Chapter 3
Transport Layer

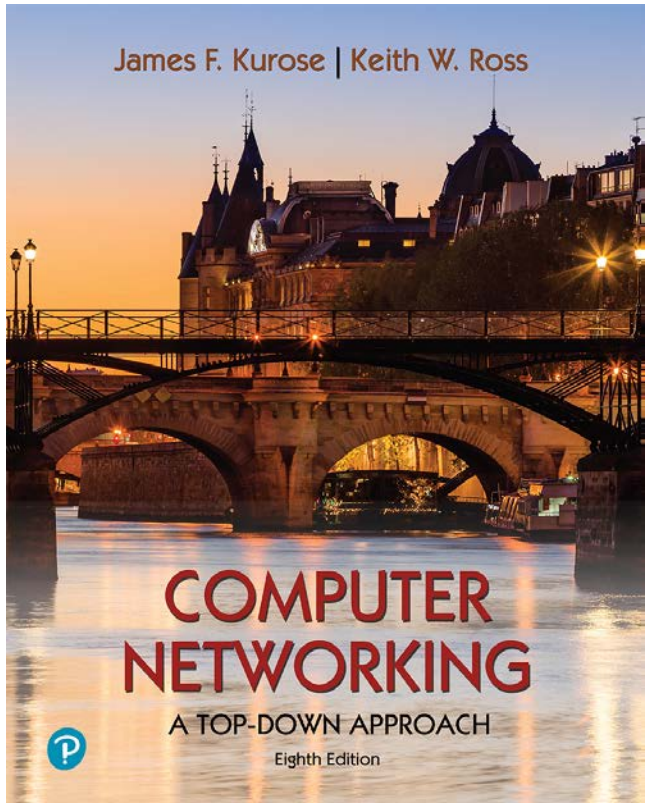
A note on the use of these PowerPoint slides:

We're making these slides freely available to all (faculty, students, readers). They're in PowerPoint form so you see the animations; and can add, modify, and delete slides (including this one) and slide content to suit your needs. They obviously represent a *lot* of work on our part. In return for use, we only ask the following:

- If you use these slides (e.g., in a class) that you mention their source (after all, we'd like people to use our book!)
- If you post any slides on a www site, that you note that they are adapted from (or perhaps identical to) our slides, and note our copyright of this material.

For a revision history, see the slide note for this page.

Thanks and enjoy! JFK/KWR



Computer Networking: A Top-Down Approach

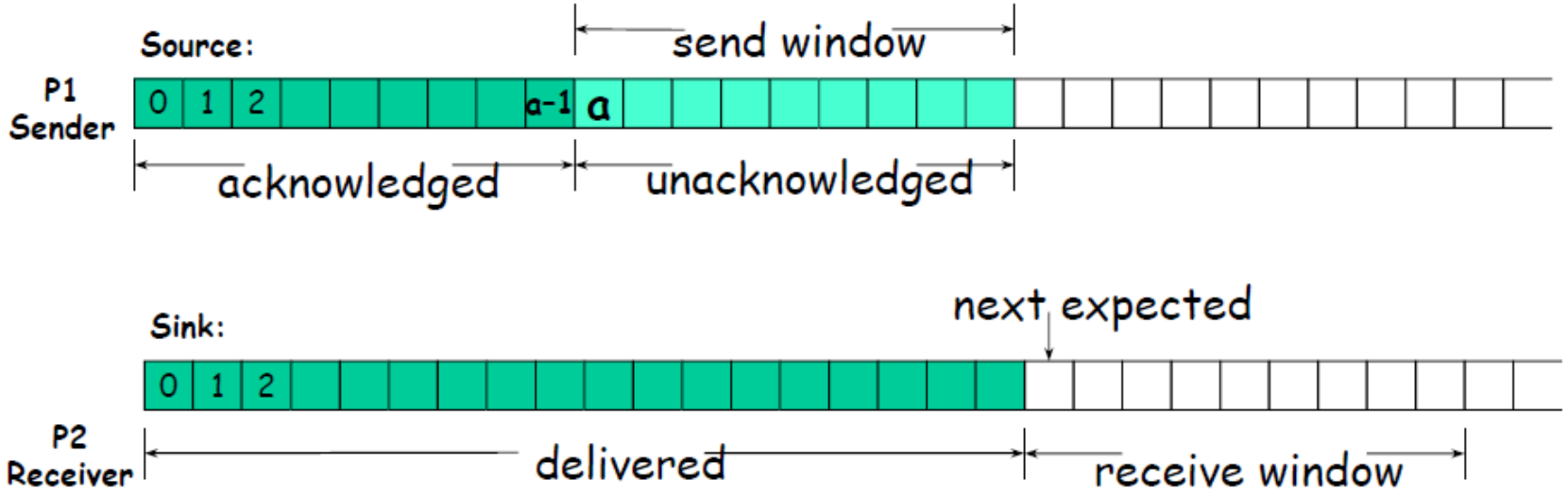
8th edition
Jim Kurose, Keith Ross
Pearson, 2020

Chapter 3: roadmap

- Transport-layer services
- Multiplexing and demultiplexing
- Connectionless transport: UDP
- Principles of reliable data transfer
- Connection-oriented transport: TCP
- Principles of congestion control
- TCP congestion control
- Evolution of transport-layer functionality

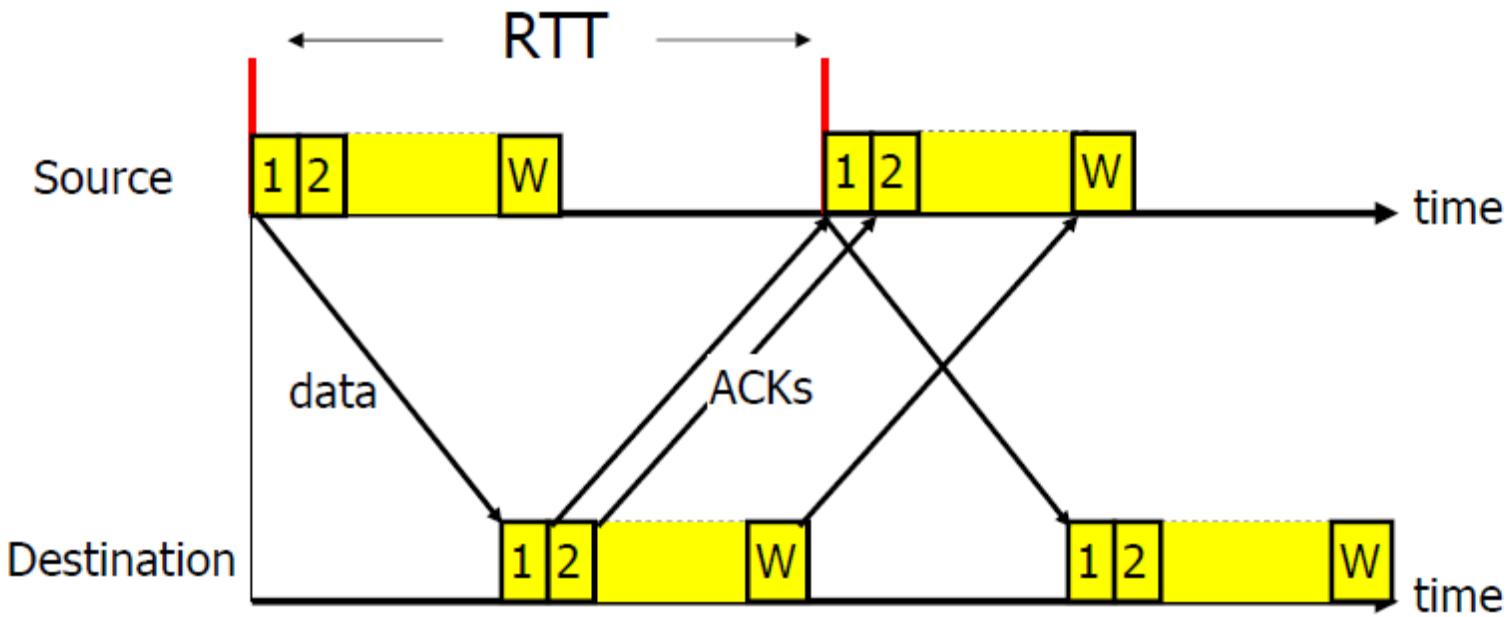
Sliding Window protocol

- Functions provided
  - reliable delivery (error and loss control)
  - in-order delivery
  - flow and congestion control
    - by varying send window size



-
در پروتکل tcp گفتیم که از Sliding Window استفاده میکنیم در سمت فرستنده Sliding Window بسته هایی که ارسال میشن و هنوز اک نشدن رو در خودش جا میده و در سمت گیرنده هم Sliding Window بسته هایی که دریافت شدن رو در خودش جا میده که بعد از مرتب شدن این بسته ها به اپلیکیشن داده میشن
Sliding Window کار اصلی لایه tcp ینی reliable delivery رو مدیریت میکنه که برای انتقال بدون خطا و بدون از دست رفتن اطلاعات از فرستنده به گیرنده است و همینطور حفظ ترتیب اون ها
و دوتا کار دیگه هم Sliding Window انجام میده:
flow and congestion control که برای این دوتا کار ما از تغییر اندازه پنجره Sliding Window استفاده میکنیم برای مدیریت حجم بافری که در سمت گیرنده داره tcp که به اون flow control گفتیم و همینطور مدیریت ازدحام در داخل شبکه که این هم معادل بافری هستش که میتونیم تصور کنیم که داخل شبکه هست

Window Size Controls Sending Rate



- ~ W packets per RTT when no loss

اساس این کار مبتنی بر این است که اندازه پنجره ارسال حجم اطلاعاتی که مبدا به شبکه و به مقصد می فرسته و همینطور Rate اطلاعات رو تعیین می کنه
چرا؟ برای اینکه وقتی که ما بسته های رو شروع میکنیم و ارسال میکنیم از مبدا به مقصد و گفتیم یک زمانی طول می کشه و وقتی بسته به مقصد رسید اک اون یه زمانی طول میکشه که برگرده به این زمان گفتیم RTT ما در یک RTT بسته هارو پشت سرهم می فرستیم برای اینکه عرض باند شبکه رو از دست ندیم و متنظر اک اون ها می مونیم تا برسیم به اندازه پنجره که اینجا دیگه ارسال متوقف می شه که اک ها برگردن و وقتی که اک ها برگشتن در حالت عادی این ها پشت سر هم برمیگردن و ما پشت سر هم می تونیم بسته های بعدی رو بفرستیم ولی اون چیزی که مهم است اینه که در یک RTT ما به اندازه یک سایز پنجره اطلاعات رو می فرستیم
با فرض اینکه پکت لاس اتفاق نیوفته می خوایم ببینیم منجر به چه RATE میشه.. صفحه بعدی..

Throughput

Max. throughput = W / RTT bytes/sec

- This is an upper bound
- Actual throughput is smaller
  - Average number in the send buffer is less than W
  - Retransmissions
- The throughput of a host's TCP send buffer is the host's send rate into the network (including original transmissions and retransmissions)

-
در یک RTT به اندازه W ینی به اندازه پنجره ارسال بایت فرستادیم پس W/RTT میشه تعداد بایتی که در واحد زمان ارسال میشه و این Throughput مارو تعیین میکنه
اگر ما در یک RTT به اندازه پنجره ارسالمون بایت بفرستیم ینی W/RTT bytes/sec که این ماکزیمم Throughput است ینی یک باند بالایی است و Throughput واقعیمون می تونه کمتر از این باشه به دلیل اینکه ما همیشه به اندازه واقعی پنجره ارسال نمیکنیم و شاید بتونیم بگیم متوسط ارسال ما به اندازه پنجره هست چون بعضی وقت ها ممکنه اصلا اپلیکیشنمون اطلاعات جدیدی نداره که بفرسته در نتیجه پنجرمون جا داره ولی ما چیزی نمی فرستیم
این ارسال هایی که انجام می گیره Retransmissions ها رو هم شامل میشه که این Retransmissions بخشیش بخاطر بسته هایی هستن که واقعا گم شدن و البته این ها جز Throughput همشون حساب میشن و بخشی اون هایی هستن که گم نشدن ولی به علت دیر رسیدن duplicates رخ داده و دوباره اینارو فرستادیم به هر حال مجموعه تمام این بسته هایی که به این ترتیب ارسال میشن به شبکه rate ارسال مبدا به شبکه و Throughput اون رو تعیین میکنه

TCP Send Window Size

- TCP flow control
  - Avoid overloading receiver
  - Receiver calculates flow control window size (**rwnd**) based on the available receiver buffer space
  - Receiver sends flow control window size to sender in TCP segment heather
  - Sender keeps Send Window size less than most recently received **rwnd** value
- TCP Congestion Control
  - Avoid overloading network
  - Sender estimates network congestion from "loss indications"
  - Sender calculates congestion window size (**cwnd**)
  - Sender keeps Send Window size less than a maximum **cwnd** value
- **Sender sets W = min (cwnd, rwnd)**

-
به این ترتیب اندازه پنجره ارسال tcp نقش تعیین کننده ای در پروتکل tcp داره:
یکی از کاربردهای اون flow control است و هدف از flow control این است که بافر سمت گیرنده پروتکل tcp رو اجازه ندیم overload بشه و overflow اتفاق بیوفته واسه همین سمت گیرنده اندازه بافر tcp خودش رو مانیتور می کنه و قسمت خالی اون رو اندازه گیری میکنه مرتبا و این رو به عنوان یک اندازه پنجره ای که فرستنده می تونه متناسب با اون اطلاعات بفرسته و اگر اینقدر بفرسته جا برای پذیرش اون ها هست به سمت فرستنده اعلام میکنه اسم این اندازه رو می ذاریم rwnd و این اندازه ای است که سمت گیرنده به سمت فرستنده اطلاع رسانی می کنه توی فیلد هدر window size بسته های tcp که از گیرنده داره به فرستنده می فرسته اینو اعلام میکنه و سمت فرستنده هم اندازه پنجره ارسال خودش رو متناسب با این انداز ست می کنه ینی روی اندازه ای کمتر از rwnd ست میکنه که حجم اطلاع رسانی که از حجم بافر ازاد سمت گیرنده تجاوز نکنه
به همین ترتیب هدف TCP Congestion Control این است که ما خود شبکه رو overload نکنیم و بافرهای توی روترهای شبکه رو overload نکنیم --> در اینجا به نوعی باید وضعیت این بافر رو چک بکنیم و ببینیم چقدر جا داره و براساس اون اندازه پنجره ارسال رو تنظیم بکنیم منتها ما اینجا فرضمون بر این است که روترها توی لایه شبکه هستن و ما اطلاع مستقیمی از وضعیت بافرها نداریم برای همین این رو از اثار و اعلامش باید تشخیص بدیم هر نوع گم شدن بسته ها در شبکه می تونه نشون دهنده این باشه که بافرها پر شدن و ین وضعیت پیش اومده و tcp از این استفاده میکنه ینی سمت فرستنده پروتکل tcp وضعیت گم شدن بسته ها رو در شبکه مانیتور میکنه واگر نشونه هایی مبنی بر گم شدن بسته ها ببینه فرض میکنه که Congestion داره اتفاق می افته که البته در شبکه های جدید که لینک ها فیبرنوری هستن و نودها از تجهیزات پیشرفته استفاده می کنن پس احتمال خطا خیلی پایین است پس اگر گم شدن بسته اتفاق بیوفته بخاطر همین Congestion است --> براین اساس سمت فرستنده خودش تشخیص میده که Congestion اتفاق افتاده و خودش براساس حدسی که از میزان Congestion داره میاد و اندازه پنجره ارسال رو تنظیم میکنه این اندازه رو ما براساس براوردمون از وضعیت ازدحام در شبکه با cwnd نشون میدیم پس cwnd براورد فرستنده است از اندازه پنجره مناسب متناسب با وضعیت ازدحام در شبکه
پس سمت فرستنده میاد اندازه پنجره ارسال رو روی مقداری که ماکزیممش cwnd است ست میکنه
پس ما برای اندازه پنجره ارسال در tcp دوتا سقف به دست اوردیم:
rwnd: اندازه پنجره ای است ک سمت گیرنده به ما اطلاع رسانی شده
cwnd: اندازه پنجره ای است که خودمون براورد کردیم در مورد وضعیت ازدحام شبکه و اندازه پنجره ما از هیچکدوم از این ها نباید بیشتر باشه و با کمترین اون باید اینو تنظیم بکنیم

TCP Congestion Control

- end-to-end control (no network assistance)
- Sender limits transmission

LastByteSent-LastByteAcked ≤ **cwnd**

Throughput ≤ **cwnd**/RTT bytes/sec

Note: For now consider **rwnd** to be very large such that the send window size is always set equal to **cwnd**

-
پس به طور خلاصه:
کنترل Congestion توسط نودهای انتهایی است ینی سمت فرستنده بدون کمک گرفتن از توی نودهای توی شبکه از خود روترها و با تنظیم کردن حد ارسال اطلاعات
که برای Congestion کنترل گفتیم که:
اندازه پنجرمون ینی اون قسمت باز پنجره که اطلاعات ارسال شده اند این باید از cwnd کمتر باشه دراین صورت حجم اطلاعاتی که توی شبکه می فرستیم در این صورت از cwnd/RTT کمتر خواهد بود
در ادامه درس ما فرض میکنیم که rwnd که ریسیور اعلام میکنه به اندازه کافی بزرگ است و هیچ وقت محدود کننده نیست پس فرضمون بر این خواهد بود که اندازه پنجره ارسال رو cwnd تعیین میکنه و از این به بعد درباره cwnd صحبت میکنیم

TCP Congestion Control

- How does sender estimate network congestion?
  - Packet loss is considered as an indication of network congestion
    - Time Out
    - Duplicate Acks
  - TCP sender reduces cwnd after a loss event

- How does sender determine **cwnd** size?
  - Sender adjusts existing cwnd according to the loss events
    - AIMD (Additive Increase Multiplicative Decrease)

-
سمت فرستنده خودش باید وضعیت Congestion در شبکه رو براورد بکنه و این رو از روی اثار و نشونه هایی که در مورد پکت لاس در شبکه می بینه می تونه براورد بکنه و ما چه اثار و نشونه هایی از پکت لاس در شبکه می تونیم در سمت فرستنده داشته باشیم؟
دوتا وضعیت هست که اگر اتفاق بیوفته به معنای گم شدن بسته ها در شبکه خواهد بود:
timeout: اگر یک بسته ای نرسه و اک نمیشه و فرستنده منتظر می شه و تایم اوت میکنه
duplicate ack: ما اگر Duplicate Acks دریافت بکنیم این هم به این معناست که یک بسته ای اون وسط گم شده
تفاوت این دو این است که:
در تایم اوت ما یک بسته ای که گم بشه بسته های بعدی هم اگر ارسال بشن گم میشن مگر این که ارسال نشده باشن ولی در Duplicate Acks ما یک بسته ای گم شده ولی بسته های بعدی که ارسال شده اند دریافت شده اند و اک شدن ولی اک بسته ای که گم شده تکرار شده
به این ترتیب سمت فرستنده tcp اگر تایم اوت و Duplicate Acks اتفاق بیوفته باید اندازه پنجره ارسال رو کاهش بده
مقدار اندازه پنجره ارسال به طور دقیق چجوری تعیین میشه؟ این براساس loss events ینی تایم اوت و Duplicate Acks تعیین میشه ولی چگونه؟ از روش AIMD

