دانشگاه صنعتی اصفهان

دانشکده مهندسی برق و کامپیوتر

پروژه پایانی درس

TCP Over UDP

شبکه های کامپیوتری

استاد درس: دکتر علی فانیان

مهلت ارسال پروژه: ۱۳ تیرماه

بهار ۱۴۰۴

لطفا پیش از انجام پروژه، به موارد زیر توجه کنید:

- حتما نام فایل نهایی بارگزاری شده باید به صورت فرمت زیر باشد. در غیراینصورت مسئولیت اشتباهات احتمالی برعهده دانشجو است. Final_Project_StudentID.pdf
- از ارسال پروژه به صورت ایمیل یا داخل تلگرام خودداری کنید. فقط پاسخ‌های ارسالی در سامانه یکتا بررسی خواهند شد.
- شباهت در کدها بررسی می‌شوند و در صورت تقلب، نمره طرفین صفر درنظر گرفته خواهد شد.
- استفاده از ابزارهای هوش مصنوعی بلامانع است، اما تسلط کامل به منطق و اجزای پروژه ضروری است.
- سوالاتی که درون پروژه مطرح شده‌اند، نیازی به تحویل کتبی ندارند اما باید روی آن‌ها تسلط داشته باشید، زیرا جزو سوالاتی هستند که در ارائه پرسیده می‌شوند.
- نمره‌ای که در پایان به پروژه شما تعلق خواهد گرفت علاوه بر پیاده‌سازی درست و کامل بخش‌های مختلف پروژه وابسته به تسلط شما هنگام ارائه به پروژه تحویل داده شده نیز خواهد بود.
- در صورت وجود هرگونه ابهام و سوال، به آیدی‌های زیر در تلگرام پیام دهید:

@amirzand815

@dzshb

@HRBx_x

مقدمه

پروتکل‌های لایه انتقال (TCP و UDP) دارای مزایا و معایب خاص خود هستند. TCP به دلیل ویژگی‌هایی نظیر اعتمادپذیری بالا، ترتیب‌دهی به بسته‌ها و مدیریت خطاها برای کاربردهایی که نیاز به انتقال داده‌های دقیق و قابل اطمینان دارند، مناسب است. با این حال، TCP ممکن است با تاخیر زیاد و مصرف منابع بیشتر همراه باشد. در مقابل، UDP به دلیل سادگی و کارایی بالا گزینه‌ای مناسب برای اپلیکیشن‌هایی است که نیاز به تاخیر پایین دارند، اما خطای در انتقال داده‌ها قابل چشم‌پوشی است.

در برخی از اپلیکیشن‌ها، ترکیب TCP و UDP یا همان TCP Over UDP استفاده می‌شود تا از اعتمادپذیری و مدیریت اتصال TCP در کنار کارایی بالای UDP بهره‌برداری شود. در این پروژه قصد داریم یک اپلیکیشن ساده در همین زمینه پیاده‌سازی کنیم.

هدف پروژه

در این پروژه، دانشجویان باید یک نسخه ساده‌شده از پروتکل TCP را تنها با استفاده از سوکت‌های UDP پیاده‌سازی کنند. هدف این است که دانشجویان با مفاهیم اساسی انتقال قابل اعتماد داده، مانند مدیریت اتصال، قابلیت دریافت داده‌هایی که با ترتیب نادرست دریافت شده اند، ACK و مدیریت قطع ارتباط آشنا شوند.

الزامات فنی

- فقط زبان‌های Python یا C/C++ مجاز هستند. به شدت توصیه می‌شود پروژه را با **پایتون** پیاده‌سازی کنید.
- فقط درون کد از اشیاء، توابع و ... مربوط به UDP استفاده شود. نباید از ماژول‌ها یا کتابخانه‌های آماده TCP استفاده شود.

موارد قابل استفاده مجاز در زبان پایتون به صورت زیر است:

Python UDP Socket

```python
import socket

# Socket Creation
sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)

# Bind Source Addresses
sock.bind((src_ip, src_port))

# Send Data
message = b"TCP Over UDP"
sock.sendto(message, (dest_ip, dest_port))

# Receive Data
data, addr = sock.recvfrom(4096)

sock.close()
```

شرح پروژه

همانطور که می‌دانید، پروتکل TCP از لحاظ اجرا به سه فاز مختلف زیر تقسیم می‌گردد:

- فاز برقراری ارتباط
- فاز انتقال داده
- فاز بستن ارتباط

در این پروژه، دانشجو موظف است سه فاز اصلی TCP را با استفاده از سوکت UDP شبیه‌سازی کند.

