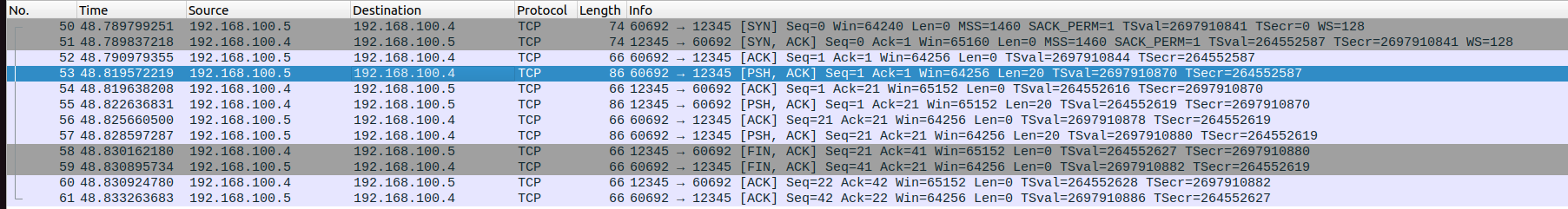
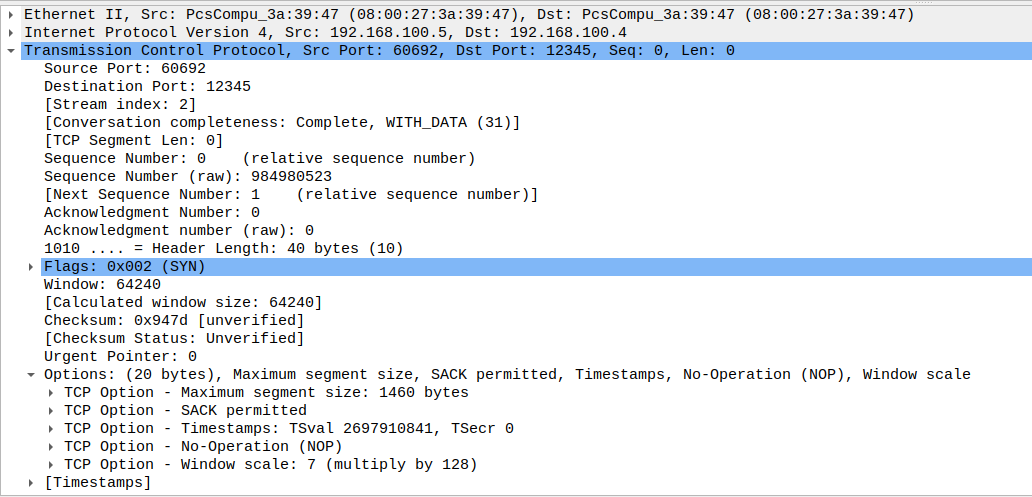
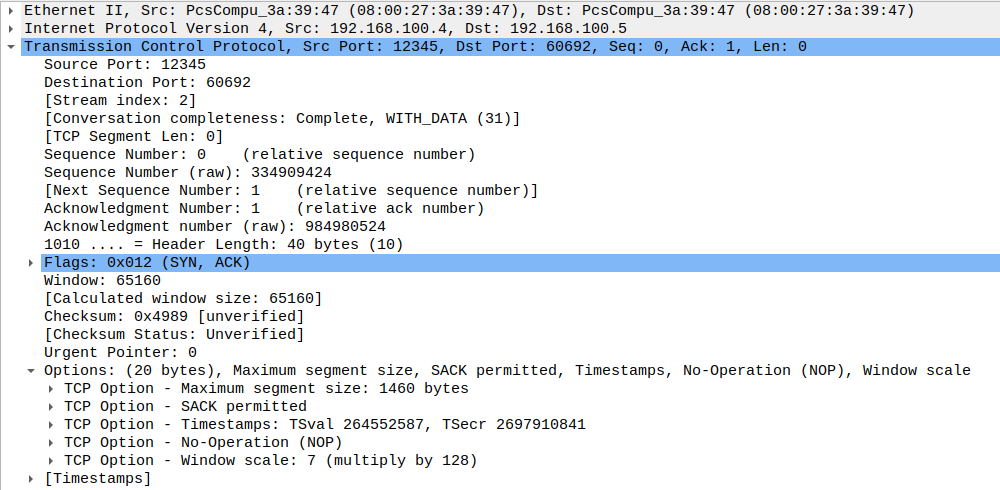
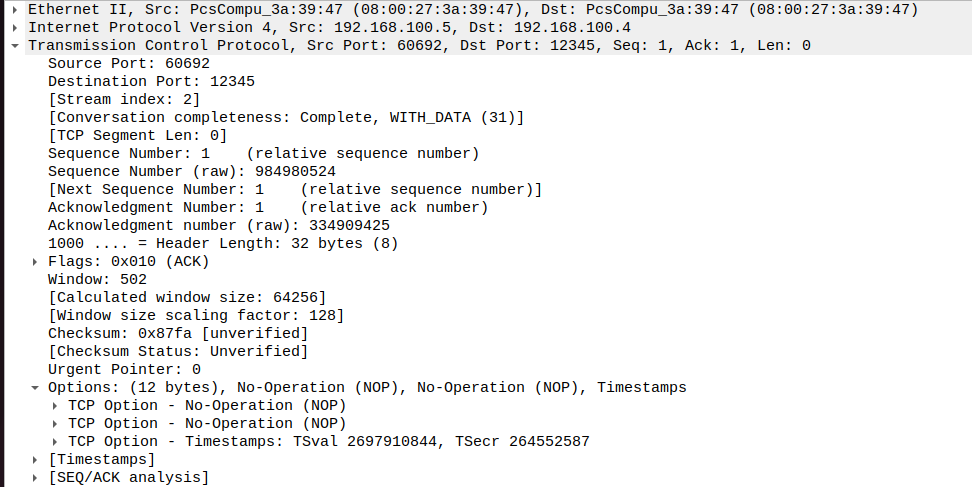
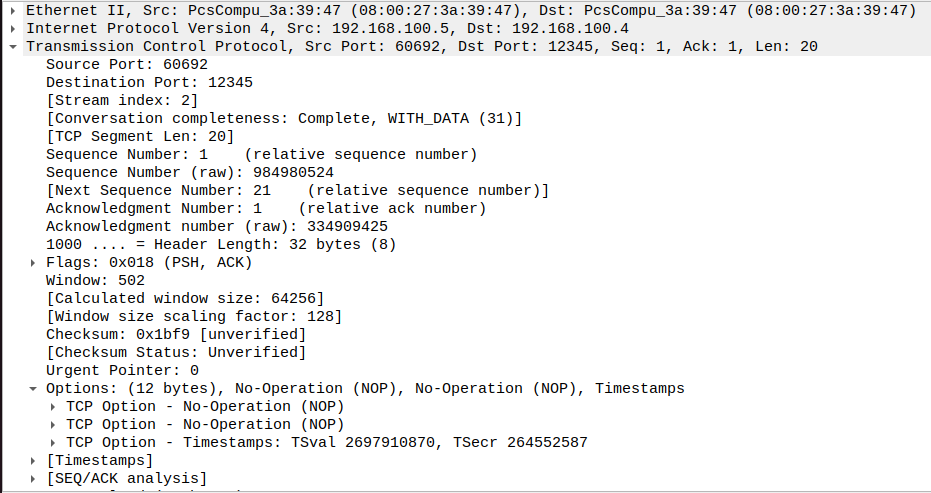
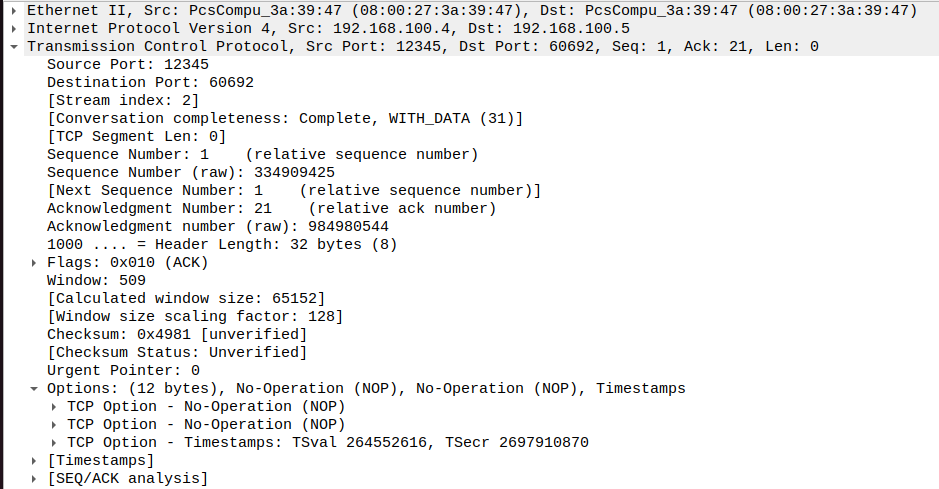
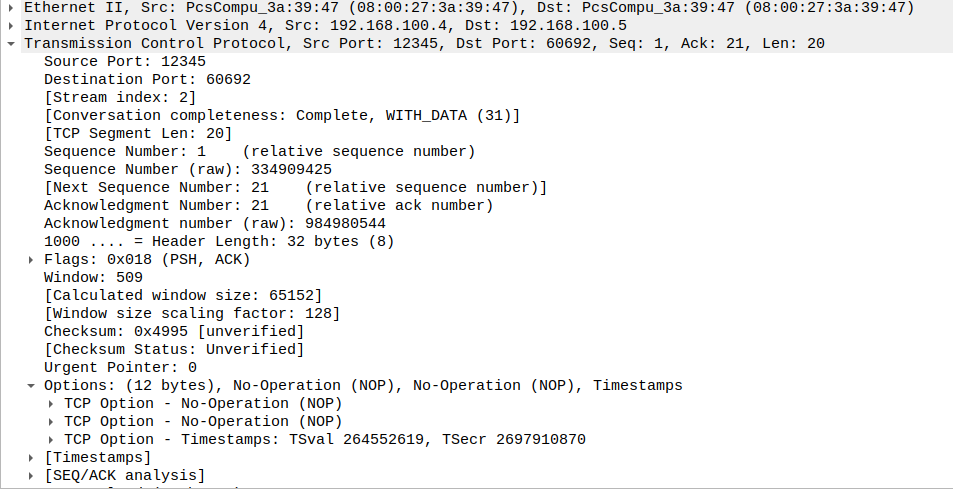
רשתות תקשורת תרגיל 2 - חלק א'  
  
ביצענו בקובץ client את השינויים הנדרשים (הוספנו עוד שורה לפני סגירת הסוקט, של שליחת מספרי תעודות הזהות שלנו:'206821258, 319024600' ועדכנו את ההודעה הראשונה שהלקוח שולח לשרת להיות השמות שלנו –'IdanSimai, IdoAharon' .  
בנוסף, סגרנו את השרת לאחר קבלת הודעה שנייה מהלקוח ולא אחרי הודעה אחת).  
הרצנו את הלקוח על מחשב אחד ואת השרת על מחשב אחר כך ששני המחשבים באותה הרשת(שניהם מכונות וירטואליות שונות עם כתובות IP שונות.  
 כתובת לקוח – 192.168.100.5, כתובת שרת - 192.168.100.4).

