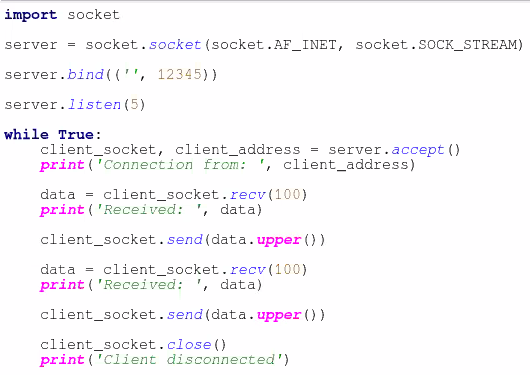
**דו"ח - אלעד ישראל 313448888 ונרקיס קרמיזי שלו 205832447**

חלק א

שינינו את קוד השרת ואת קוד הלקוח כפי שנדרש בתרגיל.

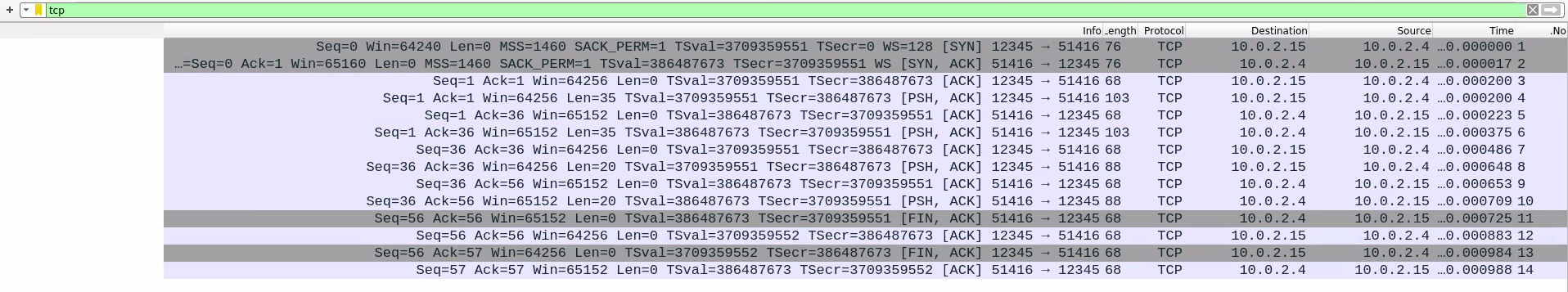
להלן קוד השרת:



להלן קוד הלקוח:



הסבר התעבורה בwireshark:

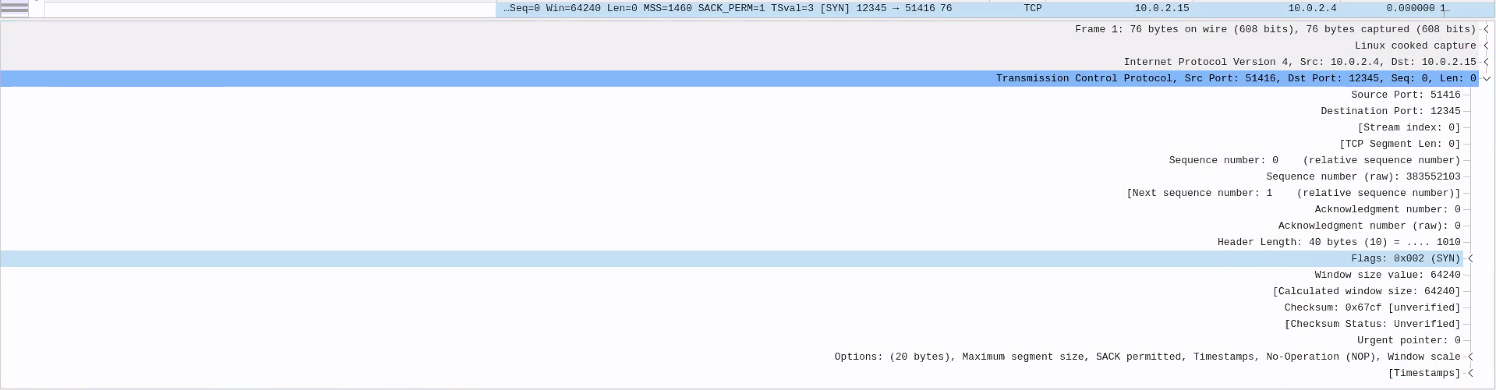


סיננו את החבילות כך שנראה רק חבילות שנשלחו בפרוטוקול TCP.

נשים לב שהIP של השרת הוא 10.0.2.15 והport שלו הוא 12345. כמו כן, נשים לב שהIP של הלקוח הוא 10.0.2.4 והport שלו הוא 51416.

ראשית, מתבצע חיבור בין השרת ללקוח בשיטת "3-way handshake".

**בחבילה 1** הלקוח שולח בקשת הסתנכרנות לשרת -

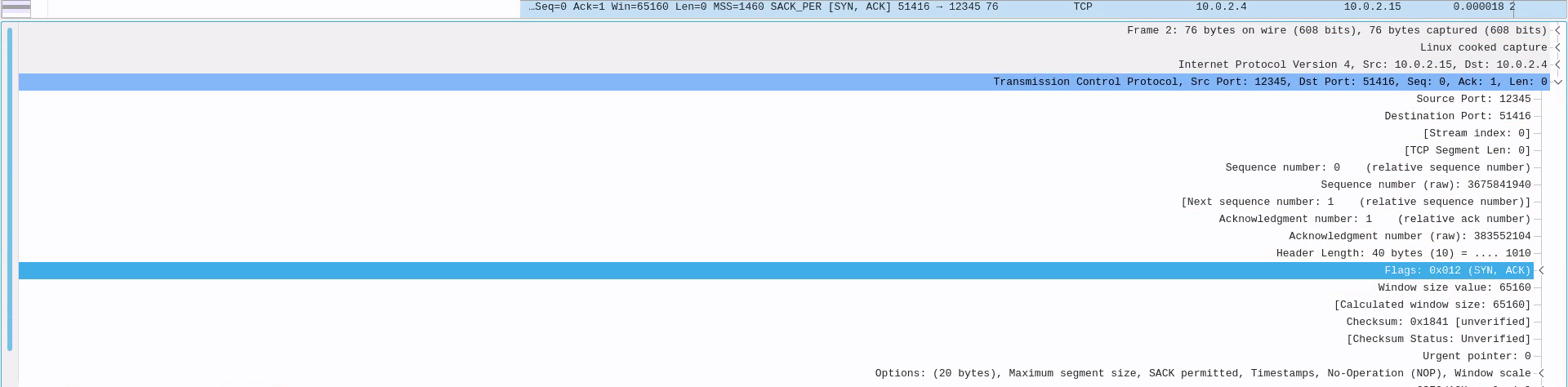


seq = 383552103 -->

אפשר לראות שבחבילה זו אין שום data אלא מדובר בheader TCP של שכבת התעבורה בלבד. כמו כן, ניתן לראות שהדגל SYN בheader זה דלוק, כי הלקוח רוצה שהשרת יסתנכרן מולו.

הלקוח בוחר באקראי את המספר 383552103 כמספר סידורי ומעדכן את השרת שהוא מתחיל לשלוח ממספר סידורי זה.

**בחבילה 2** השרת מאשר את בקשת הSYN של הלקוח ושולח לו בקשת הסתנכרנות משלו -



<-- seq = 3675841940, ack = 383552104

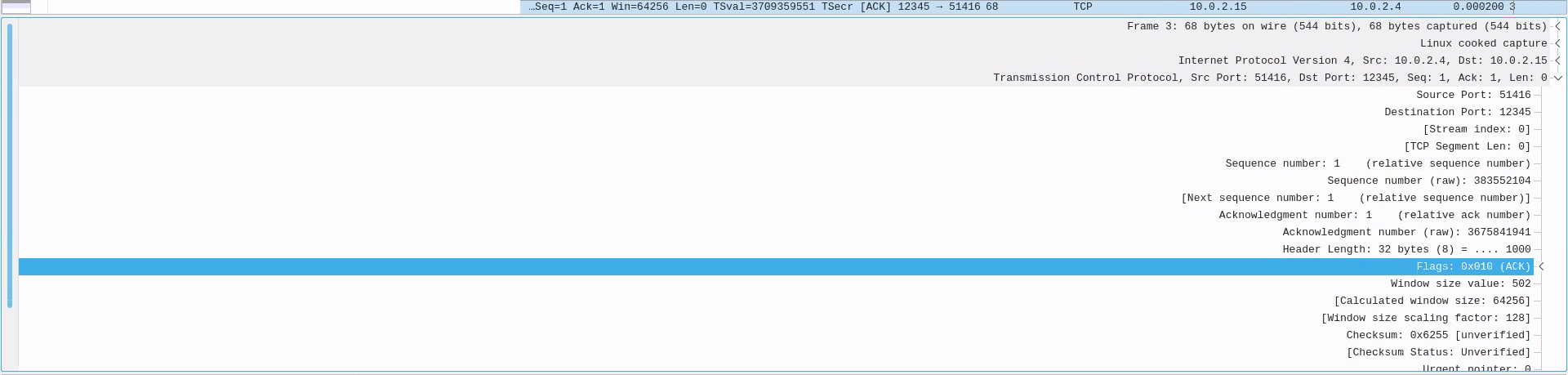
שוב אפשר לראות שבחבילה זו אין שום data אלא מדובר בheader TCP של שכבת התעבורה בלבד. כמו כן, ניתן לראות ששני דגלים דולקים -

הראשון הוא דגל הACK כי השרת רוצה לעדכן את הלקוח שקיבל את בקשת הSYN שלו. השרת מחזיר ACKnum ששווה 383552103+1 (383552104) כדי לעדכן את הלקוח שהוא אכן קיבל את 383552103 ממנו ושהוא מצפה לקבל החל מ383552104.

השני הוא דגל הSYN כי השרת רוצה שהלקוח יסתנכרן גם כן מולו.

השרת בוחר באקראי את המספר 3675841940 כמספר סידורי ומעדכן את הלקוח שהוא מתחיל לשלוח ממספר סידורי זה.

**בחבילה 3** הלקוח מאשר את בקשת הSYN של השרת -



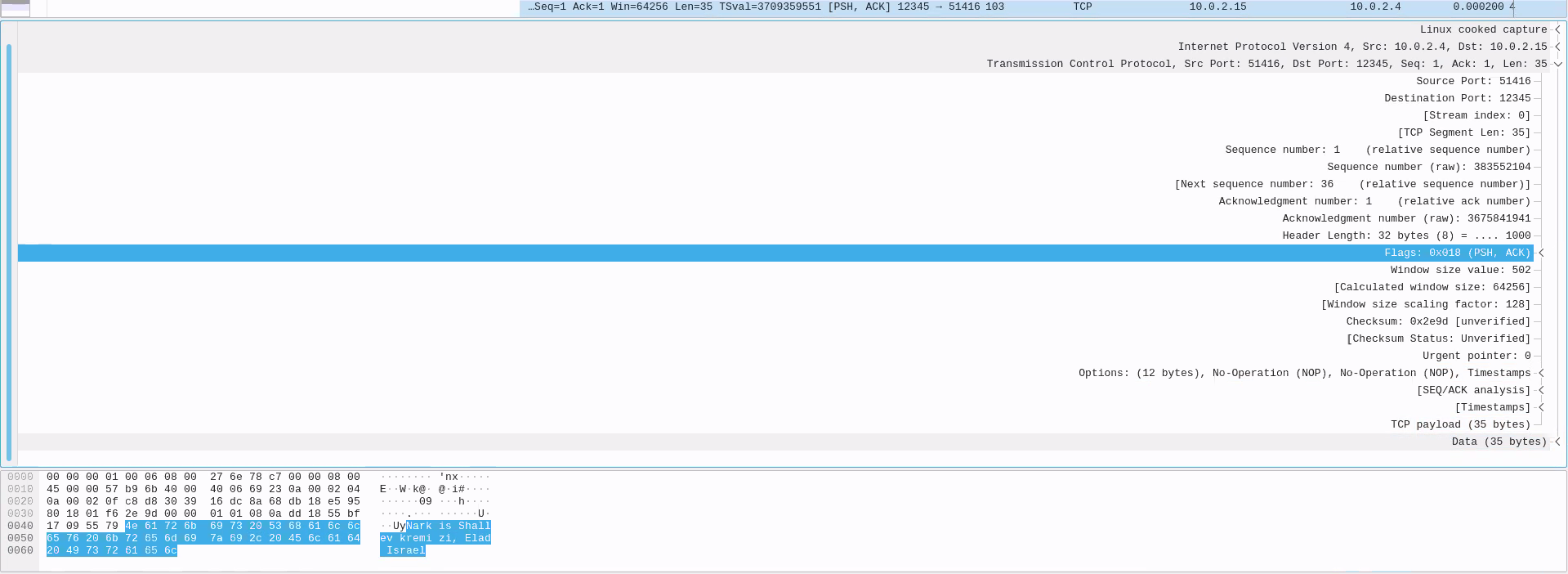
seq = 383552104, ack = 3675841941 -->

שוב אפשר לראות שבחבילה זו אין שום data אלא מדובר בheader TCP של שכבת התעבורה בלבד. כמו כן, ניתן לראות שהדגל ACK בheader זה דלוק כי הלקוח רוצה לעדכן את השרת שקיבל את בקשת הSYN שלו. הלקוח מחזיר ACKnum ששווה ל3675841940+1 (3675841941) כדי לעדכן את השרת שהוא אכן קיבל 3675841940 ממנו ושהוא מצפה לקבל החל מ3675841941.

