שימוש לאורך הספר, והגדרותיהם. המילון נועד לסייע במהלך הקריאה. ההגדרות מובאות בהקשר שלהן לפרק ולמידע שמצוין בספר, ולא נועדו לעמוד בזכות עצמן בכדי להגדיר את המושגים.

פרק 1 – תחילת מסע – איך עובד האינטרנט?

WWW – ראשי תיבות של World Wide Web. אוסף עמודי האינטרנט אליהם אנו גולשים בדפדפן.

בקשה (Request) – הודעה שנשלחת מהלקוח אל השרת כדי לבקש שירות כלשהו.

תגובה (Response) – הודעה שנשלחת מהשרת אל הלקוח כמענה לבקשה.

כתובת מקור – כתובת המציינת מי שלח חבילת מידע מסוימת.

כתובת יעד – כתובת המציינת לאן חבילה ממוענת.

ping – כלי המאפשר לבדוק קישוריות לישות מרוחקת, ואת הזמן שלוקח לחבילה להגיע אליה ובחזרה.

traceroute – כלי המאפשר למצוא את הדרך שעוברת חבילה בין המחשב שלי לנקודות קצה שונות.

GeoIP – כלי הממפה בין כתובת IP לבין המיקום הגיאוגרפי שלה.

קפיצה (Hop) – העברה של חבילת מידע בין רכיב אחד לרכיב אחר המחוברים ישירות.

שמות דומיין – כתובות קריאות לפי פרוטוקול DNS, לדוגמה "www.facebook.com" או "www.themarker.co.il".

DNS – מערכת המאפשרת המרה בין שמות דומיין וכתובות IP.

nslookup – כלי המאפשר לבצע תשאולי DNS.

פרק 2 – תכנות ב-Sockets

תקשורת שרת-לקוח (Client-Server) – סוג תקשורת בין שרת, המספק שירות כלשהו, לבין לקוח, המשתמש בשירות המסופק.

Socket – ממשק תוכנתי להעברת מידע בין תוכנות שונות. זהו API שמסופק בידי מערכת ההפעלה.

פרק 3 – Wireshark ומודל חמש השכבות

הסנפה – הפעולה בה אנו מסתכלים על חבילות המידע בדיוק כפי שנשלחו או התקבלו בכרטיס הרשת.

Wireshark – תוכנת הסנפה.

פקטה (חבילה, Packet) – חבילת מידע המכילה מוען, נמען ותוכן ההודעה. מונח זה מתאר גוש מידע בשכבת הרשת.

פרוטוקול (Protocol, תקן) – סט מוגדר של חוקים, הקובע כללים ברורים כיצד צריכה להיראות התקשורת בין הצדדים השונים.

ישות (Entity) – כל רכיב המחובר לרשת – בין אם הוא סמארטפון, מחשב נייד, שרת של Google, רכיב רשת שנמצא בדרך בין ישויות אחרות, או רכיב בקרה של תחנת כוח המחובר גם הוא לרשת לצורך שליטה מרחוק.

ISO – ארגון התקינה הבינלאומי.

OSI – מודל שבע השכבות.

ריבוב (Multiplexing) – התהליך שבו מידע ממספר מקורות משולב אל תווך משותף אחד.

הרעבה (Starvation) – תופעה בה תחנה אחת מציפה את הקו ברצף ארוך של מידע, ובכך מונעת מתחנות אחרות גישה לשדר על הקו.

Encapsulation (כימוס) – עטיפת מידע בשכבות נוספות.

Decapsulation (קילוף) – הוצאת המידע של שכבה מסוימת.

Header (תחילית) – מידע שמוסיפה כל שכבה לתחילת הפקטה, מכיל מידע שמשמש לשליטה ובקרה על הפקטה.

מסנן תצוגה (Display Filter) – מסנן את הפקטות המוצגות למסך על פי תנאי מסוים. מסנן זה רץ ברמת האפליקציה.

מסנן הסנפה (Capture Filter) – מסנן את הפקטות הנקלטות לאפליקציה על פי תנאי מסוים. מסנן זה רץ ברמת ה-Driver של מערכת ההפעלה.

