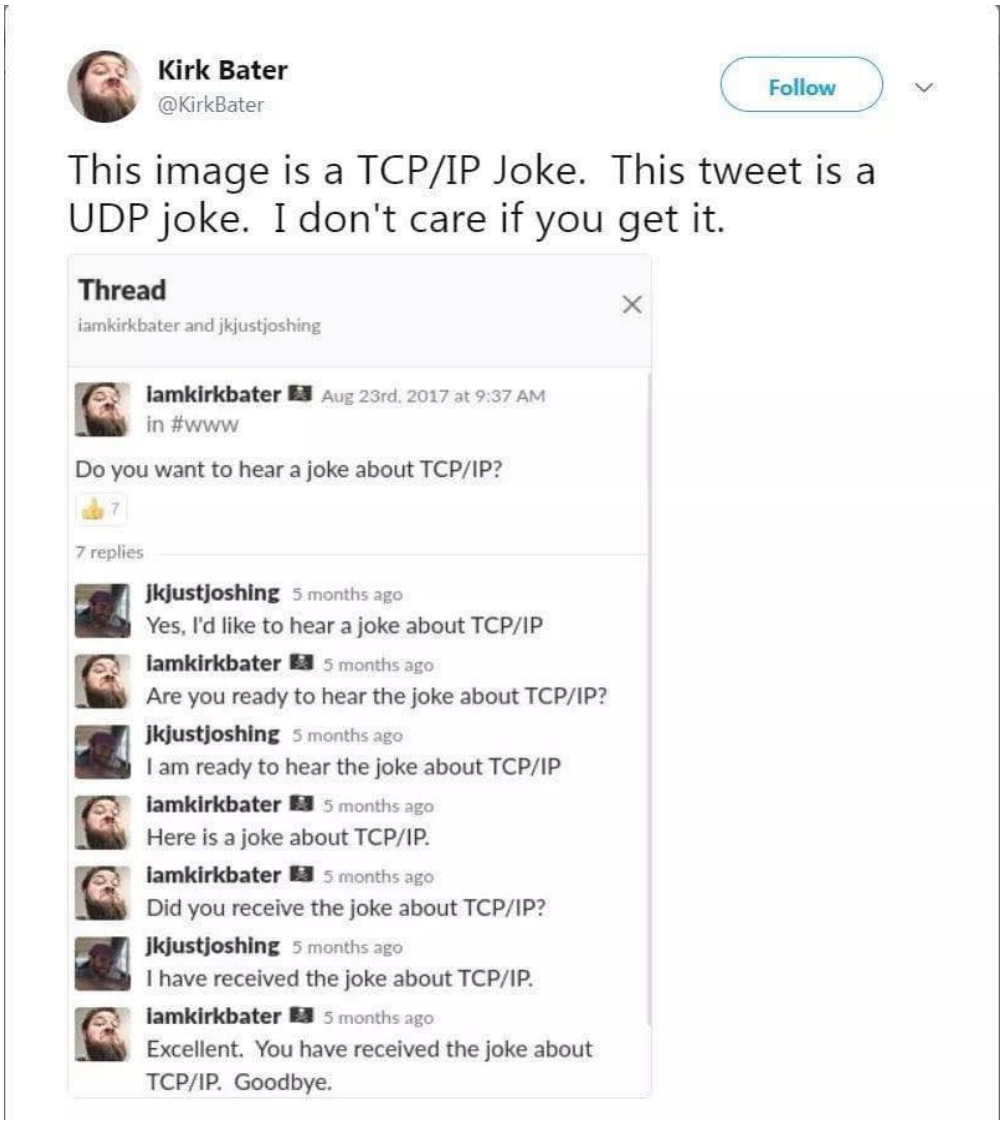
شبکه‌های کامپیوتری

پروژه‌ی سوم

پروتکل **TCP**

مهلت تحویل: ۱۰ اردیبهشت



هدف پروژه

در این پروژه به قصد آشنایی بیشتر با پروتکل TCP New Renoمی‌خواهیم قسمتهایی از آن را پیاده‌سازی کنیم. برای ارسال بسته‌ها از پروتکل UDP استفاده میکنیم.