پیاده‌سازی پیشنهادی

پیاده‌سازی شئ‌گرا الزامی نیست، اما در این پروژه توصیه می‌گردد.

برای این پروژه، سه کلاس بسته(Packet) ، سوکت (Socket) و اتصال (Connection) توصیه می‌شود.

علت جداسازی سوکت و اتصال این است که در سمت سرور، یک سوکت می‌تواند مسئول گوش دادن (listen) و پذیرش (accept) چندین اتصال همزمان باشد، بنابراین هر اتصال باید به‌صورت مجزا مدیریت شود.

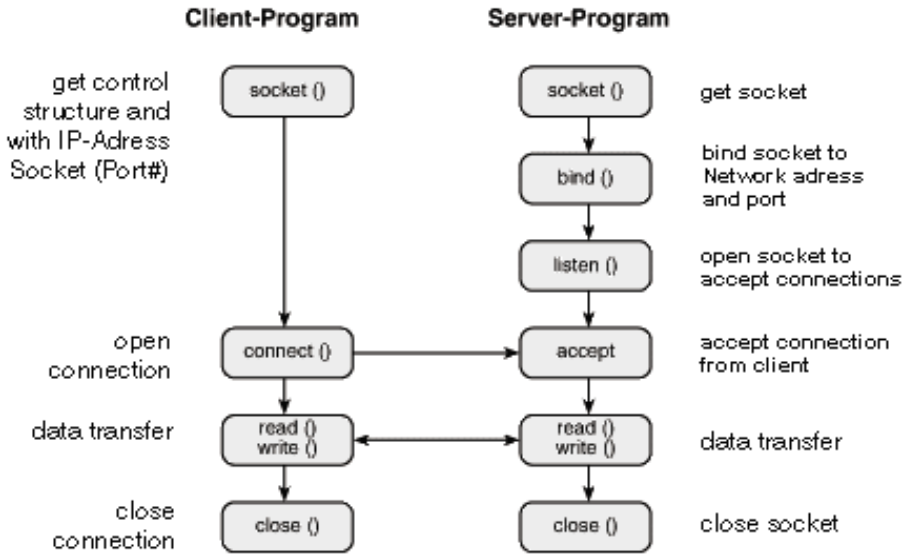
تصویر نمونه پیاده‌سازی شده:

```
PS E:\TCP-Over-UDP> python -u "e:\TCP-Over-UDP\client.py"
[2025-05-26 04:22:44] Socket Created
[2025-05-26 04:22:44] Send SYN to localhost:12345
[2025-05-26 04:22:44] Received SYN-ACK from localhost:12345
[2025-05-26 04:22:44] Send ACK to localhost:12345
[2025-05-26 04:22:44] Connection Established With localhost:12345
[2025-05-26 04:22:44] Receiver thread started on 192.168.12.10:49111
[2025-05-26 04:22:44] Received from ('127.0.0.1', 12345)

[2025-05-26 04:22:44] Sending Acknowledgment to ('localhost', 12345)
[2025-05-26 04:22:44] Received from ('127.0.0.1', 12345)
[2025-05-26 04:22:44] Sending Acknowledgment to ('localhost', 12345)
TCP Over U
[2025-05-26 04:22:44] Sending packet to (('localhost', 12345))
Packet Details:
-----------------
Source Port             : 49111
Destination Port        : 12345
Sequence Number         : 1133012452
Acknowledgment Number   : 3804222960
Control Flags           : ACK
Window Size             : 128
Data Length             : 15        bytes
Data Payload            : Hi from Client!

[2025-05-26 04:22:44] Connection with localhost:12345 is Finished.
PS E:\TCP-Over-UDP>
```

جزئیات پیاده‌سازی

موارد کلی:

۱. فرایند کلی ایجاد و برقراری و پایان اتصال باید مشابه TCP و به صورت زیر دنبال شود.



پیاده‌سازی موارد زیر اجباری است:

- bind

این متد برای تنظیم آدرس و پورت روی سوکت استفاده می‌شود.

در سمت سرور: استفاده از این متد الزامی است و وظیفه آن تعیین آدرس و پورتی است که سرور باید بر روی آن به حالت شنود(listen) درآید و منتظر اتصال‌های ورودی باشد.

