COMPUTER NETWORKING ASSIGNMENT 2

Jonathan Kelsi & Amit Moshcovitz

DECEMBER 21, 2022

BAR ILAN UNIVERSITY

Computer Networking - 89535001

חלק א'

חלק ראשון

הקדמה

בדיוק כפי שמפורט בתרגיל, שינינו את קובץ הלקוח כך שהוא שולח לשרת את השמות שלנו, מחכה לתשובה ולאחר מכן שולח את מספרי הזהות שלנו. בנוסף, עדכנו את השרת לקבל שתי הודעות במקום אחת, ולאחר מכן הוא נסגר.

```python
import socket

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.connect(('132.70.66.13', 12345))

s.send(b'Amit Moshcovitz, Jonathan Kelsi')
data = s.recv(1024)
s.send(b'[redacted]')

s.close()
```

```python
import socket
server = socket.socket(socket.AF_INET, socket.SOCK_STREAM
server.bind(('', 12345))
server.listen(5)

client_socket, client_address = server.accept()
print('Connection from: ', client_address)
data = client_socket.recv(100)
print('Received: ', data)
client_socket.send(data.upper())
client_socket.close()
print('Client disconnected')
```

נעיר שידענו לאן לחבר את הלקוח, באמצעות הפקודה ifconfig, אותה ראינו בתרגול. בעזרתה גילינו שכתובות ה-ip של המחשבים שלנו היו 172.20.10.2 ו-172.20.10.6.

```
o: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        inet6 ::1  prefixlen 128  scopeid 0x10<host>
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 402258  bytes 359247674 (359.2 MB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 402258  bytes 359247674 (359.2 MB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

lp2s0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 172.20.10.6  netmask 255.255.255.240  broadcast 172.20.10.15
        inet6 2001:4df7:2:df10:1655:e6ad:d5cb:e4a8  prefixlen 64  scopeid 0x0<g
bal>
        inet6 fe80::ebb0:4878:b97:5888  prefixlen 64  scopeid 0x20<link>
        inet6 2001:4df7:2:df10:4a52:e0cf:ba34:c89  prefixlen 64  scopeid 0x0<gl
al>
        ether dc:e9:94:8f:6a:13  txqueuelen 1000  (Ethernet)
        RX packets 6153482  bytes 5342589961 (5.3 GB)
        RX errors 0  dropped 37  overruns 0  frame 0
        TX packets 3819401  bytes 772280791 (772.2 MB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

את השרת והלקוח הרצנו על מחשבים שונים, הסנפנו את התעבורה וסיננו את החבילות ב-WireShark לפי כתובות ה-ip של השרת והלקוח. לאחר מכן, סיננו את התעבורה בעזרת:

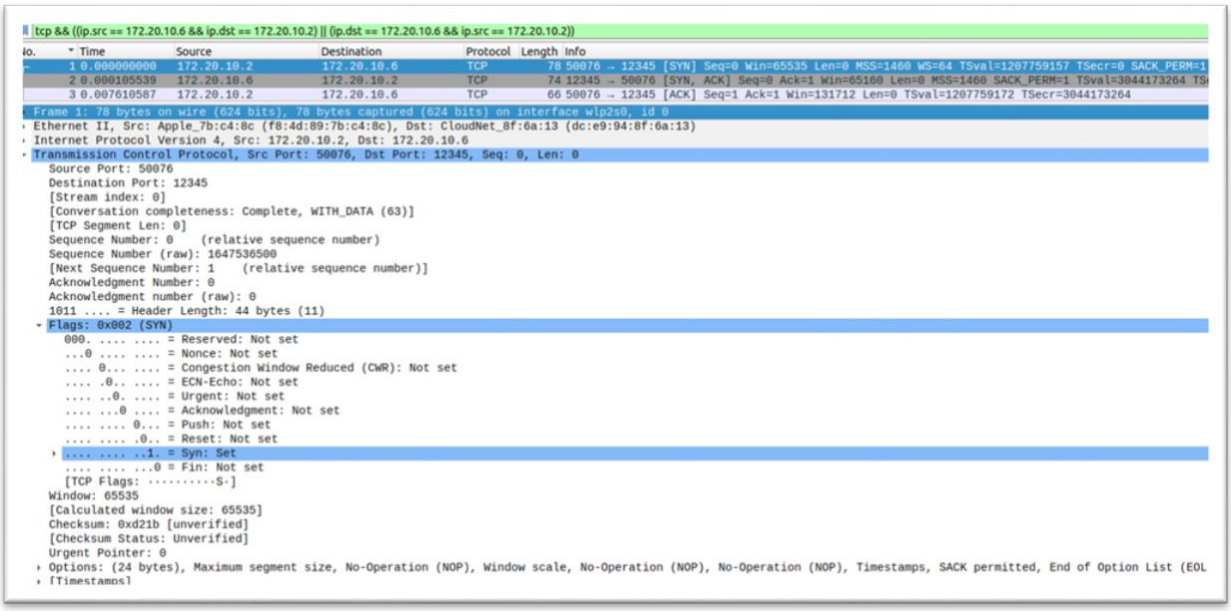
```
tcp && ((ip.src == 172.20.10.6 && ip.dst == 172.20.10.2) || (ip.dst == 172.20.10.6 && ip.src == 172.20.10.2))
```

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	172.20.10.2	172.20.10.6	TCP	78	50076 → 12345 [SYN] Seq=0 Win=65535 Len=0 MSS=1460 WS=64 TSval=1207759157 TSecr=0 SACK_PERM=1
2	0.000105539	172.20.10.6	172.20.10.2	TCP	74	12345 → 50076 [SYN, ACK] Seq=0 Ack=1 Win=65160 Len=0 MSS=1460 SACK_PERM=1 TSval=3044173264 TSecr=1207759157 WS=128
3	0.007610587	172.20.10.2	172.20.10.6	TCP	66	50076 → 12345 [ACK] Seq=1 Ack=1 Win=131712 Len=0 TSval=1207759172 TSecr=3044173264
4	0.007612054	172.20.10.2	172.20.10.6	TCP	97	50076 → 12345 [PSH, ACK] Seq=1 Ack=1 Win=131712 Len=31 TSval=1207759172 TSecr=3044173264
5	0.007694125	172.20.10.6	172.20.10.2	TCP	66	12345 → 50076 [ACK] Seq=1 Ack=32 Win=65152 Len=0 TSval=3044173272 TSecr=1207759172
6	0.007998800	172.20.10.6	172.20.10.2	TCP	97	12345 → 50076 [PSH, ACK] Seq=1 Ack=32 Win=65152 Len=31 TSval=3044173272 TSecr=1207759172
7	0.008030301	172.20.10.6	172.20.10.2	TCP	66	12345 → 50076 [FIN, ACK] Seq=32 Ack=32 Win=65152 Len=0 TSval=3044173272 TSecr=1207759172
8	0.015407738	172.20.10.2	172.20.10.6	TCP	66	50076 → 12345 [ACK] Seq=32 Ack=32 Win=131712 Len=0 TSval=1207759180 TSecr=3044173272
9	0.015409204	172.20.10.2	172.20.10.6	TCP	66	50076 → 12345 [ACK] Seq=32 Ack=33 Win=131712 Len=0 TSval=1207759180 TSecr=3044173272
10	0.015410182	172.20.10.2	172.20.10.6	TCP	86	50076 → 12345 [PSH, ACK] Seq=32 Ack=33 Win=131712 Len=20 TSval=1207759180 TSecr=3044173272
11	0.015500286	172.20.10.6	172.20.10.2	TCP	54	12345 → 50076 [RST] Seq=33 Win=0 Len=0
12	0.015411579	172.20.10.2	172.20.10.6	TCP	66	50076 → 12345 [FIN, ACK] Seq=52 Ack=33 Win=131712 Len=0 TSval=1207759180 TSecr=3044173272

ניתוח החבילות

כזכור, פרוטוקול TCP מבטיח קשר מאובטח באמצעות לחיצת יד משולשת. אכן, שלוש החבילות הראשונות שתפסנו בהסנפה מממשות את השלבים של לחיצת היד המשולשת. על כן, אין להן שכבת אפליקציה - הן נועדו אך ורק למטרת הקמת החיבור. לאחר מכן, שאר החבילות מתחלקות לשתיים - חבילות ששולחות את המידע בין השרת והלקוח, וחבילות "תגובה" כחלק מפרוטוקול TCP, שנועדו לוודא את ההעברה או לסיים את התקשורת.

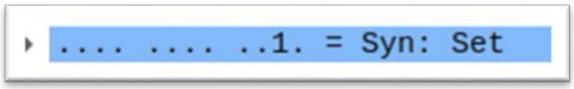
חבילה מספר 1:



נתבונן בשכבת התעבורה - תחילה מפורט כי ה-source port הינו 50076, פורט רנדומלי אשר מערכת ההפעלה הקצתה ללקוח (הרי הלקוח לא דורש פורט פרטני) וכי ה-destination port הינו 12345, פורט שביקשנו ממערכת ההפעלה להקצות עבור ריצת השרת. ממספרי הפורט ניתן להסיק שזוהי חבילה שנשלחה מהלקוח אל השרת.

לאחר מכן מפורטים המספרים הסידוריים בתחילת החיבור - בהודעה הראשונה. המספרים הסידוריים הם בעצם הדרך של השרת והלקוח לעקוב אחרי כמות המידע שנשלח והתקבל. בדיוק כמו בהודעה הראשונה עליה אנחנו מדברים, ה-sequence number נמצא בכל אחד מה-packets שנשלחים במהלך כל החיבור. במקרה שלנו התוכנית בחרה את ה-sequence number ההתחלתי להיות 1647536500. לאחר כל שליחת חבילה עם sequence number מסוים תוחזר חבילה המכילה acknowledgement number תואם המצביעה על החבילה הבאה שאנו מצפים לקבל על מנת להודיע כי המידע עד עכשיו אכן התקבל במלואו בהצלחה. במקרה שלנו ה-acknowledgement number הינו 0 כי באמת לא קיבלנו אף הודעה ולכן אנחנו מצפים להודעה הראשונה שמתחילה ב-0.

אחר כך מפורטת רשימת הדגלים, ונבחין שדגל ה-Syn דלוק:



כאשר לקוח רוצה לבסס תקשורת עם שרת מסוים באמצעות חיבור tcp, כאמור הם אמורים לבצע לחיצת יד משולשת. השלב הראשון של הלחיצה הוא שתחנת המקור (במקרה שלנו הלקוח) צריכה לשלוח הודעת SYN לשרת כחלק מפרוטוקול ה-tcp. כך השרת יודע כי בכוונת הלקוח לבצע את לחיצת הידיים המשולשת. בעצם כשדגל ה-syn דלוק, השרת והלקוח יכולים לסנכרן את המספר הסידורי והלקוח מסמן לשרת כי הוא מעוניין להתחיל את החיבור.

אחרי הדגלים, מפורט גודל החלון - מספר שמחושב בזמן התקשורת, המשתנה בהתאם לרוחב הפס וכמות המידע שהיעד מסוגל לעבד, שמייצג את כמות החבילות שאפשר לשלוח עד שצריך לעצור ולחכות לACK.