ارسال داده‌ها به شکل خام

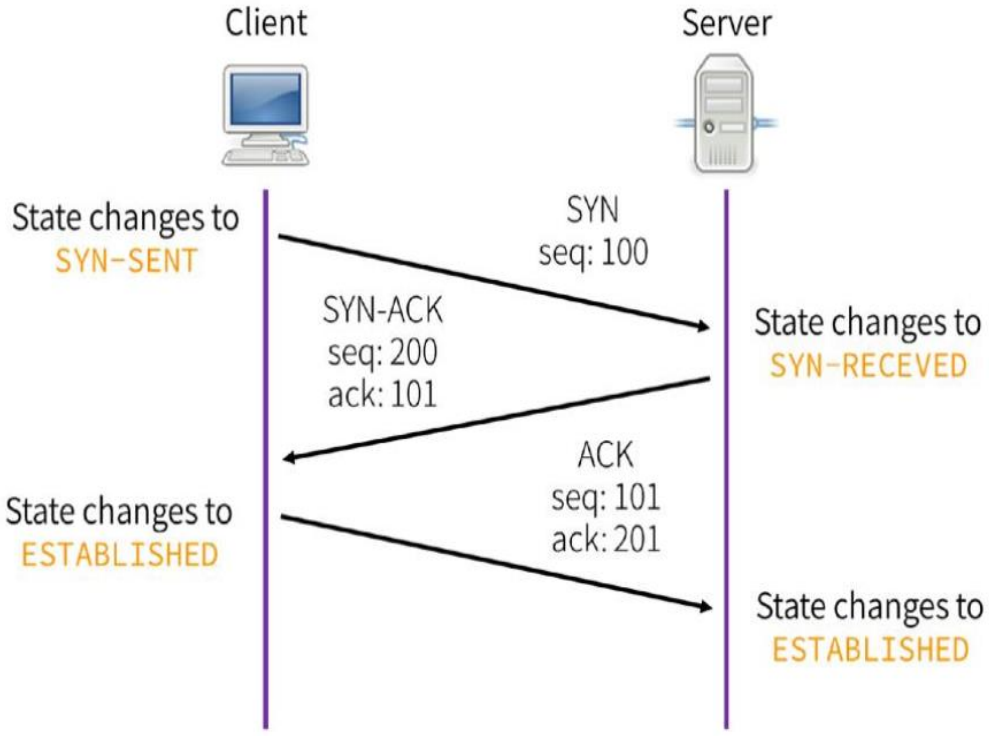
برای ارسال داده‌ها به شکل خام از پیاده‌سازی پروتکل UDP در زبان جاوا استفاده می‌کنیم. البته همانطور که میدانید پروتکل UDP خود دارای checksum است و نیازی به بررسی درستی داده‌های یک بسته توسط شما نیست. در واقع در این پروتکل هیچ تضمینی برای ارسال یک بسته وجود ندارد ولی در صورت دریافت در سمت گیرنده، داده‌ها سالم هستند.

پروتکل TCP

در این پروژه ویژگیهای زیر از پروتکل TCP مدنظر هستند:

۱) شروع اتصال به کمک **Three-way handshake**

برای شروع اتصال باید فرآیند SYN->SYN-ACK->ACK توسط کلاینت و سرور طی شود:



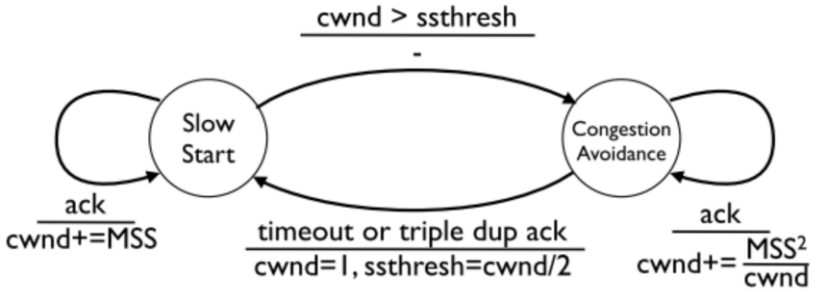
۲) ارسال مطمئن و پایپ‌لاین داده‌ها توسط پروتکل **go-back-n**:

