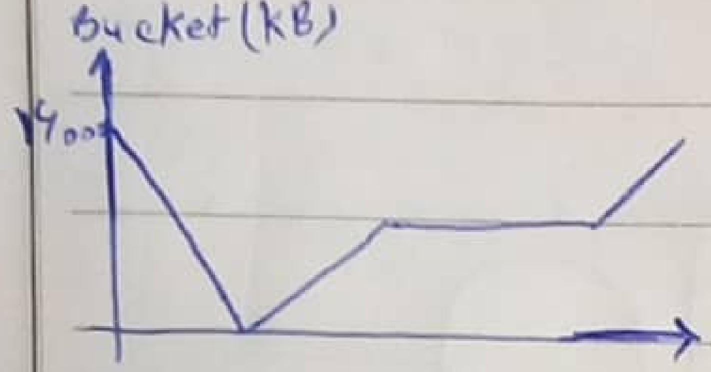
Token Bucket : Example

جلسه ۱۸ :



Capacity of B = ۱۲۰۰ KB , R = ۲۰۰

در این حالت Bucket در وهله اول خالی می‌شود ،
اما چون ظرفیتش زیاد است ، شروع به پر شدن می‌کند
و هیچ وقت در حالت کاملا خالی قرار نمی‌گیرد.
که این کار باعث می‌شود burst بزرگتری را ایجاد کند. و معمولاً بیشتر در روترها
استفاده می‌شود. چون سریعتر می‌تواند Q را خالی کند.

یک پیاده سازی که داشتیم این بود که : برخلاف تعریفی که از Token Bucket کردیم :
(نرخ Token/sec وارد Bucket می‌شود و درون Bucket مقداری Token
قرار می‌گیرد و بر این اساس کار می‌کند)

در مثال‌های قبل یه معادلی گفته شد که Bucket یک ظرفیت ثابتی دارد و همچنین نرخ ورودی‌شان هم ثابت و بیت است.

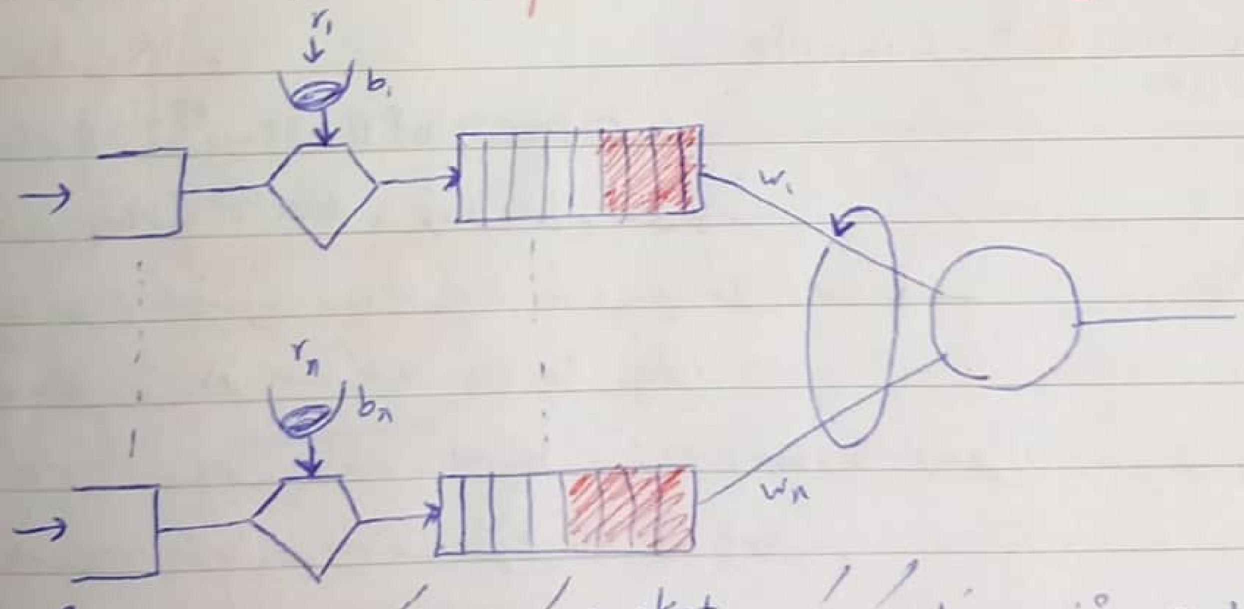
و اینکه بخواهیم این دو را معادل سازی کنیم: $\boxed{\frac{R}{\Delta t}}$ و براساس اینکه Δt چه مقدار است

می‌توان براساس $\frac{R}{\Delta t}$، در هر کلاک می‌توان به Bucket یک Token اضافه کرد.