TCP congestion control: AIMD

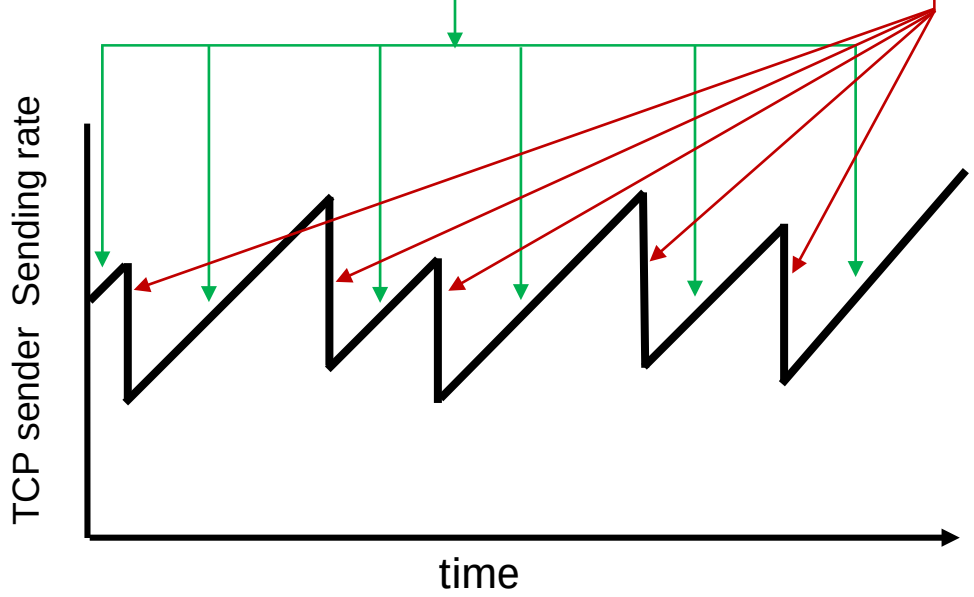
- *approach:* senders can increase sending rate until packet loss (congestion) occurs, then decrease sending rate on loss event

Additive Increase

increase sending rate by 1 maximum segment size every RTT until loss detected

Multiplicative Decrease

cut sending rate in half at each loss event



AIMD sawtooth behavior: *probing* for bandwidth

-
: AIMD
دیدگاه این است که ما اندازه پنجره ارسال رو به صورت جمع شونده افزایش میدیم ین پنجره رو بزرگتر میکنیم و به نوعی منتظر می شیم ببینیم چه اتفاقی می افته ینی توی فاز probing در واقع هستیم و این ادامه میدیم تا زمانی که یک لاس اتفاق بیوفته و این رو براساس تایم اوت یا Duplicate Acks درسمت فرستنده تشخیص میدیم
توی این فاز که بهش Additive Increase می گیم ما اندازه پنجره ارسال رو در هر RTT به اندازه یک ماکزیمم سگمنت سایز ینی MSS اضافه میکنیم
در tcp ما اندازه پنجره ارسال رو براساس بایت تنظیم می کنیم به این ترتیب cwnd به اندازه تعداد بایت MSS --> ینی حداکثر اندازه مجاز سگمنت رو به اون تعداد بایت اضافه میکنیم به پنجرمون
: Multiplicative Decrease
ما اگر یک loss events رو تشخیص دادیم میایم و اندازه پنجره رو نصف میکنیم ینی ضربدر 1/2 میکنیم این منجر به یک وضعیت خاصی در وضعیت پنجره میشه --> شکل: ینی اندازه پنجره اضافه میشه به صورت خطی تا اینکه یک لاس اتفاق بیوفته و در اینجا اندازه پنجره نصف میشه و دوباره اندازه پنجره اضافه میشه تا برسه به حالتی که لاس اتفاق بیوفته و دوباره نصف میکنیم و این همین جوری هی تکرار میشه --> این هست که براساس پروتکل tcp که برقرار شده ما اگر ریت لحظه ای ترافیک رو مانیتور بکنیم می تونیم یک همچین رفتاری رو ببینیم

TCP AIMD: more

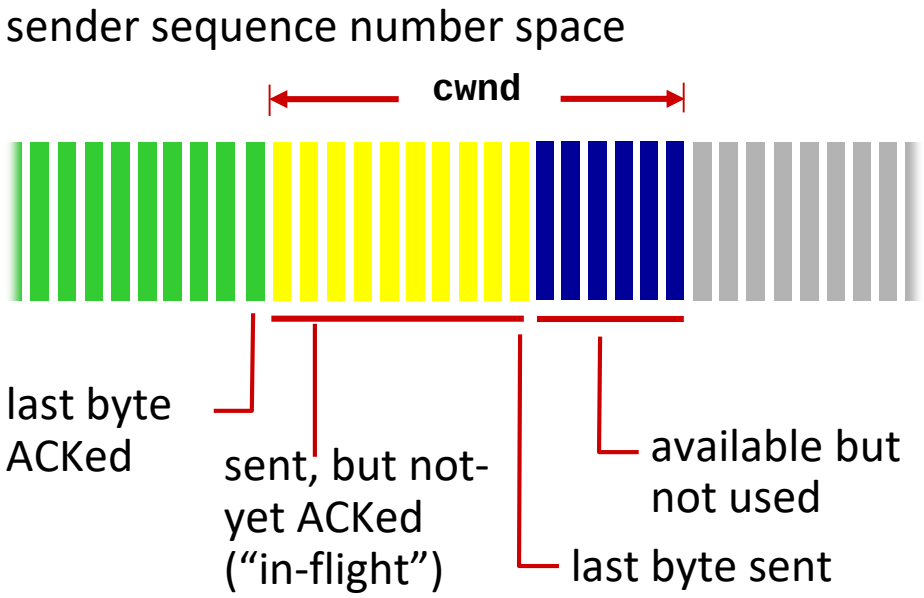
Multiplicative decrease detail: sending rate is

- Cut in half on loss detected by triple duplicate ACK (TCP Reno)
- Cut to 1 MSS (maximum segment size) when loss detected by timeout (TCP Tahoe)

Why AIMD?

- AIMD – a distributed, asynchronous algorithm – has been shown to:
  - optimize congested flow rates network wide!
  - have desirable stability properties

TCP congestion control: details



TCP sending behavior:

- *roughly:* send cwnd bytes, wait RTT for ACKS, then send more bytes

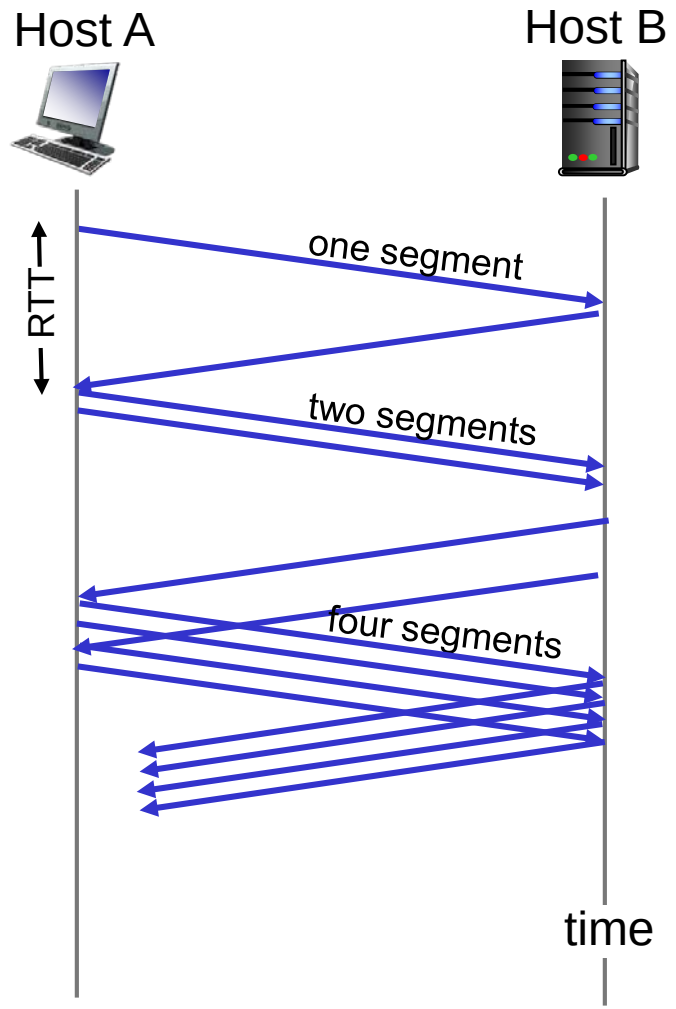
$$\text{TCP rate} \approx \frac{\text{cwnd}}{\text{RTT}} \text{ bytes/sec}$$

- TCP sender limits transmission: `LastByteSent- LastByteAcked ≤ cwnd`
- cwnd is dynamically adjusted in response to observed network congestion (implementing TCP congestion control)

-
به این ترتیب ماکزیمم اندازه پنجره برابر با cwnd گذاشته میشه پس حجم اطلاعاتی که ارسال کردیم حداکثر میتونه بشه cwnd و درعمل حتی کمتر
حجم اطلاعاتی که ارسال شده اون قسمتی از پنجره است که استفاده شده این کوچکتر از cwnd خواهد بود
و ریت هم میشه cwnd/RTT حداکثر خواهد بود
به این ترتیب ما با تنظیم cwnd داریم ریت ترافیک ورودی به شبکه رو به طور داینامیک و متناسب با وضعیت congestion توی شبکه تنظیم میکنیم

TCP slow start

- when connection begins, increase rate exponentially until first loss event:
  - initially **cwnd** = 1 MSS
  - double **cwnd** every RTT
  - done by incrementing **cwnd** for every ACK received
- *summary:* initial rate is slow, but ramps up exponentially fast



-
سوال این است که در ابتدا اندازه پنجره چطور تنظیم میشه؟
tcp از مکانیزیم slow start برای تنظیم اندازه پنجره ارسال استفاده میکنه
و هدف از slow start این است که یک براورد سریع از وضعیت ازدحام توی شبکه به دست بیاد که این رو بهش میگیم فاز probing و بعد تنظیم اندازه پنجره ارسال براساس اون ادامه پیدا بکنه
برایی این کار در slow start اندازه پنجره ارسال ابتدا برابر با 1MSS قرار گرفته میشه برای مثال اگر MSS=500byte است و و توی اون لحظه RTT اندازه گیری شده بین فرستنده و گیرنده هم 200 میلی ثانیه بوده در این صورت ما داریم با یک ریت 2500byte/sec ارسال اطلاعات به شبکه رو شروع میکنیم
به هر حال ممکنه واقعا عرض باندی توی شبکه خیلی بیشتر از MSS/RTT باشه و ما دوست داریم که خیلی سریع ترافیک ارسالی رو افزایش بدیم ت این ظرفیت موجود رو پر بکنیم
به همین دلیل توی slow start ما به جای افزایش خطی اندازه پنجره ارسال و در نتیجه ریت ارسال به شبکه این رو به صورت نمایی افزایش میدیم به این صورت که در هر RTT اندازه پنجره ارسال رو دوبرابر میکنیم --> شکل: در یک RTT ما یک بسته فرستادیم چرا یک بسته؟
چون اندازه پنجره رو ابتدار برابر 1MSS قرار دادیم ینی اندازه بایت هایی که 1 سگمنت رو فقط می تونن پر بکنن بنابراین یک بسته ارسال شده و اک اومده و بعد اینجا ما این رو دو برابر میکنیم --> مکانیزیم عملی که این کار اتفاق بیوفته این است که ما به ازای هر اکی که از سمت گیرنده دریافت میکنیم یک 1 سگمنت سایز به اندازه پنجرمون ارسال میکنیم --> پس توی شکل در ابتدا یک بسته رفته یک اک اومده و ما اندازه پنجرمون 1MSS بود حالا میشه 2MSS حالا گر این ها پشت سر هم ارسال بشن بعد از RTT یکی یکی اک هایی اینا برمیگرده و به ازای هر کدومشون اگر 1MSS اصافه بکنیم مثلا به ازای دوتا میشه 2 تا MSS و اگر 2 بود میشه 4 تا
پس عملا این است که به ازای هر اکی که از اونور دریافت میشه ما 1MSS به اندازه پنجرمون اضافه میکنیم و به این ترتیب اندازه پنجره در هر RTT دوبرابر میشه
به این ترتیب ما در مکانیزیم slow start ریت اولیمون پایین است و خیلی سریع و به صورت نمایی این افزایش پیدا میکنه

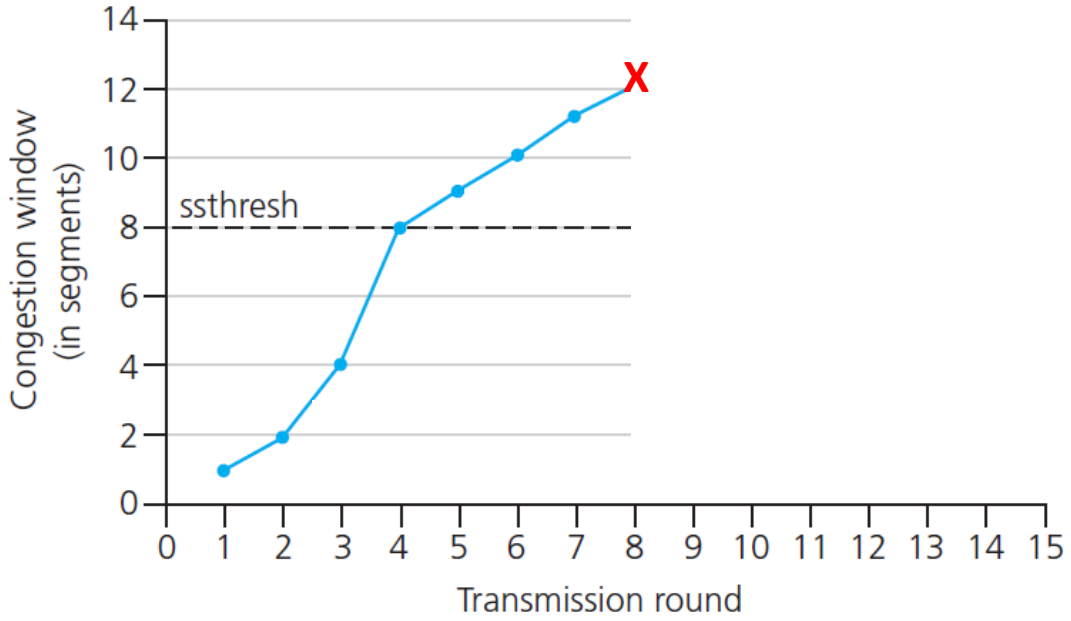
TCP: from slow start to congestion avoidance

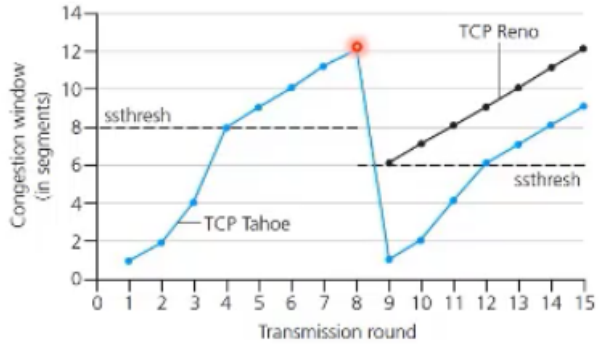
Q: when should the exponential increase switch to linear?

A: when **cwnd** gets to 1/2 of its value before timeout.

Implementation:

- variable **`ssthresh`**
- on loss event, **`ssthresh`** is set to 1/2 of **`cwnd`** just before loss event

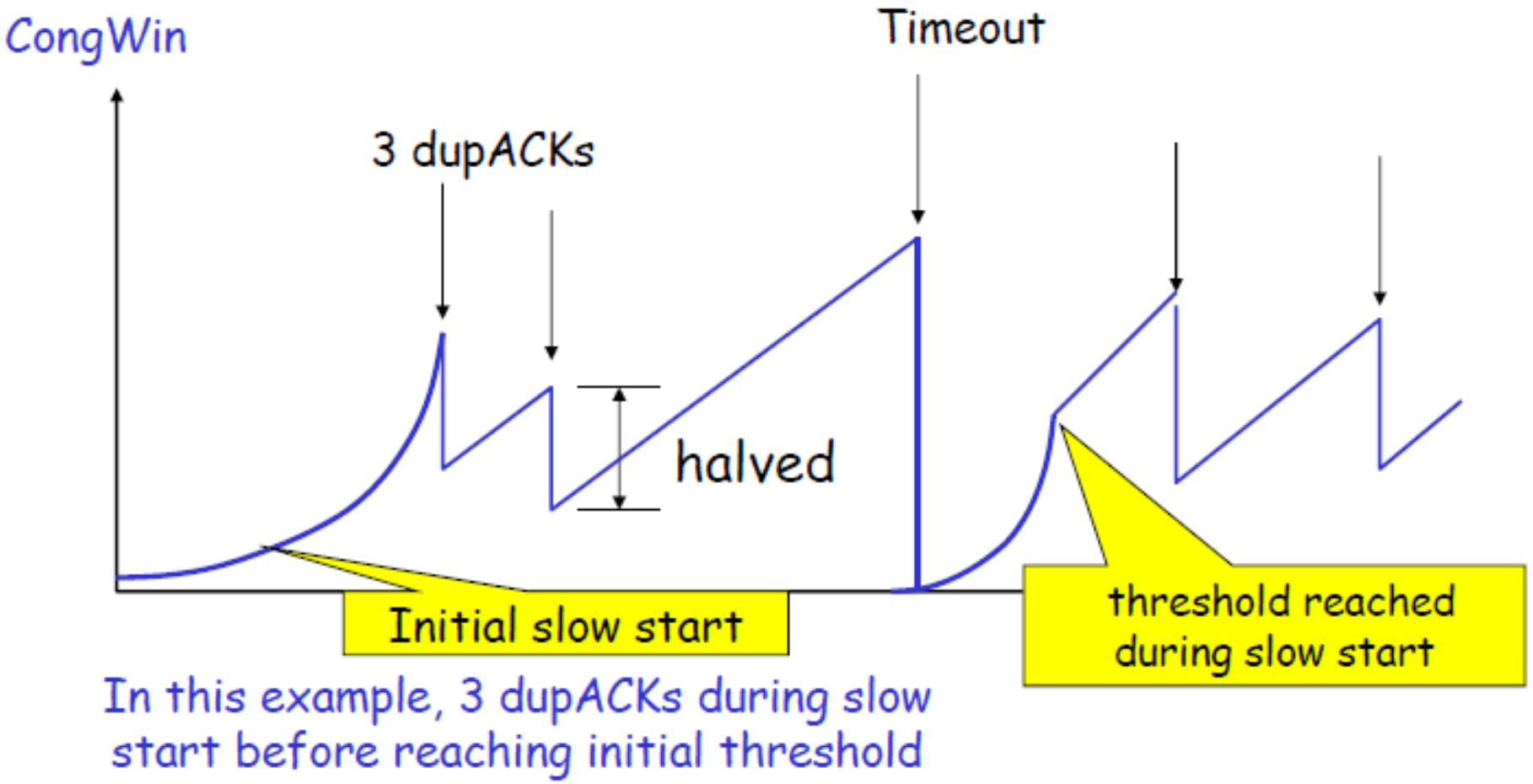




* Check out the online interactive exercises for more examples: http://gaia.cs.umass.edu