در صورت عدم دریافت بسته‌ها در سمت گیرنده، باید بر اساس قواعد موجود در پروتکل go-back-n بسته‌ها دوباره فرستاده شوند. پیشنهاد می‌شود قبل از پیاده‌سازی این لینک را مشاهده کنید.

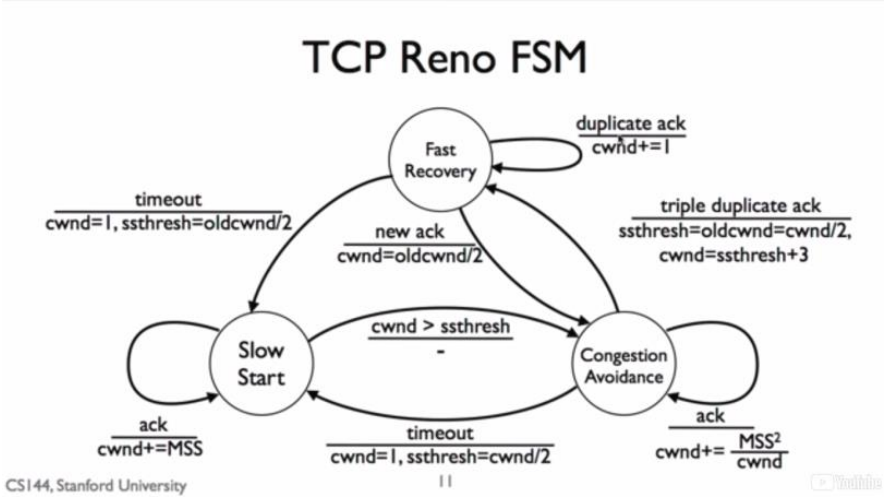
۳) کنترل ازدحام

به وسیله‌ی طول پنجره‌ی مورد استفاده برای پایپلاین داده میتوان نرخ ارسال و ازدحام شبکه را در یک اتصال کنترل کرد. در این قسمت، کنترل ازدحام TCP New Reno را پیاده‌سازی می‌کنیم. این روش کنترل ازدحام در واقع الحاقیه‌ای بر روش TCP Reno است که Fast Recovery را تغییر می‌دهد.

