**בשני החלקים הפרוטוקול הוא ipv4**

**Part A**

**Part B**

**RUDP Code Explanation**

**Part C**

**TCP: reno & Cubic**

| **Scenario** | **Average Time (ms)** | **Average Bandwidth (MB/s)** |
| --- | --- | --- |
| Reno to Reno |  |  |
| 0% Packet Loss | 0.55 | 3908.86 |
| 2% Packet Loss | 0.60 | 3364.13 |
| 5% Packet Loss | 0.62 | 3305.19 |
| 10% Packet Loss | 0.62 | 3305.19 |
| Cubic to Reno |  |  |
| 0% Packet Loss | 0.78 | 2719.30 |
| 2% Packet Loss | 0.70 | 3009.42 |
| 5% Packet Loss | 0.61 | 3341.49 |
| 10% Packet Loss | 0.64 | 3153.87 |
| Cubic to Cubic |  |  |
| 0% Packet Loss | 0.64 | 3406.15 |
| 2% Packet Loss | 0.79 | 2648.80 |
| 5% Packet Loss | 0.81 | 2745.85 |
| 10% Packet Loss | 0.70 | 3115.86 |
| Reno to Cubic |  |  |
| 0% Packet Loss | 0.71 | 2892.67 |
| 2% Packet Loss | 0.68 | 3235.99 |
| 5% Packet Loss | 0.68 | 3045.86 |
| 10% Packet Loss | 0.86 | 2491.15 |

בעקבות החלק של הבונוס שבו הרצנו את התוכנית 4\*4 פעמים, כל פעם עם 4 אפשרויות של אובדן חבילות, ובכל ריצה שלחנו את הקובץ 5 פעמים, יצרנו קובץ גדול בגודל 2 מגה־בייט באמצעות הפונקציה שניתנה בנספח C, ושלחנו את הקובץ הגדול דרך TCP\_Sender ל־TCP\_Receiver.

לפני כן, הכנו טבלאות השוואה של הנתונים.

בסקירה של הטבלה, ניתן להבחין כי עם עליית אובדן המידע, השימוש באלגוריתם Reno גרם לירידה בביצועים. מתוך זאת הגענו למסקנה כי Cubic שמר על ביצועים יחסית יציבים, גם כאשר אובדן המידע עלה. ללא אובדן מידע, Reno ל Reno הציגה את הביצועים הטובים ביותר.

**\*\*\*בכדי למנוע כפילויות\*\*\***

+ נשים לב שבכל הריצות, למרות שיש 0% איבוד פאקט לוס עדיין יש איבוד מידע!

