

TCP Reno/Vegas

10/21/2009

1

Admin.

- Questions on programming assignment 2?

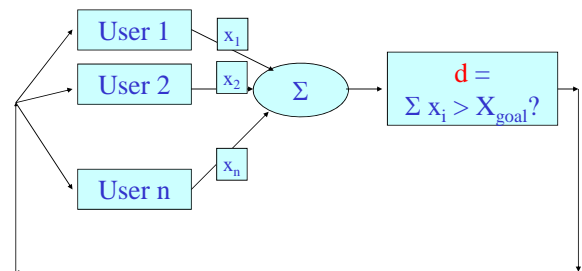
2

Recap: The Desired Properties of a Congestion Control Scheme

- Efficiency: close to full utilization but low delay
 - fast convergence after disturbance
- Fairness (resource sharing)
- Distributedness (no central knowledge for scalability)

3

Recap: A Simple Model



Flows observe congestion signal d , and locally take actions to adjust rates.

4

Four Special Cases

	<u>Additive Decrease</u>	<u>Multiplicative Decrease</u>
<u>Additive Increase</u>	AIAD ($b_I=b_D=1$)	AIMD ($b_I=1, a_D=0$)
<u>Multiplicative Increase</u>	MIAD ($a_I=0, b_I>1, b_D=1$)	MIMD ($a_I=a_D=0$)

$$x_i(t+1) = \begin{cases} a_I + b_I x_i(t) & \text{if } d(t) = \text{no cong.} \\ a_D + b_D x_i(t) & \text{if } d(t) = \text{cong.} \end{cases}$$

5

Another Look

- Consider the difference or ratio of the rates of two flows
 - AIAD
 - MIMD
 - MIAD
 - AIMD

6

Mapping A(M)I-MD to Protocol

- What do we need to apply the A(M)I-MD algorithm to a sliding window protocol?

$$x_i(t+1) = \begin{cases} a_i + x_i(t) & \text{if } d(t) = \text{no cong.} \\ b_D x_i(t) & \text{if } d(t) = \text{cong.} \end{cases}$$

7

Outline

- Recap
 - *TCP/Reno*

8

TCP Congestion Control

- Closed-loop, end-to-end, window-based congestion control
- Designed by Van Jacobson in late 1980s, based on the AIMD alg. of Dah-Ming Chu and Raj Jain
- Works well so far: the bandwidth of the Internet has increased by more than 200,000 times
- Many versions
 - TCP/Tahoe: this is a less optimized version
 - TCP/Reno: many OSs today implement Reno type congestion control
 - TCP/Vegas: not currently used

For more details: see TCP/IP illustrated; or read http://lxr.linux.no/source/net/ipv4/tcp_input.c for linux implementation

9

TCP/Reno Congestion Detection

- Detect congestion (d) in two cases and react differently:

- 3 dup ACKs
- timeout event

Philosophy:

- 3 dup ACKs indicates network capable of delivering some segments
- timeout is "more alarming"

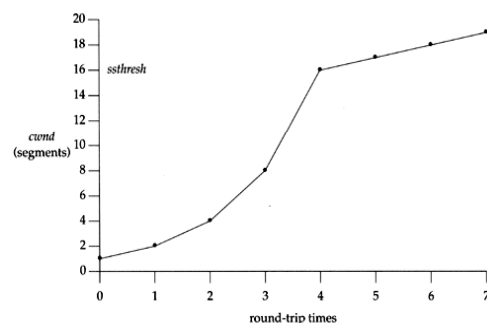
10

Basic Structure

- Two "phases"
 - slow-start: *MI*
 - congestion avoidance: *AIMD*
- Important variables:
 - *cwnd*: congestion window size
 - *ssthresh*: threshold between the slow-start phase and the congestion avoidance phase

