دانشکده‌ی مهندسی کامپیوتر

شبکه‌های کامپیوتری

نیم‌سال دوم ۰۴-۱۴۰۳

استاد: امیرمهدی صادق‌زاده

تمرین دوم

پاسخ‌دهنده: معین آعلی - ۴۰۱۱۰۵۵۶۱

فهرست مسائل

پاسخ مسئله‌ی ۱.

آ

پردازه‌های A و B به عنوان کلاینت عمل می‌کنند و از پورت‌های موقت استفاده می‌کنند. (Ephemeral-Port) اما پردازه C که در این سناریو به عنوان سرور HTTP است، که باید با یک پورت ثابت کار کند. چون صورت سوال اشاره کرده است که در پروتوکل ارتباطی ما HTTP است، پس پورت پردازه C برابر با ۸۰ است.

ب

در حالت عادی، این امکان وجود ندارد. چون که پورت و آیپی با هم Socket-Address را تشکیل می‌دهند و باید یک کلید یکتا باشد. پس اگر یک پردازه بخواهد روی یک پورت خاص bind کند، سیستم‌عامل آن پورت را برای این پردازه رزرو می‌کند و دیگر اجازه نمی‌دهد که پردازه دیگری روی آن پورت bind کند. اما حالات استثنا و خاص هم وجود دارد، مانند:

- فرض کنیم یک هاست دارای چند آیپی است. می‌توان یک پردازه روی یک آیپی و پورت و پردازه دیگر روی همان پورت و آیپی دیگر گوش کند.
- با استفاده از قابلیت SO-REUSEPORT می‌توان کاری کرد که دو پردازه روی یک پورت گوش دهند و سیستم‌عامل بین آن‌ها load-balancing انجام دهد.
- و موارد دیگر...

ج

با اینکه در لایه‌ی لینک روش‌هایی برای تشخیص خطا وجود دارد، اما این روش‌ها فقط خطاهای local را در هر لینک مجزا بین دو گره تشخیص می‌دهند. در مسیر یک بسته از مبدا تا مقصد، بسته از چندین لینک و روتر عبور می‌کند و خطاهایی ممکن است در بافر روترها یا حافظه‌ی آن‌ها رخ دهد. پس برای اطمینان از اینکه بسته به برنامه‌ی مقصد به درستی رسیده است، نیاز است در لایه‌ی انتقال یک checksum داشته باشیم.

پاسخ مسئله‌ی ۲.

- UDP ساده‌تر از TCP است (بدون کنترل اتصال، شماره‌گذاری بسته‌ها یا کنترل ازدحام)، پس سرعت انتقال داده بیشتر می‌شود.
- نیاز به حداقل تأخیر در برنامه‌هایی که نیاز دارند داده خیلی سریع برسد. مثل تماس صوتی یا لایو استریم. تأخیر کم مهم‌تر از تحویل تضمینی همه داده‌ها است. TCP به خاطر مکانیسم‌های تصحیح خطا، تأخیر بیشتری دارد که برای این نوع اپلیکیشن‌ها مناسب نیست.
- این اپلیکیشن‌ها طوری طراحی شده‌اند که حتی اگر برخی بسته‌ها از دست برود، همچنان به کار ادامه می‌دهند (مثلاً در تماس صوتی، یک کلمه گم شود ولی مکالمه قطع نشود).

برای مثال می‌توان به برنامه تماس تصویری و تماس صوتی و بازی‌های انلاین اشاره کرد.

پاسخ مسئله‌ی ۳.

آ

در پروتکل **Go-Back-N**، اندازه پنجره نقش بسیار مهمی در کارایی و کارکرد صحیح شبکه دارد. در صورتی که اندازه پنجره به درستی انتخاب نشود، دو حالت مشکل‌زا ممکن است رخ دهد:

پنجره بسیار کوچک

- کارایی پایین شبکه: ارسال‌کننده نمی‌تواند بسته‌های زیادی را قبل از دریافت تایید ارسال کند، که منجر به زمان‌های بیکار زیاد می‌شود.
- استفاده ناکارآمد از پهنای باند: ظرفیت لینک به خوبی استفاده نمی‌شود، زیرا اغلب اوقات ارسال‌کننده منتظر دریافت تأییدها است.