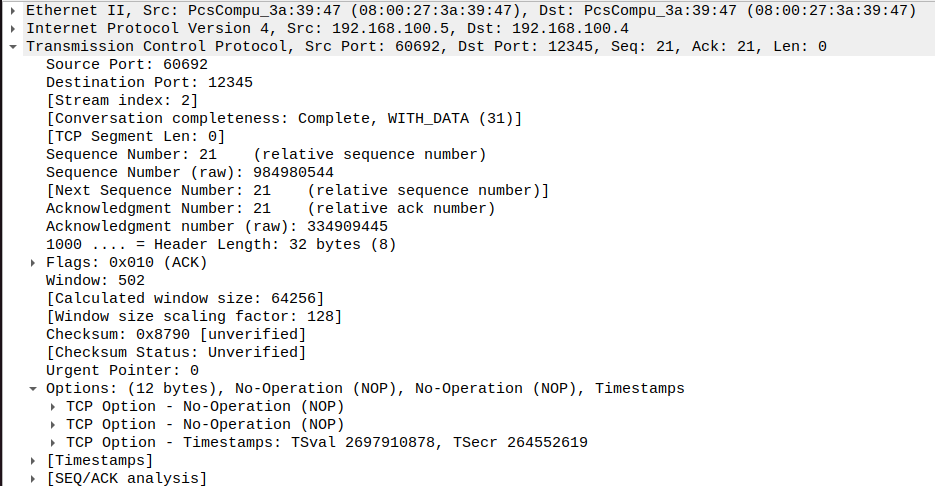
הסנפנו ב wireshark את התעבורה על ידי סינון לפי כתובות ה ip של הלקוח והשרת ולפי הפרוטוקול שבו השתמשנו שהוא TCP בצורה הבאה:  
 השתמשנו בכרטיס רשת enp2s0 לצורך הסינון בwireshark . זהו כרטיס רשת המיועד לתקשורת בין מחשבים שונים.

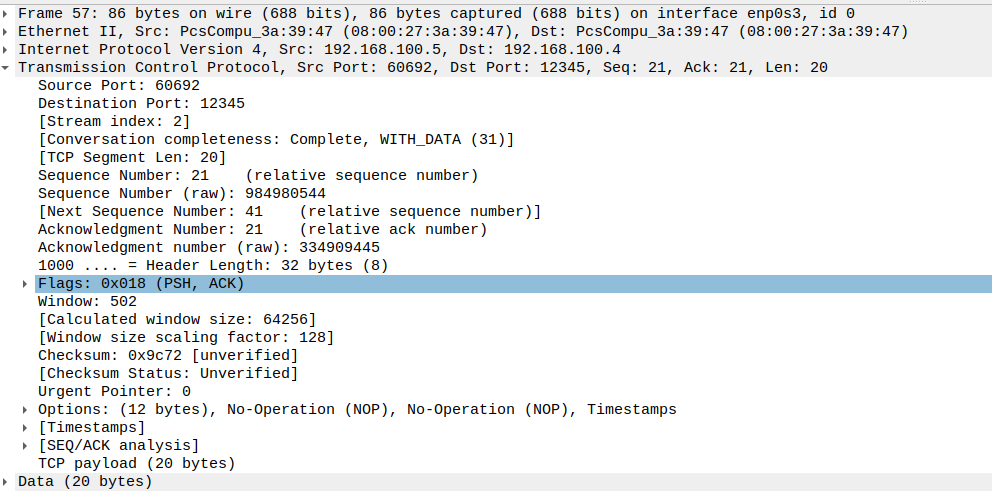
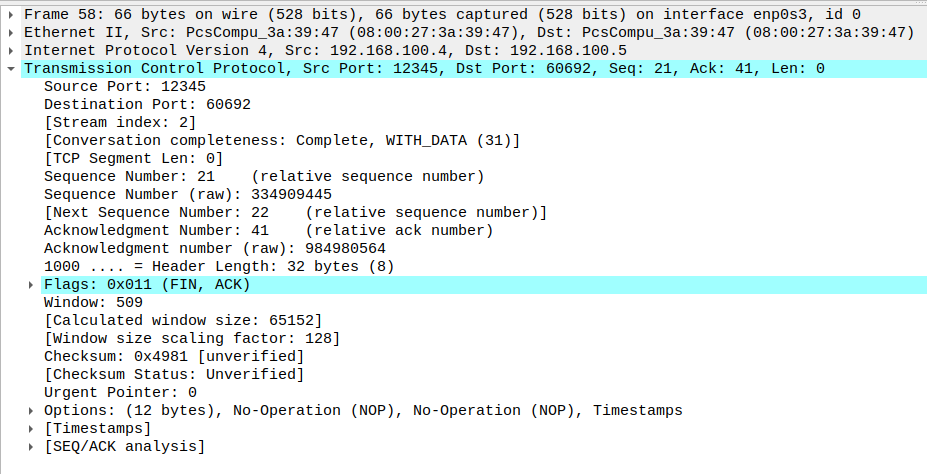
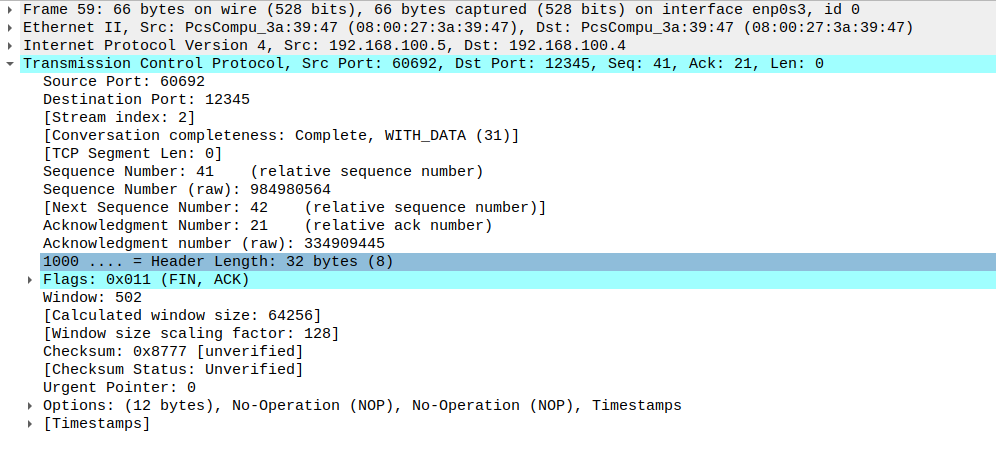
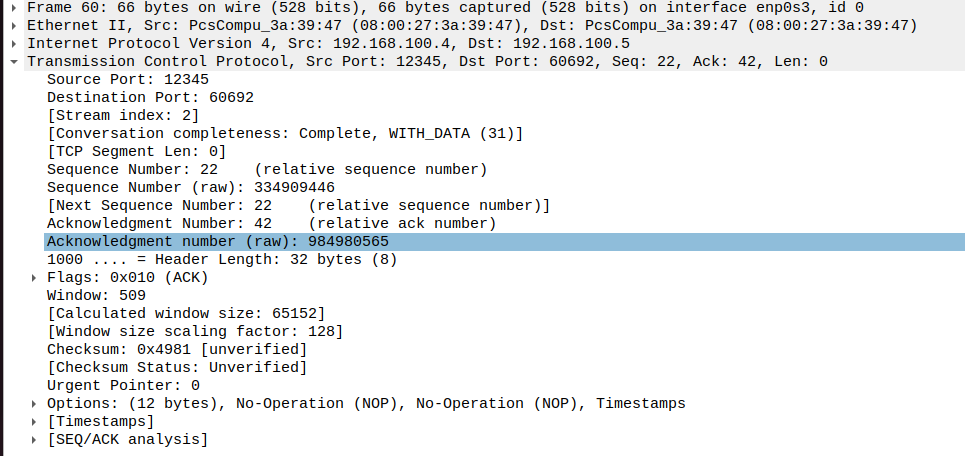
ביצענו סינון לחבילות שנשלחו מהשרת ללקוח וגם להפך.  
סה"כ קיבלנו 12 חבילות.  
  
בניגוד לפרוטוקול UDP אנו מבצעים תהליך לחיצת ידיים, הקמת חיבור.  
בשלב זה מתבצע סנכרון בין שני הצדדים - השרת והלקוח.  
שלוש החבילות הראשונות, אשר מוצגות בצילום המסך לעיל מייצגות תהליך זה.  
שלוש החבילות הללו יהיו ללא שכבת אפליקציה, יכילו את הTCP header ומטה(SYN , SYN ACK,(ACK.  
החבילה הרביעית מייצגת את שליחת ההודעה הראשונה של הלקוח לשרת(PSH ACK).  
אחריה נראה את חבילת ה ACK המגיעה מהשרת ללקוח כאישור.  
כתשובה ללקוח, השרת שולח הודעה בחזרה(PSH ACK) שבמקרה שלנו זה השמות בUPPER CASE.  
לאחר מכן, הלקוח מחזיר לוACK כאישור, כחבילה ה7.  
לאחר מכן, הלקוח שולח הודעה נוספת לשרת((PSH ACK, במקרה שלנו עם תעודות הזהות.  
החבילה הבאה היא חבלת הFIN ACK שהשרת שולח ללקוח, אשר מדברת על סיום התקשורת ביניהם.  
לאחר מכן, גם הלקוח שולח חזרה FIN ACK.  
השרת מחזיר לו ACK כאישור(אישור שכולל גם אישור על החבילה של העברת מספרי תעודת הזהות, שכן ACK = 42 מה שאומר שכל המידע התקבל).  
והלקוח מחזיר חזרה ACK אישור.  
כעת ננתח חבילה חבילה:  
חבילה (50)1:

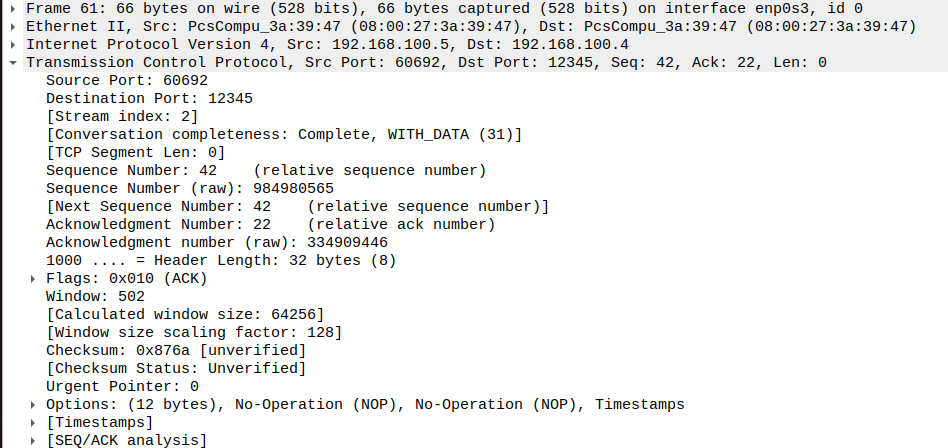
אנו יכולים לראות בwireshark שפורט המקור הוא 60692 ופורט היעד 12345.  
חבילה זו נשלחה מהלקוח לשרת.  
בנוסף לכך, הSequence number שמופיע בחבילה זו הוא 984980523.

