מטלת סיום בתקשורת ומחשוב:

מגישים:

ניב קוטק - 20823615 .

אביב שמח - 313131476.

תאריך הגשה לועזי: 05.03.2022.

תאריך הגשה עבר: ב׳ באדר ב׳ ה׳תשפ״ב.

יום הגשה: שבת.

שם המרצה: ד"ר דביר עמית זאב.

מחלקה: מדעי הטבע במחלקת מדעי המחשב ומתמטיקה.

מספר קורס: 7028310-3.

קבצים:

1. קובץ pdt
2. מחלקה של פייתון
3. 3 קבצים של ווייר שארק

תוכן עניינים:

שאלות ותשובת

שאלה: מה ההבדל בין http 1.0 ,http 1.1, http 2.0, QUIC.

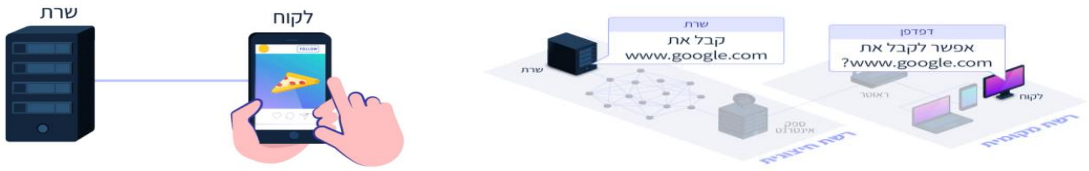
תשובה:

הגדרה אפליקציה: כינוי לתוכנה שבה אנו עושים שימוש במחשב , סמארטפונים, טאבלטים. אפליקציות עושות שימוש בתקשורת אינטרנט על מנת לשלוח ולקבל הודעות.

פרוטוקול(protocol): הוא אוסף כללים שיחד יוצרים שפה משותפת או נוהל התקשרות בין הצדדים המעורבים בתקשורת. תפקידו להסדיר את העברת המידע בין הצדדים.

הגדרה שכבת האפליקציה (Application Layer):היא אוסף הפרוטוקולים בהם עושות האפליקציות שימוש באופן ישיר, והיא מספקת הפשטה מעל תקשורת הנתונים ברשת האינטרנט

לשכבה זו קיימים פרוטוקולים רבים שבהם HTTP (פרוטוקול הגלישה באינטרנט), SMTP(פרוטוקול דואר),FTP (פרוטוקול העברת קבצים), ועוד.



הגדרת HTTP: Hyper Text Transfer Protocol פרוטוקול של רמת האפליקציה בו משתמשים בweb.-הוא מדמה מודל של שרת לקוח ,כאשר הלקוח הוא הדפדפן אשר מבקש, מקבל ומציג אובייקטים, והשרת הוא שרת ה web-ששולח את האובייקטים ללקוח על פי בקשתו .פרוטוקול זה משתמש תמיד ב, TCP-כאשר מספר ה-portשאליו מתחבר הלקוח הוא 80.

ישנם 3 סוגים של חיבורי HTTP:

HTTP 1.0(Non-persistent Http): יצירת חיבור TCP,שליחת אובייקט אחד וסגירת החיבור. שיטה זו לא יעלה עקב פתיחה וסגירה של החיבור. בנוסף לכך יכו לגרום לעיכובים רבים ברשת.

HTTP 1.1(persistent Http): יצירת חיבור TCP ,שליחת כמה אובייקטים ביחד וסגירת החיבור. שיטה זו יותר יעילה ומשפרת את זמן הגלישה.

לגישה זו יש 2 תתי-גישות:

1. persistent - הלוקח שולח בקשה חדשה רק לאחר שקיבל תשובה עבור הבקשה הקודמת.

2. persistent pipeline – הלקוח שולח בקשה חדשה מבלי לחכות לתשובה על הבקשה הקודמת.

HTTP 2.0: הפרוטוקול מבוסס על פרוטוקול הרשת SPDY שפותח על ידי גוגל. לפרוטוקול הנ"ל ישנם מספר שיפורים:

1.מחלקת את האובייקטים ל-frames וכך התעבורה עוברת מהר יותר.

2. מולטיפלקסינג (MULTIPLEXING)– היכולת של השרת והלקוח להשתמש בחיבור TCP אחד בשביל להעביר מספר אובייקטים.

3. תעדוף ותלות (Dependency, Prioritization) –יכולת לתת תעדוף של משאבים שונים על פני אחרים, על מנת לייעל את תהליך זרימת הנתונים.

4. דחיפת קבצים על ידי השרת (Server Push) –דחיפת קבצים למטמון הדפדפן לפני הבקשה על מנת לייעל את הזמן.

QUIC(Quick UDP Internet Connection): הוא פרוטוקול המפותח על גבי UDP . הפרוטוקול מאפשר חיבור מאובטח ומהיר בין הלקוח לבין האתר . על פי גוגל, פרוטוקול זה גורם לעמודי אינטרנט להטען 75% מהר יותר

מאשר בחיבור מעל TCP.

שאלות ותשובת

שאלה: למה צריך מספרי port:

תשובה:

מערכות הפעלה מודרניות תומכות בריבוי משימות ,כלומר הן מאפשרות למשתמש להפעיל בו-זמנית כמה יישומים. זיהוי מחשב היעד באמצעות כתובת או שם תחום אינו מספיק כדי לזהות את תהליך היעד כאשר מפעילים כמה יישומי תקשורת. לכן כדי לזהות תהליכים אלו קובעים לכל אחד מספר שנקרא מפתח (port). הport הוא מזהה של תהליך, והוא מאפשר לשכבת התעבורה לנתב את המנות אל תהליך היישום המתאים(תוכנות שונות עובדות עם פורטים שונים). כלומר כתובת ה-IP מנתבת את המידע למכשיר הנכון ובאמצעות פנייה לפורט מסוים בבקשה, השרת יכול לדעת לאיזו תוכנה אנו פונים . לדוגמה, אם נשלח הודעה לפורט מספר 80 (באמצעות פרוטוקול,TCP) השרת צפוי להבין שאנו פונים לתוכנת ה HTTP- ולא לתוכנה המייל, מכיוון שתוכנת ה HTTP-מאזינה על פורט 80. כמו כן,קיימים 65,536 מספרי פורט. כאשר מתכנתים כותבים אפליקציה, הם בוחרים את מספר הפורט שייעשה בו שימוש, כלומר מספר הפורט שבו יאזין השרת, ושאליו יפנו הלקוחות

Well known ports הוא כינוי למספרי פורט בטווח 0–1023 השמורים לשרתים שעובדים בפרוטוקולים מוכרים וחשובים.

דוגמאות לפורטים חשובים:

20,21 - פרוטוקול העברת קבצים (FTP)

25 - דואר יוצא (SMTP)

53 - פרוטוקול DNS

67,68 - פרוטוקול הקצאת כתובות דינמית (DHCP)

80 - פרוטוקול העברת דפי אינטרנט HTTP

110 - דואר נכנס (POP3)

161,162 - פרוטוקול SNMP

443 - פרוטוקול HTTPs להעברת דפי אינטרנט מאובטחים

1433 - פרוטוקול גישה לשרתי SQL

מזהה הבניין הוא כתובת IP ,ומזהה הדירה נקרא פורט

מזהה הבניין:
הרצל 1, תל אביב

שאלות ותשובת

שאלה: הסבירו מה זה CRC

תשובה:

Cyclic Redundancy Check הוא בקרת שגיאות הנקבע בהתאם לפרוטוקול השידור שהוסכם בין שני המתקשרים. ה crc מבוסס על קוד מחזורי שמקודד הודעות ע"י הוספה של ערך בדיקה בעל גודל קבוע. הוא מבוסס על האיזומורפיזם שבין וקטורים לפולינומים (מסתכלים על כל וקטור באורך N כפולינום שמקדמיו הם קואורדינטות הווקטור).

כלומר ה crc:

- מאפשר גילוי שגיאות, בודדות או רבות
- לא מאפשר תיקון שגיאות
- מוגדר ע"י פולינום יוצר כלשהו
- מספר סיביות הביקורת הן כמעלת הפולינום היוצר

אופן פעולה:

בהינתן פולינום יוצר מדרגה r והודעה M:

1. הוספה של r אפסים מימין להודעה.
2. חילוק בפולינום תוך שימוש בחילוק של מודלו 2.
3. חיסור השארית תוך שימוש ב xor.
4. צירוף התוצאה מימין להודעה המקורית
5. שליחת ההודעה.

הצד המקבל יבצע את שלבים 1 ו-2 ויוודא ש-r הביטים האחרונים שנשלחו זהים לתוצאה שהתקבלה.

לדוגמא לקוד CRC עם הפולינום היוצר $G(x) = x^7 + x^5 + x^3 + 1$ נזדקק ל7 סיביות.

את מילת המידע המקורית נסמן ב(M(x. נוסיף מימין מספר אפסים כדרגת הפולינום היוצר (r) ונקבל $x^r M(x)$. את חישוב זה נחלק בפולינום היוצר מודולו 2. את השארית (שבהכרח קטנה ממעלת הפולינום היוצר) נחסר מהמילה ונקבל את מילת הקוד (T(x, שאותה נשדר.

שאלות ותשובת

שאלה: מה זה subnet ולמה צריך את זה?

תשובה:

חלוקה של רשת לתתי רשתות .זה תת-רשת שלא עוברת דרך ראוטר אלה דרך מתג, כלומר בין מחשבים באותו התת-הרשת אין להם צורך להשתמש בראוטר כדי לתקשר אחד עם השני. כך לדוגמא אם מחובר לאינטרנט הביתי עם המחשב ושולח הודעה לאדם נוסף המחובר באותו רשת ביתית אז נתקשר תחת אותו subnet.

מדוע צריך:

1. כדי שמידע יעבור ממחשב למחשב, חשוב להבחין: האם המידע עובר בין שני מחשבים הנמצאים ברשת מקומית , כמו רשת ביתית או ארגונית, או בין שני מחשבים הנמצאים ברשתות שונות

2.הסיבנוט נועד לחלק את הרשת בהתאם לצרכים שלנו.-כך לדוגמא חברה יכולה להגדיר תנאים וחוקים שונים לכל רשת .

3. הקטנה משמעותית על עומס ברשת- אם הרשת הייתה מחולקת רק באמצעות נתבים העומס היה הופך להיות גדול. לכן ורק packets שמיועדים לרשתות אחרות יורשו לצאת מחוץ לרשת הביתית דרך הנתבים.

4. חיבור זול יותר של רשתות – אין צורך בכמות גדולה של נתבים וחיבורים לרשתות אחרות.

5. הקטנת טבלאות הראוטינג – אם כל נתב באינטרנט היה צריך לדעת על קיום של כל רשת קיימת , אז טבלאות הראוטינג היו הופכות להיות גדול ובכך זמן הריצה שלהם הופך להיות גדול אשר משפיע על העומס ברשת.

6. טיפול בשגיאות – בעזרת חלקות הרשת לתתי רשתות נוכל לטפל ולזהות שגיאות מקומיות וטפל בהן באופן פרטני.

איך ניתן לזהות:

צורת הסימון הנפוצה לסימון מזהה רשת היא Subnet Mask, או בעברית: מסכת רשת.

Subnet Mask היא שורה של ארבעה מספרים שבה כל מספר יכול לקבל את הערך 0 או 255. הספרות בכתובת ה-IP המופיעות במיקום המקביל למיקום של הערך 255 ב-Subnet Mask מגדירות את מזהה הרשת, ואילו הספרות המופיעות במיקום המקביל למיקום של הערך 0 מגדירות את מזהה הישות.

1. **מזהה רשת**: **Network ID** – לאיזו תת-רשת (subnet) שייכת כתובת ה IP-האם זו הרשת של הבית, של בית הקפה או אולי של מקום העבודה?
2. **מזהה ישות**: **Entity ID** – לאיזה כרטיס רשת (Network Interface Controller) בתוך תת-הרשת שייכת הכתובת? האם זהו כרטיס הרשת של המחשב שלי או של אדם אחר שגולש באותה רשת.

כעת נוכל להפריד את שני חלקי הכתובות:

שאלות ותשובת

שאלה: למה צריך כתובות mac למה לא מספיק לעבוד עם כתובות ip

תשובה:

כתובת ip:

מהי כתובת IP?

Ip קיצור של Internet Protocol הוא אמצעי הזיהוי, הכתובת, של כל שרת אינטרנט ושל כל ציוד קצה שמחובר לאינטרנט. כתובת IP היא ייחודית ומוקצית בדרך כלל באופן אוטומטי. בתקשורת בין שרת ולקוח, כתובות IP מוצמדות לכל בקשה ותגובה שעוברת ביניהם. IP, מורכבת מרצף של ארבעה מספרים ובינהם נקודות. כל מספר נמצא בטווח שבין 0 ל-255.

מהי כתובת mac (Media Access Control Addresses):

בכל ציוד תקשורת יש כרטיס רשת אשר במהלך הייצור מוטבעת עליו כתובת MAC אשר ייחודית ובלתי משתנה. כתובת ה- MAC מורכבת משישה צירופים של זוגות אותיות (A-F) ומספרים, המופרדים ביניהם. הכתובת מחולקת לשני חלקים מזהה יצרן ומזה ישות.

מדוע צריך גם mac וגם ip :

בשכבת הקו המידע עובר בין שתי תחנות סמוכות המחוברות זו לזו, בעזרת כתובות MAC.

לפקטה מוצמדות 3 כתובות: כתובת MAC ושתי כתובות IP של מחשב המקור ושל היעד הסופי של הפקטה. כתובות IP לא משתנות במהלך הדרך אותה עושה הפקטה עד למחשב היעד. לעומת זאת כתובת mac שהיא הכתובת של הראוטר הבא בדרך אל היעד משתנה. כל ראוטר שמקבל את הפקטה משנה את הכתובת ה mac לזו של הראטור הבא עד שהפקטה מגיעה לכתובת ה mac של כרטיס הרשת , אשר מגלה כי הכתובת ה ip של היעד שייכת למחשב זה.

	IP Adress	MAC Address
כינוי בעברית	כתובת לוגית	כתובת פיזית ✓
קבועה/ משתנה	יכולה להשתנות ✓	קבועה
איפה היא שמורה	במערכת ההפעלה של המחשב ✓	צרובה פיזית על כרטיס הרשת ✓
מי קובע אותה	אחראי רשת או בעל המחשב ✓	נקבעת בעת הייצור
משמעות הכתובת	מלמדת על הרשת אליה המחשב מחובר	מלמדת על היצרן שייצר את הכרטיס ✓

שאלות ותשובת

שאלה: מה ההבדל בין NAT , Router, Switch.