מעכשיו הם יכולים להתחיל לדבר וממש לשלוח הודעות אחד לשני.

נשים לב שהמספר הסידורי של הלקוח (המספר ממנו הוא מתחיל לשלוח) הוא 383552104 שזה תואם את הACKnum שחזר מהשרת (המספר ממנו השרת מצפה לקבל).

**בחבילה 4** הלקוח שולח לשרת חבילה עם הבקשה "Narkis Shallev Kremizi, Elad Israel" -



seq = 383552104, ack = 3675841941, data = 35 -->

הפעם אפשר לראות בשכבת האפליקציה שיש data בגודל 35 בתים (כנראה שבחבילה הקודמת השרת טרם הספיק לייצר data ורק רצה לעדכן שקיבל את הבקשה). כמו כן, בheader TCP של שכבת התעבורה ניתן לראות ששני דגלים דולקים -

הראשון הוא דגל הPSH שלא הרחבנו עליו בשיעור.

השני הוא דגל הACK כי בפרוטוקול TCP בכל חבילה מלבד החבילה הראשונה של הSYN מעדכנים את הצד השני בACKnum המצטבר, הACKnum הכי עדכני. במקרה שלנו ניתן לראות שערכו של ACKnum זה הוא 3675841941 כי הלקוח לא קיבל שום דבר מאז חבילה 3.

נשים לב שהמספר הסידורי גם כן לא השתנה כי הלקוח לא שלח שום דבר לשרת.

**בחבילה 5** השרת מאשר ללקוח שקיבל את החבילה -

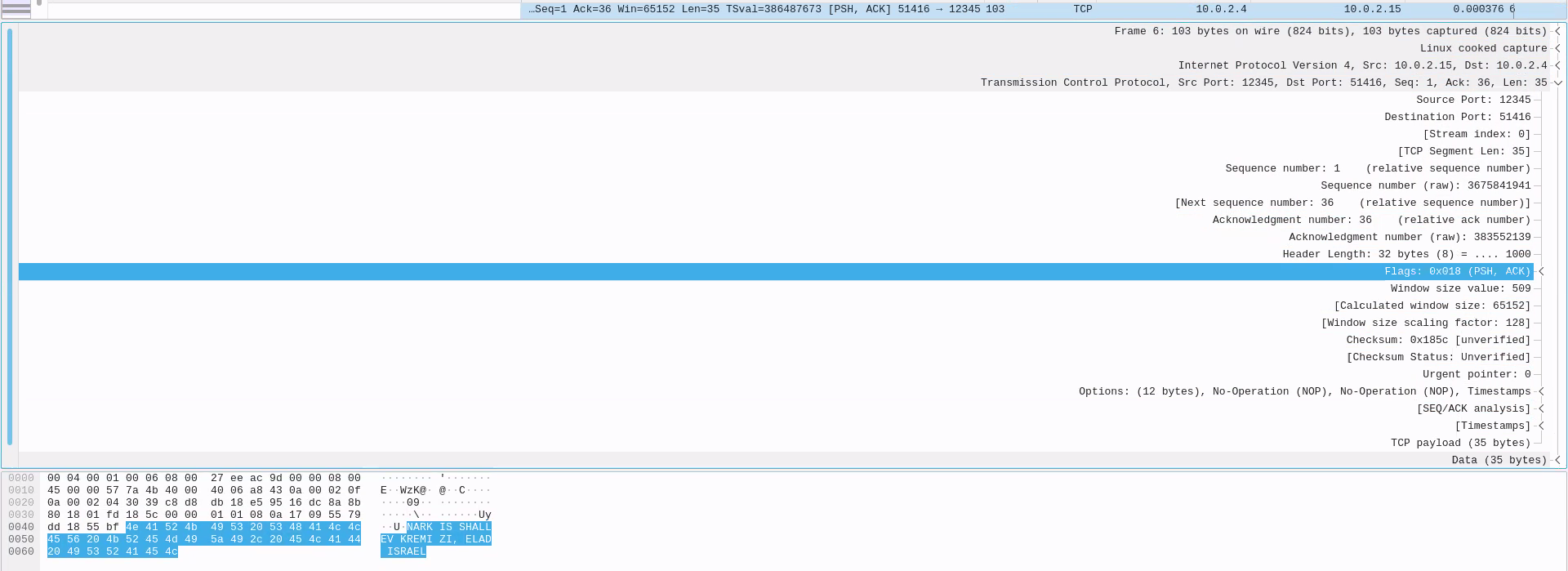


<-- seq = 3675841941, ack = 383552139

אפשר לראות שבחבילה זו אין שום data אלא מדובר בheader TCP של שכבת התעבורה בלבד. כמו כן, ניתן לראות שהדגל ACK בheader זה דלוק כי השרת רוצה לעדכן את הלקוח שקיבל את הבקשה שלו. השרת מחזיר ACKnum ששווה 383552104+35 (383552139) כדי לעדכן את הלקוח שהוא אכן קיבל ממנו מידע בגודל 35 בתים החל מ383552104 ושהוא מצפה לקבל מעכשיו החל מ383552139.

נשים לב שהמספר הסידורי של השרת (המספר ממנו הוא מתחיל לשלוח) הוא 3675841941 שזה תואם את הACKnum שחזר מהלקוח (המספר ממנו הלקוח מצפה לקבל).

**בחבילה 6** השרת שולח ללקוח חבילה עם התשובה "NARKIS SHALLEV KREMIZI, ELAD ISRAEL" -



<-- seq = 3675841941, ack = 383552139, data = 35

הפעם אפשר לראות בשכבת האפליקציה שיש data בגודל 35 בתים (כנראה שבחבילה הקודמת השרת טרם הספיק לייצר data ורק רצה לעדכן שקיבל את הבקשה). כמו כן, בheader TCP של שכבת התעבורה ניתן לראות ששני דגלים דולקים -

הראשון הוא דגל הPSH שלא הרחבנו עליו בשיעור.

השני הוא דגל הACK כי בפרוטוקול TCP בכל חבילה מלבד החבילה הראשונה של הSYN מעדכנים את הצד השני בACKnum המצטבר, הACKnum הכי עדכני. במקרה שלנו ניתן לראות שערכו של ACKnum זה הוא 383552139 כי השרת לא קיבל שום דבר מאז חבילה 5.

נשים לב שהמספר הסידורי גם כן לא השתנה כי השרת לא שלח שום דבר ללקוח.

**בחבילה 7** הלקוח מאשר לשרת שקיבל את החבילה -

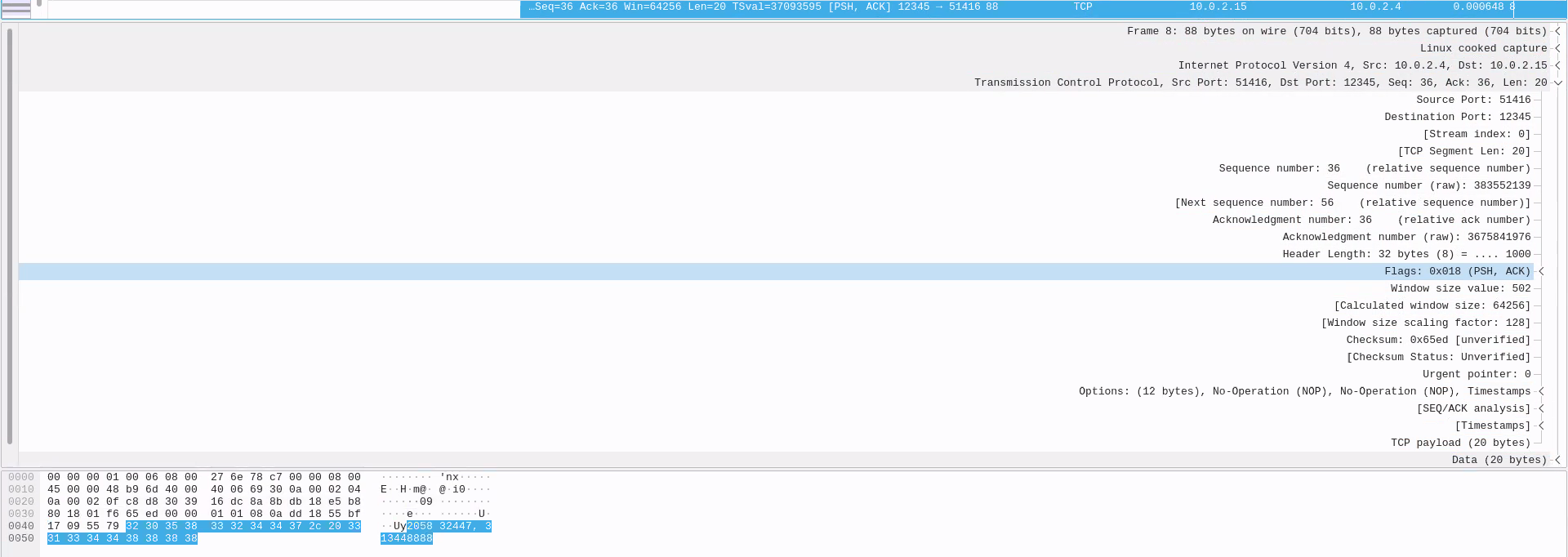


seq = 383552139, ack = 3675841976 -->

אפשר לראות שבחבילה זו אין שום data אלא מדובר בheader TCP של שכבת התעבורה בלבד. כמו כן, ניתן לראות שהדגל ACK בheader זה דלוק כי הלקוח רוצה לעדכן את השרת שקיבל את התשובה שלו. הלקוח מחזיר ACKnum ששווה 3675841941+35 (3675841976) כדי לעדכן את השרת שהוא אכן קיבל ממנו מידע בגודל 35 בתים החל מ3675841941ושהוא מצפה לקבל מעכשיו החל מ3675841976.

נשים לב שהמספר הסידורי השתנה כי כל עוד הלקוח לא קיבל timeout על חבילה 5 (ACK) ששלח לו השרת הוא מניח שהכל בסדר ושולח כבר את החבילה הבאה. הלקוח שלח לשרת בחבילה 4 מידע בגודל 35 בתים החל מ383552104 ולכן החבילה הבאה אמורה להישלח החל מ383552104+35 (383552139) בדיוק כפי שכתוב במספר הסידורי של החבילה הנוכחית.

**בחבילה 8** הלקוח שולח לשרת חבילה עם הבקשה "313448888, 205832447" -



seq = 383552139, ack = 3675841976, data = 20 -->

ההסבר כאן דומה להסבר על חבילה 4 (רק המספרים שונים).