در سمت کلاینت: استفاده از این متد اختیاری است. در صورت استفاده، پورت سمت کلاینت به‌صورت دستی مشخص می‌شود. در غیر این صورت، پورت به‌صورت تصادفی باید انتخاب شود.

سوال: در متود bind، چرا نیاز است که علاوه بر شماره پورت، آدرس IP نیز مشخص شود؟ و چه تأثیری بر رفتار سوکت دارد؟

- listen

این متد تنها روی سوکت سمت سرور قابل استفاده است وظیفه آن قرار دادن سوکت در وضعیت شنود برای دریافت اتصالات است.

در سوکت سمت سرور، باید یک صف داخلی برای نگه‌داری اتصالاتی که فرآیند اتصال آن‌ها به‌طور کامل انجام شده، اما هنوز توسط تابع accept() پردازش نشده‌اند، در نظر گرفته شود. ظرفیت این صف از طریق ورودی این متد قابل تنظیم است.

در صورتی که ظرفیت این صف به حد نهایی برسد، اتصالات جدید ورودی پذیرفته نخواهند شد و سرور در پاسخ به بسته SYN دریافتی، هیچ پاسخی ارسال نخواهد کرد.

سوال ۱: آیا اتصالات درون صف accept اگر داده‌ای دریافت کنند، آن‌ها را درون بافر خود می‌توانند ذخیره کنند؟

سوال ۲: در پیاده‌سازی TCP، قبل از برقراری کامل اتصالات، دو نوع صف داریم. درباره این دو نوع توضیح دهید.

- accept

این متد روی سوکت سمت سرور مورد استفاده قرار می‌گیرد و وظیفه آن پردازش اولین اتصال موجود در صف accept است.

در صورتی که صف اتصالات خالی باشد (هیچ اتصالی به‌طور کامل برقرار نشده باشد)، این متد وارد وضعیت blocking خواهد شد؛ به این معنا که اجرای برنامه در این نقطه متوقف می‌شود تا زمانی که یک اتصال جدید در صف قرار گیرد و قابل پردازش باشد.

مشابه برنامه‌نویسی سوکت TCP در زبان پایتون، پس از پایان این تابع باید شئ اتصال و آدرس مبدا اتصال (آی‌پی و پورت مبدا) را برگرداند. در سمت سرور از طریق شئ سوکت بازگردانده‌شده، ارتباط برقرار شده ادامه می‌یابد.

```
conn, addr = my_socket.accept()
```

- connect

این متد مختص سوکت سمت کلاینت است و جهت آغاز فرایند برقراری اتصال با سرور استفاده می‌شود. با فراخوانی این متد، کلاینت تلاش می‌کند به آدرس IP و شماره پورت مشخص‌شده به عنوان پارامترهای آن متصل شود.

در این متد، فرایند سه‌مرحله‌ای برقراری اتصال (3-Way Handshake) آغاز می‌شود که شامل ارسال بسته‌ی SYN، دریافت پاسخ SYN-ACK از سرور، و در نهایت ارسال ACK نهایی توسط کلاینت است. تنها در صورت موفقیت کامل این فرایند، اتصال برقرار شده تلقی می‌شود و کلاینت می‌تواند داده‌ها را ارسال یا دریافت کند.

در صورتی که سرور مقصد دردسترس نباشد یا این اتصال را نپذیرد، کلاینت مدتی منتظر می‌ماند و در نهایت با یک خطای مناسب، اجرای برنامه را متوقف می‌کند.

پس از برقراری اتصال موفق، سوکت سمت کلاینت مانند یک اتصال رفتار خواهد کرد.

- send

این متد تنها پس از برقراری موفقیت‌آمیز اتصال قابل فراخوانی است. این متود هم برای سوکت و هم برای اتصال تعریف می‌شود:

- ✓ برای سوکت، تنها در حالتی که سوکت سمت کلاینت اتصال موفقیت‌آمیزی با سرور برقرار کرده باشد، استفاده می‌شود.
- ✓ برای اتصال، روی اتصالات ایجادشده که توسط accept بازگردانده می‌شوند، قابل فراخوانی است.

با استفاده از آن، داده‌های مورد نظر به بافر ارسال سوکت مربوطه اضافه می‌شوند.