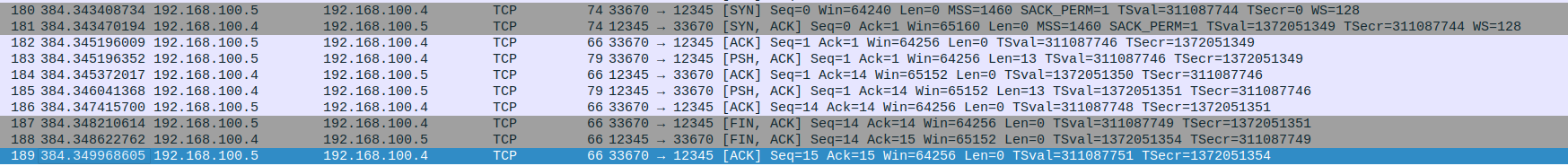
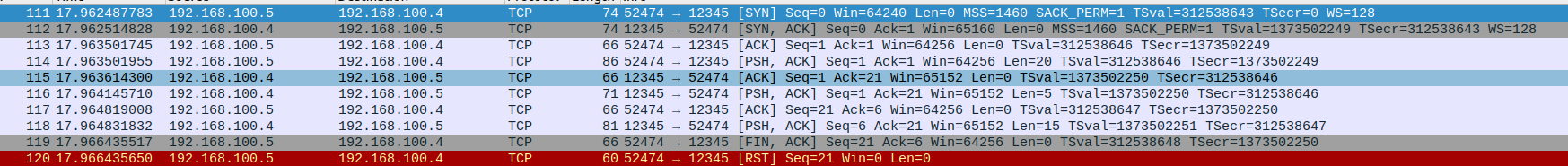
מספר זה הוא המספר הסידורי שממנו הלקוח מתחיל.  
רואים גם את השדה של number Acknowledgement שערכו הוא אפס מכיוון שדגל ה ACK כבוי ואין עדיין משמעות לערך זה.  
לאחר מכן אנו רואים את השדה length Header שאורכו 40 בתים.  
הדגל SYN דולק, זה מתבטא בצילום המסך בכך שבלשונית flags בשורה של SYN הביט דלוק.  
SYN - מלשון המילה סינכרוניזציה.  
כאשר נשלחת חבילה עם ביט זה דולק, המשמעות היא - שהלקוח רוצה להסתנכרן מול השרת.  
לאחר הדגלים, מצוין לנו גודל החלון שהוא 64240, שזוהי כמות המידע שהלקוח יכול לקבל מהשרת בבת אחת מבלי צורך ב ACK.  
יש לנו את השדה Checksum, כשהחבילה מגיעה לשרת, שכבת התעבורה מבצעת בעצמה את חישוב ה Checksum על החבילה.  
אם ערך זה שונה מהערך שמוצג בשדה ה Checksum אנו יודעים שהחבילה לא תקינה, כלומר חלק מהביטים השתבשו בדרך.  
 אם Checksum לא תקין - זורקים את החבילה, ובעתיד תשלח חבילה חדשה(מכיוון שזה TCP).  
 יש לנו לשונית של Options ששם מופיע הMSS,כלומר גודל ה data בשכבת האפליקציה לא יכול לעבור ערך זה(לכל חבילה), שערכו הוא 1460 בתים.   
  
מעכשיו לא נפרט יותר על הפורטים ושאר המידע שלא רלוונטי.  
  
  
  
חבילה (51)2:  
הSequence number שמופיע בחבילה זו הוא 334909424.  
מספר זה הוא המספר הסידורי שממנו השרת מתחיל.   
 Acknowledgement numberהוא 984980524 כך שניתן לראות   
שהוא באמת ה number sequence של חבילה 1 בתוספת ביט אחד.  
בכך הוא יודיע ללקוח שהוא קיבל ממנו את החבילה.  
הפעם, שני הדגלים דולקים - ACK וגם דגל SYN.  
הביט הדלוק של SYN מציין כי השרת רוצה להסתנכרן מול הלקוח הפעם(שכן הוא השולח של ההודעה הזו)ואילו הביט הדלוק של ACK מציין שהערך שמופיע בשדהnumber Acknowledgment הוא בעל משמעות(גדול מ0).  
החל מחבילה זו בכל החבילות הביט של ACK דולק.  
  
חבילה (52)3:  
  
חבילה זו היא החבילה האחרונה בתהליך הקמת החיבור, חבילה זו נשלחת מהלקוח לשרת, כדי לאשר שחבילת השרת התקבלה אצל הלקוח.  
הSequence number בחבילה זו הוא 984980524, כך שהוא גדל ב1 ביחס לחבילה הקודמת שנשלחה מהלקוח לשרת.  
מספר זה הוא המספר הסידורי שהשרת מצפה לו כפי שראינו בחבילה הקודמת דרך הAcknowledgement number .  
רואים גם את השדה שלAcknowledgement number שערכו הוא 334909425, זה המספר הסידורי שהלקוח אומר בעזרתו שאם השרת ישלח חבילה, הוא מצפה שהיא תהיה עם מספר סידורי זה .

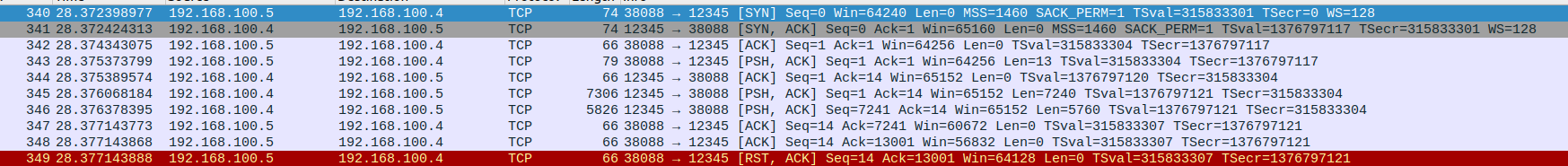
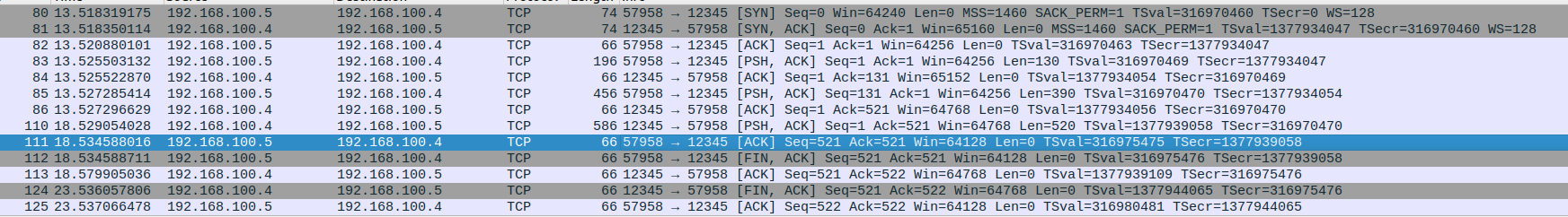
חבילה(53) 4:  
הלקוח שולח את ההודעה הראשונה לשרת, ההודעה מכילה את הטקסט הבא: "IdanSimai, IdoAharon".  
חבילה זו מייצגת את ההודעה הנ"ל כאשר הפעם כוללת בתוכה שכבת אפליקציה המכילה את תוכן ההודעה.  
מספר הבתים בשכבת האפליקציה הוא כגודל הטקסט - 20 בתים. הSequence number בחבילה זו הוא 984980524.  
המספר לא השתנה מהחבילה הקודמת, כיוון שהלקוח לא שלח מידע בשכבת האפליקציה והשרת לא שלח ACK שמצפה לקבל מידע מהלקוח עבור Sequence number גבוה יותר.  
רואים גם את השדה שלAcknowledgement number שערכו הוא 334909425, זהה להודעה הקודמת.  
  
חבילה(54) 5:  
השרת שולח חבילת ACK ללקוח, ע"מ לאשר שקיבל את החבילה שהלקוח שלח לו.  
עבור חבילה זו אין שכבת אפליקציה, (אין הודעה בפועל אלא רק ACK).  
הSequence number בחבילה זו הוא 334909425, זה המספר הסידורי שהלקוח מצפה לקבל, כמו שראינו בAcknowledgement number של החבילה הקודמת.  
רואים גם את השדה של Acknowledgement number שערכו הוא 984980544.  
בעזרת ערך זה, אנו רואים שהשרת באמת קיבל את המידע שהלקוח שלח לו, שהרי הערך גדל ב20 בתים, שזה בעצם גודל המידע שנשלח.  
  
חבילה(55) 6:  
  
בנוסף לחבילת הACK שהשרת שולח הוא שולח ללקוח חבילה עם מידע כפי שהוגדר לו לעשות בקוד(להחזיר את השמות בUPPERCASE).  
ההודעה שהשרת שולח מכילה את הטקסט הבא:  
 "IDANSIMAI, IDOAHARON".  
גם במקרה זה, גודל שכבת האפליקציה(ההודעה שנשלחה) הוא 20 בתים.  
הSequence number בחבילה זו הוא 334909425.  
רואים גם את השדה של Acknowledgement number שערכו הוא 984980544.  
שני הערכים לא השתנו מאז החבילה הקודמת, מכיוון שבחבילה הקודמת הלקוח לא שלח חבילה עם Acknowledgement number שונה.

חבילה (56) 7:

הלקוח מחזיר לשרת הודעת ACK.  
הSequence number בחבילה זו הוא 984980544, שינוי של 20 יותר מההודעה הקודמת שהלקוח שלח, וזהה ל Acknowledgement number מהחבילה הקודמת שהשרת שלח ללקוח.  
רואים גם את השדה של Acknowledgement number שערכו הוא 334909445, שוב שינוי של 20 יותר מההודעה הקודמת שהלקוח שלח, וזהה ל Sequence number של החבילה הקודמת שהשרת שלח(שכן אנו מעלים ב20 בתים את המקום ממנו אנו אומרים לצד השני להתחיל לקרוא, וקיבלנו גם עוד 20 מהשרת חזרה).  
  
חבילה (57) 8:  
זוהי החבילה השנייה שהלקוח שולח לשרת(""206821258, 319024600).  
גודל החבילה הוא שוב 20 בתים.  
הSequence number בחבילה זו הוא 984980544, ללא שינוי מההודעה הקודמת, שכן נרצה שהשרת עדיין יקרא מאותו המקום.  
רואים גם את השדה של Acknowledgement number שערכו הוא 334909445, שוב ללא שינוי כיוון שהשרת לא שלח כלום.  
  
חבילה(58) 9:  
בחבילה הזו, השרת שולח ללקוח שהוא סיים את התקשורת מהצד שלו.  
זה מסומן על ידי ביט דולק בדגל FIN.  
הלקוח עדיין יכול לשלוח הודעות לשרת.  
הSequence number בחבילה זו הוא 334909445, 20 יותר מהפעם הקודמת שכן השרת החזיר הודעה ללקוח שאורכה 20 בתים.  
רואים גם את השדה של Acknowledgement number שערכו הוא 984980564, יותר ב20 מהפעם הקודמת עקב 20 הבתים שהתקבלו מהלקוח עקב ההודעה השנייה שהוא שלח.  
  