תשובה:

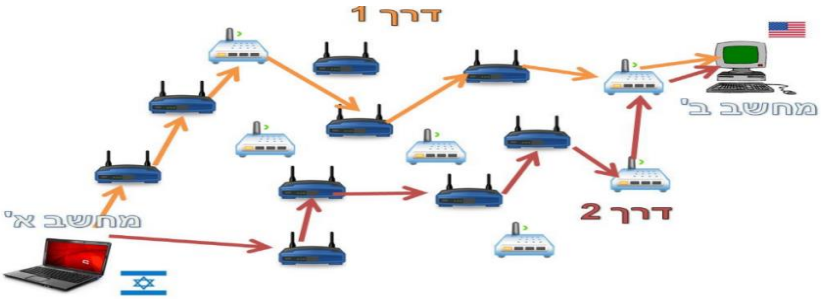
מתג (Switch): הוא רכיב ברשת מחשבים (הנמצא בשכבת הקו) המחבר בין צמתים שונים ברשת, בין אם הם מכשירי קצה ובין אם הם מרכיבי רשת בסיסיים. ה-Switch מכיר כתובות MAC, מבין את המבנה של מסגרות בשכבה הקו ויודע ל חשב Checksum. לאחר שה-Switch למד את הרשת, הוא מעביר מסגרת מהפורט בה הוא קיבל אותה אל הפורט הרלוונטי.

.

נתב (Routers) – נתב(מלשון לנתב) משמש לניתוב נתונים מרשת אחת לרשת אחרת.

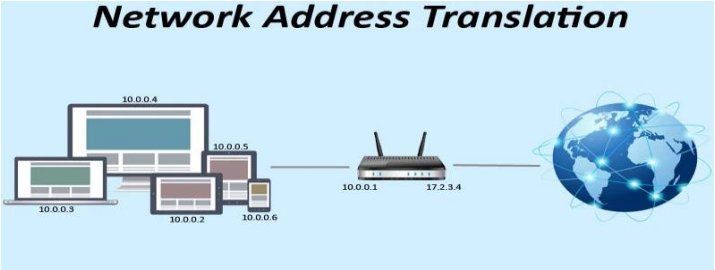
תפקידה של שכבת הרשת הוא למצוא את המסלול הטוב ביותר מאיתנו אל היעד ובחזרה.

שכבה זו לא מתעסקת בתקשורת בין ישויות סמוכות , אלא אחראית על המסלול בין שתי נקודות קצה. ניתוב המידע מתבצע על -ידי רכיבים המכונים Router(נתבים) , אשר מנתבים את הפקטות בין הרשתות השונות. כך יכולה פקטה לצאת מקו אחד, לעבור דרך מספר קווים שונים ולבסוף להגיע אל הרשת אליה היא מיועדת. את הניתוב של הפקטה הנתב עושה על ידי טבלת ניתוב.הטבלה הכוללת מזהי רשת ולאן להעביר חבילות המיועדות למזהי רשת. ברוב המקרים, הטבלה היא דינאמית ועשויה להשתנות בהתאם למצב הרשת



NAT (Translation Address Net)- ממיר כתובת פנימית לכתובת חיצונית.

לכל התקן רשת ב subnet -יש כתובת IP פנימית (בדויה) שמוכרת רק לראוטר ברשת שלה. כאשר יש צורך לשלוח פקטה מחוץ לרשת הפנימית נשתמש ב NAT אשר תחליף את הכתובת הפנימית של הישות בכתובת חיצונית. דבר זה שימושי לצורך צמצום כתובות ה-IP, בכך שבמקום שלכל מחשב תינתן כתובת IP חיצונית, כל המחשבים מיוצגים ככתובת אחת בלבד (כתובת חיצונית של ה NAT)



שאלות ותשובת

שאלה: שיטות להתגבר על המחסור בIPv4 ולפרט.

תשובה:

מבנה ה-IP הנפוץ כיום הוא IPv4 (קיצור של IP version 4). במבנה זה כל כתובת IP מורכבת מרצף של ארבעה מספרים וביניהם נקודות. כל מספר נמצא בטווח שבין 0 ל-255. עבור כל מספר בכתובת קיימות 256 אפשרויות (בין 0 ל-255).

בעבר מספר זה נראה עצום, אך כיום כבר קיים מחסור בכתובות IP.

הפתרון:

1. ליצור גירסה חדשה של פרוטוקול IP , שתתמוך בהרבה יותר כתובות מאשר 2^32 הכתובות ,ע"י מעבר למבנה כתובת IP שבאפשרותו לייצג 2^128 כתובות. למבנה החדש קוראים IPv6, כלומר IP version 6. בדרך זו אפשר לייצר כמות עצומה של כתובות IP.

IPv6		IPv4
1999	שנת השקה	1989
128 ביט	גודל	32 ביט
סימון הקסדצימלי 2001:0DB8:0234:AB00:0123:4567:8901:ABCD	פורמט	סימון עשרוני 192.0.2.0
2^{128}	מס' כתובות	2^{32}

2. ראוטר NAT ממיר כתובת פנימית לכתובת חיצונית - עד אשר החל השימוש ב-NAT- היה צורך לספק כתובת IP יחודית לכל אחת מהישויות ברשת. דבר זה חיובי מהרבה בחינות , אך במציאות בה אין כבר כתובות IP לתת .אי לכך, נוצר הרעיון של.NAT לפי רעיון זה , כל הישויות בתוך הרשת יקבלו כתובות פרטיות – כלומר כתובות שיזהו אותן בתוך הרשת בלבד , ולא בעולם החיצוני ואם ישות רוצה לשלוח פקטה לישות מרשת אחרת, הנט תחליף את הכתובת הפרטית לכתובת חיצונית ותשלח את החבילה ברשת. בצורה זו , הרשת "מבזבזת" רק כתובת IP אחת – זו של הנתב שלה, ולא n(כמספר הישויות המחברות באותו הרשת) כתובות כמו שהיא הייתה צורכת לפני השימוש ב -NAT.

שאלות ותשובת

שאלה: בהינתן מחשב חדש המתחבר לרשת אנא תארו את כל ההודעות שעוברות החל מהחיבור

הראשוני ל switch ועד שההודעה מתקבלת בצד השני של הצאט. אנא פרטו לפי הפורמט הבא:

a .סוג הודעה, פירוט הודעה והשדות הבאים:

i .כתובת IP מקור/יעד, כתובת פורט מקור/יעד, כתובת MAC מקור/יעד, פרוטוקול שכבת התעבורה.

תשובה:

א. המחשב החדש שולח broadacast (חבילת discovery) לכל המחשבים ברשת המקומית על מנת לאתר שרת DHCP

Discovery:
Ethernet: source=new computer's MAC; destination=FF:FF:FF:FF:FF:FF
IP: source=0.0.0.0; destination=255.255.255.255
UDP: source port=68; destination port=67

ב. אם קיימים שרתי DHCP ברשת המקומית בעלי כתובת פנויה לחלוקה, הלקוח יקבל חבילת **offer** עם כתובת IP מכל אחד מהם.

Offer:
Ethernet: source=DHCP's MAC; destination=new computer's MAC
IP: source=DHCP_ip; destination=255.255.255.255
UDP: source port=67; destination port=68

ג. הלקוח שולח חבילת **request** עם הנתונים אותם בחר, גם כן בשידור - broadcast על מנת לעדכן את כל השרתים בכתובת שנבחרה.

:Request
Ethernet: source=new computer's MAC; destination=FF:FF:FF:FF:FF:FF
IP: source=0.0.0.0; destination=255.255.255.255;[a]
UDP: source port=68; destination port=67

ד. השרת שולח - **acknowledge** אישור שהוא קיבל את הבקשה.

:Acknowledment
Ethernet: source=DHCP's MAC; destination=new computer's MAC
IP: source=DHCP_ip; destination=255.255.255.255
UDP: source port=67; destination port=68

שאלות ותשובת

שאלה:

נתונה הרשת הבאה.

a. AS2, AS3 מריצים OSPF

b. AS1, AS4 מריצים RIP

c. בין ה-Ass רץ BGP

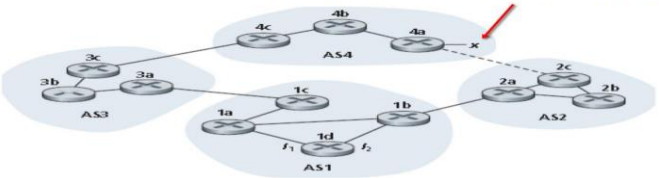
d. אין חיבור פיזי בין AS2, AS4

e. בעזרת איזה פרוטוקול לומד הנתב 3c על תת רשת x

f. בעזרת איזה פרוטוקול לומד הנתב 3a על תת רשת x

g. בעזרת איזה פרוטוקול לומד הנתב 1c על תת רשת x

h. בעזרת איזה פרוטוקול לומד הנתב 2c על תת רשת x



תשובה:

OSPF (open shortest path first) הוא פרוטוקול ניתוב היררכי אשר משתמש באלגוריתם ניתוב של תלוי מצב (Link-state). בבסיס הפרוטוקל עומד אלגוריתם דייקסטרה.

נתב המנתב חבילות בהתבסס על OSPF מנהל רישום של כל הנתבים שהוא "מכיר", והנתיבים אליהם. כאשר מגיעה אליו חבילה הוא מעביר אותה אל נתב היעד דרך הנתיב בו עלות התעבורה היא הזולה ביותר. נתבים שונים המשתמשים ב-OSPF מחליפים ביניהם רשימות על מנת להישאר מעודכנים לגבי שינויים בטופולוגיית הרשת, ולגלות נתיבים חדשים

RIP (Routing Information Protocol) - הוא פרוטוקול ניתוב המתבסס על ספירת צעדים (Hop Count) על מנת לבצע החלטות ניתוב.

נתב המשתמש ב־RIP מנהל רישום של כל הנתבים אותם הוא "מכיר", הרשתות המחוברות אליהן, וכמות הצעדים בכל נתיב לכל יעד. כאשר מגיעה חבילה אל הנתב הוא יעביר אותה בנתיב בו היא תעבור מינימום צעדים עד לרשת היעד, שיטה זו מכונה ספירת צעדים (hop count). הנתב מבקש עדכונים לגבי שינויים בטופולוגית הרשת מהנתבים המחוברים אליו כל שלושים שניות, וכך הוא נשאר מעודכן לגבי שינויים בנתיבים המובילים אל היעד, ומקבל מידע על נתבים חדשים שחוברו אל הרשת.

BGP (**Border Gateway Protocol**):

הוא פרוטוקול ניתוב המהווה את ליבת מערכת הניתוב של רשת האינטרנט. BGP הוא פרוטוקול ניתוב מבוסס קשר (Path-vector), המתפקד בשכבת היישום של מודל ה-OSI וכן בשכבת היישום של מודל ה-TCP/IP. נתב שפועל באמצעות BGP מנהל טבלה של הרשתות המחוברות אליו, והקשרים ביניהן לבין רשתות אחרות, ומבצע החלטות ניתוב על בסיס הקשרים בין הרשתות ומדיניות המוכתבת בצורה ידנית על ידי מנהל הרשת.

לכן התשובה:

e. כדי ש C3 ידבר עם x הוא צריך לעבור דרך c4 ברשת AS4 בפרוטוקול BGP

f. כדי ש a3 ידבר עם x הוא צריך לדבר עם c3 באותה רשת AS3 בפרוטוקול OSPF

g. כדי ש c1 ידבר עם x הוא צריך לדבר עם a3 ברשת AS3 בפרוטוקול BGP

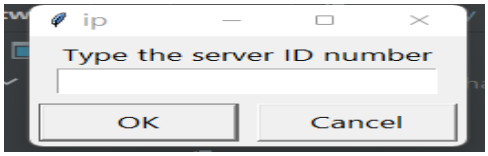
h. כדי ש c2 ידבר עם x הוא צריך לדבר עם a2 באותה רשת AS2 בפרוטוקול OSPF

ממשק גרפי

1. את ממשק הגרפי הן של השרת והן של הלקוח ניתן לראות בקבצי clientgui.py, servergui.py, בהתאמה.
2. הממשק הגרפי פותח על ידי ממשק של שפת התכנות פייתון לערכת התצוגה TK, ונחשב לכלי הסטנדרטי לבניית GUI בפייתון. השם Tkinter נובע מהצירוף Tk Interface, שכן הוא ממשק של ערכת Tk.
3. הממשק הגרפי מחלק ל2 חלקים: 1 – אתחול שדות. 2 – שימוש ע"י המשתמש.

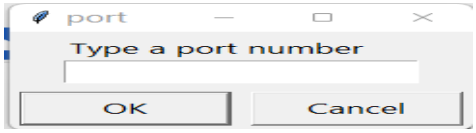
שרת:

חלק 1: אתחול שדות.



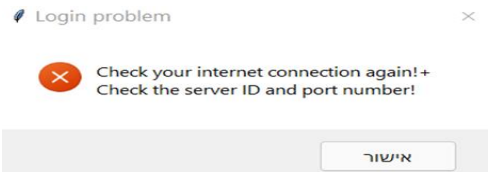
1א. נכתוב את מספר ip של השרת בתוך החלונית ip .

במידה ולא נכתוב ונלחץ על המשך – השדה יאתחל ל "0.0.0.0"



1ב. נכתב את מספר port של השרת בתוך החלונית port.

במידה ולא נכתוב ונחלץ על המשך – השדה יאתחל ל 5555



1 ג. במידה ולא נוצר קשר עם השרת נקבל חלונית הזהרה.

במידה ונרצה להמשיך , נצטרך להגדיר בשנית את השדות.

חלק 2: שימוש ע"י המשתמש.

בחלק זה תופיע תצוגה של חלון השרת עי אתחול השדות לפי חלק 1.

בחלונית קיים כפתור "stop server" אשר סוגר את חיבור השרת.

בנוסף בחלון הלבן של השרת מתקבלים מספר דיווחים:

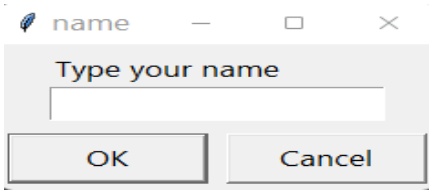
1. התחברות לקוח.
2. עזיבה לקוח.
3. בקשות לקוח.

יש לציין כי אם קיים ריבוי הודעות אזי קיים לשרת גלגלת הנמצא בצד ימין של המסך הלבן.

ממשק גרפי

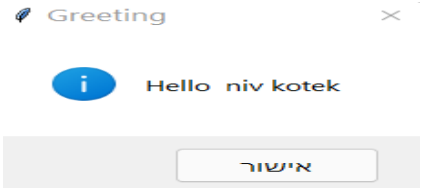
לקוח:

חלק 1: אתחול שדות.



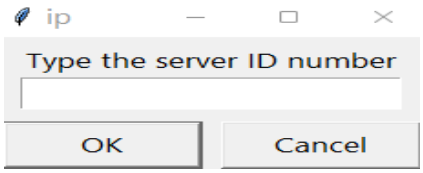
1א. נכתוב בחלונית ה name את השם.

חשוב לכתוב שם ייחודי לכל לקוח הנותן את המזהה הייחודי שלו.



