אוניברסיטת אריאל בשומרון, המחלקה למדעי המחשב
רשתות תקשורת, סמסטר ב' תשפ"ג

Congestion Control TCP

ליאור וינמן 213081763
יועד תמר 213451818

24 בדצמבר 2022

תוכן עניינים

1 פרק א' - תיאור המערכת

1.1 הקדמה

בפרק זה נעסוק בתיאור כלל המערכת שבנינו. נתאר אילו קבצים משתתפים בפרויקט, איך כל אחד מהם ממומש ומה הרציאונל והמשמעות מאחורי כל אחד מהם. בנוסף, נסביר כיצד להריץ את התוכנית.

1.2 קבצי הפרוייקט והרצתו

את הפרויקט יש להריץ על מערכת ההפעלה Ubuntu LTS גרסה 22.04 בלבד!
הפרויקט מכיל את הקבצים להן:

1. $Makefile$ - הקובץ היוצר של התוכנית.
2. $myLib.h$ - קובץ ספריה.
3. $Sender.c$ - שולח ההודעות.
4. $Receiver.c$ - מקבל ההודעות.
5. $msg.txt$ - ההודעה שאותה השולח ישלח למקבל.

על כל אחד מהקבצים נרחיב בהמשך(!) את כלל הקבצים האלו, יש להוריד ולשמור באותה התקיה.
ישנה אפשרות להוריד ישירות מה-Github , בקישור הנ"ל:

$$https://github.com/liorvi35/TCP\_CongestionControl\_C$$

לאחר ההורדה והשמירה של כלל קבצי הפרויקט, נפתח חלון טרמינל במיקום של התקיה שבה נמצאים כל הקבצים ונכתוב את הפקודה: "$make\ all$". לאחר שכתבנו את הפקודה הנ"ל יווצרו בתיקיה הקבצים הבאים:

1. $myLib.h.gch$ - קובץ ספריה מהודר.
2. $Sender.o$ - קובץ אובייקט בינארי של $Sender.c$, ז"א זהו הקובץ המתקבל לאחר הידור שלו.
3. $Receiver.o$ - קובץ אובייקט בינארי של $Receiver.c$.
4. $Sender$ - קובץ הרצת התוכנית של $Sender.c$, ז"א ההרצה מתבצעת על ידי הקובץ הנ"ל.
5. $Receiver$ - קובץ הרצת התוכנית של $Receiver.c$.

כעת לאחר שסיימנו את תהליך היצירה, נפתח שני חלונות טרמינל על מיקום התקייה עם כלל הקבצים (ניתן להשתמש בטרמינל שבו יצרנו את הקבצים, אך רצוי קודם לנקות אותו עם הפקודה "$clear$"). בטרמינל הראשון נכתוב את הפקודה "$./Receiver$" ובטרמינל השני נכתוב את הפקודה "$./Sender$" (סדר הפקודות קריטי, כיוון שעלינו קודם כל להפעיל את השרת - מקבל ההודעות לפני שניתן להפעיל את הלקוח - שולח ההודעות).

לאחר שהרצנו את שתי הפקודות האלו, בחלון אחד רצה התוכנית $Receiver.c$ ובחלון השני רצה התוכנית $Sender.c$ (בשתיהן נפתח ממשק המשתמש וניתן להכניס קלט ולקבל פלט בהתאם).
נחזור בקצרה על הדברים שנאמרו ונסכם:

- קבצי הפרויקט הם: $Makefile$, $myLib.h$, $Sender.c$, $Receiver.c$, $msg.txt$ יש להוריד ולשמור אותם באותה התקייה.
- נפתח שני טרמינלים על התקיה עם הקבצים, רצף הפקודות הבא מריץ את הפרוייקט:
  1. בטרמינל הראשון נרשום "$make\ all$" ולאחר מכן "$clear$", הדבר יגרום ליצירת הפרוייקט וניקוי הפלט הסטנדרטי.
  2. בטרמינל הראשון נרשום "$Receiver$", הדבר יגרום להרצת התוכנית של מקבל הקבצים.
  3. בטרמינל השני (**על שניהם להיות פתוחים במקביל!**) נרשום "$Sender$", הדבר יגרום להרצת התוכנית של שולח הקבצים.
  4. כעת נפתח בפנינו ממשק המשתמש, ניתן להזין קלט ולקבל פלט רצוי.

1.3 ממשקי המשתמש

לאחר שביצענו את כלל השלבים מהנושא הקודם, כעת נרצה להשתמש בתוכנית (**עד כה רק הרצנו...**).
בשתי התוכניות נפתח בפנינו ממשק נוח למשתמש אשר מדפיס למשתמש קלט ברור על הפעולות (רק אלו הרלוונטיות לידיעת המשתמש) שקרו וכמובן גם מבקש מאיתנו קלט על מנת להמשיך את הרצת התוכנית.
בנושא זה, נסקור את הממשקים של שתי התוכנית הרצות (שהן, תזכורת: $Receiver$ ו־ $Sender$) ונראה כיצד ניתן להגיע ולקבל את הפלט הרצוי.

1.3.1 ממשק ה־$Sender$

מיד בעת הפעלת התוכנית, הקובץ כבר מבצע מספר רב של פעולות (הפעם הראשונה מבוצעת אוטומטית והחל מהפעם השניה התוכנית תשאל אותנו האם ברצוננו לחזור על פעולות אלו). פלט הממשק להלן:

נעבור על הפלט שאנו רואים בתמונה:

```
liorvi35@liorvi-VirtualBox:~/Desktop/TCP_CongestionControl_C$ ./Sender
file opened!
socket created!
connected to receiver!
first half has been sent!
waiting for authentication...
authentication match!
CC algorithm has been changed to: "reno"
second half has been sent!
send file again? (y/n)
```

1. ראשית, אנו רואים כי הקובץ שאנו שולחים למקבל נפתח בהצלחה.
2. אנו רואים כי נוצר שקע התקשורת ($socket$) בהצלחה.
3. השולח התחבר בהצלחה למקבל.
4. החלק הראשון של הקובץ נשלח.
5. ממתין לקבלת הסימון המוסכם לאישור ($authentication$) מהשרת.
6. קיבל את האישור והוא אכן נכון.
7. שונה האלגוריתם לבקרת העומס ($CC\ algorithm$) מ־$cubic$ ל־$reno$.
8. החלק השני של הקובץ נשלח.
9. שאלה למשתמש - האם לחזור על התהליך הנ"ל ולשלוח את הקובץ מחדש?
10. המתנה של המשתמש להזנה של התו המתאים. יש להזין $'y'$ אם ברצוננו לשלוח מחדש ויש להזין $'n'$ אם ברצוננו לצאת.

נציין כי שורות 3-1 הן שורות אשר יהיו מודפסות רק בעת ההפעלה הראשונה (שהריי אם כבר הקובץ נפתח, השקע נוצר והשולח מחובר למקבל - אין צורך לחזור על פעולות אלו). כעת נעבור על שתי האפשרויות המתקבל (לאחר מכן נראה גם פלטים בהתאמה בממשק של המקבל).

