רשתות תקשורת מחשבים

תרגיל 7

הגשה בזוגות בלבד

משימה ראשונה: TCP והעברת קובץ

מקורות:

- RFC 793 – Transmission Control Protocol
- RFC 879 – TCP Maximum Segment Size and Related Topics
- RFC 1122 – Requirements for Internet Hosts - Communication Layers
- RFC 2018 – TCP Selective Acknowledgment Options
- RFC 2581 – TCP Congestion Control
- RFC 3390 – Increasing TCP's Initial Window

יש לזכור כי מרבית הדפדפנים מבצעים cache של אובייקטים. כאשר אנו מניחים כי נעשה שימוש חוזר בנתונים קיימים אנו נוטים לשמור את הנתונים הללו לשימוש עתידי, פעולה זו נקראת caching. כיוון שבזמן הגלישה באינטרנט אנו חוזרים לאותו אתר לא מעט פעמים, הדפדפן שלנו שומר את התוכן של האתרים אליהם נכנסו לאחרונה על מנת לקצר את הזמן עד להצגת האתר. במקרה שהדפדפן ימצא את הדף המבוקש ב-cache שלו הוא לא ישלח בקשה אל השרת אלא יציג את הדף מתוך ה-cache.

לפני ביצוע כל (!) ניסוי, וודא שהמטמון (Cache) של הדפדפן שלך ריק. כדי להסיר קבצים מתוך מטמון הדפדפן, בצע את הפעולות הבאות:

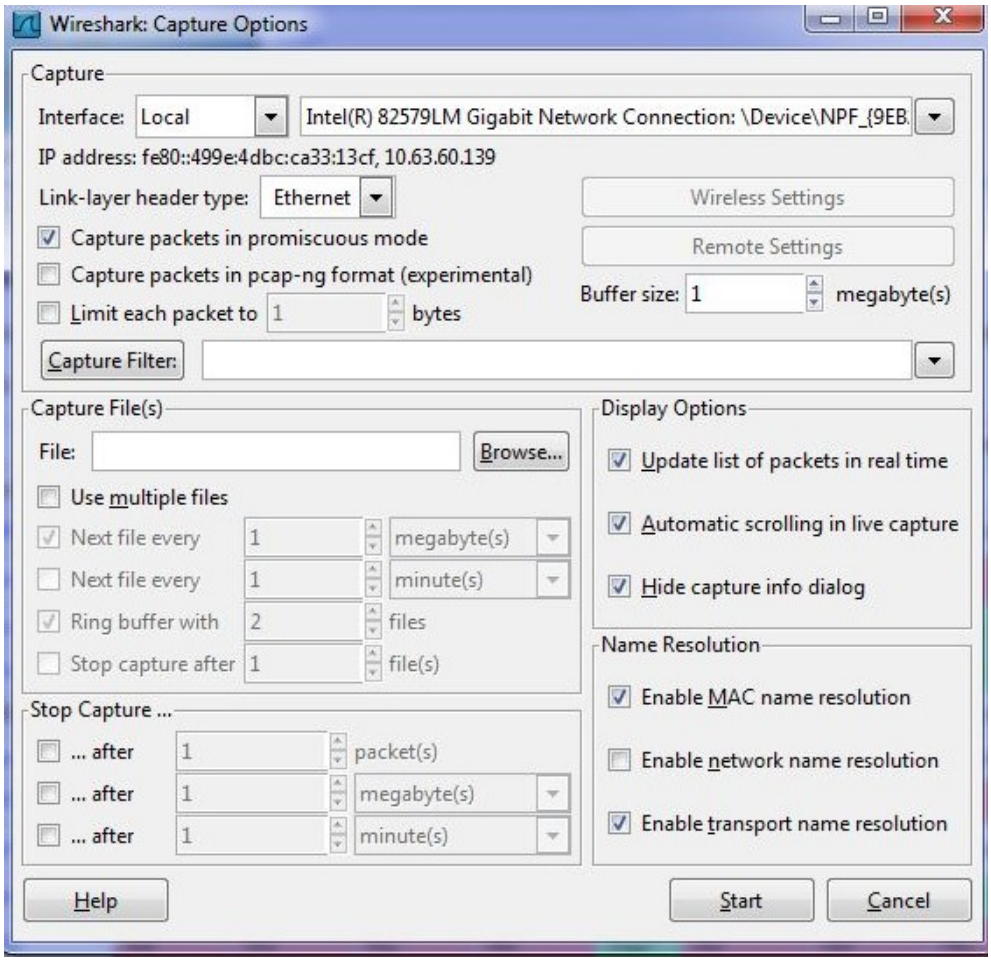
עבור Mozilla Firefox, בחר Tools => Clear Private Data.