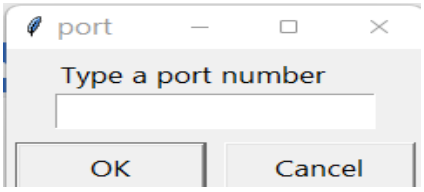
1ב. לאחר הקלדה שם הלקוח , נקבל את החלונית "Greeting".

החלונית מציגה הודעת ברכה קצרה עם שם הלקוח.



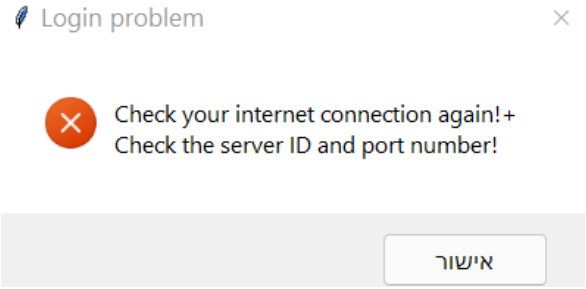
1ג. נכתוב בחלונית ip את שדה הip של השרת.

במידה ולא נאתחל את השדה, אזי השדה יקבל את המספר "127.0.0.1"



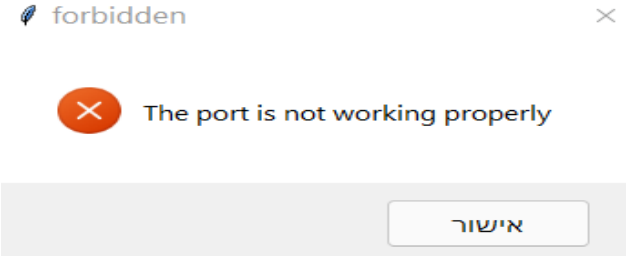
1ד. נכתוב בחלונית port את שדה ה port של השרת.

במידה ולא נאתחל את השדה, אזי השדה יקבל את המספר 5555.



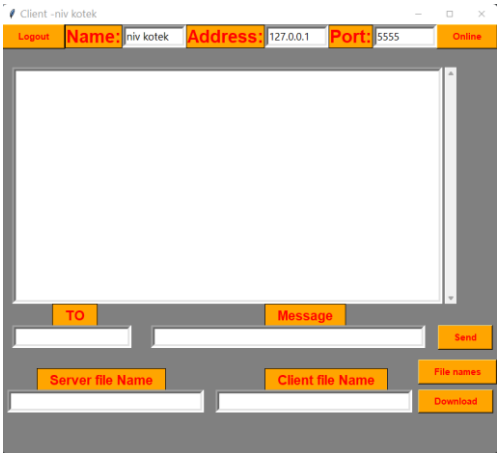
1ה. במידה ולא נוצר קשר עם השרת נקבל חלונית הזהרה.

במידה ונרצה להמשיך , נצטרך להגדיר בשנית את השדות.



1ו. במידה ולא בחרנו port תקין אזי נקבל חלונית הזהרה.

ממשק גרפי



חלק 2: שימוש ע"י המשתמש.

בחלק זה תופיע חלון של הלקוח לאחר אתחול השדות בחלק 1.

עליון:

כפתור Logout: זהו כפתור אשר שולח לשרת בקשה לניתוק הלקוח.

לאחר לחיצה על כפתור זה מתנתק הקשר בין לקוח והשרת ונסגר חלון הלקוח.

Name: כותרת אשר מציינת לאחר מכן את שם הלקוח.

Address: כותרת אשר מציינת לאחר מכן את ip של השרת.

Port: כותרת אשר מציין לאחר מכן את מספר port של השרת.

Online: כפתור אשר שולח לשרת בקשה לקבל את כל הלקוחות המחוברים

לשרת. לאחר לחיצה על כפתור זה נקבל רשימת כל הלקוחות המחוברים כעת.

אמצע:

ניתן לראות את החלון הלבן אשר עליו מתקבלים כל הדיווחים והבקשות בין הלקוח והשרת.

כמו כן נתן למצוא גלגלת בצד ימין של החלון הלבן למקרה בו קיים ריבוי הודעות.

תחתון:

TO: כותרת אשר מתחת לכותרת הנ"ל ניתן לכתוב למי תרצה לשלוח את ההודעה.

במידה ולא נכתוב למי נרצה לשלוח את ההודעה אזי ההודעה תשלח לכל האנשים המחוברים כעת.

Message: כותרת אשר מתחת לכותרת הנ"ל ניתן לכתוב את תוכן ההודעה לשליחה.

Send: זהו כפתור אשר שולח לשרת את תוכן ההודעה ולמי לשלוח את ההודעה.

(לאחר עדכון של השדות TO, Message)

File names: זהו כפתור אשר שולח בקשה לשרת לקבל את כל הקבצים נמצאים אצל השרת.

לאחר לחיצה על כפתור זה נקבל את רשימת כל הקבצים (כולל סיומת).

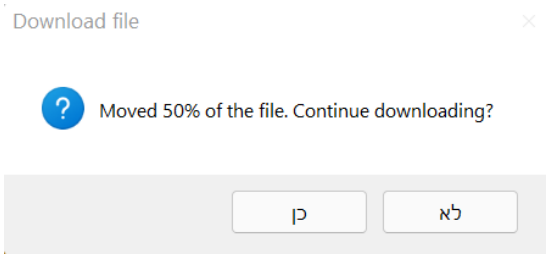
כמו כן את הקבצים של השרת ניתן למצוא בתיקה file אשר נמצא בzip

Server file Name : זהו כותרת אשר מתחת לכותרת הנ"ל ניתן לכתוב את שם הקבוץ (כולל סיומת) אותו נרצה להוריד מהשרת.

Client file Name : זהו כותרת אשר מתחת לכותרת הנ"ל ניתן לכתוב את שם הקובץ החדש אותו נרצה לשמור אצל הלקוח לאחר הורדה מהשרת.

Download : זה כפתור אשר שולח בקשה לשרת להוריד את הקבוץ משדה Server file Name ולשמור אותו בשם לפי השדה Client file Name. לאחר הורדה מהשרת ללקוח ניתן לראות את הקובץ החדש בתיקיית ה client.

ממשק גרפי



הורדה:

לאחר לחיצה על ההורדה תופיעה לאחר פרק זמן חלונית אשר תבדוק עם הלקוח האם ירצה להמשיך להוריד את הקובץ לאחר קבלה של 50% מהקובץ. במידה והלקוח לחץ על "yes" אזי ההורדה תמשיך. במידה והלקוח לחץ על "No" אזי ההורדה תיפסק.

סקלת ההורדה – לאחר 50% תופיעה בחלון הלקוח סקלה המציינת כי ירד 50% מהקובץ.

מספר הבית:

לאחר כ 50% מהקובץ תופיע אצל הלקוח (בחלון הלבן) את מספר הבית האחרון אשר נשלחה ללקוח.

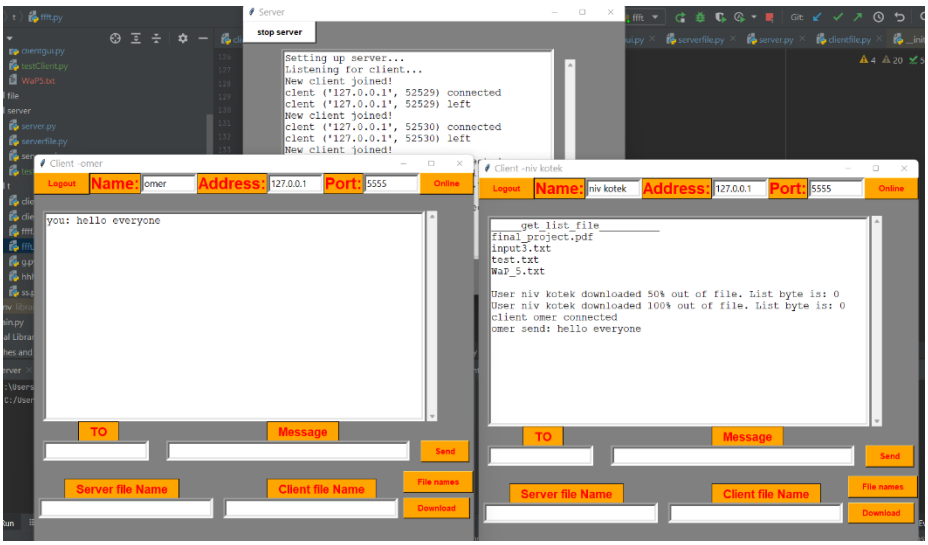
לאחר כ 100% מהקבוץ תופיע אצל הלקוח (בחלון הלבן) את מספר הבית האחרון אשר נשלחה ללקוח.

עדכונים מהשרת:

לאחר חיבור / ניתוק של לקוח נקבל עדכון מהשרת.

במידה ולקוח אחר שולח הודעה לכולם אזי נקבל ממי נשלחת ההודעה ואת תוכן ההודעה הנשלחת.

במידה ואנחנו נשלח את ההודעה אזי ההודעה תופיע גם בחלון הלבן שלנו.



מחלקת לקוח:

```
1   import socket
2   import struct
3   import threading
4   import time
5   from src.client.clientfile import ClientFile
6   from src.client.clientgui import ClientGui
7   threadLock = threading.Lock()
8   MSG = struct.Struct('!I')
9   PORT = 5555
10  HOST = "127.0.0.1"
11
12  # socket.gethostbyname(socket.gethostname())
13
14  def red_message(soc) -> str:
15      mes = soc.recv(MSG.size)
16      mesage_len = MSG.unpack(mes)[0]
17      return soc.recv(mesage_len)
18
19
20  def write_message(soc, message) -> None:
21      message = message.encode()
22      message_len = len(message)
23      message = MSG.pack(message_len) + message
24      soc.sendall(message)
25
```

שורה 1 – שימוש בספרית סוקט.

שורה 2 – שימוש בספרית מבנים.

שורה 3 - שימוש בספרית תהליכים.

שורה 4 – שימוש בספרת זמנים.

שורה 5-6 קישור בין החלקות.

שורה 7 – שימוש בבלוק תהלכים.

שורה 8 – שימוש במבנים בשביל קריאה גנרית של אורך ההודעה המתקבלת והנשלחת. המבנה הנ"ל בגדול 4 ביטים על מנת להחזיק את אורך ההודעה.

שורה 9 – מספר פורט שרירותי.

שורה 10 – מספר ip: localhost

שורה 14 – 17 – מתודה סטטית אשר קוראת את ההודעה המתקבלת. המתודה יודעת מהי אורך ההודעה על ידי שימוש במבנים. כלומר כאשר מתקבלת הודעה , אזי 4 הביטים הראשונים מציינים את אורך ההודעה.

שורה 20-24 – מתודה סטטית אשר מקבלת את הסקוט ואת ההודעה עצמה ושולחת אותו. כמו כן המתודה שולחת את ההודעה בצירוף אורך ההודעה.

מחלקת לקוח:

```
27  class Client:
28      def __init__(self, port: int = PORT, host: str = HOST, test: bool = False) -> None:
29          self._port:int = port
30          self._host:str = host
31          self._name:str = ""
32          self._socket:socket = None
33          self._protocolo_file: ClientFile = None
34          self._file_name:str = ""
35          self._port_file:int = 0
36          self._test:bool = test
37          if self._test == False:
38              self._gui: ClientGui = ClientGui(self)
39              self.start_mode()
40              self.start_thread()
41
42      def get_soc(self) -> socket:
43          return self._socket
44
45      def get_name(self) -> str:
46          return self._name
47
48      def get_port(self) -> int:
49          return self._port
50
```

שורות 28-40 : בשורות האלה נאתחל את שדות המחלקה.
מספר פורט, מספר מזהה, שם הלקוח, חיבור סוקט, חיבור לפרוטוקול שליחת קבצים. שם הקבוץ להורדה. מספר הפורט להורדת הקובץ, מזהה לבדיקות מחלקה, אתחול ממשק גרפי. אתחול פתיחת סוקט למול השרת המאזין, אתחול תהליכים במקביל.

שורה 42-43: מתודה לקבלת אובייקט הסוקט.

שורה 45-46:מתודה לקבלת אובייקט שם הלקוח.

שורה 48-49: מתודה לקבל מספר הפורט.

```
50
51      def get_ip(self) -> str:
52          return self._host
53
54      def set_name(self, name) -> None:
55          if not name == "" and not name == None:
56              self._name = name
57
58      def set_ip(self, ip) -> None:
59          if not ip == "" and not ip == None:
60              self._host = ip
61
62      def set_port(self, port) -> None:
63          if not port == "" and not port == None and not port < 123 and not port > 65535:
64              self._port = port
65
66      def start_thread(self) -> None:
67          gui_thread = threading.Thread(target=self.start_gui)
68          receive_thread = threading.Thread(target=self.receive)
69          gui_thread.start()
70          receive_thread.start()
```

שורה 51-52 : מתודה לקבלת מספר ip של השרת.

שורה 54-56: מתודה לשינוי שם הלקוח.

שורה 58-60: מתודה לשינוי ip השרת.

שורה 62 – 64: מתודה לשינוי פורט השרת.

שורה 66- 70: מתודה לאתחול התהליכים.- אתחול ממשק הגרפי, אתחול מאזין קבלת הודעות מהשרת.

מחלקת לקוח:

```
72  def start_mode(self) -> None:
73      try:
74          self._socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
75          self._socket.connect((self._host, int(self._port)))
76          write_message(self._socket, ("connect " + self._name))
77      except Exception as e:
78          print(e)
79          self._gui.gui_error("Login problem",
80                              "Check your internet connection again!+\n" + "Check the server ID and port number!")
81          exit(0)
82
83  def send_message(self, message) -> None:
84      write_message(self._socket, message)
85
86  def start_gui(self) -> None:
87      self._gui.ranGui()
```

שורה 72 – 81 : מתודה לפתיחת הקשר מסוג TCP. וחיבור בין השרת והלקוח לפי הפורט ipl. כמו כן שליחה לשרת על חיבור לקוח הכולל את השם.
במידה והקשר לא מצליח נפתחת חלונות הזהרה אשר מציינת כי החיבור נכשל בין 2 הצדדים.

שורה 83 – 84 : מתודה לשליחת הודעה לצד השרת על ידי מתודה סטטית.

שורה 86 – 87 : מתודה להתחלת ממשק הגרפי של הלקוח.

```
89  def file_Protocol(self, de, list_down) -> None:
90      threadLock.acquire()
91      write_message(self._socket, list_down)
92      time.sleep(0.3)
93      if self._gui.get_running():
94          self._protocolo_file = ClientFile(de, self._port_file, self._host, self._gui)
95      threadLock.release()
96
97  def download(self, txt_Server, txt_Client) -> None:
98      sr = txt_Server.get()
99      de = txt_Client.get()
100     txt_Server.delete(0, "end")
101     txt_Client.delete(0, "end")
102     if sr == "" or sr == None or len(sr) == 0 or de == "" or de == None or len(de) == 0:
103         return
104     list_down = "download" + " " + sr
105     thread_server_file = threading.Thread(target=self.file_Protocol, args=(de, list_down,))
106     thread_server_file.start()
```

שורה 89 – 95: מתודה המקבלת את שם הקובץ שאותו נרצה לשמור אצל הלקוח וכן הודעה לשרת עם שם הקובץ ממנו נרצה להוריד. במתודה זו נשתמש בבלוק סנכרון תהליכים ובכך במידה והלקוח רוצה להוריד קובץ ולאחר מכן להוריד קובץ נוסף אזי התהליך נעצר עד לקבלת הקובץ הראשון.