פרק 4 – שכבת האפליקציה

אפליקציה – יישומים ותוכנות שנגישות למשתמשי קצה באמצעות מחשב, סמארטפון או טאבלט, ובנויות במודל שרת-לקוח, כך שהאפליקציה משמשת כלקוח (בין אם כאפליקציה ייעודית ובין אם באמצעות הדפדפן), ומסתמכת על שרת שאיתו היא מתקשרת באמצעות פרוטוקולי אינטרנט.

משאב (Resource) – ברשת האינטרנט, הכוונה היא לכל רכיב שיכול להיות חלק מעמוד אינטרנט – תמונה, טקסט, HTML, javascript וכדומה.

URL – כתובת לזיהוי משאב ברשת האינטרנט (ראשי התיבות: Uniform Resource Locator). האופן שבו בנוי URL:

<protocol>://<host>/<resource_path>?<parameters, separated by &>

- לדוגמה: http://twitter.com/search?q=obama&mode=users
- מתאר גישה בפרוטוקול HTTP לשרת twitter.com, ומבקש את המשאב search/ עם הפרמטרים q ו-users (ראו: URL Parameters).

GET – סוג בקשת HTTP לקבלת משאב ספציפי מהשרת – כוללת את כתובת המשאב. הבקשה לא אמורה לגרום לשינוי מצב בשרת, ולא מכילה מידע (data), מלבד פרמטרים ב-URL.

POST – סוג בקשת HTTP להעברת מידע לשרת (כגון שליחת אימייל, מילוי טופס או העלאת תמונה). המבנה שלה דומה לבקשת GET, אלא שהיא מכילה גם מידע (data) בגוף הבקשה – המידע שמועבר לשרת.

HTTP Header – מופיע הן בבקשות והן בתגובות HTTP, בין שורת הכותרת לבין התוכן. מכיל רשימה של שדות, מתוך רשימה של שדות אפשריים (כגון גודל המידע שבהודעה, סוג הלקוח, סוג השרת). חלק מהשדות ניתנים לשימוש רק בבקשות, חלקן רק בתגובות, וחלקן בשני המקרים. מנגנונים מתקדמים (כגון cache ואותנטיקציה) לרוב עושים שימוש בשדות ה-header.

Status Code – קוד שמתאר את מצב ההודעה שנשלחת בתגובה לבקשת HTTP. המפורסמים ביותר הם 200 OK שמעיד על תגובה תקינה, וכן 404 Not Found, המסמן שהמשאב המבוקש לא נמצא.

HTTP Response – הודעה הנשלחת כתגובה לבקשה שקדמה לה, לרוב התגובה תישלח מהשרת ללקוח. מכילה קוד מצב (status code), שדות header ותוכן.

Content-type – שדה header שמתאר את סוג המידע (data) שמצורף לבקשה/תגובה. סוגי תוכן נפוצים: text/html, application/javascript, image/jpeg.

URL Parameters – כחלק מה-URL הנשלח בבקשה ניתן לכלול פרמטרים שבהם יעשה השרת שימוש כשיטפל בבקשה. הפרמטרים מופרדים על-ידי התו &, ויימצאו לאחר חלק ה-path שב-URL, מופרדים ממנו על-ידי סימן שאלה.

- לדוגמה, ב-URL הבא: http://twitter.com/search?q=obama&mode=users
- ישנם שני פרמטרים: הראשון בשם q עם הערך obama, והשני בשם mode עם הערך users.

HTTP Session – אינטראקציה מתמשכת (כלומר, יותר מזוג בקשה-תגובה יחיד) בין משתמש קצה באפליקציית לקוח לבין שרת. session הוא stateful, כלומר השרת "זוכר" את ההיסטוריה של ה-session, בניגוד לאופן המקורי בו פעל פרוטוקול HTTP, שמכונה stateless. לרוב יזוהה ה-session באמצעות cookie שיועבר ב-header של הבקשות.

Cookie – מחרוזת המשותפת לשרת וללקוח, הנקבעת על-ידי השרת ומועברת על-ידי הלקוח בכל בקשה, על מנת שהשרת יוכל לזהות בקשות HTTP ששייכות לאותו ה-Session. מנגנון זה עושה שימוש בשדות header. שימושים נפוצים: משתמש שעבר זיהוי ואימות, לא צריך לבצע זאת מחדש (Gmail, Facebook), עגלת קניות (Amazon).