עבור Internet Explorer, בחר Tools => Internet Options => Delete File, Clear History, Delete Cookies.

עבור Google Chrome, בחר Customize and control Google Chrome => Options => Under the Hood => Clear browsing data… and clear browsing history, empty the cache, delete cookies and other site data.

הרצה

צור capture filter אשר יגרום לתוכנה ללכוד רק תעבורת רשת המשתמשת בפרוטוקול TCP ובפורט ברירת המחדל של HTTP.



פתח את הדפדפן שלך והיכנס לכתובת: http://www.ietf.org/rfc/rfc793.txt.

הדפדפן שלך אמור להראות את RFC 793 בפורמט ASCII.

כאשר הדף נטען הפסק את הלכידה ב-wireshark. רשימת החבילות שנלכדו אמורה להיראות כך:

1	0.000000	130.230.52.139	64.170.98.32	TCP	3324 > http [SYN]
2	0.176868	64.170.98.32	130.230.52.139	TCP	http > 3324 [SYN,
3	0.176949	130.230.52.139	64.170.98.32	TCP	3324 > http [ACK]
4	0.177163	130.230.52.139	64.170.98.32	HTTP	GET /rfc/rfc793.t
5	0.353977	64.170.98.32	130.230.52.139	TCP	http > 3324 [ACK]
6	0.355493	64.170.98.32	130.230.52.139	TCP	[TCP segment of a
7	0.355622	64.170.98.32	130.230.52.139	TCP	[TCP segment of a
8	0.355655	130.230.52.139	64.170.98.32	TCP	3324 > http [ACK]
9	0.533411	64.170.98.32	130.230.52.139	TCP	[TCP segment of a
10	0.533529	130.230.52.139	64.170.98.32	TCP	3324 > http [ACK]
11	0.533560	64.170.98.32	130.230.52.139	TCP	[TCP segment of a
12	0.533708	64.170.98.32	130.230.52.139	TCP	[TCP segment of a
13	0.533755	130.230.52.139	64.170.98.32	TCP	3324 > http [ACK]
14	0.711525	64.170.98.32	130.230.52.139	TCP	[TCP segment of a

ניתוח

ענה על השאלות הבאות תוך שימוש ב-TRACE הנתון (tcp.pcap).

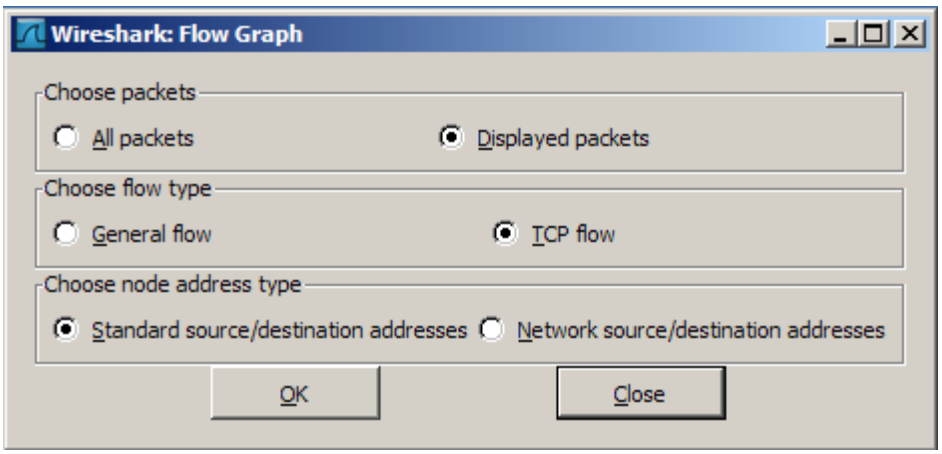
1. כדי להבין את המשמעות של HTTP, נתחיל במשימה פשוטה. על מנת לראות את החבילות כמו ששכבת האפליקציה רואה אותם, עליך לבחור חבילת TCP ואז לבחור "**TCP Stream Follow**". העתק את הודעת HTTP GET מהחלון שנפתח והדבק אותם בדו"ח.
2. אילו OPTIONS בשימוש ע"י המחשב השולח ואילו ע"י השרת?