پنجره بسیار بزرگ

- **سرریز بافر گیرنده (Receiver-Buffer-Overflow)**: اگر گیرنده نتواند با سرعت کافی بسته‌ها را دریافت و پردازش کند، بسته‌ها از بین خواهند رفت.
- **افزایش احتمال ارسال مجدد بسته‌ها**: در صورت از دست رفتن یک بسته، همه بسته‌های بعدی باید دوباره ارسال شوند که باعث افزایش بار شبکه می‌شود.

اندازه بهینه پنجره معمولاً بر اساس رابطه زیر تعیین می‌شود:

$$\text{Window-Size}_{\text{optimal}} = \text{Bandwidth-Delay-Product} + ۱$$

که در آن Bandwidth-Delay-Product بیانگر مقدار داده‌ای است که می‌تواند در مسیر شبکه در جریان باشد (در حال انتقال). عدد ۱ اضافه می‌شود تا اطمینان حاصل شود که حتی در هنگام انتظار برای تأیید آخرین بسته، ارسال‌کننده بتواند ارسال را ادامه دهد.

انتخاب اندازه پنجره باید به گونه‌ای باشد که هم کارایی شبکه را افزایش دهد و هم از سرریز بافر جلوگیری کند. این اندازه معمولاً با محاسبه دقیق BDP و اعمال حاشیه امن به دست می‌آید.

ب

داده‌های مسئله:

- $۱۰۰\text{ms} = \text{RTT}$
- اندازه هر بسته = ۱۰۰۰ بایت = ۸۰۰۰ بیت
- نرخ لینک = $۱\text{ Gbps} = ۱۰^۹\text{ bps}$
- احتمال خطا در هر بسته = p

زمان ارسال هر بسته:

$$t_{\text{trans}} = \frac{۸۰۰۰}{۱۰^۹} = ۸\ \mu s$$

پروتکل Stop-and-Wait

در Stop-and-Wait ، پس از ارسال هر بسته باید یک RTT صبر کنیم.
کارایی:

$$U_{\text{SW}} = \frac{t_{\text{trans}}}{t_{\text{trans}} + RTT} = \frac{۸ \times ۱۰^{-۶}}{۸ \times ۱۰^{-۶} + ۰/۱} \approx ۷/۹۹۹ \times ۱۰^{-۵}$$

یعنی کارایی حدودا ٪۰/۰۰۸ است.
کارایی با احتمال خطا p:

$$U_{\text{SW}} = \frac{t_{\text{trans}}}{t_{\text{trans}} + RTT} \times (۱ - p)$$

$$U_{\text{SW}} \approx ۷/۹۹۹ \times ۱۰^{-۵} \times (۱ - p)$$

پروتکل Go-Back-N

ابتدا اندازه پنجره N را محاسبه می‌کنیم:

$$N = \frac{RTT}{t_{\text{trans}}} = \frac{۰/۱}{۸ \times ۱۰^{-۶}} = ۱۲۵۰۰$$

بنابراین، $N = ۱۲۵۰۰$ برای پر کردن خط لوله کافی است.
در حالت بدون خطا:

$$U_{\text{GBN}} = \frac{N \times t_{\text{trans}}}{RTT + t_{\text{trans}}} \approx \frac{۱۲۵۰۰ \times ۸ \times ۱۰^{-۶}}{۰/۱} = ۱$$

یعنی کارایی ۱۰۰% در حالت بدون خطا.
در حضور خطا با احتمال p، کارایی کاهش می‌یابد. کارایی تقریبی:

$$U_{\text{GBN}} \approx \frac{۱}{۱ + p(N - ۱)}$$

Repeat Selective

در این حالت نیز پنجره N به همان مقدار است.
در حالت بدون خطا:

$$U_{\text{SR}} = ۱$$

در حضور خطا:

$$U_{\text{SR}} \approx ۱ - p$$

زیرا فقط بسته‌های دارای خطا دوباره ارسال می‌شوند.