חבילה(59) 10:  
בחבילה הזו, הלקוח שולח לשרת שהוא סיים את התקשורת מהצד שלו.  
זה מסומן על ידי ביט דולק בדגל FIN.  
הSequence number בחבילה זו הוא 984980564, 20 יותר מהפעם הקודמת שכן הלקוח שלח הודעה נוספת לשרת שאורכה 20 בתים.  
רואים גם את השדה של Acknowledgement number שערכו הוא 334909445, זהה לפעם הקודמת כיוון שהשרת לא שלח כלום.  
  
חבילה(60) 11:  
השרת מחזיר ACK ללקוח על בקשת סיום ההתקשרות שלו.  
עבור חבילה זו אין שכבת אפליקציה, השרת לא שולח הודעה אלא רק שולח ACK.  
הSequence number בחבילה זו הוא 334909446, 1 יותר מהפעם הקודמת שכן השרת החזיר הודעה ללקוח שאורכה 1 בתים((ACK.  
רואים גם את השדה של Acknowledgement number שערכו הוא 984980565, זהה לפעם הקודמת שכן הלקוח לא שלח כלום.

  
חבילה(61) 12:  
הלקוח מחזיר ACK ללקוח על בקשת סיום ההתקשרות שלו.  
עבור חבילה זו אין שכבת אפליקציה, הלקוח לא שולח הודעה אלא רק שולח ACK.  
הSequence number בחבילה זו הוא 984980565, 1 יותר מהפעם הקודמת שכן הלקוח החזיר הודעה ללקוח שאורכה 1 בתים((ACK.  
רואים גם את השדה של Acknowledgement number שערכו הוא 334909446, 1 יותר מהפעם הקודמת עקב הACK שהשרת שלח וקראנו בהצלחה.  
  
הסבר עבור :  
קוד השרת:  
כתובת הIP המוגדרת היא '0.0.0.0' כלומר, כל כתובת IP, כל עוד הוא הPort המתאים לסוקט.  
את הפורט אנחנו מקבלים כארגומנט בשורת הפקודה.  
בנוסף ניתן לראות כי הבאפר לכל אורך של הודעה הוא 1024 בתים.  
השרת מאזין ללקוח 1 כל פעם.  
הsocket של השרת לא נסגר והשרת רץ בלולאה אינסופית, שבלולאה זו הוא מקבל לקוח.  
בפונק' accept, אנו מקבלים זוג של conn, addr שconn הוא אובייקט מסוג סוקט שיכול לקבל data בחיבור, וaddr הוא כתובת הIP של הלקוח.  
לאחר יצירת התקשורת עם הלקוח הספציפי, מודפסת הודעה עם מי התבצעה התקשורת ומתחיל תהליך של שליחת הודעות בין השרת ל לקוח בתוך לולאה נוספת, בתוכה השרת משתמש ב socket הייעודי((conn על מנת לקבל מידע מהלקוח.  
אם הוא קיבל מידע, הוא מדפיס אותו ושולח הודעה בחזרה ללקוח עם אותו המידע באותיות גדולות.  
אחרת - לא התקבל מידע מהלקוח, אנו יוצאים מהלולאה הפנימית והתקשורת ביניהם תסתיים ע"י סגירת הסוקט.  
  
קוד הלקוח:  
הלקוח מקבל כארגומנטים בשורת הפקודה את כתובת הIP של השרת ואת מספר הPort שלו.   
גודל הבאפר, כלומר גודל ההודעה שאפשר לקרוא בכל פעם הוא 1024 בתים. יוצרים socket של פרוטוקול TCP ומתחברים לשרת לפי ה IP וה Port שהוגדרו. לאחר מכן הלקוח שולח את ההודעה לשרת:  
"Hello, World!"

והלקוח מקבל תשובה מהשרת ע"י פקודת recv ומכניס את המידע למשתנה data.  
לאחר מכן סוגרים את החיבור, ובזאת מסיים הלקוח את התקשורת מהצד שלו. לאחר מכן הלקוח מדפיס את המידע שנשלח מהשרת(המשתנה data).  
  
ניתוח החבילות בwireshark:  
  
החבילה הראשונה נשלחת מהלקוח לשרת עם הדגל SYN ע"מ להסתנכרן מולו.  
לאחר מכן גם השרת שולח ללקוח חבילה עם הדגל SYN דלוק ע"מ להסתנכרן מול הלקוח ושולח גם ACK לאישור באותה החבילה.  
בחבילה השלישית הלקוח שולח לשרת חבילת ACK ע"מ לאשר את קבלת החבילה שנשלחה אליו ע"י השרת.  
כאן הסתיים תהליך הקמת החיבור.  
מיד לאחר מכן הוא שולח חבילה נוספת ובה כולל את ההודעה "hello world" כלומר חבילה זו כוללת את שכבת האפליקציה.  
החבילה החמישית - חבילה שהשרת שולח ללקוח בה הוא מאשר את קבלת החבילה שהלקוח שלח באמצעות .ACK   
לאחר מכן, השרת שולח ללקוח חבילה נוספת הכוללת מידע בשכבת האפליקציה((PSH ACK.  
הודעה זו היא ההודעה שהלקוח שלח באותיות גדולות.  
החבילה השביעית היא חבילת ACK שהלקוח שולח לשרת ע"מ להודיע לו שקיבל את ההודעה שלו.  
לאחר מכן, הלקוח שולח חבילה עם הדגל FIN ע"מ לבשר לשרת שסיים את התקשורת מהצד שלו.  
השרת שולח ללקוח גם כן חבילת FIN שיש בה גם ACKכאישור(גם הלקוח שלח ACK מקודם יחד עם הFIN).  
החבילה האחרונה היא חבילת ACK שהלקוח שולח לשרת ע"מ לומר לו שהוא קיבל את חבילת ה FIN שלו.  
  
  
הסבר עבור :  
קוד השרת:  
זהה לחלוטין לV1, למעט שכאן גודל הבאפר הוא 5 בתים.  
  
קוד הלקוח:  
זהה לחלוטין לV1, למעט שכאן תוכן ההודעה שונה וגודל הבאפר הוא 5 בתים.  
  
ניתוח החבילות בwireshark:  
  
החבילה הראשונה נשלחת מהלקוח לשרת עם הדגל SYN ע"מ להסתנכרן מולו. לאחר מכן גם השרת שולח ללקוח חבילה עם הדגל SYN דלוק ע"מ להסתנכרן מול הלקוח, יחד עם ACK באותה החבילה כדי לאשר את קבלת החבילה הקודמת.  
בחבילה השלישית הלקוח שולח לשרת חבילת ACK ע"מ לאשר את קבלת החבילה שנשלחה אליו.  
כאן מסתיים תהליך הקמת החיבור.  
לאחר מכן הלקוח שולח חבילה נוספת ובה כולל את ההודעה:  
"World! Hello, World!"

כלומר חבילה זו כוללת את שכבת האפליקציה.  
החבילה החמישית היא חבילה שהשרת שולח ללקוח בה הוא מאשר את קבלת החבילה שהלקוח שלח באמצעות ACK.  
לאחר מכן השרת שולח ללקוח חבילה נוספת הכוללת מידע בשכבת האפליקציה(5 הבתים הראשונים שהלקוח שלח באותיות גדולות).  
החבילה הבאה היא חבילה שהלקוח שולח עם הדגל ACK ע"מ לאשר לשרת שהוא קיבל את ההודעה.  
החבילה הבאה היא חבילה שהשרת שולח ללקוח עם יתרת ההודעה של הלקוח באותיות גדולות, ובאותה החבילה הוא מחזיר ACK גם כן.  
לאחר מכן הלקוח מכריז שהוא סיים את התקשורת ע"י שליחת חבילה לשרת עם הדגל FIN דלוק.  
החבילה האחרונה היא חבילה שהלקוח שולח לשרת גם כן, חבילה זו עם דגל RST דלוק נשלחת כתוצאה מכך שהשרת המשיך לשלוח חבילות לאחר שהלקוח סגר את התקשורת(השרת לא שלח FIN).  
  
הסבר עבור :  
קוד השרת:  
זהה לחלוטין לV1, למעט שהמידע שנשלח ללקוח הוא 1000פעמים המידע שנשלח ללקוח בV1 .  
  
קוד הלקוח:  
זהה לחלוטין לV1, למעט שכאן מקבלים מידע מהבאפר פעמיים ולא פעם אחת באמצעות הפקודה recv, ומכיוון שמקבלים מידע פעמיים גם מדפיסים פעמיים.  
  
ניתוח החבילות בwireshark:  
  
החבילה הראשונה נשלחת מהלקוח לשרת עם הדגל SYN ע"מ להסתנכרן מול השרת.  
לאחר מכן גם השרת שולח ללקוח חבילה עם הדגל SYN דלוק ע"מ להסתנכרן מול הלקוח, בנוסף מאשר את קבלת חבילת הלקוח ע"י ACK באותה החבילה.  
בחבילה השלישית הלקוח שולח לשרת חבילת ACK לאישור קבלה.  
כאן מסתיים תהליך הקמת החיבור.  
לאחר מכן הלקוח שולח חבילה נוספת ובה ההודעה:  
“Hello, World!"  
כלומר חבילה זו כוללת את שכבת האפליקציה.  
החבילה החמישית - חבילה שהשרת שולח ללקוח בה הוא מאשר את קבלת החבילה שהלקוח שלח באמצעות ACK.  
לאחר מכן השרת שולח ללקוח חבילה נוספת הכוללת מידע בשכבת האפליקציה, מידע זה הוא ההודעה שהלקוח שלח באותיות גדולות אלף פעמים.  
גם בחבילה השביעית, יש את שאר הפעמים הנותרות מתוך 1000.  
לאחר מכן, הלקוח שולח לשרת חבילת ACK המאשרת שקיבל ממנו את כל 13,000 הבתים, בשתי חבילות ACK שונות כפי שהשרת שלח לו בשתי חבילות.  
החבילה האחרונה היא חבילת RST לשרת שבעצם אומרת שהלקוח חייב לסיים את הקשר ולא קרא את כל מה שהשרת שלח לו.  
מה שקורה הוא שגודל ה buffer של הלקוח הוא 1024, ובכל קריאה של ה Data מה buffer הוא בעצם קורא 1024 בתים.  
מכיוון שמתבצעות רק 2 קריאות של recv, נקראים סה"כ 2048 בתים, מה שמשאיר 10952 בתים עדיין בבאפר, מידע שהלקוח לא מתכוון לקרוא(טיפה יותר מ13000 סה"כ, אך זה בעקבות הקידוד).  
לכן נשלחת חבילת RST .  
  
הסבר עבור :  
קוד השרת:  
זהה לחלוטין לV1, למעט שלפני קריאת המידע שנמצא בבאפר, השרת מחכה 5 שניות(בעזרת (sleep.  
קוד הלקוח:  
זהה לחלוטין לV1, למעט שכאן מבצעים 4 פעמים send ובכל אחת מהן שולחים 10 פעמים את ההודעה.  
  