Cache – מנגנון שנועד לחסוך בתעבורה של משאבים ברשת, על-ידי כך שמשאבים נשמרים בדיסק המקומי של הלקוח, ויובאו מחדש מהשרת רק אם הגרסא שם השתנתה. מנגנון זה עושה שימוש בשדות header.

Conditional-GET – הבקשה הנשלחת על-ידי הלקוח לקבלת משאב שקיים עבורו עותק ב-cache. לבקשה מצורף הזמן בו נשמר המשאב ב-cache (בשדה header), והמשאב עצמו ייכלל בתגובה רק אם יש גרסה חדשה יותר בשרת. אם הגרסה שב-cache של הלקוח עדכנית, תוחזר תגובה עם קוד מצב 304 שמסמן שהמשאב לא שונה (והמשאב לא ייכלל בתגובה – כך נחסכה תעבורה "מיותרת").

Basic HTTP Authentication – מנגנון האותנטיקציה הבסיסי שנכלל בפרוטוקול HTTP עושה שימוש בשדות header וב- status code כדי לבצע את הזיהוי והאימות. המנגנון די חלש, משום שהוא אינו מצפין את שם המשתמש והסיסמה, אלא רק מקודד איתם.

שאילתת (**DNS (DNS Query** – שאלה בפרוטוקול DNS עבור שם דומיין מסוים.

אזור (Zone) – מערכת ה-DNS הינה היררכית, והתו המפריד שיוצר את ההיררכיות הוא התו נקודה ("."). כך למשל, הדומיין www.facebook.com מתאר שרת בשם "www" בתוך האזור "facebook" שבתוך האזור "com".

RR (Resource Record) – רשומה בשאילתת DNS או תשובת DNS.

פרק 5 – Scapy

Scapy – ספריית פייתון המאפשרת לעבוד עם חבילות מידע בצורה מתקדמת. מאפשרת הסנפה, יצירה ושליחה של פקטות.

Resolving – תרגום של שמות דומיין לכתובות IP, למשל באמצעות DNS.

פרק 6 – שכבת התעבורה

פורט (Port) – מזהה תוכנה באורך 16 ביטים (bits). נחוץ על מנת לרבב תקשורת בין מספר תוכנות על אותה ישות רשת.

netstat – כלי המאפשר לדעת על איזה פורטים המחשב מאזין, ואילו קישורים קיימים כרגע.

פורטים מוכרים (Well known ports) – הפורטים מהטווח שבין 0 ועד 1023 (כולל). פורטים אלו מוקצים בידי IANA עבור אפליקציות ספציפיות.

תקורה (Overhead) – מידע נוסף (יתיר) שנשלח מעבר למידע שרוצים להעביר. לדוגמה, לשליחת Header יש תקורה – עובר מידע ברשת שהוא מידע נוסף על המסר שרצינו להעביר.

פרוטוקול מבוסס קישור (Connection Oriented Protocol) – על מנת לתקשר עם ישות כלשהי באמצעות פרוטוקול מבוסס קישור, יש "להקים" קודם את הקישור, לאחר מכן להשתמש בקישור שהוקם

ולבסוף לנתק את הקישור. מבחינת המשתמש, הוא מתייחס לקישור כמו לשפופרת הטלפון: הוא מזין מידע (במקרה שלנו – רצף של בתים) לקצה אחד, והמשתמש השני יקבל את המידע בצד השני. פרוטוקולים מבוססי קישור מבטיחים הגעת המידע, וכן הגעתם בסדר הנכון.

פרוטוקול שאינו מבוסס קישור (Connectionless Protocol) – על מנת לתקשר עם ישות כלשהי באמצעות פרוטוקול שאינו מבוסס קישור, אין צורך בהרמה וסגירה של קישור, וניתן פשוט לשלוח את החבילה. בפרוטוקול מסוג זה, אין הבטחה שהחבילה תגיע ליעדה. כמו כן, אין הבטחה שהחבילות תגענה בסדר הנכון.

UDP (User Datagram Protocol) – פרוטוקול נפוץ של שכבת התעבורה. פרוטוקול זה אינו מבוסס קישור. דוגמה נפוצה לשימוש: פרוטוקול DNS.

TCP (Transmission Control Protocol) – פרוטוקול נפוץ של שכבת התעבורה. פרוטוקול זה מבוסס קישור. דוגמה נפוצה לשימוש: פרוטוקול HTTP.