אם המשתמש ביקש לצאת מהתוכנית (ז"א הזין את התו $'n'$) אזי נקבל את הפלט הבא:

```
liorvi35@liorvi-VirtualBox:~/Desktop/TCP_CongestionControl_C$ ./Sender
file opened!
socket created!
connected to receiver!
first half has been sent!
waiting for authentication...
authentication match!
CC algorithm has been changed to: "reno"
second half has been sent!
send file again? (y/n)
n
exit message has been sent
liorvi35@liorvi-VirtualBox:~/Desktop/TCP_CongestionControl_C$
```

נשים לב כי קיבלנו פלט שאומר שנשלחה הודעת יציאה ולאחר מכן התוכנית הסיימה, במילים אחרות, אם המשתמש מזין $'n'$ אזי הלקוח שולח הודעת יציאה לשרת, סוגר את התקשורת ומסיים לרוץ.

כעת נסתכל על המקרה השני, אם המשתמש ביקש לשלוח את הקובץ מחדש (ז"א הזין את התו $'y'$), נקבל את הפלט הבא:

```
liorvi35@liorvi-VirtualBox:~/Desktop/TCP_CongestionControl_C$ ./Sender
file opened!
socket created!
connected to receiver!
first half has been sent!
waiting for authentication...
authentication match!
CC algorithm has been changed to: "reno"
second half has been sent!
send file again? (y/n)
y
CC algorithm has been changed to: "cubic"
first half has been sent!
waiting for authentication...
authentication match!
CC algorithm has been changed to: "reno"
second half has been sent!
send file again? (y/n)
```

נעבור על השורות שקיבלנו לאחר שהמשתמש הזין $'y'$:

1. שונה אלגוריתם בקרת העומס מ־$"reno"$ ל־$"cubic"$.
2. החלק הראשון של הקובץ נשלח.
3. ממתין לקבלת הסימן המוסכם מהשרת.
4. קיבל את האישור והוא אכן נכון.
5. שונה האלגוריתם לבקרת העומס מ־$cubic$ ל־$reno$.
6. החלק השני של הקובץ נשלח.
7. כעת, ישנה חזרה על התהליך - ישנה שוב שאלה למשתמש האם לשלוח מחדש את הקובץ או האם לצאת מהתוכנית.

ספוילר: כך התוכנית ממשיכה לעבוד בכל בחירה של המשתמש, כלומר אם המשתמש מזין כעת $'y'$ הלקוח שוב משנה את האלגוריתם, שולח את החלק הראשון, מחכה ומקבל אישור, משנה את האלגוריתם, שולח את החלק השני ולאחר מכן שוב שואל את המשתמש האם לחזור על התהליך ואם המשתמש מזין כעת $'n'$ אזי תישלח הודעת יציאה לשרת, התקשורת תיסגר והתוכנית תיעצר.
בזאת סיימנו לפרט על הממשק של השולח.

1.3.2 ממשק ה־$Receiver$

כעת, בנושא זה, נעבור על הממשק של המקבל ונראה בהתאמה כיצד הפעולות שהשולח מבצע משפיעות על המקבל בצורה ישירה.
הפלט המתקבל מהרצת השרת:

```
liorvi35@liorvi-VirtualBox:~/Desktop/TCP_CongestionControl_C$ ./Receiver
socket created!
socket bound!
waiting for connection...
```

נשים לב כי הודפסו שלוש שורות, נעבור עליהן:

1. שקע התקשורת נוצר בהצלחה.
2. שקע התקשורת נקשר לכתובת IP (אנו משתמשים בכתובת המארח המקומי שהיא "127.0.0.1") ונקשר לכתובת $PORT$ (אנו משתמשים במספר "8395").
3. השרת ממתין לחיבור מהמשתמש.

שוב, נציין כי שלוש השורות האלו יודפסו רק פעם אחת ויחידה והיא בתחילת הרצת המקבל. כעת, נפעיל את השולח ונראה כיצד הדבר משפיע על המקבל:

```
liorvi35@liorvi-VirtualBox:~/Desktop/TCP_CongestionControl_C$ ./Receiver
socket created!
socket bound!
waiting for connection...
sender connected!
first message has been received!
authentication has been sent!
CC algorithm has been changed to: "reno"
second message has been received!
CC algorithm has been changed to: "cubic"
```

נעבור על השורות החדשות שהודפסו:

1. השולח התחבר בהצלחה.
2. החלק הראשון של הקובץ התקבל.
3. נשלח האישור על קבלת הקובץ.
4. שונה האלגוריתם לבקרת העומס מ־$cubic$ ל־$reno$.
5. החלק השני של הקובץ התקבל.
6. שונה אלגוריתם בקרת העומס מ־"$reno$" ל־"$cubic$".
7. השרת ממתין לאישור להמשך התהליך (או סיומו) מהמשתמש.

כעת, ברצוננו להסתכל מה קורה לשרת כאשר המשתמש מבקש לצאת (ז"א מזין $'n'$):

```
liorvi35@liorvi-VirtualBox:~/Desktop/TCP_CongestionControl_C$ ./Receiver
socket created!
socket bound!
waiting for connection...
sender connected!
first message has been received!
authentication has been sent!
CC algorithm has been changed to: "reno"
second message has been received!
CC algorithm has been changed to: "cubic"
exit message has been received!


###################

collected DATASET is:

time to receive part 1 of file 1 is 47903 [us]
time to receive part 2 of file 1 is 300 [us]

###################

the times of avg part:
the average of sending first file - by "cubic" - is: 47903.000 [us]
the average of sending second file - by "reno" - is: 300.000 [us]

###################
liorvi35@liorvi-VirtualBox:~/Desktop/TCP_CongestionControl_C$
```

נעבור על השורות החדשות:

1. התקבלה הודעת יציאה מהלקוח.
2. הודפסו הזמנים שלקח לקבל כל חלק מהלקוח.
3. הודפסו הזמנים הממוצעים לקבל כל אחד מהחלקים.