```
Window: 65535
```

מיד אחר כך מפורט ה-Checksum, שדה שנועד לבדוק את אמינות הנתונים. ערך השדה מחושב פעמיים - לפני השליחה ובעת קבלת החבילה. אם החישובים יוצאים שונים, סימן שהמידע תקול.

השדה האחרון בשכבה זו הוא שדה הoptions, שם בין השאר מפורט שדה ה-MSS שמייצג את כמות המידע המקסימלית שניתן לכתוב בשכבת האפליקציה בחבילה אחת.

```
▸ TCP Option - Maximum segment size: 1460 bytes
```

כעת, נסתכל על שכבת הרשת - שם אפשר לראות את כתובות ה-ip של המקור והיעד.

```
▸ Internet Protocol Version 4, Src: 172.20.10.2, Dst: 172.20.10.6
```

אכן, כתובות ה-ip הללו מסונכרנות עם הכתובות של המחשבים עליהם הרצנו את השרת ואת הלקוח, ושוב אפשר לראות כי המקור הוא אכן הלקוח, והיעד הוא אכן השרת.

לבסוף, נסתכל על שכבת הערוץ

```
▾ Ethernet II, Src: Apple_7b:c4:8c (f8:4d:89:7b:c4:8c), Dst: CloudNet_8f:6a:13 (dc:e9:94:8f:6a:13)
  ▸ Destination: CloudNet_8f:6a:13 (dc:e9:94:8f:6a:13)
  ▸ Source: Apple_7b:c4:8c (f8:4d:89:7b:c4:8c)
```

תחילה, מצוינות כתובות ה-MAC של המקור והיעד. אלו כתובות שונות מאחר והרצנו את הלקוח ואת השרת על מחשבים שונים, כפי שפורט בתרגיל.

נעיר שלאחר הרכבת שכבות התעבורה, הרשת והערוץ (לחבילה זו אין שכבת אפליקציה - כפי שהסברנו קודם), החבילה יורדת לשכבה הפיזית. שם, החבילה (שלמעשה היא כעת frame) מטיילת ברשת עד שהיא מגיעה למחשב היעד. שם, כל אחת משכבותיה מוסרות, ולבסוף היא מגיעה לשכבת התעבורה. מאחר וחבילה זו היא אחת משלוש החבילות המבססות את הקשר בין שני המחשבים, ולמעשה הראשונה מבניהן - כתגובה אליה תתקבל חבילת ה-ACK-SYN, עליה נפרט כעת.

חבילה 2:

```
3 0.007610587   172.20.10.2   172.20.10.6   TCP   66 50076 → 12345 [ACK] Seq=1 Ack=1 Win=131712 Len=0 TSval=1207759172 TSecr=3044173264
Transmission Control Protocol, Src Port: 12345, Dst Port: 50076, Seq: 0, Ack: 1, Len: 0
  Source Port: 12345
  Destination Port: 50076
  [Stream index: 0]
  [Conversation completeness: Complete, WITH_DATA (63)]
  [TCP Segment Len: 0]
  Sequence Number: 0    (relative sequence number)
  Sequence Number (raw): 224427309
  [Next Sequence Number: 1    (relative sequence number)]
  Acknowledgment Number: 1    (relative ack number)
  Acknowledgment number (raw): 1647536501
  1010 .... = Header Length: 40 bytes (10)
  Flags: 0x012 (SYN, ACK)
    000. .... .... = Reserved: Not set
    ...0 .... .... = Nonce: Not set
    .... 0... .... = Congestion Window Reduced (CWR): Not set
    .... .0.. .... = ECN-Echo: Not set
    .... ..0. .... = Urgent: Not set
    .... ...1 .... = Acknowledgment: Set
    .... .... 0... = Push: Not set
    .... .... .0.. = Reset: Not set
    .... .... ..1. = Syn: Set
      [Expert Info (Chat/Sequence): Connection establish acknowledge (SYN+ACK): server port 12345]
    .... .... ...0 = Fin: Not set
    [TCP Flags: ·******A··S·]
  Window: 65160
  [Calculated window size: 65160]
  Checksum: 0x3eb5 [unverified]
  [Checksum Status: Unverified]
  Urgent Pointer: 0
  Options: (20 bytes), Maximum segment size, SACK permitted, Timestamps, No-Operation (NOP), Window scale
    TCP Option - Maximum segment size: 1460 bytes
    TCP Option - SACK permitted
    TCP Option - Timestamps: TSval 3044173264, TSecr 1207759157
    TCP Option - No-Operation (NOP)
    TCP Option - Window scale: 7 (multiply by 128)
```

נתבונן בשכבת התעבורה - הפעם ה-source port הינו 12345 וה-destination port הינו 50076. לכן זוהי חבילה שנשלחה מהשרת ללקוח.

הפעם ה-sequence number הינו 224427309. עולה השאלה, מדוע ה-sequence number קטן יותר מההודעה הקודמת שהרי נשלחה חבילה חדשה? התשובה לכך היא שצריך לשים לב כי ההודעה הפעם היא מהשרת ללקוח. ואכן ה-sequence number של הלקוח הינו 224427309 (כל אחד מנהל רשימה משלו). הפעם ה-acknowledgement number הוא 1647536501. כפי שהסברנו מקודם המספר הזה באמת תואם ל-sequence number של החבילה הקודמת ומציין שאכן המידע הגיע בשלמותו.

בדומה להודעה הקודמת גם פה דגל ה-Syn דלוק. גם הפעם מאותה סיבה כמו בפעם הקודמת - על מנת להכיר בחיבור.

```
.... .... ..1. = Syn: Set
```

```
.... ...1 .... = Acknowledgment: Set
```

בשונה מהפעם הקודמת, גם דגל ה-ACK דלוק. מטרתו של הדגל לומר כי מהשלב הזה השדה שנמצא ב-acknowledgement number אכן תקף. ובאמת המספר שקיבלנו ב-acknowledgement number אומר שקיבלנו את הודעת ה-SYN שנשלחה מקודם כנדרש.

אחרי הדגלים, מפורט גודל החלון שהפעם הוא 65160.

```
Window: 65160
```

```
TCP Option - Maximum segment size: 1460 bytes
```

כמו כן, ה-maximum segment size הינו 1460.

כעת, נסתכל על שכבת הרשת - שם אפשר לראות את כתובות ה-ip של המקור והיעד.

```
▾ Internet Protocol Version 4, Src: 172.20.10.6, Dst: 172.20.10.2
```

בדומה למקודם, כתובות ה-ip הללו מסונכרנות עם הכתובות של המחשבים עליהם הרצנו את השרת ואת הלקוח, ועם מספרי הפורט. הפעם אפשר לראות כי המקור הוא השרת, והיעד הוא הלקוח.

לבסוף, נסתכל על שכבת הערוץ

```
▾ Ethernet II, Src: CloudNet_8f:6a:13 (dc:e9:94:8f:6a:13), Dst: Apple_7b:c4:8c (f8:4d:89:7b:c4:8c)
  ▸ Destination: Apple_7b:c4:8c (f8:4d:89:7b:c4:8c)
  ▸ Source: CloudNet_8f:6a:13 (dc:e9:94:8f:6a:13)
    Type: IPv4 (0x0800)
```

תחילה, מצוינות כתובות ה-MAC של המקור והיעד. אלו כתובות שונות מאחר והרצנו את הלקוח ואת השרת על מחשבים שונים, כפי שפורט בתרגיל.

כתגובה לחבילה זו נקבל את חבילת ה-ACK, בה המקור מידע את היעד על סיום ביסוס הקשר.

חבילה 3:

```
> Frame 3: 66 bytes on wire (528 bits), 66 bytes captured (528 bits) on interface wlp2s0, id 0
> Ethernet II, Src: Apple_7b:c4:8c (f8:4d:89:7b:c4:8c), Dst: CloudNet_8f:6a:13 (dc:e9:94:8f:6a:13)
> Internet Protocol Version 4, Src: 172.20.10.2, Dst: 172.20.10.6
v Transmission Control Protocol, Src Port: 50076, Dst Port: 12345, Seq: 1, Ack: 1, Len: 0
    Source Port: 50076
    Destination Port: 12345
    [Stream index: 0]
    [Conversation completeness: Complete, WITH_DATA (63)]
    [TCP Segment Len: 0]
    Sequence Number: 1    (relative sequence number)
    Sequence Number (raw): 1647536501
    [Next Sequence Number: 1    (relative sequence number)]
    Acknowledgment Number: 1    (relative ack number)
    Acknowledgment number (raw): 224427310
    1000 .... = Header Length: 32 bytes (8)
  v Flags: 0x010 (ACK)
      000. .... .... = Reserved: Not set
      ...0 .... .... = Accurate ECN: Not set
      .... 0... .... = Congestion Window Reduced: Not set
      .... .0.. .... = ECN-Echo: Not set
      .... ..0. .... = Urgent: Not set
      .... ...1 .... = Acknowledgment: Set
      .... .... 0... = Push: Not set
      .... .... .0.. = Reset: Not set
      .... .... ..0. = Syn: Not set
      .... .... ...0 = Fin: Not set
      [TCP Flags: ·····A····]
    Window: 2058
    [Calculated window size: 131712]
    [Window size scaling factor: 64]
    Checksum: 0x63f1 [unverified]
    [Checksum Status: Unverified]
    Urgent Pointer: 0
  > Options: (12 bytes), No-Operation (NOP), No-Operation (NOP), Timestamps
  > [Timestamps]
  > [SEQ/ACK analysis]
```

מאחר וההבדלים זניחים, לא נזכיר את שכבות הרשת והערוץ הפעם, אלא נתרכז בשכבת התעבורה - הפעם ה-source port הינו 50076 וה-destination port הינו 12345. לכן כמו במקרה הראשון זוהי חבילה שנשלחה מהלקוח אל השרת.

הפעם ה-sequence number הינו 164736501. זהו מספר הגדול ב-1 מהמספר הקודם וזה מפני שקיבלנו חבילה שמקימה את החיבור עם דגל ACK. הפעם ה-acknowledgement number הוא 1647536501. כפי שהסברנו מקודם המספר הזה באמת תואם ל-sequence number של החבילה הקודמת ומציין שאכן המידע הגיע בשלמותו.