פורט מקור (Source Port) – הפורט של התוכנה ששלחה את החבילה.

פורט יעד (Destination Port) – הפורט של התוכנה שצפויה לקבל את החבילה.

Checksum – תוצאה של פעולה מתמטית שמתבצעת על המידע. ה-Checksum מתווסף לחבילה בתור מידע יתיר ומשמש לזיהוי שגיאות. הצד המקבל מחשב Checksum בעצמו עבור כל חבילה, ומשווה אותו אל תוכן ה-Checksum שכתוב בחבילה עצמה. באם התוצאה יצאה זהה – החבילה נחשבת תקינה. אחרת – התגלתה שגיאה.

סגמנטים (Segments, מקטעים) – השם של גוש מידע בשכבת התעבורה.

Sequence Number – מספר סידורי שניתן לחבילות או חלק מהן על מנת לעקוב אחר רצף המידע. שימוש במספר סידורי מאפשר לדעת איזה חלק מהמידע הגיע, איזה חלק לא הגיע, ולהבין מה הסדר הנכון של המידע. ב-TCP, ישנו מספר סידורי לכל בית (byte). לכל אחד מהבתים ברצף יש מספר סידורי משלו. בכל חבילה שנשלח, יהיה המספר הסידורי שמציין את הבית הנוכחי בחבילה.

ACK (Acknowledgement) – חבילה שנועדה לאשר שהתקבל מידע מן הצד השני. כשם שהמספרים הסידוריים של TCP מתייחסים לבתים (bytes) ברצף המידע, כך גם מספרי ה-ACK. מספר ה-ACK ב-TCP מציין את המספר הסידורי של הבית הבא שמצופה להתקבל.

Three Way Handshake (לחיצת יד משולשת) – הדרך להרמת קישור ב-TCP. כוללת שלוש חבילות: חבילת SYN, חבילת SYN+ACK וחבילת ACK.

ISN (Initial Sequence Number) – ערך ה-Sequence Number ההתחלתי של תקשורת TCP. ערך זה נבחר באופן רנדומלי.

פרק 7 – שכבת הרשת

ניתוב (Routing) – תהליך ההחלטה על הדרך שבה יש להגיע מנקודה א' לנקודה ב' ברשת.

IP (Internet Protocol) – פרוטוקול שכבה שלישית הנפוץ באינטרנט.

ipconfig – כלי המאפשר לראות מידע על הגדרות הרשת שלנו. למשל, הכלי מראה מה כתובת ה-IP שלנו.

כתובת IP – כתובות לוגיות של שכבת הרשת בפרוטוקול IP. כתובת IPv4 מיוצגת באמצעות ארבעה בתים.

מזהה רשת (Network ID) – חלק בכתובת לוגית (כתובת IP) המציין לאיזו רשת שייכת הכתובת.

מזהה ישות (Host ID) – חלק בכתובת לוגית המציין לאיזה כרטיס רשת שייכת הכתובת, בתוך הרשת.

Subnet Mask (מסיכת רשת) – מגדיר כמה ביטים (bits) מתוך כתובת ה-IP מייצגים את מזהה הרשת.

Broadcast – כתובת המציינת כי החבילה צריכה להישלח אל כל הישויות ברשת.

Loopback – כתובת המציינת שהחבילה לא צריכה לעזוב את כרטיס הרשת, אלא "להישאר במחשב".

נתב (Router) – רכיב רשתי בשכבה שלישית. מטרתו היא לקשר בין מחשבים ורשתות ברמת ה-IP. רוב מלאכת הניתוב מתבצעת בידי נתבים.

טבלת ניתוב (Routing Table) – טבלה הכוללת מזהי רשת ולאן להעביר חבילות המיועדות למזהי רשת אלו. ברוב המקרים, הטבלה היא דינאמית ועשויה להשתנות בהתאם למצב הרשת. נתבים, וגם רכיבים אחרים, משתמשים בטבלאות ניתוב כדי לדעת לאן להעביר את החבילות המגיעות אליהם.

route – כלי המאפשר להסתכל על טבלאות ניתוב ולערוך אותן.

Default Gateway – הנתב המשויך אל רכיב כלשהו. כל חבילה שלא התאמתה על חוק ספציפי בטבלת הניתוב, תישלח אל ה-Default Gateway.