כלומר, אם המשתמש מבקש לצאת גם השרת מסיים לעבוד ופולט את המידע שאסף מאז תחילת תהליך ההרצה (להלן $Dataset$).
כעת נראה מה קורה כאשר המשתמש מבקש לשלוח שוב את הקובץ:

```
liorvi35@liorvi-VirtualBox:~/Desktop/TCP_CongestionControl_C$ ./Receiver
socket created!
socket bound!
waiting for connection...
sender connected!
first message has been received!
authentication has been sent!
CC algorithm has been changed to: "reno"
second message has been received!
CC algorithm has been changed to: "cubic"
first message has been received!
authentication has been sent!
CC algorithm has been changed to: "reno"
second message has been received!
CC algorithm has been changed to: "cubic"
```

נשים לב כי קורה בדיוק אותו תהליך כמו בתחילת ההרצה. כעת נניח כי המשתמש ביקש לצאת, נראה מה קורה מבחינת הדפסות הזמנים:

```
liorvi35@liorvi-VirtualBox:~/Desktop/TCP_CongestionControl_C$ ./Receiver
socket created!
socket bound!
waiting for connection...
sender connected!
first message has been received!
authentication has been sent!
CC algorithm has been changed to: "reno"
second message has been received!
CC algorithm has been changed to: "cubic"
first message has been received!
authentication has been sent!
CC algorithm has been changed to: "reno"
second message has been received!
CC algorithm has been changed to: "cubic"
exit message has been received!


###################

collected DATASET is:

time to receive part 1 of file 1 is 43014 [us]
time to receive part 2 of file 1 is 273 [us]
time to receive part 1 of file 2 is 335 [us]
time to receive part 2 of file 2 is 249 [us]

###################

the times of avg part:
the average of sending first file - by "cubic" - is: 21674.500 [us]
the average of sending second file - by "reno" - is: 261.000 [us]

###################
liorvi35@liorvi-VirtualBox:~/Desktop/TCP_CongestionControl_C$
```

נשים לב כי קורה כאן אותו תהליך כמו בפעם הראשונה כאשר המשתמש ביקש לצאת אבל הפעם, מודפסים יותר זמנים (שהריי הפעם, הקובץ נשלח פעם נוספת).

ספויילר נוסף: גם כאן התוכנית ממשיכה לעבוד באותה דרך כמו שראינו כעת, כלומר, אם המשתמש מזין $'n'$ אזי השרת מקבל הודעת יציאה, מדפיס זמנים לקבלת כל חלק ואת הזמנים הממוצעים ומסיים לעבוד. ואם המשתמש מזין $'y'$ (נניח שהמשתמש הזין $'y'$ מספר $t \in \mathbb{N}$ כלשהו של פעמים) אזי יודפסו $2 \cdot t$ שורות של זמנים לקבלת החלקים של t הקבצים שנשלחו ועוד שתי שורות להדפסת הזמנים הממוצעים.

בזאת סיימנו לפרט על הממשק של המקבל.

1.4 הקובץ היוצר - $Makefile$

נחזור חזרה לבסיס, הקוד אשר כתבנו נכתב בשפה (העילית) C. המחשב אינו יודע ישירות להריץ את קובץ הקוד ולכן עלינו לבצע מספר תהליכים לפני שנוכל בכלל להריץ את התוכנית שכתבנו, התהליכים הללו הם הידור וקישור. נסביר, המחשב (המעבד) כלל אינו יודע מה זו השפה שאנו כותבים בה ומה המשמעות שלה, הוא מבין רק את השפה הבינארית, לכן כדי שנוכל להריץ עלינו לתת למחשב משהו שהוא יודע לעבוד איתו. ישנם שני תהליכים עיקריים אשר קובץ ה־$Makefile$ מטפל בהם והם:

1. הידור (קומפילציה) - תהליך זה ממיר את קבצי שפת התכנות שלנו (קבצים עם סיומות "$.c$" ו־ "$.h$") לקבצי אובייקטים בשפה הבינארית (קבצים עם סיומת "$.o$" ו־ "$.h.gch$"), כעת יש בידינו קבצים אשר המחשב יודע לתפעל וניתן להמשיך לעבוד עימם.

2. קישור (לינקוג') - לאחר שהשגנו קבצים בשפה הבינארית, נוכל שלייצר מהם קבצי הרצה לתוכניות שלנו (שכן, כעת המחשב כן מבין אם נגיד לו להריץ קבצים בינאריים) תהליך זה ממיר את הקבצים הבינאריים שהשגנו בתהליך הקודם לקבי הרצה (הערה: במערכת ההפעלה Ubuntu LTS לא לכל קובץ חייבת להיות סיומת, אבל לדוגמה במערכת הפעלה Windows קובץ הרצה חייב להיות בעל סיומת "$.exe$"), שאותם נוכל להריץ ולהפעיל ולממש איתם יחסי קלט־פלט.

על מנת לייצר בקלות את קבצי ההרצה ולהריץ את התוכנית, יצרנו קובץ הנקרא $Makefile$ ועל ידי פקודת בת שורה אחת הקובץ מריץ (אם ישנו בכך צורך) את שני התהליכים שציינו מלעיל ומייצר לנו בקלות קבצי הרצה. נעבור על הקוד שלו ונסביר כיצד הוא עובד:

```
# macros - for more dynamic Makefile
CC = gcc # compiler
FLAGS = -Wall -g # compilation flags

# for not creating 'all' and 'clean' files
.PHONY: all clean

# final targets
all: Receiver Sender

# (linkage) making executables from objects:
Receiver: Receiver.o
	$(CC) $(FLAGS) -o Receiver Receiver.o

Sender: Sender.o
	$(CC) $(FLAGS) -o Sender Sender.o

# (compilation) making object files from '.c' and '.h' files:
Receiver.o: Receiver.c myLib.h
	$(CC) $(FLAGS) -c Receiver.c myLib.h

Sender.o: Sender.c myLib.h
	$(CC) $(FLAGS) -c Sender.c myLib.h

# delete all files that created after 'make' command (not deleting '.c' files)
clean:
	rm -f *.o *.h.gch Receiver Sender
```

1. שורות $2-3$: בשורות אלו אנו יוצרים קיצורי כתיבה ("מאקרו"). זאת על מנת להפוך את הכתיבה שלנו ליותר קריאה ודינאמית. אנו יוצרים מאקרו אחד עבור המהדר (קומפיילר) שישמש אותנו בתהליך ההמרה (אנו משתמשים ב־ $Gnu\ Compiler\ Collection$, במילים אחרות זהו המהדר הסטנדרטי ־ gcc) ובנוסף, אנו יוצרים מאקרו עבוד דגלי הקומפילציה, הדגלים שעימם אנו נקמפל הם "$-Wall, -g$" משמעות הדגל הראשון היא להציג בפנינו לא רק שגיאות קומפילציה אלא גם אזהרות קומפילציה שמהוות סיכון לתוכנית, הדגל השני עוזר למנפה (להלן "$debugger$") למצוא אזהרות ושגיאות.
שילוב שני הדגלים האלו מהדר את הקוד שלנו בצורה יותר קפדנית ובכך אנו מקבלים תוכנית תקינה ונקייה יותר.

2. שורה 6: בשורה זו אנו מצהירים כי התוויות ($Makefile$ היא שפת תוויות עם משימות לביצוע) "all" ו־ "$clean$" הן אינן קבצים ואין לייצר קבצים הנקראים כך. הסיבה לכך כי אין לנו צורך ושימוש בהם, אילו רק תוויות לצורך נוחות השימוש (למשתמש ־ כי הוא מזין בכל פעם סה"כ שורה אחת כדי לבצע הרבה) ועל כן אין לנו באמת צורך , כמו כן אין.