با افزایش p :

- در Stop-and-Wait ، کارایی به شدت پایین است و تغییر زیادی با خطا نمی‌کند زیرا در بهترین حالت هم کارایی بسیار کم است.
- در Go-Back-N ، با افزایش p، کارایی شدیدا افت می‌کند زیرا باید همه بسته‌های بعد از یک خطا را دوباره ارسال کرد.
- در Selective-Repeat ، افت کارایی بسیار ملایم‌تر است، زیرا فقط بسته‌های معیوب بازفرست می‌شوند.

برای لینک‌هایی که احتمال خطا در آن‌ها زیاد است، استفاده از پروتکل Selective-Repeat مناسب‌تر است زیرا کارایی آن در مقابل خطا مقاوم‌تر است.

ج

پروتکل Go-Back-N

در این پروتکل اگر یک بسته گم شود، تمام بسته‌های بعدی (تا انتهای پنجره) باید دوباره ارسال شوند.
تعداد متوسط بسته‌های ارسالی:

$$\text{Expected-transmissions-per-packet} = \frac{۱}{۱-p}$$

اما، چون در Go-Back-N از لحظه‌ای که یک خطا رخ دهد تمام بسته‌های بعدی نیز دوباره ارسال می‌شوند، میانگین تعداد بسته‌های اضافی در اثر خطا تقریباً:

$$\text{Overhead-per-lost-packet} \approx N - ۱$$

پس، تعداد متوسط بسته‌های ارسالی برای ارسال M بسته:

$$\text{Total-packets} \approx M \times \left(\frac{۱}{۱-p} + p \times (N-۱) \right)$$

تعداد متوسط ACK ها:

$$\text{Total-ACKs} = M$$

چون فقط بسته‌های دریافت‌شده نیازمند ACK هستند و ACK ها گم نمی‌شوند.

پروتکل Repeat Selective

$$\text{Expected-transmissions-per-packet} = \frac{۱}{۱ - p}$$

پس، برای M بسته:

$$\text{Total-packets} = M \times \frac{۱}{۱ - p}$$

تعداد متوسط ACK ها:

$$\text{Total-ACKs} = M$$

چون هر بسته‌ای که دریافت می‌شود، بلافاصله ACK می‌شود.

پروتکل TCP

در TCP بدون ACK Delayed برای هر بسته دریافت‌شده، بلافاصله یک ACK ارسال می‌شود.
تعداد متوسط بسته‌های ارسالی:
TCP از Repeat Selective مشابه‌تر است. در صورت از دست رفتن بسته، تنها همان بسته بازفرست می‌شود:

$$\text{Expected-transmissions-per-packet} = \frac{۱}{۱ - p}$$

پس:

$$\text{Total-packets} = M \times \frac{۱}{۱ - p}$$

تعداد متوسط ACK ها:

$$\text{Total-ACKs} = M$$

نتیجه‌گیری

- در Go-Back-N ، به علت ارسال مجدد کل پنجره پس از هر خطا، تعداد بسته‌های ارسالی بسیار بیشتر از Selective-Repeat و TCP است.
- در Selective-Repeat و TCP ، فقط بسته‌های از دست‌رفته دوباره ارسال می‌شوند و تعداد ACK ها برای هر سه پروتکل برابر با تعداد کل بسته‌ها است.
- بنابراین Selective-Repeat و TCP بهینه‌ترین عملکرد را از نظر تعداد بسته‌های ارسالی دارند.

پاسخ مسئله‌ی ۴.

آ

محاسبه پنجره لازم برای پر کردن لینک:

$$\text{پنجره لازم} = ۳۰ \times ۱۰^۶ \times ۰/۳ = ۹ \times ۱۰^۶ \text{ بیت} \longrightarrow \frac{۹ \times ۱۰^۶}{۱۲,۰۰۰} = ۷۵۰ \text{ سگمنت}$$

