CS 5224

Transport

Dr. Chan Mun Choon
School of Computing, National University of Singapore

Outline

- **Window Flow Control**
- (Very brief) Review of TCP
- TCP throughput modeling
- TCP variants/enhancements

Window Flow Control

- A session between a transmitter A and a receiver B is said to be window flow controlled if there is an upper bound on the number of data units that have been transmitted by A and are not yet known by A to have received by B
  - The upper bound is known as the window size
- The receiver B notifies the transmitter A that it has disposed of a data unit by sending a special message to A, which is called a permit/acknowledgement/allocate message
- Upon receive an acknowledgement, A is free to send more data
- Acknowledgement can be sent as a special packet or can be piggybacked on regular data packet

Window Flow Control

- Let the round-trip time be d, the window size be W and the transmission time of a single data unit be X
  - WX is the time it takes to sent an entire window of data
- Flow control is inactive if d <= WX
  - Since acknowledges comes back before the entire window of data is sent
  - Sending rate is capped by (1/X)
- Flow control is active if d > WX
  - Sending rate is capped by W/d
  - Therefore, the rate of a window flow control system is **min {1/X, W/d}**

Issues

- Cannot guarantee minimum throughput since rate is a function of end-to-end delay which varies
- Choice of window size
  - Choose small windows to avoid delays and congestion
  - Choose large window for higher throughput
- Determining the proper window size and adjusting that size in response to congestion is not easy (see for example TCP)
  - depends of traffic load
  - depends on the length of the route chosen, a longer route will need a large window size
  - Generally, smaller window size during high load and larger window size during light to medium load

Rate-Based Flow Control

- Besides window-based flow control, rate-based flow control is also possible
  - Similar issues, what rate should be chosen?
- Earlier and current versions of TCP are widow-based, but there are also rate-based versions

Outline

- Window Flow Control
- **Review of TCP**
- TCP throughput modeling
- Enhancements to TCP

TCP Congestion Control

- The idea is for each source:
  - to determine how much capacity is available in the n/w
  - hence to know how many packets it can have in transit
  - use the arrival of an ack as a signal that one of its packet has left the n/w
  - then insert a new packet into the n/w - self clocking

TCP Congestion Control *(cont)*

TCP uses four different mechanisms:

1. Congestion Avoidance : behaviour with mid congestion
2. Slow Start: behaviour after serious congestion
3. Fast Retransmit
4. Fast Recovery

⌘ Since TCP's congestion control is window-based, all these mechanisms affect the size of the congestion window

TCP Congestion Control

- end-end control (no network assistance)
- sender limits transmission:
  `LastByteSent-LastByteAcked ≤ CongWin`
- Roughly, $\boxed{\text{rate} = \frac{\text{CongWin}}{\text{RTT}} \text{ Bytes/sec}}$
- **`CongWin`** is dynamic, function of perceived network congestion

<u>How does sender perceive congestion?</u>

- loss event = timeout (why?) *or* 3 duplicate acks (why 3?)
- TCP sender reduces rate (**`CongWin`**) after loss event

Additive Increase/Multiplicative Decrease

- also known as congestion avoidance algorithm
- TCP maintains a variable called congestion window "cwnd" for each connection, to limit the data in transit
- cwnd is the congestion control's counterpart to flow control's advertised window
- maintained in bytes
- sender can transmit up to **minimum of cwnd and advertised window**
- cwnd is decreased when congestion level goes up and increased when congestion level goes down

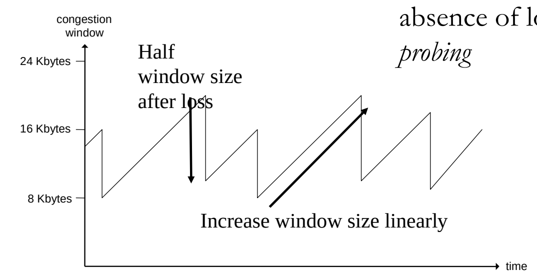
TCP AIMD

<u>multiplicative decrease:</u>

cut **`CongWin`** in half after loss event

<u>additive increase:</u>

increase **`CongWin`** by 1 MSS every RTT in the absence of loss events: *probing*



Long-lived TCP connection

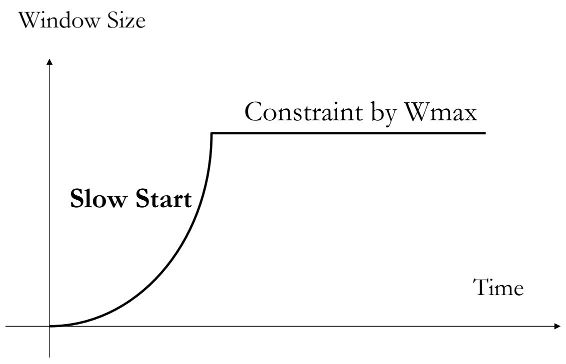
Why AIMD?