```
.... ...1 .... = Acknowledgment: Set
```

בדומה להודעה הקודמת, דגל ה-ACK דלוק. מטרתו של הדגל לומר כי מהשלב הזה השדה שנמצא ב-acknowledgement number אכן תקף. ובאמת המספר שקיבלנו ב-acknowledgement number אומר שקיבלנו את הודעת ה-SYN שנשלחה מקודם כנדרש.

אחרי הדגלים, מפורט גודל החלון שהפעם הוא 2058.

```
Window: 2058
```

חבילה 4:

```
Frame 4: 97 bytes on wire (776 bits), 97 bytes captured (776 bits) on interface wlp2s0, id 0
Ethernet II, Src: Apple_7b:c4:8c (f8:4d:89:7b:c4:8c), Dst: CloudNet_8f:6a:13 (dc:e9:94:8f:6a:13)
Internet Protocol Version 4, Src: 172.20.10.2, Dst: 172.20.10.6
Transmission Control Protocol, Src Port: 50076, Dst Port: 12345, Seq: 1, Ack: 1, Len: 31
    Source Port: 50076
    Destination Port: 12345
    [Stream index: 0]
    [Conversation completeness: Complete, WITH_DATA (63)]
    [TCP Segment Len: 31]
    Sequence Number: 1    (relative sequence number)
    Sequence Number (raw): 1647536501
    [Next Sequence Number: 32    (relative sequence number)]
    Acknowledgment Number: 1    (relative ack number)
    Acknowledgment number (raw): 224427310
    1000 .... = Header Length: 32 bytes (8)
  Flags: 0x018 (PSH, ACK)
      000. .... .... = Reserved: Not set
      ...0 .... .... = Nonce: Not set
      .... 0... .... = Congestion Window Reduced (CWR): Not set
      .... .0.. .... = ECN-Echo: Not set
      .... ..0. .... = Urgent: Not set
      .... ...1 .... = Acknowledgment: Set
      .... .... 1... = Push: Set
      .... .... .0.. = Reset: Not set
      .... .... ..0. = Syn: Not set
      .... .... ...0 = Fin: Not set
      [TCP Flags: ·******AP***]
    Window: 2058
    [Calculated window size: 131712]
    [Window size scaling factor: 64]
    Checksum: 0x8525 [unverified]
    [Checksum Status: Unverified]
    Urgent Pointer: 0
  Options: (12 bytes), No-Operation (NOP), No-Operation (NOP), Timestamps
    TCP Option - No-Operation (NOP)
    TCP Option - No-Operation (NOP)
    TCP Option - Timestamps: TSval 1207759172, TSecr 3044173264
  [Timestamps]
  [SEQ/ACK analysis]
    TCP payload (31 bytes)
Data (31 bytes)
```

נתבונן בשכבת התעבורה - פעם נוספת ה-source port הינו 50076 וה-destination port הינו 12345. לכן זוהי חבילה שנשלחה מהלקוח אל השרת.

ה-sequence number שלנו עכשיו הוא 1647536501, זה מאוד הגיוני מהסיבה שבאמת לא קיבלנו כלום ולכן הוא נשאר בדיוק כמו בחבילה הקודמת ששלח הלקוח לשרת. ה-acknowledgement number שלנו עכשיו הוא 224427310, זה מפני שההודעה הקודמת שנשלחה ללקוח הייתה בעלת sequence number של 224427309.

כלומר כך tcp בעצם הודיע שהחבילה הבאה שהוא ממתין לה בעלת sequence number שמתחיל ב-224427310.

נשים לב שדגל ה-ACK דלוק כי באמת ה-acknowledgement number חשוב בשלב הזה של החיבור. ישנו עוד דגל דלוק, דגל ה-PSH. עקרונית לא באמת דיברנו על זה כל כך בהרצאה ולכן לא נפרט באמת אבל עקרונית דגלי ה-PSH מורים למערכת ההפעלה לשלוח (לצד השולח) ולקבל (לצד המקבל) את הנתונים באופן מיידי.

נתבונן בשכבת האפליקציה – נשים לב כי מועבר בחבילה מידע באורך 31 בתים. מידע זה הוא בעצם השמות שלנו ששלחנו מהלקוח לשרת.

הערה: מעתה ואילך, כל שאר החבילות עוקבות אחרי תבנית קבועה - נשלחת חבילה עם מידע לצד השני שנעקבת על ידי חבילת תגובה המוסרת אישור או בקשה להפסיק את התקשורת. מספר האישור יהיה המספר הסידורי של החבילה הקודמת בתוספת של 1. המספר הסידורי יגדל באורך המידע המועבר בחבילה. לכן, לא נפרט על כל חבילה. רעיונית הן מייצגות אותו דבר וההבדל היחידי יהיה המספרים עצמם (המספר הסידור ומספר האישור).

כעת נתבונן בחבילה מהסוג השני מבין השניים שהצגנו כעת, על מנת שתהיינה לנו דוגמאות מוחשיות לשני המקרים.

חבילה 7:

```
▸ Frame 7: 66 bytes on wire (528 bits), 66 bytes captured (528 bits) on interface wlp2s0, id 0
▸ Ethernet II, Src: CloudNet_8f:6a:13 (dc:e9:94:8f:6a:13), Dst: Apple_7b:c4:8c (f8:4d:89:7b:c4:8c)
▸ Internet Protocol Version 4, Src: 172.20.10.6, Dst: 172.20.10.2
▾ Transmission Control Protocol, Src Port: 12345, Dst Port: 50076, Seq: 32, Ack: 32, Len: 0
    Source Port: 12345
    Destination Port: 50076
    [Stream index: 0]
    [Conversation completeness: Complete, WITH_DATA (63)]
    [TCP Segment Len: 0]
    Sequence Number: 32    (relative sequence number)
    Sequence Number (raw): 224427341
    [Next Sequence Number: 33    (relative sequence number)]
    Acknowledgment Number: 32    (relative ack number)
    Acknowledgment number (raw): 1647536532
    1000 .... = Header Length: 32 bytes (8)
  ▾ Flags: 0x011 (FIN, ACK)
      000. .... .... = Reserved: Not set
      ...0 .... .... = Nonce: Not set
      .... 0... .... = Congestion Window Reduced (CWR): Not set
      .... .0.. .... = ECN-Echo: Not set
      .... ..0. .... = Urgent: Not set
      .... ...1 .... = Acknowledgment: Set
      .... .... 0... = Push: Not set
      .... .... .0.. = Reset: Not set
      .... .... ..0. = Syn: Not set
    ▸ .... .... ...1 = Fin: Set
    ▸ [TCP Flags: ·····A···F]
    Window: 509
    [Calculated window size: 65152]
    [Window size scaling factor: 128]
    Checksum: 0x69b7 [unverified]
    [Checksum Status: Unverified]
    Urgent Pointer: 0
  ▾ Options: (12 bytes), No-Operation (NOP), No-Operation (NOP), Timestamps
    ▸ TCP Option - No-Operation (NOP)
    ▸ TCP Option - No-Operation (NOP)
    ▸ TCP Option - Timestamps: TSval 3044173272, TSecr 1207759172
  ▸ [Timestamps]
```

השרת מעוניין לסגור את החיבור. כדי לעדכן את הלקוח הוא שולח הודעה עם דגל ה-FIN דלוק ובכך הלקוח יודע שהשרת אכן מעוניין לסגור את החיבור.

```
.... .... ...1 = Fin: Set
```

אבל לא רק שהשרת סוגר את החיבור, הוא צריך גם לעדכן את הלקוח שהחבילה האחרונה התקבלה. זאת הסיבה שדגל ה-ACK דלוק:

```
.... ...1 .... = Acknowledgment: Set
```

גודל החלון בעת שליחת אותה חבילה עומד על 509, מספר קטן יחסית.

```
Window: 509
```

עבור חבילה זאת, המספר הסידורי הוא 224427341 שזה בדיוק המספר הסידורי בחבילה הקודמת + אורך ההודעה שנשלחה באותה חבילה. מספר האישור הוא 1647536532 שזה המספר הסידורי של הודעה מספר 4 ועוד 1. זה מפני שקראנו בקבלת הודעה מספר ארבע, 31 בתים ולכן אנחנו מצביעים לבית הבא אותו אנו מצפים לקבל בהמשך.

```
Sequence Number (raw): 224427341
```

```
Acknowledgment number (raw): 1647536532
```

חבילה 11:

```
▸ Frame 11: 54 bytes on wire (432 bits), 54 bytes captured (432 bits) on interface wlp2s0, id 0
▸ Ethernet II, Src: CloudNet_8f:6a:13 (dc:e9:94:8f:6a:13), Dst: Apple_7b:c4:8c (f8:4d:89:7b:c4:8c)
▸ Internet Protocol Version 4, Src: 172.20.10.6, Dst: 172.20.10.2
▾ Transmission Control Protocol, Src Port: 12345, Dst Port: 50076, Seq: 33, Len: 0
    Source Port: 12345
    Destination Port: 50076
    [Stream index: 0]
    [Conversation completeness: Complete, WITH_DATA (63)]
    [TCP Segment Len: 0]
    Sequence Number: 33    (relative sequence number)
    Sequence Number (raw): 224427342
    [Next Sequence Number: 33    (relative sequence number)]
    Acknowledgment Number: 0
    Acknowledgment number (raw): 0
    0101 .... = Header Length: 20 bytes (5)
  ▾ Flags: 0x004 (RST)
      000. .... .... = Reserved: Not set
      ...0 .... .... = Nonce: Not set
      .... 0... .... = Congestion Window Reduced (CWR): Not set
      .... .0.. .... = ECN-Echo: Not set
      .... ..0. .... = Urgent: Not set
      .... ...0 .... = Acknowledgment: Not set
      .... .... 0... = Push: Not set
    ▸ .... .... .1.. = Reset: Set
      .... .... ..0. = Syn: Not set
      .... .... ...0 = Fin: Not set
      [TCP Flags: ·····R··]
    Window: 0
    [Calculated window size: 0]
    [Window size scaling factor: 128]
    Checksum: 0xc52b [unverified]
    [Checksum Status: Unverified]
    Urgent Pointer: 0
  ▸ [Timestamps]
```

בדיוק כמו שלוש החבילות הראשונות, מטרת חבילה זאת אינה העברת מידע בשכבת האפליקציה. נזכר כי בשלב יותר מוקדם של התוכנית השרת שלח חבילת FIN ובכך סגר את החיבור. הבעיה היא שהלקוח עדיין שולח מידע לשרת כאשר החיבור סגור. זאת כמובן התנהגות בעייתית ולכן במצב הזה נשלחת הודעה עם דגל ה-RST. נשים לב כי חבילת RST יכולה להישלח במהלך זרם של הודעות. כלומר זה סיום פחות "יפה" של החיבור ולעיתים אף מסמל נטישה של צד אחד את החיבור.