-
نکته:
slow start نه تنها در ابتدای شروع به کار کانکشن استفاده میشه بلکه در حین کانکشن هم هر موقع لاس اتفاق یبوفته که اینو با تایم اوت تشخیص میدیم عین همین کار تکرارمیشه ینی وقتی که تایم اوت اتفاق اتفاد cwnd رو 1MSS تنظیم میشه و بعد به صور نمایی این افزایش پیدا میکنه و در هر RTT دوبرابر میشه تا به یک حداکثری برسه و بعد به صورت خطی شروع میکنیم Additive Increase رو افزایش میدیم چرا وقتی که تایم اوت اتفاق میافته ما از مکانیزیم slow start استفاده میکنیم؟ چون تایم اوت وقتی اتفاق می افته که ما یک بسته گم شده بسته های بعدی هم گم شدن ینی بسته های متوالی که داریم می فرستیم از یه جا به بعد همشون گم شدن که این نشون دهنده وضعیت ازدحام شدید است در نتیجه ما ریت ارسال رو خیلی پایین میکنیم
ولی اگر Duplicate Acks اتفاق افتاده باشه و ما پکت لاس رو براساس Duplicate Acks تشخیص داده باشیم در این صورت وضعیت خیلی بحرانی نیست ینی ما یک بستمون گم شده ولی بسته های بعدی دارن می رسن پس لزومی نداره که ما عکس العمل تند داشته باشیم در نتیجه ما میایم cwnd رو نصف میکنیم
این روش که برای Duplicate Acks گفتیم برای ورژن tcp reno است که ورژن جدیدتری از tcp است
این صفحه:
بعد از تایم اوت اندازه پنجره ارسال رو برابر 1MSS قرار میدیم
نمودار: اینجا محور افقی زمان رو نشون میده و محور عموی اندازه پنجره congestion رو نشون میده --> در ابتدا این رو برابر با 1MSS قرار میدیم و بعد اینو به صورت نمایی افزایش میدیم و سوال این است که تا کجا باید اینو ادامه بدیم و بعد کجا باید سوییچ بکنیم به حالت افزایش خطی؟ اینجا یک ssthresh داریم که وقتی اندازه پنجرمون congestionمون به ssthresh رسید اینو رو به صورت خطی ادامه میدیم و ssthresh به این صورت تعیین میشه که قبل از اینکه ما loss event رو تشخیص بدیم اندازه پنجره هرچی بود اون رو نصف میکنیم و اون رو به عنوان ssthresh استفاده می کنیم --> مثلا اگر تایم اوت اتفاق افتاده قبلا پنجره ارسالمون 16 بود نصفش میشه 8 به این ترتیب ما ازاین مقدار به بعد به صورت خطی ادامه میدیم ینی در هر زمان به اندازه 1MSS به اندازه پنجرمون اضافه میشه و همینطور مثلا ادامه پیدا میکنه و مثلا توی 12 فرض میکنیم 3 تا اک تکراری دریافت کردیم وقتی که 3 تا اک تکراری دریافت بکنیم میایم اندازه پنجره رو ارسال رو نصف میکنیم ینی میشه 6 الان و ssthresh هم میشه نصف اندازه پنجره congestion قبل از اتفاق لاسمون است که توی 12 بوده پس این هم میشه 6 و به این ترتیب ما اندازه پنجرمون بالای ssthresh است پس به صورت خطی اینو افزایش میدیم خط مشکلی الان شده

TCP Reno (example scenario)



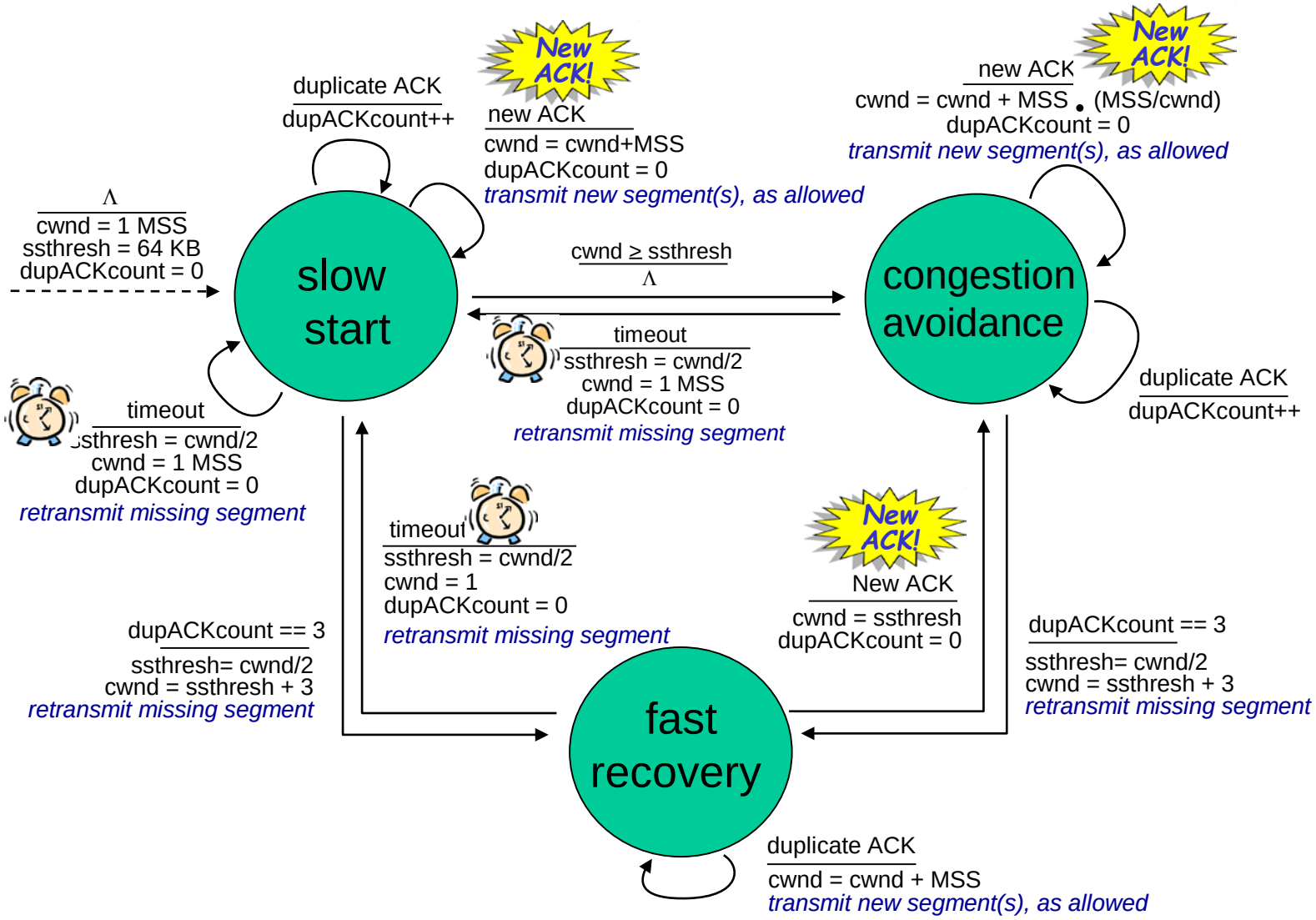
ادامه صفحه قبلی:
مقدار threshold در ابتدای کار چه مقدار باید تنظیم بشه؟
برای اولین slow start که در ابتدای کانکشن اینو شروع میکنیم در این صورت اندازه threshold میگه توی یک مقدار خیلی بزرگ ست بشه ینی ما ابتدای کار threshold رو در نظر میگیریم و از همون ابتدا به صورت نمایی ادامه تا به اولین اتفاق لاس برسیم و بعد از اون انداه پنجره هرچی هست نصف میشه واین به عنوان threshold استفاده میشه و این threshold مقدار ثابتی نیست و در طول ارتباط تغییر میکنه
این صفحه:
مثال:
در ابتدا با مقدار congestion window یک شروع کردیم و به صورت نمایی اینو افزایش دادیم تا رسیدیم به 3 تا اک تکراری و این نصف میشه و threshold هم نصف میشه و به صورت خطی ادامه میدیم و...
حالا تایم اوتی که اتفاق افتاده: در تایم اوت ما میایم اینو از ابتدا شروع میکنیم ینی روی یک تنظیم می کنیم و به صورت نمایی ادامه میدیم تا به threshold برسیم و این threshold نصف مقدار پنجره ازدحام درست قبل از تایم اوت است و بعد به صورت خطی ادامه میدیم و باز به همین صورت...

Summary (TCP Reno)

- When cwnd is below Threshold, sender in slow-start phase, window grows exponentially (until loss event or exceeding threshold).
- When cwnd is above Threshold, sender is in congestion-avoidance phase, window grows linearly.
- When timeout occurs, Threshold set to cwnd/2 and cwnd is set to 1 MSS.
- When a triple duplicate ACK occurs, Threshold set to cwnd/2 and cwnd set to Threshold (also fast retransmit happens).

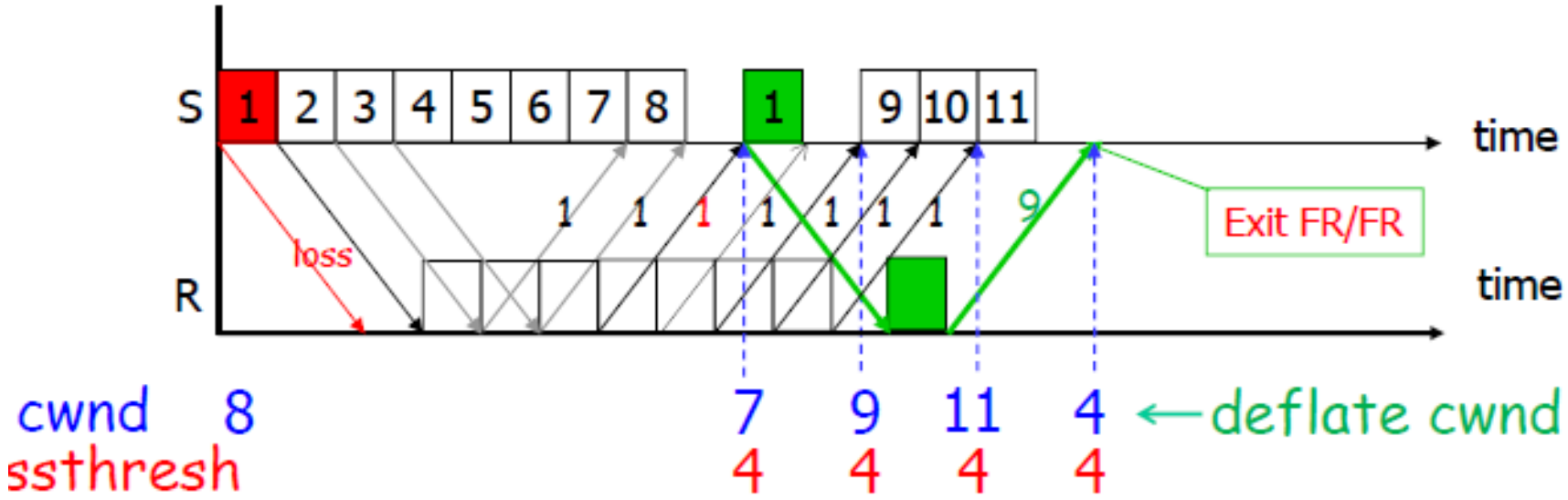
-
خلاصه:
تا زمانی که cwnd پایین threshold است ما در مد slow start هستیم و اندازه پنجره ازدحام به صورت نمایی اضافه میشه و این ادامه پیدا میکنه تا یک اتفاق لاس انجام بشه و بعد مثل قبل باید انجام بدیم هر کدوم که بود --> همون حرفایی که صفحه های قبلی زدیم
اگر اندازه پنجره ازدحام بالای threshold بود به این حالت میگیم حالت congestion avoidance و این حالت حالتی است که اندازه پنجرمون به صورت خطی افزایش پیدا میکنه
اگر تایم اوت اتفاق بیوفته threshold مون میشه نصف اندازه cwnd قبل از تایم اوت و Cwnd خودش ست میشه روی 1MSS
اگر سه تا اک تکراری دریافت بکنیم threshold میشه cwnd/2 و خود cwnd هم ست مشه روی همون مقدار threshold اینجا یک اتفاق دیگه هم می افته به نام fast retransmit که میگیم جلوتر

Summary: TCP congestion control



خلاصه:
ابتدا در سمت فرستنده در حالت slow start قرار داریم در این حالت cwnd=1MSS است و ssthresh یک مقدار خیلی بزرگ که اینجا 64KB است و یک پارامتری هم داریم به اسم dupACKcount ینی شمارنده اک های تکراری که در ابتدا این صفر است
اگر تایم اوت اتفاق بیوفته: اگر در مرحله slow start تایم اوت اتفاق بیوفته دوباره برمیگردیم به ابتدا مرحله slow start ینی مقادیر قبلی مثل قبل ولی ssthresh حالا میشه اینجا cwnd/2
اگر اک تکراری اتفاق بیوفته: توی این حالت اون شمارنده رو یکی زیاد میکنیم و در این حالت اگر اک جدید بیاد cwnd = cwnd +MSS ینی به ازای هر اک جدید ما یک 1MSS اضافه میکنیم به cwnd و اون شمارنده صفر میشه و به دنبال این ما بسته جدید می فرستیم و اگر پنجره باز جا داشته باشه بسته های بیشتری می فرستیم --> در این حالت داریم به صورت نمایی اندازه پنجره رو افزایش میدیم و این رو ادامه میدیم تا جایی که cwnd بزرگتر و مساوی ssthresh بشه
در این حالت ما وارد حالت congestion avoidance میشیم در این حالت ما هنوز داریم نرخ ارسال رو افزایش میدیم و اندازه پنجره رو افزایش میدیم ولی به صورت خطی این افزایش انجام میشه و هر اک جدیدی که دریافت میشه همون فرمولی که نوشته توی اسلاید میشه cwnd مون--> این اندازه به گونه ای است که ما اگر به اندازه تعداد بسته های توی پنجره که ارسال شدن و اکشون داره برمیگرده ....نصفه است

Fast Recovery entry and exit



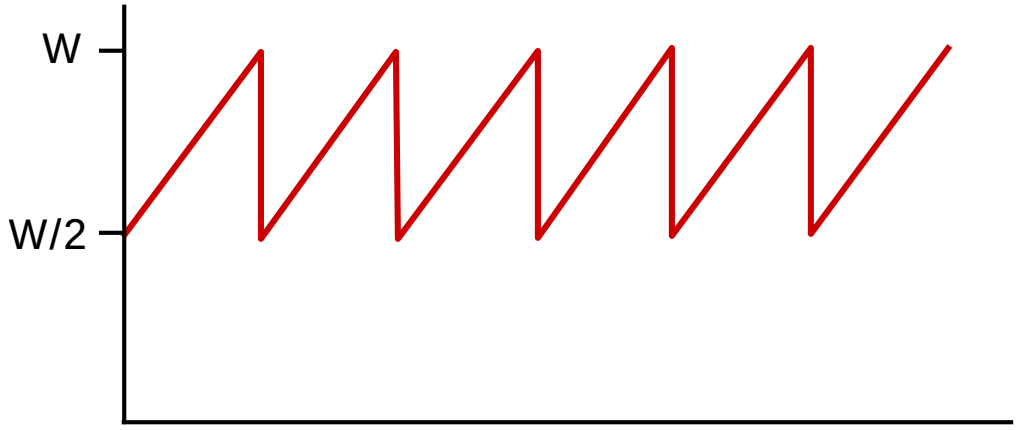
- Above scenario: Packet 1 is lost, packets 2, 3, and 4 are received; 3 dupACKs with seq. no. 1 returned
- Fast retransmit
  - Retransmit packet 1 upon 3 dupACKs
- Fast recovery (in steps)
  - Inflate cwnd with #dupACKs such that new packets 9, 10, and 11 can be sent while repairing loss

:fast retransmit
در این مثال ما 1 رو ارسال کردیم و 1 گم شده و به مقصد نرسیده و پشت سرش بسته های دیگه ارسال شدن
و فرض میکنیم اندازه پنجره 8 سگمنت سایز بوده
2 و 3 و 4 وقتی ارسال شدن این ها با ترتیب نادرست در سمت گیرنده دریافت شدن و در نتیجه اخرین اکی که ارسال شده بود که قاعدتا اک یک است یکی اک بسته قبل از یک که یک رو درخواست کرده بود این اک تکرار میشه ینی 3 تا اک تکراری اینجا خواهیم داشت و سمت فرستنده این یک رو دوباره می فرسته و اگرفرض کنیم این دفعه بسته رسیده به مقصد و مقصد هم بسته های قبلی رو نگهداری کرده در نتیجه وقتی که 1 رو دریافت کردیم و قبلش تا 8 رو هم دریافت کرده بودیم و بسته بعدی که انتظارشو داریم 9 است الان پس اک 9 ارسال میشه پس به این ترتیب در یک زمان کوتاهی ارسال اطلاعات رو به روال عادی برمی گردونیم که به این میگیم Fast Recovery
اگر ما برحسب پروتکل congestion کنترل tcp اندازه پنجره رو نصف بکنیم در این حالت نمی تونیم این یک رو بفرستیم چون پنجرمون اجازه نخواهد داد برای همین ما در این پروتکل Tcp این کارو انجام میدیم مثلا اینجا که cwnd=8 بود و 3 تا اک تکراری دریافت کردیم و طبق پروتکل این باید بشه 4 ولی عملا به اندازه اک های تکراری که دریافت کردیم اینو اضافه می کنیم و اینجا 3 تا اک تکراری بوده پس میشه 7 و در ادامه اک های تکراری بعدی که دریافت بشه ینی اینجا بعد از 2 و 3 و 4 اومدیم 5 و 6 و 7 و 8 هم فرستادیم و اک های اونا هم برمیگرده و اونا هم اضافه می کنیم ینی ما به ازای هر اک تکراری داریم اندازه پنجره رو اضافه میکنیم و اینو ادامه میدیم تا جایی که اک جدید بیاد مثلا توی این مثال میشه اک 9 که جدید بود در اینجا الان برمیگردیم به حالت عادی که همون 4 میشه و 4 رو ست می کنیم و ادامه میدیم --> فلسفه این 4 چیه؟ ینی میخوایم ریت ارسال رو پایین بیاریم و چرا قبلش اندازه پنجره رو بزرگتر گرفتیم؟ برای اینکه فرصت این Recovery رو بدیم ینی فرصت fast retransmit رو بدیم و اجازه بدیم وضعیت sliding window بیاد و Recover بشه و بعد حالت عادی که شد با ریت پایین تر ارسال رو ادامه میدیم

TCP throughput

- avg. TCP thruput as function of window size, RTT?
  - ignore slow start, assume there is always data to send
- W: window size (measured in bytes) where loss occurs
  - avg. window size (# in-flight bytes) is ¾ W
  - avg. thruput is 3/4W per RTT

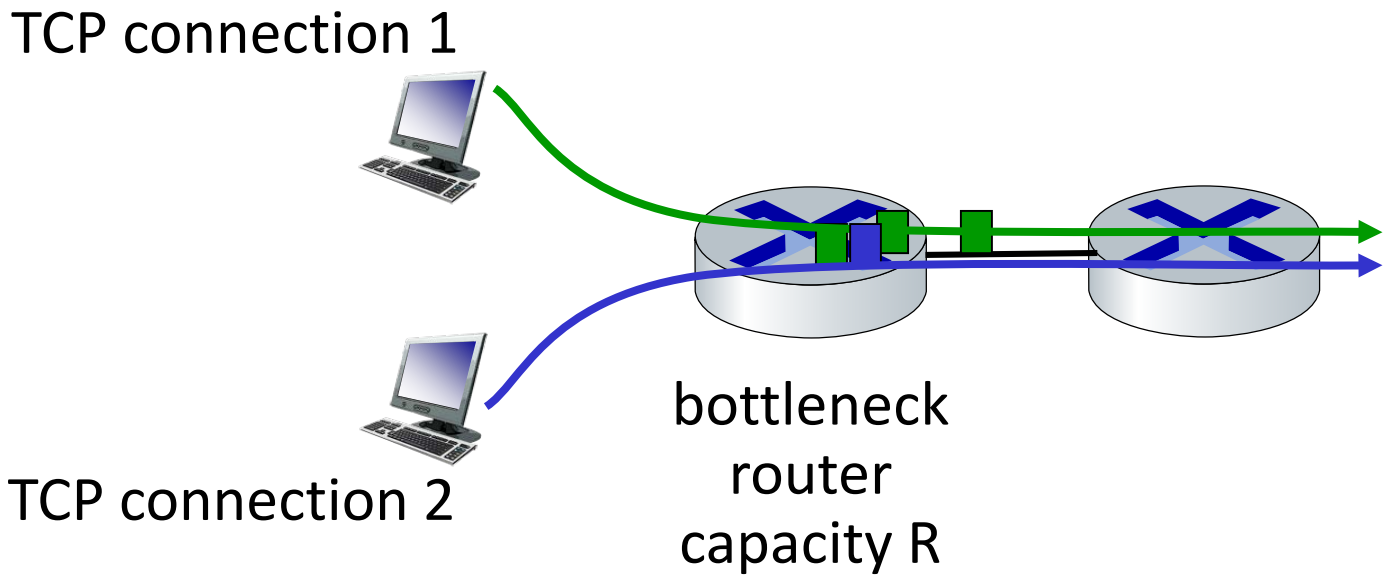
$$\text{avg TCP thruput} = \frac{3}{4}\frac{W}{RTT} \text{ bytes/sec}$$