زمان رسیدن به حداکثر پنجره:

$$\text{زمان لازم} = (۷۵۰ - ۱) \times ۰/۳ = ۲۲۴/۷ \text{ ثانیه}$$

محاسبه میانگین Throughput

$$\text{میانگین اندازه پنجره} = \frac{۷۵۰}{۲} = ۳۷۵ \text{ سگمنت}$$

$$\text{Throughput} = \frac{۳۷۵ \times ۱۲,۰۰۰}{۰/۳} = ۱۵\,\text{Mbps}$$

ب

هنگامی که هاست A مشغول ارسال داده به هاست B است و در زمان t_1 دیگر داده‌ای برای ارسال ندارد، ارتباط وارد Idle می‌شود. در این شرایط دو حالت کلی وجود دارد:

- اگر مدت بیکاری کمتر از مقدارRTO (Retransmission-Timeout) باشد، معمولاً اندازه پنجره حفظ شده و ارسال با همان پارامترهای قبلی ادامه می‌یابد.
- اگر مدت بیکاری طولانی باشد و از RTO عبور کند، بهتر است اندازه پنجره (cwnd) به مقدار اولیه کاهش یابد و کنترل ازدحام مجدداً از ابتدا آغاز شود، زیرا وضعیت شبکه ممکن است تغییر کرده باشد.

پس اگر وقفه طولانی باشد، بهتر است کنترل ازدحام را از ابتدا شروع کنیم.

ج

وقتی پنجره به ۷۵ درصد مقدار قبلی کاهش می‌یابد:

TCP-Reno

به‌صورت سنتی در هنگام congestion-loss ، پنجره را نصف می‌کند. حالا اگر به‌جای نصف، فقط تا ۷۵ درصد کاهش یابد، پس از congestion-loss افت سرعت کمتر می‌شود. بعد از کاهش، دوباره با روند خطی رشد می‌کند.

TCP-CUBIC

به‌صورت پیش‌فرض در congestion-loss ، مقدار کاهشش توسط پارامتر beta تنظیم می‌شود. اگر این مقدار به ۰/۷۵ تنظیم شود، افت آن کمتر می‌شود. بعد از افت، چون افت کمتر است، قسمت منحنی پایین‌تر، سریع‌تر طی می‌شود.

وقتی پنجره دو برابر شود:

TCP-Reno

پس از congestion-loss ، ناگهان نرخ ارسال بسیار افزایش پیدا می‌کند (غیرمعمول و منجر به congestion شدید). احتمالا دوباره congestion و packet-loss شدیدتری رخ می‌دهد.

TCP-CUBIC

اگر پس از congestion-loss ، به‌جای کاهش، دو برابر شود: بلافاصله وارد ناحیه‌ای با $W > W_{MAX}$ خواهد شد. سریعا congestion جدید و loss packet رخ می‌دهد. پس الگوریتم مجبور می‌شود دوباره شدیدتر افت کند.

پاسخ مسئله‌ی ۵.

آ

این دستور قصد دارد آدرس آیپی متناظر با دامنه sharif.edu را پیدا کند. با این تفاوت که از یک DNS سرور با آدرس 4.2.2.4 استفاده می‌کند نه از DNS سرور پیش‌فرض سیستم.

پاسخ داده شده دارای اطلاعات زیر هست:

- آدرس آیپی ورژن ۴ متناظر با این دامنه
- مقدار TTL مربوط به این آدرس آیپی . این مقدار برابر ۸ است در این درخواست.
- زمان پاسخ سرور DNS برابر با ۲۲۱ msec است.
- پاسخ روی پورت ۵۳ و پروتوکل UDP ارسال شده است.
- زمان اجرای دستور. البته احتمالا در این مورد زمان ست نشده بوده و در جواب زمان Unix-Timestamp برگردانده شده است.
- اندازه‌ی پیام ارسال شده از سمت سرور که در این جواب ۵۵ بایت است.
- عبارت Truncated یعنی پاسخ DNS خیلی بزرگ بوده، داخل UDP جا نشده، و dig فقط بخشی از آن را نشان داده است.

ب

رکورد MX مشخص می‌کند که ایمیل‌هایی که به این دامنه ارسال می‌شوند، باید به کدام سرورهای ایمیل تحویل داده شوند. این درخواست از DNS سرور نوشته شده درخواست می‌کند تا رکوردهای MX دامنه‌ی sharif.edu را پیدا کند.