3. שורה 9: בשורה זו אנו יוצרים תווית אשר תשמש אותנ ליצירת שני הקבצים של השולח ושל המקבל, בו זמנית. תווית זו היא למטרת נוחות, כדי שנוכל בפקודה אחת לייצר שני קבצים ולא נצטרך לייצר כל אחד ידנית בנפרד.

4. שורות $12-14$ ו־ שורות $15-16$: בשורות אלו מתבצע תהליך הקישור, התווית משתמשת בקובץ אובייקט בינארי של השולח או המקבל ויוצרת לו קובץ הרצה בהתאם.

5. שורות $19-20$ ו־ שורות $22-23$: בשורות אלו מתבצע תהליך ההידור, התווית משתמשת בקובץ קוד הכתוב בשפת C ובקובץ ספריה, משלבת את שניהם ומקבלת קובץ בינארי.

6. שורות $26-27$: בשורות אלו אנו יוצרים תווית אשר בעת הרצתה תשמש אותנו למחיקת כל הקבצים אשר נצרו. תווית זו גם היא למטרת נוחות וסדר, כדי שבפקודה אחת נוכל למחוק את כל ה "בלאגן" שנוצר לנו.

1.5 קובץ ספריה - $myLib.h$

כאשר אנו מתעסקים בתכנות בשפת C, יש לנו כמה וכמה סוגים של קבצים.
ראשית, ישנם (מן הסתם) קבצי קוד הכתובים בשפה הנ"ל ומממשים את הלוגיקה ואת הפקודות השונות.
עם זאת, אלה לא הקבצים היחידים, ישנם גם קבצי "כותרות"/"ספריות" ($"header"$), אשר מהווים בסיס ויסוד לקבצי הקוד. קבצי הספריות מכילים הצהרות על קבועים והצהרות על פונקציות (הצהרות בלבד) כך שאם נצרף את קובץ הכותרת לקובץ הקוד, הקוד יהיה מודע לכל הקבועים והפונקציות ויוכל להשתמש בהם (אם קיים מימוש), למעשה קבצים אלה הם עיבוד מראש ($"pre-processed"$).
כיוון שכדי לממש את הלקוח והשרת, היינו צריכים להשתמש בכלילה של קבצי ספריות רבים ויצירת מספר קבועים שחשוב שיהיו גם בלקוח וגם בשרת, החלטנו ליצור ספריה משלנו על מנת לשמור על הסדר, הנוחות והארגון בקוד.
נעבור על הקוד של הספריה שלנו ונסביר את הכתוב:

```c
/* including libraries */
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <arpa/inet.h>
#include <netinet/in.h>
#include <netinet/tcp.h>
#include <errno.h>
#include <unistd.h>
#include <time.h>
#include <sys/time.h>

/* defining constants */
#define IP "127.0.0.1" // server's ip address
#define PORT 8395 // connection port
#define BUFSIZE 8192 // size of the buffer
#define AUTH "0000000111111001" // authentication code
#define OK "ok" // ready message
#define CONNECTIONS 50 // number of clients that server can listen simultaneously
```

בלי הגבלת הכלליות, קודם כל נסביר את הקבועים שהגדרנו:

1. הגדרנו את כתובת ה־IP אשר תהיה בשימוש בעת שליחה לשרת (זוהי הכתובת של השרת והלקוח שולח את המידע לכתוב הזו).
2. הגדרנו את מספר ה־$PORT$ שיהיה בשימוש וישמש לתקשורת בין הלקוח לשרת. המספר 8395 הוא אקראי ואין סיבה מיוחדת לבחירה הזו (במילים אחרות השתמשנו במספר $PORT$ כרצוננו).
3. הגדרנו את גודל החוצץ ($"buffer"$) אשר יהיה בשימוש, החוצץ יחזיק בכל פעם את ההודעות המתקבלות והנשלחות מהלקוח לשרת וההפך.
4. הגדרנו את הסימן המוסכם (האישור) שהמקבל שולח לשרת. הגענו למספר הזה כך:
ארבעת הספרות האחרונות של ת"ז של ליאור הן - 1763, נעביר מספר זה לבינארית ונציג בתור 16 ביטים - נקבל: 0000011011100011. ארבעת הספרות האחרונות של ת"ז של יועד הן - 1818 נמיר לבינארית - נקבל: 0100011100000101. כעת, נבצע פעולת xor בין שתי התוצאות, נקבל בדיוק: 0000000111111001.
5. הגדרנו הודעת אישור על התחלת התקשורת בין השולח למקבל.
6. הגדרנו הודעת יציאה אשר נשלחת מהשולח למקבל כאשר המשתמש מעוניין להפסיק לשלוח קבצים ולצאת

7. הגדרנו את המספר המקסימלי של לקוחות שהמקבל יכול להאזין בו זמנית (ספיציפית כאן, נדרש להאזין ללקוח אחד בלבד שהוא השולח אבל באופן כללי בגלל שניתן, הגדרנו יותר).

כעת, נסביר את הספריות הספריות שהשתמשנו בהן:

1. שורות $1-3$: ספריות קלט ופלט סטנדרטיות - שימוש חובה בכל תוכנית. הספריה השלישית יוצרת אובייקט חדש עבור בוליאנים, לא השתמשנו בספריה זו אבל בכל מקרה החלטנו להוסיף כיוון שזוהי ספרייה סטנדרטית ולמקרה שתוכל לבוא לעזרתינו.

2. שורה 5: ספריה עם פעולות רבות לטיפול במחרוזות, תורמת רבות לנוחות השימוש במחרוזות.

3. שורות $6-10$ ושורה 12: אלו ספריות עבור תכנות שקעים ("GNIMMARGORP TEKCOS"). ספריות חובה בשביל המטלה.

4. שורה 11: זוהי ספריה בעל אובייקט אשר מחזיק את שגיאת המערכת האחרונה שקרתה, שימושית מאוד כיוון שבמקרים של תקלות החזרנו למערכת ההפעלה את מספר השגיאה (פתרון אלגנטי יותר מאשר להחזיר 1 גנרי).

5. שורות $13-14$: אלו ספריות אשר מחזיקות זמנים, השימוש שלנו איתן היה בחישוב הזמנים של קבלות החלקים של הקובץ.