-
اگر فرض کنیم که در حالت پایدار با افزایش خطی اندازه پنجرمون به w می رسه و اینجا اون نقطه ای که است که حد تحمل شبکه است و اینجا اولین drop اتفاق می افته و اینجا اندازه پنجره نصف میشه و برابر با w/2 میشه و بعد به طورخطی این رو افزایش مبدیم و بعد انتظار میره در همون حدود w دوباره drop بعدی اتفاق بیوفته این برای حالتی است که شرایط شبکه تغییرات زیادی نکرده
به این ترتیب پنجره ما در حالت پایدار بین w, w/2 همچین نوسانی خواهد داشت و در این صورت اون چیزی که throughput تعیین میکنه اندازه متوسط اون است
این اندازه متوسط اگر اندازه پنجره همچین حالتی داشته باشه برابر با 3/4 است --> ینی w+ w/2 و کلش تقسیم بر 2 میشه 3/4w
که دراین رابطه W اندازه پنجره در شرایط پایدار شبکه است و اون اندازه ای است که برای این کانکشن شروع میکنه به drop کردن
RTT هم علاوه بر ارتباطش با فاصله و تعداد هاپ بین راه با وضعیت تاخیر صف در نودهای بین راه هم مرتبط است و اون هم تابعی است از وضعیت ترافیک در حالت پایداری که الان است

TCP fairness

Fairness goal: if *K* TCP sessions share same bottleneck link of bandwidth *R*, each should have average rate of *R/K*

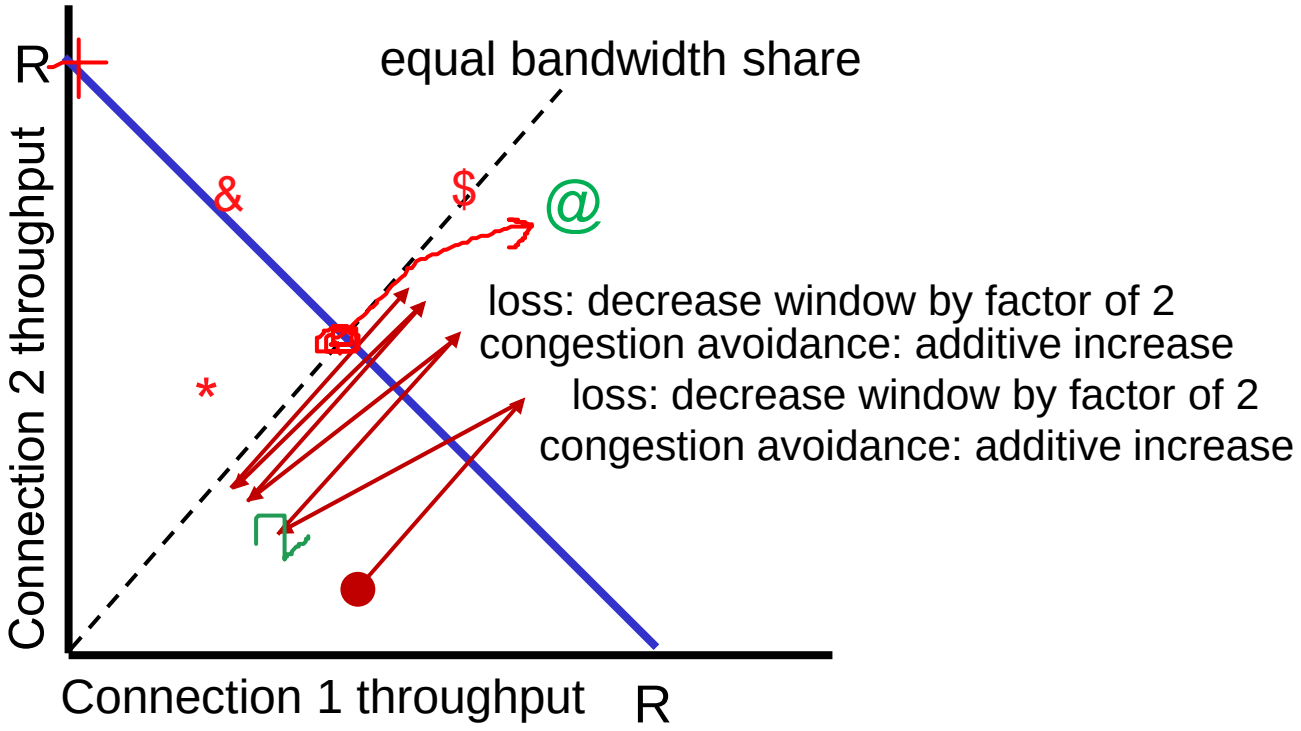


-
اگر دوتا کانکشن tcp در یک لینکی با ظرفیت r مشترک هستن هر کدوم از اونها r/2 عرض باند لینک رو استفاده می کنن
پس اگر k تا کانکشن داشته باشیم و هر کدوم R/K عرض باند لینک رو استفاده بکنن می تونیم بگیم که کانکشن ها به صورت fairness ینی عادلانه دارند از عرض باند لینک مشترک استفاده می کنند --> اینو قبل تر گفته بودیم
ایا واقعا tcp این عدالت رو براورد میکنه یا نه؟
هر کانکشن براساس وضعیت ترافیکی شبکمون اندازه پنجره اش تنظیم میشه و در حالت پایدار throughput میشه 3/4w/RTT که این کانکشن می فرسته و اون عرض باندی که داره از این لینک استفاده میکنه و ایا این مستقل از throughput کانکشن های دیگری که از این لینک می گذرند هست یا با اونها ارتباط داره؟ ایا این عدالت واقعا در tcp وجود داره یا نه؟
ما ادعا میکنیم که Tcp عادلانه عمل میکنه
ادامش صفحه بعدی..

Q: is TCP Fair?

Example: two competing TCP sessions:

- additive increase gives slope of 1, as throughout increases
- multiplicative decrease decreases throughput proportionally



Is TCP fair?

A: Yes, under idealized assumptions:

- same RTT
- fixed number of sessions only in congestion avoidance

-
در این دیاگرام ما تغییرات منحنی throughput دوتا کانکشن نسبت بهم دیگه رسم میکنیم:
در محور افقی throughput کانکشن 1 و در محور عموی throughput کانکشن 2
رای مثال اگر throughput کانکشن 2 صفر باشه throughput کانکشن 1 هر مقداری باشه روی این محور هرمقداری رو مشخص میکنه و اگر کانکشن 1 throughput اش صفر باشه و اگر کانکشن 2 یک ترافیکی رو بفرسته مقدار اون نقطه ای در محور عمودی خواهد بود و اگر تمام عرض باند رو اشغال بکنه + میشه در حالت کلی throughput کانکشن 1 مثلا x باشه و throughput کانکشن 2 مثلا y باشه یک نقطه در صفحه مشخص میشه*
اگر جمع کل throughput کانکشن 1 و کانکشن 2 کل عرض باند رو اشغال بکنه یک نقطه روی خط & خواهد بود این خط & مجموع نقاطی رو نشون میده که جمع مقدار های x,y برابر با R است
اگر throughput کانکشن 1 و کانکشن 2 برابر باشه نقطه روی خط $ خواهد بود و اگر این ها به طور مساوی عرض باند لینک رو استفاده بکنند و اون رو پر بکنند ما توی نقطه @ خواهیم بود
به صورت کلی فرض میکنیم در یک نقطه ای هستیم که توی اسلاید خودش نشون داده و در حالت پایدار هر دوی اینا به صورت خطی در حال افزایش هستن و اندازه پنجره به صورت خطی اضافه میشه و در نتیجه throughput به صورت خطی اضافه میشه و میزان افزایش این دوتا برابر است چرا؟ اگر ماکزیمم سگمنت سایز در کل شبکه برای همه کانکشن ها یکسان است در این صورت هر کدوم از این ها در یک RTT به اندازه 1MSS به اندازه پنجرشون اضافه می کنند و در نتیجه میشه گفت این ها throughput شون به یک اندازه افزایش پیدا میکنه در نتیجه نقطمون روی این خط توی اسلاید که خطی است به موازی خط $ که شیبش یک است جابه جا میشه و به این ترتیب نقطمون بعد از مدت ها به خط & می رسه و وقتی که & می رسه معنیش این است که میزان throughput کانکشن 1 و کانکشن 2 مجموعشون کل عرض باند رو اشغال کرده و اگر ما کماکان throughput هارو افزایش بدیم و روی همین خط قرمز توی اسلاید ادامه میدیم و حالا ما نزدیک مرحله congestion هستیم چون مجموع ترافیکی که این دوتا کانکشن روی این لینک دارند می فرستن از عرض باند اون بیشتر است پس اینجا صف داره تشکیل میشه و قاعدتا یه جایی بلاخره طول صف اونقدر خواهد شد که شروع می کنه به drop کردن و اگر ما به مرز congestion رسیده باشیم drop از هردوتا کانکشن اتفاق می افته و هر دوی این ها سه تا اک تکراری دریافت می کنن و بعد از اون اندازه پنجرشون رو نصف میکنن و وقتی که نصف می کنن به نقطه 2 می رسیم و این نقطه باز مثل قبلی عمل میکنه ینی هر دوی این ها به صورت خطی افزایش پیدا می کنند و به همین صورت می ره باز جلو و بعد از چندین تکراری نقطمون می افته روی خط $ ینی همین نوسانات داره اتفاق می افته ولی روی خط میانی ینی عرض باندمون تقریبا به طور مساوی داره بین دوتا کانکشن تقسیم میشه پس در شرایط مساوی پروتکل tcp به صورت عادلانه عرض باند رو بین کانکشن های روی لینک تقسیم می کنه

Fairness: must all network apps be "fair"?

Fairness and UDP

- multimedia apps often do not use TCP
  - do not want rate throttled by congestion control
- instead use UDP:
  - send audio/video at constant rate, tolerate packet loss
- there is no "Internet police" policing use of congestion control

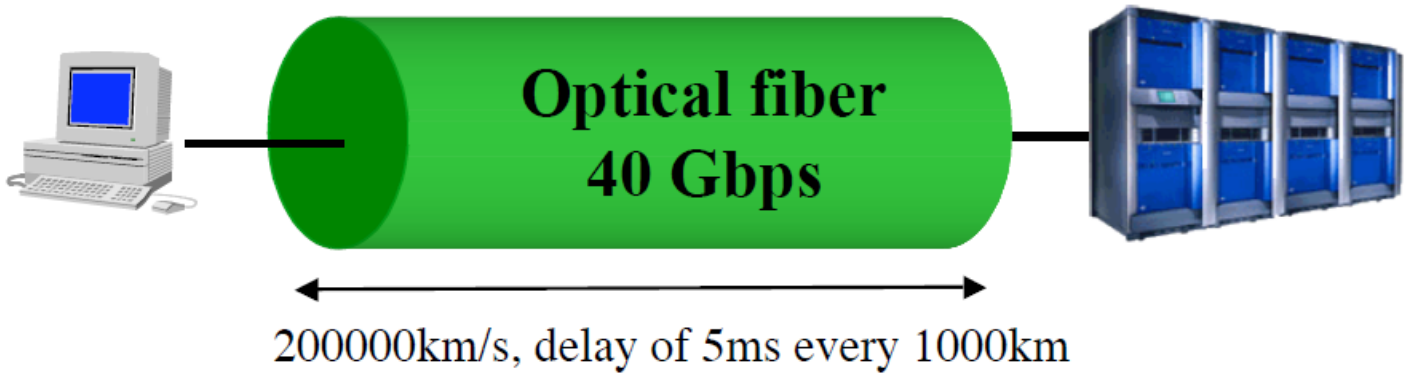
Fairness, parallel TCP connections

- application can open *multiple* parallel connections between two hosts
- web browsers do this , e.g., link of rate R with 9 existing connections:
  - new app asks for 1 TCP, gets rate R/10
  - new app asks for 11 TCPs, gets R/2

-
ایا همیشه اون شرایطی که گفتیم برای عادلانه برقرار است؟
ما فرض کردیم که لینکمون بین یه تعدادی کانکشن tcp به صورت اشتراکی داره استفاده میشه ولی میدونیم در شبکه غیر از کانکشن tcp کانشن های udp هم داریم ویکسری از پروتکل ها از udp به جای tcp استفاده می کنند به طور مشخص مالتی مدیا این ها اپلیکیشن هایی هستن که به صورت ریل تایم از udp استفاده می کنند و اینها از Tcp استفاده نمی کنند چون اگر پکت لاس یا اروری اتفاقی بیوفته در Tcp انتقال متوقف میشه و موکل میشه به ارسال مجدد اون پکت و این چیزی نیست ک یک کانکشن ریل تایم بتونه اینو تحمل بکنه
نکته: udp براساس این که اپلیکیشن چه throughput مطلبه ترافیکش داره وارد لینکمون میشه یک مسئله دیگر در این رابطه کانکشن های tcp موازی است: در این موارد کانکشن های موازی برای افزایش throughput اون کانکشن است این مسئله عادلانه ما رو زیر سوال می بره چون عادلانه بین کانکشن های Tcp به تعداد کانکشن های Tcp است ینی مثلا اگر دوتا کانکشن باشه عرض باند به نسبت 1/2 بینشون تقسیم میشه --> مثلا 9 تا کانکشن برقرار کرده به جای یک کانکشن و کانکشن دوممون با یک کانکشن برقرار میشه الان در مجموع 10 تا کانکشن tcp روی این لینک داریم و عرض باند به نسبت R/10 تقسیم میشه بین اینها و از این R/10 کانکشن جدیدمون یکی از این ها می تونه استفاده بکنه و کانکشن قبلیمون 9 تا از این هارو ینی 9/10 عرض باند رو --> ینی 9/10 عرض باند رو کانکشن اول استفاده میکنه و 1/10 رو کانکشن دوم و اگر کانکشن دوممون بخواد نصف عرض باند رو استفاده بکنه باید یه چیزی در حد 11 تا کانکشن موازی و همزمان برای خودش درست بکنه تا بتونه این نسبت رو متعادل بکنه

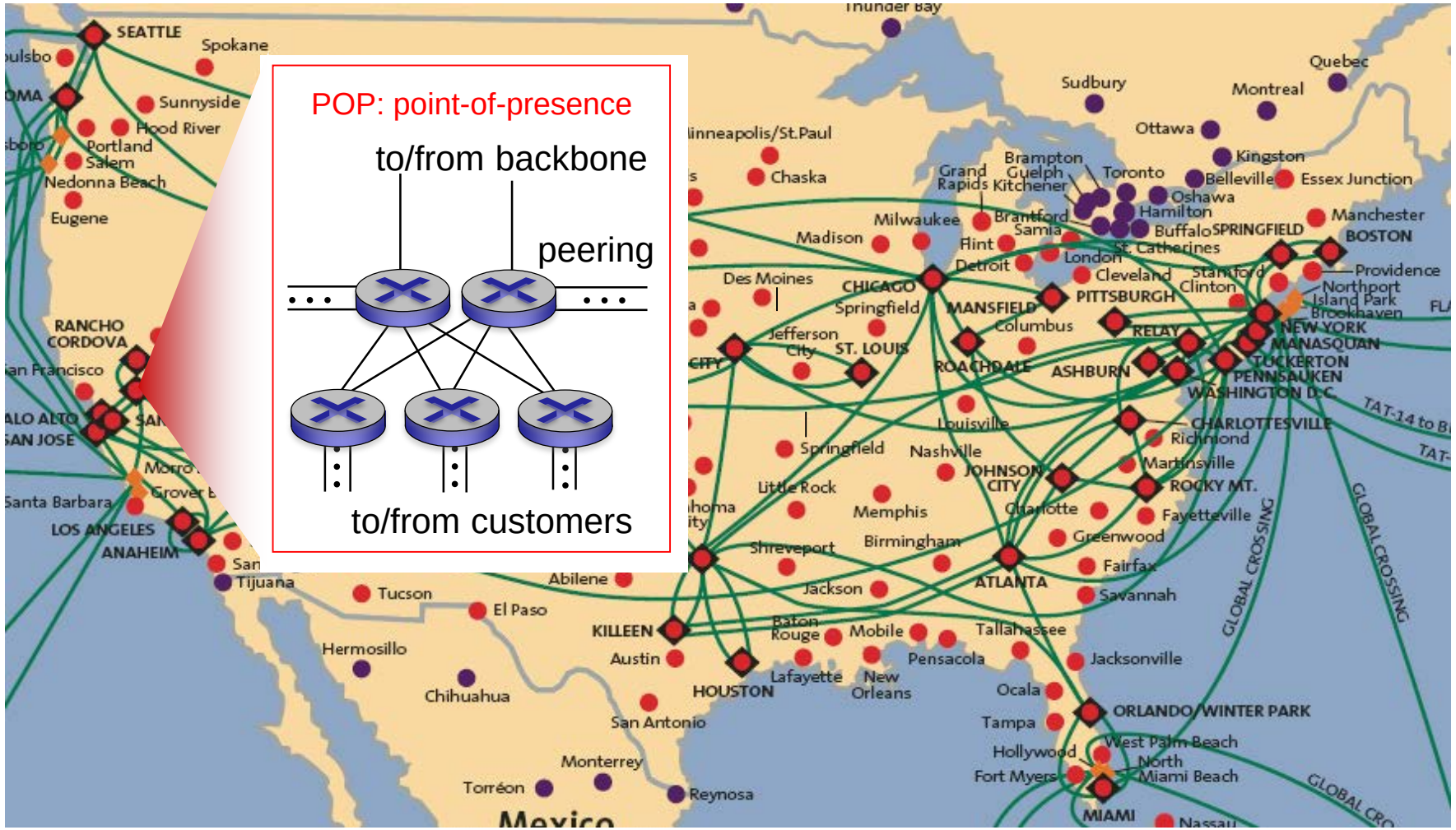
TCP in high-speed links

- Today's backbone links are optical, DWDM–based,
- High bit rates (1~100 Gbps)
- Long distances (Thousands of Km)
- Transmission time <<< propagation time