  
ניתוח החבילות בwireshark:

החבילה הראשונה נשלחת מהלקוח לשרת עם הדגל SYN ע"מ להסתנכרן מולו. לאחר מכן גם השרת שולח ללקוח חבילה עם הדגל SYN דלוק ע"מ להסתנכרן מול הלקוח, בנוסף מאשר את קבלת חבילת הלקוח ע"י ACK באותה החבילה. בחבילה השלישית הלקוח שולח לשרת חבילת ACK ע"מ לאשר את קבלת החבילה.  
כאן מסתיים תהליך הקמת החיבור.  
לאחר מכן הוא שולח חבילה נוספת ובה כולל את

ההודעה:

“Hello, World!Hello, World!Hello, World!Hello, World!Hello, World!Hello, World!Hello, World!Hello,

World!Hello, World!Hello, World!"

כלומר חבילה זו כוללת את שכבת האפליקציה.  
החבילה החמישית - חבילה שהשרת שולח ללקוח בה הוא מאשר את קבלת החבילה שהלקוח שלח באמצעות ACK.  
לאחר מכן הלקוח שולח לשרת חבילה נוספת באורך 456 עם שאר התוכן, כלומר חבילה עם שכבת אפליקציה.  
לאחר מכן השרת שלח ללקוח חבילת ACK המאשרת את קבלת המידע מהלקוח.  
כעת, בחבילה הבאה אנו רואים פעם ראשונה שהשרת שולח מידע ללקוח, המידע נאסף בבאפר ונשלח באופן מרוכז לאחר 5 שניות באותיות גדולות. זה נגרם כתוצאה מהטיימר שהיה בקוד, ניתן גם לראות שעברו 5 שניות בwireshark.  
החבילה הבאה היא חבילת ACK שהלקוח שולח לשרת המאשרת את קבלת המידע.  
לאחר מכן הלקוח שולח חבילה עם הדגל FIN ע"מ לבשר לשרת שמבחינתו סיים את התקשורת. כתוצאה מכך, השרת שולח חבילת ACK המאשרת שקיבל את חבילת ה FIN.  
לאחר מכן, שולח השרת חבילה נוספת המבשרת שגם הוא סיים את התקשורת עם הלקוח, והוא מבצע זאת על ידי הדלקת הדגל FIN גם כן.  
החבילה האחרונה היא חבילת ACK שהלקוח שולח לשרת המאשרת את קבלת חבילת ה FIN שלו.

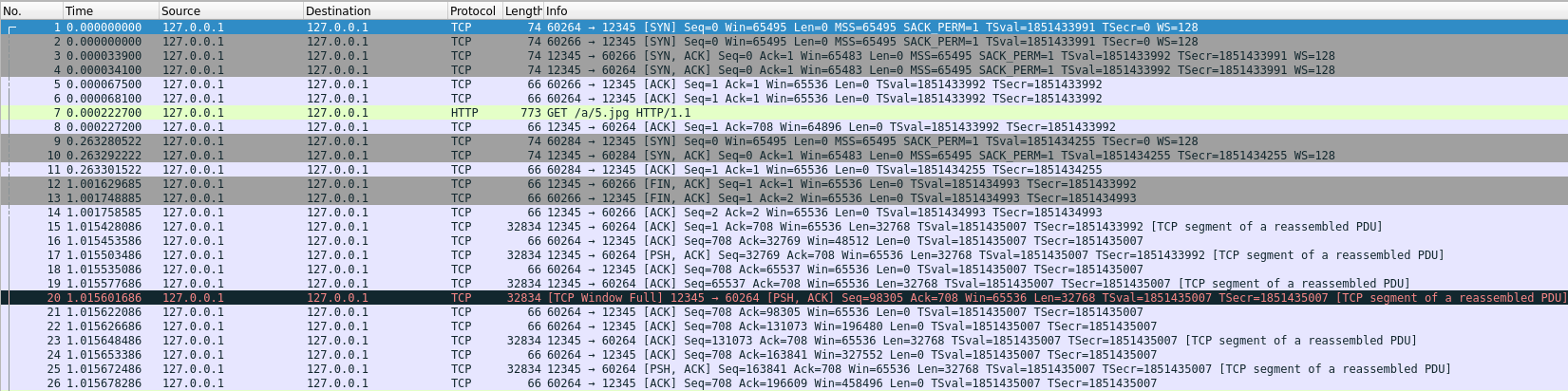
**חלק ב**

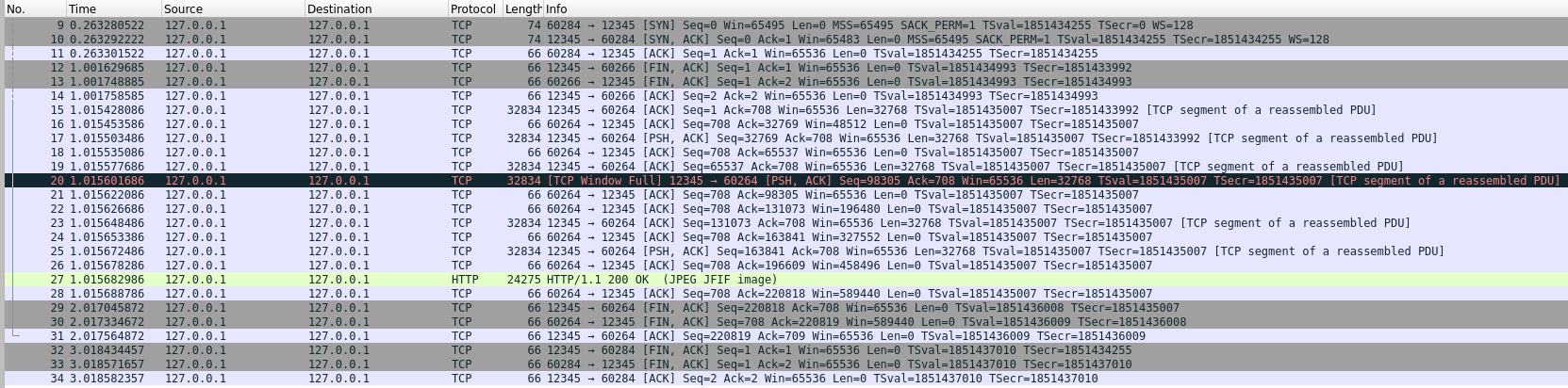
הרצה של /a/5.jpg :

