מימוש פרוטוקול QUIC ב Python-

מבוא

פרוטוקול QUIC הוא פרוטוקול העברת נתונים מאובטח ויעיל שפותח על ידי Google, שמטרתו לשפר את ביצועי התקשורת ברשת בהשוואה לפרוטוקולים קיימים כמו TCP. בפרויקט זה, נבצע מימוש מנוון של פרוטוקול QUIC ב-Python, המאפשר תקשורת בין לקוח לשרת להעברת קבצים בצורה יעילה.

מסגרות (Frames)

בפרוטוקול QUIC, מסגרות (Frames) משמשות להעברת יחידות נתונים בסיסיות בתוך חבילות (Packets).

כל מסגרת מכילה מידע על סוג הנתונים, מזהה זרימה (Stream ID), מיקום (Offset) ואורך הנתונים.

מבנה המסגרת:

Frame type: מציין את סוג המסגרת (כגון HANDSHAKE, ACK, DATA, CLOSE).

Stream id: מזהה הזרימה שאליה משתייכת המסגרת.

Offset: מציין את המיקום של הנתונים בזרימה.

Length: אורך הנתונים במסגרת.

במימוש שלנו, נעשה שימוש בפונקציות אסינכרוניות לטיפול במסגרות, מה שמאפשר לנו להתמודד עם קלט ופלט בצורה יעילה יותר.

חבילות (Packets)

חבילות (Packets) הן אובייקטים שמכילים אחת או יותר מסגרות ומשמשות להעברת הנתונים ברשת. כל חבילה כוללת כותרת שמכילה מידע חיוני לתקשורת, כגון מספר סידורי (Packet Number), מזהה החיבור (Connection ID), ועוד.

מבנה החבילה:

- Header form: מציין את פורמט הכותרת.
- Flags: דגלים שונים המשמשים לניהול התקשורת.
- Dest connection id: מזהה חיבור היעד.
- Packet number: מספר סידורי של החבילה.
- Frames: רשימת המסגרות הכלולות בחבילה.

ניהול זרימות (Streams)

בפרוטוקול QUIC, זרימות (Streams) משמשות להעברת נתונים מרובים על גבי חיבור יחיד. כל זרימה יכולה להעביר נתונים בגדלים משתנים, והזרימות מנוהלות בצורה עצמאית בתוך החיבור.

מבנה הזרימה:

כל זרימה היא אובייקט נפרד המכיל מזהה זרימה (Stream ID), רשימת מסגרות, מונה עבור נתונים שנשלחו ומתקבלים, ועוד.

המימוש מאפשר העברה של מספר זרימות במקביל, מה שמאפשר להעביר קבצים מרובים על גבי חיבור יחיד בצורה יעילה.

ניהול הזרימות בפרוטוקול QUIC מספק יתרון משמעותי על פני פרוטוקולים אחרים כמו TCP בכך שהוא מאפשר ניהול מקבילי של מספר זרימות בצורה אסינכרונית, מה שמפחית את ההמתנה ומייעל את השימוש ברוחב הפס.

תהליך התקשורת בין הלקוח לשרת

1. שלב ההאזנה (Listen)

השרת מתחיל במצב של האזנה (Listen) לחיבורים נכנסים.

הוא יוצר סוקט UDP ומחכה להודעות (packets) מהלקוח. השרת יכול לקבל הודעה מלקוח אחד בלבד, ויתחיל בתהליך הקשר כאשר יתקבל חיבור.

2. שלב החיבור (Connect)

הלקוח מתחיל את תהליך החיבור על ידי שליחת הודעת התחלה (Handshake) לשרת. הודעת ה-Handshake כוללת מזהה חיבור מקומי (`con_id`) שנבחר אקראית. השרת מקבל את ההודעה, שומר את מזהה החיבור של הלקוח ומחזיר הודעת ACK שמכילה את מזהה החיבור שלו (`r_con_id`) .

זהו שלב חיוני לביסוס הקשר המאובטח בין הלקוח לשרת.

3. שלב פתיחת זרמים (Streams)

לאחר שהחיבור הושלם בהצלחה, הלקוח מבקש מהשרת לפתוח מספר זרמים (Streams) לשם העברת מידע. כל זרם מייצג קובץ או רצף נתונים מסוים שעובר מהשרת ללקוח. כל זרם יכול להיות מוגדר בגודל חבילות שונה, דבר שמאפשר שיפור בביצועים וניצול יעיל של רוחב הפס.

4. שלב שליחת וקבלת הודעות (Packets)

ברגע שהזרמים נפתחו, השרת מתחיל לשלוח הודעות (packets) שמכילות מסגרות (frames) לפי הסדר ללקוח. כל הודעה מכילה מספר מסגרות, וכל מסגרת כוללת נתונים או פקודות בקרה) כמו אישור קבלה - ACKאו סגירת זרם.(CLOSE -

הלקוח, מצדו, מקבל את ההודעות, מפענח את המסגרות ומבצע את הפעולות הנדרשות - כמו כתיבת הנתונים לקובץ או שליחת הודעות ACK בחזרה לשרת כדי לאשר קבלת נתונים.

5. שלב סגירת החיבור (Close)

לאחר שכל הנתונים הועברו בהצלחה והזרמים נסגרו, השרת שולח ללקוח הודעת סגירה .(CLOSE) הלקוח מקבל את ההודעה הזו, סוגר את החיבור ומסיים את התהליך.

שימוש בפעולות אסינכרוניות

במימוש זה, נעשה שימוש נרחב בפעולות אסינכרוניות (asyncio) כדי לשפר את הביצועים ולנהל את התקשורת בצורה יעילה יותר. שימוש בasyncio - מאפשר לנו לבצע מספר משימות במקביל, כגון הקשבה לחיבורים חדשים, שליחת נתונים וקבלת נתונים, מבלי לחסום את הריצה של התוכנית.

יתרונות השימוש בasyncio - כוללים:

- ניהול מקבילי של מספר חיבורים.
- שיפור ביצועים על ידי הפחתת השהיות.
- יכולת תגובה מהירה יותר לאירועים שונים בתקשורת.

הרצת הפרויקט

כדי להריץ את הפרויקט, יש לבצע את השלבים הבאים:

הרצת השרת:

יש להריץ את הקובץ Server.py באמצעות הפקודה:

```
python3 QuicServer.py [PORT]
```

התוכנית מקבלת את הפורט אליו יקשיב השרת בעת פתיחתו.

הרצת הלקוח:

יש להריץ את הקובץ Client.py באמצעות הפקודה:

```
python3 QuicClient.py [HOST] [PORT] [NUM OF STREAMS]
```

התוכנית מקבלת את כתובת הIP או שם המארח של השרת . מספר הפורט של השרת . ואת כמות הזרמים לפתיחה ומתחילה את החיבור עם השרת.

יש לוודא שהשרת מופעל לפני הלקוח כדי לאפשר את תהליך ההתחברות בצורה תקינה.

בדיקות (Tests)

בפרויקט זה, הוספנו מספר בדיקות יחידה (Unit Tests) כדי לוודא את התקינות והביצועים של כל רכיב בפרוטוקול. הבדיקות כוללות:

- בדיקת חיבור ואימות בין הלקוח לשרת.
- בדיקת העברת נתונים באמצעות זרימות מרובות.
- בדיקת סגירת חיבור וסיום תקשורת בצורה מסודרת.

כדי להריץ את הבדיקות, יש להשתמש במסגרת הבדיקות unittest של Python.