- Why pick AIMD ?

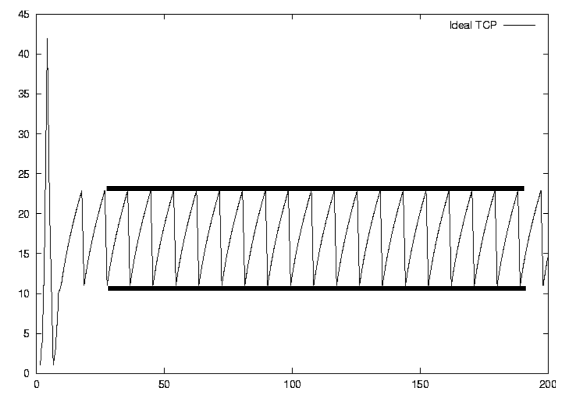
Outline

- Window Flow Control
- Review of TCP
- **TCP throughput modeling**
- Enhancements to TCP

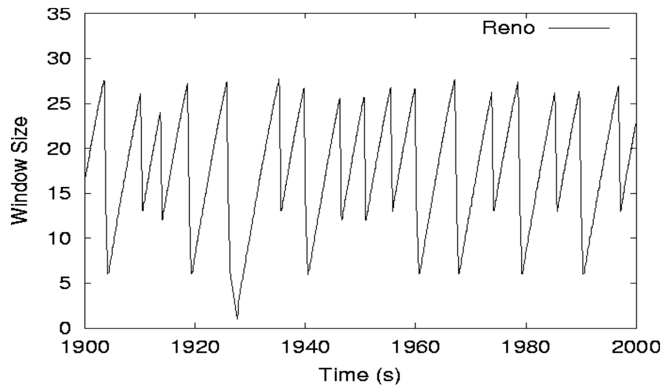
TCP Window Evolution : No Loss



TCP Window Evolution : Loss, Ideal Case

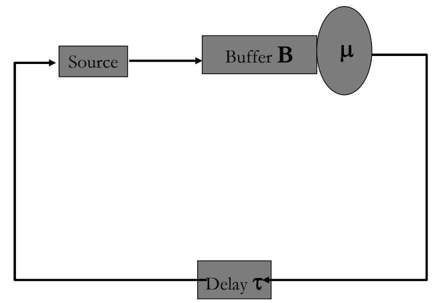


TCP Window Evolution : Loss



TCP Throughput Modeling

• Lakshman and Madhow, “The performance of networks with high bandwidth-delay products and random loss,” IEEE/ACM Transaction of Networking, Jun 1997.



System Model
• infinite data source
• single bottleneck link with capacity μ
• FIFO buffer of size B
• All delays except for queuing and service are lumped into a single delay τ

TCP Reno Throughput Modeling

The source uses TCP Reno.

1. After a new ACK
   1. If W < Wt, set W = W + 1 (Slow Start)
   2. Else W = W + 1/[W] (Congestion Avoidance)
2. After 3 duplicate ACKs
   1. Retransmit
   2. Wt = W/2; W = Wt
   3. Resume with congestion avoidance
3. If timer expire
   1. Retrasmit
   2. Wt = W/2; W = 1
   3. Resume with slow start

Evolution with only Buffer Overflow Loss

- Let $\beta = B/(\mu\tau + 1)$ and $T = \tau + 1/\mu$
  - $\mu\tau$ is also called the **bandwidth-delay product**
- Maximum window size $W_{max} = \mu T + B = \mu\tau + B + 1$
- Consider only congestion avoidance phase, after a packet loss the window size W = Wmax/2
- Rate of window increase dW/da = 1/W
  - (da = rate of acknowledgement)

TCP Reno Throughput

- For $W <= \mu T$ (phase A)
  - $t_A = T(\mu T - W\max/2)$
  - $n_A = ((W\max/2)t_A + t_A^2/(2T)) / T$
- For $W >= \mu T$ (phase B)
  - $t_B = (W\max^2 - (\mu T)^2)/(2\mu)$
  - $n_B = \mu t_B$
- Throughput = $(n_A + n_B) / (t_A + t_B)$
- Question: what is the minimum value of B in order to maximize throughput?

$$\frac{W}{2} + \left(\frac{W}{2} + 1\right) + \cdots + W = \sum_{n=0}^{W/2} \left(\frac{W}{2} + n\right)$$

TCP Throughput with loss

- Question: Given **loss rate** and **rtt**, what is the throughput?
  - Loss rate and rtt are network-related parameters measured
- Again assume single packet loss and let a cycle be define as the duration between 2 packet loss
- Let maximum window size be W. Hence the window size changes between W/2 to W and the average window size is 3W/4
  - Throughput is approximated as (3W/4)/RTT