ICMP (Internet Control Message Protocol) – פרוטוקול בשכבה השלישית, הנועד למציאת תקלות ברשת ולהבנת מצב הרשת.

TTL (Time To Live) – שדה ב-IP Header שמציין כמה Hops החבילה עוד יכולה לעבור בטרם תיזרק. כל נתב או רכיב אחר שמעביר את החבילה הלאה מחסיר 1 מערך השדה.

DHCP (Dynamic Host Configuration Protocol) – פרוטוקול המשמש להקצאה דינאמית של כתובות IP וללמידה של פרטי הרשת.

MTU (Maximum Transmission Unit) – מאפיין של רשת המתאר את הגודל המקסימלי של גוש מידע שיכול לעבור ברשת זו.

פרגמנטציה (Fragmentation) – חלוקה של חבילת מידע למקטעים קטנים יותר, מתבצע בכדי לשלוח חבילות הגדולות יותר מה-MTU.

כתובות IP פרטיות – שלושה טווחי כתובות IP שהוקצו בידי IANA על מנת לחסוך בכתובות IP בעולם. בתוך רשתות מקומיות, ישויות יכולות לקבל כתובות פרטיות, שיזהו אותן בתוך הרשת בלבד, ולא בעולם החיצוני. כתובות אלו אינן ניתנות לניתוב. מכאן שנתב באינטרנט שרואה חבילה המיועדת לכתובת פרטית עתיד "לזרוק" אותה.

NAT (Network Address Translation) – דרך להעביר מידע בין ישויות בעלות כתובות IP פרטיות לישויות מחוץ לרשת. לשם כך, רכיב ה-NAT מחליף את כתובת ה-IP של הישות בעלת כתובת ה-IP הפרטית בכתובת ה-IP של רכיב ה-NAT עצמו, לו יש כתובת IP חיצונית שניתן לנתב.

IPv6 – הגירסה החדשה של פרוטוקול IP. כתובות בגירסה זו של הפרוטוקול הן באורך 16 בתים (bytes).

פרק 8 – שכבת הקו

Ethernet – פרוטוקול של שכבת הקו, בו משתמשים כרטיסי רשת מסוג Ethernet (המחוברים באופן קווי).

כתובת MAC – כתובות פיזיות של שכבת הקו. לכל כרטיס רשת יש כתובת כזו. כתובת MAC בפרוטוקול Ethernet הינה בגודל שישה בתים (bytes).

מסגרת (Frame) – גוש מידע בשכבה השנייה.

ARP (Address Resolution Protocol) – פרוטוקול שנועד למפות בין כתובות לוגיות של שכבת הרשת לכתובות פיזיות של שכבת הקו.

פורט (Port) – "כניסה" ברכיב רשת אליה ניתן לחבר כבל רשת. הערה: על אף שמדובר באותו השם כמו פורטים של שכבת התעבורה, המשמעות שונה.

Hub (רכזת) – רכיב של השכבה הפיזית, השכבה הראשונה. הוא נועד כדי לחבר כמה ישויות רשת יחד. ה-Hub אינו מכיר כתובות Ethernet או IP, מבחינתו הוא רק מעביר זרם חשמלי מפורט אחד אל פורטים אחרים. כאשר מחשב שמחובר ל-Hub שולח מסגרת, ה-Hub מעתיק את המסגרת ושולח אותה לכל הפורטים שלו, מלבד לזה שממנו המסגרת נשלחה.

Switch (מתג) – רכיב של שכבת הקו, השכבה השנייה. אי לכך, ה-Switch מכיר כתובות MAC, מבין את המבנה של מסגרות בשכבה שנייה (למשל מסגרות Ethernet), ויודע לחשב Checksum. לאחר שה-Switch למד את הרשת, הוא מעביר מסגרת מהפורט בה הוא קיבל אותה אל הפורט הרלוונטי בלבד.

התנגשות (Collision) – מצב בו שתי ישויות (או יותר) משדרות בערוץ המשותף בו זמנית. במקרה זה, המידע שנשלח יגיע באופן משובש – כלומר, המידע שיגיע הוא לא המידע שהישות התכוונה לשלוח.

Multicast – כתובות מסוג זה שייכות ליותר מישות אחת.

Unicast – כתובת מסוג זה שייכת לישות אחת בלבד.