پاسخ داده شده دارای اطلاعات زیر هست:

- این دامنه دارای دو رکورد MX است.
- این رکوردها هردو دارای اولویت ۱۰ هستند.
- این رکوردها دارای TTL برابر ۶۰ هستند.
- باقی اطلاعات مشابه با بخش قبل هستند.

وقتی ایمیلی ارسال می‌شود، سرور فرستنده تلاش می‌کند ایمیل را به سروری با کمترین عدد اولویت بفرستد. اگر چند سرور اولویت یکسان داشته باشند، یکی از آن‌ها انتخاب می‌شود (اغلب به طور تصادفی یا بر اساس load-balancing).

پاسخ مسئله‌ی ۶.

در ادامه هر یک از روش‌ها را جدا توضیح می‌دهیم:

TCP-Scanning

در این روش، اسکنر تلاش می‌کند اتصال کامل TCP را با پورت مقصد برقرار کند. یعنی ابتدا بسته‌ی SYN ارسال می‌شود، اگر پورت باز باشد SYN-ACK برمی‌گردد، و سپس ACK ارسال می‌شود تا اتصال کامل شود.
مزایا:

- بسیار دقیق است. وقتی ارتباط کامل برقرار شد، کاملاً مطمئن هستید که پورت باز است.

معایب:

- بسیار راحت توسط سیستم مقصد شناسایی می‌شود (در لاگ‌ها ثبت می‌شود).
- نسبت به روش‌های دیگر کندتر است.
- به راحتی فایروال جلوی آن را می‌تواند بگیرد

SYN-Scanning

در این روش، فقط بسته‌ی SYN ارسال می‌شود. اگر پورت باز باشد، SYN-ACK پاسخ داده می‌شود، ولی اسکنر دیگر ACK نمی‌فرستد و اتصال را نیمه‌کاره رها می‌کند.
مزایا:

- سریع‌تر از Scanning TCP است.
- در لاگ‌های سیستم هدف کمتر دیده می‌شود.

معایب:

- ممکن است توسط فایروال‌ها یا IDS/IPS ها مسدود شود.

FIN-Scanning

در این روش، بسته‌ی FIN به پورت مقصد ارسال می‌شود. اگر پورت بسته باشد، دستگاه پاسخ RST می‌دهد. اگر باز باشد، هیچ پاسخی نمی‌دهد (بر اساس استاندارد .(TCP
مزایا:

- می‌تواند از بعضی فایروال‌ها عبور کند

معایب:

- روی سیستم‌های ویندوز اغلب کار نمی‌کند، چون ویندوز به بسته FIN پاسخ RST می‌دهد حتی اگر پورت باز باشد.
- قابل اطمینان نیست مگر در سیستم‌های خاص (مثل Unix/Linux).

UDP-Scanning

در این روش، بسته UDP به پورت ارسال می‌شود. اگر پورت بسته باشد، معمولا پیام ICMP-Port-Unreachable برمی‌گردد. اگر باز باشد، معمولا پاسخی دریافت نمی‌شود.
مزایا:

- تنها روش کاربردی برای شناسایی پورت‌های باز UDP

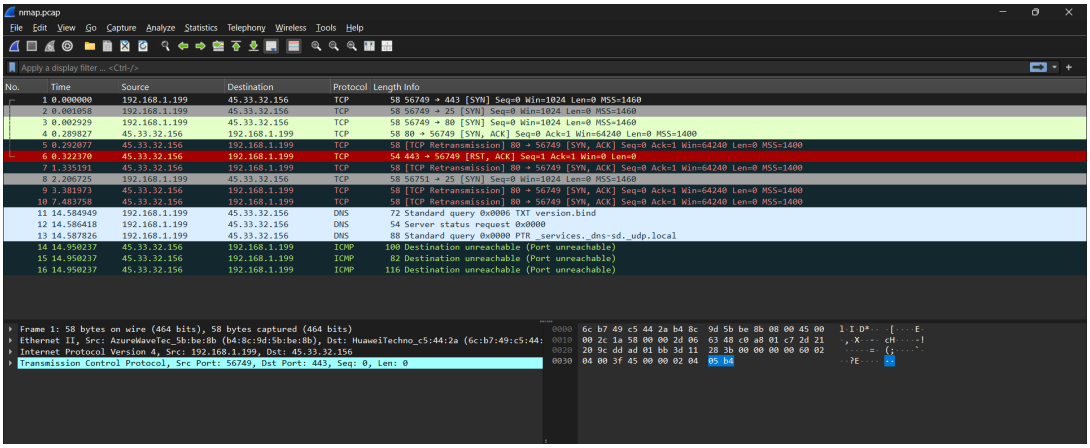
معایب:

- سخت‌ترین روش برای تشخیص وضعیت واقعی (باز یا بسته).
- کند است چون نیاز به تایم‌اوت دارد.
- ممکن است توسط فایروال‌ها مسدود شود.

جمع‌بندی موارد:

- TCP دقیق و کند و آشکار
- SYN سریع و نیمه مخفی
- FIN مخفی‌تر ولی کمتر قابل اطمینان
- UDP تنها راه برای بررسی UDP ولی کند و غیرقابل پیش‌بینی

ابتدا فایل داده شده را در نرم‌افزار Wireshark باز می‌کنیم و سپس در تب analyze و بخش filters display تعدادی رکورد برای شناسایی پورت‌های داد هشده اضافه می‌کنیم.



پورت ۲۵: با توجه به فریم ۲ و ۸ و اینکه هیچ پاسخی از سرور نیامده است، پس این پورت Filterd است.

پورت ۸۰: با توجه به فریم ۳ و ۴ و ۵ و ۷ این پورت Open است.

پورت ۴۴۳: با توجه به فریم ۱ و ۶ این پورت Closed است. چون که فلگ بسته RST فرستاده است.

پورت ۵۳: با توجه به فریم ۱۱ و ۱۲ و۱۳ و ۱۴ و ۱۶ ، این پورت Closed است. چون که در بسته ICMP کد ۳ داده که یعنی Unreachable Port .

پاسخ مسئله‌ی ۷.

ساختار کلی پروتکل بین کلاینت و سرور اینگونه است:

- ۱ بایت مختص packet-type
- ۲ بایت مختص sequence-number
- ۴ بایت مختص checksum
- باقی هم برای payload

انواع Packet ها

۰ : درخواست نام فایل

۱ : داده

۲ : ACK

۳ : ERROR

۴ : EOF

۵ : NACK

فلوی کار سرور:

منتظر درخواست است ⟵ اگر فایل نبود ERROR می‌فرستد ⟵ اگر بود داده‌ها را به بسته‌های کوچک تقسیم و ارسال می‌کند ⟵ منتظر ACK یا NACK برای هر بسته ⟵ پس از اتمام، EOF می‌فرستد.

فلوی کار کلاینت:

درخواست فایل را می‌فرستد ⟵ منتظر دریافت داده می‌ماند ⟵ بسته‌ها را با checksum بررسی می‌کند ⟵ در صورت درست بودن ACK می‌فرستد، در غیر این صورت NACK ⟵ پس از دریافت EOF دانلود تمام می‌شود.