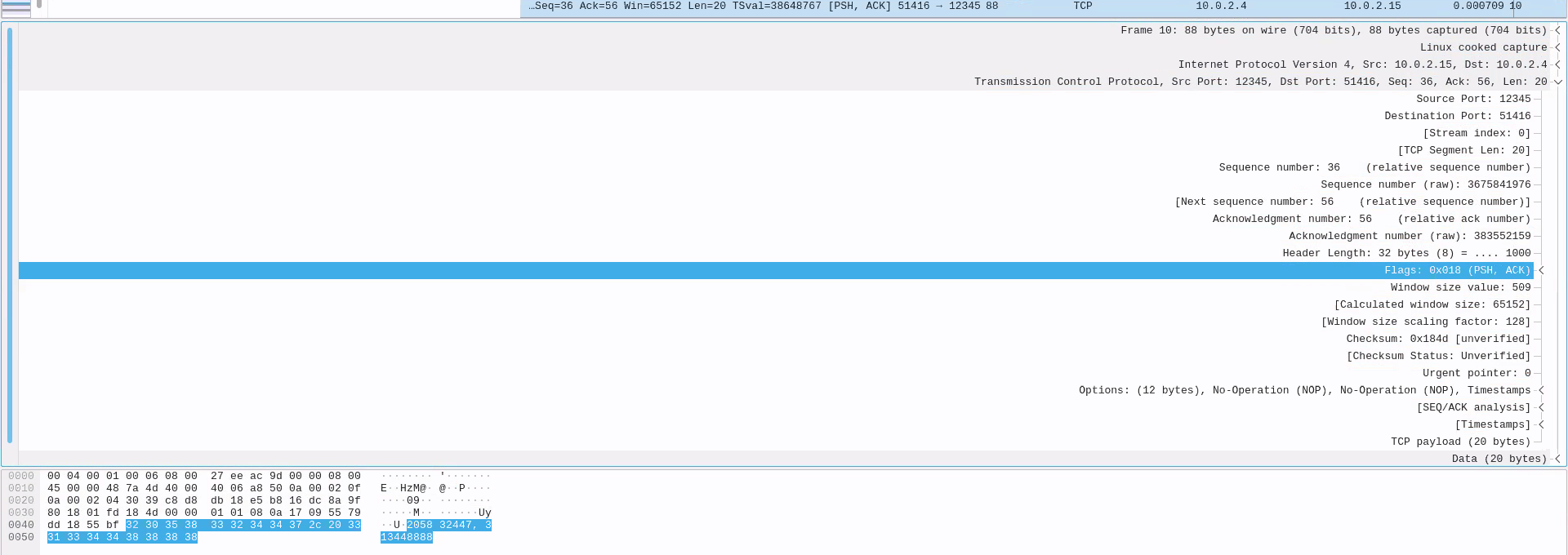
**בחבילה 9** השרת מאשר ללקוח שקיבל את החבילה -



<-- seq = 3675841976, ack = 383552159

ההסבר כאן דומה להסבר על חבילה 5 (רק המספרים שונים).

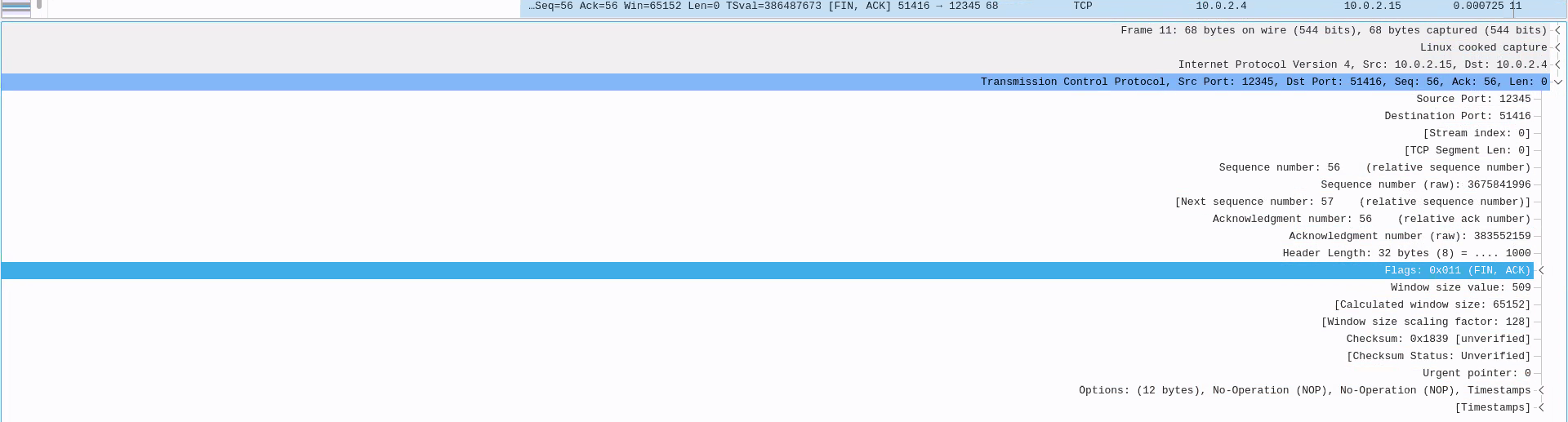
**בחבילה 10** השרת שולח ללקוח חבילה עם התשובה "313448888, 205832447" -



<-- seq = 3675841976, ack = 383552159, data = 20

ההסבר כאן דומה להסבר על חבילה 6 (רק המספרים שונים).

**בחבילה 11** השרת שולח בקשת התנתקות ללקוח -



<-- seq = 3675841996, ack = 383552159

אפשר לראות שבחבילה זו אין שום data אלא מדובר בheader TCP של שכבת התעבורה בלבד. כמו כן, ניתן לראות ששני דגלים דולקים -

הראשון הוא דגל הFIN כי השרת רוצה לסגור את החיבור.

השני הוא דגל הACK כי בפרוטוקול TCP בכל חבילה מלבד החבילה הראשונה של הSYN מעדכנים את הצד השני בACKnum המצטבר, הACKnum הכי עדכני. במקרה שלנו ניתן לראות שערכו של ACKnum זה הוא 383552159 כי השרת לא קיבל שום דבר מאז חבילה 9.

נשים לב שהמספר הסידורי **כן השתנה** למרות שהשרת לא קיבל ACK על החבילה הקודמת ששלח ללקוח (כנראה הACK מתעכב או משהו אבל כל עוד השרת לא קיבל timeout על החבילה הוא מניח שהכל בסדר ושולח כבר את החבילה הבאה). השרת שלח ללקוח מידע בגודל 20 בתים החל מ3675841976 ולכן החבילה הבאה אמורה להישלח החל מ3675841976+20 (3675841996) בדיוק כפי שכתוב במספר הסידורי של החבילה הנוכחית.

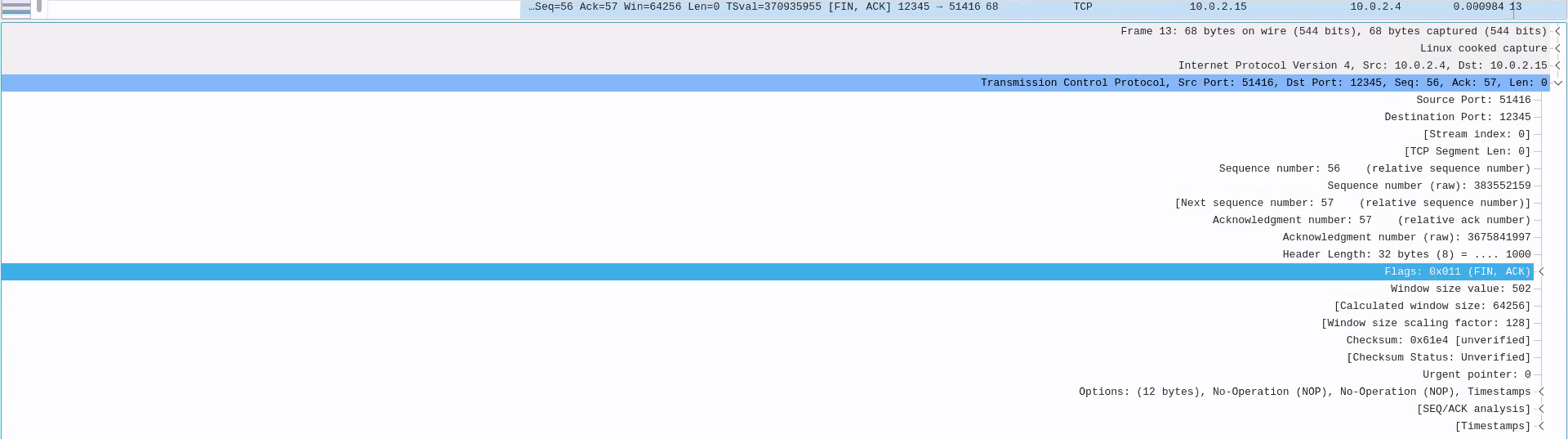
**בחבילה 12** הלקוח מאשר לשרת שקיבל את החבילה (10) -



seq = 383552159, ack = 3675841996 -->

ההסבר כאן דומה להסבר על חבילה 7 (רק המספרים שונים).

**בחבילה 13** הלקוח מאשר את בקשת הFIN של השרת ושולח לו בקשת התנתקות משלו -



seq = 383552159, ack = 3675841997 -->

שוב אפשר לראות שבחבילה זו אין שום data אלא מדובר בheader TCP של שכבת התעבורה בלבד. כמו כן, ניתן לראות ששני דגלים דולקים -

הראשון הוא דגל הACK כי הלקוח רוצה לעדכן את השרת שקיבל את בקשת הFIN שלו. הלקוח מחזיר ACKnum ששווה 3675841996+1 (3675841997) כדי לעדכן את השרת שהוא אכן קיבל את בקשת הFIN שלו ושהוא מצפה לקבל החל מ 3675841997.

השני הוא דגל הFIN כי הלקוח רוצה לסגור את החיבור גם כן.

נשים לב שהמספר הסידורי לא השתנה כי הלקוח לא שלח שום דבר לשרת.

**בחבילה 14** השרת מאשר את בקשת הFIN של הלקוח -



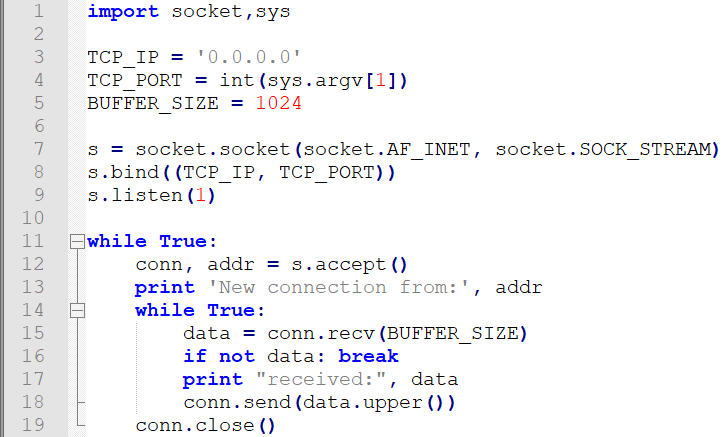
<-- seq = 3675841997, ack = 383552160

שוב אפשר לראות שבחבילה זו אין שום data אלא מדובר בheader TCP של שכבת התעבורה בלבד. כמו כן, ניתן לראות שהדגל ACK בheader זה דלוק כי השרת רוצה לעדכן את הלקוח שקיבל את בקשת הFIN שלו. השרת מחזיר ACKnum ששווה ל383552159+1 (383552160) כדי לעדכן את השרת שהוא אכן קיבל את בקשת הFIN שלו.

נשים לב שהמספר הסידורי של השרת (המספר ממנו הוא מתחיל לשלוח) הוא 3675841997 שזה תואם את הACKnum שחזר מהלקוח (המספר ממנו הלקוח מצפה לקבל).

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

קוד השרת V1:



הסבר מילולי -

השרת יוצר socket שמוגדר לפי פרוטוקול TCP וכתובות מסוג IPv4. לאחר מכן, השרת מבצע bind (קשירה) בין הsocket שלו לIP=0.0.0.0 וport כלשהו שמתקבל כארגומנט. נשים לב שקשירה לIP=0.0.0.0 אומרת שהשרת יוכל לקבל הודעות מכל כתובת IP שנשלחת למכונה. השרת מתחיל להאזין ללקוחות ומאפשר ללקוח אחד בלבד לחכות בתור בכל רגע נתון.