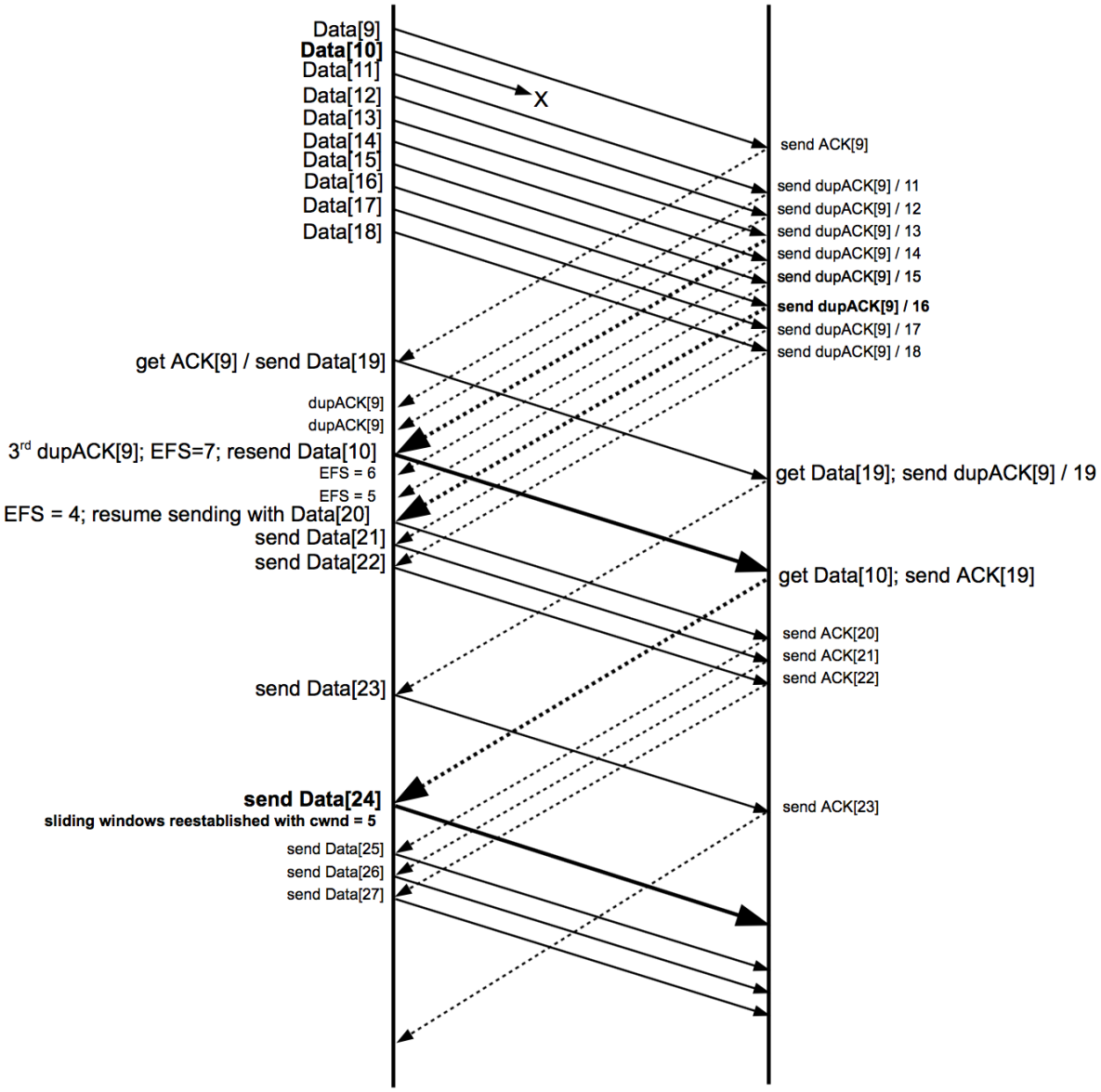
در TCP Tahoe در صورتی که بسته‌ی شماره n گم شود، گیرنده بسته‌ی dupAck برای بسته‌ی n-۱ را ارسال می‌کند. فرستنده پس از دریافت سه dupAck، فرض می‌کند که بسته‌ی n گم شده و آن را دوباره ارسال می‌کند و به Slow Start می‌رود. نمودار TCP Tahoe در شکل زیر قرار دارد:



TCP Reno در صورت دریافت سه dupAck به حالت جدید به نام Fast Recovery می‌رود و در صورت دریافت ack جدید به حالت congestion avoidance بر می‌گردد.



در شکل زیر چگونگی استفاده از dupAck برای ارسال بسته‌ی گم شده در TCP Reno نشان داده شده است:



درصورتی که در بیش از یک بسته در هر پنجره گم شود، TCP Reno قادر به تشخیص بسته‌های گم شده نیست.