+ מקרה שכיח, wireshark לא מקבל ack , שולח ack כפול, מפספס ack כפול ברצף

**From Reno to Reno**

* **0% packet loss**

\* Statistics \*

Run #1 Data: Time=0.77 ms ; Speed=2604.17MB/s

Run #2 Data: Time=0.67 ms ; Speed=2967.36MB/s

Run #3 Data: Time=0.44 ms ; Speed=4494.38MB/s

Run #4 Data: Time=0.39 ms ; Speed=5102.04MB/s

Run #5 Data: Time=0.46 ms ; Speed=4376.37MB/s

- Average time: 0.55 ms

- Average bandwidth: 3908.86 MB/s

****

* **2% packet loss**

\* Statistics \*

Run #1 Data: Time=0.70 ms ; Speed=2853.07MB/s

Run #2 Data: Time=0.51 ms ; Speed=3944.77MB/s

Run #3 Data: Time=0.58 ms ; Speed=3454.23MB/s

Run #4 Data: Time=0.61 ms ; Speed=3257.33MB/s

Run #5 Data: Time=0.60 ms ; Speed=3311.26MB/s

- Average time: 0.60 ms

- Average bandwidth: 3364.13 MB/s

****

* **5% packet loss**

\* Statistics \*

Run #1 Data: Time=0.65 ms ; Speed=3081.66MB/s

Run #2 Data: Time=0.50 ms ; Speed=3992.02MB/s

Run #3 Data: Time=0.76 ms ; Speed=2621.23MB/s

Run #4 Data: Time=0.60 ms ; Speed=3322.26MB/s

Run #5 Data: Time=0.57 ms ; Speed=3508.77MB/s

- Average time: 0.62 ms

- Average bandwidth: 3305.19 MB/s

****

* **10% packet loss**

\* Statistics \*

Run #1 Data: Time=0.65 ms ; Speed=3081.66MB/s

Run #2 Data: Time=0.50 ms ; Speed=3992.02MB/s

Run #3 Data: Time=0.76 ms ; Speed=2621.23MB/s

Run #4 Data: Time=0.60 ms ; Speed=3322.26MB/s

Run #5 Data: Time=0.57 ms ; Speed=3508.77MB/s

- Average time: 0.62 ms

- Average bandwidth: 3305.19 MB/s

****

**From Reno to Cubic**

* **0% packet loss**

\* Statistics \*

Run #1 Data: Time=1.21 ms ; Speed=1657.00MB/s

Run #2 Data: Time=0.64 ms ; Speed=3129.89MB/s

Run #3 Data: Time=0.73 ms ; Speed=2739.73MB/s

Run #4 Data: Time=0.72 ms ; Speed=2758.62MB/s

Run #5 Data: Time=0.60 ms ; Speed=3311.26MB/s

- Average time: 0.78 ms

- Average bandwidth: 2719.30 MB/s

****

* **2% packet loss**

\* Statistics \*

Run #1 Data: Time=0.99 ms ; Speed=2028.40MB/s

Run #2 Data: Time=0.62 ms ; Speed=3241.49MB/s

Run #3 Data: Time=0.55 ms ; Speed=3669.72MB/s

Run #4 Data: Time=0.57 ms ; Speed=3496.50MB/s

Run #5 Data: Time=0.77 ms ; Speed=2610.97MB/s

- Average time: 0.70 ms

- Average bandwidth: 3009.42 MB/s

****

* **5% packet loss**

\* Statistics \*

Run #1 Data: Time=0.63 ms ; Speed=3194.89MB/s

Run #2 Data: Time=0.52 ms ; Speed=3838.77MB/s

Run #3 Data: Time=0.51 ms ; Speed=3937.01MB/s

Run #4 Data: Time=0.62 ms ; Speed=3205.13MB/s

Run #5 Data: Time=0.79 ms ; Speed=2531.65MB/s

- Average time: 0.61 ms

- Average bandwidth: 3341.49 MB/s

****

* **10% packet loss**

\* Statistics \*

Run #1 Data: Time=0.60 ms ; Speed=3327.79MB/s

Run #2 Data: Time=0.53 ms ; Speed=3759.40MB/s

Run #3 Data: Time=0.71 ms ; Speed=2824.86MB/s

Run #4 Data: Time=0.69 ms ; Speed=2894.36MB/s

Run #5 Data: Time=0.68 ms ; Speed=2962.96MB/s

- Average time: 0.64 ms

- Average bandwidth: 3153.87 MB/s

****

**From Cubic to Reno**

* **0% packet loss**

\* Statistics \*