1.6 קובץ הטקסט - $msg.txt$

הקובץ שבחרנו לשלוח מהלקוח לשרת הוא קובץ גנרי לחלוטין המכיל את הספרות $2-9$. גודל הקובץ הוא $1.1MB$ (הקובץ עומד בדרישות כיוון שהוא גדול מ $1MB$).
תמונה להמחשת הקובץ (כמובן שזה לא כל הקובץ, הוא גדול מידי ולא ניתן להציג את כל כולו):

```
293589359859376927887392479335664356255999777975275893383898232879329287243882264474289625367533894796784 5
259954667588875526952336242549985329223858687456898973482399733648547622246956435255323476649578652743852 6
796736887737836694926223434522379228938634223646726548473462888782377476659595735743727599982555442639727 4
569554787667299835225953963727368626452239988844566329875497574559877582576829767982323549676429696642435 8
862539786592435754888247659998626624986452234862969527283637298788932749378692943847797335898662627472963 8
363233924258228433747874373992392948939558338395422632932979472756533277286946394858827668337978344466393 2
944989362536694762748743294647982278599678338367589774322832338743956555264762563954344565845459542688877 5
666324787885462245243265492667575277383298762425227986439599542635767965649635577226536845576894726436344 7
283368974788485454524673468964437998367965694572223684236924965628752349835894532322965336375339353568498 8
368556259399494832395662598798862898245555876524292378688242397293695479243796459784687599229566424393248 2
248389366869653883972227367447434968289478796823268676267593228572864442532673222583289565947999464588872 5
299822435386873834683792759843632396629478662294877834572539874339772445695829739535788465443934676779969 6
583652938672456845822628847257449399767982737528767746897326493329843262653873833736682779742494665448492 5
822346883937955632643387586235665249572366277487943874934943659797599769849782647752325533674572327537696 6
793884894484482723498653955859986357463663674269224999434729762985452398699882966493937938348496239489767 4
864655559366399985845944773759986649524272888248238525935329296385459855873427293446769343879332952678454 8
725364968832345947236322567265775582427458699573934248323652499529574995694833856326468242766429325826674 4
943492994884838226926897348769945856477266972985679336338355565536847323366277236756795486755745282882288 2
723296966885979994529568285832782646628628349557873732989472446757427296356639567486864437859795842548855 3
459736353465796666699996747657735784798548943254482763248885864632295782663399855289532592365875962539239 5
277285783792454453738344772772485575872823435643764835386886964794244573665238426399433663769562923368865 3
942638293358298435287755657435497857884978646248882553773792679357387374559483653634694922772844664948237 5
482928945828873538787994994628739529282388783899866439762693769753743883733768662777244993667856575767582
768658727888428637842656844372792335884439959822453639945457652778655622674266286639553899459298397973979 4
387368885559968836378257654684655268558584632766352676924882658986464338448548499996364434538328692786525 9
```

אין יותר מידי מה להסביר בחלק זה, לכן נעבור לחלק הבא.

1.7 קובץ השולח - *Sender.c*

1.8 קובץ המקבל - *Receiver.c*

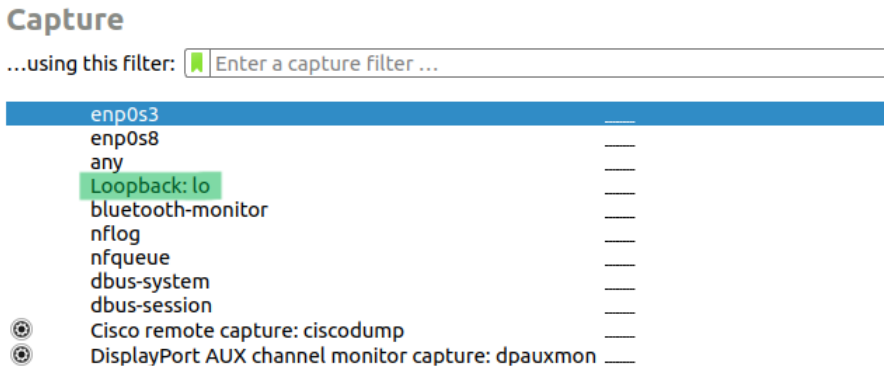
2 פרק ב' - המחקר על אלגוריתמי בקרת העומס

2.1 הקדמה

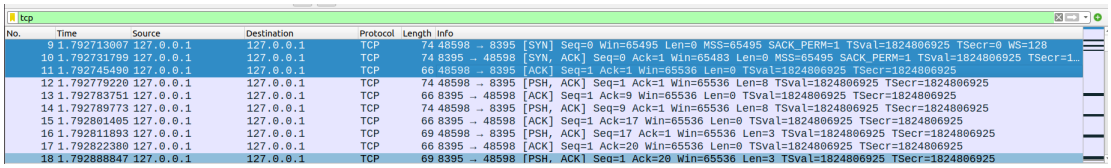
במטלה זו נדרשו לכתוב שני קבצים, האחד לקוח (שולח) והשני שרת (מקבל) ולבצע ביניהם פעולות של תעבורות רשת, שהן שליחה וקבלה של הודעות.
פעולות אלו של תעבורות הרשת, בוצעו בעזרת פרוטוקול התקשורת TCP אשר הוא פרוטוקול תקשורת אמין אשר מתמקד בהוגנות על ידי חילוק רוחב הפס של תעבורת הרשת (הפרוטוקול מחלק את רוחב הפס באופן שווה בין כלל המשתמשים המחוברים ברגע נתון). השגת החלוקה השווה הזו מתאפשרת בעזרת אלגוריתמי בקרת ותכנון עומס ($Congestion\ Control$). בפרק זה נעסוק כיצד איבוד חבילות (ובעזרת שימוש באלגוריתמי הבקרה) משפיע על זמני השליחה והקבלה של קבצים ברשת בפרוטוקול זה.
את הסקירה של תעבורת הרשת והחבילות הנשלחות והתקבלות, נבצע בעזרת התוכנה "$Wireshark$", נשתמש בה באופן הבא:

1. נבחר אחוז איבוד חבילות מסויים.
2. נפעיל את סריקת ה-"$Wireshark$".
3. נריץ את המקבל.
4. נריץ את השולח.
5. נשלח את הקובץ 5 פעמים.
6. נעצור את שתי התוכניות.
7. נתבונן בתוצאות שקיבלנו - תעבורת הרשת וזמני הקבלה שחושבו.
8. נסיק מסקנות מתאימות.

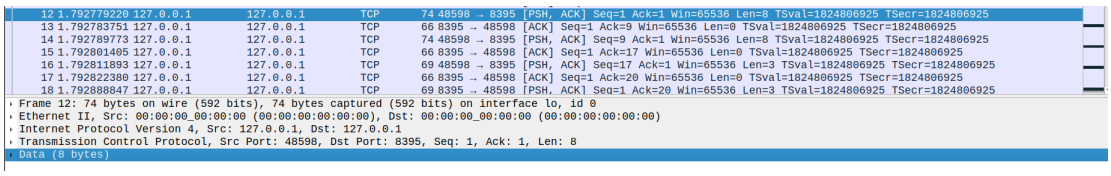
נפעיל את ה-"$Wireshark$", נבחר את האפשרות לרחרח חבילות ב- "LOOPBACK : LO", שזוהי תקשורת עם המארח המקומי ("$localhost : 127.0.0.1$"). כיוון שהתוכנות שכתבנו רצות על המחשב שלנו עצמו ולא מתחברות לשאר רשת האינטרנט, נבחר באפשרות זו.
בחירה זו תהיה זהה לכל הבדיקות!



