جلسه ۱۲ : TCP

TCP جایی هست که بحث های Connection control معنی پیدا می کند.

گرچه Connection Control در لایه Transport نیست ولی در لایه های دیگر هم بحث

می شود اما TCP بطور ویژه مکانیزم Connection control دارد.

Year. Month. Date. ( ) Subject :

## ARQ schemes (Ack protocals) :

داخل Tcp یک قابلیت و یک فرآیندی به نام acknowledgement وجود دارد

و یا acknowledge از ARQ یا از اتوماتیک فرآیند Automatic Repeat ReQuest تبعیت می‌کند. که ARQ برای کنترل کردن رخ داد خطا در سیستم است.

گر آنکه یک packet ای با مشکل به گیرنده رسید چه برخوردی با آن داشته باشد.

و راه حل های مختلفی ارائه می دهد.

۱) Error detection : چک می‌کند که آیا خطایی رخ داده است یا نه. و اگر خطایی رخ داده بود در ساده‌ترین حالت چیزی ارسال نمی‌کند.

۲) اگر بسته به درستی دریافت شده بود یک acknowledgment ساده به سمت فرستنده ارسال می‌کند

۳) فرستنده یک زمان timeout را صبر می‌کند تا ack برسد و اگر ack نرسید مجدداً آن را ارسال می‌کند.

* ۳ تا مسئله اصلی ARQ وجود دارد :

۱) Error detection

۲) Receiver Feedback ← همان acknowle- کردن است.

۳) Retransmission ← که فرستنده انجام می دهد.

* در Automatic Repeat ReQuest سه تا type وجود دارد :

۱) stop and wait ARQ : فرستنده packet را ارسال می‌کند و منتظر گیرنده می‌ماند ack را ارسال کند و در یک time out هم صبر می‌کند. (برای هر packet همین کار است)

Year: Month: Date. ( ) Subject :

از stop and wait کمتر استفاده می‌شود و هیچ در کنار آن مفهوم Pipelining وجود دارد. به جای اینکه منتظر بمونیم تا یکی کار انجام شود در کنار آن یکی دیگر کارهای دیگر انجام می‌شود.

در pipe یک سری window را مشخص می‌کنیم و براساس اندازه windowها packet ها را ارسال می‌کنیم.

① چالشی که در Pipe وجود دارد اینست که بحث flow control پیش می‌آید (بافر) که اگر پُر شد سر هم push شود over flow پیش می‌آید.

② ممکن است loss شود ③ بحث ack وجود دارد.

Go back N ARQ (pipeline) : یک پنجره به اندازه n در نظر می‌گیرد. این طوری است که packet ها در sliding window وجود دارد را ارسال می‌کند و صبر می‌کند تا ack ها برسند.

مثلاً تعداد ۰ و ۱ به ترتیب ارسال می‌شوند بعد ۲ ← loss می‌شود بعد ۳ و ۴ و ...

هم به ترتیب ارسال می‌شوند اما چون ۲ بعد از timeout ← ack ارسال شده دوباره ارسال‌ها را از ۲ شروع می‌کند. (حتی سالم‌ها را دوباره ارسال می‌کند)

TALASH

Year. Month. Date. ( ) Subject :

Selective Repeat ARQ : دقیقاً مثل Go back N بصورت (pipe line) sliding window ارسال می‌کند. ولی با این تفاوت که: بعد از time out اول packet ها که ack آن نیامده را ارسال می‌کند. و مثل قبلی تمام (حتی سالم‌ها) را دوباره ارسال نمی‌کند.

* TCP بیشتر Go Back N است. چون TCP مکانیزم acknowledment دارد

UDP خاصیت acknow ندارد چون اگر به چیزی خراب نشد کلاً دور می‌ریزد.

* ARQ هم در لایه Transport و هم در لایه datalink استفاده می‌شود. بالاتر از لایه فیزیکی ARQ استفاده می‌شود. یعنی ARQ که صفحه کرنگی است.