הכל



פקטה

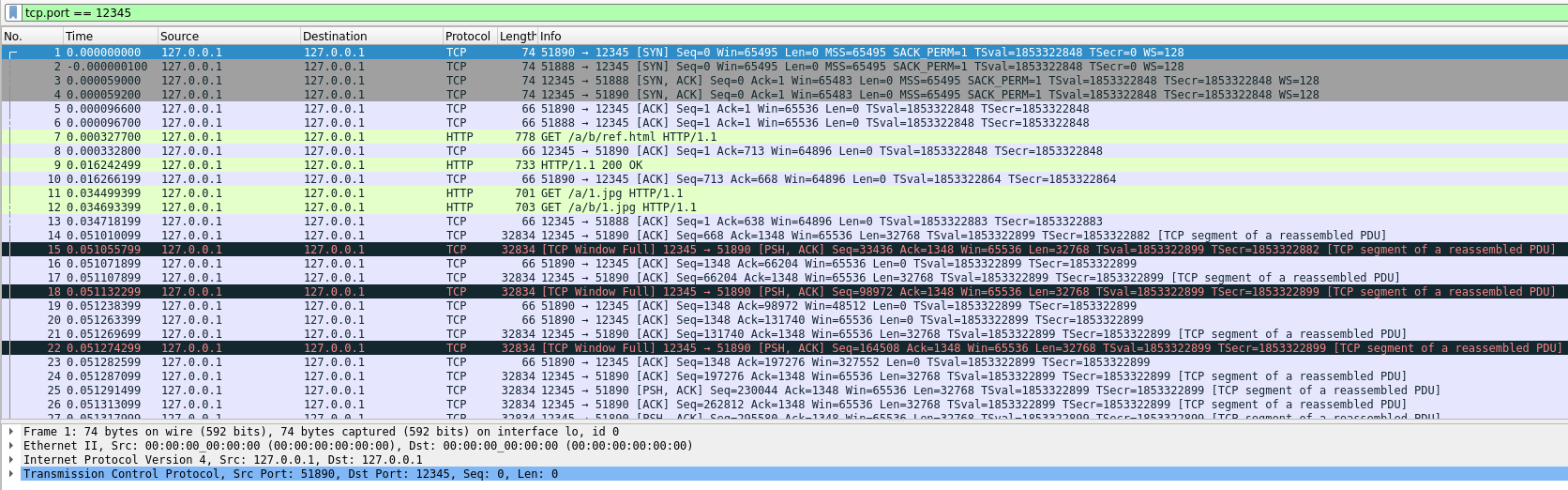




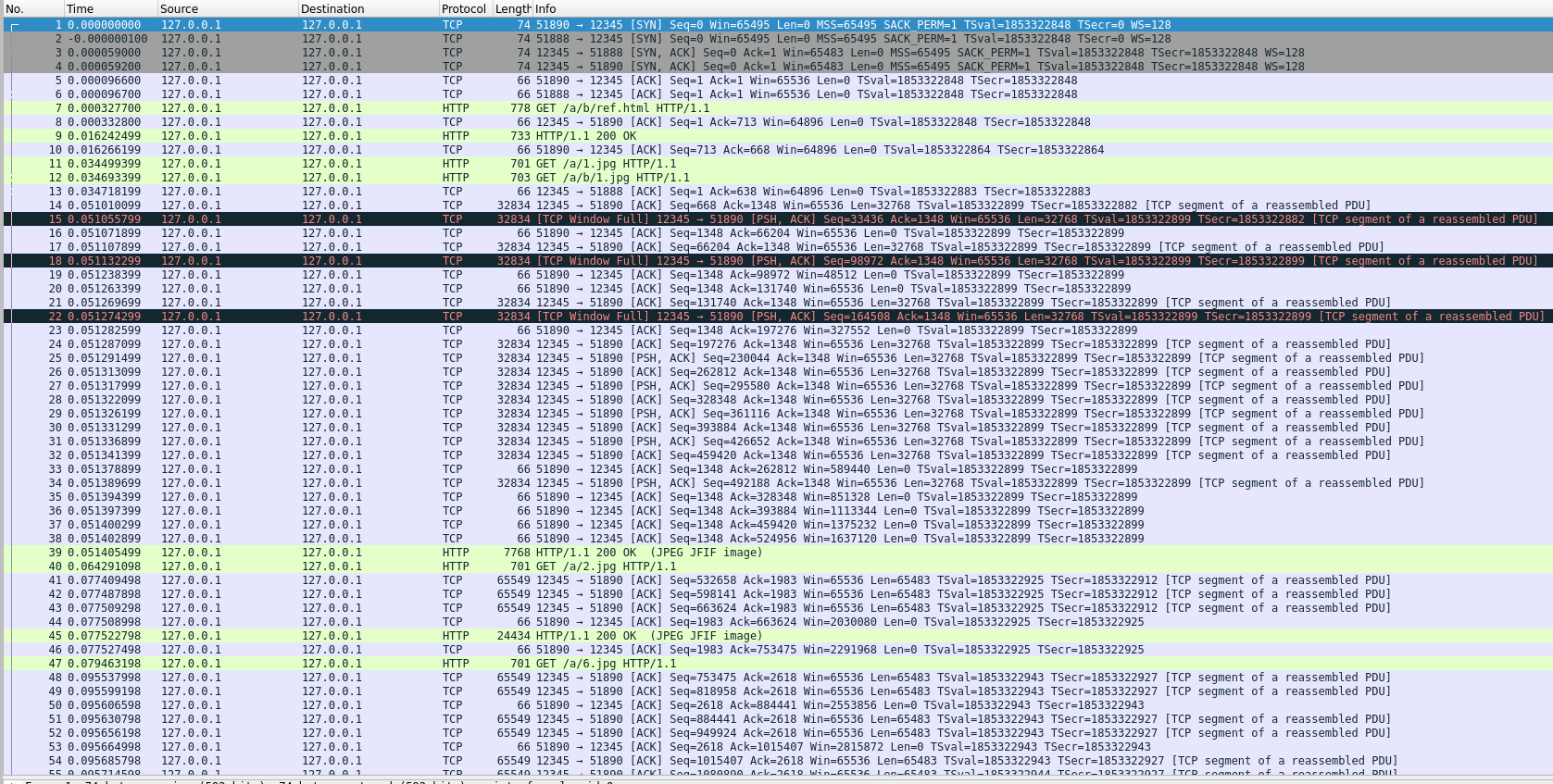
ניתן לראות כי יש כאן חיבור יחיד שבו הדפדפן שולח בקשה לשרת לקבל את הקובץ 5.jpg שהוא קובץ של תמונת עוגת וניל טעימה וקריספית. מפני שתוכן התמונה גדול (תמונה שוקלת יותר מסתם מחרוזת) כמו בבקשות אחרות שנראה פה השרת מחלק את החבילה לסגמנטים, הוא לא יכול לשלוח את כל התמונה בבת אחת כי גודלה הוא גדול יותר מהMSS ולכן הוא מחלק אותה לחתיכות ושולח אותן אחת-אחת. הלקוח מקבל את כל אחד מהחלקים האלה, מחזיר ACK עליהם ואחר כך התקשורת נסגרת משני הצדדים ע"י FIN הדדי.

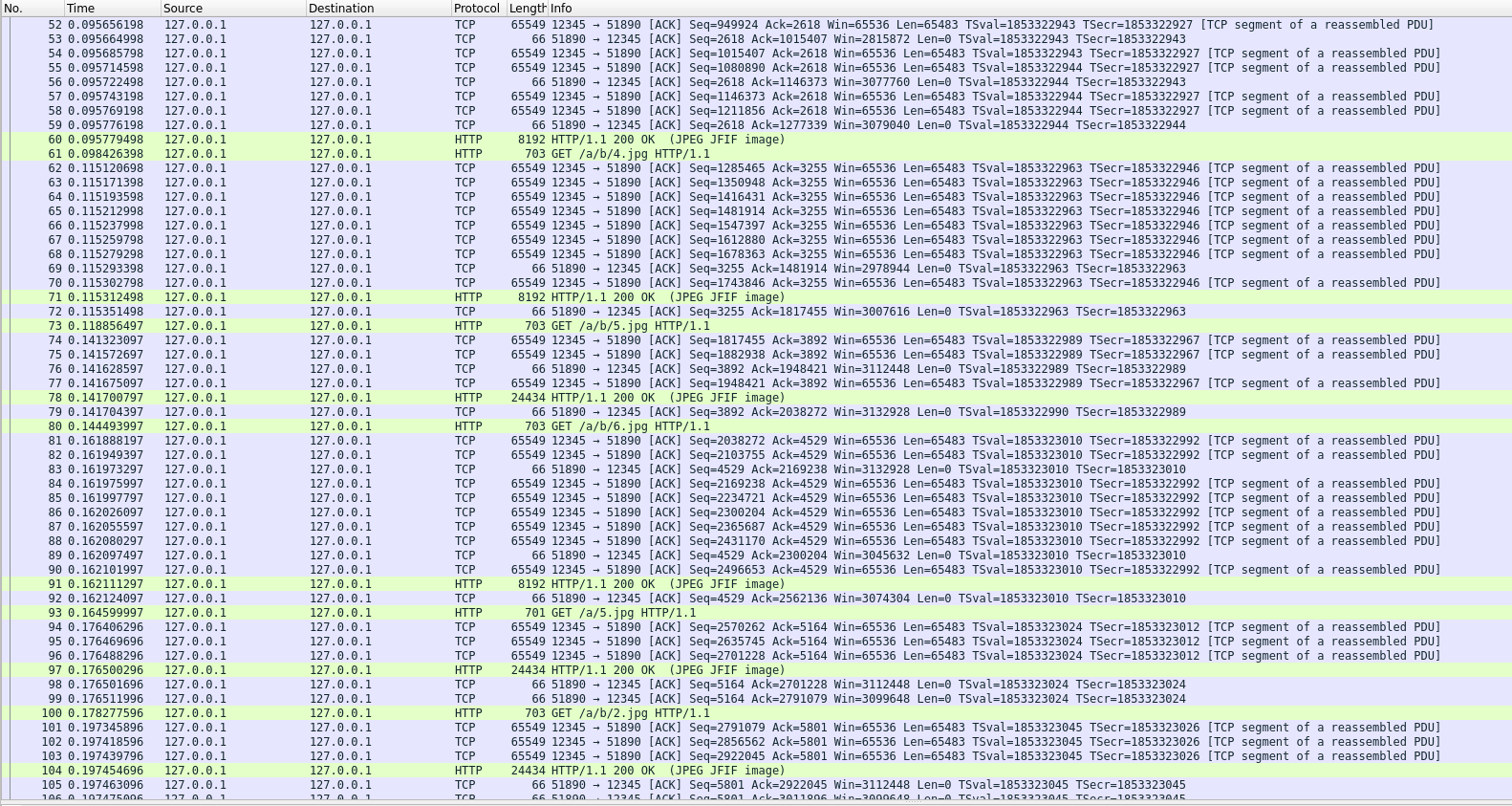
הרצה של /a/b/ref.html :

הכל



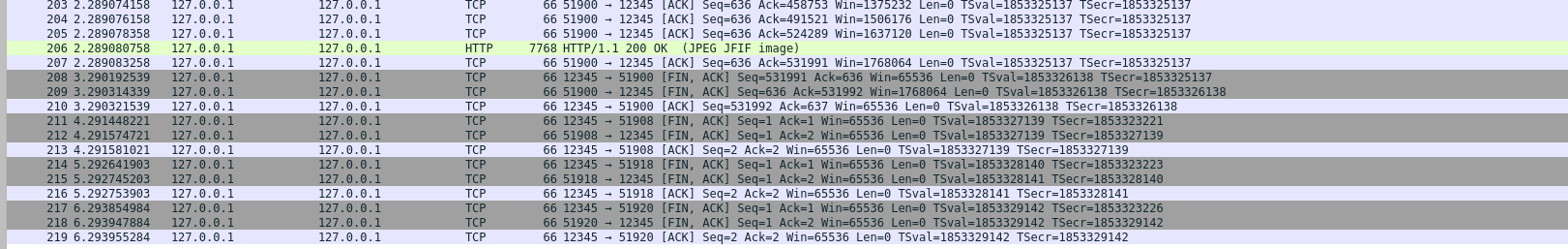
פקטה







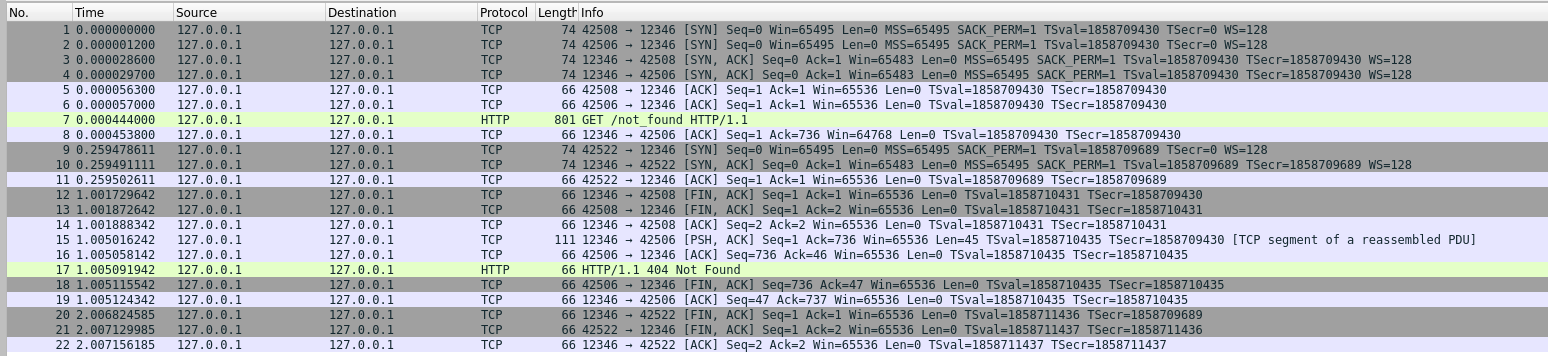




כפי שניתן לראות מאורך הבקשה, שהדפדפן שלנו (הלקוח) פונה לשרת בבקשה לקבל את הקובץ a/b/ref.html שזה קובץ html שבתוכו יש ייחוסים לקבצים אחרים, לכן השרת מוריד את הקבצים הנוספים הללו דרך אותו החיבור ובמקביל הוא פותח חיבורים נוספים שאיתם הוא יכול להוריד במקביל את הקבצים כדי לחסוך בזמן. כלומר, הדפדפן שואף לפתוח מספר חיבורים במקביל מול השרת ולא רק אחד, כי הדבר חוסך זמן. לכן, במקרה זה השרת פתח מספר חיבורים שבהם הוא טיפל אחד אחרי השני, כך שכל חיבור מבקש את התמונה המתאימה מהשרת והוא בתגובה מחזיר את תוכן התמונה או הקובץ המבוקש. השורות השורות שכתוב בהן Window Full באדום אומר שהחלון TCP שלמדנו עליו בהרצאה מלא ולכן הוא לא ימשיך לשלוח בקשות עד שהוא יקבל Ack ויתפנה מקום בחלון.