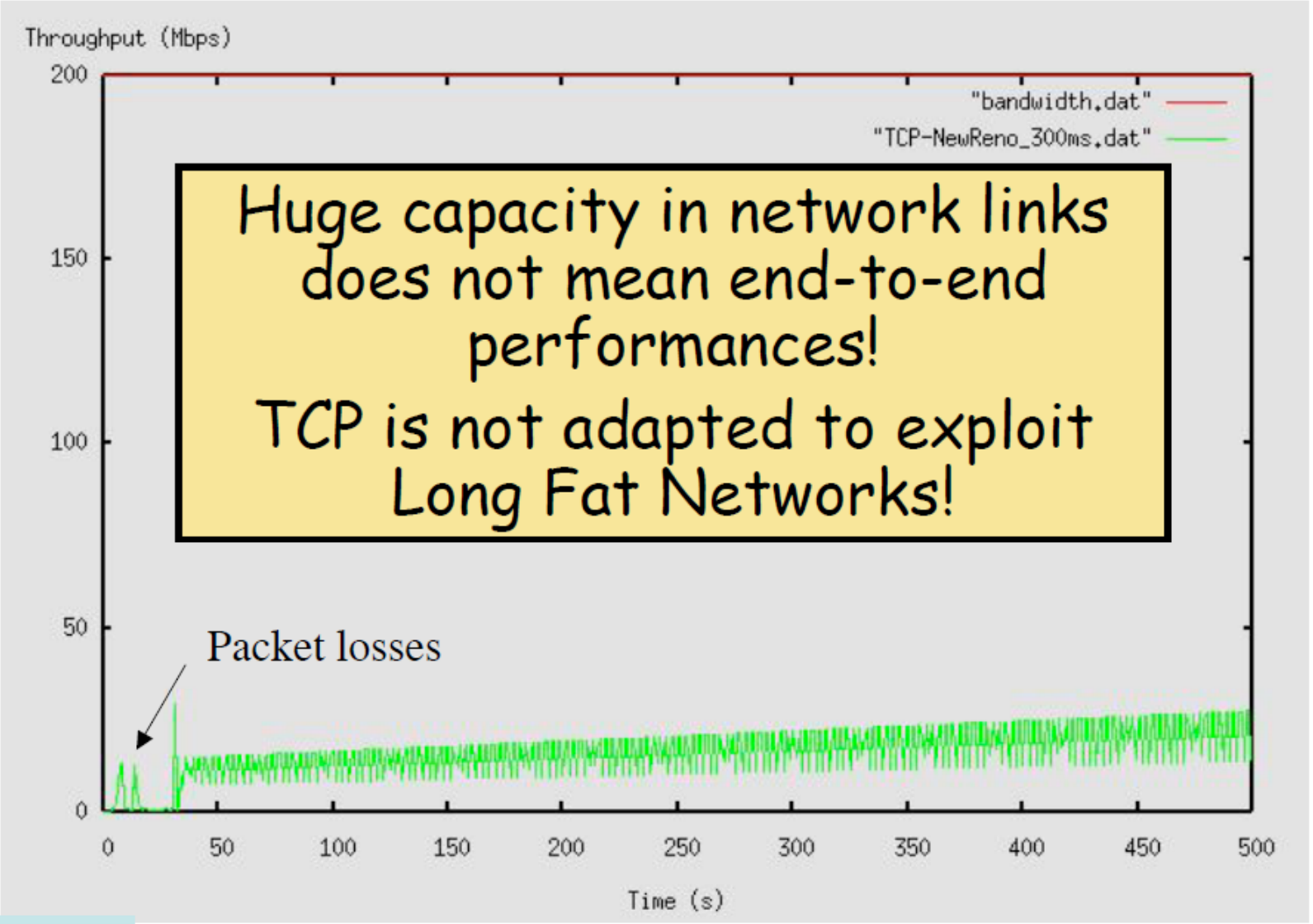


-
یک کانکشن tcp چقدر از عرض باند لینک ارتباطی رو می تونه استفاده بکنه؟
امروز لینک های ارتباطی ظرفیت بالایی دارند به دلیل استفاده از فیبرنوری یا تکنولوژی های مثل DWDM
بسیار از لینک های backbone معمولا برای مسافت های طولانی استفاده میشه
نتیجه این ها این میشه که Transmission time کمتر میشه بخاطر سرعت بالای لینک
propagation time بیشتر میشه به دلیل اینکه لینک ها در فواصل طولانی تری دارند استفاده میشن و هرچی طول لینک بیشتر باشه به هر حال سرعت انتشار موج در لینک ثابت است
مثال:
یک لینک فیبرنوری با ریت ارسال 40Gbps و با سرعت 200000km/s به ازای هر 1000 کیلومتر propagation timeمون حدود 5 میلی ثانیه خواهد بود

Tier-1 ISP: e.g., Sprint



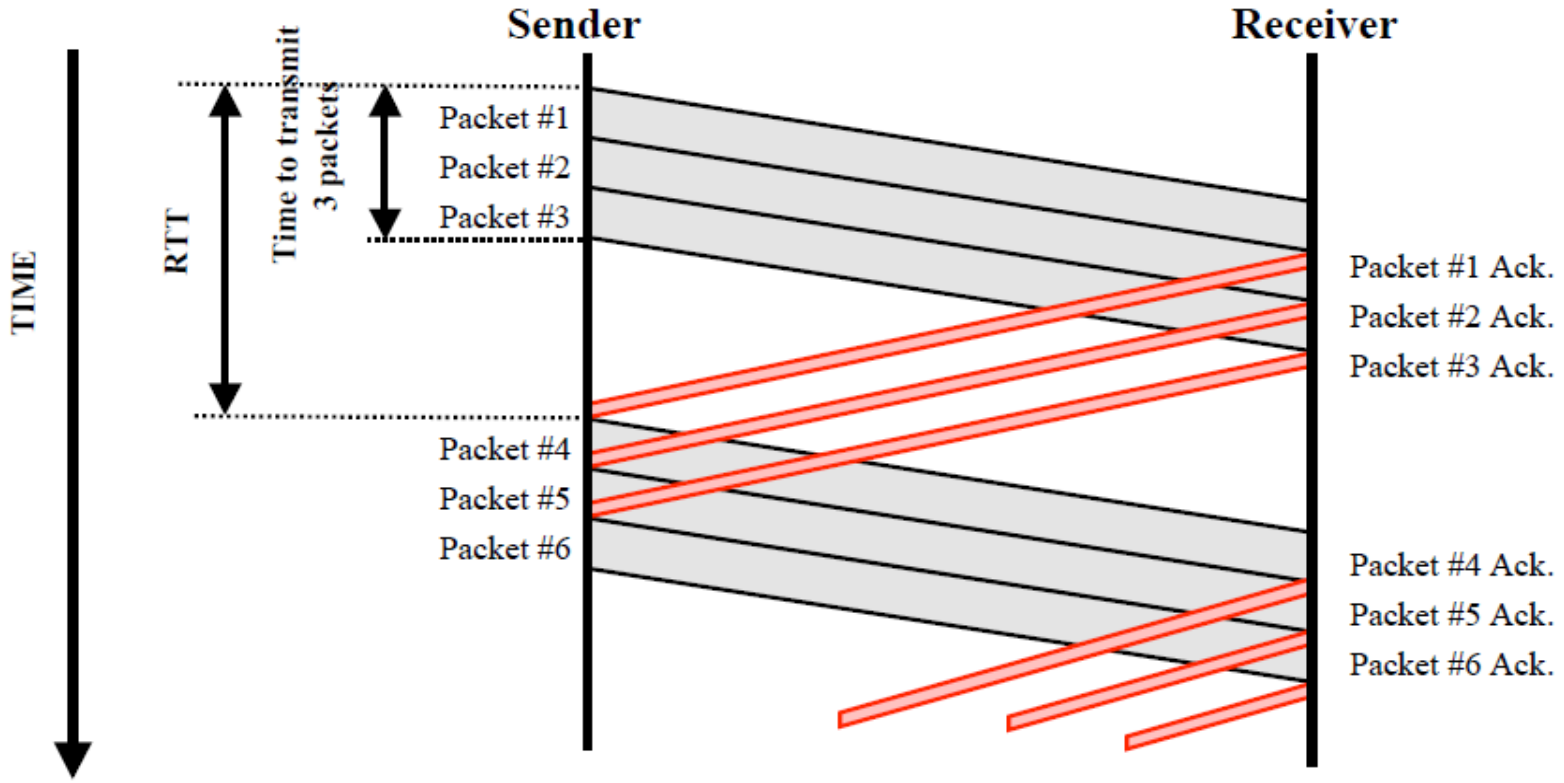
TCP throughput on a 200 Mbps link



نکته:
در لینکی که 200Mbps عرض باند اون است کانکشن tcp مثل این شکل تونسته از عرض باند استفاده بکنه ینی یه چیزی حدود 10Mbps
چرا اینجوی شده؟ این برمیگیرده به ماهیت tcp چون Tcp به شکلی که ابتدا طراحی شده نمی تونه عرض باند های بالا رو پر بکنه
مشکل کجاست؟
صفحه بعدی..

Problem: window size

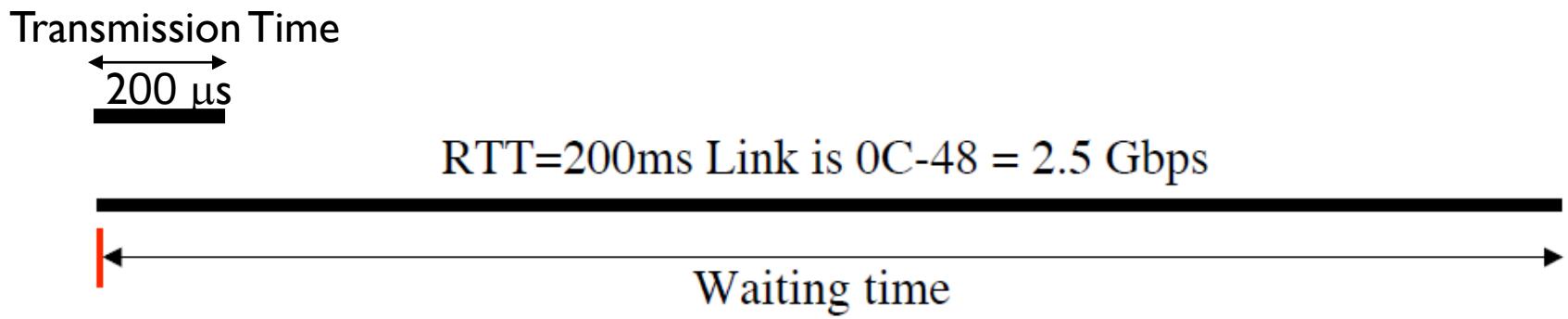
- The default maximum window size is 64Kbytes.
- Then the sender has to wait for acks.



-
مشکل window size است
گفتیم که throughput کانکشن tcp ارتباط داره با اندازه window size و RTT
مقدار پیش فرض اندازه حداکثر پنجره در Tcp هست 64Kbyte به این ترتیب RTT زیاد باشه که در لینک های طولانی زیاد خواهد بود و ریت ارسال هم اگه بالا باشه حتی اگر پنجره اندازه اش ماکزیمم باشه ما در زمان کوچیکی ارسال همه بسته ها رو می تونیم تموم بکنیم ولی بعد باید منظر بازگشت اک ها باشیم و این ممکنه طولانی باشه --> برای لینک های طولانی این زمان ممکنه زیاد باشه و در نتیجه ما به ازای یک پنجره کامل در یک زمان کوچیکی همه اطلاعات رو می فرستیم و بعد در یک زمان طولانی منتظر میشیم تا اک ها برگردن تا بتونیم ارسال رو ادامه بدیم

Problem: window size

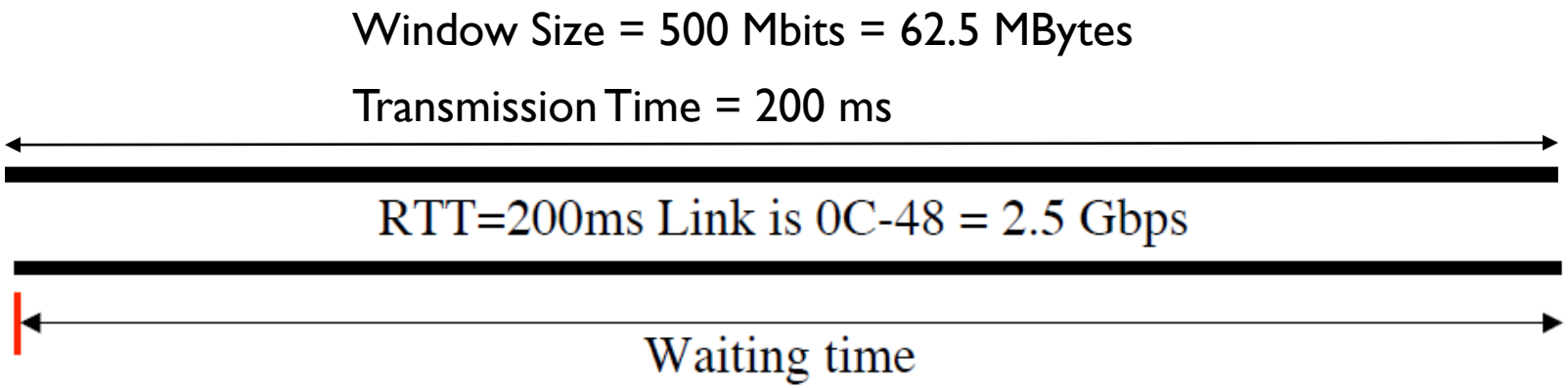
- Less than 0.1% of the link bandwidth is utilized
- A big file transfer takes minutes



-
برای مثال در این لینک با سرعت 2.5Gbps این waiting time می تونه برای فاصله طولانی اگر فرض کنیم RTT=200ms باشه ما 64Kbyte رو ما در 25 میکروثانیه می تونیم ارسال بکنیم که این 0.01 درصد عرض باند شبکه رو utilized می کنه و با یک چنین سرعتی ارسال یک فایل بزرگ میتونه چندین دقیقه طول بکشه علارغم عرض باند بالای لینکمون

Problem: window size

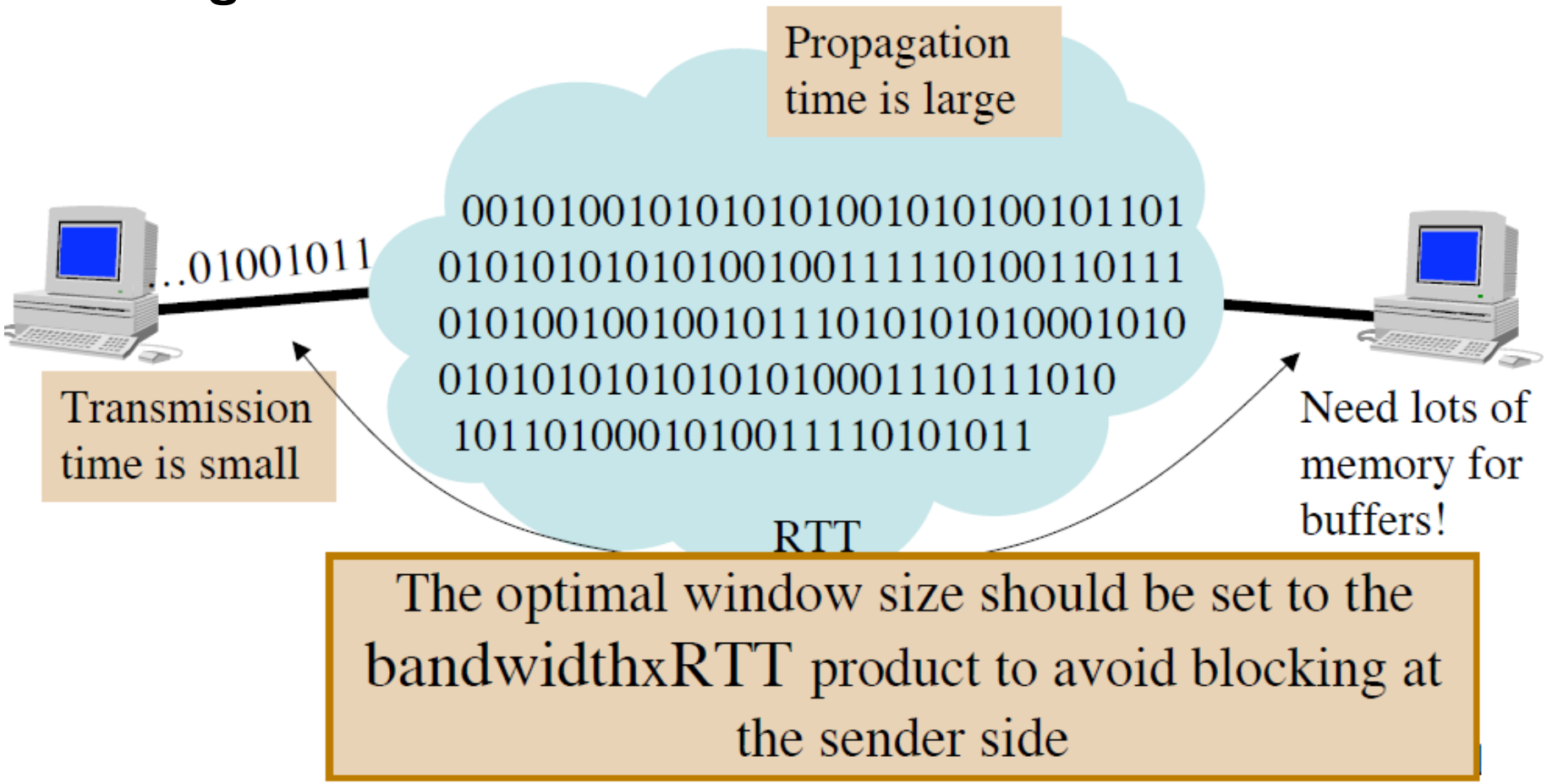
- Much larger window size to utilize the link BW
- To keep sending till the first ACK
  - W = RTTx R bits



-
راه حل چیست؟
می تونه این باشه که ما اندازه پنجره ارسال رو بیشتر بکنیم و اجازه بدیم پنجره ارسال خیلی بیشتری استفاده بشه در tcp
اگر بخوایم برای یک لینک با سرعت Rbit/sec در کل زمان RTT ینی در کل زمانی که اولین بسته می ره و اکش برمیگرده ارسال رو ادامه بدیم اندازه پنجره باید RTT*R باشه
توی مثال قبلی این مقدار میشه 62.5Mbyte

High capacity network

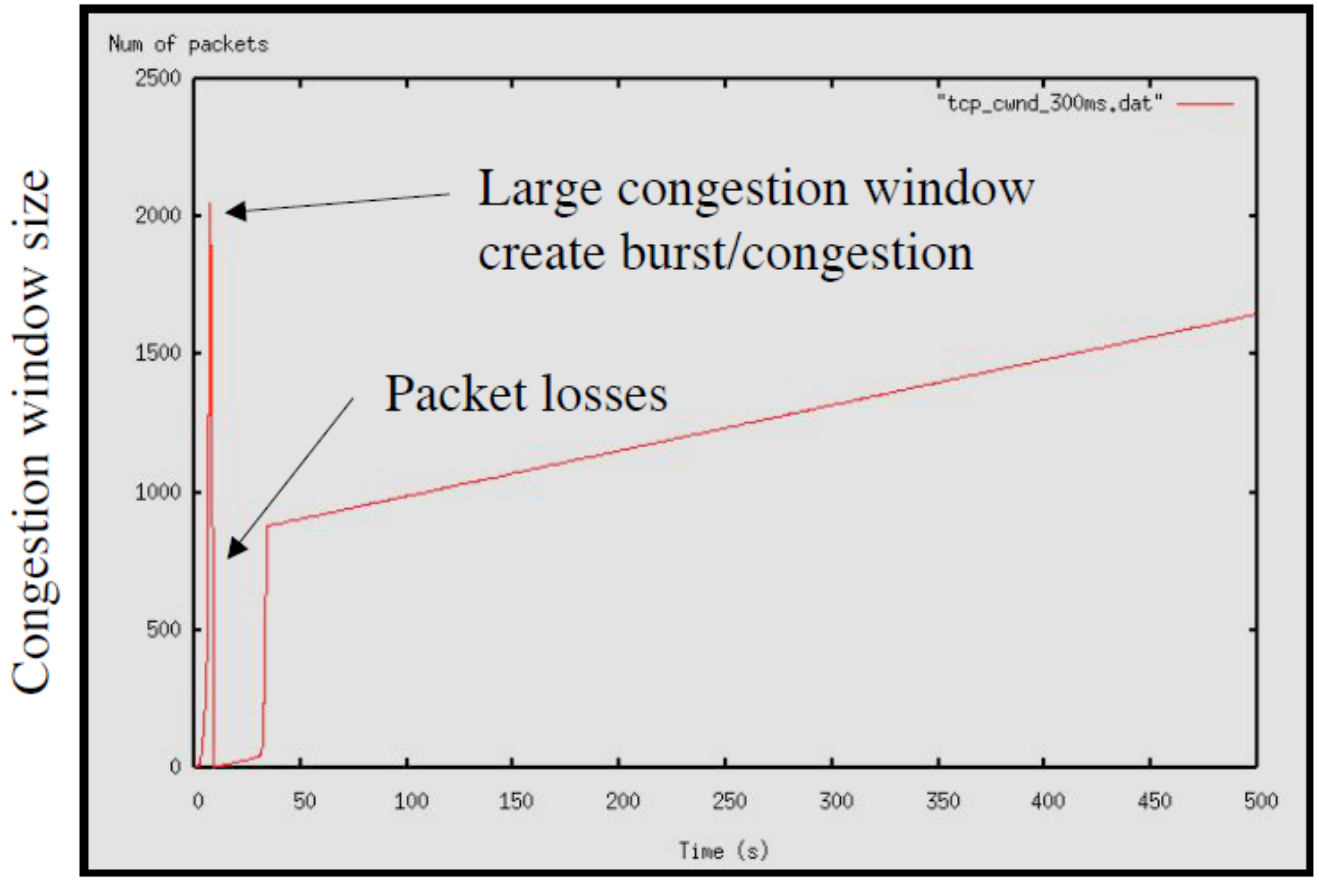
LFN: Long Fat Network



فرض میکنیم ما این کارو انجام دادیم:
در این صورت چه مسئله ای پیش میاد؟
مسئله این است که تمام بیت هایی که به این ترتیب ما پنجره ارسالمون فضای بیشتری به ما میده که بفرستیم و منتظر اک باشیم این بیت ها کجا هستن؟
این بیت ها در این زمان RTT توی لینک های بین راه در حال انتشار هستن تا به مقصد برسن و بعد اکشون برگرده به این ترتیب ما میگیم LFN ینی لینک هایی که تعداد زیادی بیت در حال عبور از اون ها هستن
مشکل چه زمانی پیش میاد؟
سمت گیرنده ما باید حجم بافر زیاد داشته باشه که بتونه این حجم این اطلاعات رو بگیره و ذخیره بکنه برای این پروتکل sliding window که در سمت گیرنده بخاطر جابه جا شدن بسته ها شاید مجبور باشه اطلاعات رو نگه داره تا بتونه با ترتیب درست این ها رو به اپلیکیشن بده پس یک بافر بزرگ هم اینجا نیاز داریم که البته مسله خاصی نیست
ولی مسئله اصلی این است که اگر خطا اتفاق بیوفته یا پکت لاس اتفاق بیوفته چی میشه؟
صفحه بعدی..