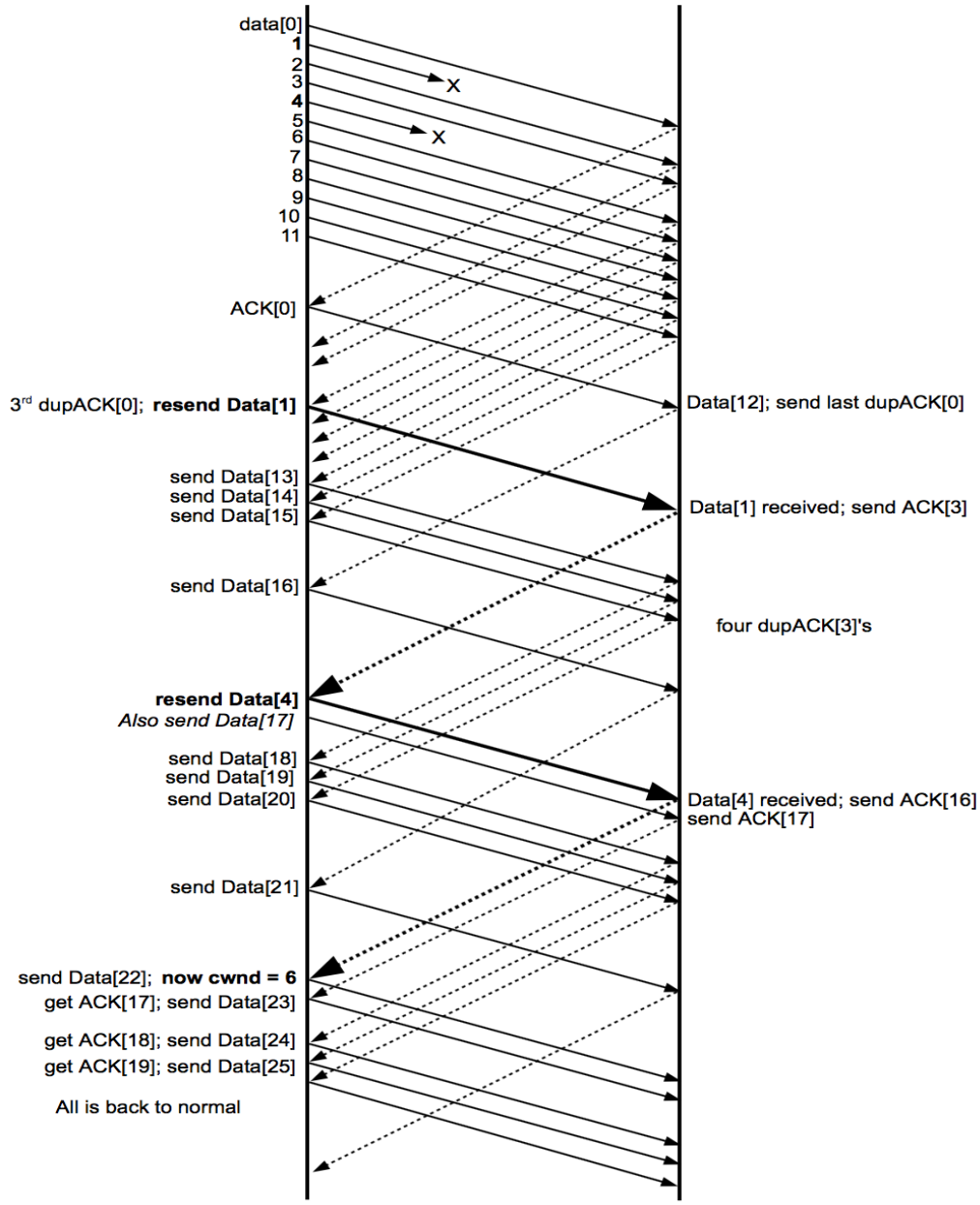
TCP NewReno این مشکل را حل می‌کند. در حالت قبلی در صورتی که بسته‌ی گم شده پس از ارسال دوباره به گیرنده برسد، پیغام ack جدید از سوی گیرنده ارسال می‌شود و این پیغام در واقع به معنای رسیدن تمام بسته‌های آن پنجره است. زیرا قبل از این ack جدید تمام بسته‌های dupAck نشانگر رسیدن دیگر بسته‌های پنجره بودند.