Run #1 Data: Time=0.92 ms ; Speed=2178.65MB/s

Run #2 Data: Time=0.61 ms ; Speed=3267.97MB/s

Run #3 Data: Time=0.76 ms ; Speed=2635.05MB/s

Run #4 Data: Time=0.58 ms ; Speed=3466.20MB/s

Run #5 Data: Time=0.69 ms ; Speed=2915.45MB/s

- Average time: 0.71 ms

- Average bandwidth: 2892.67 MB/s

****

* **2% packet loss**

\* Statistics \*

Run #1 Data: Time=1.17 ms ; Speed=1703.58MB/s

Run #2 Data: Time=0.56 ms ; Speed=3558.72MB/s

Run #3 Data: Time=0.57 ms ; Speed=3502.63MB/s

Run #4 Data: Time=0.55 ms ; Speed=3669.72MB/s

Run #5 Data: Time=0.53 ms ; Speed=3745.32MB/s

- Average time: 0.68 ms

- Average bandwidth: 3235.99 MB/s

****

* **5% packet loss**

\* Statistics \*

Run #1 Data: Time=0.87 ms ; Speed=2296.21MB/s

Run #2 Data: Time=0.54 ms ; Speed=3738.32MB/s

Run #3 Data: Time=0.63 ms ; Speed=3154.57MB/s

Run #4 Data: Time=0.79 ms ; Speed=2525.25MB/s

Run #5 Data: Time=0.57 ms ; Speed=3514.94MB/s

- Average time: 0.68 ms

- Average bandwidth: 3045.86 MB/s

****

* **10% packet loss**

\* Statistics \*

Run #1 Data: Time=1.28 ms ; Speed=1560.06MB/s

Run #2 Data: Time=0.75 ms ; Speed=2670.23MB/s

Run #3 Data: Time=0.64 ms ; Speed=3144.65MB/s

Run #4 Data: Time=0.66 ms ; Speed=3044.14MB/s

Run #5 Data: Time=0.98 ms ; Speed=2036.66MB/s

- Average time: 0.86 ms

- Average bandwidth: 2491.15 MB/s

****

**From Cubic to Cubic**

* **0% packet loss**

\* Statistics \*

Run #1 Data: Time=0.90 ms ; Speed=2232.14MB/s

Run #2 Data: Time=0.63 ms ; Speed=3154.57MB/s

Run #3 Data: Time=0.56 ms ; Speed=3546.10MB/s

Run #4 Data: Time=0.38 ms ; Speed=5277.04MB/s

Run #5 Data: Time=0.71 ms ; Speed=2820.87MB/s

- Average time: 0.64 ms

- Average bandwidth: 3406.15 MB/s

****

* **2% packet loss**

\* Statistics \*

Run #1 Data: Time=1.06 ms ; Speed=1888.57MB/s

Run #2 Data: Time=0.75 ms ; Speed=2656.04MB/s

Run #3 Data: Time=0.54 ms ; Speed=3717.47MB/s

Run #4 Data: Time=0.90 ms ; Speed=2234.64MB/s

Run #5 Data: Time=0.73 ms ; Speed=2747.25MB/s

- Average time: 0.79 ms

- Average bandwidth: 2648.80 MB/s



* **5% packet loss**

\* Statistics \*

Run #1 Data: Time=1.32 ms ; Speed=1514.00MB/s

Run #2 Data: Time=0.72 ms ; Speed=2789.40MB/s

Run #3 Data: Time=0.51 ms ; Speed=3898.64MB/s

Run #4 Data: Time=0.88 ms ; Speed=2285.71MB/s

Run #5 Data: Time=0.62 ms ; Speed=3241.49MB/s

- Average time: 0.81 ms

- Average bandwidth: 2745.85 MB/s

****

* **10% packet loss**

\* Statistics \*

Run #1 Data: Time=0.42 ms ; Speed=4728.13MB/s

Run #2 Data: Time=0.59 ms ; Speed=3367.00MB/s

Run #3 Data: Time=0.66 ms ; Speed=3044.14MB/s

Run #4 Data: Time=1.05 ms ; Speed=1908.40MB/s

Run #5 Data: Time=0.79 ms ; Speed=2531.65MB/s

- Average time: 0.70 ms

- Average bandwidth: 3115.86 MB/s

****

**Part C**

1. כפי שנתקלנו בניתוח הנתונים, כאשר אין אובדן נתונים, רנו תמיד הציגה את הביצועים הטובים ביותר. אך עם עליית האובדן בנתונים, קיוביק הציגה ביצועים יציבים יחסית בהשוואה לרנו, שהראתה ירידה חדה יותר.