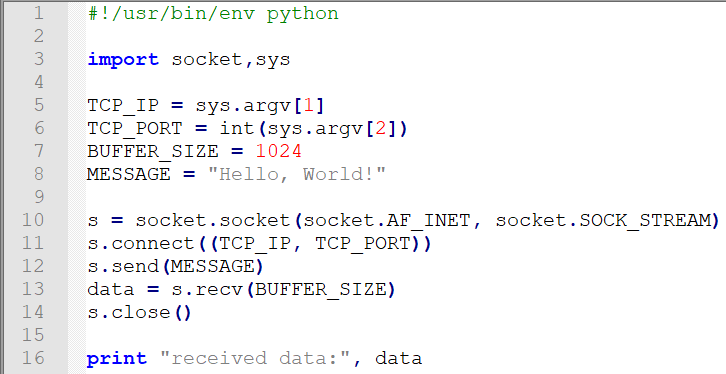
בלולאה, השרת מקבל בקשת התקשרות של לקוח ונוצר חיבור ייעודי לצורך ההתקשרות ביניהם. השרת מדפיס "New connection from:" + הIP והport של הלקוח שהתחבר אליו.

בלולאה הפנימית, השרת מקבל בקשה מהלקוח שגודלה לכל היותר 1024 בתים (גודל הבאפר) ומדפיס "received" + הבקשה. לאחר מכן, הוא שולח את אותה בקשה באותיות גדולות אל הלקוח. השרת ממשיך לקבל עוד ועוד בקשות מהלקוח ולטפל בהן באותה צורה עד שהבקשה המתקבלת ריקה. במקרה כזה, הוא יוצא מהלולאה, מסיים את ההתקשרות עם הלקוח הספציפי ועובר ללקוח הבא.

פלט השרת -



קוד הלקוח V1:



הסבר מילולי -

הלקוח פותח socket שמוגדר לפי פרוטוקול TCP וכתובות מסוג IPv4.

לאחר מכן הוא מתחבר לכתובת הIP והport של השרת שהוא קיבל כארגומנט, ושולח את ההודעה "Hello, World!".

אח"כ הוא מקבל את התשובה שהשרת שולח בחזרה לתוך הבאפר שהוגדר בהתחלה (1024 בתים).

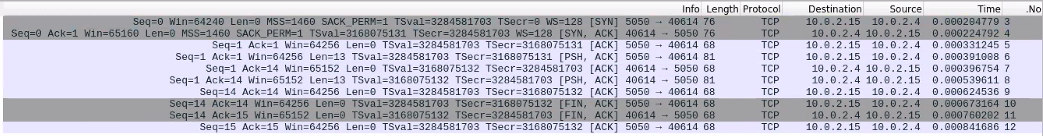
לבסוף, הוא סוגר את הsocket ומדפיס את הdata שהתקבל.

פלט הלקוח -



הסבר התעבורה בwireshark -

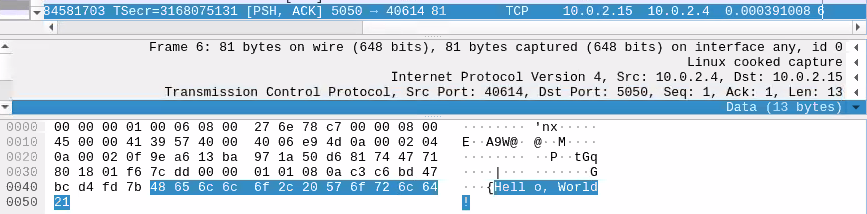
נשים לב, הIP של הלקוח הוא 10.0.2.4, ושל השרת 10.0.2.15.



בחבילה 3 הלקוח שולח בקשת הסתנכרנות לשרת.

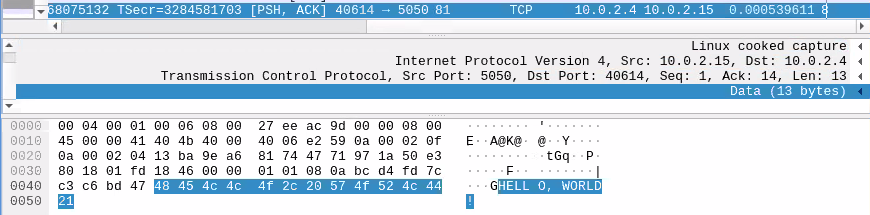
בחבילה 4 השרת מאשר את בקשת הSYN של הלקוח ושולח לו בקשת הסתנכרנות משלו.

בחבילה 5 הלקוח מאשר את בקשת הSYN של השרת.

בחבילה 6 הלקוח שולח לשרת חבילה עם הבקשה "Hello, World!" - 

בחבילה 7 השרת מאשר ללקוח שקיבל את החבילה.

בחבילה 8 השרת שולח ללקוח חבילה עם התשובה "HELLO, WORLD!" -



בחבילה 9הלקוח מאשר לשרת שקיבל את החבילה.

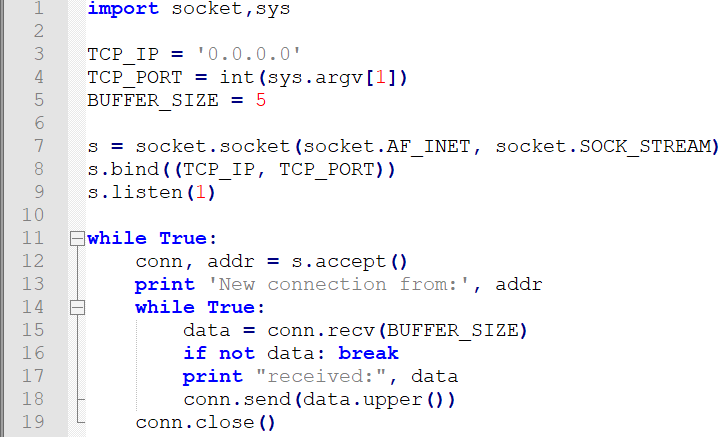
בחבילה 10 הלקוח שולח בקשת התנתקות לשרת.

בחבילה 11 השרת מאשר את בקשת הFIN של הלקוח ושולח לו בקשת התנתקות משלו.

בחבילה 12 הלקוח מאשר את בקשת הFIN של השרת.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

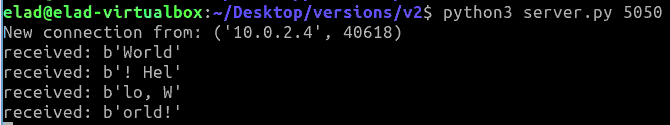
קוד השרת V2:



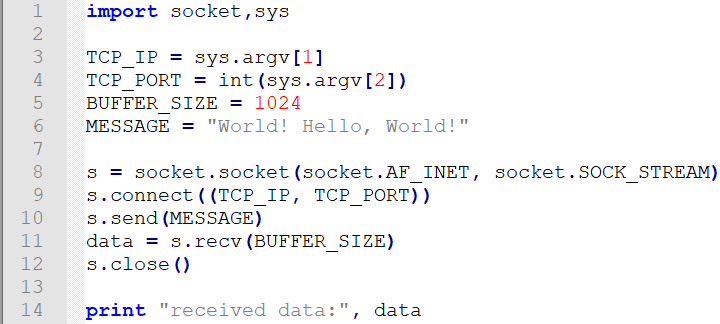
הסבר מילולי -

השרת זהה לשרת V1 מלבד זה שבלולאה הפנימית הוא מקבל בקשה מהלקוח שגודלה לכל היותר 5 בתים ולא 1024 בתים.

פלט השרת -



קוד הלקוח V2:



הסבר מילולי -

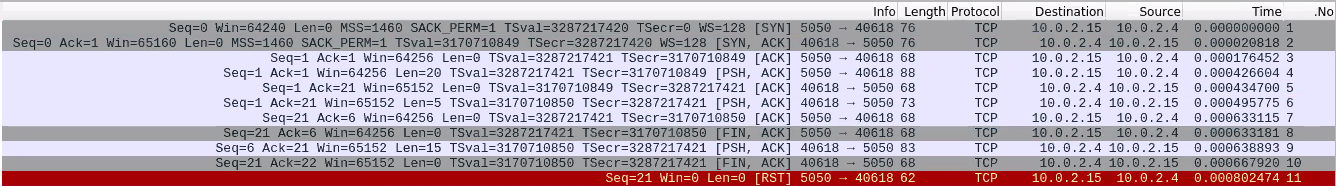
ההבדל היחיד בין הקליינט של V1 לV2 הוא שההודעה הנשלחת בV2 יותר ארוכה- "World! Hello, World!".

פלט הלקוח -



הסבר התעבורה בwireshark -

נשים לב, הIP של הלקוח הוא 10.0.2.4, ושל השרת 10.0.2.15.

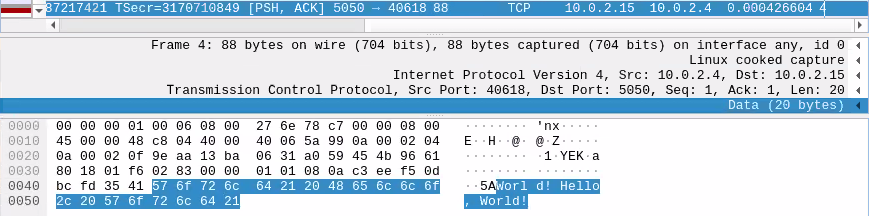


בחבילה 1 הלקוח שולח בקשת הסתנכרנות לשרת.

בחבילה 2 השרת מאשר את בקשת הSYN של הלקוח ושולח לו בקשת הסתנכרנות משלו.

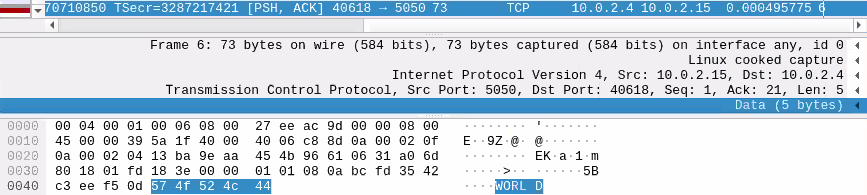
בחבילה 3 הלקוח מאשר את בקשת הSYN של השרת.

בחבילה 4 הלקוח שולח לשרת חבילה עם הבקשה "World! Hello, World!" -



בחבילה 5 השרת מאשר ללקוח שקיבל את החבילה.

בחבילה 6 השרת שולח ללקוח חבילה עם התשובה "WORLD" -



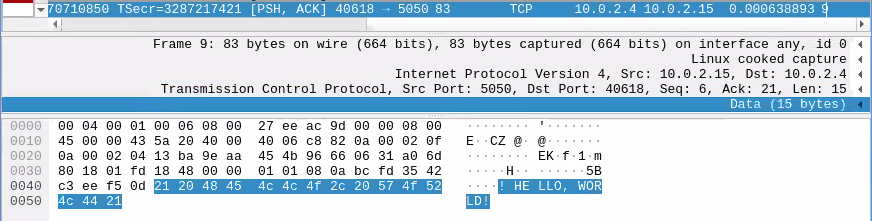
אומנם הלקוח שולח בחבילה 4 את כל 15 הבתים אך שכבת האפליקציה של השרת קוראת מהבאפר של שכבת התעבורה של השרת בכל פעם רק 5 בתים ומיד מטפלת בהם ושולחת אותם. לכן, רק WORLD (חמשת הבתים הראשונים) נשלח בחזרה.

בנוסף, בזמן שהשרת מחכה לACK מהלקוח (כלומר מחכה לחבילה 7)- בינתיים שכבת האפליקציה של השרת ממשיכה לרוץ על לולאת הwhile הפנימית ולעשות recv מהבאפר של שכבת התעבורה ומיד לאחר מכן send. כתוצאה מכך כששכבת התעבורה של השרת שולחת את המידע הבא (חבילה 9) - כל המידע משכבת האפליקציה כבר נשלח ביחד (כי לכולו נעשה send עד ששכבת התעבורה המשיכה לשלוח).

בחבילה 7הלקוח מאשר לשרת שקיבל את החבילה.

בחבילה 8 הלקוח שולח בקשת התנתקות לשרת.