Hybrid ARQ (HARQ) : فقط retransmission را در نظر نمی‌گیرد و به هر دلیلی ack ارسال نشود (گم شدن، خراب شدن و --) آنگاه دوتا راه را انتخاب می‌کند: ① کدینگ انجام می‌دهد: HARQ Incremental Redundancy (IR) یکسری Redundancy (افزونگی) به بسته‌ها اضافه می‌کند که اگر مشکلی پیش آمد بتواند بجای اینکه ARQ ۲ انجام دهد error correcting (تصحیح خطا) انجام دهد. در اینجا فرستنده

Year. Month. Date. ( ) Subject :

با استفاده از بیت‌هایی که Redundancy بودند ← خطاها را تصحیح می‌کند.

② HARQ chase combining (CC) : packet های دریافتی را concatenate می‌کند و کنار هم می‌گذارد و کسری اطلاعات جمعی برنت می‌آورد و سعی می‌کند اول packet که خطا داشته را ← packet اصلی را بدون خطا بسازد.

خصوصیات Tcp : ① Point-to-point : یک فرستنده - یک گیرنده

② reliable, in-order byte steam : Tcp براساس شکست به شکست (دنباله‌ای) کار می‌کند. Tcp ← بایت به بایت کار می‌کند و هر بایتی شماره‌ای دارد. براساس byte steam

③ Full duplex data : هردو فرستنده و گیرنده دخیل هستند و هردو برای یکدیگر دیتا می‌فرستند. (ارتباط دوطرفه).

و maximum segment size (Mss) هم برای Tcp تعریف می‌شود.

که براساس اینکه linklayer چقدر می‌تواند پشتیبانی کند Mss تعیین می‌شود.

TALASH

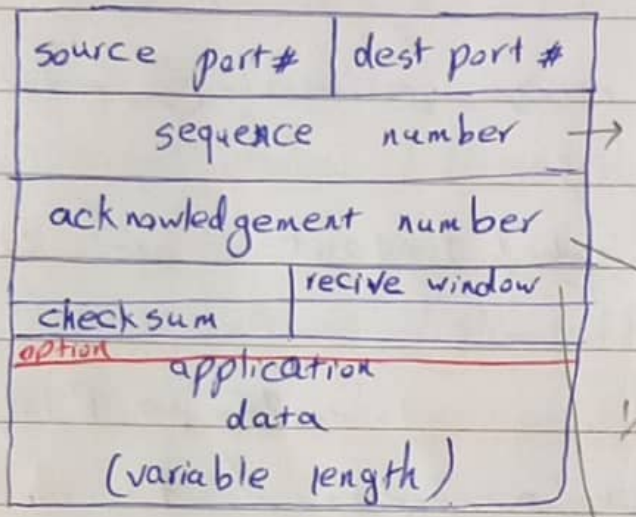
Year. Month. Date. ( ) Subject :

④ Cumulative Acks (جمع شونده): در Go back N استفاده می‌شود.

⑤ pipelining ⑥ connection-oriented: بحث handshaking

⑦ flow controlled دارد.

## Tcp segment structure:

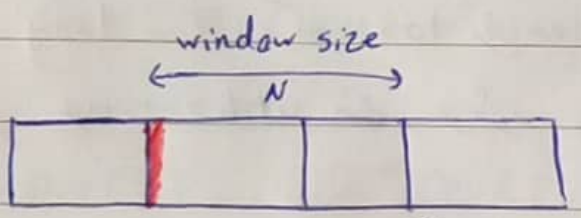



توی Tcp ما بایت‌ها را شماره‌گذاری می‌کنیم.

seq # به شروع بایت‌های اولی که قرار داده شده است.

شماره ارسال مجدد را اعلام می‌کند یعنی قبلی‌ها رو گرفته‌ام از این شماره به بعد را ارسال کن.

تحمل اندازه بافر




**Tcp sequence number:**

window دارای سایز N هست که به صورت pipe اجرا می‌شود

بسری از packet ها ارسال شده‌اند ولی هنوز ack آنها نیامده در window نگه داری می‌شوند.