اما در صورتی که بیش از یک بسته گم شود، رسیدن یک ack جدید تنها نشانگر رسیدن قسمتی از بسته‌های پنجره است و هنوز بسته(ها)ی نیاز به ارسال دوباره دارد. به همین دلیل ack های جدید در TCP NewReno به Partial Ack معروف هستند.

راه‌حل ارائه شده این است که در صورت دریافت یک Partial Ack با شماره‌ی n-۱، فرستنده فرض می‌کند که همه‌ی بسته‌ها تا n-۱ رسیده‌اند و بسته‌ی n گم شده است. (در صورتی که در حالت قبلی، فرض بر رسیدن تمام بسته‌های پنجره بود)

شکل زیر نشانگر یک حالت فرضی برای گم شدن یک بسته در TCP NewReno است. بسته‌ی شماره ۱ و ۴ گم شده‌اند. فرستنده بعد از دریافت ۳ dupAck با شماره ۰، بسته‌ی ۱ را ارسال می‌کند. پس از رسیدن بسته‌ی ۱ به گیرنده، ack شماره ۳ ارسال می‌شود. زیرا تا آن

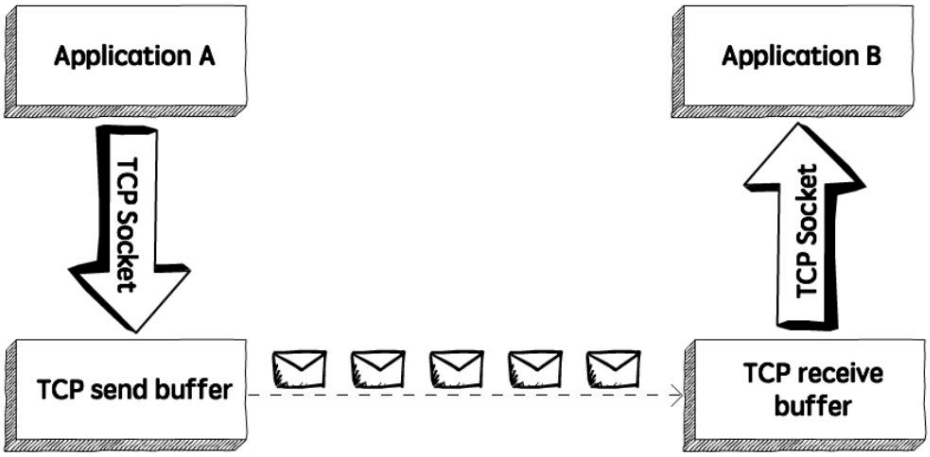
لحظه همه‌ی بسته‌ها به جز بسته‌ی ۴ به مقصد رسیده‌اند. فرستنده بسته‌ی ۴ را دوباره ارسال می‌کند. بعد از دریافت بسته‌ی ۴، فرستنده ack ۱۶ را می‌فرستد که به معنای رسیدن همه‌ی بسته‌ها تا شماره‌ی ۱۷ است. دقت کنید که در طول این فرآیند فرستنده با دریافت هر dupAck متوجه می‌شود یک بسته‌ی جدید به گیرنده رسیده است و بسته‌ی بعدی را ارسال می‌کند. به همین دلیل است که در هنگام ارسال دوباره‌ی ۴، همزمان بسته‌ی ۱۷ با آن ارسال می‌شود.



برای اطلاع از جزئیات دقیق NewReno به صفحه‌ی RFC مربوط به آن مراجعه کنید. هم‌چنین جزئیات بیشتری از مثال ارائه شده در قسمت ۱۳,۴ و ۱۳,۵ این لینک قابل دسترسی است.

شما در این پروژه باید TCP NewReno را که عملا همان Reno با تغییرات گفته شده در Fast Recovery است پیاده‌سازی کنید.