Side effect of large windows

- TCP becomes very sensitive to packet losses in LFN



-
اگر یک پکت لاس اتفاق بیوفته ما پنجره ارسال رو می بندیم و از اول شروع میکنیم و به تدریج اینو افزایش میدیم و این زمانی طول می کشه تا اندازه پنجرمون بیاد به اندازه کافی باز بشه و توی این فاصله ما دوباره داریم عرض باند لینک رو از دست میدیم
پس به عبارت دیگر هر پکت لاس یا هر عاملی که باعث پکت لاس بشه میتونه باعث ؟؟ بشه

High capacity networks

- Sustaining high congestion windows:

A Standard TCP connection with:

– 1500-byte packets;

– a 100 ms round-trip time;

– a steady-state throughput of 10 Gbps;

would require:

– an average congestion window of 83,333 segments;

– and at most one drop (or mark) every 5,000,000,000 packets (or equivalently, at most one drop every 1 2/3 hours).

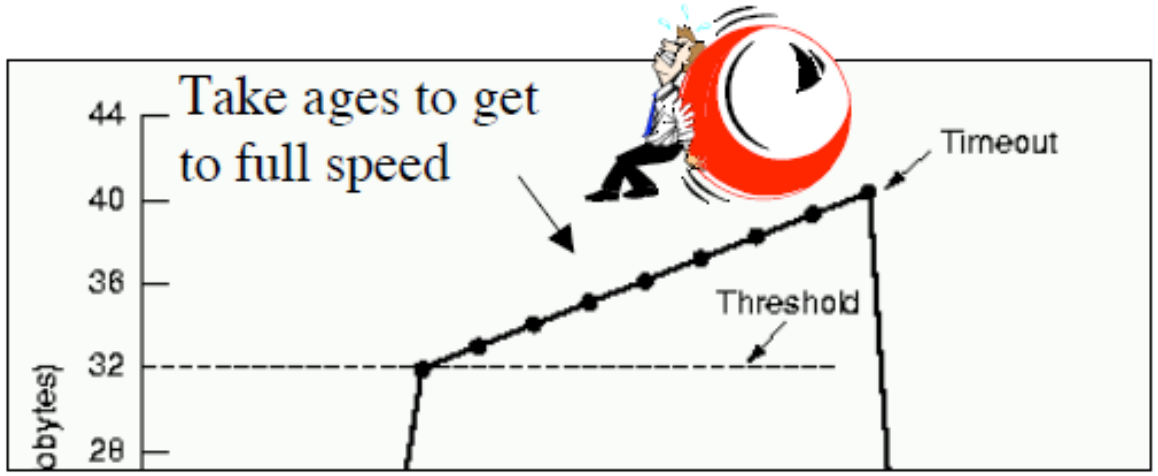
This is not realistic.

From S. Floyd

به این ترتیب برای مثال:
اگر لینکمون 10Gbps باشه و فاصلمون به گونه باشه که RTT=100ms باشه و اندازه ماکزیمم پکت هم 1500 بایت هست
در این صورت اندازه congestion window مون به صورت متوسط باید 83333 سگمنت باشه در این صورت برای اینکه ما utilized اش لینک خیلی پایین نیاد حداکثر یک پکت لاس در هر 5 میلیون پکت باید اتفاق بیوفته ینی با لینک 10Gpbs و پکت 1500 بایت حدودا هر 100 دقیقه یک پکت لاس بیشتر نباید اتفاق بیوفته و اگر ریت پکت لاس بیشتر از این باشه utilizetion لینکمون پایین خواهد امد و throughput مفیدمون خیلی پایین خواهد امد

Additive increase is still too slow

- With 100ms of round trip time, a connection needs 203 minutes (3h23) to get 1Gbps starting from 1Mbps



-
چرا این اتفاق میافته؟
چون در پروتکل congestion tcp این Additive Increase خیلی طول می کشه و خیلی کند داره افزایش پیدا میکنه
به عنوان مثال اگر ما از یک 1Mbps شروع بکنیم تا برسیم به 1Gbps اگر RTT=100ms باشه این حدود 3 ساعت طول خواهد کشید
این مشکلی است که امروزه وجود داره و tcp به شکل اولیه اش در شبکه ها امروزی کارایی خیلی خوبی نداره
چاره این کار چی هست؟
صفحه بعدی..

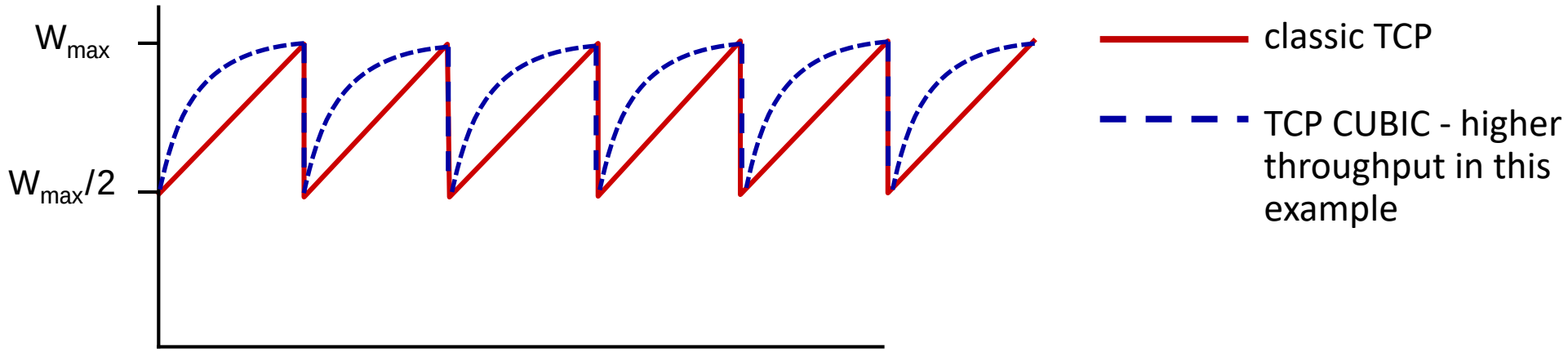
Solution?

- Several parallel TCP connections
  - Requires less configuration in the standard TCP
- New TCP Protocols for high-speed links
  - Research
    - Fast TCP
  - OS:
    - Cubic

-
میتونه استفاده از کانکشن های Tcp موازی باشه در این صورت ما علارغم این که یک کانکشن خیلی نمتونه عرض باند رو utilized بکنه و اگر پکت لاسی اتفاق بیوفته یک مقدار طول می کشه تا بیاد throughput اش افزایش پیدا بکنه ولی اگر ما کانکشن های موازی متعددی ایجاد بکنیم برای همون ارتباط:
اولا وقتی که پکت لاس اتفاق بیوفته یکی از این ها کانکشن ها تحت تاثیر قرار میگیره نه همشون مگر اینکه congestion شدیدی اتفاق بیوفته
دوما throughput ماکزیمم که باید بهش برسه خیلی بالا نیست و این در زمان کوتاه تری به اون می رسه در مقایسه با حالتی که همه عرض باند توسط یک کانکشن قرار باشه که پر بشه
راه حل بعدی اون:
اصلاحات پروتکل tcp است

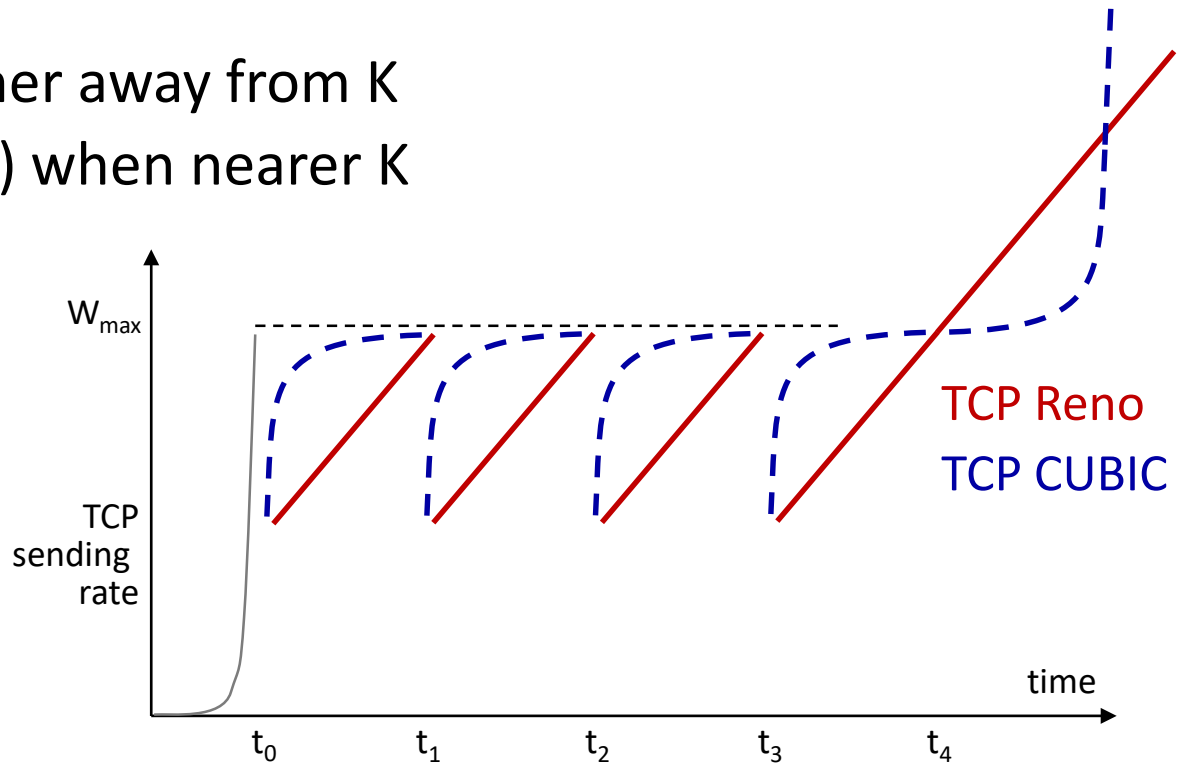
TCP CUBIC

- Is there a better way than AIMD to "probe" for usable bandwidth?
- Insight/intuition:
  - W_{max}: sending rate at which congestion loss was detected
  - congestion state of bottleneck link probably (?) hasn't changed much
  - after cutting rate/window in half on loss, initially ramp to to W_{max} *faster*, but then approach W_{max} more *slowly*



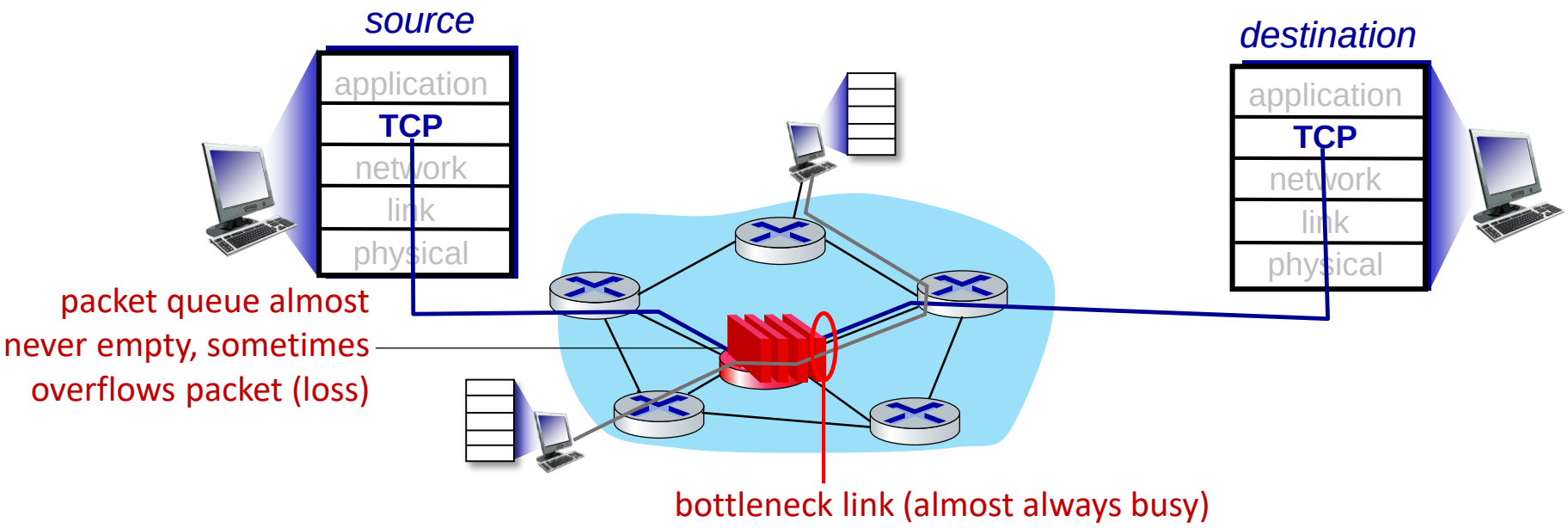
TCP CUBIC

- K: point in time when TCP window size will reach W_{max}
  - K itself is tuneable
- increase W as a function of the *cube* of the distance between current time and K
  - larger increases when further away from K
  - smaller increases (cautious) when nearer K
- TCP CUBIC default in Linux, most popular TCP for popular Web servers



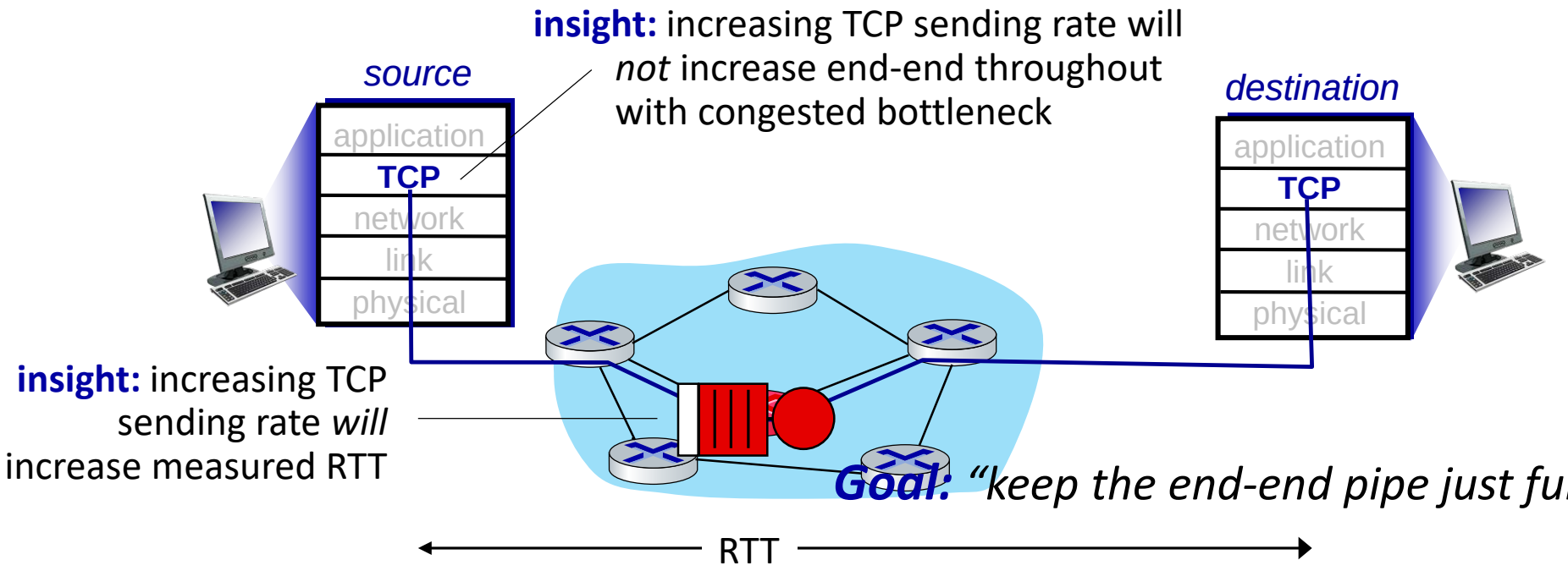
TCP and the congested "bottleneck link"

- TCP (classic, CUBIC) increase TCP's sending rate until packet loss occurs at some router's output: the *bottleneck link*



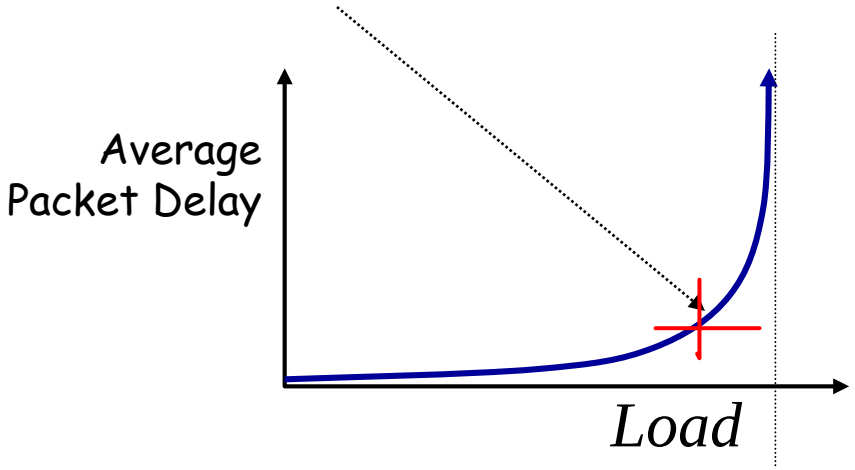
TCP and the congested "bottleneck link"

- TCP (classic, CUBIC) increase TCP's sending rate until packet loss occurs at some router's output: the *bottleneck link*
- understanding congestion: useful to focus on congested bottleneck link



Congestion Avoidance

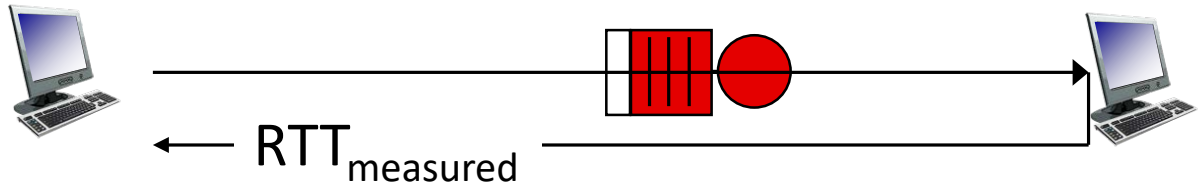
- TCP reacts to congestion *after* it takes place. The data rate changes rapidly and the system is barely stable (or is even unstable).
- Can we *predict* when congestion is about to happen and avoid it? E.g. by detecting the knee of the curve.