   רמז: חקור את 2 חבילות ה-TCP הראשונות.
3. מהו ה-Maximum Segment Size (MSS) של המחשב השולח ומה של השרת?

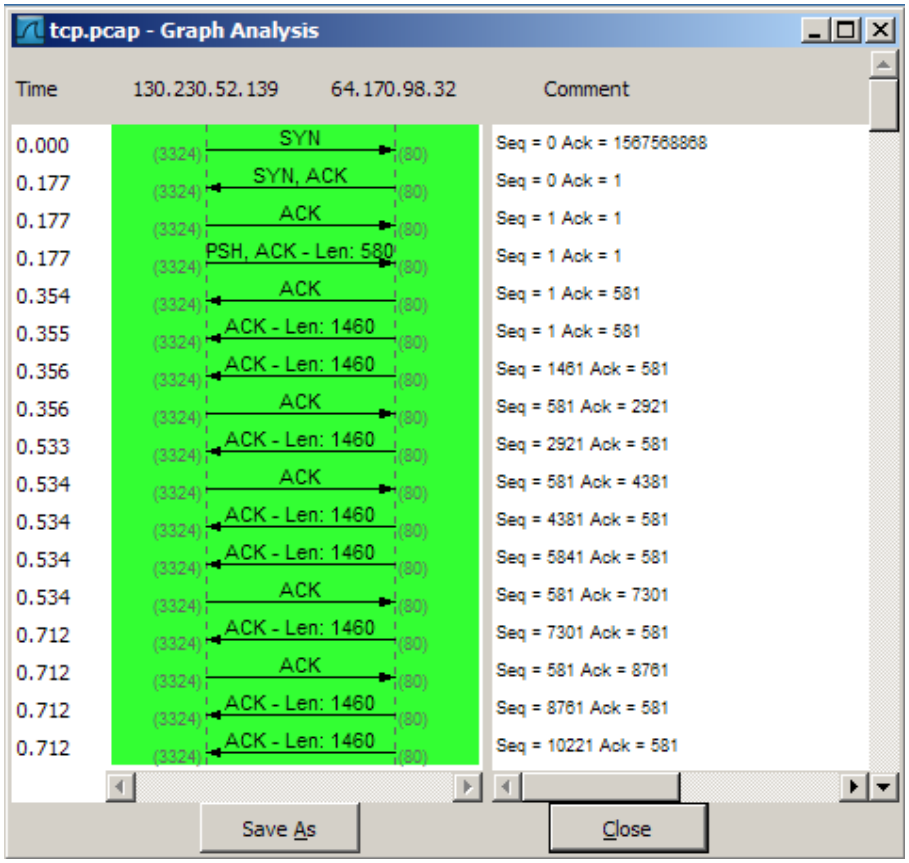
   רמז: MSS הוא הגודל המקסימאלי של הנתונים שיכולים להיכנס לסגמנט של TCP.
4. מהו החלון ההתחלתי (נקרא IW כקיצור של INITIAL WINDOW: size of the cwnd after the three- handshake is completed) של השרת?

רמז: ה-IW הוא גודל חלון ה-congestion של השולח ב-RTT הראשון של שלב העברת הנתונים בהתאם ללחיצת היד המשולשת של TCP והחלפת הודעות request/reply של HTTP.

כדי לזהות את ה IW (כלומר, מספר מנות נתונים שנשלחות ב-RTT הראשון של שלב העברת הנתונים), בחר חבילת TCP כלשהי ברשימת החבילות של חיבור ה-TCP, בחר Flow Graph מהתפריט Statistics, ציין את סוג הזרימה TCP Flow, ולאחר מכן לחץ על OK. נסה לגלות את ה RTT מתוך הגרף שמתקבל.