TCP Throughput

- The number of packets sent in a cycle is
  - $W/2 + (W/2 + 1) + \ldots + W$
  - $= 3W^2 / 8 + 3W/4$
  - The loss rate $p = 1/(3W^2/8 + 3W/4)$, if W is large, then $p \sim 1/(3W^2 / 8)$
    - $W = (8/3p)^{0.5}$
- Finally, throughput = $3/4\ (8/3p)^{0.5} / \mathrm{RTT}$

$$\left(\frac{3}{2p}\right)^{0.5} \frac{1}{RTT} \approx 1.22 \frac{1}{RTT\sqrt{p}}$$

TCP Throughput Model

- In the previous models, there are only single buffer overflow loss and there is no timeout. A better model with timeout can be obtained, assuming timeout value to be To, p is small and loss events are independent.

$$Throughput \approx \frac{1}{1.22RTT\sqrt{p} + T_o \min(1,3\sqrt{\frac{3p}{8}})\,p(1+32p^2)}$$

- There are a number of other models using different assumptions of the error distribution

Outline

- Window Flow Control
- Review of TCP
- TCP throughput modeling
- **TCP Variants/**Enhancements

TCP Variants

- The congestion avoidance strategies of the most common implementations of TCP are based on loss detection, for example TCP Tahoe, Reno, and Sack
- There are a number TCP variants which does not use loss detection for window size adjustment
  - TCP Vegas (1994)
  - TCP Westwood (2000)
  - Fast TCP

TCP Vegas

- TCP Vegas computes two throughput, expected and actual
  - Expected thru'put (E) = window size / minimum RTT
  - Actual thru'put (A) = window size/RTT
  - If (E – A) < α, increase window size linearly
  - If (E – A) > β, decrease window size linearly
  - where $\alpha < \beta$

Issues with TCP Vegas

- Hard to fine-tune and be robust
  - Random queuing delay: use minRTT to approximate expected throughput
  - Route changes cause changes in minRTT and can cause significant error in expected throughput estimation
  - Delayed ACKs complicates rate estimation
  - Link layer loss, retransmission, compression
    - Vegas tries to keep window small and is more sensitive to loss
- Reno may be dumb, but it is robust
- *Reference: http://flophouse.com/~neal/uw/linux-vegas*

TCP Westwood

- Use the same congestion avoidance of TCP Reno but computes Eligible Rate Estimate (ERE) to decide window size (cwnd) and slow start threshold (ssthresh).
  - $b_k = d_k / (t_k - t_{k-1})$
  - $R_k = \alpha_k R_{k-1} + 0.5(1-\alpha_k)(b_k + b_{k-1})$
- After triple duplicates
  - **cwnd = R_k * RTTmin**
- After a timeout
  - **ssthresh = R_k * RTTmin**
  - **cwnd = 1**

RFC 3449: TCP with Path Asymmetry

- Asymmetric links are exhibited by several network technologies, including
  - cable data networks
  - digital video broadcast
  - Very small aperture satellite terminals (VSAT)
  - ADSL
- Many of these networks are being deployed as high speed Internet access networks
- Asymmetry can be caused by
  - differences in transmit and receive capacity
  - shared media in the reverse direction
- Problem caused by the imperfection and variability of ACK feedback

Path Asymmetry

- In general, the downstream bandwidth (from Internet to user) is larger than the upstream bandwidth (from user to Internet)
- Upstream bottleneck link has sufficient buffer
  - TCP Performance is a strong function of the normalized bandwidth ratio k
  - If downstream link is 10Mbps and upstream link is 50Kbps, the ratio = 10/0.05 = 200
  - With 1KB data packets and 40 bytes ACKs, the ratio of the packet sizes is 25
  - K = 200/25 = 8

Path Asymmetry

- Assuming that we send one ACK for each data packet, in order to utilize 10Mbps in the downstream, 10/0.008 = 1250 data packets will be transmitted
- For 1250 data packets, 1250 ACKs will be needed, which is 1250*40*8 = 400Kbps > 50Kbps
- Maximum throughput attained by downstream link is only (50K/320) * 8K= 1.25Mbps
- Alternatively, an ACK should be sent for every 40 * (10M/50K) = 8KB in order to fully utilized the downstream link
- Whenever K>1 or K>0.5, ACK clocking breaks down

Path Asymmetry

- If the upstream link is slow, saturated and the buffer is large enough
  - Affects TCP by causing excessive RTT
  - May trigger false time out
- If upstream buffer is small and ACKs can be dropped
  - Since ACKs are cumulative, may not affect performance as much
  - Increases the likelihood of data burst in the downstream direction and increases the likelihood of data packet loss
  - May affect fast retransmission and recovery since it takes longer to receive three duplicate ACKs