בחבילה 9 השרת שולח ללקוח חבילה עם התשובה "! HELLO, WORLD!" (כלומר כל המידע שנותר לשליחה) -

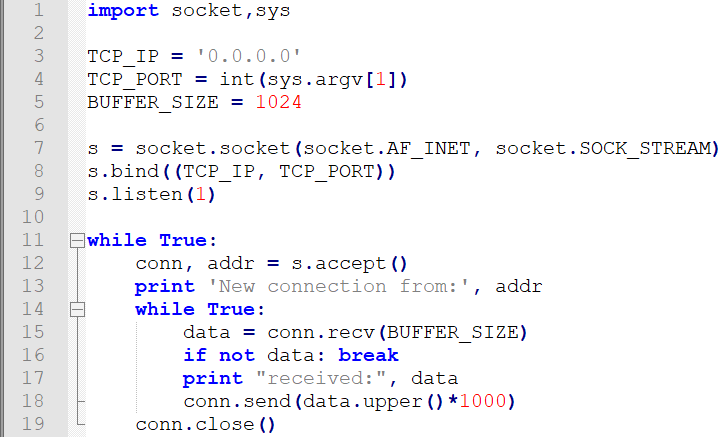


בחבילה 10 השרת מאשר את בקשת הFIN של הלקוח ושולח לו בקשת התנתקות משלו.

בחבילה 11 נשלח RST מהלקוח לשרת מכיוון שהגיעה אל הלקוח חבילה שהוא לא ציפה לה- חבילה 9, חבילה עם מידע שנשלחה מהשרת לאחר שהקליינט כבר ביקש מבחינתו מהשרת לסגור את החיבור(מכיוון שהמידע שהוחזר חזר ב2 'פעימות' בגלל הקריאה האיטית עקב הבאפר המצומצם של השרת). RST זה קיצור של reset- חבילה זו אומרת למקבל (השרת) שעליו להפסיק להשתמש בחיבור הזה באופן מיידי. זו בעצם סגירה מיידית של החיבור ע"י צד אחד (ללא נוהל לחיצת ידיים).

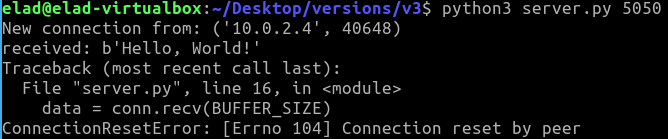
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

קוד השרת V3:

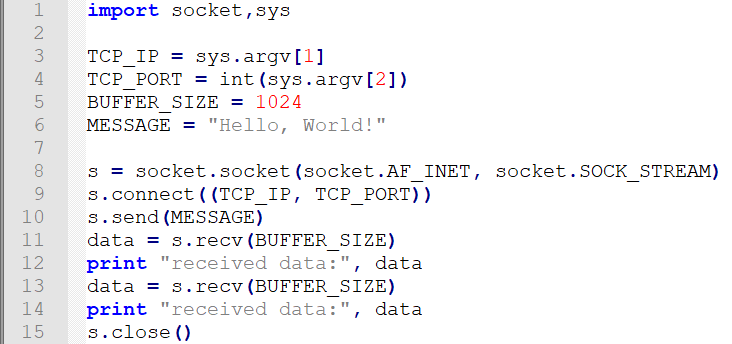


הסבר מילולי -

השרת זהה לשרת V1 מלבד זה שבלולאה הפנימית הוא שולח את אותה בקשה שקיבל מהלקוח לא רק באותיות גדולות אלא גם משורשרת 1000 פעמים.

פלט השרת - מתחבר לקליינט, מדפיס את הבקשה שהוא קיבל מהקליינט, ולאחר מכן מדפיס שגיאה שאומרת שהקליינט שלח RST בזמן שהשרת המתין וניסה לקבל ממנו מידע (עבור הבקשה הבאה- מידע עם תוכן או מידע ריק לסיום החיבור). 

קוד הלקוח V3:



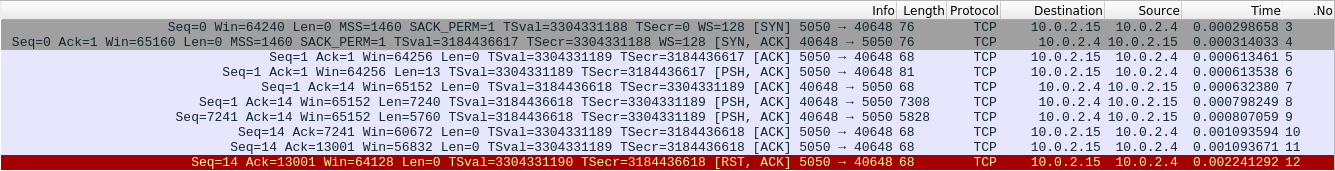
הסבר מילולי -

הקליינט זהה לV1 חוץ מקבלת התשובה מהשרת. בגרסא זו הקבלה מתבצעת פעמיים. כלומר הקליינט מקבל פעם אחת עד 1024 תווים לתוך הבאפר, ולאחר מכן מקבל שוב עד 1024 תווים (מדפיס ואז דורס את מה שהיה בבאפר).

פלט הלקוח - קריאה ופלט של 1024 בתים(כגודל הבאפר שלו) ואז שוב קריאה ופלט של 1024 הבתים הבאים.

הסבר התעבורה בwireshark -

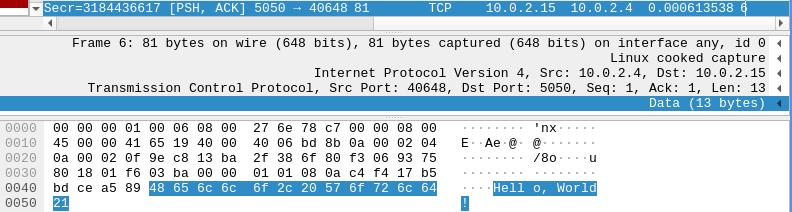
נשים לב, הIP של הלקוח הוא 10.0.2.4, ושל השרת 10.0.2.15.



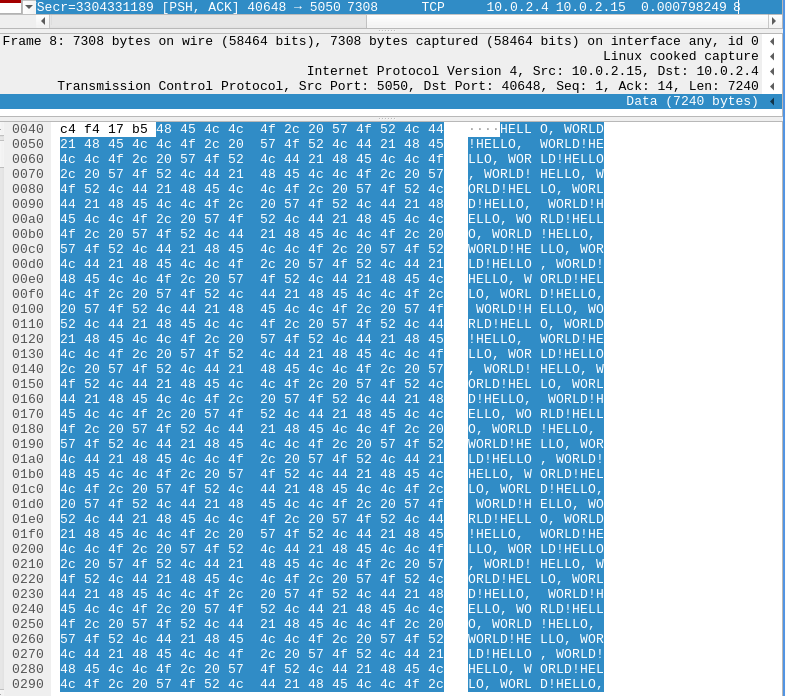
בחבילה 3 הלקוח שולח בקשת הסתנכרנות לשרת.

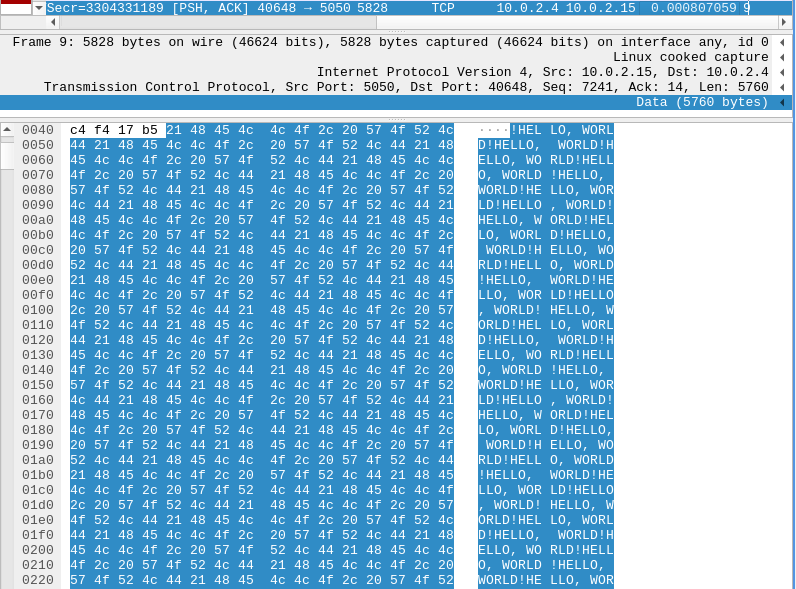
בחבילה 4 השרת מאשר את בקשת הSYN של הלקוח ושולח לו בקשת הסתנכרנות משלו.

בחבילה 5 הלקוח מאשר את בקשת הSYN של השרת.

בחבילה 6 הלקוח שולח לשרת חבילה עם הבקשה "Hello, World!" - 

בחבילה 7 השרת מאשר ללקוח שקיבל את החבילה.

בחבילה 8 ו-9 השרת שולח ללקוח חבילה עם התשובה "HELLO, WORLD!" משורשרת 1000 פעמים(13 בתים\*1000=13000 בתים בסה"כ).   
התשובה נשלחת בתור 2 חבילות. כל אחת מהן גדולה מהMSS.   
איך זה יכול להיות? מכיוון שמערכת ההפעלה והכרטיס תומכים בoffloading. במקרה הזה TCP עושה offload ומוריד מעצמו את האחריות לזה- במקום שהסגמנטציה תיעשה ע"י מערכת ההפעלה(על המעבד של המחשב) היא מתבצעת ע"י כרטיס הרשת עצמו(על המעבד שלו). מה שמוצג בwireshark זה לפני שכרטיס הרשת עשה את הסגמנטציה, כלומר אנחנו רואים את החבילה לפני שהיא מועברת לחלוקה בכרטיס ולכן היא גדולה מהMSS, אבל בפועל זה יחולק(אבל זה יקרה אחרי ההסנפה של wireshark).  
למה 2 חבילות ולא 1? מכיוון שהגודל המקסימלי של חבילה אחת שהוא יודע לחלק בעצמו הוא 7240 בתים.  




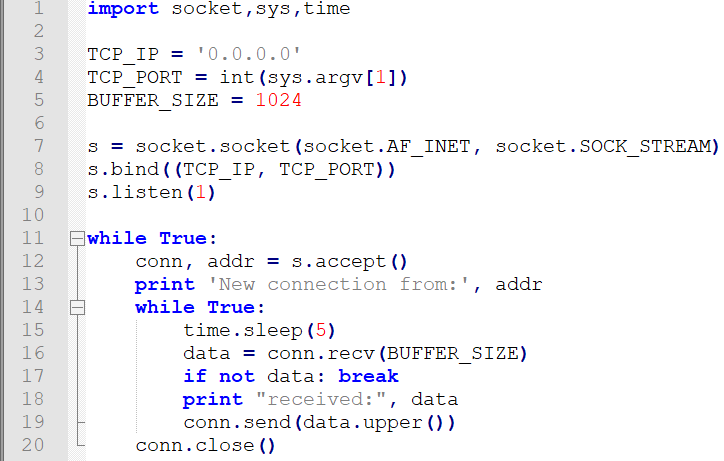
בחבילה 10הלקוח מאשר לשרת שקיבל את חבילה 8.