-
منحنی تغییرات تاخیر متوسط پکت در سطح شبکه: قبلا گفتیم که یک چنین رابطه ای با لود شبکه داره ینی با افزایش لود تاخیر یک مقداری یک افزایش پیدا می کنه ولی مقدارش خیلی کم است و از یک نقطه ی به بعد دیدیم اگر لود یک مقدار کمی افزایش پیدا بکنه تاخیر یک مقدار زیادی افزایش پیدا خواهد کرد و این نقطه + نقطه ای است که ما به عنوان congestion ازش یاد میکنیم ینی وقتی که از نقطه + رد شدیم وارد مرحله congestion شدیم و در این مرحله اضافه شدن مقدار کمی لود میتونه باعث ناپایدار شدن صفحه ها بشه در شبکه و تاخیرها رو تا حد زیادی افزایش بده
راه حلهایی که بررسی کردیم تا الان راه حل هایی هستند که از نقطه + به بعد وارد عمل میشن ینی وقتی که ما به مرحله ای رسیدیم که پکت لاس شروع شده و ابتدا تک و توک بسته ها گم میشن پس در نتیجه ما ابتدا Duplicate Ack داریم و سعی میکنیم با پایین اوردن لود ینی با پایین اوردن ریت ترافیک ورودی به شبکه که لود رو کم میکنه سعی میکنه وضعیت رو از نقطه + بیاریم پایین تر و در حالت پایدارتری قراربدیم و اگر این وضعیت بدتر بشه تایم اوت اتفاق می افته و سعی می کنیم حد خیلی بیشتری لود رو کم بکنیم تا بتونیم از این وضعیت خارج بشیم ولی در هر صورت چه در حالت Duplicate Ack و چه تایم اوت به هرحال congestion اتفاق افتاده حالا چه در مرحله ابتدایی یا چه در مرحله بعدتر و این congestion به هرحال برای شبکه خوب نیست مثلا تاخیرها بیشترمیشه
سوال این است که ما ایا میتونیم قبل ازاینکه به این مرحله برسیم کاری بکنیم؟ ینی وقتی که لودمون داره از نقطه + رد میشه و میخواد بره به سمتی که congestion اتفاق بیوفته ما اینو تشخیص بدیم و قبل از اینکه Congestion اتفاق بیوفته لود رو کم بکنیم و وضعیت رو تعدیل بکنیم:حالا میخوایم روش هایی روبگیم که از congestion جلوگیری بکنیم ینی Congestion Avoidance
روش هایی Congestion Avoidance:
DECbit
RED
host-based Congestion Avoidance

Delay-based TCP congestion control

Keeping sender-to-receiver pipe "just full enough, but no fuller": keep bottleneck link busy transmitting, but avoid high delays/buffering



$$\text{measured throughput} = \frac{\text{\# bytes sent in last RTT interval}}{RTT_{measured}}$$

Delay-based approach:

- RTT_{min} - minimum observed RTT (uncongested path)
- uncongested throughput with congestion window cwnd is cwnd/RTT_{min}

if measured throughput "very close" to uncongested throughput
    increase cwnd linearly /* since path not congested */
else if measured throughput "far below" uncongested throughout
    decrease cwnd linearly /* since path is congested */

-
tcp vegas: هدف از این روش این است که قبل از اینکه Congestion شروع بشه نزدیک شدن به مرحله Congestion رو تشخیص بده و این کار رو Tcp vegas با مانیتور کردن RTT انجام میده
یکی از پارامترهایی که با افزایش طول صف ها در شبکه و افزایش لود شبکه اتفاق میافته افزایش RTT است و Tcp vegas این رو مانیتور میکنه و هر موقع تشخیص داد که RTT داره افزایش پیدا میکنه و به مرحله Congestion داره نزدیک میشه این میاد ریت ورود به ترافیک شبکه رو کم میکنه که این راهش این است است که Congestion window رو کم بکنه که ترافیک کانکشن رو کمتر بکنه اگر لود شبکه زیاد بشه و تعداد بسته هایی که به روتر می رسن بیشتر باشه در این صورت در پورت خروجی بافر پر میشه و صف تشکیل میشه و در نهایت این به مرحله ای می رسه که بافر تقریبا پر میشه و بسته ها شروع میشن به drop شدن و اون مرحله ای است که Congestion اتفاق می افته ولی قبل از اینکه به اینجا برسه طول صفمون ما همین جا چیزی که میتونیم مشاهده بکنیم این هست که بسته هایی که اینجا قرار میگیرن ینی در انتهای صف و تا بیان برسن به سر صف و ارسال بشن به لینک، تاخیر معادل زمان ارسال تک تک این بسته ها که جلوی صف قرار دارن رو تحمل می کنن به عبارت دیگر تاخیر بسته افزایش پیدا میکنه و اگر این تاخیر به ازای چند روتر پشت سر هم در مسیر افزاش پیدا بکنه جمع تاخیر از مبدا به مقصد و همینطور توی مسیر برگشت اک از مقصد به مبدا و به عبارت دیگر اون چیزی که داره افزایش پیدا میکنه RTT است --> RTT داره مانیتورمیشه و هرکجا احساس کرد که RTT از یه حدی بیشتر شده دیگه معطل نمیشه تا مرحله به پکت لاس برسه از همین جا شروع میکنه Congestion windowرو پایین میاره پس RTT بیشتر ینی نزدیک شدن به مرحله Congestion توی شبکه

Delay-based TCP congestion control

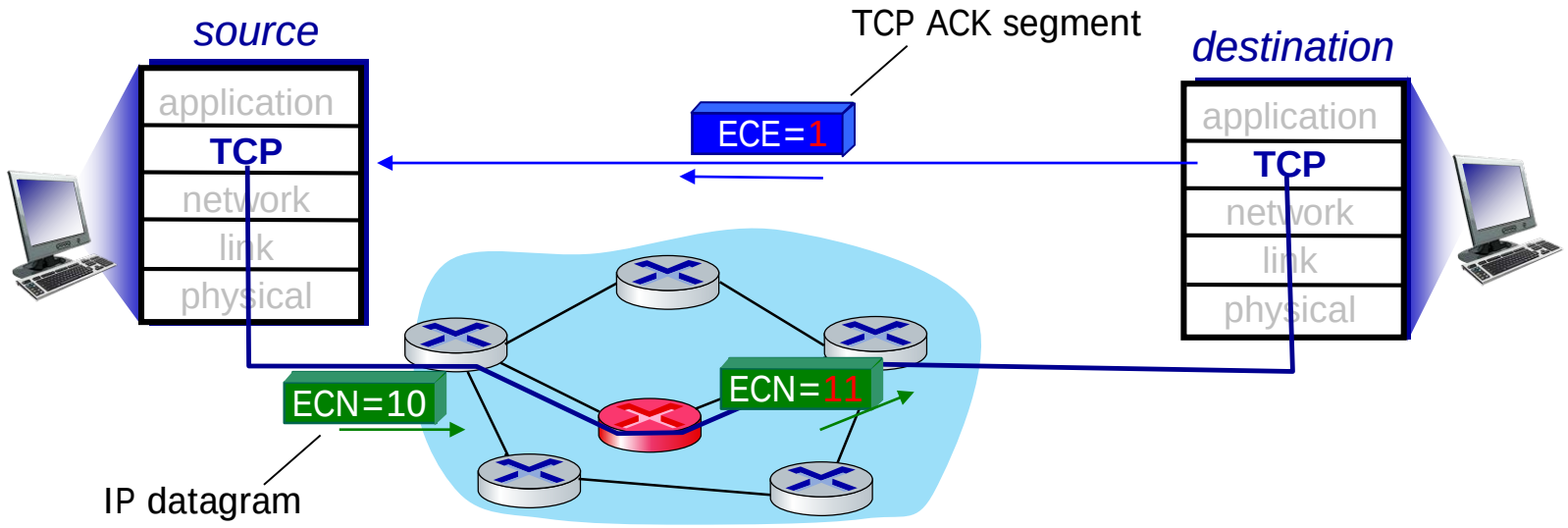
- congestion control without inducing/forcing loss
- maximizing throughout ("keeping the just pipe full… ") while keeping delay low ("…but not fuller")
- a number of deployed TCPs take a delay-based approach
  - BBR deployed on Google's (internal) backbone network

    (Bottleneck Bandwidth and Round-trip propagation time)

Explicit congestion notification (ECN)

TCP deployments often implement *network-assisted* congestion control:

- two bits in IP header (ToS field) marked *by network router* to indicate congestion
  - *policy* to determine marking chosen by network operator
- congestion indication carried to destination
- destination sets ECE bit on ACK segment to notify sender of congestion
- involves both IP (IP header ECN bit marking) and TCP (TCP header C,E bit marking)



-
روش دیگه ای که برای Congestion Avoidance مطرح شده استفاده از ECN است که این روش مبتنی بر این است که روتر توی مسیر هم کمک میکنه به تشخیص وقوع یا نزدیک شدن به وقوع Congestion در این روش در هدر بسته ip دو بیت استفاده میشه که پیشنهاد میشه از بیت های tos field استفاده بشه
در این روش روتر چجوری تشیخص میده که Congestion اتفاق افتاده؟
روتر به راحتی می تونه اندازه طول صف پورت رو مانیتور بکنه و یک threshold براش ست میشه و اگر از این threshold بیشتر شد طول صف این رو تشخیص میده به عنوان اینکه congestion میخواد اتفاق بیوفته
در این روش افزایش طول صف رو روتر مستقیما اطلاع میده از طریق مقصد پس بسته ای که ارسال میشه بیت ECN اش صفر است این بسته می رسه به روتری که در مرحله congestion قرار داره و روتر بیت ها رو ست میکنه و این بسته به مقصد می رسه و مقصد توی اک اون که برمیگردونه بیت ECN رو یک میکنه و این بیت رو مبدا تشخیص میده و براساس اون ریت ترافیک ارسالی به شبکه رو کاهش میده

Random Early Detection (RED)

- RED is similar to DECbit, and was designed to work well with TCP.
- RED implicitly notifies sender by dropping packets.
- Drop probability is increased as the *average* queue length increases.
- (Geometric) moving average of the queue length is used so as to detect long term congestion, yet allow short term bursts to arrive.

ادامه پایینی: این روش متوسط گیری باعث میشه که تغییرات لحظه ای و آنی طول صف منجر به کاهش شبکه نشه --> اگر واقعا طول صف به طور پایداری در حال افزایش است و توی نمونه های متعددی اضافه شده در این صورت ما مکانیزیم رو به جریان می ندازیم وگرنه اگر یک لحظه طول صف افزایش پیدا کرد و بعد کاهش پیدا کرد و پایین مونده در این صورت ترافیک رو کاهش نمیدیم

$$AvgLen_{n+1} = (1-\alpha) \times AvgLen_n + \alpha \times Length_n$$

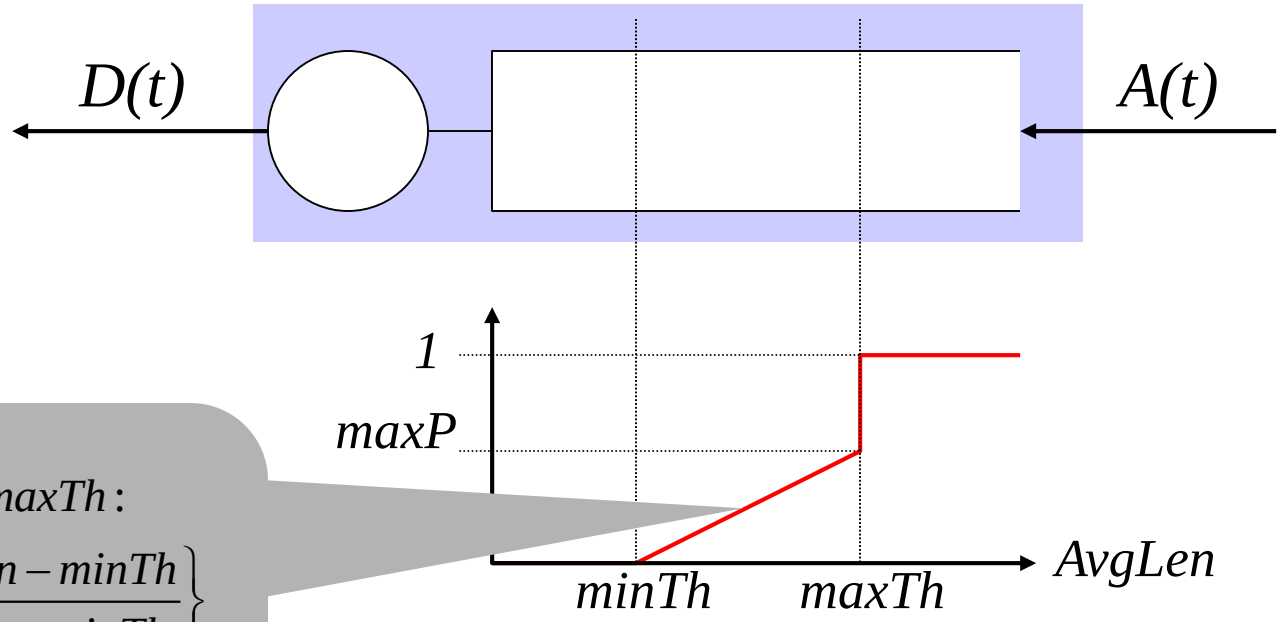
$$\text{i.e. } AvgLen_{n+1} = \sum_{i=1}^{n} Length_i (\alpha)(1-\alpha)^{n-i}$$

فرمول: روتر طول متوسط صف پورت خروجی رو مانتیور میکنه و براساس اون تصمیم میگیره برای drop کردن بسته ها و این طول متوسط به این صورت حساب میشه و طول متوسط یک متوسط گیری زمانی است و به این صورت است که: فرض میکنیم در یک مرحله تمام نمونه های قبلی اندازه گیری شده و متوسط به دست اومده حالا یک پکت جدید میاد و طول صف رو مجددا اندازه گیری میکنیم حالا به این ترتیب ما یک Avglen از قبل داریم و یک length به ازای پکت جدیدی که توی صف قرار گرفته به دست میاد و متوسط این دو تا رو با این ضرایب به دست میاریم 1 منهای الفا = طول متوسطی که تا قبل داشتیم و الفا در طول جدید (الفا چیزی بین صفر و یک است) و به این ترتیب متوسط جدید به دست میاد --> این عملا یک فیلتر پایین گذر است که اعمال میشه روی طول صف

به صورت زیرش می تونیم بنویسیم که بسته به این که الفا چقدر باشه میزان تاثیر نمونه جدید روی متوسط نمونه های قبلی رو تعیی میکنه اگر الفا عدد بزرگی باشه این نمونه میتونه کل متوسط رو تحت تاثیر قرار بده و اگر کوچیک باشه یک نمونه جدیدی روی متوسط دراز مدتمون تاثیر گذار نیست

روش DECbit: در این روش هر پکت یک بیت در هدر خودش داره به نام DECbit برای congestion notification
روتر ما طول صف پورت خروجی رو مانیتور میکنه و این طول رو دائم اندازه گیری میکنه و متوسط گیری میکنه و طول متوسط صف بررسی میکنه و اگر طول متوسط صف از یک مقداری بیشتر بشه در این صورت این رو به عنوان این که داره به یک congestion نزدیک میشه تشخیص میده و در اینصورت میاد و بیت DECbit رو ست میکنه توی بسته ای که داره ارسال میشه به پورت خروجی و این بسته به مقصد می رسه و مقصد این رو در بسته اک که برمیگردونه این بیت رو ست میکنه و مبدا از این وضعیت مطلع میشه و اندازه پنجره کاهش میده
در این روش الگوریتمی که برای مدیریت ترافیک ورودی به شبکه استفاده می شه براساس DECbit به این صورت که اگر این بیت در کمتر از 50 درصد اک هایی که از مقصد دریافت میشن ست میشن این رو براین برداشت میکنه که وضعیت صف خیلی بد نیست و می تونه ترافیک بیشتری رو تحمل بکنه برای همین میاد و به صورت خطی و به ارومی ترافیک ورودی رو افزایش میده و این کارو میتونه با افزایش اندازه پنجره انجام بده و اگر در بیش از 50 درصد اک هایی که دریافت می کنه ست شده باشه این رو به عنوان اینکه congestion داره اتفاق میافته تشخیص میده و ریت رو به صورت ضرب شونده کاهش میده
این صفحه:
روش دیگه ای که برای Congestion Avoidance در شبکه پیشنهاد شده استفاده از RED است که روش کارش خیلی شبیه DECbit است ینی روتر نزدیک شدن وضعیت congestion رو تشخیص میده و این رو به مبدا خبر میده و مبدا براساس اون میاد لود ورودی به شبکه رو کنترل میکنه
منتها تفاوتی که وجود داره این است که در روش DECbit روتر مستقیما یک بیت رو ست میکرد و به این ترتیب به مقصد و از اون طریق اون به مبدا خبر میداد که congestion داره اتفاق می افته ولی در روش RED روتر این کارو به صورت غیرمستقیم انجام میده --> چگونه می تونه روتر به مبدا خبر بده که به congestion داره نزدیک میشه؟ این کار رو با drop کردن یک بسته انجام میده --> هر موقع متوسط طول صف از یک اندازه ای بیشتر شد میاد و یک بسته رو دستی drop میکنه و چون بسته های بعدی به هرحال می رسن این باعث میشه در مبدا Duplicate Ack اتفاق بیوفته و در نتیجه مکانیزیم tcp congestion کنترل به کار می افته و اندازه پنجره رو میاد نصف میکنه پس به این ترتیب ما قبل از این که واقعا بسته ها شروع به drop بکنن یک بسته رو drop میکنیم
چرا اجازه ندیدم که این drop ها واقعی باشن و بعد مکانیزیم tcp کار خودشو انجام بده؟ یک مزیت هایی در اینکه ما پیشاپیش و زودتر drop میکنیم وجود داره ولی به عنوان یک نمونه در اینجا تعداد drop ها می تونه خیلی کمتر باشه ما با یک drop وضعیت رو کنترل میکنیم در حالی که اگر وضعیت صف به مرحله congestion نزدیک شده باشه dropهای خیلی بیشتری اتفاق می افته

RED Drop Probabilities



If $minTh < AvgLen < maxTh$:

$$\hat{p}_{AvgLen} = maxP\left\{\frac{AvgLen - minTh}{maxTh - minTh}\right\}$$

$$\Pr(\text{Drop Packet}) = \frac{\hat{p}_{AvgLen}}{1 - count \times \hat{p}_{AvgLen}}$$

count counts how long we've been in $minTh < AvgLen < maxTh$ since we last dropped a packet. i.e. drops are spaced out in time, reducing likelihood of re-entering slow-start.