הרצה של /not\_found :

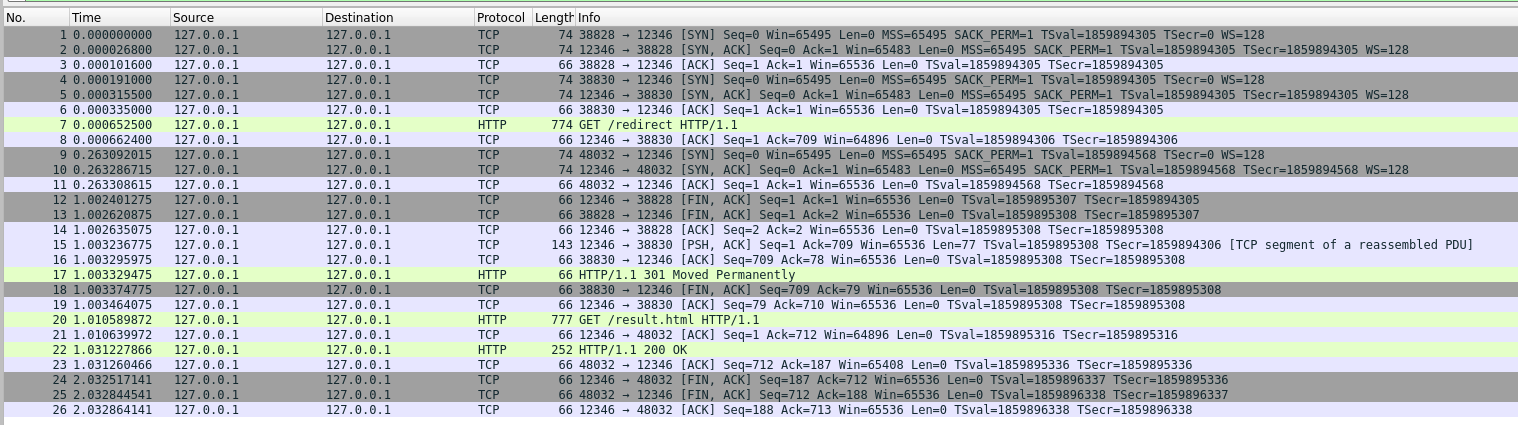
פקטה



בבקשה זו ביקשנו מהשרת קובץ שלא קיים, ולכן ציפינו לקבל בתור תשובה את 404 Not Found. הלקוח ביקש בקשה של קובץ not\_found שלא קיים בצד השרת, שכתוצאה מכך השרת ראה שאין בצד שלו קובץ כזה ולכן החזיר את הודעת השגיאה 404. לאחר מכן, הלקוח לא ביקש בקשות נוספות ולכן החיבור הסתיים.

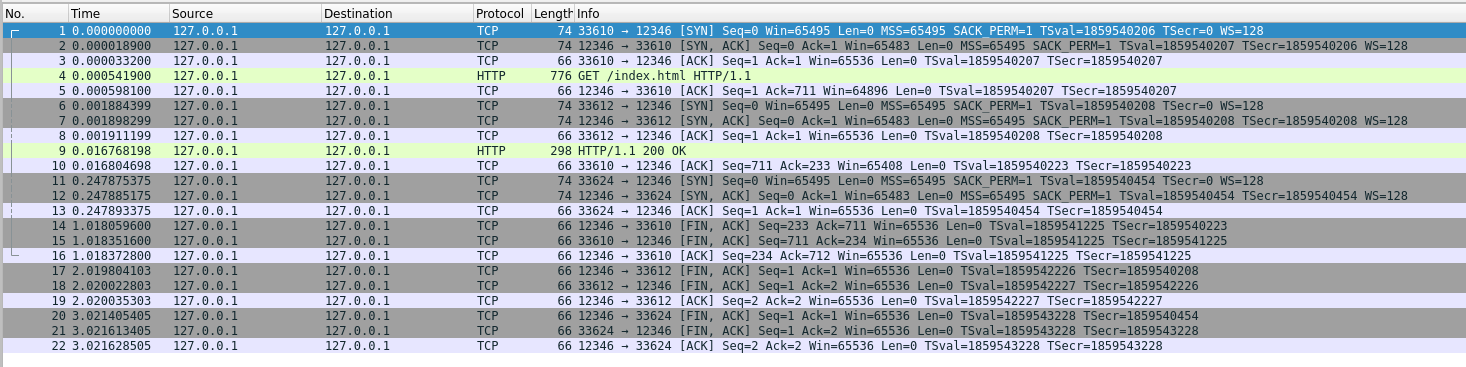
הרצה של /redirect :

פקטה



כפי שניתן לראות בבקשה הזו מתבצעים שני חיבורים עם השרת, מתבצעות שתי בקשות לשרת ומתקבלות שתי תשובות מהשרת ביחד עם שתי בקשות FIN. בחיבור הראשון הלקוח (דפדפן הכרום) מבקש בקשת Get של redirect, עבורה מוחזרת התשובה 301 Moved Permanently. ביחד עם אותה תשובה הוא מוסיף את סימן הFIN כדי להגיד ללקוח שהוא סיים את העבודה מהצד שלו ורוצה לסיים את התקשורת. הלקוח בתגובה מסיים את התקשורת ומיד לאחר מכן מבקש את הקובץ result.html ולכן תהליך החיבור עם השרת מתבצע מחדש והשרת שולח ללקוח את הקובץ שהוא ביקש. אנו רואים שערך ה connection היה keep-alive מפני שבבקשה השניה השרת ממשיך להאזין ורק לאחר שעוברת שניה (ניתן לראות בזמן קפיצה של מ1 שניות ל2 שניות) שולח בקשת FIN ללקוח שנענית בבקשת FIN חוזרת והחיבור נסגר.

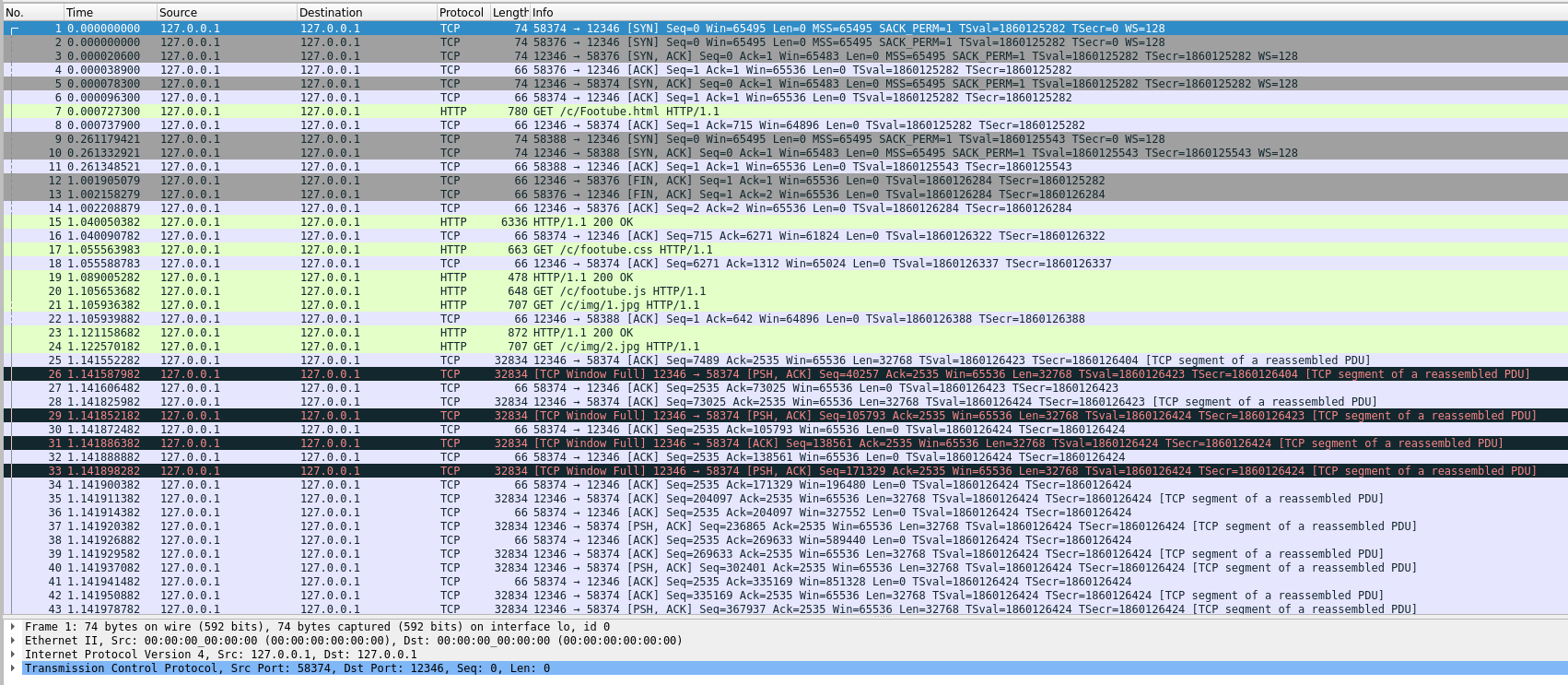
הרצה של /index.html :