```
> .... .... .1.. = Reset: Set
```

גודל החלון הינו 0 שכן סגרנו את החיבור ואנחנו לא מתכוונים לשלוח שום חבילה נוספת.

```
Window: 0
```

ה-acknowledgement number הוא 0 כי אנחנו לא שולחים ack על אף הודעה בגלל, כמו שכבר הסברנו, החיבור נסגר מבחינת השרת. ה-sequence number שלנו עכשיו הוא 224427342, זה מאוד הגיוני מהסיבה שבאמת בחבילה הקודמת המספר הסידורי היה קטן ב-1.

```
Acknowledgment number (raw): 0
```

```
Sequence Number (raw): 224427342
```

חבילה 12:

```
▸ Frame 12: 66 bytes on wire (528 bits), 66 bytes captured (528 bits) on interface wlp2s0, id 0
▸ Ethernet II, Src: Apple_7b:c4:8c (f8:4d:89:7b:c4:8c), Dst: CloudNet_8f:6a:13 (dc:e9:94:8f:6a:13)
▸ Internet Protocol Version 4, Src: 172.20.10.2, Dst: 172.20.10.6
▾ Transmission Control Protocol, Src Port: 50076, Dst Port: 12345, Seq: 52, Ack: 33, Len: 0
    Source Port: 50076
    Destination Port: 12345
    [Stream index: 0]
    [Conversation completeness: Complete, WITH_DATA (63)]
    [TCP Segment Len: 0]
    Sequence Number: 52    (relative sequence number)
    Sequence Number (raw): 1647536552
    [Next Sequence Number: 53    (relative sequence number)]
    Acknowledgment Number: 33    (relative ack number)
    Acknowledgment number (raw): 224427342
    1000 .... = Header Length: 32 bytes (8)
  ▾ Flags: 0x011 (FIN, ACK)
      000. .... .... = Reserved: Not set
      ...0 .... .... = Nonce: Not set
      .... 0... .... = Congestion Window Reduced (CWR): Not set
      .... .0.. .... = ECN-Echo: Not set
      .... ..0. .... = Urgent: Not set
      .... ...1 .... = Acknowledgment: Set
      .... .... 0... = Push: Not set
      .... .... .0.. = Reset: Not set
      .... .... ..0. = Syn: Not set
    ▸ .... .... ...1 = Fin: Set
    ▸ [TCP Flags: ·····A···F]
    Window: 2058
    [Calculated window size: 131712]
    [Window size scaling factor: 64]
    Checksum: 0x638d [unverified]
    [Checksum Status: Unverified]
    Urgent Pointer: 0
  ▾ Options: (12 bytes), No-Operation (NOP), No-Operation (NOP), Timestamps
    ▸ TCP Option - No-Operation (NOP)
    ▸ TCP Option - No-Operation (NOP)
    ▸ TCP Option - Timestamps: TSval 1207759180, TSecr 3044173272
  ▸ [Timestamps]
```

הלקוח סיים לשלוח את כל המידע שתכנן לשלוח לשרת (לא בהכרח כל החבילות הגיעו כמו שהסברנו בחבילות קודמות בפרט בחבילה 10). הוא רוצה לסיים את החיבור ולכן שולח חבילה עם FIN ו-ACK כאשר על ACK כבר דיברנו, והוא בעצם מאשר את קבלת המידע של אחת מההודעות הקודמות ו-FIN מסמל את סיום החיבור.

```
.... ...1 .... = Acknowledgment: Set
```

```
.... .... ...1 = Fin: Set
```

הפעם ה-windows size הוא 2058.

```
Window: 2058
```

ה-acknowledgement number הוא 224427342 בדיוק ה-sequence number של החבילה הקודמת. כלומר החבילה הגיעה ללקוח בשלמותה. ה-sequence number שלנו עכשיו הוא 1647536552, וזה באמת מתאים לחבילה הקודמת.

```
Acknowledgment number (raw): 224427342
```

```
Sequence Number (raw): 1647536552
```

חבילה 13:

```
› Frame 13: 54 bytes on wire (432 bits), 54 bytes captured (432 bits) on interface wlp2s0, id 0
› Ethernet II, Src: CloudNet_8f:6a:13 (dc:e9:94:8f:6a:13), Dst: Apple_7b:c4:8c (f8:4d:89:7b:c4:8c)
› Internet Protocol Version 4, Src: 172.20.10.6, Dst: 172.20.10.2
› Transmission Control Protocol, Src Port: 12345, Dst Port: 50076, Seq: 33, Len: 0
    Source Port: 12345
    Destination Port: 50076
    [Stream index: 0]
    [Conversation completeness: Complete, WITH_DATA (63)]
    [TCP Segment Len: 0]
    Sequence Number: 33    (relative sequence number)
    Sequence Number (raw): 224427342
    [Next Sequence Number: 33    (relative sequence number)]
    Acknowledgment Number: 0
    Acknowledgment number (raw): 0
    0101 .... = Header Length: 20 bytes (5)
  ⌄ Flags: 0x004 (RST)
      000. .... .... = Reserved: Not set
      ...0 .... .... = Nonce: Not set
      .... 0... .... = Congestion Window Reduced (CWR): Not set
      .... .0.. .... = ECN-Echo: Not set
      .... ..0. .... = Urgent: Not set
      .... ...0 .... = Acknowledgment: Not set
      .... .... 0... = Push: Not set
    › .... .... .1.. = Reset: Set
      .... .... ..0. = Syn: Not set
      .... .... ...0 = Fin: Not set
      [TCP Flags: ·····R··]
    Window: 0
    [Calculated window size: 0]
    [Window size scaling factor: 128]
    Checksum: 0xc52b [unverified]
    [Checksum Status: Unverified]
    Urgent Pointer: 0
  › [Timestamps]
```

חבילה 11 מגיע לשרת אבל חשוב לזכור שהשרת עדיין סיים את התוכנית! לכן עדיין הוא לא מקבל את ההודעה והוא פעם נוספת מאותה סיבה כמו בחבילה 10 שולח חבילה עם דגל RST דלוק.

```
.... .... .1.. = Reset: Set
```

החלון עדיין בגודל 0 מפני שבכלל סגרנו את החיבור והשרת אינו מעוניין לשלוח עוד הודעות ללקוח.

```
Window: 0
```

ה-acknowledgement number מאופס ל-0 מפני שהחיבור אמור להיות סגור ואנחנו לא מצפים לאף חבילה להגיע.

```
Acknowledgment Number: 0
```

ה-sequence number הוא 224427342. זה בדיוק אותו מספר שהיה אצל החבילה הקודמת ששלח השרת וזה בדיוק מה שהיינו מצפים שכן השרת לא מקבל הודעות חדשות (הוא סגר את החיבור).

