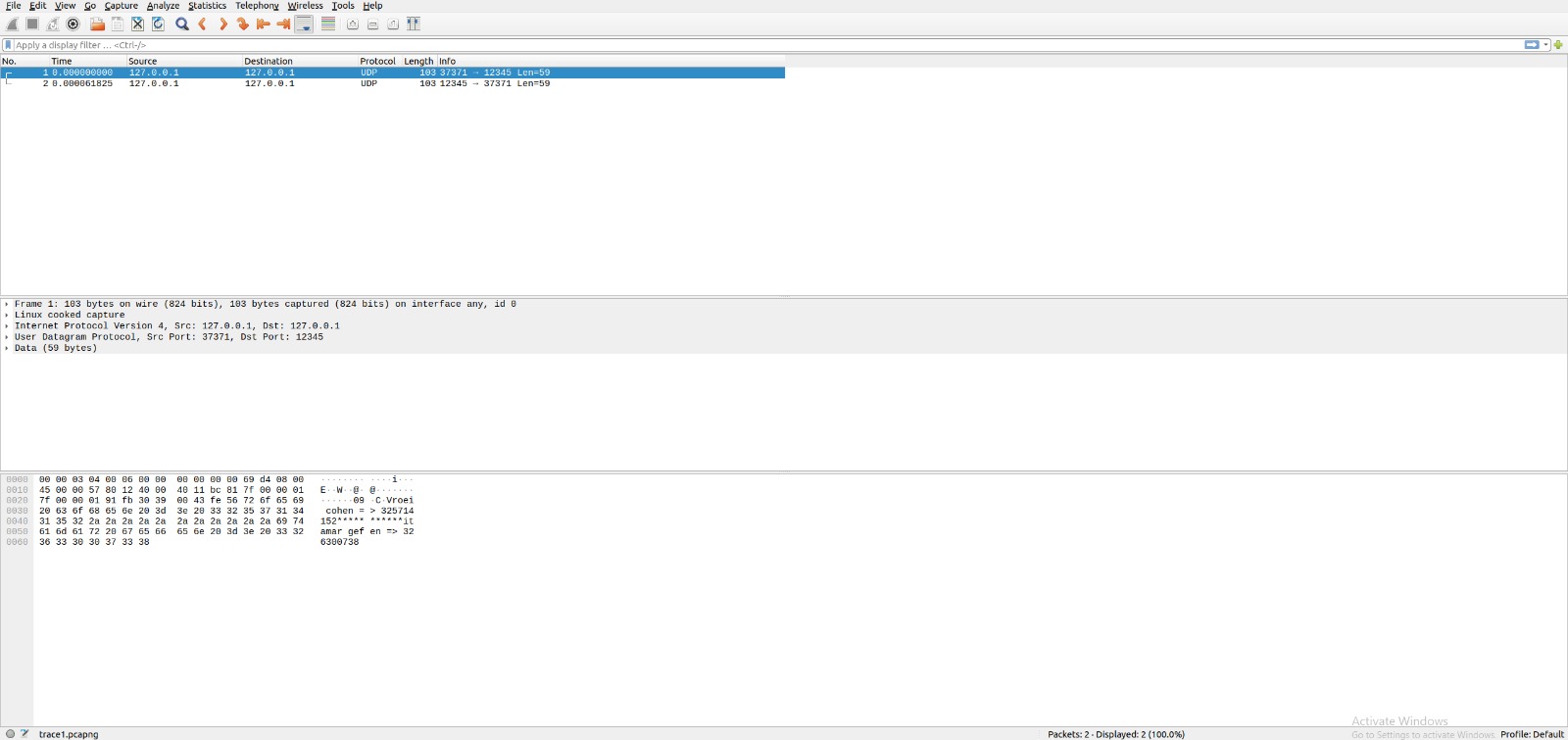
**ש.ב. 1**

איתמר גפן ורועי כהן

**חלק 1**

1 + תחילת 2) סיננו את התעבורה הרצויה בין הרשת והלקוח ויצאנו את התעבורה בקובץ תחת השם

"trace1.pcapng” שמצורף במודל, לאחר פתיחת הקובץ דרך wireshark יפתח המסך להלן:



2) **איך הסנפנו את התעבורה בין הלקוח לשרת?** פתחנו ווירשרק, בחרנו להאזין ל any, כלומר כל כרטיס רשת עם הפילטר

udp && udp.port == 12345

סינון זה מוודא שנסניף רק חבילות udp שעוברות דרך פורט 12345 שאנחנו עשינו לו bind בשרת שלנו.