توجه: دقت کنید که پس از فراخوانی این متود، بلافاصله ارسال داده‌ها شروع نمی‌شود و زمان ارسال آن، بستگی به اندازه پنجره، مقادیر درون پنجره و ... دارد. این متود تنها داده‌های مورد نظر را به بافر ارسال اضافه می‌کند. ارسال آن توسط یک ترد که در ادامه توضیح داده می‌شود، انجام خواهد شد.

دقت کنید در صورتی که بافر پر باشد، برنامه به حالت blocking می‌رود و تا زمان باز شدن فضای کافی، اجرا متوقف خواهد ماند.

- receive

این متد پس از برقراری موفقیت‌آمیز اتصال قابل فراخوانی است و وظیفه‌ی آن تحویل داده‌های دریافت‌شده به اپلیکیشن است. متد receive دارای یک پارامتر ورودی است که تعداد بایت‌های مورد انتظار برای خواندن از بافر دریافت را مشخص می‌کند.

این متد هم برای سوکت و هم برای اتصال تعریف می‌شود:

- ✓ برای سوکت سمت کلاینت: تنها زمانی قابل استفاده است که اتصال موفقی با سرور برقرار شده باشد.
- ✓ برای اتصال سمت سرور: تنها روی اتصالاتی که توسط متد accept بازگردانده شده‌اند قابل فراخوانی است.

در صورتی که تعداد بایت‌های درخواست‌شده در بافر دریافت موجود باشد، همان مقدار به اپلیکیشن تحویل داده می‌شود.در غیر این‌صورت، برنامه به حالت blocking می‌رود و تا زمان دریافت داده‌ی کافی، اجرا متوقف خواهد ماند.

پس از تحویل داده به اپلیکیشن، اندازه‌ی پنجره‌ی دریافت(Sliding Window) به میزان داده‌های خوانده‌شده به جلو حرکت می‌کند، تا امکان دریافت داده‌های جدید فراهم شود.

توجه: دقت کنید که حتی اگر این متود فراخوانی نشود، دریافت داده‌ها توسط یک ترد(توضیح در بخش بعد) انجام می‌شود. بنابراین دریافت داده و قراردادن در بافر دریافت توسط خود اتصال انجام می‌شود. این متود داده‌های بافر را به اپلیکیشن تحویل می‌دهد.

- close

این متد هم برای سوکت و هم اتصال مورد استفاده قرار می‌گیرد. این متود برای بستن سوکت و آزادسازی منابع مرتبط با آن استفاده می‌شود و باید پس از اتمام استفاده از سوکت، فراخوانی شود.

رفتار این متد بسته به موقعیت (سوکت سمت کلاینت یا سرور یا روی شئ اتصال) به شرح زیر است:

- ✓ در سمت سوکت کلاینت و همچنین شئ اتصال:

برای اتصال و همچنین سوکت کلاینت، در حالتی که تصال فعالی وجود داشته باشد، متد close باید فرآیند خاتمه‌ی اتصال را آغاز کند؛ بدین صورت که یک بسته‌ی FIN ارسال می‌شود و وارد مرحله‌ی termination در پروتکل می‌شود.

✓ در سمت سوکت سرور:

فراخوانی این متد باعث می‌شود سوکت شنونده (listening socket) بسته شود. در این حالت:

۱. اتصالاتی که توسط accept بازگردانده شده‌اند، همچنان معتبر باقی می‌مانند و می‌توان از آن‌ها استفاده کرد.
۲. اتصالات درون صف accept نیز بایستی با ارسال FIN پایان یابند.
۳. سرور نباید هیچ اتصال جدیدی را بپذیرد و باید از پذیرش درخواست‌های SYN جدید خودداری کند.

در هر دو سمت، پس از فراخوانی این متد، امکان برقراری اتصال جدید وجود نخواهد داشت.

۳. در هر سوکت سمت سرور، باید یک ترد مداوم در حال دریافت بسته‌های ورودی باشد. این ترد وظایف زیر را برعهده دارد:
- دریافت داده‌های مربوط به اتصالات فعال و تحویل به اتصال مربوطه (عملیات demultiplexing)
- بررسی بسته‌های **SYN** که برای برقراری اتصال جدید دریافت می‌شوند و انجام عملیات handshake در صورت اعتبار آن‌ها و در نهایت پس از موفقیت، افزودن آن‌ها به صف accept
- شناسایی بسته‌های نامعتبر یا مربوط به اتصالات ناشناخته، و نادیده گرفتن یا پاسخ مناسب(مثلا بسته RST) به آن‌ها.