בחבילה 11הלקוח מאשר לשרת שקיבל את חבילה 9.

בחבילה 12 נשלח RST מהלקוח לשרת מכיוון שהלקוח מנסה לסגור את החיבור למרות שיש בשכבת התעבורה שלו הרבה מידע שמחכה להתקבל אצלו (מה שנשאר מהמידע של חבילה 8 והמידע של חבילה 9). RST זה קיצור של reset- חבילה זו אומרת למקבל (השרת) שעליו להפסיק להשתמש בחיבור הזה באופן מיידי. זו בעצם סגירה מיידית של החיבור ע"י צד אחד (ללא נוהל לחיצת ידיים). כלומר הקליינט "טרק את הטלפון בפנים" של השרת במקום פשוט להתעלם מזה שהשרת שחל לו עוד מידע שהוא לא קרא. זה נחשב יותר מנומס מאשר להתעלם ולהשאיר את השרת "באוויר" מבלי שידע מה קרה עם הלקוח.

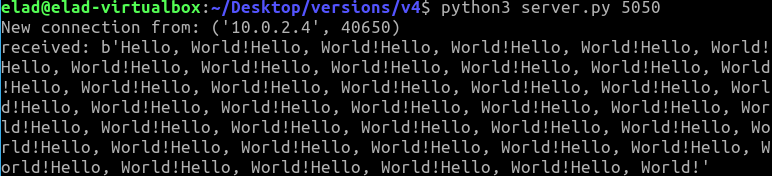
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

קוד השרת V4:

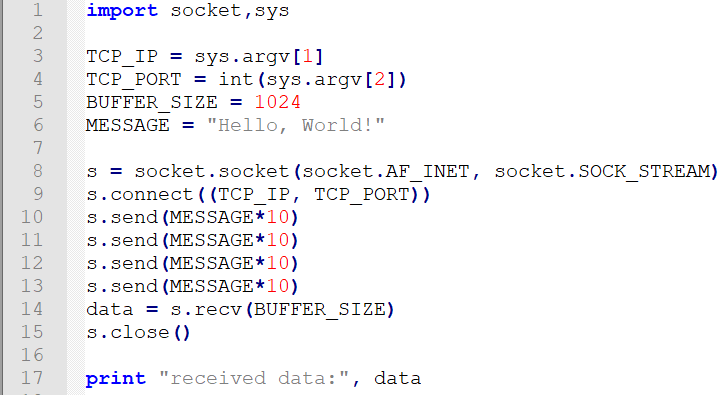


הסבר מילולי -

השרת זהה לשרת V1 מלבד זה שבלולאה הפנימית הוא ישן 5 שניות לפני שהוא מקבל בקשה מהלקוח.

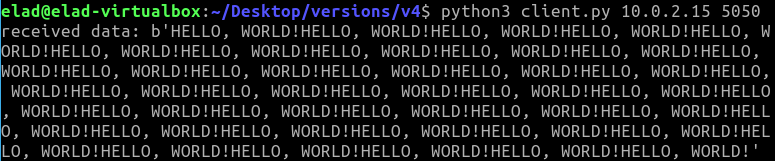
פלט השרת - חיבור חדש ולאחר קצת יותר מ5 שניות מודפס השאר.

קוד הלקוח V4:

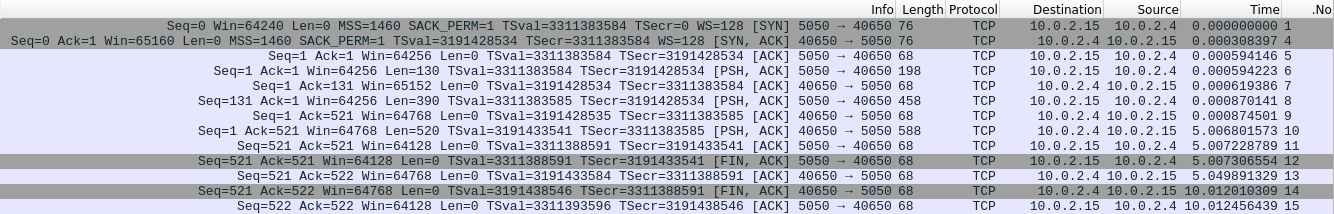


הסבר מילולי -

הקליינט זהה לV1 חוץ משליחת ההודעה לשרת. הקליינט משרשר את "Hello, World!" עשר פעמים, ושולח אותו כך ב4 שליחות שונות אחת אחרי השנייה.

פלט הלקוח - תגובת השרת לבקשה, מודפסת רק לאחר קצת יותר מ5 שניות.

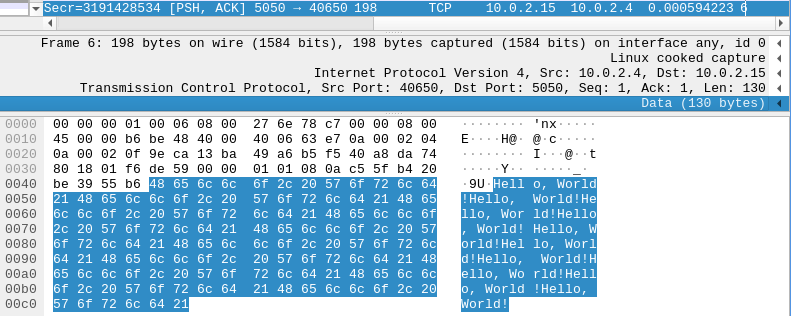
הסבר התעבורה בwireshark -

נשים לב, הIP של הלקוח הוא 10.0.2.4, ושל השרת 10.0.2.15.

בחבילה 1 הלקוח שולח בקשת הסתנכרנות לשרת.

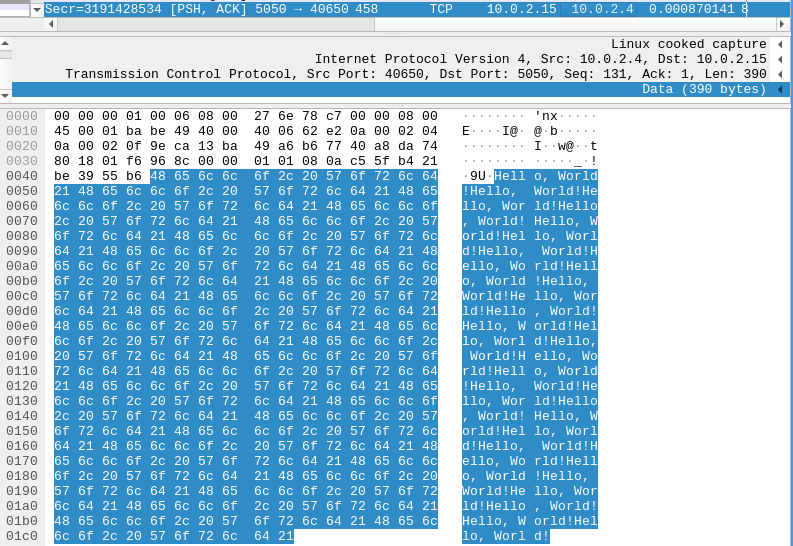
בחבילה 4 השרת מאשר את בקשת הSYN של הלקוח ושולח לו בקשת הסתנכרנות משלו.

בחבילה 5 הלקוח מאשר את בקשת הSYN של השרת.

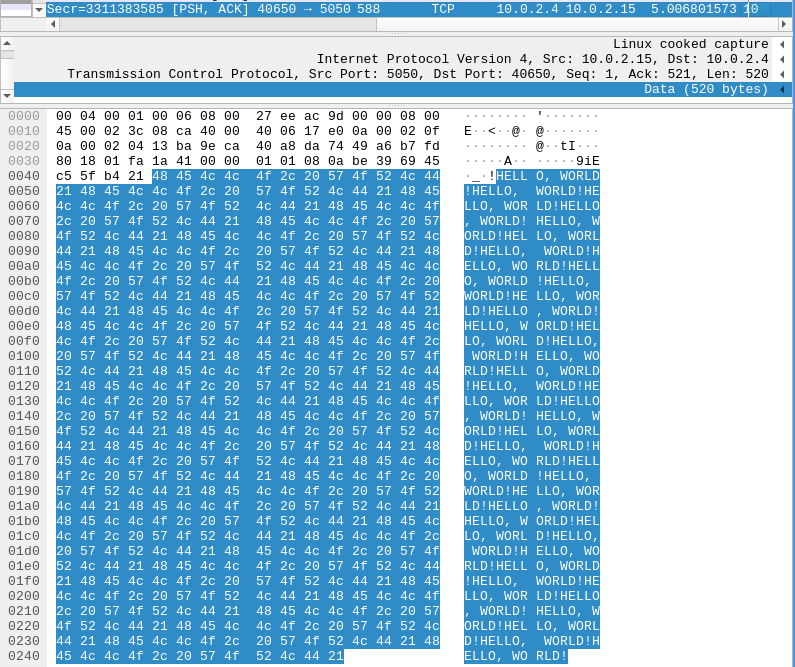
בחבילה 6 הלקוח שולח לשרת חבילה עם הבקשה "Hello, World!" משורשרת 10 פעמים (13\*10=130 בתים) - 

בחבילה 7 השרת מאשר ללקוח שקיבל את החבילה.

בחבילה 8 הלקוח שולח לשרת חבילה עם הבקשה "Hello, World!" משורשרת 30 פעמים(13\*10\*3=390 בתים) - שהרי הלקוח עושה עוד 3 פעמים send (חוץ מהמידע שנשלח בחבילה הקודמת) ובכל אחד כזה שולח את "Hello, World!" משורשרת 10 פעמים. TCP מחבר את החבילות ושולח אותם יחד.



בחבילה 9 השרת מאשר ללקוח שקיבל את החבילה האחרונה.

עד רגע זה השרת לא שלח כלום מכיוון שהוא נמצא בsleep של 5 שניות מיד לאחר יצירת החיבור עם הקליינט. לכן בשלב זה לא יקרה כלום עד לשנייה ה5 לאחר הכניסה לsleep. בשניה ה5 השרת יתעורר לחיים, יקבל את המידע שממתין לו בבאפר של שכבת התעבורה (בשלב זה זהו כל המידע שהקליינט שלח- כי הקליינט כבר הספיק מזמן לשלוח הכל), ידפיס אותו, וישלח תשובה לכל 4 הבקשות שנשלחו מהקליינט יחד. ולכן:  
בחבילה 10 השרת שולח ללקוח חבילה עם התשובה "HELLO, WORLD!" משורשר 40 פעמים. ניתן לראות שחבילה זו נשלחת מעט לאחר 5 שניות מתחילת ההתקשרות.

בחבילה 11הלקוח מאשר לשרת שקיבל את החבילה.

בחבילה 12 הלקוח שולח בקשת התנתקות לשרת.

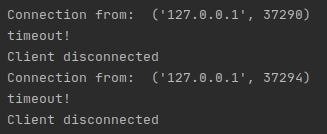
בחבילה 13 השרת מאשר את בקשת הFIN של הלקוח.

בחבילה 14 השרת שולח לו בקשת התנתקות משלו.

בחבילה 15 הלקוח מאשר את בקשת הFIN של השרת.