שורה 97 – 106 : מתודה המקבלת את תיבת הטקסט של שם קבוץ השרת וכן שם הקובץ שאליו נרצה לשמור. המתודה קוראת מתוך תיבות הטקסט ופותחת תהליך להורדת הקובץ.

מחלקת לקוח:

```
107
108     def write_msg(self, txt_to, txt_Message) -> None:
109         to = txt_to.get()
110         message = txt_Message.get()
111         if len(to) != 0:
112             message = "set_msg " + to + " " + message
113             txt_to.delete(0, "end")
114         write_message(self._socket, message)
115         txt_Message.delete(0, "end")
116
117     def receive(self) -> None:
118         while self._gui.get_running():
119             try:
120                 if self._gui.get_gui_ran():
121                     message = red_message(self._socket)
122                     if message[0:5].decode() == "port!":
123                         self._port_file = int(message[5:])
124                     else:
125                         self._gui.writing_window(message)
126             except ConnectionAbortedError:
127                 break
128             except Exception as e:
129                 print(e)
```

שורה 108 – 115 : המתודה מקבלת את השם אליו נרצה לשלוח את ההודעה וכן את תוכן ההודעה.

המתודה קוראת מתוך 2 תיבות הטקסט ושולחת לשרת בקשה לשיחת הודעה. במידה ולא מצוין למי לשלוח את ההודעה אזי ההודעה נשלחת לכל הלקוחות המחוברים בשרת.

שורה 117-130: המתודה מקשיבה לכל ההודעת המתקבלות. במידה ומתקבלת הודעה שאינה קשורה למספר הפורט להורדת הקבוץ אזי ההודעה תופיע בממשק הגרפי בחלון הלבן של אותו לקוח.

במידה וקיים בעיה אזי ממשק הגרפי נסגר וכן נסגר שקע הלקוח.

רעיון כללי:

מחלקה זו מראה את צד הלקוח.

הלקוח מתחבר לשרת על ידי מספר פורט וכתובת ipו .

הלקוח שולח בקשות לשרת לפי רצון המשתמש והשרת שולח לו את מבקשתו.

במידה ולא קיים אצל השרת בקשה מסוימת אזי הבקשה נשלחת אך השרת לא שולח בחזרה את התשובה לבקשה.

בין הלקוח לשרת קיים שפת דיבור משותפת להבנת השאלה והתשובה.

מחלקת קבצי לקוח:

```
1
2   import pickle
3   import socket
4   size_bits = 2000
5   time_out = 6
6
7   class UDP_MSG:
8       def __init__(self, seq, dat):
9           self.seq = seq
10          self.dat = dat
11
12  class ClientFile:
13      def __init__(self,file_name,port,ip,gui) -> None:
14          self._port=port
15          self._ip=ip
16          self._file_name=file_name
17          self._gui=gui
18          self.window_size = 5
19          self.seq_num = 0  # Sequence number
20          self.index_data_buffer = {}  # data and index buffer
21          self.index = []
22          self.lost_packages = []
23          self._stop=False
24          self._file_size=0
25          try:
26              self.UDP_socket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
27          except Exception :
28              return
29          self.start_soc_udp()
30          self.UDP_socket.close()
```

שורה 2: שימוש בספרית pickle על מנת ליצור שפה פקטות עם מספר מזהה.

שורה 3: שימוש בספרית סוקט.

שורה 4: גודל קריאה של הודעה.

שורה 5: זמן קצוב עד לקבלת ההודעה.

שורה 7-10: מחלקה צדדית בשביל שימוש בספרית pickle על מנת לבדוק את התוכן ואת מספר המזהה של הפקטה.

שורה 12-24: אתחול שדות המחלקה המקבל כקלט את שם הקובץ. מספר פורט, ip ואובייקט של ממשק הגרפי.

אתחול שדות: מספר פורט, מספר ip , שם הקובץ, אובייקט ממשק הגרפי, גודל חלון השליחות. מספר מזהה של הפקטה , מבנה נתונים של מילון לאחזקת הפקטה על מפתח וערך, רשימה של מזהים , רשימה של פקטות אבודות, ערך בוליאני לעצירת לאר קבלה של 50% מהקובץ. גודל הקובץ.

שורה 25-30 : חיבור שקע מעל UDP והתחלת הרצת מעבור הקובץ. בסיום – סגירת השקע.

מחלקת קבצי לקוח:

```
32  def start_soc_udp(self) -> None:
33      self.UDP_socket.sendto("SYN".encode(), (self._ip, self._port))
34      try:
35          server_response = self.get_response()
36      except Exception:
37          return
38      if server_response == "SYN_ACK":
39          self.UDP_socket.sendto("ACK".encode(), (self._ip, self._port))
40          print("ACK")
41          self.ran()
42      self.UDP_socket.close()
43
44  def ran(self) -> None:
45      server_window_size = self.UDP_socket.recvfrom(size_bits)
46      if int(server_window_size[0]) != self.window_size:
47          self.UDP_socket.sendto("NACK".encode(), (self._ip, self._port))
48          return
49      else:
50          self.UDP_socket.sendto("ACK".encode(), (self._ip, self._port))
51      self._file_size= int(self.UDP_socket.recvfrom(size_bits)[0])
52      if self._file_size > 0:
53          self.UDP_socket.sendto("ACK".encode(), (self._ip, self._port))
54      else:
55          self.UDP_socket.sendto("NACK".encode(), (self._ip, self._port))
56          return
57      self.len_data = 0
58      self.loop()
59
```

שורה 32-42 – מתודה אשר בודקת חיבור מהצד השרת. מתודה זו פותחת קשר על ידי 3 -way-Handshake.

כלומר כדי לפתוח קשר , בצד השני צריך להיות socket פתוח ומאזין וצריך לפתוח socket בצד הלקוח.

לאחר פתיחת הקשר באופן בטוח בין השרת ללקוח , המתודה קוראת למתודה ran להתחלת ניהול העברת הקובץ.

שורה 44-58: מתודה זו בודקת מספר דברים: האם גדול החלון של הלקוח ושל השרת זהה, וכן שליחת מצד השרת את גדול הקובץ להורדה. בדיקת הדברים נעשה באופן בטוח על ידי שיחת ACK במידה ונכון. ושליחת NACK במידה ואינו נכון. נוסף על כך המתודה קוראת למתודה loop ללולאה להעברת הקבצים.

מחלקת קבצי לקוח:

```
60      def loop(self) -> None:
61          while True:
62              try:
63                  if self.seq_num < self.window_size:
64                      self.UDP_socket.settimeout(time_out)
65                      da = self.UDP_socket.recvfrom(size_bits)[0]
66                      MSG = pickle.loads(da)
67                      ind_len = MSG.seq
68                      data = MSG.dat
69                      if data == "ACK_END" or int(ind_len) == - 1:
70                          self.check()
71                          self.writing()
72                          if self.fin():
73                              return
74                      else:
75                          self.index_data_buffer[ind_len] = data
76                          self.seq_num += 1
77                          if ind_len in self.lost_packages:
78                              self.lost_packages.remove(ind_len)
79                          self.len_data += len(data)
80
81                          if 100.0 * self.len_data/self._file_size>50.0 and self._stop:
82                              self.UDP_socket.settimeout(700)
83                              if len(self.index)==0:
84                                  self.index.append(0)
85                              self._stop=False
86                              ans= self._gui.download_question(self.index[-1])
87                              if ans ==True:
88                                  a="T"
```

שורה 60-88: מתודה זו רצה בלולאה אין סופית עד להעברת כל הקובץ.

בתנאי הראשון 63-68 המתודה בודקת האם לא עברנו את גדול החלון של הלקוח. כאשר מתודה נכנסת לתוך התנאי אזי קיים אתחול של פרק זמן לקבלת תשובה לפקטה מהשרת. קריאה של ערך הקובץ לפי חלוקה לפקטות (כאשר נשלחת פקטה מהשרת – אזי נשלח גם ערך הפקטה וגם המזהה הסידורי של הפקטה). בהמשך התנאי המתודה קוראת את ההודעה מהשרת לפי שימוש בספרית pickle (הוסבר לפני).

תנאי השני 69-76: מתודה בודקת האם השרת שלח על סיום שליחת הקובץ. במידה והתנאי מתקיים נבדוק האם הפקטות הגיעו על ידי מתודה check , לאחר מכן נכתוב לקובץ ובסוף נסיים עם סגירת קשר אמין(כל אחד מהצדדים חייב לסגור את החיבור ולהגיב ל FIN של הצד השני ב ack). במידה והתאני לא מתקיים מוספים את הערך והמספר הסידורי של הפקטה למבנה נתונים יעיל ומוסיפים אחד לחלון שליחות .

תנאי 77-79 : בודק האם אחד מהפקטות שהגיע נמצא ברשימת הפקטות האבודות. במידה וזה מתקיים הפקטה תמחק מהפקטות האבודות.

תנאי 81-95: תנאי זה הוא על מנת לבדוק האם הגיעו 50% מהפקטות. במידה והגיע אזי נשלח הודעת חלון ללקוח האם רוצה להמשיך בהורדה. במידה והתשובה היא לא אזי התהליך נעצר באופן בטוח.

```
89                          else:
90                              a="F"
91                          self.UDP_socket.sendto(a.encode(), (self._ip, self._port))
92                          if ans ==False:
93                              self.fin()
94                              return
95                      print("%d%% " % (100.0 * self.len_data / self._file_size))
96                      if len(self.index) != 0:
97                          if ind_len == self.index[-1] + 1:
98                              self.index.append(ind_len)
99                          else:
100                             self.packages_lost(ind_len)
101                     else:
102                         self.index.append(ind_len)
103             else:
104                 self.check()
105                 self.lost_packages = []
106                 self.seq_num = 0
107
108         except Exception as e:
109             print(e)
110             return
```

שורות 96-110:

תנאי 96: תנאי זה בודק האם קיים איבר ברשימה של כל מספרים של הפקטות. במידה וקיים בודק האם המספר הקודם +1 שווה למספר הפקטה החדשה. במידה והתנאי מתקיים מוסיף ומידה והתנאי לא מתקיים במתודה שולחת למתודה של פקטות אבודות.

102-105 – במידה והגנו לחלנו מקסימלי את המתודה בודקת על ידי מתודה check שאכן כל הפקטות הגיעו.

לאחר מכן מאפסת את השדות. כמו כן במידה וקיים תקלה , החיבור נסגר.

מחלקת קבצי לקוח:

```
112     def get_response(self):
113         try:
114             rec_msg = self.UDP_socket.recvfrom(size_bits)
115             return rec_msg[0].decode()
116         except Exception as e:
117             print(e, " ", "End of time")
118             return "End_time"
119
120     def send_request(self, msg, server_addr) -> bool:
121         try:
122             self.UDP_socket.sendto(msg, server_addr)
123             return True
124         except:
125             return False
```

שורות 112-118: מתודה זו מקבלת את ההודעות אשר נשלחות מהשרת. במידה והזמן עבר נשלחת הודעה על סיום זמן.

שורה 120-125 : מתודה זו שולחת הודעה לשרת. במידה וקיים תקלה מחזירה False.

```
127     def check(self) -> None:
128         if len(self.lost_packages) == 0:
129             print("ACK_ALL")
130             self.UDP_socket.sendto("ACK_ALL".encode(), (self._ip, self._port))
131             win = int(self.get_response())
132             self.window_size=win
133             return
134         else:
135             self.UDP_socket.sendto(("NACK" + str(self.lost_packages[0])).encode(), (self._ip, self._port))
136             da = self.UDP_socket.recvfrom(size_bits)[0]
137             MSG = pickle.loads(da)
138             ind_len = MSG.seq
139             data = MSG.dat
140             self.len_data += len(data)
141             self.index_data_buffer[ind_len] = data
142             print("ind_len",ind_len)
143             self.lost_packages.remove(int(ind_len))
144             # self.index.append(ind_len)
145         self.index.sort()
146         self.check()
```

שורות 127 -146: מתודה זו בודקת את כל הפקטות אשר נשלחו בגודל החלון.

128-133 – תנאי זה בודק האם לא קיים פקטות אבודות. במידה ולא קיים הלקוח שולח הודעה אשר אומרת שהגיעו כל הפקטות. ולאחר מכן מתעדכן גדול החלון לפי בדיקת התעבורה אצל השרת.

134-146 – במידה וקיים פקטות אשר אבדו אזי הלקוח שולח לשרת את כל הפקטות אשר אינם הגיעו. לאחר ששולח הכל בודק הלקוח שוב האם הגיע כל הפקטות. בשליחה מצורף כי ההודעה אינה הגיעה NACK בצירוף מספר הסידורי של הפקטה.

```
148     def writing(self) -> None:
149         with open(self._file_name, 'ab') as f:
150             for i in range(len(self.index_data_buffer)):
151                 f.write(self.index_data_buffer[i])
152         f.close()
```

שורות 148 – 152 : מתודה זו פותחת קובץ ומעתיקה את כל התוכן אשר התקבל מהשרת לקובץ.

בסוף המתודה סוגרת את הקובץ.

מחלקת קבצי לקוח:

```
154     def packages_lost(self, ind) -> None:
155         print("packages_lost", ind)
156         self.index.sort()
157         if (ind - 2) == self.index[-1]:
158             self.index.append(ind - 1)
159             self.lost_packages.append(ind - 1)
160             self.index.append(ind)
161             self.seq_num += 1
162         else:
163             ind2 = self.index[-1]
164             ind1 = ind
165             self.index.append(ind)
166             i = 1
167             while ind2 + i < ind1:
168                 print(ind2, "!!", ind1 - i)
169                 self.index.append(ind1 - i)
170                 self.lost_packages.append(ind1 - i)
171                 self.seq_num += 1
172                 i += 1
173         self.index.sort()
```

שורות 154-173: מתודה זו מוסיפה לרשימה את כל הפקטות אשר אינם הגיעו (פקטות אבודות).

שורה 157 – 161: בתנאי בודק האם ההבדל המספר הסידורי הוא 1. במידה וזה נכון מוספים את הפקטה האבודה לרשימת הפקטות האבודות.