۴. هر اتصال، باید یک ترد مربوط به مدیریت و بررسی بافرهای ارسال و دریافت باید وجود داشته باشد. وظایف آن:

بافر ارسال:

✓ در صورتی که داده جدید به بافر ارسال اضافه شود، ارسال آن‌ها بلافاصله شروع شود.
✓ پنجره‌ی ارسال sliding window در سراسر برنامه طول ثابت دارد.
✓ باید پس از ارسال، فرستنده باید منتظر دریافت ACK بماند و در صورت دریافت، پنجره را به جلو ببرد.
✓ قابلیت تفسیر و پذیرش ACK تجمیعی ضروری است.
✓ اگر ACK در زمان مشخصی(یک Timeout ثابت) نرسید، بسته مربوطه باید مجدداً ارسال شود.
✓ پس از دریافت سه Ack تکراری، بلافاصله عملیات بازارسال انجام شود.
✓ هر بسته باید دارای شماره ترتیبی (Sequence Number) باشد تا ترتیب و تأیید درستی آن مشخص شود.
✓ داده‌های بزرگ باید به بسته‌های کوچکتر بر اساس حداکثر اندازه‌ی مجاز تقسیم شوند (فرایند Segmentation). برای این منظور یک متغیر گلوبال در سطح برنامه خود تعریف کنید که مقدار MSS را ذخیره کند.
✓ در هر لحظه فقط به اندازه‌ی پنجره اجازه‌ی ارسال وجود دارد. باید از ارسال بیشتر جلوگیری شود.

بافر دریافت:

- پس از دریافت صحیح بسته حاوی داده، ارسال Ack ضروری است.
- در صورتی که بسته‌ی دریافتی حاوی داده‌ای نباشد و تنها Ack یک پیام قبلی باشد، نیازی به ارسال Ack نیست.
- همچنین Ack تجمیعی نیز باید در صورت نیاز ارسال شود.

- داده‌هایی که به صورت صحیح اما خارج نوبت دریافت شده اند، بایستی به درستی در بافر ذخیره شوند و هچنین Ack آخرین بایت دریافتی که به صورت صحیح دریافت شده است، مجدد ارسال شود.
- اگر بسته‌ای که قبلا به صورت صحیح دریافت شده، مجددا دریافت شود، آن نادیده گرفته شود و Ack مناسبی ارسال شود.

۵. مشابه TCP، شمارنده‌های Sequence Number و Acknowledgement برمبنای تعداد بایت‌های موجود در بسته تنظیم می‌شوند نه شماره بسته‌ها.

۶. بافر سمت فرستنده و گیرنده از نظر ظرفیت، محدودیتی وجود ندارد.

۷. حداقل فیلدهایی که بسته‌های این پروتکل دارند، به صورت زیر است:

- پورت مبدا و مقصد

پورت مبدا همواره تصادفی و و از میان پورت‌های غیر از پورت‌های شناخته‌شده (بالاتر از ۱۰۲۴) انتخاب شود.

- شمارنده Sequence Number

شمارنده در ابتدای اتصال، به صورت تصادفی انتخاب شود.

- شمارنده Acknowledgement
- فلگ‌های SYN, ACK, FIN, RST

زمانی که بسته‌ای نامعتبر دریافت شود، در پاسخ بسته‌ای با فلگ RST فرستاده خواهد شد. منظور از نامعتبر، بسته‌هایی که فلگ نامناسب به کار برده‌اند یا مقدار Sequence Number یا Acknowledgement Number نامعتبری دارند، در زمان قبل اتصال یا بعد از برقراری اتصال فعال است.

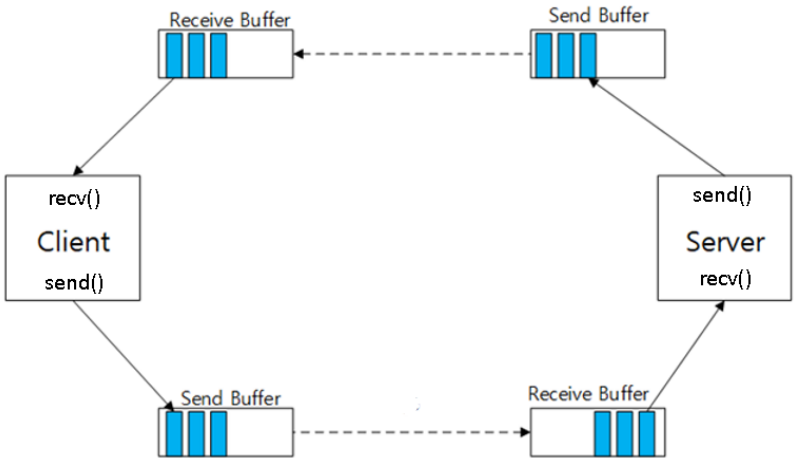
- داده (payload) در صورت وجود
- طول داده (payload)