כתוצאה מכך, נראה שקיוביק עשויה להיות פתרון יעיל יותר בתנאים של אובדן מידע מתמשך.

1. לאחר השוואת הביצועים של שני הפרוטוקולים, בחרנו להשתמש ב־Reno עבור TCP. ההשוואה הדגישה את יעילותו הרבה יותר של TCP ביחס ל־RUDP שלנו, במיוחד כאשר התקבלה איבוד גבוהה של מידע. מהירות העברת המידע הממוצעת הייתה גבוהה יותר עבור TCP, והזמן הממוצע לשליחת הקובץ היה נמוך יותר. בנוסף, במימוש שלנו של RUDP, נעסקנו בבקרת זרימה על ידי שיטת "stop and wait", שהיא פחות מהירה מהשיטות האחרות. דגשנו על העברת מידע אמינה יותר מאשר על העברה מהירה של המידע.

להלן זמני הריצה שיצאו לנו בארבעת המקרים:

rudp

0% Packet Loss

Run #1 Data: Time=0.008516 S ; Speed=234.852043 MB/S

Run #2 Data: Time=0.008787 S ; Speed=227.608968 MB/S

Run #3 Data: Time=0.006891 S ; Speed=290.233638 MB/S

Run #4 Data: Time=0.010070 S ; Speed=198.609732 MB/S

Run #5 Data: Time=0.005985 S ; Speed=334.168755 MB/S

Average time: 0.008050 S

Average speed: 257.094627 MB/S

2% Packet Loss

Run #1 Data: Time=0.006650 S ; Speed=300.751880 MB/S

Run #2 Data: Time=0.006368 S ; Speed=314.070352 MB/S

Run #3 Data: Time=0.006946 S ; Speed=287.935502 MB/S

Run #4 Data: Time=0.006348 S ; Speed=315.059861 MB/S

Run #5 Data: Time=0.006789 S ; Speed=294.594196 MB/S

Average time: 0.006620 S

Average speed: 302.482358 MB/S

5% Packet Loss

Run #1 Data: Time=0.006642 S ; Speed=301.114122 MB/S

Run #2 Data: Time=0.006349 S ; Speed=315.010238 MB/S

Run #3 Data: Time=0.006503 S ; Speed=307.550361 MB/S

Run #4 Data: Time=0.005579 S ; Speed=358.487184 MB/S

Run #5 Data: Time=0.006271 S ; Speed=318.928401 MB/S

Average time: 0.006269 S

Average speed: 320.218061 MB/S

10% Packet Loss

Run #1 Data: Time=0.007292 S ; Speed=274.273176 MB/S

Run #2 Data: Time=0.007387 S ; Speed=270.745905 MB/S

Run #3 Data: Time=0.009089 S ; Speed=220.046210 MB/S

Run #4 Data: Time=0.012500 S ; Speed=160.000000 MB/S

Run #5 Data: Time=0.008718 S ; Speed=229.410415 MB/S

Average time: 0.008997 S

Average speed: 230.895141 MB/S

1. למרות קיום מימושים יעילים ומהירים יותר של RUDP, בהתבסס על המימוש שלנו ועל ניתוח התוצאות שהצגנו, נבחר להשתמש תמיד ב־TCP.

בחירת השימוש ב־TCP מתבססת למעשה על היציבות והאמינות שהוא מציע עבור הצרכים שלנו.

**Part D – Open**

**שאלה 1**

הגדלת סף ההתחלה האיטית (SSThreshold) ב-TCP בתחילת חיבור מועילה ביותר בתרחישים שבהם יש RTT גדול (זמן נסיעה הלוך ושוב) והחיבור ארוך. הסיבה לכך היא ש-RTT גדול מציין שיש latency משמעותי ברשת, וחיבור ארוך אומר שיש כמות משמעותית של נתונים לשלוח.