3) השרת השתמש בפורט 12345. זה בא לידי ביטוי בקוד בפונקציה bind שקושרת את הסוקט לפורט.

בקוד של הלקוח אפשר לראות ששולחים לפורט 12345.

הלקוח לא עושה bind לפורט מסוים – הוא מקבל פורט כלשהו ממערכת ההפעלה, והשרת יכול לענות לו כי הלקוח רושם את מספר הפורט שלו בהודעה שהוא שולח לשרת בשכבת התעבורה.

הפורט משמש בשכבת התעבורה. הוא משמש כמזהה יחודי לסוקט (האובייקט שדרכו האפליקציה מקבלת ומעבירה הודעות). בעזרתו שכבת התעבורה בצד המקבל יודעת לאיזה סוקט (אפליקציה) להעביר את ההודעה.

**החבילה שהלקוח שלח לשרת:**

Text

Description automatically generatedכמו שנאמר לעיל, ניתן לראות בתמונה שבתחילית ששייכת לשכבת התעבורה מצוין שהלקוח שולח את החבילה לפורט מספר 12345 (דסטינייטיון פורט), ורושם גם את הפורט שלו (סורס פורט).

source port

הפורט שהלקוח קיבל ממערכת ההפעלה

destination port

הפורט של השרת, הלקוח שולח אליו את ההודעה

**החבילה שהשרת החזיר ללקוח:**

Table

Description automatically generated with medium confidence

ניתן לראות שבתחילית של שכבת התעבורה כתוב שהשרת שולח את החבילה לפורט מספר 36773 (דסטינייטיון פורט) – שזה מה שהיה הסורס פורט בהודעה הקודמת (שהלקוח שלח לשרת).

4) בגלל שהרצנו את הלקוח והשרת על אותו מחשב, הם שלחו אל ומכתובת ה-ip 127.0.0.1 . זאת כתובת מיוחדת שמשמעותה "אני". משתמשים בה לתעבורה בין אפליקציות שונות באותו מחשב – כשאין צורך להוציא את המידע החוצה. אפשר לראות את זה בחבילות שהסנפנו בווירשרק:

Graphical user interface

Description automatically generated with low confidence

Text

Description automatically generated**הפלט של ifconfig:**

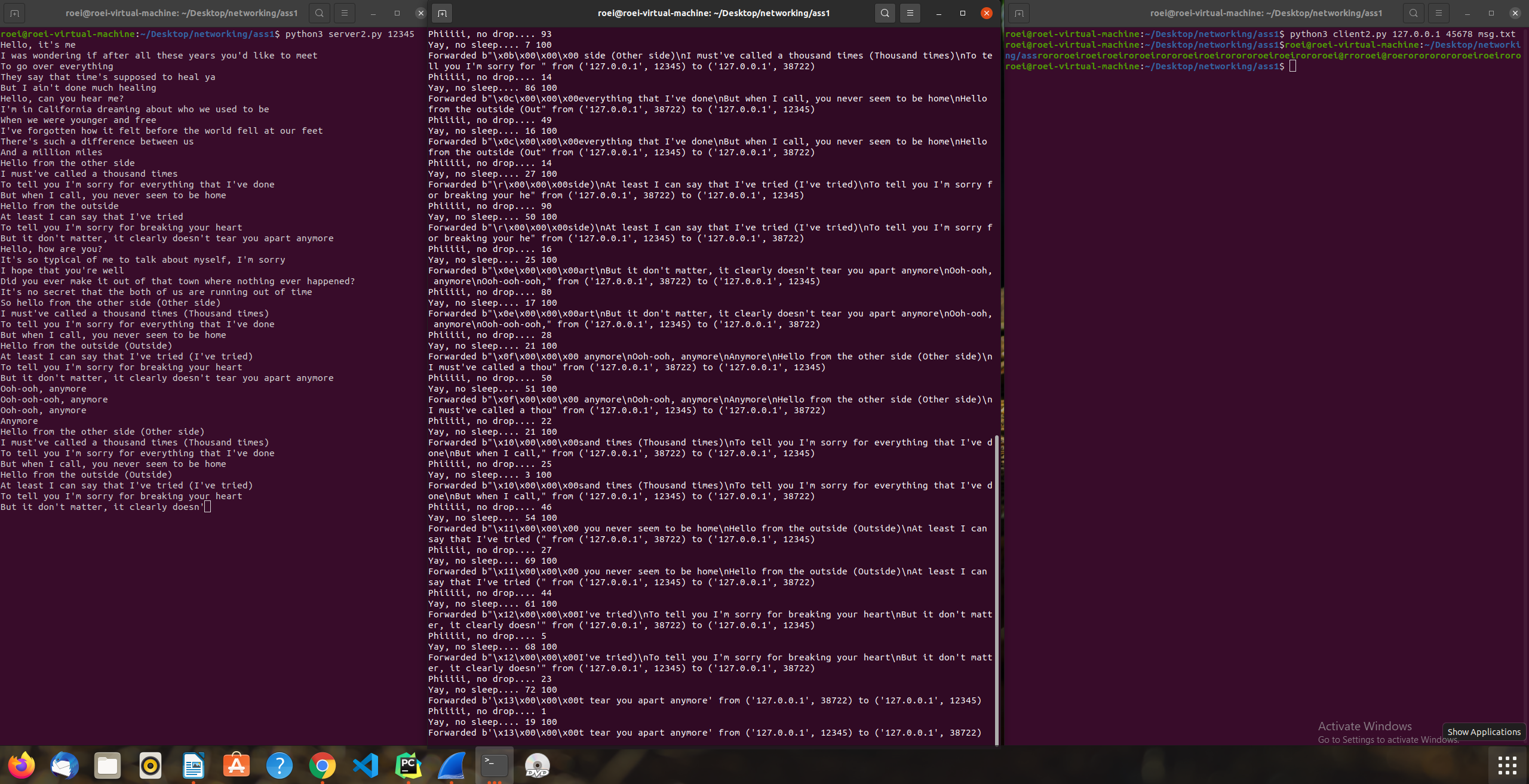
הפלט של הפקודה מציג 2 כרטיסים.

הראשון הוא כרטיס הרשת "האמיתי" שאיתו מתקשרים עם מחשבים אחרים, וה-ip שלו הוא 10.0.2.255

השני הוא כרטיס ה loopback שנועד לשליחת הודעות בתוך אותו מחשב, שבו השתמשנו (כיוון שהשרת והלקוח שניהם רצו על אותו מחשב), ואכן ה-ip שלו הוא 127.0.0.1

**חלק 2**

**הרצת משימה 1 – “playing nice”**

foo לא מעכב חבילות ולא זורק אותן, לכל חבילה שהוא מקבל הוא מוודא שגודלה אינו עולה על 100 בתים (אם כן הוא זורק אותה) ומעביר אותה לנמען. תצלום הרצה:

**A screenshot of a computer

Description automatically generated with medium confidenceהרצת משימה 2 – זריקת חבילות באקראיות “playing basketball”**

**A screenshot of a computer

Description automatically generated with medium confidenceהרצת משימה 3 – עיכוב חבילות "playing grandpa”**

**A screenshot of a computer

Description automatically generated with medium confidenceהרצת משימה 4 – הכל ביחד**

**איך מימשנו**

מימשנו עם stop&wait. כלומר, הלקוח שולח לשרת הודעה אחת בכל פעם ומחכה לתשובה. אם הוא לא מקבל את התשובה תוך הזמן שהגדרנו ל-timeout, הוא שולח שוב, ככה עד שהוא מקבל את התשובה.