Flow control :



گاهی ممکن است سرعت دریافت و پردازش بسته‌ها توسط گیرنده کمتر از سرعت ارسال بسته‌ها توسط فرستنده باشد و در نتیجه بافر ورودی پر شود. برای جلوگیری از این امر، از flow control استفاده می‌شود. ایده‌ی اصلی برای پیاده‌سازی flow control این است که گیرنده به فرستنده اطلاع دهد که در بافر خود چقدر جای خالی دارد و فرستنده متناسب با آن، دیتا بفرستد.

برای اطلاع از جزئیات دقیق تر به این لینک مراجعه شود.

الگوریتم Nagle (بخش امتیازی):

همان‌طور که می‌دانیم، اندازه‌ی Header بسته‌های TCP برابر ۴۰ بایت است. بسته‌های زیادی در شبکه رد و بدل می‌شوند که مقدار payload آن‌ها بسیار کمتر از ۴۰ بایت (مثلا ۱ بایت) است. اگر این داده‌ی قرار گرفته در این بسته‌ها به صورت جدا جدا ارسال شود، حجم زیادی از پهنای باند صرف ارسال Header می‌شود.

یک راه حل خوب این است که داده‌ی چندین بسته با payload کوچک را در یک بافر ذخیره کنیم و همه‌ی آن را در یک بسته ارسال کنیم که نسبت payload به Header قابل توجه باشد.

الگوریتم Nagle تا زمانی که بسته‌ی ack نشده وجود دارد سعی می‌کند داده‌ها را در بافر ذخیره کند و وقتی از میزان MSS بزرگتر شد، آن را ارسال کند. شبه‌کد این الگوریتم به این صورت است:

```
if there is new data to send
  if the window size >= MSS and available data is >= MSS
    send complete MSS segment now
  else
    if there is unconfirmed data still in the pipe
      enqueue data in the buffer until an acknowledge is received
    else
      send data immediately
```

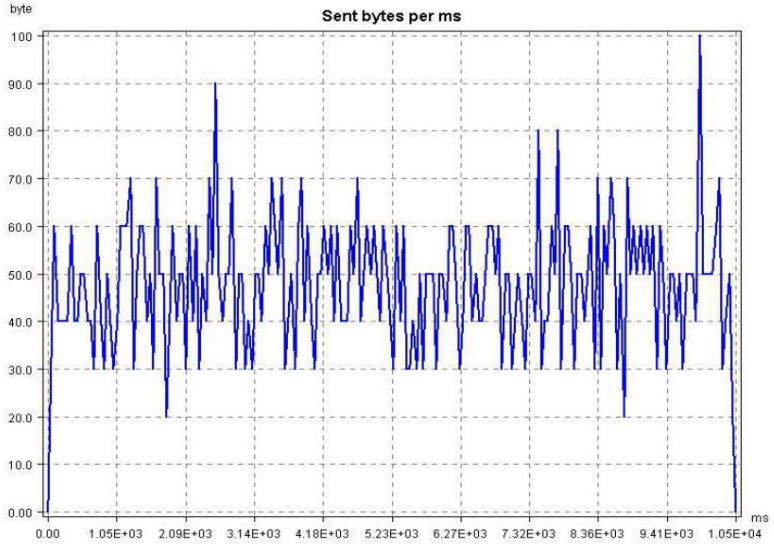
شما می‌توانید این الگوریتم را برای ارسال داده در فرستنده پیاده‌سازی کنید و از نمره‌ی امتیازی بهره‌مند شوید.