שורה 162-173 : במידה והתנאי אינו מתקיים אזי עוברים בלולאה על כל הפקטות החסרות ומוסיפים אותם.

```
175     def fin(self) -> bool:
176         self.UDP_socket.sendto("FIN".encode(), (self._ip, self._port))
177         da = self.UDP_socket.recvfrom(size_bits)[0]
178         MSG = pickle.loads(da)
179         server_response = MSG.dat
180         if server_response == "FIN":
181             self.UDP_socket.sendto("ACK".encode(), (self._ip, self._port))
182             self._gui.fin_file((100.0 * self.len_data / self._file_size), self.index[-1])
183
184             return True
185         else:
186             return False
```

שורות 175- 186: מתודה זו סוגרת את השקע באופן אמין על ידי שליחת FIN לשרת. – כלומר כל אחד מהצדדים חייב לסגור את החיבור באופן בטוח, ולהגיב ל FIN של הצד השני ב ACK

מחלקת ממשק הגרפי של הלקוח

```
1
2   from tkinter import *
3   from tkinter import simpledialog
4   from tkinter import messagebox as mb
5   from tkinter import ttk
6
7
8   class ClientGui:
9
10      def __init__(self, client):
11          self._client = client
12          self._running = True
13          self._gui_ran = False
14          self._root = None
15          self._chatWindow = None
16          self.mpb = None
17          self.start_mode()
```

שורה 2-5 : שימוש בספרית ממשק הגרפי של פייתון.

שורות -9-19: אתחול שדות: המתודה מקבלת כקלט אוביקט מסוג המחלקה לקוח.

שדות: לקוח , שדה בוליאני הבודק את הרצת התוכנית, שדה בוליאני להרצת הממשק, שדה של שורש החלון, שדה של חלון הכתיבה , שדה של תצוגת אחוזי העברת הקובץ. לבסוף אתחול הרצת ה ממשק.

```
18
19      def get_running(self) -> bool:
20          return self._running
21
22      def get_gui_ran(self) -> bool:
23          return self._gui_ran
24
25      def writing_window(self, message) -> None:
26          self._chatWindow.config(state="normal")
27          self._chatWindow.insert("end", message)
28          self._chatWindow.yview("end")
29          self._chatWindow.config(state="disabled")
30
```

שורה 19 – 20: מתודה זו מחזירה האם הממשק עדיין פעיל.

שורה 22-23: מתודה זו מחזירה האם החלון עדיין פעיל.

שורה 25-29: מתודה זו מקבלת הודעה וכותבת אותה לחלון הצאט.

מחלקת ממשק הגרפי של הלקוח

```
30
31      def start_mode(self) -> None:
32          # # Draw start name
33          name_cl = Tk()
34          name_cl.withdraw()
35          self._client.set_name(simpledialog.askstring("name", "Type your name", parent=name_cl))
36          mb.showinfo("Greeting", f" Hello {self._client.get_name()}")
37
38          # # Draw start ip
39          ip = Tk()
40          ip.withdraw()
41          self._client.set_ip(simpledialog.askstring("ip", "Type the server ID number", parent=ip))
42
43          # # Draw start port
44          pr = Tk()
45          pr.withdraw()
46          port = simpledialog.askstring("port", "Type a port number", parent=pr)
47          if port == '':
48              port = self._client.get_port()
49          else:
50              port = int(port)
51
52          if port < 123 or port > 65535:
53              self.gui_error("forbidden", "The port is not working properly")
54              exit(0)
55          self._client.set_port(port)
```

שורות 31-55: מתודה זו מאתחלת את החלק הראשון של הממשק הגרפי.

33-36 : בחירת שם הלקוח.

39-41: בחירת ip

44-46: בחירת מספר פורט.

47-50 : במידה ולא קיים מספר פורט.

52-55: בדיקת האם הפורט תקין.

```
57      def stop_gui(self) -> None:
58          message = "disconnect"
59          self._client.send_message(message)
60          self._running = False
61          self._gui_ran = False
62          self._root.quit()
63          self._client.get_soc().close()
64          exit(0)
65
66      def gui_error(self, title, mes) -> None:
67          mb.showerror(title, mes)
68          self.stop_gui()
```

שורות 57-64: מתודה זה מפסיקה את הרצת התוכנית ושלוחת לשרת הודעה על ניתוק הלקוח.

שורה 66-68: מתודה זו מציגה חלון הזהרה על ידי שליחת כקלט את כותרת ההודעה ותוכן ההודעה.

מחלקת ממשק הגרפי של הלקוח

```
70  def ranGui(self) -> None:
71      self._root = Tk()
72      self._root.title("Client -" + self._client.get_name())
73      self._root.minsize(800, 700)
74      self._root.configure(background="grey")
75
76      # Draw a button logout
77      button_logout = Button(self._root, text='Logout', bg="orange", fg="red",
78                             font=("Arial Bold", 10), pady=5, command=self.stop_gui,
79                             padx=20)
80      button_logout.grid(row=0, column=0)
81
82      # Draw a label name
83      label_name = Label(self._root, text="Name:", bd=1, relief=SOLID, font=("Arial Bold", 20), bg="orange",
84                         fg="red", anchor=NE,
85                         wraplength=200)
86      label_name.grid(row=0, column=1)
87
88      # Draw a txt name
89      txt_name = Entry(self._root, width=10, borderwidth=6)
90      txt_name.grid(row=0, column=2)
91      txt_name.insert(0, self._client.get_name())
92
93      # Draw a label address
94      label_address = Label(self._root, text="Address:", bd=1, relief=SOLID, font=("Arial Bold", 20), bg="orange",
95                            fg="red",
```

שורה 70: מתודה זו מבצעת לולאה אינסופית על חלון הממשק.

71-74: הגדרת החלון.

77-80: הגדרת לחצן יציאה.

83-86: הגדרת כותרת השם.

89-91: שם הלקוח.

```
93      # Draw a label address
94      label_address = Label(self._root, text="Address:", bd=1, relief=SOLID, font=("Arial Bold", 20), bg="orange",
95                            fg="red",
96                            anchor=NE,
97                            wraplength=200)
98      label_address.grid(row=0, column=3)
99
100     # Draw a txt address
101     txt_address = Entry(self._root, width=10, borderwidth=6)
102     txt_address.grid(row=0, column=4)
103     txt_address.insert(0, self._client.get_ip())
104
105     # Draw a label Port
106     label_port = Label(self._root, text="Port:", bd=1, relief=SOLID, font=("Arial Bold", 20), bg="orange",
107                        fg="red",
108                        anchor=NE,
109                        wraplength=200)
110     label_port.grid(row=0, column=5)
111
112     # Draw a txt port
113     txt_port = Entry(self._root, width=10, borderwidth=6)
114     txt_port.grid(row=0, column=6, columnspan=2)
115     txt_port.insert(0, self._client.get_port())
116
117     # Draw a button Online
118     Online = Button(self._root, text='Online', bg="orange", fg="red",
119                     font=("Arial Bold", 10), pady=5, command=lambda: self._client.send_message("get_users"),
120                     padx=20)
121     Online.grid(row=0, column=8)
122
```

94-98: הגדרת כותרת ה ip.

101-103: מספר ip.

106-110- הגדרת כותרת פורט.

113-115: מספר פורט.

118-121: לחצן לבדיקת מי מחובר כעת.

מחלקת ממשק הגרפי של הלקוח

```
122
123         # Add a Scroll bar to chatWindow
124         scroll = Scrollbar(self._root, orient=VERTICAL)
125         # Draw a txt Message board
126         self._chatWindow = Text(self._root, bd=1, width=50, borderwidth=6, yscrollcommand=scroll.set)
127         self._chatWindow.place(height=385, width=700, bordermode=OUTSIDE, relx=0.02, rely=0.1)
128         scroll.config(command=self._chatWindow.yview)
129         scroll.place(x=720, y=70, height=385)
130
131         # Draw a label TO
132         label_to = Label(self._root, text="TO", bd=1, relief=SOLID, font=("Arial Bold", 15), bg="orange",
133                          fg="red",
134                          anchor=NE,
135                          wraplength=200, pady=5,
136                          padx=20)
137         label_to.place(relx=0.1, rely=0.65)
138
139         # Draw a txt TO
140         txt_to = Entry(self._root, width=20, borderwidth=6)
141         txt_to.place(relx=0.02, rely=0.7)
142
143         # Draw a label Message
144         label_Message = Label(self._root, text="Message", bd=1, relief=SOLID, font=("Arial Bold", 15), bg="orange",
145                               fg="red",
146                               anchor=NE,
147                               wraplength=200, pady=5,
148                               padx=20)
149         label_Message.place(relx=0.53, rely=0.65)
150
```

124-129: הגדרת חלונית הצאט עם קישור לגלגלת במידה וקיים ריבוי הודעות.

132-137: הגדרת כותרת למי לשלוח את ההודעה.

140-141: חלון לכתיבה למי לשלוח את ההודעה.

144-149: כותרת של ההודעה.

```
150
151         # Draw a txt  Message
152         txt_Message = Entry(self._root, width=48, borderwidth=6)
153         txt_Message.place(relx=0.30, rely=0.7)
154
155         # Draw a button Send
156         button_Send = Button(self._root, text='Send', bg="orange", fg="red",
157                              command=lambda: self._client.write_msg(txt_to, txt_Message), font=("Arial Bold", 10),
158                              pady=5,
159                              padx=20)
160         button_Send.place(relx=0.88, rely=0.7)
161
162         # Draw a label Server file Name
163         label_Server = Label(self._root, text="Server file Name", bd=1, relief=SOLID, font=("Arial Bold", 15),
164                              bg="orange",
165                              fg="red",
166                              anchor=NE,
167                              wraplength=200, pady=5,
168                              padx=20)
169         label_Server.place(relx=0.07, rely=0.8)
170
171         # Draw a text Server file Name
172         txt_Server = Entry(self._root, width=34, borderwidth=6)
173         txt_Server.place(relx=0.01, rely=0.85)
174
```

152-153: חלון לכתיבת ההודעה.

156-160: לחצן לשליחת ההודעה.

163-169: כותרת לשם הקובץ של השרת.

172-173: חלון לכתיבת שם קובץ של השרת להורדה.

מחלקת ממשק הגרפי של הלקוח

```
176         label_Client = Label(self._root, text="Client file Name", bd=1, relief=SOLID, font=("Arial Bold", 15),
177                              bg="orange",
178                              fg="red",
179                              anchor=NE,
180                              wraplength=200, pady=5,
181                              padx=20)
182         label_Client.place(relx=0.53, rely=0.8)
183
184         # Draw a  button File names
185         File_names = Button(self._root, text='File names', bg="orange", fg="red",
186                             font=("Arial Bold", 10), pady=5, command=lambda: self._client.send_message("get_list_file"),
187                             padx=20)
188         File_names.place(relx=0.84, rely=0.78)
189
190         # Draw a text Client file Name
191         txt_Client = Entry(self._root, width=34, borderwidth=6)
192         txt_Client.place(relx=0.43, rely=0.85)
193
194         # Draw a button Send
195         download = Button(self._root, text='Download', bg="orange", fg="red",
196                           command=lambda: self._client.download(txt_Server, txt_Client), font=("Arial Bold", 10),
197                           pady=4,
198                           padx=20)
199         download.place(relx=0.84, rely=0.85)
200         self._gui_ran = True
201         self._root.protocol("WM_DELETE_WINDOW", self.stop_gui)
202         self._root.mainloop()
```

176-182: כותרת לכתיבת שם קובץ הלקוח.

185-188: לחצן לקבלת רשימת הקבצים הנמצאים אצל השרת.

191-192: חלון לכתיבת שם קובץ הלקוח.

195-199: לחצן להורדת הקבוץ.

200-202: הגדרות של המשכיות החלון.

```
204     def download_question(self, byte) -> bool:
205         # Draw a Download bar
206         self.mpb = ttk.Progressbar(self._root, orient="horizontal", length=600, mode="determinate")
207         self.mpb.place(relx=0.05, rely=0.95)
208         self.mpb["maximum"] = 100
209         self.mpb["value"] = 50
210         self.writing_window(f"User {self._client.get_name()} downloaded 50% out of file. List byte is: {byte}\n")
211         cl = Tk()
212         cl.withdraw()
213         answer = mb.askquestion("Download file", "Moved 50% of the file. Continue downloading?", parent=cl)
214         if answer == 'yes':
215             return True
216         else:
217             return False
218
219     def fin_file(self, sum, byte) -> None:
220         self.mpb = ttk.Progressbar(self._root, orient="horizontal", length=600, mode="determinate")
221         self.mpb.place(relx=0.05, rely=0.95)
222         self.mpb["maximum"] = 100
223         self.mpb["value"] = 0
224         if sum >= 100:
225             self.writing_window(f"User {self._client.get_name()} downloaded 100% out of file. List byte is: {byte}\n")
226         else:
227             self.writing_window(f"User {self._client.get_name()} downloaded {sum} out of file. List byte is: {byte}\n")
228
```

204-217: מתודה המציגה חלון האם להמשיך בהורדת הקובץ מהשרת לאחר 50%. כמו כן המתודה מעדכנת את סקלת ההורדה.

219-227: מתודה אשר מסיימת את שליחת הקבצים ומעדכנת את שדה הסקלה.

מחלקת שרת

```python
1   import select
2   import socket
3   import struct
4   import threading
5   import time
6   import src.file.path as p
7   from src.server.serverfile import ServerFile
8   from src.server.servergui import ServerGui
9
10  threadLock = threading.Lock()
11  IP = "0.0.0.0"
12  # socket.gethostbyname(socket.gethostname())
13  PORT = 5555
14  MSG = struct.Struct('!I')
15
16
17  def red_message(soc) -> str:
18      mes = soc.recv(MSG.size)
19      mesage_len = MSG.unpack(mes)[0]
20      return soc.recv(mesage_len).decode()
21
22
23  def write_message(soc, message) -> None:
24      message = message.encode()
25      message_len = len(message)
26      message = MSG.pack(message_len) + message
27      soc.sendall(message)
```

שורה 1: שימוש ספרית select אשר מזהה האם קיים חיבור נוסף לשרת/ קיים הודעה לשרת מלקוח (מאזינה לשקע)

שורה 2: שימוש בספרית סוקט.

שורה 3: שימוש בספרית מבנים.

שורה 4: שימוש בספרית תהליכים.

שורה 5: שימוש בזמנים.

שורות 6-8: קישור מחלקות.

שורה 10: בלוק לתהליך.

שורה11: ip דיפולטיבי.

שורה 13: פורט דיפולטיבי.

שורה 14: הגדרת מבנה בגדול 4 ביטים.

שורה 20 – 17 – מתודה סטטית אשר קוראת את ההודעה המתקבלת. המתודה יודעת מהי אורך ההודעה על ידי שימוש במבנים. כלומר כאשר מתקבלת הודעה , אזי 4 הביטים הראשונים מציינם את אורך ההודעה.

שורה 27-23 – מתודה סטטית אשר מקבלת את הסקוט ואת ההודעה עצמה ושולחת אותו. כמו כן המתודה שולחת את ההודעה בצירוף אורך ההודעה.