به صورت خودکار محاسبه می‌شود. در صورتی که پیلود نداشته باشیم، مقدار آن صفر خواهد بود.

۸. به مفهوم و فلسفه وجودی بافرهای ارسال و دریافت دقت کنید.

وقتی در برنامه‌ی شبکه‌ای از متد send استفاده می‌کنید، در واقع داده‌ها را مستقیماً روی شبکه ارسال نمی‌کنید، بلکه آن‌ها را به بافر ارسال می‌فرستید. send تنها واسطه‌ای بین برنامه و بافر ارسال است و کنترل واقعی ارسال داده روی شبکه بر عهده‌ی سوکت است.

متود recv وظیفه دارد داده‌هایی که از شبکه وارد شده‌اند و در بافر دریافت ذخیره شده‌اند را از این بافر بخواند و به برنامه تحویل دهد. وقتی یک بسته از طریق شبکه توسط سوکت دریافت می‌شود، ابتدا وارد این بافر می‌شود و تا زمان فراخوانی recv در بافر باقی می‌ماند. بنابراین، متد recv صرفاً واسطی برای دسترسی به داده‌هایی است که قبلاً توسط سوکت دریافت شده و در بافر قرار گرفته‌اند.



بنابراین تفاوت اصلی بین متدهای send و recv با بافرهای ارسال و دریافت در این است که این متودها تنها ابزارهایی هستند برای تعامل با این بافرها، نه اجزای واقعی شبکه. send تلاش می‌کند داده را وارد بافر ارسال کند، و recv تلاش می‌کند داده را از بافر دریافت بخواند. در واقع، هیچ‌کدام از این متدها به‌تنهایی قادر به مدیریت کامل روند ارسال یا دریافت نیستند، بلکه آنچه اتفاق می‌افتد، تحت کنترل سوکت قرار دارد.

فاز اتصال:

در این فاز اتصال بین کلاینت و سرور برقرار می‌شود.

۱. باید در این مرحله فرایند 3-Way-Handshake شبیه‌سازی شود.
۲. بایستی در سمت سرور، یک صف accept داشته باشیم که محدودیت اندازه این صف توسط متود listen مشخص میگردد.
۳. در صورتی که پس از درخواست اتصال از سمت کلاینت، پاسخی دریافت نشود، در بازه‌های زمانی در چند نوبت درخواست اتصال جدید مجددا ارسال شود.
۴. شمارنده‌های Sequence Number در این بخش ابتدا به صورت رندوم تولید شوند.

فاز انتقال داده:

درباره این فاز در بخش متودهای send receive به صورت کامل توضیحات داده شد. در این فاز در هر دو سمت فرستنده و گیرنده بایستی یک بافر و پنجره لغزان تعریف شود.

فاز پایان اتصال:

در این فاز اتصال بین کلاینت و سرور بسته خواهد شد. باید فرایند پایان اتصال دقیقا مشابه TCP (ارسال FIN) پیاده‌سازی شود.

موارد امتیازی

نکته مهم: دقت کنید که موارد زیر امتیازات برابر ندارند و برحسب سختی، پیچیدگی و ... امتیازات متفاوتی به آن‌ها تعلق خواهد گرفت.

۱. ثبت لاگ از تمامی عملیات‌ها به صورت دقیق و با جزئیات
۲. سرور به صورت همزمان بتواند چند اتصال را بپذیرد و با آن‌ها ارتباط برقرار کند.
۳. پایان ارتباط با فلگ RST (مشابه روشی که اتصالات TCP در ویندوز بسته می‌شوند)
۴. یک برنامه ساده کلاینت-سروری و استفاده از این پروژه به عنوان ابزار ارتباطی
۵. داده‌های ارسالی به صورت خام و مشخص ارسال نشوند:

به اینصورت که اگر این داده‌ها برروی شبکه به وسیله Wireshark یا ... شنود شوند، به صورت اولیه نباشند. برای مثال اگر فرستنده کلمه HelloWorld را ارسال کند، پس از شنود بسته‌ی آن، دقیقا همین عبارت دیده نشود. دقت کنید که در این بخش، بهترین پیاده-سازی نمره بیشتری خواهد داشت.