لاگ‌های مرتبط با سرور حین اجرا فایل تست داده شده:

```
../Assignments/HW2/practical  main $X!?  v3.12.3  10:50  python3 server.py
Server listening on 0.0.0.0:5005

[WAIT] Waiting for new file request...
[REQUEST] Client ('127.0.0.1', 38635) requested file: test_1.txt
[SEND] Sent DATA packet Seq=0 Checksum=507
[ACK] Received ACK for Seq=0
[SEND] EOF packet sent. File transfer completed.

[WAIT] Waiting for new file request...
[REQUEST] Client ('127.0.0.1', 45414) requested file: test_1.txt
[SEND] Sent DATA packet Seq=0 Checksum=507
[ACK] Received ACK for Seq=0
[SEND] EOF packet sent. File transfer completed.

[WAIT] Waiting for new file request...
[REQUEST] Client ('127.0.0.1', 39150) requested file: test_2.txt
[SEND] Sent DATA packet Seq=0 Checksum=0
[ACK] Received ACK for Seq=0
[SEND] Sent DATA packet Seq=1 Checksum=0
[ACK] Received ACK for Seq=1
[SEND] Sent DATA packet Seq=2 Checksum=0
[ACK] Received ACK for Seq=2
[SEND] Sent DATA packet Seq=3 Checksum=0
[ACK] Received ACK for Seq=3
[SEND] Sent DATA packet Seq=4 Checksum=0
[ACK] Received ACK for Seq=4
[SEND] EOF packet sent. File transfer completed.

[WAIT] Waiting for new file request...
[REQUEST] Client ('127.0.0.1', 49425) requested file: test_1.txt
[SEND] Sent DATA packet Seq=0 Checksum=507
[ACK] Received ACK for Seq=0
[SEND] EOF packet sent. File transfer completed.

[WAIT] Waiting for new file request...
```

لاگ‌های مرتبط با فایل تست:

```
…/Assignments/HW2/practical  main $X!?  v3.12.3  10:51  sudo ./test.sh
########## Test cases ##########
Test 1:
    - Download a small file
    - Check if the downloaded file is the same as the original
Test 2:
    - Download a large file (~10 KB)
    - Check if the downloaded file is the same as the original
################################
Test 1 passed
Test 2 passed
```

من برای شبیه‌سازی از دست رفتن بسته‌ها و لینک غیرقابل‌اتکا، داخل کد کلاینت و سرور یک احتمال ثابتی دخیل کردم تا پکت‌ها به مشکل بخورند. سپس آن را روی یک فایل بزرگ‌تر تست کردم و پکت‌های Drop را بررسی کردم که به درستی کار کنند:

```
Server listening on 0.0.0.0:5005

[WAIT] Waiting for new file request...
[REQUEST] Client ('127.0.0.1', 45705) requested file: large.txt
[CORRUPT] Simulated corruption in packet Seq=0
[SEND] Sent DATA packet Seq=0 Checksum=29781
[ACK] Received ACK for Seq=0
[SEND] Sent DATA packet Seq=1 Checksum=29772
[ACK] Received ACK for Seq=1
[SEND] Sent DATA packet Seq=2 Checksum=29564
[ACK] Received ACK for Seq=2
[DROP] Simulated packet loss Seq=3
[TIMEOUT] Resending packet Seq=3
[SEND] Sent DATA packet Seq=3 Checksum=29758
[ACK] Received ACK for Seq=3
[SEND] Sent DATA packet Seq=4 Checksum=29772
[ACK] Received ACK for Seq=4
[DROP] Simulated packet loss Seq=5
[TIMEOUT] Resending packet Seq=5
[SEND] Sent DATA packet Seq=5 Checksum=29564
[ACK] Received ACK for Seq=5
[SEND] Sent DATA packet Seq=6 Checksum=29758
[ACK] Received ACK for Seq=6
[SEND] Sent DATA packet Seq=7 Checksum=29772
[ACK] Received ACK for Seq=7
[SEND] Sent DATA packet Seq=8 Checksum=29564
[ACK] Received ACK for Seq=8
[SEND] Sent DATA packet Seq=9 Checksum=29758
[ACK] Received ACK for Seq=9
[CORRUPT] Simulated corruption in packet Seq=10
[SEND] Sent DATA packet Seq=10 Checksum=29797
[ACK] Received ACK for Seq=10
[SEND] Sent DATA packet Seq=11 Checksum=29564
[ACK] Received ACK for Seq=11
[CORRUPT] Simulated corruption in packet Seq=12
[DELAY] Simulating delay of 0.27s for packet Seq=12
[SEND] Sent DATA packet Seq=12 Checksum=29781
```

همانطور که مشخص است، بعضی از بسته‌ها Drop شده‌اند و seq افزایش نیافته‌است و مجدد ارسال شده است.

همچنین برخی از بسته‌ها Timeout شده‌اند که در عکس لاگ سرور مشخص است. در نهایت فایل $large.txt$ به درستی و کامل برای client ارسال شده‌است.