פרק 10 – השכבה הפיזית

סיבית (Bit) – קיצור של המושג ספרה בינארית, ספרה שיכולה להחזיק אחד משני ערכים: 0 או 1. סיבית היא תרגום של המושג הלועזי bit, שהוא קיצור לביטוי binary digit.

תווך (Medium) – ברשתות תקשורת, תווך התקשורת הוא החומר, או האמצעי הפיזי, המשמש להעברת המידע.

קידוד (Encoding או Coding) – תהליך בו מידע מתורגם לאותות מוסכמים. כל שיטת מימוש של השכבה הפיזית מגדירה אופן בו מתרגמים את הספרות 0 ו-1 לסימנים מוסכמים על גבי התווך.

גל (Wave) – התפשטות (או התקדמות) של הפרעה מחזורית בתווך במרחב. גל יכול לנוע בחומר (כמו גלים במים), אך גם באוויר (כמו גל קול) ואף בוואקום (כמו גלים אלקטרומגנטיים, שיכולים לנוע בוואקום ובתווכים רבים אחרים).

הגל האלקטרומגנטי (Electromagnetic Wave) – הוא סוג של גל שנע בתווכים שונים במרחב (אוויר, מים, זכוכית, ריק/ואקום, ועוד) באמצעות שינוי של השדות החשמליים והמגנטיים. האור שמגיע מהשמש ושאותו אנו רואים הוא גל אלקטרומגנטי שהתדר שלו נמצא בטווח שהעין רואה (400-800 THz). עוצמת האור שווה לאמפליטודה של הגל האלקטרומגנטי, שמעידה על חוזק התנודה בשדות החשמליים והמגנטיים.

Modulation (אפנון) – היא העברה של מידע על גבי גל נושא. הרעיון באפנון הוא להרכיב גל של מידע (כגון גל קול של מוסיקה) על גבי גל ״נושא״. גל נושא הוא גל ״חלק״ (גל שמאוד קרוב לפונקציית סינוס) בתדר גבוה יותר מגל המידע.

אפנון מבוסס אמפליטודה (Amplitude Modulation) – בשיטת אפנון זו, ״מרכיבים״ גל של מידע בתדר נמוך על גבי גל ״נושא״ בתדר גבוה וקבוע. בשיטה זו, האמפליטודה של הגל הנושא (בתדר הקבוע) תשתנה לפי האמפליטודה של גל המידע.

אפנון מבוסס תדר (Frequency Modulation) – בשיטת אפנון זו, משנים את התדר של הגל הנושא על פי האמפליטודה של גל המידע.

מודם (Modem) – קיצור (באנגלית) של Modulator & Demodulator – מכשיר שמאפנן ומשחזר ביטים על גבי ערוץ תקשורת.

Duplex (דופלקס) – מאפיין של מערכות תקשורת דו כיווניות בין שתי נקודות. מערכת שהיא Half Duplex מאפשרת לשני צדדים לתקשר אחד עם השני באופן דו כיווני אך לא סימולטני. דוגמה למערכת Half Duplex היא ווקי-טוקי (מכשיר קשר אלחוטי), בו רק צד אחד יכול לדבר בזמן שהצד השני מקשיב. כששני הצדדים מנסים לדבר, אף אחד לא שומע את השני. מערכת שהיא Full Duplex מאפשרת לשני צדדים לתקשר אחד עם השני באופן מלא וסימולטני, זאת אומרת ששני הצדדים יכולים לדבר באותו הזמן. הטלפון הוא דוגמה למערכת Full Duplex, מאחר והיא מאפשרת לשני דוברים לדבר בו זמנית וגם לשמוע אחד את השני.

כבל CAT5 – כבל שמאגד בתוכו 4 כבלי זוגות, כל זוג בצבע שונה. אפשר לראות את כבלי הזוגות המלופפים סביב עצמם בתמונה העליונה. אם לדייק, ישנם מספר סוגי כבלים כאלו: CAT3, CAT5e, וגם CAT6. מבחוץ הם כולם נראים אותו דבר, אך מבפנים הם נבדלים באיכות בידוד ההפרעות החשמליות. איכות הבידוד משפיעה על קצב העברת הביטים. כשאתם הולכים לחנות לקנות כבל רשת, ברוב המקרים תצטרכו כבל CAT5.