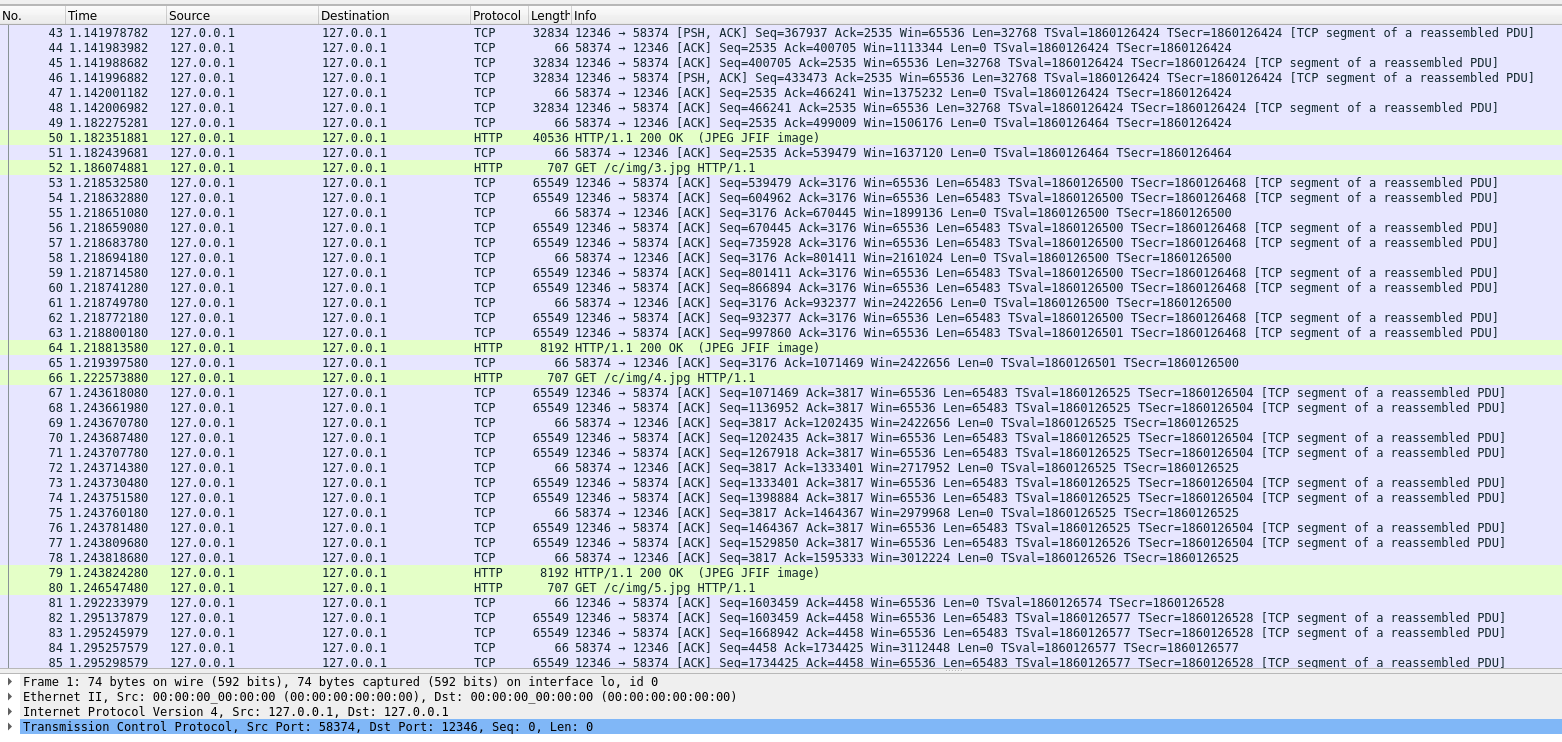


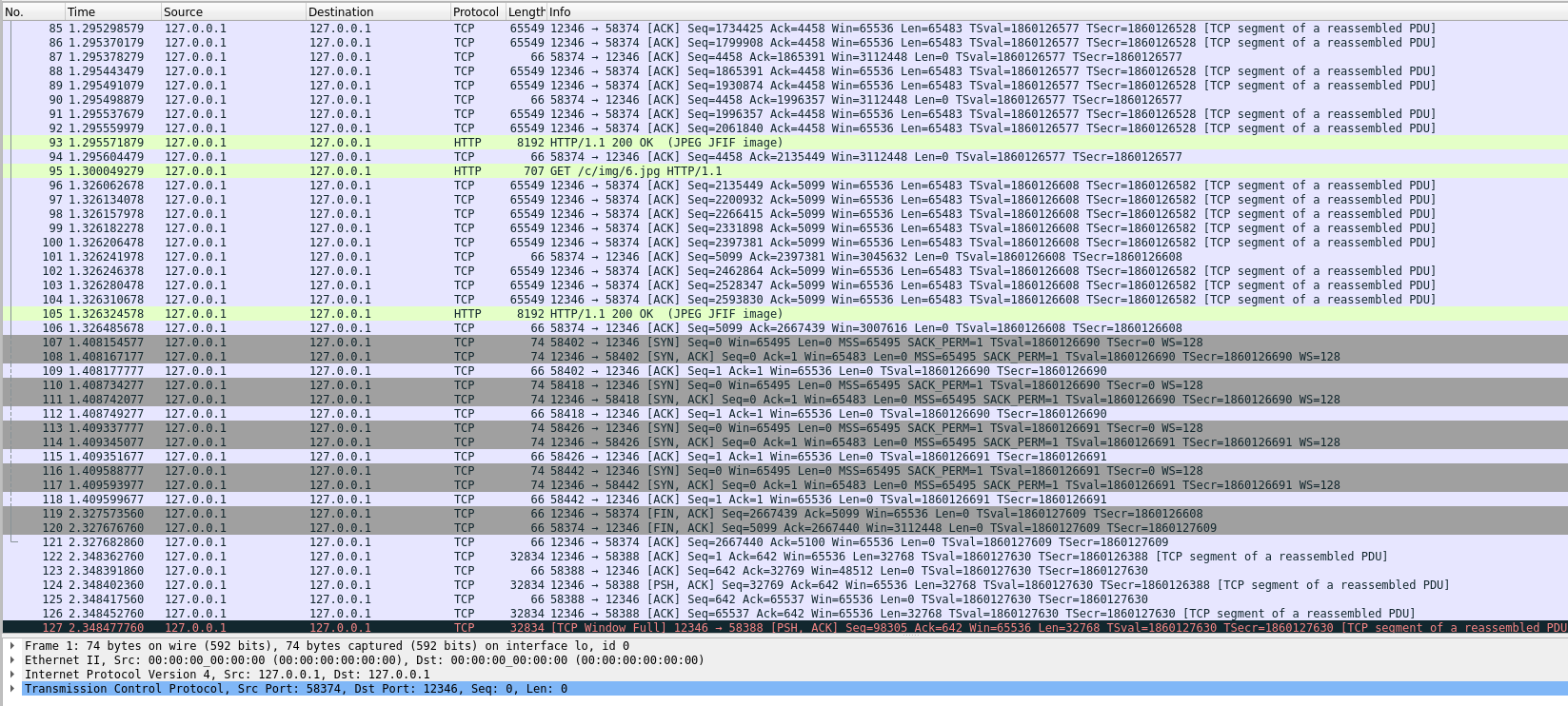
הלקוח שולח בקשת GET לקובץ בשם index.html, מתבצע חיבור בין הלקוח לשרת והשרת מחזיר ללקוח (הדפדפן של כרום) את הקובץ אותו ביקש. כמו במקרה דומה שפירטנו פה, השרת שולח לדפדפן בקשת FIN באופן מפורש מפני שהדפדפן הכיל בבקשתו את השדה connection: keep-alive מה שגרם לשרת להשאר מחובר ורק לאחר שניה שלא קרה בה כלום השרת שלח בקשת FIN מפורשת ללקוח, הלקוח שולח בקשת FIN בחזרה והחיבור נסגר.

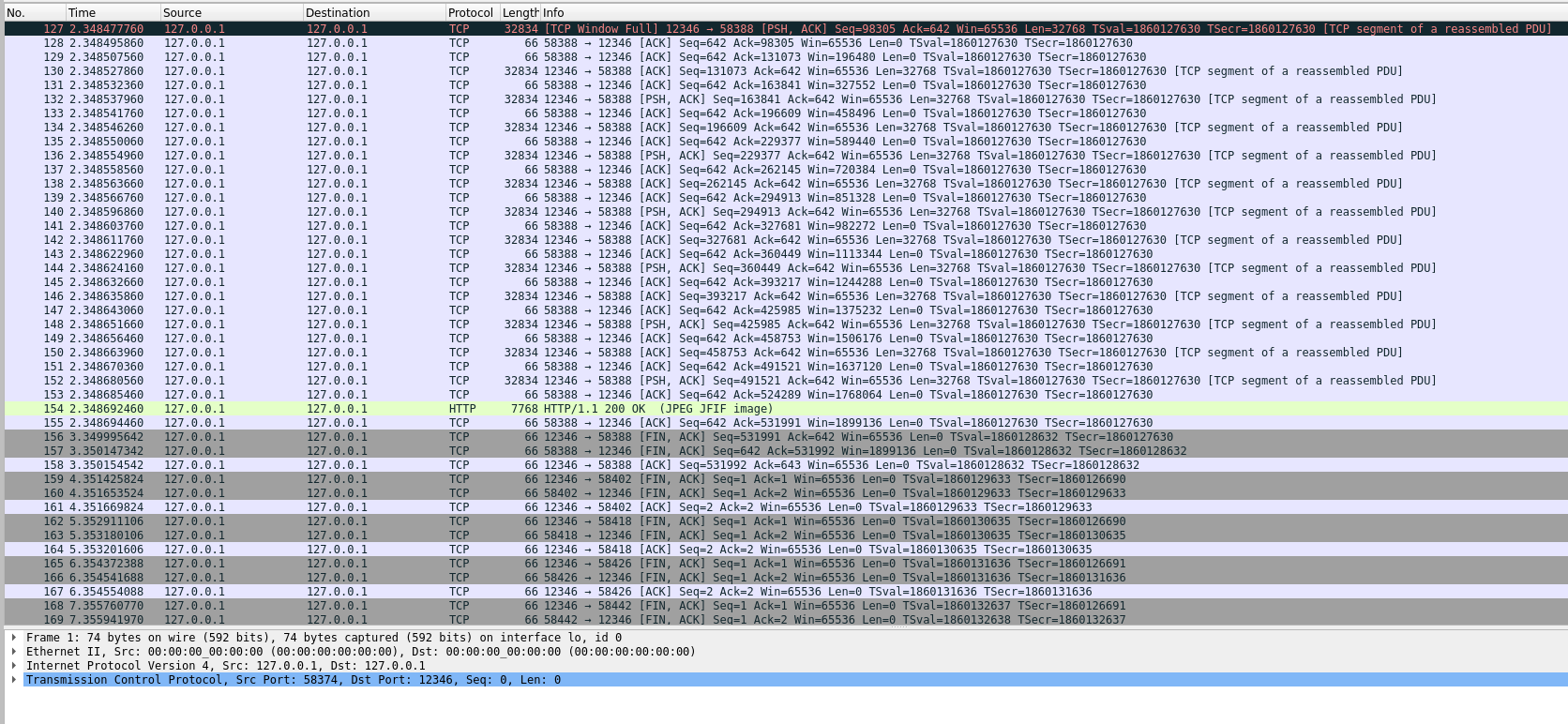
הרצה של /c/Footube.html :

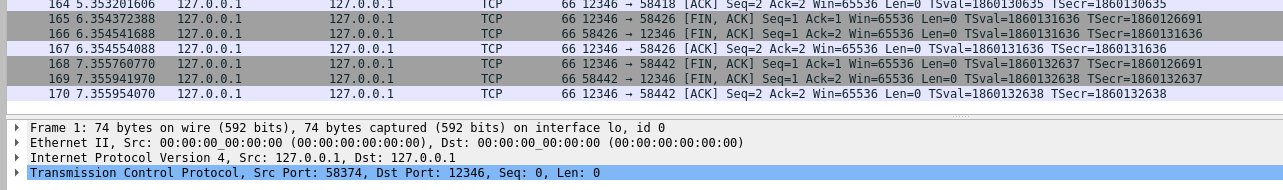
פקטה











גם במקרה הזה אנו רואים שיש חיבורים רבים לשרת מפני שבתוך דף ה html שביקשנו יש בקשות רבות לקבצים אחרים כגון תמונות, קבצי CSS JS ועוד, לכן באופן דומה לקובץ html האחר שפתחנו השרת פותח מספר חיבורים במקביל כדי לחסוך בזמן שלוקח לו להחזיר את התשובה למשתמש ובכל בקשה כזו הוא מעביר ללקוח את אחד הקבצים הקשורים לקובץ html. ניתן לראות שהTCP Window מתמלא (אפילו כמה פעמים) ולכן לא ממשיך לקבל בקשות עד לקבל ACK שלאחריה הוא ממשיך. עבור כל חבילה כזו שנשלחת ניתן לראות את הACKים העוברים מהלקוח כדי להודיע לשרת שהקבצים הגיעו. אפשר לראות גם פאקטות שמתחלקות לחלקים מפני שהן גדולות מהMSS (משהו ששכיח בהעברת תמונות שהן בדרך כלל כבדות, ויותר כבדות ממחרוזות ומקבצי טקסט) ולכן הן נשלחות בחלקים ולא בבת אחת. לאחר שכל הקבצים עברו ניתן לראות שליחת FINים הדדיים בין השרת ללקוח בכל החיבורים והחיבורים מסתיימים.