بسری از packet ها هم هستند که در این window هستند اما هنوز ارسال نشده‌اند.

seq # شماره اولین بایت ❗ را مشخص می‌کند.

**Acknowkdgement number:** شماره seq # بایتی که انتظار دارد طرف مقابل برایش ارسال کند. و برای هر packet یکی یکی ارسال نمی‌کند بلکه یک ack می‌فرستد و می‌گوید که تا قبل از این همه packet ها را دریافت کرده‌ام.

Year. Month. Date. ( ) Subject :

یعنی cumulative ack هست .

و معمولاً به چیزی شبیه Go back N استفاده می‌شد.

Q : ممکن است packet ها به ترتیب نرسند و out-of-order پیش بیاد

که out-of-order در Tcp در استاندارد مشخص نشده و بسته به کسی که Tcp را

در سیستمش پیاده سازی می‌کند خودش باید تصمیم بگیره .

Q : چگونه timeout را تنظیم کنیم ؟ به پنجره زمانی گفتیم و بعد از یه مدتی اصلاحش کنیم

RTT هم خوبه .

too short : که به دردی نمی‌خورد همش باید retransmission داشته باشیم .

too long ، اگر loss رخ بده دیر جوابگو میشه (مشکل)

و utilization خراب می‌شود . و مشکل flow control هم پیش میاد .

می‌توان sample RTT بسازیم و RTT ها را شکست به شکست اندازه بگیریم .

اما time out هر لحظه تغییر می‌کند . که می‌توان برای رفع مشکل از روش‌های آماری

RTT

استفاده کنیم .

$$EstimatedRTT = (1-\alpha) * EstimatedRTT + \alpha * sampleRTT$$

نمونه جدیدی که الان بدست آوردیم. آخرین مشخص شده قبلی

- exponential weighted moving average (EWMA)

یعنی یکسری sample داریم که به صورت مرتب ادامه (اسلایدر ویدیو) توضیح می‌شود:
که میانگین می‌گیریم و میانگین را با زمان آپدیت می‌کنیم ولی عامل پنجره ثابت است.
(یعنی RTT رو اندازه‌گیریم اما average گیری مرتب انجام دهیم)

- influence of past sample decreases exponentially fast

- typical value: $\alpha = 0.125$

* timeout interval: Estimated RTT plus "safety margin"

* Timeout Interval = Estimated RTT + 4 * Dev RTT
(Estimated RTT: estimated RTT; 4 * Dev RTT: "safety margin")

* DevRTT: EWMA of sampleRTT deviation from Estimated RTT

* DevRTT = $(1-\beta)$ * DevRTT + $\beta$ * |sampleRTT − Estimated RTT|
(میانگین؛ انحراف معیار)

$\beta = 0.25$

TALASH

Year. Month. Date. ( ) Subject :

Tcp sender : براساس یکسری event کار می‌کند.

(مثلاً به داده‌ای که از application layer می‌آید براساس اول داده، یک شماره ایجاد می‌کند و seq # بهش تخصیص میده و درصورتی که ارسال را انجام بده timer را شروع می‌کند.) ← event : data received from application

(وقتی که timeout رخ بده: شماره‌ای که هنوز ack نشده را ارسال می‌کند و timer را restart می‌کند.) ← event: time out

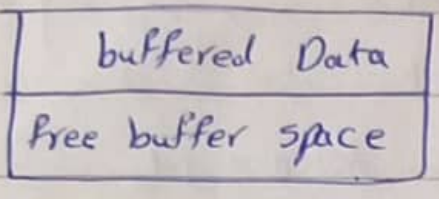
(اگر برایش ack ارسال شد (یکی که قبلاً unack بوده) ← بعضی ack را آپدیت می‌کند و به timer جدید برایش اختصاص میده) ← event : Ack received

* Tcp یه سری عکس توی اسلایدها هست نگاه کن.