۶. وجود تایمر برای حالات زیر:
   - ✓ فرآیند handshake اگر تا زمان مشخصی اتصال کامل نشد، عملیات لغو و اتصال بسته شود.
   - ✓ برای اتصال برقرارشده اگر تا مدت مشخصی هیچ بسته‌ای ارسال یا دریافت نشود، اتصال را به صورت خودکار ببندد(بسته FIN به طرف مقابل ارسال شود)

۷. اضافه‌کردن حالت Non-Blocking یا timeout (حالتی که پس از یک بازه زمانی مشخص اگر از حالت Blocking خارج نشود، به صورت خودکار خارج شود) به متودهای accept، send و recv

۸. پنجره ارسال متغیر

امکان تغییر پنجره ارسال به صورت داینامیک درون سوکت وجود داشته باشد.

۹. کنترل جریان (وابسته به مورد ۸)

برای بافر گیرنده یک محدودیت اندازه در نظر گرفته شود. در هر بسته یک فیلد Window اضافه شود و در بسته‌های Ack، مقدار فضای باقی‌مانده در بافر گیرنده به فرستنده ارسال شود. فرستنده پس از دریافت، مقدار آن را درون یک متغیر rwnd ذخیره می‌کند. همچنین باید Sliding Window ارسال را براساس آن تنظیم کند.

در صورتی که اپلیکیشن، داده‌های دریافتی را استفاده نکند، پس از مدتی بافر گیرنده پر می‌شود. در اینصورت گیرنده باید درون فیلد Window، مقدار صفر را ارسال کند. در اینصورت اندازه پنجره فرستنده صفر خواهد شد(ارسال به صورت موقت متوقف خواهد شد)

سوال: فرض کنید که در پروتکل TCP، ارسال داده بخاطر کنترل جریان(تکمیل ظرفیت بافر گیرنده) متوقف می‌شود. فرستنده از کجا باید متوجه شود که ظرفیت بافر گیرنده آیا خالی شده است و می‌تواند بسته‌ی جدیدی را بپذیرد؟

۱۰. پیاده‌سازی مکانیزم واقعی مربوط به پاسخ سوال بالا

۱۱. کنترل ازدحام (وابسته به مورد ۸)

در این بخش، یک متغیر cwnd تعریف می‌شود و به صورت زیر مقدار آن تغییر می‌کند.

- اگر Timeout پس از ارسال داده رخ دهد، cwnd برابر اندازه‌ی یک MSS تنظیم شود.
- در صورتی که سه عدد ACK تکراری (Duplicate ACK) دریافت شود، cwnd نصف می‌شود.
- همچنین پس از دریافت هر ACK جدید و معتبر، cwnd به اندازه‌ی یک MSS افزایش می‌یابد.

در نهایت مقدار Sliding Window براساس آن تنظیم می‌شود.

سوال ۱: چه نوع اطلاعاتی درباره‌ی وضعیت شبکه را می‌توان از رخداد Timeout در مقابل Duplicate ACK استخراج کرد؟

سوال ۲: چرا پس از Timeout، cwnd نسبت به حالت سه عدد ACK تکراری طول کمتری به خود می‌گیرد؟

نکته مهم: در صورتی که هردو مورد امتیازی کنترل جریان و کنترل ازدحام را پیاده‌سازی می‌کنید، باید در نهایت Sliding Window برابر حداقل مقدار دو متغیر cwnd و rwnd تعیین گردد.

Sliding Window = min(cwnd, rwnd)

۱۲. مدت زمان Timeout پس از ارسال بسته و عدم دریافت Ack بسته مجدد ارسال می‌شود، متغیر باشد.

سوال: در پروتکل TCP این مدت زمان چگونه محاسبه می‌شود؟

سوال: در پروتکل TCP، در صورتی که فیلدهای ۳۲ بیتی Sequence Number یا Acknowledgement Number سرریز کنند، وضعیت اتصال و انتقال داده‌ها به چه صورت خواهد شد؟

۱۳. پیاده سازی مکانیزم TCP پس از سرریز فیلدهای بالا

موفق باشید.