```
Sequence Number (raw): 224427342
```

חלק א'
חלק שני

גרסה ראשונה – V1

הבנת קוד השרת: תחילה, מוגדרת כתובת ה-ip של השרת להיות 0.0.0.0. כלומר, לסוקט של השרת מוקמת גישה לכל כרטיסי הרשת, והוא יקבל את התעבורה במחשב אשר מכוונת לפורט שלו - אותו הוא מקבל כארגומנט לתוכנית. לאחר מכן מוגדר קבוע גלובלי BUFFER_SIZE ומאותחל עם הערך 1024. קבוע זה מייצג את כמות המידע המקסימלית שהשרת מאפשר לקרוא בכל פעם (בבתים). לאחר מכן, נוצר סוקט המתקשר בפרוטוקול TCP ונותנים לו את הכתובת שהגדרנו. הסוקט נכנס למצב האזנה שם הוא מאזין לכל היותר ללקוח אחד כל פעם. כעת, השרת נכנס ללולאה אינסופית בה הוא מקבל לקוח, קורא ממנו מידע (וכל קריאה קוראת לכל היותר BUFFER_SIZE בתים) כל עוד הלקוח שולח, מדפיס את המידע, שולח אותו בחזרה ללקוח באותיות גדולות, סוגר את התקשורת וחוזר חלילה.

הבנת קוד הלקוח: הלקוח יוצר סוקט המתקשר גם הוא בפרוטוקול TCP, ומתחבר איתו לשרת בכתובת ip ובפורט שמקבלים כארגומנטים לתוכנית. כמו כן גם בלקוח מוגדר קבוע גלובלי BUFFER_SIZE, זהה לקבוע בשרת. לאחר החיבור, הלקוח שולח לשרת את ההודעה "Hello, World!" ומדפיס את התגובה של השרת.

הסברים על התעבורה:

Apply a display filter ... <Ctrl-/>

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	172.20.10.2	172.20.10.6	TCP	78	50135 → 12346 [SYN] Seq=0 Win=65535 Len=0 MSS=1460 WS=64 TSval=10920904 TSecr=0 SACK_PERM=1
2	0.000054897	172.20.10.6	172.20.10.2	TCP	74	12346 → 50135 [SYN, ACK] Seq=0 Ack=1 Win=65160 Len=0 MSS=1460 SACK_PERM=1 TSval=3045744293 TSecr=10920904 WS=128
3	0.007621298	172.20.10.2	172.20.10.6	TCP	66	50135 → 12346 [ACK] Seq=1 Ack=1 Win=131712 Len=0 TSval=10920936 TSecr=3045744293
4	0.007622694	172.20.10.2	172.20.10.6	TCP	79	50135 → 12346 [PSH, ACK] Seq=1 Ack=1 Win=131712 Len=13 TSval=10920936 TSecr=3045744293
5	0.007702456	172.20.10.6	172.20.10.2	TCP	66	12346 → 50135 [ACK] Seq=1 Ack=14 Win=65152 Len=0 TSval=3045744300 TSecr=10920936
6	0.008012144	172.20.10.6	172.20.10.2	TCP	79	12346 → 50135 [PSH, ACK] Seq=1 Ack=14 Win=65152 Len=13 TSval=3045744301 TSecr=10920936
7	0.014892677	172.20.10.2	172.20.10.6	TCP	66	50135 → 12346 [ACK] Seq=14 Ack=14 Win=131712 Len=0 TSval=10920943 TSecr=3045744301
8	0.014894144	172.20.10.2	172.20.10.6	TCP	66	50135 → 12346 [FIN, ACK] Seq=14 Ack=14 Win=131712 Len=0 TSval=10920943 TSecr=3045744301
9	0.015049826	172.20.10.6	172.20.10.2	TCP	66	12346 → 50135 [FIN, ACK] Seq=14 Ack=15 Win=65152 Len=0 TSval=3045744308 TSecr=10920943
10	0.102753104	172.20.10.2	172.20.10.6	TCP	66	50135 → 12346 [ACK] Seq=15 Ack=15 Win=131712 Len=0 TSval=10920951 TSecr=3045744308

כפי שהסברנו בחלק הראשון של חלק א', כאשר שני מחשבים מבסיסים קשר ביניהם עם פרוטוקול TCP, הם מבצעים לחיצת יד משולשת. תחילה, הלקוח שולח לשרת חבילת SYN לפתיחת הקשר - להסתנכרן עם השרת. לאחר מכן, השרת שולח ללקוח חבילת SYN-ACK בה הוא מאשר את קבלת חבילת הלקוח, ופתיחת קשר ביניהם. כדי לחתום את הלחיצה, הלקוח שולח חבילת ACK לשרת - בה הוא מאשר את האישור, ובכך התקשורת מבוססת.

כעת, כאשר התקשורת מבוססת השניים מתחילים "לדבר": הלקוח שולח את ההודעה "Hello, World!" בחבילה הרביעית, והחבילה החמישית היא חבילת אישור מצד השרת. לאחר מכן השרת שולח ללקוח את תגובתו (המרת המסר לאותיות גדולות) ובחבילה השביעית הלקוח מאשר שהוא קיבל את תגובת השרת. לבסוף, הלקוח שולח חבילת FIN לשרת בה הוא מודיע לו שהוא מסיים את התקשורת. השרת מגיב בחבילת ACK-FIN בה הוא מאשר את הגעת חבילת ה-FIN של הלקוח, ומבקש גם הוא לסיים את התקשורת. החבילה האחרונה היא חבילת ACK של הלקוח בה הוא מאשר את הגעת חבילת ה-FIN של השרת.

גרסה שנייה – V2

הבנת קוד השרת: הקוד של גרסה זו זהה לחלוטין לקוד של הגרסה הקודמת - עם השינוי שBUFFER_SIZE מאותחל ל-5. כלומר כמות הבתים המקסימלית שהשרת יכול לקרוא בכל recv היא 5.

הבנת קוד הלקוח: גם בקוד הלקוח לא בוצעו כמעט שינויים. ההבדלים היחידים הם שהפעם ההודעה שהלקוח שולח לשרת היא "World!, Hello World!" וגם בלקוח כמות הבתים המקסימלית שאפשר לקרוא היא 5.

הסברים על התעבורה:

1	0.000000000	172.20.10.2	172.20.10.6	TCP	78	50152 → 12343 [SYN] Seq=0 Win=65535 Len=0 MSS=1460 WS=64 TSval=1476939290 TSecr=0 SACK_PERM=1
2	0.000056783	172.20.10.6	172.20.10.2	TCP	74	12343 → 50152 [SYN, ACK] Seq=0 Ack=1 Win=65160 Len=0 MSS=1460 SACK_PERM=1 TSval=3045907010 TSecr=1476939290 WS=128
3	0.006511253	172.20.10.2	172.20.10.6	TCP	66	50152 → 12343 [ACK] Seq=1 Ack=1 Win=131712 Len=0 TSval=1476939298 TSecr=3045907010
4	0.006512720	172.20.10.2	172.20.10.6	TCP	86	50152 → 12343 [PSH, ACK] Seq=1 Ack=1 Win=131712 Len=20 TSval=1476939298 TSecr=3045907010
5	0.006597231	172.20.10.6	172.20.10.2	TCP	66	12343 → 50152 [ACK] Seq=1 Ack=21 Win=65152 Len=0 TSval=3045907017 TSecr=1476939298
6	0.006954621	172.20.10.6	172.20.10.2	TCP	71	12343 → 50152 [PSH, ACK] Seq=1 Ack=21 Win=65152 Len=5 TSval=3045907017 TSecr=1476939298
7	0.013069860	172.20.10.2	172.20.10.6	TCP	66	50152 → 12343 [ACK] Seq=21 Ack=6 Win=131712 Len=0 TSval=1476939304 TSecr=3045907017
8	0.013120357	172.20.10.6	172.20.10.2	TCP	81	12343 → 50152 [PSH, ACK] Seq=6 Ack=21 Win=65152 Len=15 TSval=3045907023 TSecr=1476939304
9	0.013070837	172.20.10.2	172.20.10.6	TCP	66	50152 → 12343 [FIN, ACK] Seq=21 Ack=6 Win=131712 Len=0 TSval=1476939304 TSecr=3045907017
10	0.013327024	172.20.10.6	172.20.10.2	TCP	66	12343 → 50152 [FIN, ACK] Seq=21 Ack=22 Win=65152 Len=0 TSval=3045907023 TSecr=1476939304
11	0.024018006	172.20.10.2	172.20.10.6	TCP	54	50152 → 12343 [RST] Seq=21 Win=0 Len=0
12	0.024019403	172.20.10.2	172.20.10.6	TCP	54	50152 → 12343 [RST] Seq=22 Win=0 Len=0

כפי שהסברנו בחלק הראשון של שלב א', כאשר שני מחשבים מבסיסים קשר ביניהם עם פרוטוקול TCP, הם מבצעים לחיצת יד משולשת. תחילה, הלקוח שולח לשרת חבילת SYN לפתיחת הקשר - להסתנכרן עם השרת. לאחר מכן, השרת שולח ללקוח חבילת SYN-ACK בה הוא מאשר את קבלת חבילת הלקוח, ופתיחת קשר ביניהם. כדי לחתום את הלחיצה, הלקוח שולח חבילת ACK לשרת - בה הוא מאשר את האישור, ובכך התקשורת מבוססת.

כעת, כאשר התקשורת מבוססת השניים מתחילים "לדבר": הלקוח שולח את ההודעה "World! Hello, World!" בחבילה הרביעית, והחבילה החמישית היא חבילת אישור מצד השרת. לאחר מכן השרת שולח ללקוח את תגובתו. לכאורה, אנו מצפים שההודעה תועבר ב-4 חבילות שונות (כי השרת מבצע קריאות בגודל 5 בתים, ואורך המידע הוא 20 בתים). אולם, פרוטוקול TCP מאחד את ההודעה ושולח אותה בשתי פקטות שונות, פשוט ארבע הקריאות הן מהבאפר של TCP, ואותן WireShark לא מציג. אם כן, בחבילה השישית השרת שולח את החלק הראשון של תגובתו, החבילה השביעית היא אישור מצד הלקוח והחבילה השמינית היא יתר התגובה. לאחר מכן נשלחת חבילה עם דגלי FIN ו-ACK דלוקים שמסמנת שהלקוח רוצה לסיים את התקשורת, והוא מאשר את הגעת החבילה הקודמת. השרת מגיב באופן דומה. אולם, התקשורת נסגרת בטרם עת - ולכן כאשר הלקוח רוצה להגיב, הוא שולח חבילות עם דגל הRST דולק, כי אין לו למי לשלוח.

גרסה שלישית – V3

הבנת קוד השרת: הקוד של גרסה זו כמעט זהה לקוד של הגרסה הראשונה - רק שהפעם המידע שהשרת שולח לקוח, הוא המידע המקורי משורשר עם עצמו 1000 פעמים.

הבנת קוד הלקוח: גם קוד הלקוח דומה למדי לקוד בגרסה הראשונה. ההבדלים הם שבמקום לקרוא מידע פעם אחת, מקבלים פעמיים (וגם מדפיסים אותו פעמיים).

הסברים על התעבורה:

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	172.20.10.2	172.20.10.6	TCP	78	50157 → 12344 [SYN] Seq=0 Win=65535 Len=0 MSS=1460 WS=64 TSval=1900716398 TSecr=0 SACK_PERM=1