חיבור RJ-45 – השקע והתקע של כבלי הרשת הסטנדרטיים, כולל צורתם וסידור הכבלים הפנימיים לפי צבע, מוגדר בתקן שנקרא RJ – Registered Jack. עבור כבלי רשת Ethernet, התקן הוא RJ-45, אך ישנם תקנים דומים גם עבור כבלים אחרים כגון כבל הטלפון (RJ-11). יש לציין שחיבור זה נקרא גם חיבור P8C8, ובמקומות בהם כך הוא נקרא, הכוונה היא לאותו סוג חיבור כמו RJ-45.

תקן Base-T10 – תקן זה מגדיר כיצד משתמשים בכבלי CAT5 וחיבורי RJ-45 כדי להעביר ביט בודד על גבי הכבל.

כבל רשת מוצלב (Ethernet Crossover Cable) – הינו כבל רשת בו כבלי הזוגות של השליחה וקבלה הוצלבו, דבר המאפשר לחבר שני מחשבים ישירות אחד לשני, ללא Switch ביניהם.

תקשורת מיקרוגל (Microwave Transmission) – העברת מידע באמצעות גלים אלקטרומגנטיים בתווך אורך גל שניתן למדוד בסנטימטרים. גלי מיקרוגל הם גלים בטווח התדרים בין 1GHz ל-30GHz.

סיב אופטי (Optical Fiber) – הינו סיב עשוי זכוכית או פלסטיק, המאפשר העברת אור בתוכו למרחקים ארוכים עם אובדן מינימלי של עוצמה.

ממסר (Relay) – רכיב שמקבל אות תקשורתי, מגביר אותו ומשדר אותו הלאה. תפקידו להאריך את המרחק אליו ניתן להעביר אות תקשורתי.

פרק 14

פקודות וכלים

פרק זה כולל כלים ופקודות בהם נעשה שימוש לאורך הספר, עם פירוט המטרה והשימוש בהם. הפרק נועד לסייע לקורא להתמצא כיצד להשתמש בכלי מסוים, או לקוראים המעוניינים להכיר את החלופות לפקודות המוצגות מעל מערכת הפעלה מבוססת UNIX.

הרשימה

מטרת הכלי	היכן הוצג בספר	שימוש ב-Windows	שימוש ב-UNIX
לבדוק קישוריות לישות מרוחקת, ואת הזמן שלוקח לחבילה להגיע אליה ובחזרה.	תחילת מסע – איך עובד האינטרנט?/ כתובות IP	ping -n 4 www.google.com	ping -c 4 www.google.com
למצוא את הדרך שעוברת חבילה בין המחשב שלי לנקודות קצה שונות.	תחילת מסע – איך עובד האינטרנט ? / ענן האינטרנט	tracert -d www.google.com	traceroute -n www.google.com
לבצע תשאולי DNS.	תחילת מסע – איך עובד האינטרנט? /DNS	nslookup www.google.com	nslookup www.google.com
התחברות כלקוח לשירות מרוחק.	שכבת האפליקציה/ התבוננות מודרכת בתגובת HTTP	telnet google.com 80	telnet google.com 80
תשאול DNS עם סוג שאילתה מסוים.	שכבת האפליקציה/ תרגיל 4.15 – תשאול רשומות מסוגים שונים	nslookup -t<TYPE> <host/address>	nslookup set type=<TYPE> <host/address>
לאפס את מטמון רשומות ה-DNS.	Scapy/ תרגיל 5.1 מודרך – הספה של DNS	ipconfig /flushdns	תלוי בגירסה הספציפית. הסבר ניתן למצוא כאן: http://goo.gl/AF0Ul3
לספק מידע על חיבורי רשת, ספציפית חיבורי בשכבת התעבורה.	שכבת התעבורה/ תרגיל 6.1 מודרך – אילו פורטים פתוחים במחשב שלי?	netstat -na	netstat -na