אזי, התרחיש שבו השינוי הזה יהיה הכי מועיל הוא:

בקשר ארוך על גבי רשת לא אמינה עם RTT גדול.

RTTגדול: RTT גדול מרמז שלוקח זמן ניכר לחבילה לעבור מהשולח למקבל ובחזרה. במקרים כאלה, האלגוריתם של TCP Slow Start עלול לגרום לכך שהשולח לא ינצל את רוחב הפס הזמין באותו הרגע במהלך השלב הראשוני של החיבור בשל האופי העקבי של בקרת הצפיפות של TCP.

קשר ארוך: בחיבור ארוך, יש כמות משמעותית של נתונים להעביר. אם השולח עקבי מדי בקצב השידור הראשוני שלו בגלל SSTthreshold קטן, עשוי לקחת די הרבה זמן עד לקצב השליחה האופטימלי, מה שיוביל לתפוקה לא אופטימלית לאורך כל החיבור, לכן כדאי לקחת קשר ארוך על מנת למנוע מצב שבו קצב השליחה יקטן/יאבד פקטות בגלל עומס יתר.

רשת לא אמינה: רשת לא אמינה מציגה אובדן פקטות, מה שיכול להפעיל מנגנוני בקרת עומס של .TCP

על ידי הגדלת SSTthreshold בתחילת החיבורTCP , יכול להשיג קצב שליחה ראשוני גבוה יותר, עם פוטנציאל למלא את צינור הרשת מהר יותר לפני שמתרחשים אירועי עומס הגורמים לאיבוד פקטות.

דבר זה יכול לעזור להפחית את ההשפעה של אובדן פקטות על הthroughput, במיוחד בתרחישים שבהם יש RTT גדול ומשך חיבור ארוך.

לסיכום, הגדלת SSTthreshold ב-TCP בתחילת קשר ארוך ברשת לא אמינה עם RTT גדול יכולה לסייע באופטימיזציה של התפוקה בכך שהיא מאפשרת לשולח להעלות את מהירות קצב השליחה שלו ולנצל טוב יותר את רוחב הפס הזמין לפני שמתרחשים אירועי עומס, ובכך ממקסמים את ההשפעה המירבית של הגדלת ה SSTthreshold בתחילת הקשר.

**שאלה 2**

בהתחשב בכך ש-

ניתן להסיק ש S צריכה להיות קטנה מ

כמו כן, מכיוון שהחיבור מסתיים בפעם הראשונה שהוא מגיע לביטוי

הערך של S לא צריך לחרוג מ

דרך חישוב: נתחיל לשלוח 1, ובכל פעם נכפיל ב2 (בלי איבוד מידע ולכן לא יפול אף פעם).

ולכן כמות הפעמים שנשלח הוא , וכל חבילה עם זמן העברה RTT

מכן נקבל שהזמן שיתקבל הוא log s \* RTT .

ולכן נסכום מi=0 עד את 2 בחזקת i . וכל שליחה בגודל mss לפי הנתונים.

חישוב הסיגמה מראה ש-א׳ נכונה.

התפוקה תהיה ~

**שאלה 3**

1. **Round Time Trip (RTT):**

1. **Window Size:**
2. **Delay Time:**
3. **Transmission Delay:**

**Packet Size = 4 Kbyte = 4000 bytes**

**Distance = 1000 meters**

**Propagation = 2 x 10^8**

**bandwidth = 8 Gbps = 8,000,000,000 Bps**

1. x = 4
2. נציב בנוסחות הנתונות למציאת Window Sizeכאשר אנחנו מקבלים תפוקה מקסימלית.

נמצא את ה :

נמצא את ה :

נמצא את ה :

לבסוף נחשב את ה Window Size בעזרת מה שחישבנו:

קיבלנו שכדי למקסם את הthroughput הwindow size שלנו יהיה בגודל 251 פקטות, )בלי איבוד נתונים)

כאשר המצב הוא עם הנתונים שקיבלנו בשאלה.