2	0.000058176	172.20.10.6	172.20.10.2	TCP	74	12344 → 50157 [SYN, ACK] Seq=0 Ack=1 Win=65160 Len=0 MSS=1460 SACK_PERM=1 TSval=3046026125 TSecr=1900716398 WS=128
3	0.006255060	172.20.10.2	172.20.10.6	TCP	66	50157 → 12344 [ACK] Seq=1 Ack=1 Win=131712 Len=0 TSval=1900716408 TSecr=3046026125
4	0.006256457	172.20.10.2	172.20.10.6	TCP	79	50157 → 12344 [PSH, ACK] Seq=1 Ack=1 Win=131712 Len=13 TSval=1900716408 TSecr=3046026125
5	0.006333001	172.20.10.6	172.20.10.2	TCP	66	12344 → 50157 [ACK] Seq=1 Ack=14 Win=65152 Len=0 TSval=3046026131 TSecr=1900716408
6	0.006683597	172.20.10.6	172.20.10.2	TCP	1514	12344 → 50157 [ACK] Seq=1 Ack=14 Win=65152 Len=1448 TSval=3046026132 TSecr=1900716408
7	0.006703013	172.20.10.6	172.20.10.2	TCP	1514	12344 → 50157 [ACK] Seq=1449 Ack=14 Win=65152 Len=1448 TSval=3046026132 TSecr=1900716408
8	0.006706854	172.20.10.6	172.20.10.2	TCP	1514	12344 → 50157 [ACK] Seq=2897 Ack=14 Win=65152 Len=1448 TSval=3046026132 TSecr=1900716408
9	0.006709298	172.20.10.6	172.20.10.2	TCP	1514	12344 → 50157 [ACK] Seq=4345 Ack=14 Win=65152 Len=1448 TSval=3046026132 TSecr=1900716408
10	0.006711743	172.20.10.6	172.20.10.2	TCP	1514	12344 → 50157 [PSH, ACK] Seq=5793 Ack=14 Win=65152 Len=1448 TSval=3046026132 TSecr=1900716408
11	0.006726828	172.20.10.6	172.20.10.2	TCP	1514	12344 → 50157 [ACK] Seq=7241 Ack=14 Win=65152 Len=1448 TSval=3046026132 TSecr=1900716408
12	0.006728784	172.20.10.6	172.20.10.2	TCP	1514	12344 → 50157 [ACK] Seq=8689 Ack=14 Win=65152 Len=1448 TSval=3046026132 TSecr=1900716408
13	0.006731158	172.20.10.6	172.20.10.2	TCP	1514	12344 → 50157 [ACK] Seq=10137 Ack=14 Win=65152 Len=1448 TSval=3046026132 TSecr=1900716408
14	0.006765030	172.20.10.6	172.20.10.2	TCP	1482	12344 → 50157 [PSH, ACK] Seq=11585 Ack=14 Win=65152 Len=1416 TSval=3046026132 TSecr=1900716408
15	0.014731375	172.20.10.2	172.20.10.6	TCP	66	50157 → 12344 [ACK] Seq=14 Ack=13001 Win=118720 Len=0 TSval=1900716418 TSecr=3046026132
16	0.017800764	172.20.10.2	172.20.10.6	TCP	54	50157 → 12344 [RST, ACK] Seq=14 Ack=13001 Win=120064 Len=0

כפי שהסברנו בחלק הראשון של שלב א', כאשר שני מחשבים מבסיסים קשר ביניהם עם פרוטוקול TCP, הם מבצעים לחיצת יד משולשת. תחילה, הלקוח שולח לשרת חבילת SYN לפתיחת הקשר - להסתנכרן עם השרת. לאחר מכן, השרת שולח ללקוח חבילת SYN-ACK בה הוא מאשר את קבלת חבילת הלקוח, ופתיחת קשר ביניהם. כדי לחתום את הלחיצה, הלקוח שולח חבילת ACK לשרת - בה הוא מאשר את האישור, ובכך התקשורת מבוססת.

לאחר שהתקשורת מבוססת, הלקוח שולח את ההודעה "Hello, World!" בחבילה הרביעית, והחבילה החמישית היא חבילת אישור מצד השרת. לאחר מכן השרת שולח ללקוח את תגובתו. התגובה שלו היא "HELLO, WORLD!" משורשר אלף פעמים. והרי ש-TCP מחלק את תגובה זו ל-8 חבילות שונות אותן הוא שולח ברצף. לאחר מכן, החבילה הארבע עשר היא חבילת אישור מצד הלקוח, בה הוא שולח ACK מצטבר על כל החבילות הקודמות. הלקוח קורא מהבאפר 1024 בתים פעמיים, אבל עדיין יש מעל 10,000 בתים אותם הוא לא קרא, לכן נשלחת חבילת RST להודיע על כך לשרת.

גרסה רביעית – V4

הבנת קוד השרת: שוב, הקוד דומה מאוד לקוד של הגרסה הראשונה. ההבדל הפעם הוא שלפני כל recv, השרת מחכה 5 שניות.

הבנת קוד הלקוח: קוד הלקוח דומה לגרסתו הראשונה, רק שבגרסה זו שולחים את המידע משורשר עם עצמו 10 פעמים, 4 פעמים.

הסברים על התעבורה:

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	172.20.10.2	172.20.10.6	TCP	78	50167 → 12345 [SYN] Seq=0 Win=65535 Len=0 MSS=1460 WS=64 TSval=492802827 TSecr=0 SACK_PERM=1
2	0.000084088	172.20.10.6	172.20.10.2	TCP	74	12345 → 50167 [SYN, ACK] Seq=0 Ack=1 Win=65160 Len=0 MSS=1460 SACK_PERM=1 TSval=3046245371 TSecr=492802827 WS=128
3	0.000580412	172.20.10.2	172.20.10.6	TCP	66	50167 → 12345 [ACK] Seq=1 Ack=1 Win=131712 Len=0 TSval=492802837 TSecr=3046245371
4	0.000581879	172.20.10.2	172.20.10.6	TCP	196	50167 → 12345 [PSH, ACK] Seq=1 Ack=1 Win=131712 Len=130 TSval=492802837 TSecr=3046245371
5	0.006718557	172.20.10.6	172.20.10.2	TCP	66	12345 → 50167 [ACK] Seq=1 Ack=131 Win=65152 Len=0 TSval=3046245377 TSecr=492802837
6	0.013047822	172.20.10.2	172.20.10.6	TCP	456	50167 → 12345 [PSH, ACK] Seq=131 Ack=1 Win=131712 Len=390 TSval=492802844 TSecr=3046245377
7	0.013087421	172.20.10.6	172.20.10.2	TCP	66	12345 → 50167 [ACK] Seq=1 Ack=521 Win=64768 Len=0 TSval=3046245384 TSecr=492802844
8	5.008585098	172.20.10.6	172.20.10.2	TCP	586	12345 → 50167 [PSH, ACK] Seq=1 Ack=521 Win=64768 Len=520 TSval=3046250379 TSecr=492802844
9	5.086299566	172.20.10.2	172.20.10.6	TCP	66	50167 → 12345 [ACK] Seq=521 Ack=521 Win=131200 Len=0 TSval=492807918 TSecr=3046250379
10	5.089723728	172.20.10.2	172.20.10.6	TCP	66	50167 → 12345 [FIN, ACK] Seq=521 Ack=521 Win=131200 Len=0 TSval=492807918 TSecr=3046250379
11	5.132358206	172.20.10.6	172.20.10.2	TCP	66	12345 → 50167 [ACK] Seq=521 Ack=522 Win=64768 Len=0 TSval=3046250503 TSecr=492807918
12	10.013979951	172.20.10.6	172.20.10.2	TCP	66	12345 → 50167 [FIN, ACK] Seq=521 Ack=522 Win=64768 Len=0 TSval=3046255385 TSecr=492807918
13	10.107499105	172.20.10.2	172.20.10.6	TCP	66	50167 → 12345 [ACK] Seq=522 Ack=522 Win=131200 Len=0 TSval=492812936 TSecr=3046255385

כפי שהסברנו בחלק הראשון של שלב א', כאשר שני מחשבים מבסיסים קשר ביניהם עם פרוטוקול TCP, הם מבצעים לחיצת יד משולשת. תחילה, הלקוח שולח לשרת חבילת SYN לפתיחת הקשר - להסתנכרן עם השרת. לאחר מכן, השרת שולח ללקוח חבילת SYN-ACK בה הוא מאשר את קבלת חבילת הלקוח, ופתיחת קשר ביניהם. כדי לחתום את הלחיצה, הלקוח שולח חבילת ACK לשרת - בה הוא מאשר את האישור, ובכך התקשורת מבוססת.

אחרי לחיצת היד המשולשת, אנו מצפים לראות ארבע חבילות נשלחות מהלקוח לשרת, שבכל אחת מהן ההודעה "Hello, World!" משורשרת עם עצמה 10 פעמים. אולם, כפי שלמדנו, לעיתים TCP יאחד חבילות. אכן, גם כאן המידע מתאחד ונשלח בשתי חבילות ולא 4. אם כן, בחבילה הרביעית החצי הראשון של הודעות הלקוח נשלח, החבילה החמישית היא חבילת אישור, החבילה השישית היא החצי השני של הודעות הלקוח והחבילה השביעית היא גם חבילת אישר מצד השרת. לאחר מכן, הלקוח שולח כתגובה את כל המידע שקיבל באותיות גדולות - מידע זה מועבר בחבילה השמינית. החבילה התשיעית היא חבילת אישור מהלקוח. לבסוף, הלקוח שולח חבילת FIN לשרת בה הוא מודיע לו שהוא מסיים את התקשורת. השרת מגיב בחבילת ACK-FIN בה הוא מאשר את הגעת חבילת ה-FIN של הלקוח, ומבקש גם הוא לסיים את התקשורת. החבילה האחרונה היא חבילת ACK של הלקוח בה הוא מאשר את הגעת חבילת ה-FIN של השרת.

נעיר כי כאשר השרת נמצאת בהשהיה, עדיין הלקוח שולח חבילות שתוכנם נשמר כאשר השרת יוצא מההשהיה. המצב מתאפשר מפני שרק שכבת האפליקציה נמצאת בהשהיה, אבל מאחורי הקלעים המידע נשמר ב-buffer. כאשר השרת יוצא מההשהיה, הוא יכול לעשות receive למידע שנמצא ב-buffer.

חלק ב'

ניתוח התעבורה

בחלק זה, התבקשנו להריץ את הקוד שכתבנו, להסניף את התעבורה ב-WireShark ולנתח אותה. אם כן, תחילה נציג את דוגמת ההרצה שלנו: התחלנו בלבקש מהשרת את הקובץ '/' - שלפי הנחיות התרגיל הוא למעשה הקובץ 'index.html'. לאחר מכן ביקשנו את הקובץ 'a/2.jpg' - אחת מהתמונות על המחשב. אחר כך ביקשנו את הקובץ 'redirect', שלפי הנחיות התרגיל מפנה אותנו לקובץ result.html. לבסוף, ביקשנו מהשרת קובץ שלא קיים - a.html.

נציין כי לאחר שהרצנו את התוכנית והסנפנו את התעבורה ב-wireshark סיננו את החבילות עם:

```
tcp && ip.src == 127.0.0.1 && ip.dst ==127.0.0.1 && tcp.port == 8080
```

שהרי השרת והלקוח רצים על אותו מחשב.

בקשת '/' ('index.html'):

1	0.000000000	127.0.0.1	127.0.0.1	TCP	74	37990 → 8080 [SYN] Seq=0 Win=65495 Len=0 MSS=65495 SACK_PERM=1 TSval=3864195187 TSecr=0 WS=128
2	0.000013759	127.0.0.1	127.0.0.1	TCP	74	8080 → 37990 [SYN, ACK] Seq=0 Ack=1 Win=65483 Len=0 MSS=65495 SACK_PERM=1 TSval=3864195187 TSecr=3864195187 WS=128
3	0.000027168	127.0.0.1	127.0.0.1	TCP	66	37990 → 8080 [ACK] Seq=1 Ack=1 Win=65536 Len=0 TSval=3864195187 TSecr=3864195187
4	0.000616705	127.0.0.1	127.0.0.1	HTTP	724	GET / HTTP/1.1
5	0.000636820	127.0.0.1	127.0.0.1	TCP	66	8080 → 37990 [ACK] Seq=1 Ack=659 Win=64896 Len=0 TSval=3864195188 TSecr=3864195188
6	0.000834543	127.0.0.1	127.0.0.1	TCP	130	8080 → 37990 [PSH, ACK] Seq=1 Ack=659 Win=65536 Len=64 TSval=3864195188 TSecr=3864195188 [TCP segment of a reassembled…
7	0.000890835	127.0.0.1	127.0.0.1	TCP	66	37990 → 8080 [ACK] Seq=659 Ack=65 Win=65536 Len=0 TSval=3864195188 TSecr=3864195188
8	0.000900334	127.0.0.1	127.0.0.1	HTTP	225	HTTP/1.1 200 OK
9	0.000907528	127.0.0.1	127.0.0.1	TCP	66	37990 → 8080 [ACK] Seq=659 Ack=224 Win=65408 Len=0 TSval=3864195188 TSecr=3864195188
10	0.079582449	127.0.0.1	127.0.0.1	HTTP	650	GET /favicon.ico HTTP/1.1
11	0.079799099	127.0.0.1	127.0.0.1	TCP	131	8080 → 37990 [PSH, ACK] Seq=224 Ack=1243 Win=65536 Len=65 TSval=3864195267 TSecr=3864195267 [TCP segment of a reassemb…
12	0.079813836	127.0.0.1	127.0.0.1	TCP	66	37990 → 8080 [ACK] Seq=1243 Ack=289 Win=65536 Len=0 TSval=3864195267 TSecr=3864195267
13	0.079831576	127.0.0.1	127.0.0.1	HTTP	3192	HTTP/1.1 200 OK
14	0.079838141	127.0.0.1	127.0.0.1	TCP	66	37990 → 8080 [ACK] Seq=1243 Ack=3415 Win=63360 Len=0 TSval=3864195267 TSecr=3864195267
15	1.080947044	127.0.0.1	127.0.0.1	TCP	66	8080 → 37990 [FIN, ACK] Seq=3415 Ack=1243 Win=65536 Len=0 TSval=3864196268 TSecr=3864195267
16	1.127411573	127.0.0.1	127.0.0.1	TCP	66	37990 → 8080 [ACK] Seq=1243 Ack=3416 Win=64896 Len=0 TSval=3864196315 TSecr=3864196268
17	9.323084280	127.0.0.1	127.0.0.1	TCP	66	37990 → 8080 [FIN, ACK] Seq=1243 Ack=3416 Win=65536 Len=0 TSval=3864204510 TSecr=3864196268
18	9.323104883	127.0.0.1	127.0.0.1	TCP	66	8080 → 37990 [ACK] Seq=3416 Ack=1244 Win=65536 Len=0 TSval=3864204510 TSecr=3864204510

כמו שהסברנו כבר באריכות בחלק א' של התרגיל, שלוש החבילות הראשונות אחראיות ליצירת חיבור ה-tcp בין השרת ללקוח. שלוש החבילות האחרונות אחראיות לסיום החיבור בין השרת ללקוח, וגם על זה דיברנו כבר באריכות בחלק הראשון של התרגיל. אין עוד חיבורים או ניתוקים בכל התהליך ולכן יש רק חיבור אחד.
כשבקשנו מהדפדפן את הקובץ הרצוי ('/'), נשלחה לשרת בקשת GET בפרוטוקול HTTP. השרת החזיר הודעת 200 OK, ואיתה את תוכן הדף הרצוי. כמו כן, חשוב לציין שמיד אחר כך הלקוח גם ביקש את הקובץ favicon.ico - התמונה שנמצאת בראש הטאב בדפדפן. השרת החזיר הודעת 200 OK ואיתה את התמונה בבתים. מספירה, בחיבור זה נשלחו שתי בקשות.

בקשת 'a/2.jpg':

19	9.323260631	127.0.0.1	127.0.0.1	TCP	74	51520 → 8080 [SYN] Seq=0 Win=65495 Len=0 MSS=65495 SACK_PERM=1 TSval=3864204511 TSecr=0 WS=128
20	9.323271177	127.0.0.1	127.0.0.1	TCP	74	8080 → 51520 [SYN, ACK] Seq=0 Ack=1 Win=65483 Len=0 MSS=65495 SACK_PERM=1 TSval=3864204511 TSecr=3864204511 WS=128
21	9.323281095	127.0.0.1	127.0.0.1	TCP	66	51520 → 8080 [ACK] Seq=1 Ack=1 Win=65536 Len=0 TSval=3864204511 TSecr=3864204511
22	9.323432233	127.0.0.1	127.0.0.1	HTTP	731	GET /a/2.jpg HTTP/1.1
23	9.323442290	127.0.0.1	127.0.0.1	TCP	66	8080 → 51520 [ACK] Seq=1 Ack=666 Win=64896 Len=0 TSval=3864204511 TSecr=3864204511
24	9.323735766	127.0.0.1	127.0.0.1	TCP	133	8080 → 51520 [PSH, ACK] Seq=1 Ack=666 Win=65536 Len=67 TSval=3864204511 TSecr=3864204511 [TCP segment of a reassembled…
25	9.323751830	127.0.0.1	127.0.0.1	TCP	66	51520 → 8080 [ACK] Seq=666 Ack=68 Win=65536 Len=0 TSval=3864204511 TSecr=3864204511
26	9.323785913	127.0.0.1	127.0.0.1	TCP	32834	8080 → 51520 [ACK] Seq=68 Ack=666 Win=65536 Len=32768 TSval=3864204511 TSecr=3864204511 [TCP segment of a reassembled …
27	9.323797227	127.0.0.1	127.0.0.1	TCP	66	51520 → 8080 [ACK] Seq=666 Ack=32836 Win=48512 Len=0 TSval=3864204511 TSecr=3864204511
28	9.323814059	127.0.0.1	127.0.0.1	TCP	32834	8080 → 51520 [PSH, ACK] Seq=32836 Ack=666 Win=65536 Len=32768 TSval=3864204511 TSecr=3864204511 [TCP segment of a reas…
29	9.323825933	127.0.0.1	127.0.0.1	TCP	66	51520 → 8080 [ACK] Seq=666 Ack=65604 Win=15744 Len=0 TSval=3864204511 TSecr=3864204511
30	9.326478674	127.0.0.1	127.0.0.1	TCP	66	[TCP Window Update] 51520 → 8080 [ACK] Seq=666 Ack=65604 Win=65536 Len=0 TSval=3864204514 TSecr=3864204511
31	9.326506890	127.0.0.1	127.0.0.1	TCP	32834	8080 → 51520 [ACK] Seq=65604 Ack=666 Win=65536 Len=32768 TSval=3864204514 TSecr=3864204514 [TCP segment of a reassembl…
32	9.326518274	127.0.0.1	127.0.0.1	TCP	32834	[TCP Window Full] 8080 → 51520 [PSH, ACK] Seq=98372 Ack=666 Win=65536 Len=32768 TSval=3864204514 TSecr=3864204514 [TCP…
33	9.327036991	127.0.0.1	127.0.0.1	TCP	66	51520 → 8080 [ACK] Seq=666 Ack=131140 Win=65536 Len=0 TSval=3864204514 TSecr=3864204514
34	9.327060668	127.0.0.1	127.0.0.1	TCP	32834	8080 → 51520 [ACK] Seq=131140 Ack=666 Win=65536 Len=32768 TSval=3864204514 TSecr=3864204514 [TCP segment of a reassemb…
35	9.327072750	127.0.0.1	127.0.0.1	TCP	32834	[TCP Window Full] 8080 → 51520 [PSH, ACK] Seq=163908 Ack=666 Win=65536 Len=32768 TSval=3864204514 TSecr=3864204514 [TC…
36	9.327090770	127.0.0.1	127.0.0.1	TCP	66	51520 → 8080 [ACK] Seq=666 Ack=163908 Win=196480 Len=0 TSval=3864204514 TSecr=3864204514
37	9.327105157	127.0.0.1	127.0.0.1	HTTP	24208	HTTP/1.1 200 OK (JPEG JFIF image)
38	9.327117030	127.0.0.1	127.0.0.1	TCP	66	51520 → 8080 [ACK] Seq=666 Ack=196676 Win=327552 Len=0 TSval=3864204514 TSecr=3864204514
39	9.327126040	127.0.0.1	127.0.0.1	TCP	66	51520 → 8080 [ACK] Seq=666 Ack=220818 Win=458496 Len=0 TSval=3864204514 TSecr=3864204514
40	10.324952997	127.0.0.1	127.0.0.1	TCP	66	8080 → 51520 [FIN, ACK] Seq=220818 Ack=666 Win=65536 Len=0 TSval=3864205512 TSecr=3864204514
41	10.371414243	127.0.0.1	127.0.0.1	TCP	66	51520 → 8080 [ACK] Seq=666 Ack=220819 Win=458496 Len=0 TSval=3864205559 TSecr=3864205512
42	13.481559162	127.0.0.1	127.0.0.1	TCP	66	51520 → 8080 [FIN, ACK] Seq=666 Ack=220819 Win=458496 Len=0 TSval=3864208669 TSecr=3864205512
43	13.481585841	127.0.0.1	127.0.0.1	TCP	66	8080 → 51520 [ACK] Seq=220819 Ack=667 Win=65536 Len=0 TSval=3864208669 TSecr=3864208669

שוב, שלוש החבילות הראשונות אחראיות ליצירת חיבור ה-tcp בין השרת ללקוח, ושוב שלוש החבילות האחרונות אחראיות לסיום החיבור בין השרת ללקוח, וגם על זה דיברנו כבר באריכות בחלק הראשון של התרגיל. אין עוד חיבורים או ניתוקים בכל התהליך ולכן יש רק חיבור אחד. כשהזנו לדפדפן את הקובץ הרצוי a/2.jpg, נשלחה לשרת בקשת GET בפרוטוקול HTTP. השרת הפעם גם כן החזיר הודעת OK 200, ואיתה את תוכן הדף הרצוי (תוכן התמונה) בצורה בינארית. נעיר שהפעם הלקוח לא ביקש את התמונה favicon.ico, כי היא נשמרת ב-cache של הדפדפן כדי לחסוך בקשות מיותרות. מספירה, בחיבור זה נשלחה בקשה אחת.

בקשת (redirect):

44	13.481812339	127.0.0.1	127.0.0.1	TCP	74	51536 → 8080 [SYN] Seq=0 Win=65495 Len=0 MSS=65495 SACK_PERM=1 TSval=3864208669 TSecr=0 WS=128
45	13.481827844	127.0.0.1	127.0.0.1	TCP	74	8080 → 51536 [SYN, ACK] Seq=0 Ack=1 Win=65483 Len=0 MSS=65495 SACK_PERM=1 TSval=3864208669 TSecr=3864208669 WS=128
46	13.481841324	127.0.0.1	127.0.0.1	TCP	66	51536 → 8080 [ACK] Seq=1 Ack=1 Win=65536 Len=0 TSval=3864208669 TSecr=3864208669
47	13.482405578	127.0.0.1	127.0.0.1	HTTP	732	GET /redirect HTTP/1.1
48	13.482425692	127.0.0.1	127.0.0.1	TCP	66	8080 → 51536 [ACK] Seq=1 Ack=667 Win=64896 Len=0 TSval=3864208670 TSecr=3864208670
49	13.482546798	127.0.0.1	127.0.0.1	TCP	143	8080 → 51536 [PSH, ACK] Seq=1 Ack=667 Win=65536 Len=77 TSval=3864208670 TSecr=3864208670 [TCP segment of a reassembled…
50	13.482568310	127.0.0.1	127.0.0.1	HTTP	66	HTTP/1.1 301 Moved Permanently
51	13.482657009	127.0.0.1	127.0.0.1	TCP	66	51536 → 8080 [ACK] Seq=667 Ack=78 Win=65536 Len=0 TSval=3864208670 TSecr=3864208670
52	13.482850967	127.0.0.1	127.0.0.1	TCP	66	51536 → 8080 [FIN, ACK] Seq=667 Ack=79 Win=65536 Len=0 TSval=3864208670 TSecr=3864208670
53	13.482867443	127.0.0.1	127.0.0.1	TCP	66	8080 → 51536 [ACK] Seq=79 Ack=668 Win=65536 Len=0 TSval=3864208670 TSecr=3864208670
54	13.489755268	127.0.0.1	127.0.0.1	TCP	74	51548 → 8080 [SYN] Seq=0 Win=65495 Len=0 MSS=65495 SACK_PERM=1 TSval=3864208677 TSecr=0 WS=128
55	13.489770563	127.0.0.1	127.0.0.1	TCP	74	8080 → 51548 [SYN, ACK] Seq=0 Ack=1 Win=65483 Len=0 MSS=65495 SACK_PERM=1 TSval=3864208677 TSecr=3864208677 WS=128
56	13.489784881	127.0.0.1	127.0.0.1	TCP	66	51548 → 8080 [ACK] Seq=1 Ack=1 Win=65536 Len=0 TSval=3864208677 TSecr=3864208677
57	13.490055309	127.0.0.1	127.0.0.1	HTTP	735	GET /result.html HTTP/1.1
58	13.490064458	127.0.0.1	127.0.0.1	TCP	66	8080 → 51548 [ACK] Seq=1 Ack=670 Win=64896 Len=0 TSval=3864208677 TSecr=3864208677
59	13.490247095	127.0.0.1	127.0.0.1	TCP	130	8080 → 51548 [PSH, ACK] Seq=1 Ack=670 Win=65536 Len=64 TSval=3864208678 TSecr=3864208677 [TCP segment of a reassembled…
60	13.490257572	127.0.0.1	127.0.0.1	TCP	66	51548 → 8080 [ACK] Seq=670 Ack=65 Win=65536 Len=0 TSval=3864208678 TSecr=3864208678
61	13.490274543	127.0.0.1	127.0.0.1	HTTP	180	HTTP/1.1 200 OK
62	13.490279293	127.0.0.1	127.0.0.1	TCP	66	51548 → 8080 [ACK] Seq=670 Ack=179 Win=65536 Len=0 TSval=3864208678 TSecr=3864208678
63	14.498755290	127.0.0.1	127.0.0.1	TCP	66	8080 → 51548 [FIN, ACK] Seq=179 Ack=670 Win=65536 Len=0 TSval=3864209678 TSecr=3864208678
64	14.531433893	127.0.0.1	127.0.0.1	TCP	66	51548 → 8080 [ACK] Seq=670 Ack=180 Win=65536 Len=0 TSval=3864209719 TSecr=3864209678
65	18.733664276	127.0.0.1	127.0.0.1	TCP	66	51548 → 8080 [FIN, ACK] Seq=670 Ack=180 Win=65536 Len=0 TSval=3864213921 TSecr=3864209678
66	18.733683971	127.0.0.1	127.0.0.1	TCP	66	8080 → 51548 [ACK] Seq=180 Ack=671 Win=65536 Len=0 TSval=3864213921 TSecr=3864213921

בשונה מהפעמים הקודמות, כאשר אנחנו מבקשים redirect מתבצעים שני חיבורים לשרת ולא רק אחד. הסיבה לכך היא שכאשר אנחנו מחכים לתוצאת ה-redirect, מתבצע time out ולאחר שליחת הודעת ה-301 הלקוח מתחבר מחדש לשרת, ומבקש את הקובץ המפורט בהודעת ה-301. במקרה שלנו result.html.