مثلاً در مثال‌های قبل $R=200^{kb}$ و $\Delta t=1$ در نظر گرفته بودیم.

WFQ : (Weighted Fair Queuing) یکسری ترافیک وجود داشت که در کلاس‌های مختلفی وجود داشت و اولویت‌های بین آنها با وزن‌ها مشخص می‌شود.

Token Bucket + WFQ = provable Maximum Delay in a Queue



ترکیب این دو مشخص خواهد کرد که هر packet که وارد یکی از این مسیرها می‌شود

حداکثر چقدر تأخیر را تحمل خواهد کرد.

Year.　Month.　Date.　(　)　Subject :

فرض کنیم n تا کلاس ترافیک داریم که از این n تا هر کدام اولویت خودشان را در WFQ

دارند و هر کدام یک Token Bucket برایشان تعریف شده است. فرض می‌کنیم یک ترافیکی وارد

مسیر یک می‌شود و bucket ها پر شده است و ترافیکی وجود نداشته (در وهله اول).

در این حالت اولین بسته‌ای که آخرین وارد شده است بیشترین تأخیر را تحمل خواهد کرد.

۱۹:۴۰

به اندازه $\frac{w_1}{\sum w_i}$ ظرفیت به آن تخصیص داده خواهد شد. ⇐ تأخیر ایجاد شده برای packet ها $\frac{\text{طول Packet}}{\frac{w}{\sum w_i}}$

حداکثر تأخیری که یک packet متحمل می‌شود زمانی هست که وارد یک مسیر شود و به عنوان

b_1 امین وارد شود و b_1 تا تأخیر را متحمل می‌شود.

حداکثر تأخیر

$$d_{max} = \frac{b_1}{R \times w_1 / \sum w_i}$$

* Token چگونه ساخته می‌شود؟ می‌توان حافظه یا یک کامپیوتر داشته باشیم که با کلاک Token اضافه می‌کند. یک فلگ هم می‌تواند باشد.

* روش‌های congestion control:

1) TCP congestion control: مهمترین جایی که congestion control انجام می‌شود

در resource Allocation، به نوعی کاربردی نسبت به congestion control است و طراحی‌های متعددی مانند router centering، host centering، reservation Base و window base، back base، waited می‌تواند باشد.

Year. Month. Date. ( ) Subject :

اما در Tcp congestion دو تا طراحی وجود دارد ~~[illegible]~~

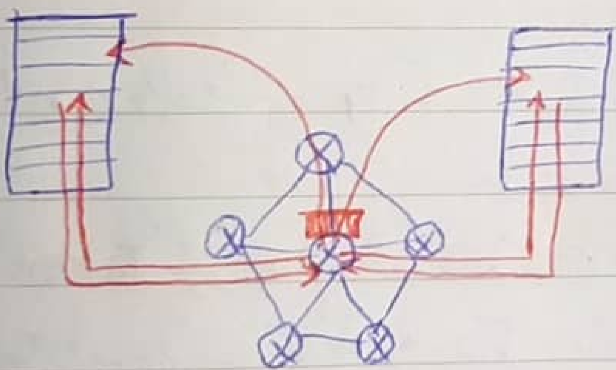
- End-end congestion control : در این حالت فقط هاست های ابتدا و انتها فقط درگیر congestion con می شوند. و هاست های میانی هیچ کمکی به ما نمی کنند

+ هاست ها براساس یک مکانیزمی ، یک پروتکل Tcp congestion استاندارد را انجام می دهند

- Network - assisted congestion con : در این حالت Network هم می تواند کمک کردن
کمک کند به نحوی که در router center ، روترهای میانی هم می توانند به طور مستقیم feed back ارسال کنند.