מחלקת שרת

```
30  class Server:
31      def __init__(self, port: int = PORT, ip: str = IP) -> None:
32          self._port_file = 55000
33          self._port: int = port
34          self._ip: str = ip
35          self._clients = {}
36          self._sockets = []
37          self._file_names = p.get_path()
38          self._root_path_file = p.get_root_path() + "\\"
39          self._file_name_read: str = ""
40          self._gui: ServerGui = ServerGui(self)
41          self._server_socket: socket = None
42          self._down = False
43          self.starting_path()
44          self.start_mode()
45          self.starting_Processes()
46
47      def get_soc(self) -> socket:
48          return self._server_socket
49
50      def get_port(self) -> int:
51          return self._port
52
53      def get_ip(self) -> str:
54          return self._ip
55
56      def set_port(self, port: int) -> None:
57          if port == "" or port == None:
58              port = PORT
59          self._port = port
60
```

שורות 31-45: מתודה לאתחול שדות המחלקה. המתודה מקבלת כקלט את מספר הפורט ואת ip .

השדות: מספר פורט להורדת הקובץ. פורט שרת. Ip שרת. מבנה נתונים אשר שומר את שם הלקוח (מפתח) ואת הסוקט (הערך.). רשימה של כל הסוקטים. רשימה של כל הקבצים. נתיב הקובץ. שם הקובץ להורדה. אובייקט ממשק גרפי. אובייקט סוקט. סיום ההורדה. אתחול נתיב. אתחול שקע. אתחול תהליכים.

שורה 47-48: קבלת סוקט.

שורה 50-51: קבלת פורט.

שורה 53-54: קבלת מספר ip

שורה 56-59: שינוי מספר פורט ובדיקת תקינות הפורט.

```
61      def set_ip(self, ip: str) -> None:
62          if ip == "" or ip == None:
63              ip = IP
64          self._ip = ip
65
66      def starting_path(self) -> None:
67          self._file_names.remove("__pycache__")
68          self._file_names.remove("path.py")
69          self._file_names.append("\n")
70
71      def starting_Processes(self) -> None:
72          # thread_server_file = threading.Thread(target=self.file_Protocol)
73          gui_thread = threading.Thread(target=self.start_gui)
74          receive_thread = threading.Thread(target=self.receive)
75          gui_thread.start()
76          # thread_server_file.start()
77          time.sleep(0.5)
78          receive_thread.start()
```

שורה 61-64: מתודה לשינוי ip ובדיקה.

שורה 66-69: מתודה אשר מוציאה את כל הקבצים הלא שייכים מתוך רשימת הקבצים של השרת.

שורה 71-78: מתודה אשר מאתחלת את כל התהלכים.

אתחול תהליך ממשק הגרפי וכן אתחול קבלת הודעות מלקוחות.

מחלקת שרת

```
79
80      def start_mode(self) -> None:
81          try:
82              self._server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
83              self._server_socket.bind((self._ip, int(self._port)))
84              self._server_socket.listen()
85              self._clients["server"] = self._server_socket
86              self._sockets.append(self._server_socket)
87          except Exception as e:
88              print(e)
89              self._gui.gui_error("Login problem",
90                                  "Check your internet connection again!\n" + "Check the  ID and port number!")
91              exit(0)
92
93      def get_name(self, client) -> str:
94          for v in self._clients.keys():
95              if self._clients[v] is client:
96                  return v
97
98      def start_gui(self) -> None:
99          self._gui.ranGui()
100
```

שורות 80-91: מתודה זו מאתחלת את שקע הפתיחה של השרת מעל TCP.

82- פתיחת סוקט

83- חיבור שקע(קישור כתותב ופורט עם סוקט).

84- האזנה למספר לקוחות.

85-86: הכנסה נתונים למבנה נתונים.

87-91: במידה וקיים שגיאה.

שורות 93-96: מתודה זו מוצאת את שם הלקוח לפי הסוקט. לבסוף מחזירה את שם הלקוח.

שורות 98-99: מתודה זו מאתחלת את ממשק הגרפי של השרת.

```
100
101     def receive(self) -> None:
102         while self._gui.get_running():
103             ready, _, _ = select.select(self._sockets, [], [])
104             for user in ready:
105                 try:
106                     if user is self._server_socket:
107                         (client_socket, client_address) = user.accept()
108                         self._gui.writing_window("New client joined!\n")
109                         self._sockets.append(client_socket)
110                         self._gui.writing_window(f"client {client_address} connected\n")
111                     else:
112                         self.send_message(user)
113                 except Exception:
114                     self._gui.stop_gui()
115                     exit(0)
```

שורות 101-115: מתודה זו מאזינה לכל החיבורים \ההודעות הנשלחים לשרת כל עוד ממשק הגרפי פעיל.

103- שימוש בספרית select אשר מאזינה לכל ההודעות דרך אותו פורט וכתובת.

104- מעבר על כל החיבורים.

106- אם קיים חיבור חדש אזי מצרף אותו לרשימת הלקוחות וכן שולח הודעה לכולם על חיבור חדר של לקוח.

112- אם הלקוח אינו חדש אזי הוא שלח הודעה. ולכן נשלח את ההודעה ללקוחות בהתאם לתוכן ההודעה.

מחלקת שרת

```
117     def sent_all(self, msg, client) -> None:
118         for connection in self._sockets:
119             if connection is not client and connection is not self._server_socket:
120                 write_message(connection, msg)
121
122     def send_message(self, client) -> None:
123         try:
124             data: str = red_message(client)
125         except Exception:
126             for v in self._clients.keys():
127                 if self._clients[v] is client:
128                     del self._clients[v]
129                     self._sockets.remove(client)
130                     self.sent_all(f"client {v} left\n", [])
131                     self._gui.writing_window(f"clent {client.getpeername()} left\n")
132                     client.close()
133                     break
134             return
135
```

שורות 117-120: זהו מתודה אשר מקבלת כקלט את תוכן ההודעה ואת הלקוח שאינו אנו רוצים לשלוח לו.

המתודה עוברת על כל הסוקטים הנמצאים אצל השרת במנה נתונים ושולחת לכולם את ההודעה חוץ מהלקוח אשר מצויין בקלט של התודה.

שורות 122-134: מתודה זו מקבלת כקלט את סוקט הלקוח אשר שולח את ההודעה לשרת. ומבצעת את הפעולה לפי דרישת תוכן ההודעה.

124- קריאת ההודעה.

126-132 במידה ולקוח התנתק\ קיים בעיה – אזי מוחקים את הלקוח הנ"ל מרשימת הלקוחות.

```
136         if str(data).startswith("connect"):
137             st = data.replace("connect ", '')
138             print(st)
139             self._clients[st] = client
140             self.sent_all(f"client {st} connected\n", client)
141             return
142         if data[0:8] == "download":
143             if data[9:] not in self._file_names:
144                 return
145             write_message(client, "port!" + str(self._port_file))
146             list_data = data.split()
147             self._file_name_read = self._root_path_file + "" + list_data[1]
148             self._port_file += 1
149             thread_server_file = threading.Thread(target=ServerFile, args=(self._file_name_read, self._port_file - 1,))
150             if self._port_file > 55016:
151                 self._port_file = 55000
152             thread_server_file.start()
153             self._down = True
154             self._gui.writing_window(f"User {self.get_name(client)}  downloads {list_data[1]} file\n")
155             return
156         if data[0:] == "get_users":
157             self._gui.writing_window(f"User {self.get_name(client)} want to get the list of online\n")
158             for v in self._clients.keys():
159                 if self._clients[v] is not client:
160                     if v == "server":
161                         write_message(client, "_____get_users_________\n")
162                     else:
163                         write_message(client, (v + ", "))
164             write_message(client, '\n')
165             return
```

שורה 136: אם ההודעה על חיבור לקוח חדש אזי מוספים את הלקוח החדש למבנה הנתונים וכן מעדכנים את כל הלקוחות המחוברים על חיבור של לקוח חדש למערכת השרת.

שורה 142: אם ההודעה הוא על הורדת קובץ אזי קוראים את שם הקובץ ובודקים האם נמצא את השרת. במידה ולא תהליך ההורדה לא לקורה. כמו כן השרת שולח את מספר הפורט ללקוח לפתיחת הסוקט להעברת הקובץ. נוסף לכך נפתח תהליך של חיבור פרוטוקול של שליחת הקבצים של השרת ללקוח מעל UDP.

לבסוף נשלחת הודעה אל השרת על כך שהלקוח מוריד קובץ.

שורה 156: אם ההודעה הוא על בקשת רשימת כל הלקוחות המחוברים אזי נשלחת הודעה לשרת על בקשה של לקוח לקבלת רשימת הלקוחות. לאחר מכן עוברים בלולאה על כל השומות ושולחים את הרשימה ללקוח אשר ביקש.

מחלקת שרת

```
165                 return
166             if data[0:] == "get_list_file":
167                 self._gui.writing_window(f"User {self.get_name(client)} want to get the list of file\n")
168                 write_message(client, "_____get_list_file__________\n")
169                 write_message(client, '\n'.join(self._file_names))
170                 return
171
172             if data[0:] == "disconnect":
173                 print("disconnect")
174                 for v in self._clients.keys():
175                     if self._clients[v] is client:
176                         del self._clients[v]
177                         self._sockets.remove(client)
178                         self.sent_all(f"client {v} left\n", [])
179                         self._gui.writing_window(f"clent {client.getpeername()} left\n")
180                         client.close()
181                         break
182                 return
183
184             if data[0:7] == "set_msg":
185                 ms = data.split()
186                 for v in self._clients.keys():
187                     if ms[1] == v:
188                         print(v)
189                         write_message(self._clients[v], (self.get_name(client) + " send: " + ms[2] + "\n"))
190                         write_message(client, ("you: " + ms[2] + "\n"))
191                         return
192             self.sent_all(str(self.get_name(client) + " send: " + data + "\n"), client)
193             write_message(client, ("you: " + data + "\n"))
```

שורה 166: אם ההודעה הוא על בקשת כל הקבצים אשר נמצאים אשר השרת. אזי נשלחת הודעה לשרת על בקשה זו. לאחר מכן שולחים ללקוח שביקש את כל הרשימה של הלקוחות.

שורה 172: אם ההודעה הוא על ניתוק הקשר. אזי מוחקים את הלקוח מהרשימה ושולחים לכולם הודעה על ניתוק לקוח.

שורה 184: אם ההודעה הוא על שליחת הודעה ללקוח אחד לפי השם: אזי מוצאים את השם שלו ושולחים לו את ההודעה.

שורה 192: אם ההודעה לא נכנסה לשום תנאי אזי ההודעה נשלחת לכל הלקוחות.

רעיון כללי:

מחלקה זו מראה את צד השרת.

השרת מתחבר לשקע על ידי מספר פרט וכתובת.

השרת מאזין לכל החיבורים לפי מספר הפורט והכתובת.

השרת קורא את ההודעה ומבצע את הבקשה של הלקוח לפי תוכן ההודעה.

השרת מעדכן את כל הלקוחות על חיבור נכנס ויוצא.

מחלקת קבצי שרת:

```
1
2   import pickle
3   import socket
4   import time
5   import os
6   ip = "0.0.0.0"
7   port = 9989
8   time_out = 6
9   fragment_size = 1024
10
11  class UDP_MSG:
12      def __init__(self, seq, dat):
13          self.seq = seq
14          self.dat = dat
15  class ServerFile:
16
17      def __init__(self,file_name,port):
18          self._port=port
19          self._file_name=file_name
20          self.window_size = 5
21          self.packet_num = -1
22          self.seq_num = 0
23          self.data_buffer = {}
24          self.max_time_out = 4
25          self.number_time_out = 0
26          self.f = None
27          self._add_window = True
28          self._linar = False
29          self._sum_ack = 0
30          chek=True
31          self.len_data=0
```

```
30          chek=True
31          self.len_data=0
32          try:
33              self.UDP_socket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
34              self.UDP_socket.bind((ip, self._port))
35          except Exception as e:
36              print(e)
37              chek =False
38          if chek:
39              self.UDP_socket.settimeout(time_out)
40              self.ran()
41              self.f.close()
42              self.UDP_socket.close()
```

שורה 2: שימוש בספרית pickle בשביל שימוש בשליחת פקטה עם מספר סידורי.

שורה 3: שימוש בספרית סוקט.

שורה 4: שימוש בספרית זמנים.

שורה 5: שימוש בספרית os בשביל קבצים .

שורה 6: מספר כתובת.

שורה 7: מספר פורט.

שורה 8: זמן עד לקבלת תשובה.

שורה 9: גודל ההודעה.

שורות 11-14: שימוש במחלקה לשימוש בספרית pickle.

שורות 15-31: מתודה המאתחלת את השדות. מקבלת כקלט את שם הקובץ ואת מספר הפורט לחיבור עם הלקוח אשר רוצה להעביר.

שדות: פורט. שם הקובץ. גודל החלון. מספר פקטה. מספר סידורי. מבנה נתונים של מילון המחזיק את מספר הפקטה (מפתח) ותוכן הפקטה (ערך). מספר מקסימלי לשיחת פעם נוספת של פקטה במקרה של time out . מספר ה time out. אובייקט של פתיחת קובץ. משתנה בוליאני הבודק האם להגדיל את חלון השליחה. משתנה בוליאני הבודק האם להגדיל את החלון בצורה לינארית . משתנה הסופר את כמו האck . במידה ויש 3 ack אחד אחרי השני אזי נגדיל את חלון השליחה . משתנה זמני לבדיקת חיבור שקע תקין. אורך סך כל הפקטות.

שורות 33-34: חיבור סוקט וחיבור שקע שרת.

שורה 39- שיינוי זמנים.

שורה 40-42 – התחלת הרצת שרת הקבצים. סגירת הקובץ וסגירת הסוקט

מחלקת קבצי שרת:

```
44
45      def ran(self):
46          msg, client_addr = self.UDP_socket.recvfrom(fragment_size)
47          if msg.decode() == "SYN":
48              self.send_request("SYN_ACK".encode(), client_addr)
49          else:
50              return
51          clint_response = self.get_response()
52          if clint_response == "ACK":
53              file_size = os.path.getsize(self._file_name)
54
55              # Send window
56              self.send_request(str(self.window_size).encode(), client_addr)
57              server_response = self.get_response()
58              if server_response == "NACK" or server_response != "ACK":
59                  return
60              print("file", file_size)
61              self.send_request(str(file_size).encode(), client_addr)
62              server_response = self.get_response()
63              if server_response != "ACK":
64                  return
65              self.f = open(self._file_name, "rb")
66              data = self.f.read(fragment_size)
67              final_packet = int(file_size / fragment_size)  # Index of final packet
68              boo = True
69              self.loop(final_packet, client_addr, data,file_size,,boo)
70          else:
71              return
```

שורות 45-71: מתודה זו פותחת קשר Handshake – way -3 וכן שולחת ללקוח את גודל החלון.

46-53 - פותחת קשר Handshake – way -3 – כדי לפתוח קשר בצד השני צריך להיות socket פתוח שמאזין.

56-59 – שליחת גודל החלון.

60-64 – שליחת גודל הקובץ.