```
client_sock.send(b'HTTP/1.1 301 Moved Permanently\r\nConnection: close\r\nLocation: /result.html\r\n\r\n')
```

גם הפעם, שלוש החבילות הראשונות אחראיות ליצירת חיבור ה-tcp בין השרת ללקוח. כשהזנו לדפדפן את הקובץ הרצוי redirect/, נשלחה לשרת בקשת GET בפרוטוקול HTTP, וכתגובה קיבלנו הודעת 301. לאחר הניתוק והחיבור מחדש, נשלחת עוד בקשת GET והפעם הודעת OK 200 עם התוכן של result.html. שלוש החבילות האחרונות אחראיות לסיום החיבור בין השרת ללקוח, וגם על זה דיברנו כבר באריכות בחלק הראשון של התרגיל.

נעיר כי באחת הפעמים שהרצנו את הקוד, ראינו ב-WireShark שהלקוח מבקש אוטומטית את result.html - מה הפתיע אותנו בהתחשב בעובדה ששלחנו redirect.html. לאחר מחקר קטן גלינו את הסיבה: chrome בעצם שמר את המידע בזיכרון ה-cache שלו מהרצה קודמת ולכן ידע כבר שאנחנו מבקשים בעצם את result.html. השרת מחזיר בבקשה השנייה הודעת OK 200, ואיתה את תוכן הדף הרצוי (תוכן התמונה) בצורה בינארית.

נוסיף שהפעם הלקוח לא ביקש את התמונה favicon.ico, כי היא נשמרת ב-cache של הדפדפן כדי לחסוך בקשות מיותרות. מספירה, כאן היו שני חיבורים ונשלחה בקשה אחת בכל חיבור.

בקשת 'a.html' (קובץ לא קיים):

67	18.733831408	127.0.0.1	127.0.0.1	TCP	74	37800 → 8080 [SYN] Seq=0 Win=65495 Len=0 MSS=65495 SACK_PERM=1 TSval=3864213921 TSecr=0 WS=128
68	18.733842722	127.0.0.1	127.0.0.1	TCP	74	8080 → 37800 [SYN, ACK] Seq=0 Ack=1 Win=65483 Len=0 MSS=65495 SACK_PERM=1 TSval=3864213921 TSecr=3864213921 WS=128
69	18.733851732	127.0.0.1	127.0.0.1	TCP	66	37800 → 8080 [ACK] Seq=1 Ack=1 Win=65536 Len=0 TSval=3864213921 TSecr=3864213921
70	18.734276930	127.0.0.1	127.0.0.1	HTTP	730	GET /a.html HTTP/1.1
71	18.734292156	127.0.0.1	127.0.0.1	TCP	66	8080 → 37800 [ACK] Seq=1 Ack=665 Win=64896 Len=0 TSval=3864213922 TSecr=3864213922
72	18.734370379	127.0.0.1	127.0.0.1	TCP	111	8080 → 37800 [PSH, ACK] Seq=1 Ack=665 Win=65536 Len=45 TSval=3864213922 TSecr=3864213922 [TCP segment of a reassembled…
73	18.734379528	127.0.0.1	127.0.0.1	TCP	66	37800 → 8080 [ACK] Seq=665 Ack=46 Win=65536 Len=0 TSval=3864213922 TSecr=3864213922
74	18.734397967	127.0.0.1	127.0.0.1	HTTP	66	HTTP/1.1 404 Not Found
75	18.735160153	127.0.0.1	127.0.0.1	TCP	66	37800 → 8080 [FIN, ACK] Seq=665 Ack=47 Win=65536 Len=0 TSval=3864213922 TSecr=3864213922
76	18.735176007	127.0.0.1	127.0.0.1	TCP	66	8080 → 37800 [ACK] Seq=47 Ack=666 Win=65536 Len=0 TSval=3864213922 TSecr=3864213922

כמו בכל הפעמים הקודמות, גם כאן שלוש החבילות הראשונות אחראיות ליצירת חיבור ה-tcp בין השרת ללקוח, ושלוש החבילות האחרונות אחראיות לסיום החיבור בין השרת ללקוח. אין עוד חיבורים או ניתוקים בכל התהליך ולכן יש רק חיבור אחד. לאחר ביסוס התקשורת, הלקוח מבקש מהשרת את הקובץ a.html. מאחר והקובץ לא קיים, השרת מחזיר בתגובה הודעת Not Found 404, וסוגר את התקשורת עם הלקוח. מספירה, בחיבור זה נשלחה בקשה אחת.