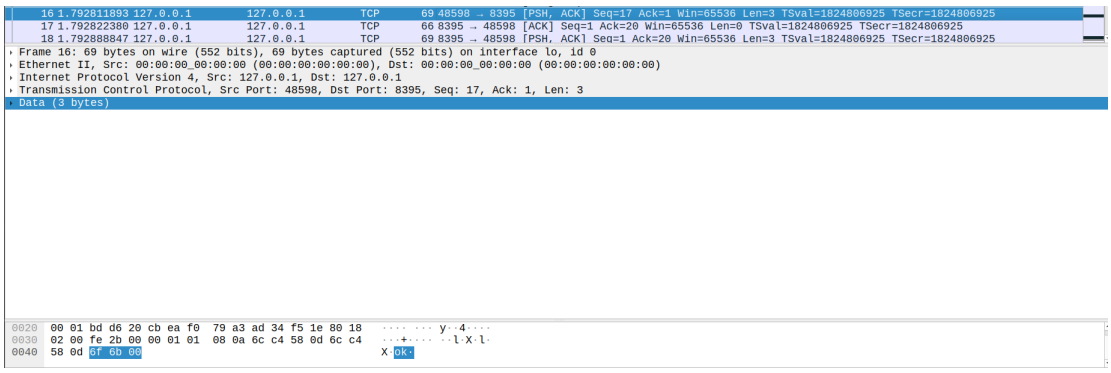
לאחר שבחרנו, נפעיל את הלקוח והשרת ונצפה לקבל תעבורה. בעת שימוש בפרוטוקול "TCP", על הלקוח קודם כל לוודא שהשרת זמין ומוכן לתקשורת (כיוון שזהו פרוטוקול אמין), הבדיקה הזו תבוצע על ידי "לחיצת ידיים משולשת", הבדיקה מבוצעת כך: נשלחת חבילה ראשונית מהשולח למקבל שהיא חבילת "SYN" חבילו זו אינה מכילה מידע. לאחר מכן, נשלחת חבילה שניה מהמקבל לשולח חבילה זו היא חבילת "$SYN-ACK$" פירושה שהשרת זמין ומאשר שקיבל את החבילה הראשונה שנשלחה. לסיום, נשלחת חבילה שלישית "ACK", חבילה זו כבר יכולה (להכיל מידע רלוונטי שהלקוח רוצה לשלוח).
לאחר שפתחנו קשר עם השרת, נתחיל לבצע את הבדיקות שלנו עם אחוזי איבוד הפקטות:



ראשית, נשלח לשרת את הגודל של החלק הראשון של הקובץ ולאחר מכן נשלח גם את החלק השני של הקובץ (התמונה היא של השליחה של החלק הראשון, החלק השני זהה לחלוטין):



כעת, כדי להודיע לשרת שאנחנו מוכנים להתחיל לשלוח לו את הקובץ, אנו שולחים לו הודעת "ok" ומצפים שיחזיר לנו אותה, נראה:



לאחר מכן, ניתן לראות בחבילה 17 כי השרת קיבל את ההודעה ואכן חבילה 18 היא התשובה שאנו מצפים לה:

```
17 1.792822380 127.0.0.1   127.0.0.1   TCP   66 8395 → 48598 [ACK] Seq=1 Ack=20 Win=65536 Len=0 TSval=1824806925 TSecr=1824806925
18 1.792888847 127.0.0.1   127.0.0.1   TCP   69 8395 → 48598 [PSH, ACK] Seq=1 Ack=20 Win=65536 Len=3 TSval=1824806925 TSecr=1824806925
› Frame 18: 69 bytes on wire (552 bits), 69 bytes captured (552 bits) on interface lo, id 0
› Ethernet II, Src: 00:00:00_00:00:00 (00:00:00:00:00:00), Dst: 00:00:00_00:00:00 (00:00:00:00:00:00)
› Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
› Transmission Control Protocol, Src Port: 8395, Dst Port: 48598, Seq: 1, Ack: 20, Len: 3
› Data (3 bytes)
```

```
0020  00 01 20 cb bd d6 ad 34  f5 1e ea f0 79 a6 80 18   ··· ····4 ····y···
0030  02 00 fe 2b 00 00 01 01  08 0a 6c c4 58 0d 6c c4   ···+···· ··l·X·l·
0040  58 0d 6f 6b 6b                                     X·okk
```

החל מנקודה זו, יהיה רלוונטי להסתכל על ההשלכות של איבודי החבילות ברשת.

2.2 0% איבוד חבילות

ראשית נפתח חלון טרמינל ונרשום את הפקודה הבאה על מנת לאפס את איבוד החבילות (במקרה והיה דלוק):

$$\text{"}sudo\ tc\ qdisc\ del\ dev\ lo\ root\ netem\text{"}$$

כעת נסתכל מה קורה בתעבורה.

2.2.1 שליחת החבילות

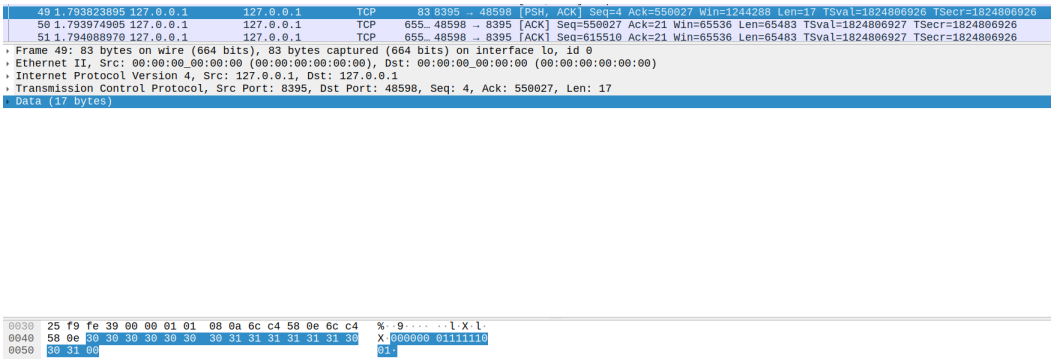
כעת, נשלחות החבילות עם המידע הרלוונטי באמת, ז"א נשלחות מהלקוח לשרת החבילות עם המידע מהקובץ ונשלחות מהשרת ללקוח חבילות של אישורים למיניהם.
הקובץ גדול, לכן הלקוח שולח אותו לשרת במקטעים ("סגמנטים").
נראה כי בחבילה הזו נשלח הסגמנט הראשון (לדוגמה):

```
20 1.792957232 127.0.0.1   127.0.0.1   TCP   328… 48598 → 8395 [ACK] Seq=20 Ack=4 Win=65536 Len=32768 TSval=1824806926 TSecr=1824806925
21 1.792996254 127.0.0.1   127.0.0.1   TCP    66 8395 → 48598 [ACK] Seq=4 Ack=32788 Win=65536 Len=0 TSval=1824806926 TSecr=1824806926
22 1.792997489 127.0.0.1   127.0.0.1   TCP   328… 48598 → 8395 [PSH, ACK] Seq=32788 Ack=4 Win=65536 Len=32768 TSval=1824806926 TSecr=1824806925
23 1.793024970 127.0.0.1   127.0.0.1   TCP    66 8395 → 48598 [ACK] Seq=4 Ack=65556 Win=65536 Len=0 TSval=1824806926 TSecr=1824806926
24 1.793175336 127.0.0.1   127.0.0.1   TCP   328… 48598 → 8395 [ACK] Seq=65556 Ack=4 Win=65536 Len=32768 TSval=1824806926 TSecr=1824806926
› Frame 20: 32834 bytes on wire (262672 bits), 32834 bytes captured (262672 bits) on interface lo, id 0
› Ethernet II, Src: 00:00:00_00:00:00 (00:00:00:00:00:00), Dst: 00:00:00_00:00:00 (00:00:00:00:00:00)
› Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
› Transmission Control Protocol, Src Port: 48598, Dst Port: 8395, Seq: 20, Ack: 4, Len: 32768
› Data (32768 bytes)
```