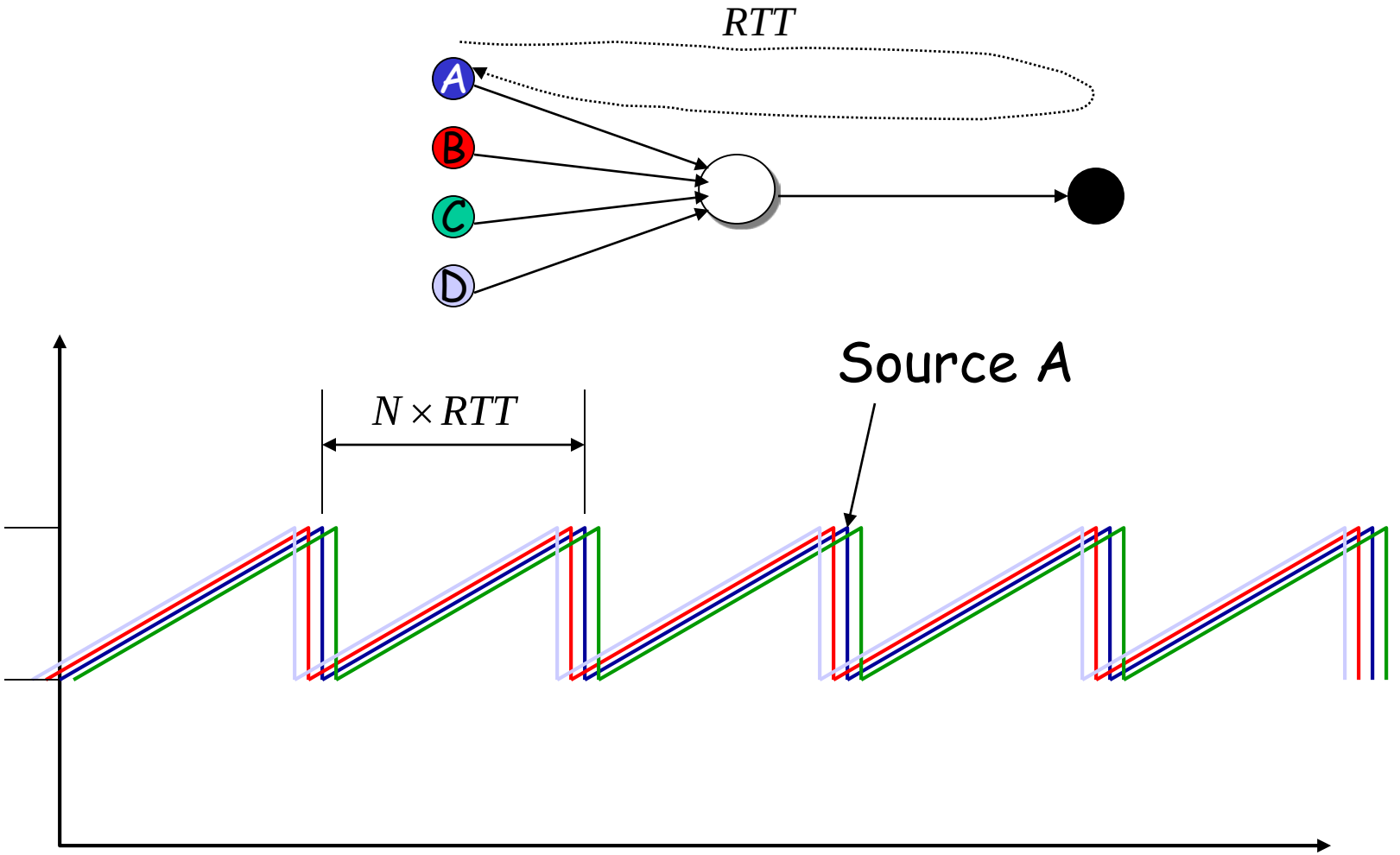
در RED ما اگر طول متوسط صف از یک threshold بیشتر شد بسته رو drop میکنیم این drop براساس یک احتمالی اتفاق می افته ینی با یک احتمالی بسته رو drop میکنیم ینی الزاما این نیست که بسته حتما drop بشه بلکه با یک احتمالی بسته drop میشه
این احتمال چگونه به دست میاد؟ این اگر بافر پورت خروجی باشه و ارسال بسته ها به خروجی توسط پورت انجام بگیره در شرایطی که تعداد بسته های ورودی به بافر بیشتر از تعدادی که سرویس می تونن داده بشن و می تونن ارسال بشن باشه بسته ها توی صف قرار میگیرن و صف تشکیل میشه
اندازه طول صف توی منحنی نشون داده شده --> متناسب با اندازه طول صف ما متوسط صف در این منحنی نشون میدیم و اگر طول متوسط صف از یک اندازه ای که بهش می گیم مینیمم threshold ینی minTH کمتر باشه هیچ drop انجام نمیشه و اگر بیشتر باشه با یک احتمالی انجام میشه و این احتمال از این منحنی به دست میاد و در این منحنی ما محور عمودی احتمال drop بسته است و اگر طول متوسط از minTh بیشتر بشه تا maxTh ما به صورت خطی احتمال drop رو افزایش میدیم و توی این مرحله هدف این است که یک بسته با یک احتمالی drop بشه و این باعث بشه که Duplicate Ack اتفاق بیوفته و منبع اون ترافیک و اون کانکشنی که براش این Drop اتفاق افتاده اندازه پنجره اش رو نصف بکنه و تا حدودی ترافیک ورودی رو کاهش بده ولی اگر اندازه متوسط طول صف از maxTh بیشتر شد احتمال drop میشه یک ینی دیگه ما اینجا هر بسته ای رو Drop می کنیم و این به معناست که داریم تلاش میکنیم در مبدا تایم اوت اتفاق بیوفته و ترافیک ورودی تا حد زیادی کاهش پیدا بکنه
فرمول:
این PavgLen که به دست میاد رو مستقیما برای drop استفاده نمیکنیم بلکه از Pr(drop packet) استفاده میکنیم
در اینجا count یک شمارنده ای است که این شمارنده هر موقع که drop اتفاق بیوفته صفر میشه و بعد به ازای هر پکتی که میاد توی صف قرار میگیره یکی اضافه میشه و همینطور اضافه میشه تا drop بعد و drop بعدی که اتفاق افتاد count صفر میشه و این count نشون دهنده چی هست؟
count نشان دهنده این است که از اخرین drop تا حالا چندتا پکت اومده اگر همین پکت قبلی drop شده بود coun مون یک است و در اینصورت اگر count یک باشه Pr به دست میاریم و هرچی count بیشتر بشه به این معناست که با احتمال بیشتری drop میکنیم و این به این معناست بسته هایی بلافاصله بعد از یک drop احتمال drop اشون کمتر باشه و هرچی بیشتر فاصله بیوفته احتمال dropشون بیشتر بشه و در راستای این است که اگر یک drop اتفاق افتاده ما فرصت بدیم بسته های بعدی drop نشن تا اون Duplicate Ack بتونه اتفاق بیوفته

Properties of RED

- Drops packets before queue is full, in the hope of reducing the rates of some flows.
- Drops packet for each flow *roughly* in proportion to its rate.
- Drops are spaced out in time.
- Because it uses average queue length, RED is tolerant of bursts.
- Random drops hopefully desynchronize TCP sources.

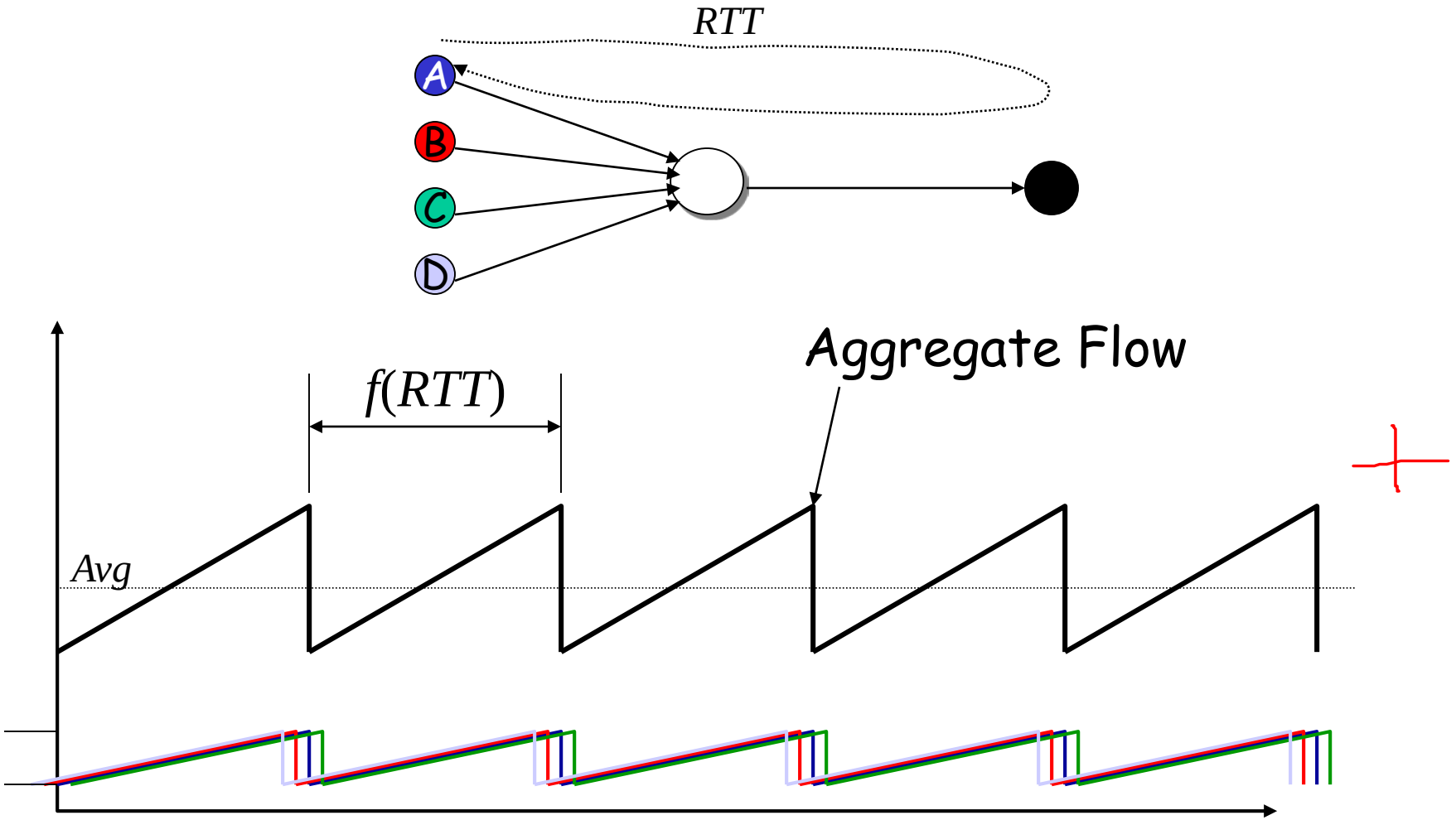
-
ویژگی های RED:
اساس کار این است که ما یک تعداد از پکت ها رو قبل از اینکه صف کاملا پر بشه drop میکنیم با این امید که یکسری از ترافیک هامون بخاطر مکانیزم congestion کنترل بیان و ترافیک ورودی رو کاهش بدن و صف وضعیتش بهتر بشه
ما اینو چون به صورت رندوم drop میکنیم ترافیک هایی که ریت بالاتری دارند بیشتر تحت تاثیر اون قرار میگیرن تا ترافیک های که ریت پایین تری دارند و با ریت پایین تری بسته می فرستن توی شبکه و این خود به خود باعث میشه که ترافیک هایی که نقش بیشتری در لود شبکه دارن زودتر تحت تاثیر قرار بگیرن و بیشتر در کاهش ترافیک ورودی به شبکه نقش داشته باشند و اون کاهش هم قابل توجه خواهد بود چون این ترافیکی هست که ریت بالاتری داره می فرسته پس کاهش اون بیشتر تاثیرگذار خواهد بود تا اونی که ریت پایینتری داره توی شبکه می فرسته
این dropها کاری می کنیم که با هم فاصله پیدا بکنن و این باعث میشه که اگر این drop ها مال یک کانکشن باشن فرصت Duplicate Ack پیدا بشه برای اون کانکشن و نره تایم اوت اتفاق بیوفته و دوما ترافیکش خیلی تحت تاثیر Drop های عمدی قرار نگیره
چون به هر حال ما طول متوسط صف رو داریم برمبناش عمل میکنیم و نه طول لحظه ای صف رو در نتیجه اگر bursts هایی اتفاق بیوفته برای ترافیک و به طور لحظه ای طول صف افزایش هایی داشته باشه این ها باعث نمیشن که ترافیک ورودی به شبکه کاهش پیدا بکنه این ها حالت های گذرایی هستن و به مفهوم این که congestion واقعا داره اتفاق می افته نیستن
drop های رندوم باعث میشه که کانکشن های tcp که روی این لینک برقرار هستن و به صورت مشترک دارن از عرض باند لینک استفاده می کنن این ها desynchronize بشن

Synchronization of sources

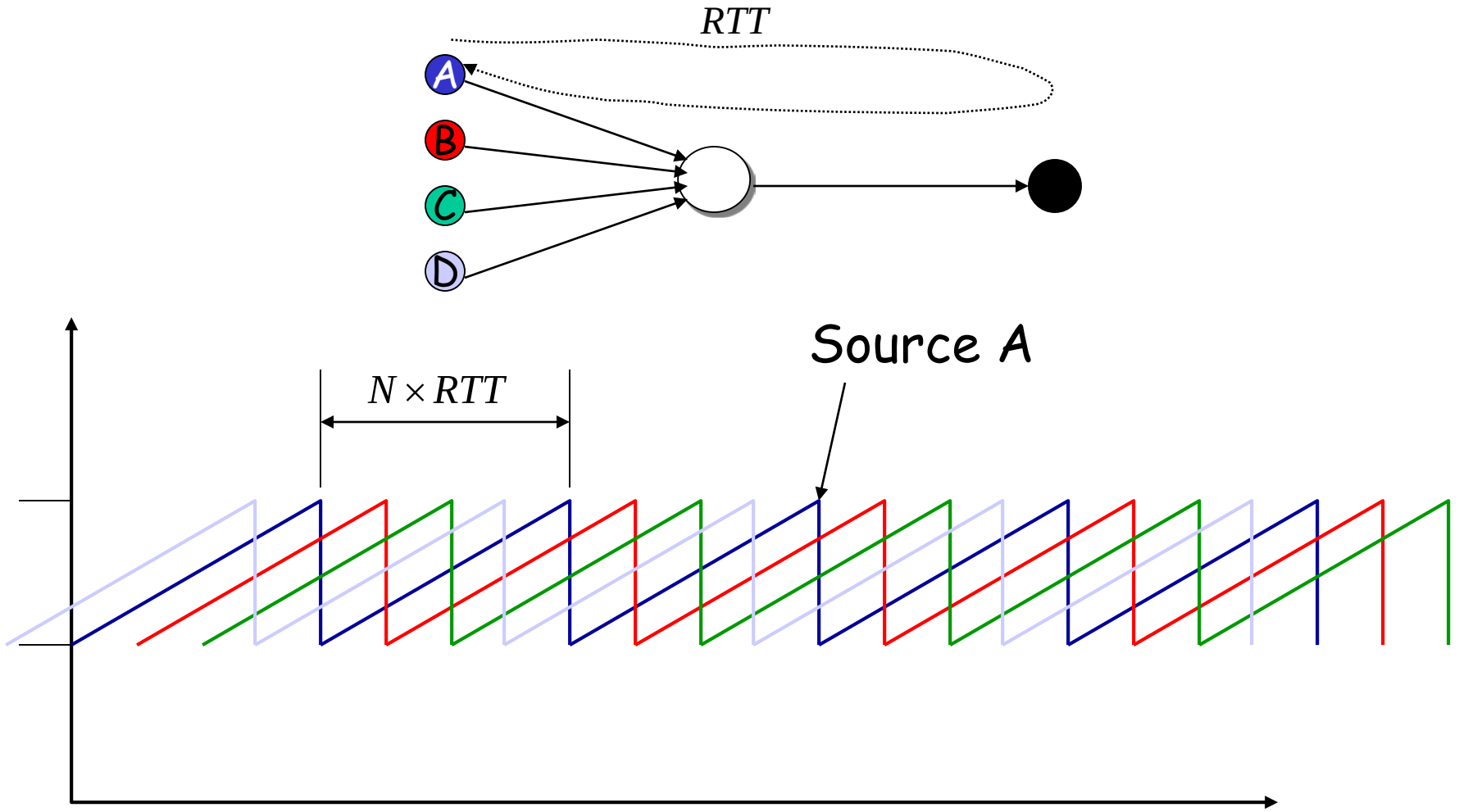


desynchronize چی هست؟
در حالتی که یک تعدادی کانکشن Tcp روی یک لینک به صورت مشترک قرار گرفته اند یک وضعیت پیش میاد به نام desynchronize منابع و اون این است که این حالت دندانه اره ای که در اندازه پنجره و در نتیجه در throughput این ها پیش میاد
این حالت دندانه اره ای برای همه این ها سینک میشه و با هم اتفاق می افته مثلا اینجا منبع سبز و قرمز و ابی ترافیکشون داره افزایش پیدا میکنه به صورت خطی و لینکمون می رسه به حالت congestion وقتی که congestion اتفاق افتاد drop از همه این ها اتفاق می افته و همه اینها اون رو تشخیص میدن و همه این ها براشون Duplicate Ack پیش میاد و اندازه پنجرشون رو نصف میکنن و متناسب با اون throughput ارسالیشون نصف میشه و دوباره به صورت خطی افزایش پیدا می کنند و این مرتب ادامه پیدا میکنه و پریود این تغییرات هم N*RTT ینی یک تعدادی RTT است و درواقع یک ثابت زمانی بزرگتر از RTT است که داره اتفاق می افته چون چند RTT طول میشکه که به مرحله پکت لاس برسیم و اندازه پنجره رو نصف بکنیم در این صورت اگر برایند throughput همه این کانکشن ها رو باهم به دست بیاریم منحنی صفحه بعدی میشه که این جمع لحظه ای ترافیک های کانکشن های روی این لینک هستن که اگر به این ترتیب تغییراتی داشته باشن جمع کل اون ها به صورت لحظه ای چنین تغییراتی خواهند داشت +
حالا اگر این سورس هارو Desynchronized بکنیم:
Desynchronized به این معناست که این وضعیت دندانه اره ای با هم سینک نباشن ینی اگر ترافیک ابی در اون لحظه Drop براش اتفاق می افته و اندازه پنجره اش رو نصف میکنه برای ترافیک قرمز این اتفاق نمی افته در اون لحظه به طورکلی همزمان نباشن --> این همزمان نبودن چه مزیتی داره؟ اگر اینها رو به صورت لحظه ای با هم جمع بکنیم این دفعه ترافیک به صورت * است به این ترتیت می بینیم که ترافیک لحظه ای با متوسطمون تقریبا برابر است و این می تونه برابر با عرض باند لینکمون باشه --> چه عاملی می تونه باعث بشه که Desynchronized اتفاق بیوفته؟ اون رندوم drop های RED می تونه باعث این وضعیت بشه چون به صورت رندوم بسته داره drop میشه ما هنوز به مرحله congestion واقعی نرسیدیم که همه شروع بکنن به Drop کردن و دستی داریم Drop میکنیم و این Drop ها از هم فاصله می گیرن مثلا یکی از این Dropها به ترافیک ابی تعلق پیدا میکنه و بعدی بعد از یه مدتی به قرمز تعلق پیدا میکنه و بعدی بعد از یه مدتی به سبز و به این ترتیب این Drop ها در زمان توزیع میشن و با یک احتمالی به کانکشن های مختلف اصابت میکنن و در نتیجه باعث میشن که اونها ترافیکشون رو کم بکنن و این باعث این Desynchronized میشه

Synchronization of sources



Desynchronized sources



Desynchronized sources