65-69 – פתיחת הקובץ לקריאה והמשך למתודה של לולאה אינסופית

```
73      def loop(self, final_packet, client_addr, data,file_size,,boo):
74          while True:
75              if self.packet_num == final_packet:
76                  MSG = UDP_MSG(-1, "ACK_END")
77                  b_msg = pickle.dumps(MSG)
78                  self.send_request(b_msg, client_addr)
79                  bo = self.check(client_addr)
80                  if bo == False:
81                      return
82                  message = self.get_response()
83                  if message == "FIN":
84                      self.fin(client_addr)
85                      message = self.get_response()
86                      if message == "ACK":
87                          time.sleep(0.1)
88                          self.f.close()
89                          return
90              if self.seq_num < self.window_size:
91                  self.packet_num += 1
92                  self.seq_num += 1
93                  self.data_buffer[self.packet_num]=data
94                  if self.packet_num == 5 or self.packet_num == 6 or self.packet_num == 7:
95                      data = self.f.read(fragment_size)
96                      continue
97                  MSG = UDP_MSG(self.packet_num, data,)
98                  b_msg = pickle.dumps(MSG)
99                  self.send_request(b_msg, client_addr)
100                 time.sleep(0.2)
101                 self.len_data += len(data)
```

שורות 73-101: מתודה זו מקבלת כקלט את מספר הפקטה האחרונה. את סוקט הלקוח. תוכן ההודעה לשליחה. גודל הקובץ. משתנה בוליאני לבדיקת 50% מהפקטות אשר נשלחו.

75- אם זה הפקטה האחרונה אזי נשלח ללקוח שהשרת סיים שלוח את כל הפקטות. לאחר מכן נבדוק עם הלקוח שאכן כל הפקטות הגיעו ולבסוף נסגור את הקשר באופן אמין על ידי FIN.

90- אם חלון השליחות עדיין פנוי אזי נשים את תוכן ההודעה במבנה נתונים לפי מספר הסידורי של כל פקטה.

94-96 – בדיקה לאיבוד פקטות.

96-199 – שליחת הפקטה ללקוח.

מחלקת קבצי שרת:

```
102                 if 100.0 * self.len_data / file_size > 50.0 and boo:
103                     boo=False
104                     self.UDP_socket.settimeout(500)
105                     message = self.get_response()
106                     if message == "F":
107                         message = self.get_response()
108                         if message =="FIN":
109                             self.fin(client_addr)
110                             return
111
112                 print("%d%% " % (100.0 * self.len_data / file_size))
113                 data = self.f.read(fragment_size)
114
115             else:
116                 bo = self.check(client_addr)
117                 if bo == False:
118                     return
119
120     def get_response(self):
121         try:
122             rec_msg = self.UDP_socket.recvfrom(fragment_size)
123             return rec_msg[0].decode()
124         except Exception as e:
125             print(e)
126             return "time out"
```

שורה 102-110: בדיקה אם עברו 50% מהקבצים אל הלקוח. במידה ועברו נבדוק את תשובת הלקוח האם רוצה להמשיך בהעברת הקבצים.

112- הדפסת אחוז העברה.

113- קריאה של קטע נוסף.

115-118: אם עברו את גדול החלון אזי נבדוק האם הלקוח קיבל את כל הפקטות אשר נשלחו על ידי השרת.

שורות 120-126: מתודה אשר קוראת את ההודעה מהלקוח וחזירה אותה. במידה והזמן חלף אזי נשלח ההודעה time out.

```
128     def send_request(self, msg, client_addr):
129         try:
130             self.UDP_socket.sendto(msg, client_addr)
131             return True
132         except:
133             return False
134
135     def check(self, client_addr) -> bool:
136         windo = self.window_size
137         self.UDP_socket.settimeout(time_out)
138         message = self.get_response()
139         if message == "ACK_ALL":
140             self.ack_all()
141             self.send_request(str(self.window_size).encode(), client_addr)
142             return True
143         if message == "FIN":
144             self.fin(client_addr)
145             message = self.get_response()
146             if message == "ACK":
147                 time.sleep(0.1)
148                 self.f.close()
149                 return False
```

שורות 128-133: מתודה זו מקבלת כקלט את תוכן ההודעה ואת ההדר של הלקוח ושלוחת ללקוח את ההודעה.

שורות 135- 149: מתודה זו בודקת את כל הפקטות אשר נשלחו בטווח של החלון שליחות.

139 – אם הלקוח שלך שהגיע הכל – אזי מתודה תשלח למתודה ack_all אשר מגדילה את חלון השליחות בהתאם למספר איבוד הפקטות. ובסוף שולחת ללקוח את גדול החלון החדש.

143 – אם הלקוח שלך שהוא רוצה לסיים אזי ננתקח את הקשר באופן בטוח על ידי שליחה למתודה fin.

מחלקת קבצי שרת:

```
150
151             if message == "time out":
152                 if self.number_time_out < self.max_time_out:
153                     self.time_out(client_addr, windo)
154                     self.check(client_addr)
155                     self.window_size /= 2
156                 else:
157                     self.fin(client_addr)
158                     self.f.close()
159                     return False
160             if message[0:4] == "NACK":
161                 self.nack(message, client_addr)
162                 self.check(client_addr)
163             else:
164                 self.check(client_addr)
165
166         def ack_all(self) -> None:
167             if self._add_window:
168                 self._sum_ack += 1
169                 if self._linar and self._sum_ack < 2:
170                     self.window_size += 2
171                 else:
172                     self.window_size *= 2
173
174             self._add_window = True
175             self.seq_num = 0
176             self.number_time_out = 0
```

שורה 151- אם הזמן עבר (כלומר הלקוח לא ענה לשרת) אזי נשלח שוב ללקוח את כל הפקטות אשר באותו חלון שליחה. וכן נעדכן את גדול החלון בחלקי 2. כמו כן נעלה את השדה של time out. ובמידה והלקוח לא ענה לאחר 3 פעמים של שליחה מחדש של הפקטות אזי השרת סוגרת את הקשר.

שורה 160 – אם ההודעה הוא על אי קבלת פקטה מסוימת אזי השרת שולח ללקוח את אותו הפקטה לפי מספר הסידורי אשר ביקש הלקוח.

שורות 166-176: זהו מתודה אשר מעדכנת את השדות ובודקת האם לבגדיל את החלון בצורה לינארית או בצורה אקספוננציאלית.

```
177
178         def fin(self, client_addr) -> None:
179             MSG = UDP_MSG(-1, "FIN")
180             b_msg = pickle.dumps(MSG)
181             self.send_request(b_msg, client_addr)
182
183         def time_out(self, client_addr, windo) -> None:
184             self.number_time_out += 1
185             self._add_window = False
186             i = self.packet_num - windo + 1
187             while i <= self.packet_num:
188                 print("i", i)
189                 data = self.data_buffer[i]
190                 MSG = UDP_MSG(i, data)
191                 b_msg = pickle.dumps(MSG)
192                 self.send_request(b_msg, client_addr)
193                 time.sleep(0.1)
194                 i += 1
195
196         def nack(self, message, client_addr) -> None:
197             print(message)
198             self._add_window = False
199             self._linar = True
200             self._sum_ack = 0
201             self.window_size -= 1
202             message = int(message[4:])
203             data = self.data_buffer[message]
204             MSG = UDP_MSG(message, data)
```

שורות 178-181: מתודה זו שולחת סיום ללקוח.(FIN)

שורות 183-194: המתודה מקבלת כקלט את ההדר של הלקוח ואת גדול החלון. המתודה שולחת מחדש את כל הפקטות של חלון השליחות.

שורות 196-204: המתודה מקבלת כקלט את ההדר של הלקוח וכן את ההודעה. המתודה מקטינה את החלון עקב איבוד פקטות וכן שולחת ללקוח את הפקטות אשר לא נשלח.

מחלקת ממשק הגרפי של השרת:

```
1   from tkinter import *
2   from tkinter import messagebox as mb, simpledialog
3
4   class ServerGui:
5
6       def __init__(self, server) -> None:
7           self._server = server
8           self._running = True
9           self._chatWindow = None
10          self._root=None
11          self.start_mode()
12
13      def get_running(self) -> bool:
14          return self._running
15
16      def writing_window(self, message) -> None:
17          self._chatWindow.insert(END, message)
18
19      def stop_gui(self) -> None:
20          self._running = False
21          self._root.quit()
22          self._server.get_soc().close()
23          exit(0)
24
```

שורה 1-2 : שימוש בספרית ממשק הגרפי של פייתון.

שורות 6-11: מתודה אשר מקבלת כקלט אובייקט מסוג מחלקת השרת. ומאתחלת את השדות.

השדות: השרת. משתנה בוליאני האם הממשק עדיין רץ. אובייקט לכתיבת הודעות בחלון הלבן של השרת.

שורות 13-14: מתודה אשר מחזירה האם הממשק עדיין רץ.

שורות 16-17: מתודה אשר מקבלת כקלט אש ההודעה וכותבת לחלון הלבן של השרת.

שרות 19-24: מתודה את מפסיקה את ממשק הגרפי וסוגרת את השקע.

```
25      def start_mode(self) -> None:
26          # Draw start ip
27          ip = Tk()
28          ip.withdraw()
29          self._server.set_ip(simpledialog.askstring("ip", "Type the server ID number", parent=ip))
30
31          # Draw start port
32          pr = Tk()
33          pr.withdraw()
34          self._server.set_port(simpledialog.askstring("port", "Type a port number", parent=pr))
35          if int(self._server.get_port()) < 123 or int(self._server.get_port()) > 65535:
36              self.gui_error("forbidden", "The port is not working properly")
37
38      def gui_error(self, title, mes) -> None:
39          mb.showerror(title, mes)
40          self.stop_gui()
```

שרות 25-36: מתודה זו מאתחלת את החלק הראשון של הממשק הגרפי.

29 – חלון לבחירת כתובת השרת.(במידה ולא נבחר – הערך יהיה 0.0.0.0.

34- חלון לבחירת מספר פורט. (במידה ולא נבחר – הערך יהיה 5555)

35 – בדיקה האם הפורט תקין. -במידה ולא תופיע חלון הזהרה על כך.

שרות 38-40 – מתודה זו מציגה חלון של הזהרות לשרת.

מחלקת ממשק הגרפי של השרת:

```
42  def ranGui(self) -> None:
43      self._root = Tk()
44      self._root.title("Server")
45      self._root.minsize(700, 500)
46      self._root.configure(background="grey")
47      scroll = Scrollbar(self._root, orient=VERTICAL)
48      self._chatWindow = Text(self._root, bd=1, width=50, borderwidth=6, yscrollcommand=scroll.set)
49      self._chatWindow.place(height=385, width=500, bordermode=OUTSIDE, relx=0.1, rely=0.1)
50      scroll.config(command=self._chatWindow.yview)
51      scroll.place(x=590, y=70, height=385)
52      self.writing_window("Setting up server...\n" + "Listening for client...\n")
53
54      # Draw a button stop server
55      button_logout = Button(self._root, text='stop server', bg="white", fg="black",
56                             font=("Arial Bold", 10), pady=5, command=self.stop_gui,
57                             padx=20)
58      button_logout.grid(row=0, column=0)
59      self._root.protocol("WM_DELETE_WINDOW", self.stop_gui)
60      self._root.mainloop()
61
```

שורות 42- 60: מתודה זו מריצה את החלון בלולאה אינסופית.

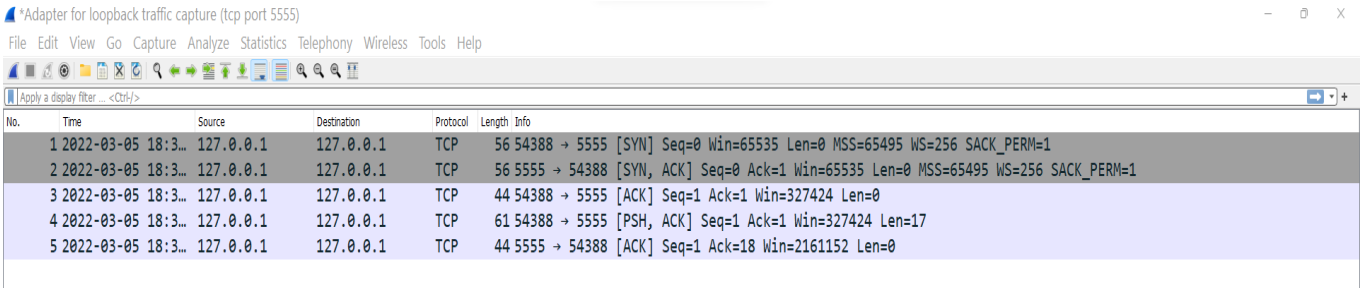
43 – אתחול שורש החלון.

44-46 – אתחול החלון.

47-52 – אתחול חלון הלבן של השרת שבו השרת מקבל את ההודעות החשובות. כמו כן התאמה לסקלה של חלון השליחות.

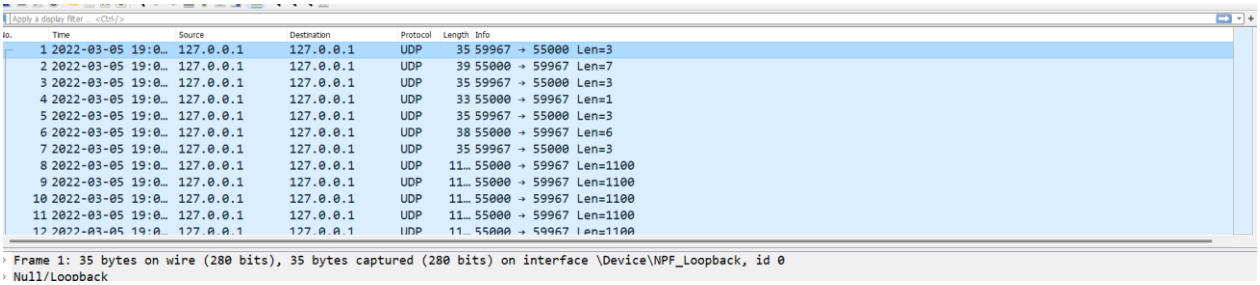
55-60 - לחצן אשר מפסיק את השרת.

Wireshark Network

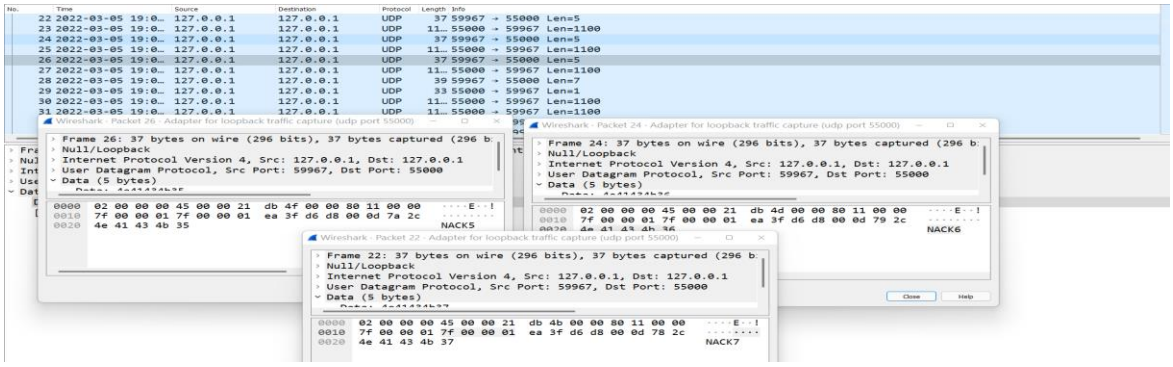


בחלון הנ"ל ניתן לראות את פתיחת הקשר של השרת והלקוח מעל TCP. כאשר השרת מקשיב לפורט 5555 והלקוח עם פורט 54388. ניתן לראות את לחיצת היידים בין הלקוח והשרת.