که validation های مانند Tcp. ECN ، ATM ، DECbit

ECN: explicit congestion Notification (هاست های دو طرف مدیریت را انجام می دهند)



classic Tcp : classic Tcp که به آن congestion control هم به آن اضافه شده است و ورژن استاندارد و اولیه است در آن Network هیچ کمکی نمی کند و خود سیستم ها کارشان را انجام می دهند ، به این علت که Ip هیچ وقت feed back ارسال نمی کند و استفاده ای از لایه های پایین تر شبکه نمی کند.

در classic Tcp فرستنده می خواهد نرخ rate های ارسالی را محدود کند
برای اینکه از congestion رخ داده جلوگیری کند.

TALASH

Year. Month. Date. ( ) Subject :

ـ بنابراین چیز مسئله بوجود می‌آید: ① چگونه می‌توانند rate های ارسالی را محدود کنند؟

② sender ها چگونه متوجه رخ دادن congestion شوند؟

③ با چه روشی (پروتکلی) می‌توان این نرخ را محدود کرد؟

ج سوال اول: با استفاده از flow control و تعریف یک window برای congestion

ج سوال دوم: با استفاده از Time out و با استفاده از selective repeat

(three duplicate Ack) که three duplicate بهتر از time out است چون

حاصل ۴/۵ ack به درستی ارسال شده‌اند.

ج سوال سوم: متوجه نشدم ☺

* نحوه مدیریت نرخ ارسال

last Byte sent - last Byte Acked ≤ min(cwnd, rwnd)

فاصله پنجره (طول پنجره)

قبلاً با flow control (rwnd) مدیریت می‌شد اما اینجا از congestion window هم استفاده می‌شود

و min این دو مد نظر است. که به نوعی تنظیم نرخ هم انجام می‌شود چون ممکن

است buffer سایز بزرگی داشته باشد اما ما cwnd آن را محدود می‌کنیم.

$$\text{Tcp rate} \approx \frac{cwnd}{RTT} \quad \text{byte/sec}$$

* در Tcp یک فرآیندی به نام self-clocking وجود دارد که با فرض اینکه هیچ گونه

مشکلی (lost و ...) وجود ندارد، وقتی send انجام می‌شود براساس اینکه چقدر

Ack دریافت می‌شود می‌توانیم اندازه window را تنظیم کرد و براساس اینکه Ack ها سریع و یا

کند می‌آیند می‌توان اندازه پنجره را با سرعت زیاد یا کم براساس clocking تنظیم کرد.

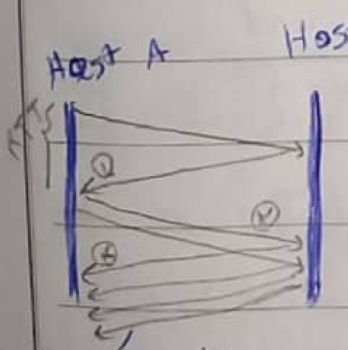
که یک clocking به صورت virtual است و واقعی نیست.
و clock به صورت یکنواخت نیست و به صورت متغیر کار می کند.

* سرنخ ها principle (قاعده کلی) برای Tcp congestion control :

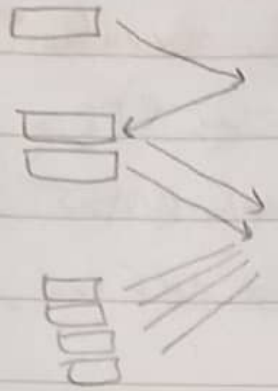
① اگر lost رخ داد می توانیم sender's rate را کاهش دهیم.
② اگر Acknowledg رخ داد، می توانیم sender را افزایش دهیم.
③ وضعیت link را probe (بررسی) کنیم (Bandwidth probing) که بتوانیم بر اساس آن دو مورد قبلی را انجام دهیم.

* بخش های مختلف Tcp congestion :
① slow start ② congestion avoidance ③ Fast recovery

وقتی که روش Tcp congestion وجود دارد دو مورد slow start و congestion avoidance به طور ویژه به همراه خود دارد.
اما لزوماً fast recovery ندارد.



① slow start : فرض کنیم یک کامپیوتر شروع به کار می کند و Tcp connection را برقرار کرده است در slow start اولین کاری که انجام می شود: cwnd = 1 MSS و بعد صبر می کند اگر Acknowledg گرفت به ازای هر Ack یک عدد MSS به cwnd اضافه می کند.
(یعنی: اگر یک segment فرستاد و ack آمد، یک عدد MSS اضافه می کند)



یعنی در واقع rate به صورت exponentially (نمایی) در حال افزایش است.