מטרת הכלי	היכן הוצג בספר	שימוש ב-Windows	שימוש ב-UNIX
להציג מידע על כרטיסי רשת.	שכבת הרשת/ מה כתובת ה- IP שלי?	ipconfig /all	ifconfig -a
להציג טבלאות ניתוב.	שכבת הרשת/ מהי טבלת הניתוב שלי?	route print	route -n
להתנתק משרת ה-DHCP ולוותר על פרטי הרשת.	שכבת הרשת/ תרגיל 7.8 מודרך – קבלת IP באמצעות DHCP	ipconfig /release	dhclient -r
לקבל את פרטי הרשת משרת ה-DHCP.	שכבת הרשת/ תרגיל 7.8 מודרך – קבלת IP באמצעות DHCP	ipconfig /renew	dhclient
להציג את מטמון ה-ARP.	שכבת הקו/ מטמון (Cache) של ARP	arp -a	arp -n
למחוק רשומה ממטמון ה-ARP.	שכבת הקו/ מטמון (Cache) של ARP	arp -d 192.168.1.100	arp -d 192.168.1.100

זכויות יוצרים – מקורות חיצוניים

http://commons.wikimedia.org/wiki/File:World%E2%80%99s_first_dual-core_smartphone_comes_to_europe.jpg
http://commons.wikimedia.org/wiki/File:Ethernet_RJ45_connector_p1160054.jpg
http://openclipart.org/detail/189964/pipe-by-barretr-189964
https://www.flickr.com/photos/ckelly/4846654926
https://www.flickr.com/photos/topgold/4399244167
http://pixabay.com/en/basket-buy-order-shopping-green-156678/
http://pixabay.com/en/buttons-shopping-cart-buy-24573/
http://pixabay.com/en/computer-desktop-keyboard-system-98400/
http://pixabay.com/en/envelope-e-mail-letter-mail-post-154134/
http://www.troyjessup.com/headers/
http://www.clker.com/cliparts/0/a/6/b/12065771771975582164reporter_flat.svg.med.png
https://www.iconfinder.com/icons/23912/router_wifi_icon#size=128
https://www.iconfinder.com/icons/47998/history_qualification_icon#size=128
http://www.opensecurityarchitecture.org/cms/library/icon-library
http://www.clipartbest.com/free-computer-clipart
http://www.iconarchive.com/show/pretty-office-9-icons-by-custom-icon-design/Magnifying-glass-icon.html
http://www.iconarchive.com/show/icons8-metro-style-icons-by-visualpharm/Ecommerce-Idea-icon.html
http://www.iconarchive.com/show/aerial-icons-by-chromatix/work-icon.html
http://www.iconarchive.com/show/my-seven-icons-by-itzikgur/Videos-1-icon.html
http://commons.wikimedia.org/wiki/File%3AInternational_Morse_Code.PNG
http://commons.wikimedia.org/wiki/File:2006-01-14_Surface_waves.jpg#mediaviewer/File:2006-01-14_Surface_waves.jpg
http://commons.wikimedia.org/wiki/File:Amplitude-modulation_he.svg
http://commons.wikimedia.org/wiki/File:Frequency_Modulation.svg#mediaviewer/File:Frequency_Modulation.svg

http://commons.wikimedia.org/wiki/File:TP_Neostrada_Thomson_SpeedTouch_.546jpg
http://commons.wikimedia.org/wiki/File:2.4_GHz_Wi-
Fi_channels_(802.11b,g_WLAN).svg#mediaviewer/File:2.4_GHz_Wi-
Fi_channels_(802.11b,g_WLAN).svg
http://commons.wikimedia.org/wiki/File:Wave-
he.png#mediaviewer/%D7%A7%D7%95%D7%91%D7%A5:Wave-he.png
http://commons.wikimedia.org/wiki/File:CAT5e_Cable.jpg#mediaviewer/File:CAT5e_Cable.jpg
http://commons.wikimedia.org/wiki/File:Cat_5.jpg
http://commons.wikimedia.org/wiki/File%3AEthernet_MDI_crossover.svg
http://commons.wikimedia.org/wiki/File:Parabolic_antennas_on_a_telecommunications_tower_o
n_Willans_Hill.jpg
http://commons.wikimedia.org/wiki/File:Multimode_stepindex_optical_fiber.svg

© 2019 Google Inc. All rights reserved. Google and the Google Logo are registered trademarks of Google Inc.
© 2019 Google Inc. All rights reserved. YouTube™ is a trademark of Google Inc.
© 2019 Google Inc. All rights reserved. Chrome™ browser is a trademark of Google Inc.

Python and the Python logos are trademarks or registered trademarks of the Python Software Foundation, used with permission from the Foundation.