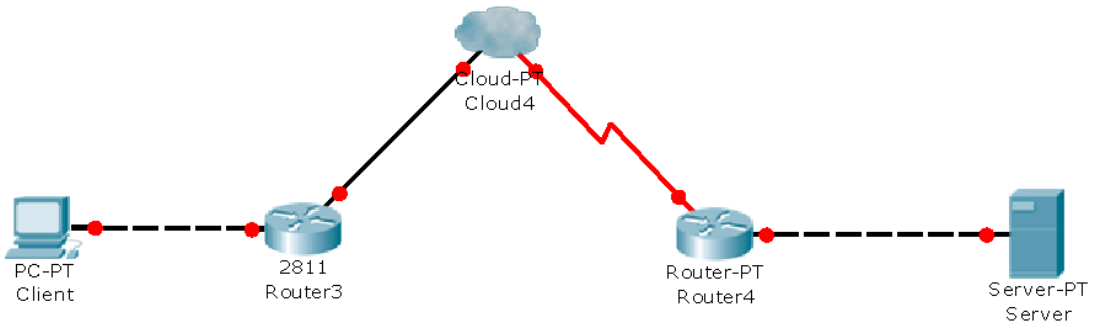
אמור להתקבל משהו כמו זה:



5. חקור את כמות המידע שנשלחת מהשרת למחשב שלך לכל יחידת זמן. הוסף את הדיאגרמה לדו"ח והסבר אותה.

רמז: בחלונית "Packet List" בחר חבילת TCP המצוינת כ-" TCP segment of a reassembled PDU". בחר בתפריט **Statistics** את הפריט **TCP Stream Graph** ובחר **Time-Sequence Graph (Stevens)**.

6. חקור את ה-HEADER של TCP:
   א. איזה דגל של TCP נשאר תמיד דלוק?
   ב. אילו שדות לא משתנים ומדוע? מה התפקיד של כל שדה?
   ג. אילו שדות משתנים ומדוע? מה התפקיד של כל שדה?
7. עיין ב-TRACE (TCP-CONNECT-wrong.pcap). ישנן שתי שיחות TCP (שיחה ראשונה חבילות 1 – 3, שיחה שניה חבילות 15 - 17). הסבר מה לא תקין בתהליך יצירת הקשר בכל אחת מהשיחות. תקן את התקלה. (הסבר במדויק מה צריך לשנות בחבילות על מנת שיצירת הקשר תהיה ע"פ התקן).
8.



בציור לעיל ישנו לקוח (Client) השולח בקשה אל השרת (Server). הבקשה אכן מגיעה אל השרת והוא עונה לבקשה והיא אכן מגיעה אל כרטיס הרשת של הלקוח. אבל בתוכנה שעל המחשב של הלקוח מתקבלת הודעה כי לא התקבלה תשובה מהשרת. (התוכנה תקינה לחלוטין ואין בה תקלות תיכנות!)

עיין ב-Trace המצורף (TCP-PUSH-only.pcap) והסבר מדוע מתקבלת הודעה זו אצל הלקוח.

רמז: שים לב לשתי ההודעות אשר השרת שולח כתשובה לבקשה של הלקוח, באיזה סדר הלקוח מקבל את ההודעות?.

9. PACKET ETHERNET ב-HEX:

00 30 1a 00 07 b4 00 01 02 00 24 7e 03 b5 85 08 00 45 00
03 f9 06 39 2f b1 0c ed 40 00 80 06 8f 17 0a 3f 38 4d 0a 3f
ff ff 1a 4f 00 00 02 04 05 b4 01 1c d6 06 00 00 00 00 70 02
01 04 02

מה פורט המקור ומה פורט היעד? (שים לב: החבילה מציגה את הנתונים ב-Hex, Ethernet header מכיל 14 בתים ו- IP header מכיל 20 בתים)

10. **שאלת הבנה כללית:**

מדוע שיחת ה-TCP מתעכבת בזמן שניתן לראות כי UDP ממשיך לשלוח נתונים?

11. ב-TRACE הנתון (TCP-retransmition.pcap) ישנה החלפת נתונים בין שני מחשבים: אחד בעל כתובת IP - 84.229.173.71 והשני בעל כתובת IP - 109.64.17.214. הכן פילטר תצוגה שיציג רק את הנתונים העוברים בין שני מחשבים אלו.

12. מתי התחיל תהליך congestion avoidance? כיצד זיהית זאת?

13. ישנו שידור מחדש של נתונים. באיזה מספר packet זה קורה? מה גרם לשידור מחדש?

משימה שניה: חישוב checksum

קריאה בלבד

נחקור את חישובי ה-checksum של UDP ו-IP בחבילות DNS. הסבר מפורט על חישוב CHECKSUM נמצא ב-RFC 1071.

הדרך לחשב checksum

פרוטוקולים שונים של TCP\IP משתמשים בביטים מסוימים על מנת לבדוק שגיאות. ב-IP לדוגמא, ה-checksum מחושב עבור התוכן של ה-header בלבד וכלול בשדה מיוחד. מכיון שה-checksum צריך להיות מחושב בכל נתב, האלגוריתם נבחר בשל היותו קל למימוש על ידי תוכנה ולא משום שהוא הטוב ביותר על מנת לאתר שגיאות.