בשביל שעיכובים של הודעות לא יגרמו לכך שהשרת לא ידפיס את אותה הודעה פעמיים, הגדרנו "פרוטוקול": הוספנו לתחילת כל הודעה תחילית עם המספר הסידורי של ההודעה (מתחיל ב- 1). ככה השרת יכול להדפיס הודעה רק אם המספר הסידורי שלה גדול ממספר ההודעות שהודפסו עד עכשיו. בדומה, כשהלקוח מקבל תשובה מהשרת הוא יכול לבדוק אם המספר שלה מתאים למספר ההודעה שלתשובה אליה הוא מצפה.

נדגים בעזרת צילומי מסך מההסנפות של ווירשארק: (השרת האזין בפורט 23456 ו- foo האזין בפורט 56789. במקרה הזה יצא שהלקוח האזין בפורט 49532. לכן, כשנראה בווירשארק למשל סורס פורט 23456 ודסטינייטיון פורט 56789 נדע שמדובר בהודעה שנשלחה מהשרת ל- foo.)

התעבורה בווירשארק: השרת האזין בפורט 23456 ו- foo האזין בפורט 56789. לכן השתמשנו בפילטר

udp && (udp.port == 12345 || udp.port == 45678)

כדי לסנן את כל התעבורה, ושמרנו בקובץ.

**מקרה 1** – foo לא עושה בעיות.

Graphical user interface, application

Description automatically generated

בשורה 1 (המודגשת) הלקוח שולח ל- foo את הודעה מספר 1. בשורה 2 foo מעביר אותה לשרת, בשורה 3 השרת שולח את התשובה ל- foo, ובשורה 4 foo מעביר את התשובה ללקוח.

הלקוח מקבל תשובה, ולכן בשורה 5 הוא כבר שולח ל- foo את הודעה מספר 2.

**Table

Description automatically generatedמקרה 2** – השרת זורק/מעכב הודעה

בשורה 10 (המודגשת) הלקוח שולח ל- foo את הודעה מספר 3. foo זורק/מעכב את ההודעה (זה לא ממש משנה לצורך העניין), ולכן עוברת שנייה, זמן ה- timeout שהגדרנו, והלקוח עדיין לא קיבל תשובה. לכן, בשורה 11 שולח את ההודעה מספר 3 שוב. אותו דבר קורה ככה עד שורה 15. Graphical user interface, text, table

Description automatically generated with medium confidence בשורה 15 foo סוף-סוף מעביר לשרת, ובשורות שלאחר מכן ניתן לראות שהשרת עונה ל- foo ובשורה 17 foo מחזיר את התשובה ללקוח.

Graphical user interface, table

Description automatically generated

בשורה 18 הלקוח כבר ישלח את הודעה מספר 4.

**מקרה 3** – foo מעכב הודעה של הלקוח, וכתוצאה מכך היא מגיעה לשרת אחרי שהוא כבר הדפיס אותה. כלומר, השרת מקבל הודעה עם מספר שקטן ממספר ההודעה שאליו הוא מצפה. במקרה זה השרת לא ידפיס אותה שוב, אלה רק ישלח אותה בחזרה ללקוח (כדי שאם הלקוח לא קיבל את התשובה בפעם הקודמת הוא יקבל עכשיו).

Table

Description automatically generated

בשורה 45 (המודגשת) הלקוח שולח בפעם הראשונה את הודעה 7 ל- foo. בשורה 46 foo מעביר אותה לשרת. השרת מדפיס אותה, ובשורה 47 הוא שולח את התשובה אליה ל- foo.

Table

Description automatically generated

כפי שניתן לראות, בשורה 48 השרת פתאום מקבל מ- foo את הודעה 5. השרת לא ידפיס אותה, אלה רק ישלח בחזרה תשובה עליה (שורה 49).

Table

Description automatically generated

ניתן לראות ש- foo זרק/עיכב את התשובה שהשרת שלח על הודעה 7, ולכן (בשורה 50) הלקוח שולח אותה שוב. בשורה 53 הלקוח כבר מקבל את התשובה עליה.

באופן דומה, אם foo מעכב תשובה של השרת, וכתוצאה מכך היא מגיעה ללקוח כשהוא כבר מצפה לתשובה על הודעה אחרת - כלומר הלקוח מקבל תשובה עם מספר שקטן מהמספר שאליו הוא מצפה, אז הלקוח יתעלם מהתשובה הזאת. הוא ימשיך לחכות לתשובה המתאימה.