**ניתוח חלק ב:**

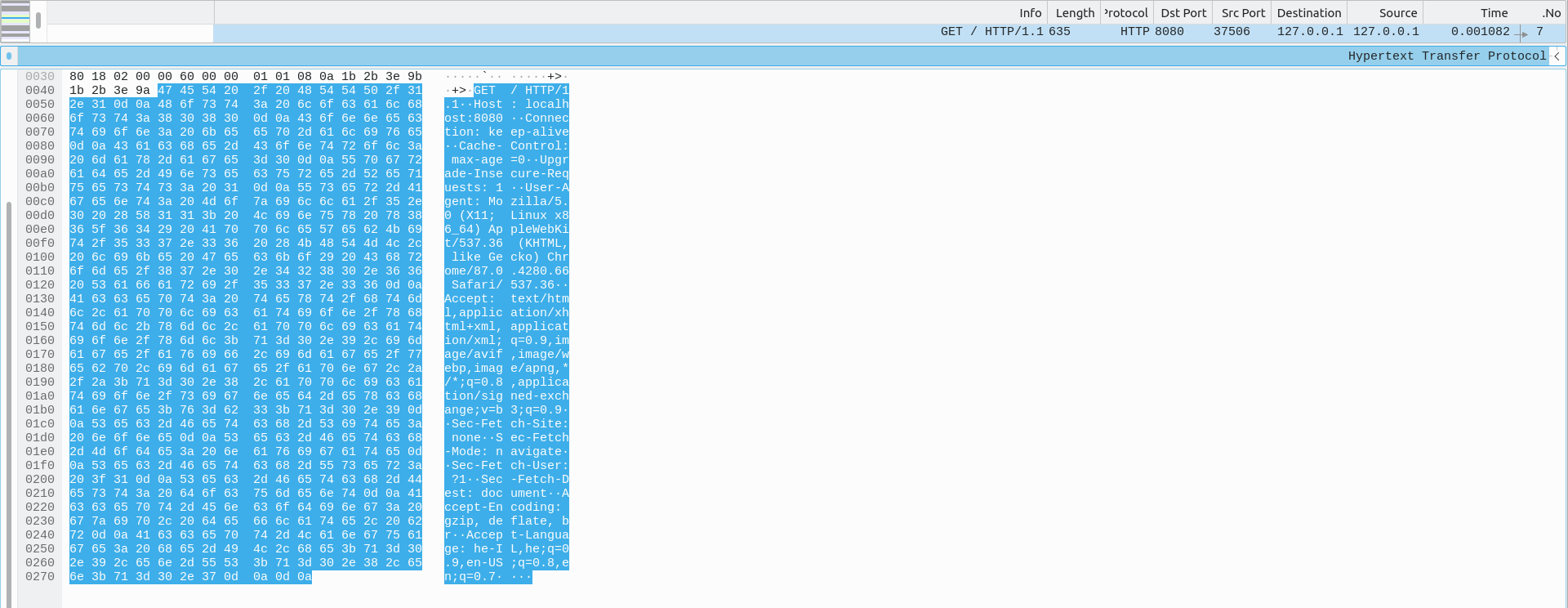
פניה לכתובת localhost:8080 (כלומר ל"/" - הroot)

הסבר התעבורה בwireshark -

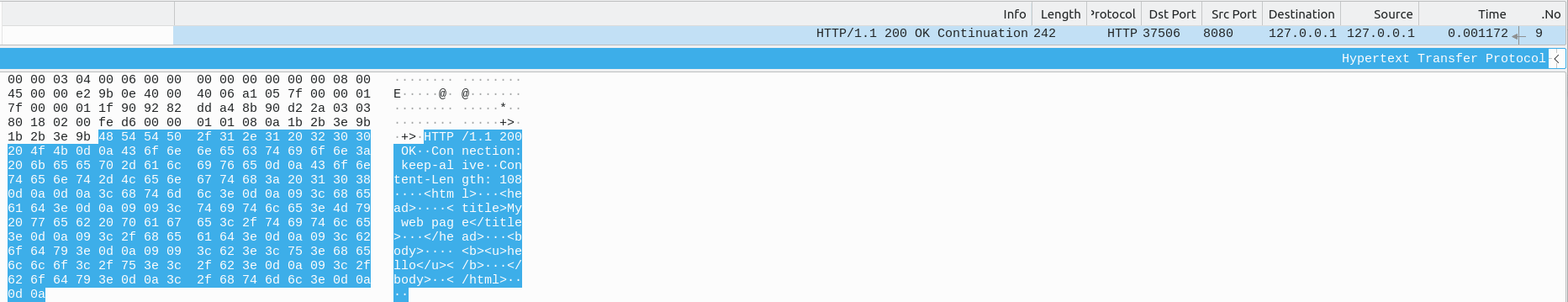
בהכנסת הכתובת localhost:8080 לדפדפן הוא מתחבר לשרת שלנו (שנמצא בפורט 8080 ובכתובת IP של localhost, כלומר 127.0.0.1) בתור קליינט מפורט 37506, ולאחר מכן שוב בתור קליינט אחר מפורט 37510. [[1]](#footnote-0)

ישנם 2 חיבורים(2 פרוצדורות handshake של חיבור ו2 של ניתוק).

רק בחיבור של הקליינט מפורט 37506 מתבצעת בקשה- בחבילה 7 הוא שולח בקשה לשרת עבור הקובץ "/"



ובחבילה 9 הוא מקבל ממנו תשובה הכוללת את הדף הנדרש(תכולת קובץ index.html שהתאמנו לבקשה "/")

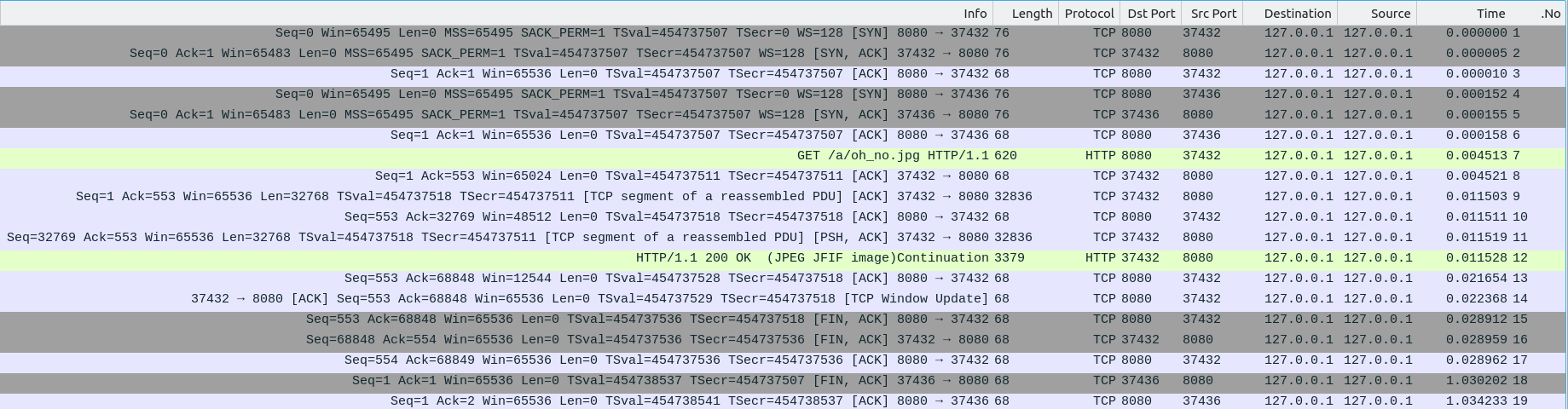


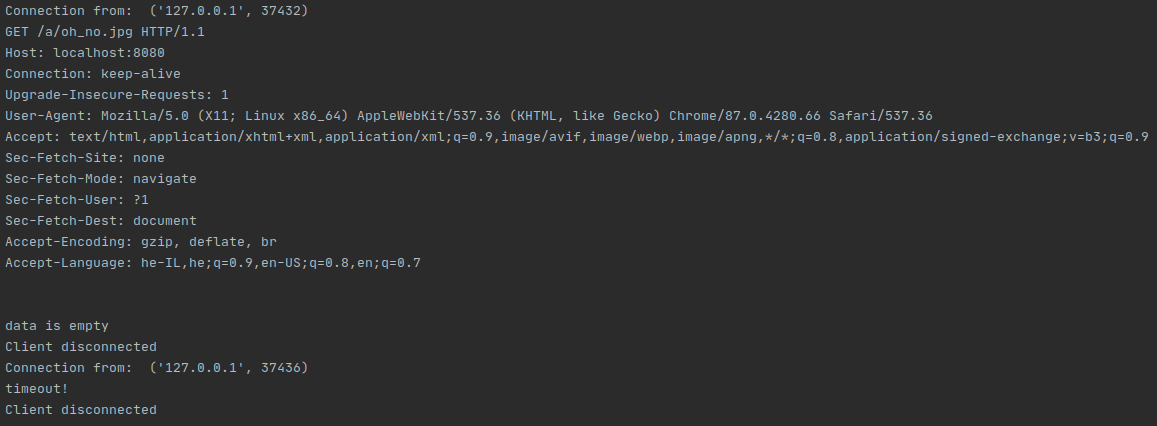
לא נשלחות בקשות נוספות לאחר מכן, ולכן בקוד השרת מתבצע timeout והוא עובר לחכות לקליינטים הבאים.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

פניה לכתובת localhost:8080/a/oh\_no.jpg

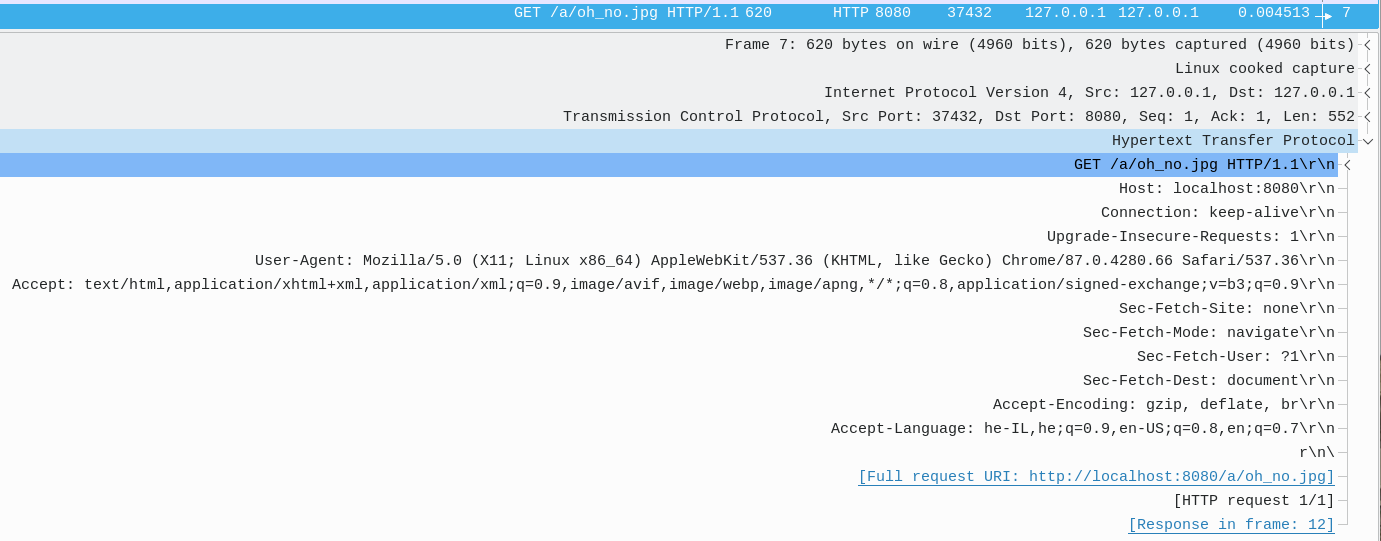
הסבר התעבורה בwireshark -





בהכנסת הכתובת localhost:8080/a/oh\_no.jpg לדפדפן הוא מתחבר לשרת שלנו (שנמצא בפורט 8080 ובכתובת IP של localhost, כלומר 127.0.0.1) בתור קליינט מפורט 37432, ולאחר מכן שוב בתור קליינט אחר מפורט 37436.

ישנם 2 חיבורים(2 פרוצדורות handshake של חיבור ו2 של ניתוק).