באופן כללי, האלגוריתם פשוט ביותר:

א. בהתחלה, שדה ה-checksum ממולא באפסים.

ב. כל זוג bytes מוצמד להיות מחושב בצורה של מספר בעל 16 ביטים.

ג. מחשבים סכום של modulo $(2^{16}-1)$ על ה-bytes.

ד. מחשבים השלמה ל-1 של הסכום וממקמים את המספר בשדה ה-checksum.

ה. לאחר מכן, כדי לאמת את ה-checksum מחשבים שוב modulo $(2^{16}-1)$ על אותה סדרה של bytes, כולל שדה ה-checksum . אם התוצאה היא סדרת אחדות, הבדיקה הצליחה.

דוגמא

נניח שיש לנו 4 בתים (מוצגים ב-HEXA): B3, 7A, B6, EB. בתים אלה מתחלקים ל-2 מילים של 16 ביטים, כך: B37A ו-B6EB (1011011011101011 , 1011001101111010). לכן, חישוב modulo $(2^{16}-1)$ יהיה:

$$\begin{array}{r} 1011001101111010 \\ \underline{1011011011101011} \\ 1\ 0110101001100101 \end{array}$$

בחישוב מודולו, כל יתרה של חישוב מתווספת בחזרה לכל הסכום. לכן נבצע את החישוב הבא:

$$\begin{array}{r} 0110101001100101 \\ \underline{0000000000000001} \\ 0110101001100110 \end{array}$$

ההשלמה ל-1 היא: 1001010110011001, או 9599 ב-HEX.

$$FFFF \equiv B37A + B6EB + 9599 \bmod FFFF$$

הבדלים בין הפרוטוקולים

Ipv4

מחשבים את ה-checksum על ה-header של חבילת ה-IP. החלק של הנתונים ייבדק בשכבות אחרות. אם נמצאה שגיאה, החבילה תיזרק. השולח ממלא את שדה ה-checksum באפסים ומריץ את האלגוריתם. נשים לב: כאשר נתב משנה את שדה ה-TTL (מוריד ב-1), עליו לחשב מחדש את ה-checksum וזאת הסיבה שהנתס צריך לחשב כל פעם.

Ipv6

שדה ה-checksum עבור ה-header הוסר על מנת להפחית את זמן עיבוד החבילות בנתב. חבילות שהועברו דרך שכבות תחתונות כמו Ethernet בדרך כלל כבר נבדקו. יתירה מזאת, השכבות העליונות, כמו UDP ו-TCP יבצעו אף הן בדיקה משלהן.

UDP

ה-checksum בודק שגיאות ב-datagram (כולל את חלק הנתונים), אך הוא אופציונאלי. אם השולח לא מעוניין לבצע בדיקה, על שדה ה-checksum להכיל אפסים. במידה והשולח חישב את ה-checksum ויצא שהכל אפסים, הוא ימלא הכל באחדות (ייצוג נוסף של 0). חישוב ה-checksum זהה ל-IP למעט 2 היבטים. ראשית, אם אורכו של ה-DataGram אינו כפולה של 16 ביטים, הוא ירופד באפסים. (הריפוד הינו רק עבור החישוב ולא יישלח כנתונים). שנית, UDP מוסיף לשם החישוב בלבד pseudo header לתחילת ה-datagram. ה-pseudo header נועד לוודא שה-datagram מגיע למחשב ול-port הנכונים עם אורך נכון. לבסוף, אם ה-datagram יימצא פגום, החבילה תיזרק והמקור לא יידע על כך.

מבנה ה-pseudo header :

Source Address		
Destination Address		
00000000	Protocol	Datagram Length

TCP

ה-checksum בודק שגיאות ב-segment (כולל את חלק הנתונים), והשימוש בו הוא חובה. החישוב וה-pseudo header זהים ל-UDP.

משימה שלישית: בדיקת zero-window של TCP

במשימה זו נחקור את המימוש של TCP עבור בקרת הזרימה, בעזרת ניתוח של traces דגומים.