* این رشد نمایی را تا کجا باید ادامه دهیم؟

(۱) زمانی time out رخ دهد، دوباره به همان cwnd = 1 MSS می‌رسیم. یعنی عملاً شبکه restart می‌شود. و می‌گوییم که متغیری بنام ssthresh (slow start threshold) تعریف می‌کنیم که برابر است با $\frac{cwnd}{2}$

هدف آن این است که: با توجه به رشد نمایی که سریع به آن نقطه‌ای می‌رسیم که congestion رخ داده است، بنابراین ssthresh از این کار جلوگیری می‌کند (نصف از آنجایی که congestion رخ داده است)

(۲) می‌توان زمانی که به ssthresh رسیدیم نحوه افزایش cwnd را تغییر دهیم. یعنی وارد فاز congestion avoidance (مورد دوم) می‌شود.

(۳) زمانی که three duplicate Ack رخ می‌دهد، (۱) وارد فاز بعدی Fast recovery می‌شود و یا در بعضی از سیستم‌ها وقتی three duplicate رخ می‌دهد (۲) از یک سر دوم می‌شود و restart می‌شود. یعنی دو حالت وجود دارد. و به ظاهر حالت دوم است که می‌گوییم می‌توان لزوماً Fast recovery هم وجود نداشته باشد.

Year. Month. Date. ( ) Subject :

۲) congestion avoidance : یکی از راه‌های معروف که استفاده می‌شود.
در هر RTT به ازای هر segment که ارسال می‌شود و ack دریافت شد، یک واحد MSS را اضافه کنیم. تفاوت با slow start این است که در slow start اگر ۱۰۰ تا بود می‌شد ۲۰۰ تا اما در اینجا اگر ۱۰۰ تا باشد می‌شود ۱۰۱.

مثال: MSS = 1,460 byte cwnd = 14,600 byte
then 10 segment are being sent within an RTT.
یعنی در هر cwnd ما ۱۰ تا mss داریم و این ۱۰ تا در RTT ارسال می‌شود.
اگر بگیم Ack رسید می‌توانیم به اندازه $\frac{1}{10}$ از mss را افزایش دهیم و ارسال انجام دهیم.

* slow start ⇒ RTT → cwnd
* congestion avoid → RTT → cwnd + MSS → یعنی به ازای هر RTT یک MSS اضافه می‌شود.

در این مثال به ازای هر MSS که اضافه می‌شود به اندازه $\frac{1}{10}$ پنجره بزرگتر می‌شود.

* این نوع رشد کردن تا کجا باید ادامه داشته باشد و چگونه خاتمه می‌شود؟

① time out رخ دهد.
② triple duplicate Ack : اول ببرسیم که در کدام نقطه congestion رخ داده است و بعد اندازه congestion حقیر است و آنرا نصف می‌کند و از نقطه نصف شروع می‌کند. (یا) می‌تواند براساس اینکه سه بار ack موفق داشته است، سه تا MSS اضافه می‌کند. و سپس ۳ تا بالاتر می‌رود و از آنجا کار خود را شروع می‌کند و سپس وارد Fast recarery می‌شود.

TALASH

۳) **Fast Recovery** : فرض بر این است که شبکه مطلوب نیست و هنوز ترافیک lost وجود دارد. و اینگونه است که به ازای هر duplicate Ack که دریافت می‌کند، MSS را اضافه می‌کند تا زمانی که تمام Ack های missing segment ها را دریافت کند و بعد از انجام این کار وارد فاز congestion avoidance می‌شود.

* وقتی در Fast هستیم یعنی اینکه یک سری سگمنت های lost شده اند و هنوز ack دریافت نشده است و به محض دریافت از این فاز خارج شده و وارد congestion avoidance می‌شود.

* و اگر اینجا time out رخ داد وارد فاز slow-start می‌شود. (ممکن است که صبر کنیم ولی ack سگمنت های lost شده دریافت نشود بنابراین time out رخ می‌دهد و وارد slow شروع)

Tcp Tahoe : fast recovey ندارد، یعنی چه time out و چه triple duplicate رخ دهد cwnd = 1MSS است.

Tcp Reno : fast recovey دارد (قبلاً بیشتر استفاده می‌شده ولی الان کمتر شده)