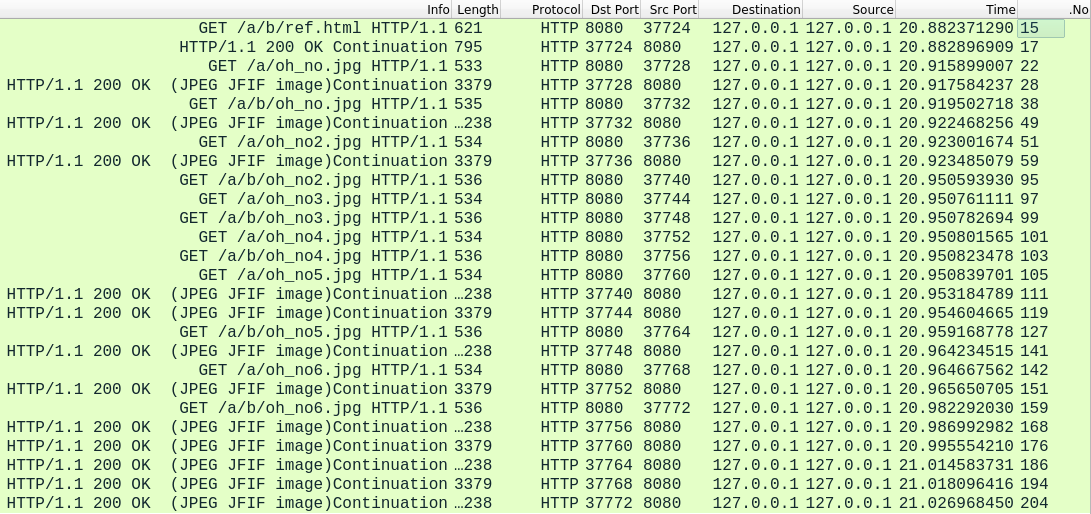
רק בחיבור של הקליינט מפורט 37432 מתבצעת בקשה- בחבילה 7 הוא שולח בקשה לשרת עבור הקובץ "/a/oh\_no.jpg"

בחבילות 8-12 התמונה נשלחת במספר סגמנטים(הבקשה נענית).

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

פניה לכתובת localhost:8080/a/b/ref.html

מפאת קוצר היריעה רק בקשות הHTTP מצורפות. התקשורת המלאה מפורטת בקובץ הPCAP.



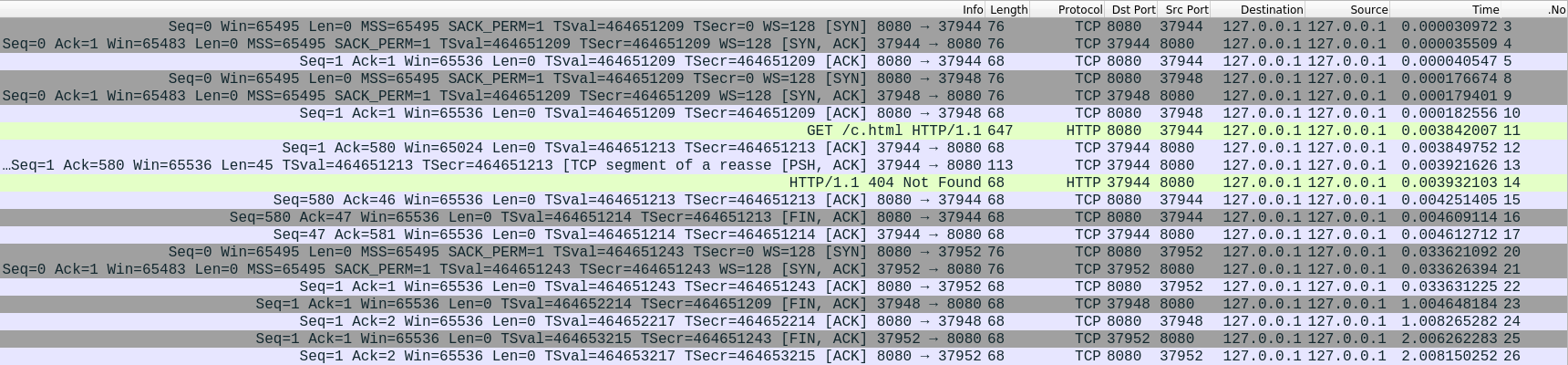
בהכנסת הכתובת localhost:8080/a/b/ref.html לדפדפן הוא מתחבר לשרת שלנו (שנמצא בפורט 8080 ובכתובת IP של localhost, כלומר 127.0.0.1) בתור קליינט מפורט 37724. הבקשה נענית והוא מקבל את הקובץ מהשרת. בקבלת הקובץ ref.html הדפדפן רואה שיש עוד הרבה אלמנטים של img, ולכן שולח בקשות נוספות(כל בקשה מקליינט נפרד) לשרת שלנו לקבלת תמונות אלו.

ישנם 13 חיבורים(13 פרוצדורות handshake של חיבור ו13 של ניתוק).

כאמור ישנה בקשה של הקליינט הראשון לקובץ ref.html ולאחר מכן עוד בקשה מכל קליינט נוסף(קליינט 2-13) לכל תמונה בקובץ.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

פניה לכתובת localhost:8080/c.html (קובץ שאינו קיים)



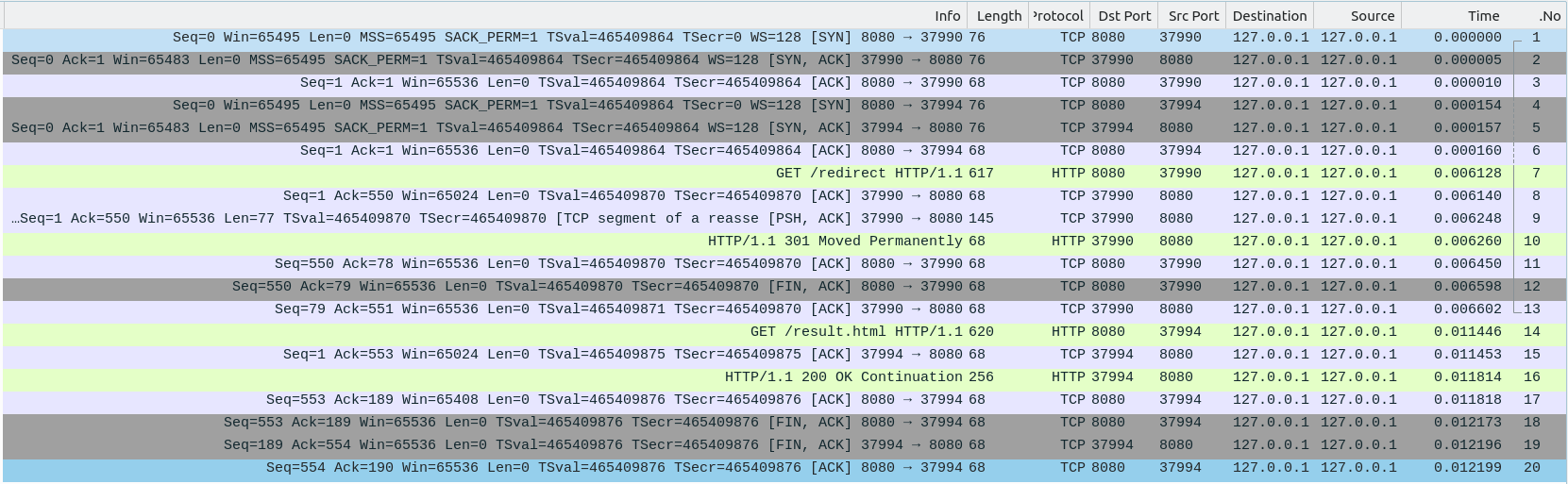
בהכנסת הכתובת localhost:8080/c.html לדפדפן הוא מתחבר לשרת שלנו (שנמצא בפורט 8080 ובכתובת IP של localhost, כלומר 127.0.0.1) בתור קליינט מפורט 37944, וקליינט נוסף מפורט 37948, ועוד אחד מפורט 37952. הקליינט מפורט 37944 שולח את בקשת הget אל השרת, ומכיוון שקובץ זה אינו קיים אצל השרת הוא מחזיר בתגובה קוד שגיאה "404 Not Found".

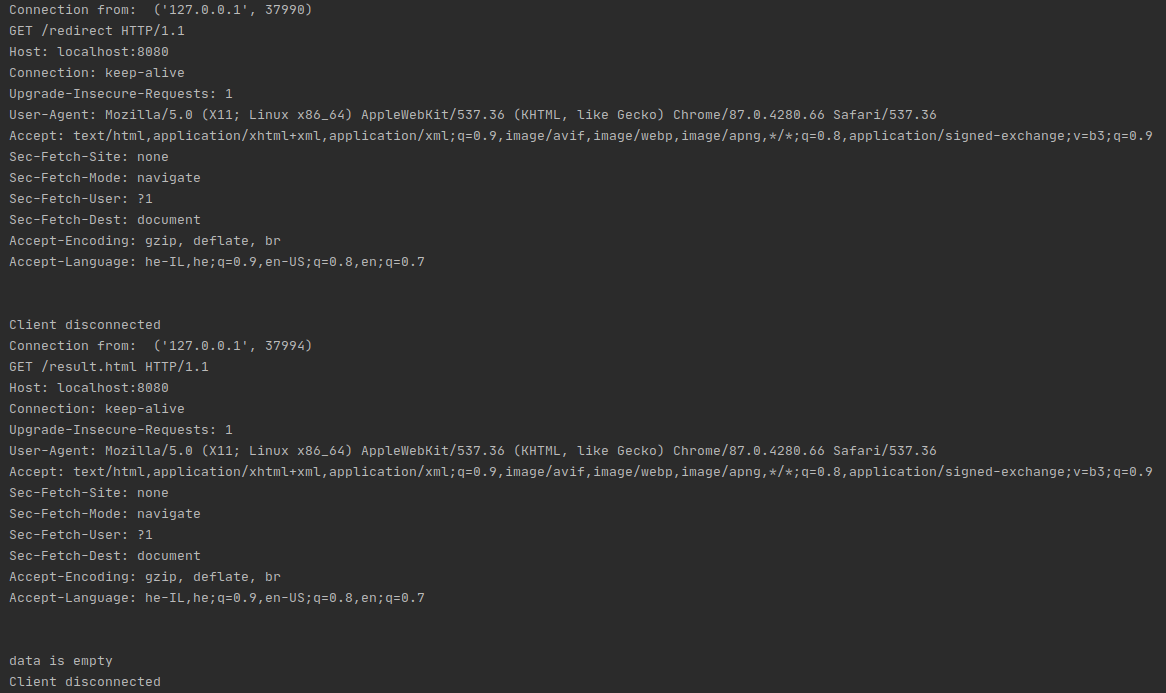
ישנם 2 חיבורים(2 פרוצדורות handshake של חיבור ו2 של ניתוק).

כאמור ישנה בקשה של הקליינט הראשון לקובץ c.html.

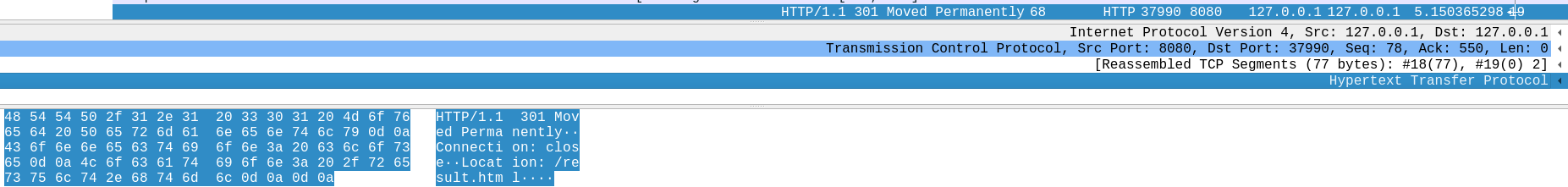
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

פניה לכתובת localhost:8080/redirect (ניתוב לresult.html)





בהכנסת הכתובת localhost:8080/redirect לדפדפן הוא מתחבר לשרת שלנו (שנמצא בפורט 8080 ובכתובת IP של localhost, כלומר 127.0.0.1) בתור קליינט מפורט 37990, וקליינט נוסף מפורט 37994. הקליינט מפורט 37990 שולח את בקשת הget, והשרת בתגובה משיב שהקובץ הועבר והמיקום שלו כרגע נמצא ב"result.html/".



כעת לאחר שהדפדפן קיבל את ההודעה אל הקליינט הראשון- הקליינט השני מפורט 37994 שולח בקשת get לקובץ במיקום החדש- "result.html/", והשרת מחזיר לו את דף זה.

ישנם 2 חיבורים(2 פרוצדורות handshake של חיבור ו2 של ניתוק).

כאמור ישנה בקשה של הקליינט הראשון לredirect ולאחר מכן בעקבות התשובה שהשרת החזיר- עוד בקשה מהקליינט השני לקובץ העדכני.

1. מספרי הפורט בתמונה מהterminal משובשים(עקב החלפת הדוגמא). המספרים בwireshark ובהסבר הם הנכונים. [↑](#footnote-ref-0)