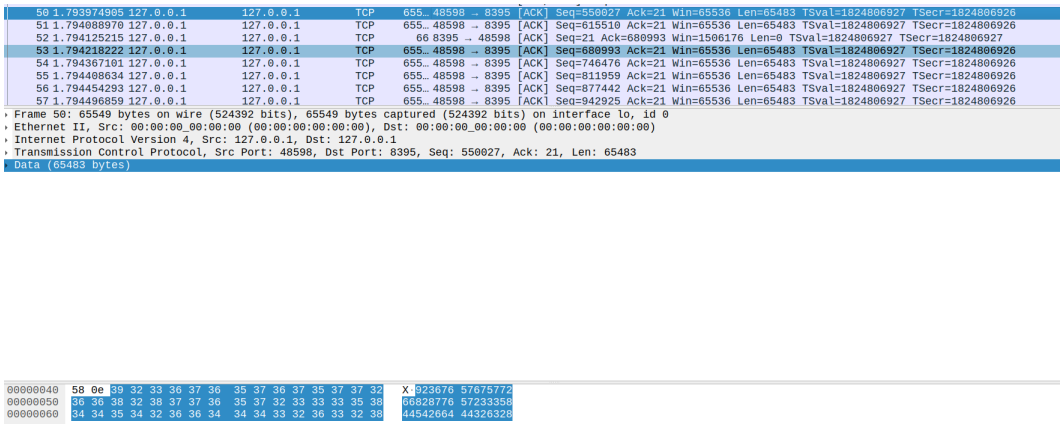
```
0040  58 0d 32 39 33 35 38 39  33 35 39 38 35 39 33 37   X·293589 35985937
0050  36 39 32 37 38 38 37 33  39 32 34 37 39 33 33 35   69278873 92479335
0060  36 36 34 33 35 36 32 35  35 39 39 39 37 37 37 39   66435625 59997779
```

כיוון שאין איבוד חבילות מופעיל, אזי נשים לב כי גם כל החבילות מתחת לחבילה 20 הן גם חבילות של שליחה וקבלה ואין חבילות אשר נאבדות באמצע.

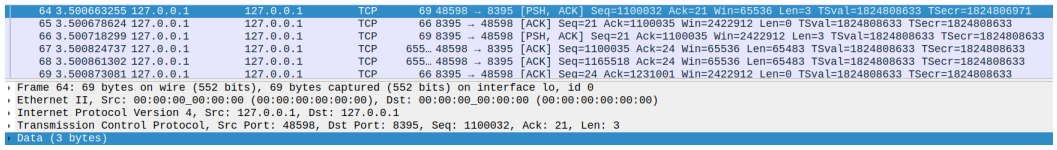
כעת נרצה לקבל אימות (אוטנטיקציה) מהשרת על השליחה של החלק הראשון, נראה:

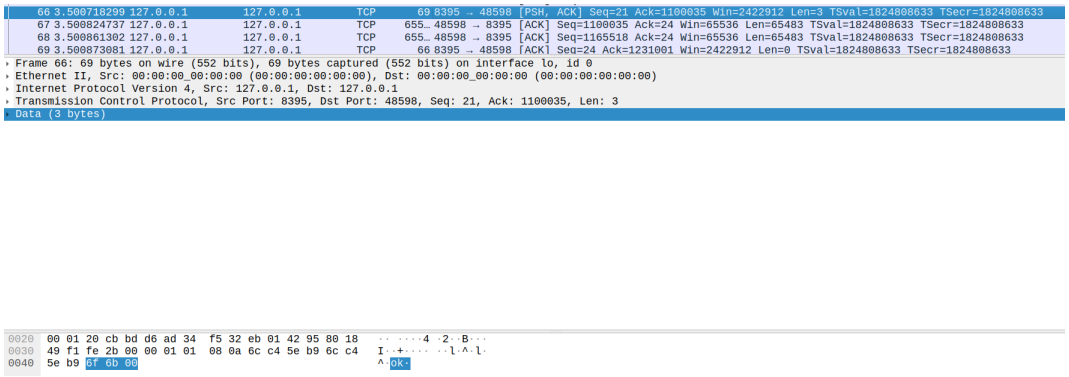


ושוב כיוון שאין איבוד חבילות מופעל אין חבילות שנאבדות. כעת נשלח את החלק השני של הקובץ (להלן סגמנט לדוגמה):

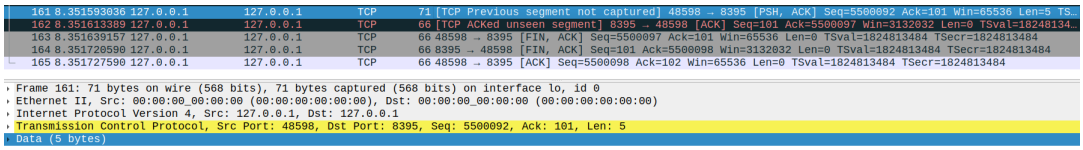


כעת, מכיוון שאנו חוזרים על התהליך (לפי הדרישות צריך להריץ לפחוות 5 פעמים), אנו שולחים הודעת "ok" לשרת וגם מצפים שהוא יחזיר לנו אותה, נראה זאת בתמונות הבאות:



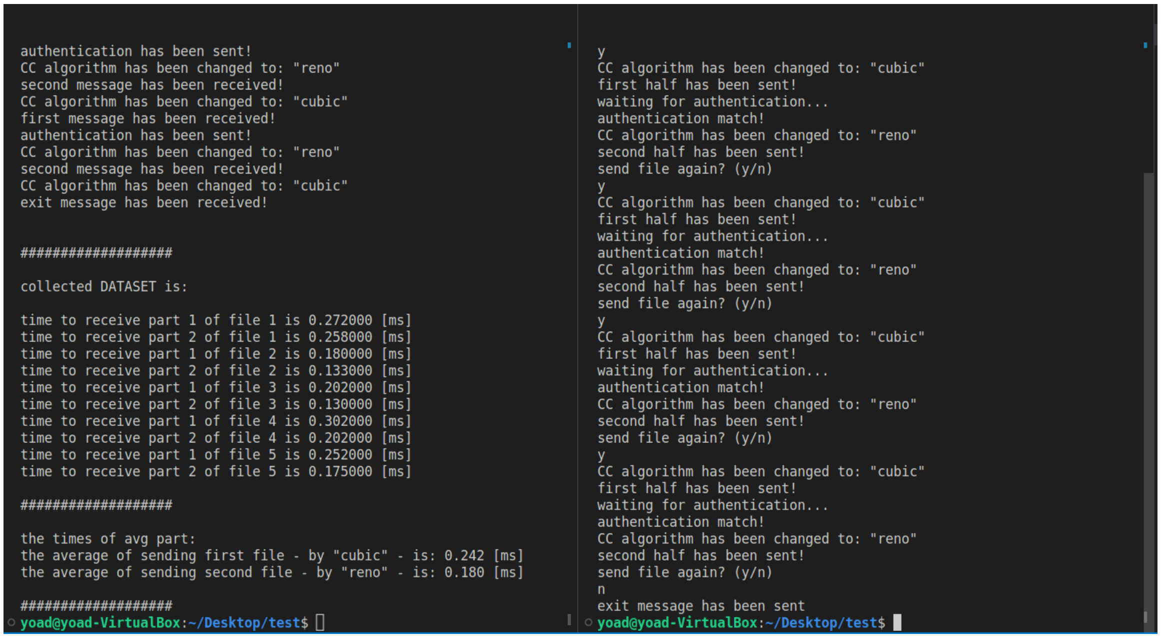


כעת, אנו חוזרים על כל התהליך, הפלטים שאנו מקבלים זהים לחלוטין ולכן נסתפק בצילומים שראינו עד כה.
כעת, כאשר הלקוח מבקש לסיים את התהליך, נשלחת הודעת יציאה נראה:



2.2.2 תוצאות

עד כה ראינו את תעבורת הרשת בתהליך התקשורת, כעת נרצה לדון כיצד אפס אחוז של איבוד פקטות, ושני אלגוריתמי בקרת העומס משפיעים על הזמנים.
ראשית, נראה את הפלט של התוכניות בסוף הריצה. מצד ימין זהו פלט בצד הלקוח ובצד ימין זהו הפלט בצד השרת:

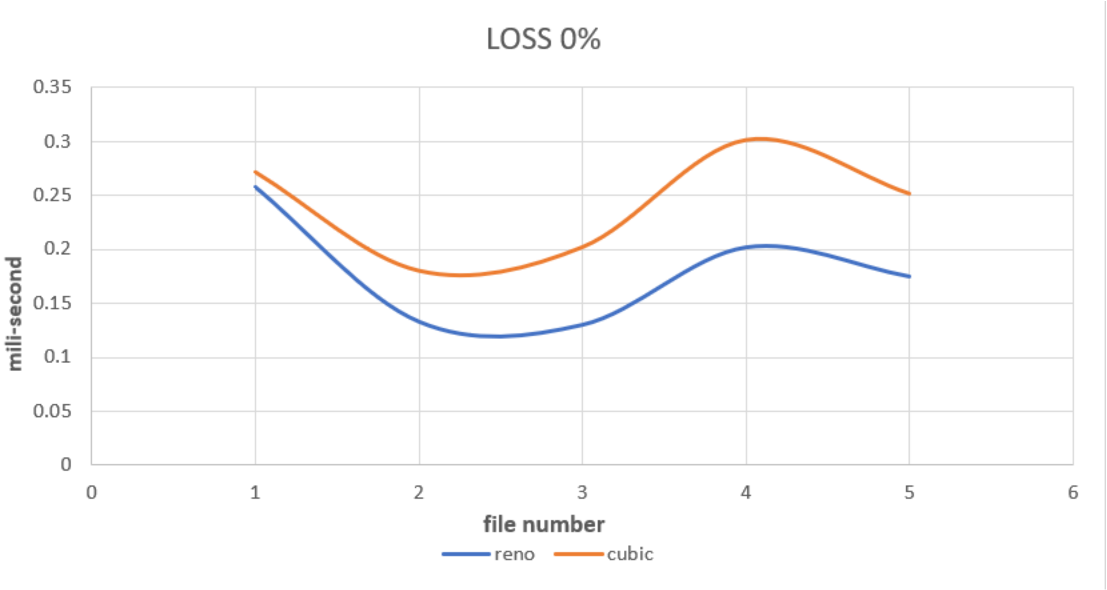


```
authentication has been sent!
CC algorithm has been changed to: "reno"
second message has been received!
CC algorithm has been changed to: "cubic"
first message has been received!
authentication has been sent!
CC algorithm has been changed to: "reno"
second message has been received!
CC algorithm has been changed to: "cubic"
exit message has been received!


###################

collected DATASET is:

time to receive part 1 of file 1 is 0.272000 [ms]
time to receive part 2 of file 1 is 0.258000 [ms]
time to receive part 1 of file 2 is 0.180000 [ms]
time to receive part 2 of file 2 is 0.133000 [ms]
time to receive part 1 of file 3 is 0.202000 [ms]
time to receive part 2 of file 3 is 0.130000 [ms]
time to receive part 1 of file 4 is 0.302000 [ms]
time to receive part 2 of file 4 is 0.202000 [ms]
time to receive part 1 of file 5 is 0.252000 [ms]
time to receive part 2 of file 5 is 0.175000 [ms]

###################

the times of avg part:
the average of sending first file - by "cubic" - is: 0.242 [ms]
the average of sending second file - by "reno" - is: 0.180 [ms]

###################
yoad@yoad-VirtualBox:~/Desktop/test$ 
```

```
y
CC algorithm has been changed to: "cubic"
first half has been sent!
waiting for authentication...
authentication match!
CC algorithm has been changed to: "reno"
second half has been sent!
send file again? (y/n)
y
CC algorithm has been changed to: "cubic"
first half has been sent!
waiting for authentication...
authentication match!
CC algorithm has been changed to: "reno"
second half has been sent!
send file again? (y/n)
y
CC algorithm has been changed to: "cubic"
first half has been sent!
waiting for authentication...
authentication match!
CC algorithm has been changed to: "reno"
second half has been sent!
send file again? (y/n)
y
CC algorithm has been changed to: "cubic"
first half has been sent!
waiting for authentication...
authentication match!
CC algorithm has been changed to: "reno"
second half has been sent!
send file again? (y/n)
n
exit message has been sent
yoad@yoad-VirtualBox:~/Desktop/test$ 
```

נסכם את התוצאות (הפלט הרלוונטי) בטבלה הבאה ונציג את גרף התוצאות:

אלגוריתם/מספר־קובץ	RENO	CUBIC
1	0.272[MS]	0.258[MS]
2	0.180[MS]	0.133[MS]
3	0.202[MS]	0.130[MS]
4	0.302[MS]	0.202[MS]
5	0.252[MS]	0.175[MS]



המסקנה המתקבלת ־ אלגוריתם ה־$reno$ מהיר יותר.