Tcp flow control : در applicatin layer گیرنده به سرعت پردازش دارد. به همین خاطر سمت گیرنده کانکشن Tcp ⇐ Transport layer یک بافر قرار میده و شماره‌ها و بایت‌هایی که آمدند را در بافرش قرار میده تا با سرعت applicatin تطبیق بده. به همین خاطر اگر فرستنده با سرعت زیاد ارسال را انجام بده، بافر پر می‌شود.

TALASH

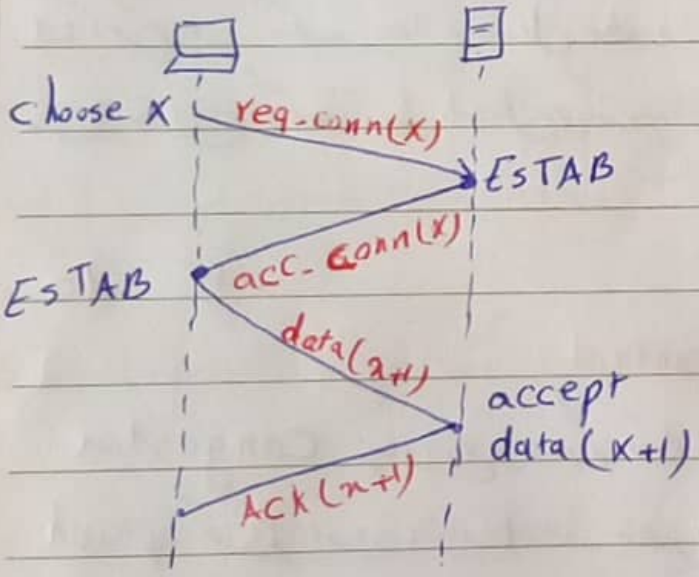
در سمت گیرنده یه بافر بنام RcvBuffer وجود دارد.




که بوسیله این بافر، گیرنده "advertises" می‌کند یعنی که اعلام می‌کند چقدر فضای خالی دارد.

Handshake : یعنی فرستنده و گیرنده باهم صحبت می‌کنند که یه کانکشن Tcp رو ایجاد کنند و establish کنند. و بعد از این شروع به ارتباط کنند.

که برای این کار طول buffer size رو باهم چک می‌کنند و segment size رو چک می‌کنند و seq# ها رو باهم بررسی می‌کنند.




شاید req-conn(x) بعد از timeout برسه. که در اینصورت server دوباره establish میشه.

و سرور دوباره منتظر یه صحبت جدید کلاینت میشه. که دیگه کلاینتی وجود نداره که بی‌خودی یه port سرور اشغال میشه.

« شکل توی اسلایدها رو نگاه کن »

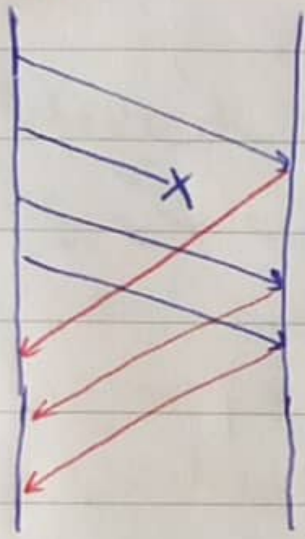
Year. Month. Date. ( ) Subject:

## Tcp 3-way handshake :

* seq می‌فرستد و Ack رو دریافت می‌کند.
وقتی که کلاینت جواب Ack رو گرفت establish می‌شه. و Acknum=y+1 میشه. و بعد سرور establish میشه.

* مشکلاتی که 2-way بود و 3-way برطرف میشه.

## Tcp Fast retransmit :



در Tcp ترکیبی از Go back N, selective Repeat است.
عموماً Tcp از GBN استفاده می‌کند. اگر یک سگمنت به مشکل برخورد مجدداً ارسال می‌کند.
اما اگر یکی از سگمنت‌ها به مشکل بخورد یه هوشمندی وجود داره و فقط همان سگمنت را ارسال می‌کند.