ניתן לראות את הקובץ ההאזנה בשם tcp.



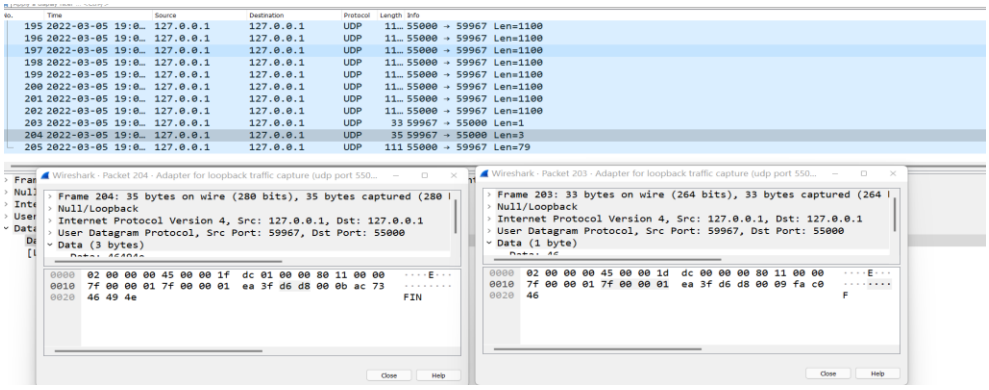
בחלון הנ"ל ניתן לראות את פתיחת הקשר של השרת והלקוח מעל UDP אמין. השרת מאזין לפורט 55000 והלקוח לפורט 59967. ניתן לראות בהחלה (שורות 1-7) את פתיחת הקשר האמין. ולאחר מכן את שליחת הקובץ לפי חלקים. (ניתן לראות את ההקלטה מתוך הקובץ lost_udp)



בחלון הנ"ל ניתן לראות איבוד פקטות בין השרת והלקוח – וכתוצאה מכך הלקוח מבקש מהשרת שליחה נוספת של הפקטות מספר 5 , 6 ,7. ניתן לראות בין 3 החלוניות את הבקשה לשליחה נוספת על די

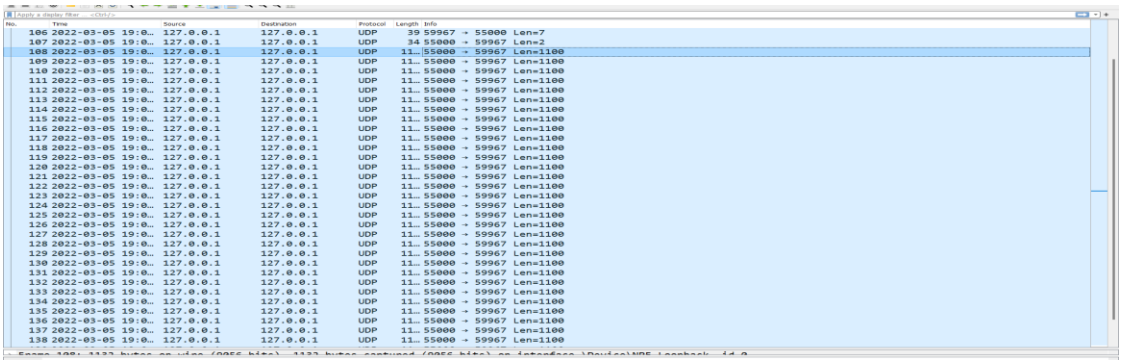
NACK5 AND NACK6 AND NACK7 ולאחר כל בקשה ניתן לראות את השליחה של השרת של אותו פקטה שלא הגיעה. (ניתן לראות את ההקלטה מתוך הקובץ lost_udp)

Wireshark Network



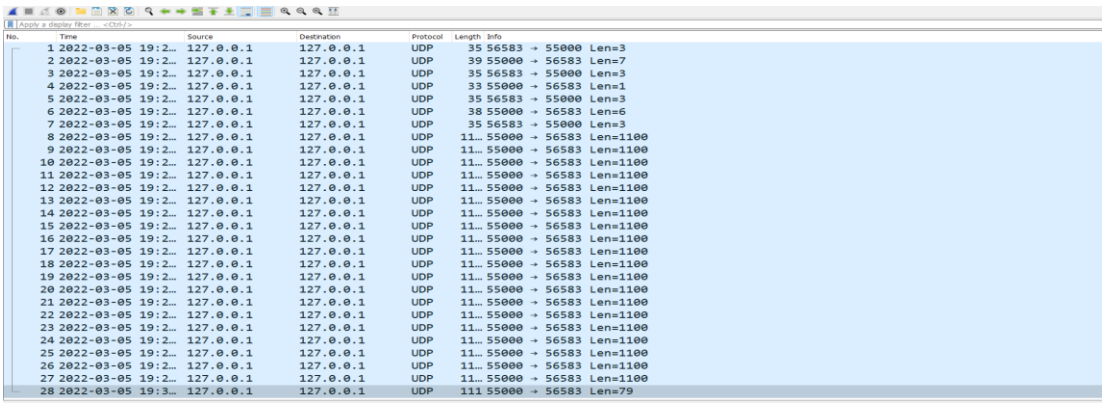
בחלון הנ"ל ניתן לראות כי לאחר קבלה של 50% מהקובץ המשתמש מבקש לא להמשיך את הורדת הקבצים.
ולכן ניתן לראות בחלונית הימנית את שליחת ה F אשר מסמלת כי המשתמש אינה רוצה להמשיך.

ולכן נשלח fin לשרת לניתוק הקשר. (ניתן לראות את ההקלטה מתוך הקובץ lost_udp)



בחלון הנ"ל ניתן לראות כי החלון גדל בצורה אקספוננציאלית לאחר קבלת 3 ACK_ALL ללא בעיות איבוד חבילות. כלומר כאשר אין בעיה של איבוד חבילות החלון גדל בצורה אקספוננציאלית.

כאשר קיים איבוד חבילות החלון גדל בצורה לינארית. וכאשר יש time out החלון מתקטן לחצי(ניתן לראות את ההקלטה מתוך הקובץ lost_udp)

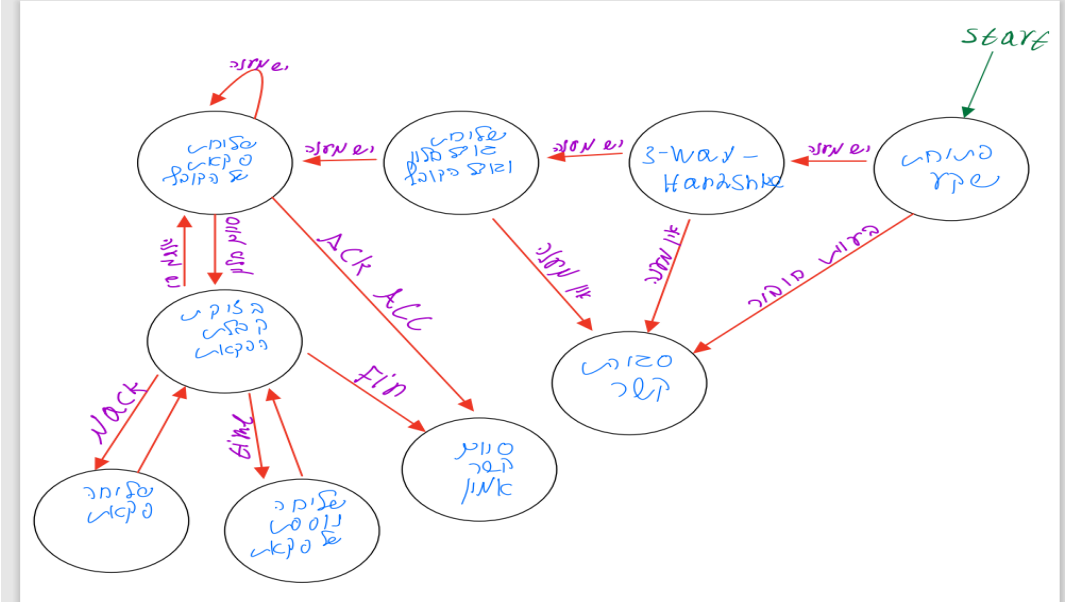


בחלון הנ"ל ניתן לראות כי השרת שלח 3 פעמים את אותם פקטות עקב אי קבלת תשובה מצד הלקוח (לאחר שזמן התגובה הסתיים). ולכן לאחר כ 3 פעמים של שליחת אותם פקטות השרת סוגר את השקע עקב אי קבלת תשובה מצד הלקוח. (ניתן לראות את ההקלטה בקובץ time_udp)

שאלות ותשובות חלק ב:

שאלה: ציירו דיאגרמת מצבים בהם המערכת עובדת

תשובה:



הסבר: פותחים שקע של UDP בין 2 הצדדים. פותחים בין 2 הצדדים את הקשר על ידי 3 way-Handshake.

במקרה ואין מענה נגיע למצב בור אשר סוגר את הקשר. אם קיים מענה אזי נמשיך בשליחת גדול החלון וגודל קובץ בין 2 הצדדים. אם קיים מענה נמשיך לשליחת הקובץ. כאשר ה windo התמלא אזי נבדוק שכל הפקטות הגיעו. במידה ולא נגיע לאחד המצבים אשר שולח את הפקטות החסרות. לאחר מכן נמשיך בשליחה נוספת עד לקבלת ACK_ALL .

שאלות ותשובות חלק ב:

שאלה: כיצד המערכת מתגברת על איבוד חבילות

תשובה:

במערכת שלנו כאשר נוצר איבוד חבילות אזי אצל הלקוח נכנס לתוך מבנה נתונים מספר הסידורי אשר לא הגיע ללקוח. זאת אומרת כאשר הלקוח מקבל את הפקטות הוא מקבל בנוסף לכך את מספר הסידורי של אותו פקטה. ולכן כאשר הלקח מגלה שאין רצף סדיר בקבלת הפקטה הבאה אזי הלקוח מכניס לתוך מבנה נתונים את הפקטות החסרות ולאחר שמסתיים גודל החלון הלקוח שולח לשרת בקשה לשליחה נוספת של הפקטות החסרות . כתוצאה מכך השרת מקבל את ההודעה על הפקטות האבודות ,שולח ללקוח את הפקטות החסרות וכן מקטין את גדול החלון עקב איבוד חבילות.

איך יודע להקטין או להגדיל את החלון:

congestion control:

א. גודל החלון בהתחלה הוא 5.
ב. אם לא קיים איבוד חבילות אזי החלון גדל בצורה אקספוננציאלית.
ג. אם קיים איבוד חבילות אזי החלון קטן בגודל של איבוד החבילות ולאחר מכן גדל בצורה לינארית.
ד. לאחר כשלוש פעמים שאין איבוד חבילות של אותו חלון. החלון הופך לגדול בצורה אקספוננציאלית.

לסיכום:

בעזרת שליחה נוספת של החבילות האבודות והקטנת החלון המערכת שלנו מתגברת על איבוד החבילות.

שאלה: כיצד המערכת מתגברת על בעיות latency

תשובה:

במערכת שלנו קיים מנגנון של זמנים. כלומר כאשר השרת שולח הודעה ללקוח ולאחר פרק זמן מצפה ממנו לקבל תשובה- במידה ועבר פרק הזמן הקצוב לכל פקטה, השרת שולח פעם נוספת את כל הפקטות של כל חלון השליחה פעם נוספת. במצב זה השרת חוזר על התהליך כ 3 פעמים. במידה ולאחר 3 פעמים הלקוח לא עונה לשרת – השרת מנתק את הקשר. יש לציין כי אם קיים השהייה ברשת החלון קטן פי 2.

לסיכום:

בזכות תזמון זמנים לכל פקטה וכן הקטנת המחלון המערכת מתגברת על הבעיה הנ"ל.

איך להריץ + רעיון כללי

<u>להפעלת השרת:</u>

נכנסים ל cmd ומקשים את התניב לתיקיית server . לאחר מכן כותבים python server.py

<u>להפעלת הלקוח:</u>

נכנסים ל cmd ומקשים את התניב לתיקיית client . לאחר מכן כותבים python client.py

<u>רעיון הכללי של UDP אמין</u>

רצף פקטות: השרת שולח את מספר החבילה יחד עם החבילה עצמה. – הדבר מאפשר ללקוח לזהות פקטות של מסודרות או חסרות.

עצור-והמתן לחלון: לאחר שליחת כל החבילות בחלון נתון (אשר משתנה), השרת ממתין לאישור הלקוח לפני שליחת החבילות האחרות. הלקוח בודק את הפקטות החסרות ומדווח לשרת.

שידור חוזר של פקטות אבודות -חזרה סלקטיבית: הלקוח מבקש השרת לשדר מחדש חבילה חסרה אחת או יותר. לאחר מכן נבדקת שוב האם כל החבילות הגיעו .

סידר מחדש של פקטות: פקטות לא מסודרות מסודרות מחדש אצל הלקוח ברגע שכל החבילות התקבלות בחלון הנתון.

אימות לקוח: על הלקוח ליצור תחילה חיבור עם השרת לפני שהוא יוכל לבקש קבצים. השרת משיג זאת על ידי אחסון כתובת ה-IP של הלקוח ומספר היציאה ברשימה כאשר הלקוח מתחבר לראשונה לשרת. כאשר הוא מקבל חבילת בקשה לשליחה, הוא צולב את פרטי הלקוח עם רשימת הלקוחות שלו. אם נמצא התאמה, הוא שולח חבילת ACK כדי לאשר אימות, אחרת הוא דוחה את הבקשה על ידי שליחת חבילת NACK. – כלומר לחיצת ידיים בין השרת ללקוח.

בדיקת גודל חלון: השרת מבטיח שגודל החלון של הלקוח ושל השרת יהיה תמיד אותו דבר. – הדבר נעשה על ידי שליחת גודל החלון כל פעם לאחר סיום החלון.

congestion control:

א. גודל החלון בהתחלה הוא 5.
ב. אם לא קיים איבוד חבילות אזי החלון גדל בצורה אקספוננציאלית.
ג. אם קיים איבוד חבילות אזי החלון קטן בגודל של איבוד החבילות ולאחר מכן גדל בצורה לינארית.
ד. לאחר כשלוש פעמים שאין איבוד חבילות של אותו חלון. החלון הופך לגדול בצורה אקספוננציאלית.