מקורות:

RFC 793 – Transmission Control Protocol

RFC 1122 – Requirements for Internet Hosts - Communication Layers

RFC 2988 – Computing TCP's Retransmission Timer

בצד המקבל, TCP משתמש בשדה window בכל ACK כדי לדווח לצד השולח כמה נתונים הוא מצפה לקבל. אם באופן זמני למקבל אין מספיק מקום ב-buffer, הוא שולח ACK עם ערך 0 בשדה window. כאשר מתפנה מקום, המקבל שולח ACK נוסף עם ערך מתאים. מכיון ש-ACK זה יכול להיאבד, החיבור עלול להתנתק לנצח. TCP (RFC 1122, RFC 793) דורש ממחשב שמקבל ערך 0 בשדה window, שכל עוד הוא לא מקבל ערך שונה מאפס כעבור זמן מסוים, ישלח סגמנט keep-alive, שדורש לדעת מהו גודל החלון בצד המקבל. על סגמנט זה נדרש המקבל להשיב ב-ACK ולאשר שהחיבור עדיין קיים. אם החיבור התנתק והמחשב המרוחק עדיין חי, יישלח RST במקום ACK.

לפי RFC 1122, מימוש יכול לשלוח סגמנט keep-alive ללא נתונים (נקרא גם zero-window probe). עם זאת, הוא יכול גם להיות מוגדר כך שישלח בית אחד של זבל. בנוסף לכך, השולח צריך להגדיל את מרווחי הזמן בין שליחות עוקבות של keep-alive בצורה מעריכית, כמו שנעשה ב-retransmission.

משימה

פתח את הקובץ zero.pcap וענה על השאלות הבאות:

בקובץ זה ניתן לראות שיחת TCP בה המשתמש מוריד קובץ מאתר אינטרנט.

1. מהו שם הקובץ המורד?

2. מהו הגודל המקסימאלי של החלון של הלקוח?

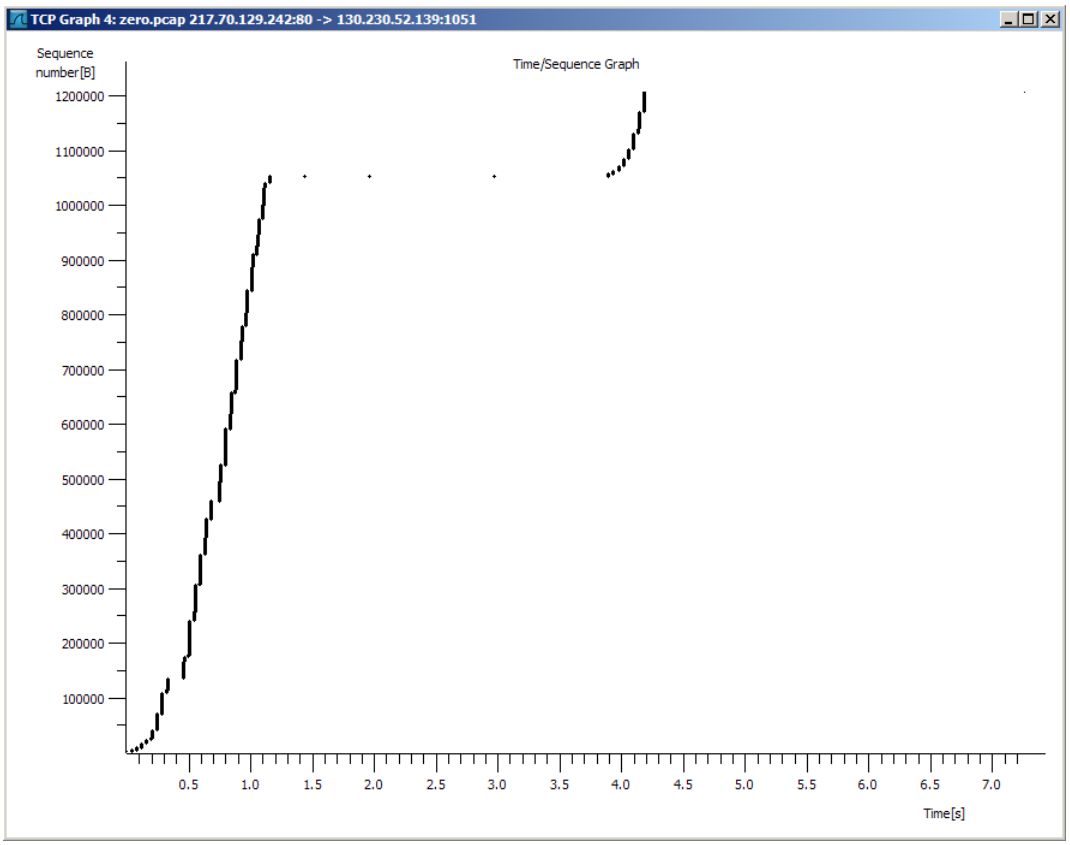
מהו גודל החלון של השרת בתחילת השיחה?