استفاده از کلاس :EnhancedDatagramSocket

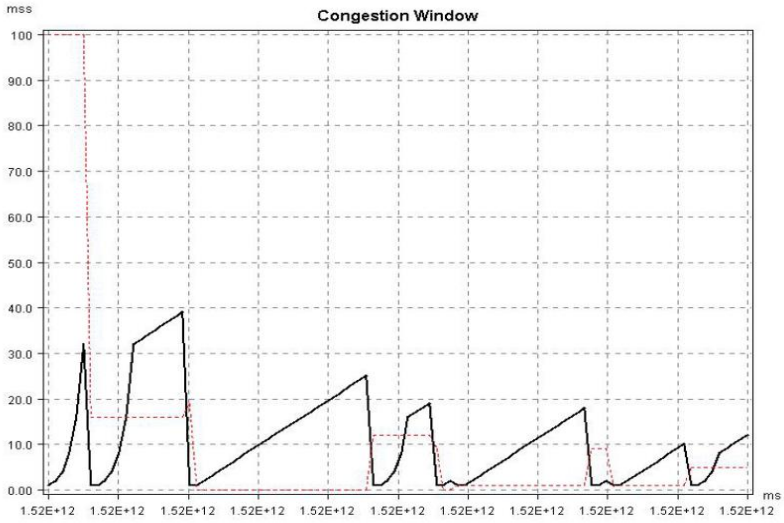
برای شبیه‌سازی برخی از ویژگی‌های لینک، یک کلاس با نام DatagramSocket به شما داده شده است که در واقع از کلاس EnhancedDatagramSocket ارث‌بری کرده است؛ در نتیجه از تمام متدهای این کلاس از جمله send، close، receive و ... پشتیبانی می‌کند. دقت کنید که نیازی به پیاده‌سازی فرآیند Fin Ack ندارید و استفاده از Close کافی است.

در این پروژه شما باید از این کلاس برای ارسال پیام‌ها استفاده کنید و با تغییر پارامترهای آن می‌توانید برخی از ویژگیهای لینک را نیز شبیه‌سازی کنید.

در کدهای قرار داده شده دو نمودار به طور خودکار همراه برنامه‌ی شما رسم میشوند. اولین نمودار تعداد بایت‌های ارسال شده توسط برنامه‌ی شما در واحد زمان است:



دومین نمودار طول پنجره (رنگ مشکی) و آستانه‌ی حالت (Slow Start رنگ قرمز) بر حسب زمان را مشخص میکند. برای رسم این نمودار کافی است متدهای getWindowSize و getSSThreshold را پیاده‌سازی کنید و در هنگام تغییر دو متغیر مربوطه متد onWindowChange را فراخوانی کنید.



کلاس‌های نهایی

شما در نهایت باید کلاس‌های TCPServerSocket و TCPSocket را توسط ارث‌بری توسعه دهید. برای راحتی کار دو نمونه از این کلاس‌ها با نام‌های TCPSocketImpl و TCPServerSocketImpl در اختیار شما قرار داده می‌شود. هم‌چنین نحوه‌ی استفاده از آن‌ها در توابع Sender و Receiver آمده است.

دقت شود که انتقال packet ها از فرستنده به گیرنده باید **قابل اطمینان** باشد. یعنی تمام سناریوهای ممکنِ از بین رفتن یک packet مانند network loss، پر شدن بافر گیرنده و ... را باید درنظر بگیرید و برای هرکدام راهکار مطلوب را ارائه دهید.

نکات تکمیلی:

۱- برای ارسال داده‌ها تحت شبکه تنها امکان استفاده از کلاس EnhancedDatagramSocket وجود دارد و استفاده از کلاسهای دیگر مانند ServerSocket، Socket،DatagramSocket و ... تقلب محسوب میشود.

۲- توجه نمایید که payload بسته‌های UDP که توسط EnhancedDatagramSocket فرستاده می‌شوند، نباید از ۱۴۸۰ بایت بیشتر شود.

۳- تمام مقدارهای اولیه‌ی متغیرها همانند SSThreshold،CongestionWindowSize و ... توسط خود شما انتخاب میشوند و مناسب است به صورت تجربی بهترین آنها را برای بالا بردن کارایی انتخاب نمایید.

۴- این پروژه به صورت گروهی است ولی لزوما نمره‌ی دو نفر یکسان نیست.

۵- سوالات خود را در مورد پروژه از طریق فروم CECM مطرح کنید.

۶- کد پروژه در لینک زیر موجود است.

https://github.com/nikiibayat/TCP-over-UDP