רמז: חקור את הפרטים של ה-packets הראשונים של TCP.

3. בחלונית "Packet List", בחר בחבילה המסומנת כ-"TCP segment of a reassembled PDU".

לך לתפריט **Statistics**, בחר ב-**TCP Stream Graph**, ובחר **Time-Sequence Graph (STEVENS)**.

החלון הבא יופיע:



הסבר מה שאתה רואה. התייחס לתופעה המתרחשת בנקודת זמן 1.2 שניות וב-3.9 שניות.

האם השינוי הוא בצד הלקוח או בצד השרת?

רמז: בחלונית "Packet List", חקור את ה-packets במרווחי הזמן שמתחילים מ-1.2 שניות.

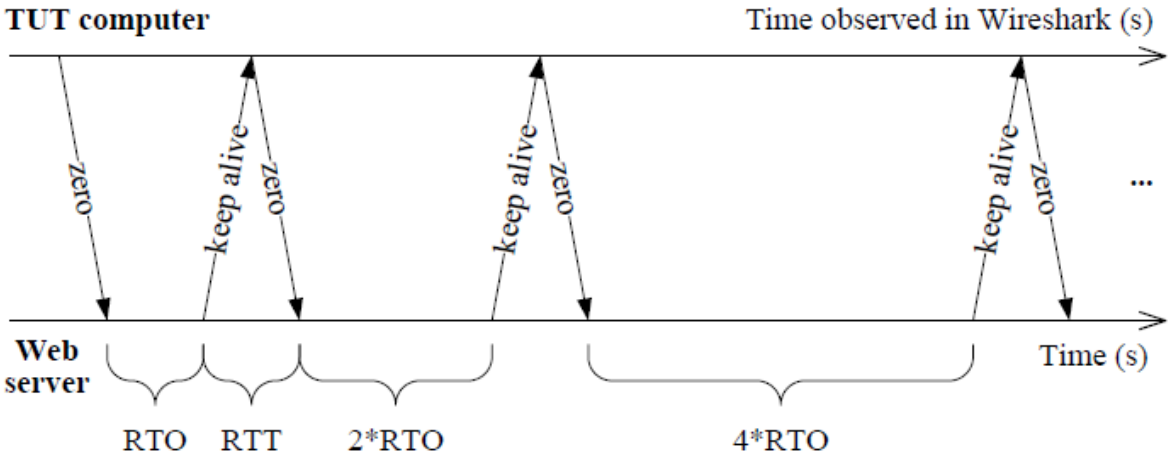
4. חקור את הפרטים של מסגרת 1214.

כמה בתים של נתונים מכיל סגמנט זה של keep-alive? מהם בתים אלו? האם זה מתאים למפרט של TCP (RFC 1122)?

רמז: שים לב, wireshark מפרש בטעות בתים אלו כ-trailer של Ethernet.

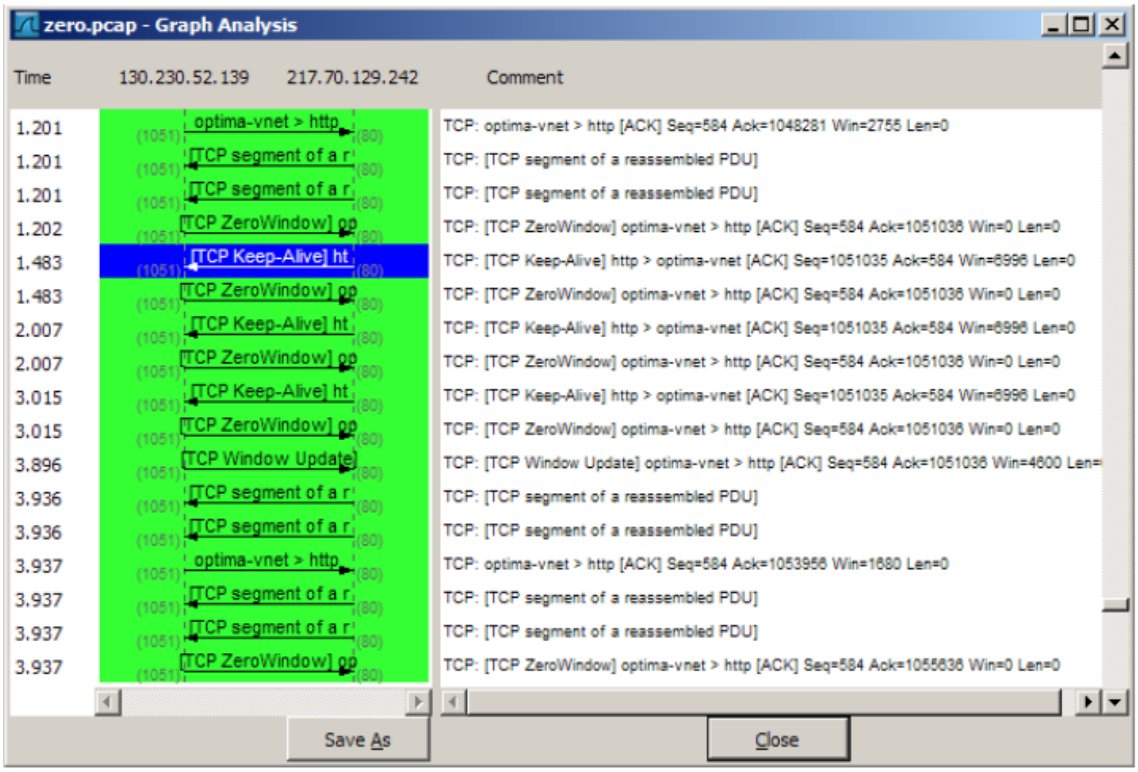
5. כפי שנובע מ-RFC 1122, על הצד השולח לשלוח את בדיקת keep-alive הראשונה שלו כאשר הנתון zero-window קיים כבר תקופה של RTO (Retransmission TimeOut period), ועליו להגדיל את מרווחי הזמן בין בדיקות עוקבות בצורה **מעריכית**. מהו ערך ה-RTO ההתחלתי של השרת?

רמז: כדי לענות על שאלה זו, עיין בשרטוט הבא. קל לראות שהמרווח בין הזמן שחבילת ה-Zero-Window הראשונה (מסגרת 1213) נשלחה לבין הזמן שחבילת ה-Keep-alive הראשונה (מסגרת 1214) הגיעה שווה ל-RTT+RTO שניות. למרות שה-RTT משתנה עם הזמן, ניתן למצוא את הערך שלו בצורה בטוחה מהחבילה הכי קרובה (מסגרת 1213). כדי לחשב זאת הפחת את הזמן של חבילה 1213 מהזמן של חבילה 1214. תוכל לבדוק אם הגעת לחישוב הנכון ע"י השוואה לתוצאה של wireshark. כדי להעזר ב wireshark, יש לבחור את מסגרת 1213 בחלונית "Packet List", בחלונית "Packet Details" יש להרחיב את העץ **Transmission Control Protocol** ולבחון את השדה **[SEQ/ACK analysis]**. כך אנו מקבלים: RTO = (1.483208 – 1.201527) – RTT.



Probing of zero window

```
⊞ Frame 1213 (54 bytes on wire, 54 bytes captured)
⊞ Ethernet II, Src: Notebook_d3:25:19 (00:06:1b:d3:25:19), Dst: All-HSRP-r
⊞ Internet Protocol, Src: 130.230.52.139 (130.230.52.139), Dst: 217.70.129.
⊟ Transmission Control Protocol, Src Port: optima-vnet (1051), Dst Port: ht
    Source port: optima-vnet (1051)
    Destination port: http (80)
    Sequence number: 584    (relative sequence number)
    Acknowledgement number: 1051036    (relative ack number)
    Header length: 20 bytes
  ⊞ Flags: 0x10 (ACK)
    Window size: 0
  ⊞ Checksum: 0xfce7 [correct]
  ⊟ [SEQ/ACK analysis]
      [This is an ACK to the segment in frame: 1212]
      [The RTT to ACK the segment was: 0.000035000 seconds]
    ⊟ [TCP Analysis Flags]
        [This is a ZeroWindow segment]
```



6. האם השרת מגדיל את המרווחים בין חבילות ה-Keep-alive בצורה מעריכית? ציין את נקודות הזמן הנצפות.