11

Visualization of the Two Phases



12

Slow Start: MI

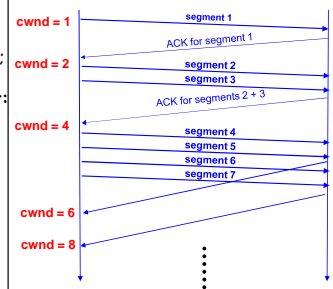
- What is the goal?
 - getting to equilibrium gradually but **quickly**
- Implements the MI algorithm
 - **double *cwnd*** every RTT until **network congested**
→ get a rough estimate of the optimal of *cwnd*

13

Slow-start

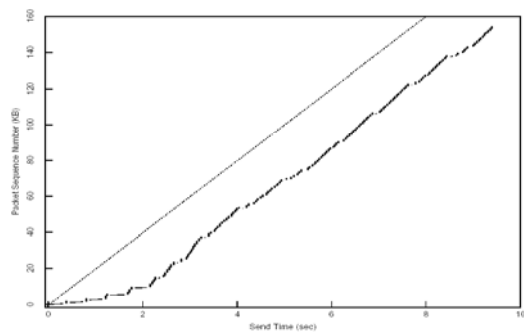
Initially:
 $cwnd = 1;$
 $ssthresh = \text{infinite (e.g., 64K)};$

For each newly ACKed segment:
 if ($cwnd < ssthresh$)
 /* slow start*/
 $cwnd = cwnd + 1;$



14

Startup Behavior with Slow-start



See [Jac89]

15

TCP/Reno Congestion Avoidance

- Maintains equilibrium and reacts around equilibrium
- Implements the AIMD algorithm
 - increases window by 1 per round-trip time (how?)
 - cuts window size
 - to half when detecting congestion by 3DUP
 - to 1 if timeout
 - if already timeout, doubles timeout

16

TCP/Reno Congestion Avoidance

Initially:
 $cwnd = 1;$
 $ssthresh = \text{infinite (e.g., 64K)};$

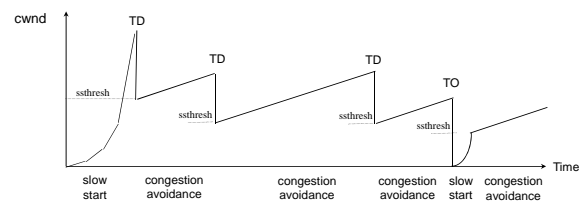
For each newly ACKed segment:
 if ($cwnd < ssthresh$)
 /* slow start*/
 $cwnd = cwnd + 1;$
 else
 /* congestion avoidance; *cwnd* increases (approx.)
 by 1 per RTT */
 $cwnd += 1/cwnd;$

Triple-duplicate ACKs:
 /* multiplicative decrease */
 $cwnd = ssthresh = cwnd/2;$

Timeout:
 $ssthresh = cwnd/2;$
 $cwnd = 1;$
 (if already timed out, double timeout value; this is called exponential backoff)

17

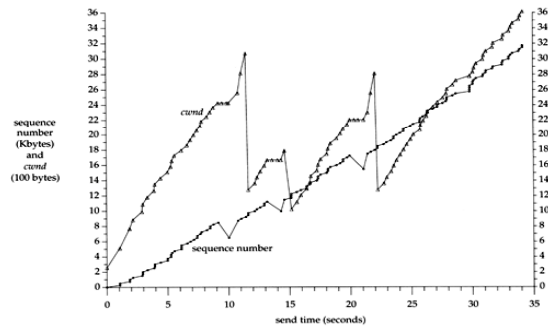
TCP/Reno: Big Picture



TD: Triple duplicate acknowledgements
 TO: Timeout

18

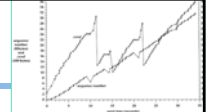
A Session



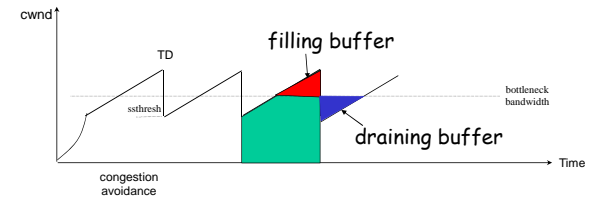
Question: when cwnd is cut to half, why sending rate is not?

19

TCP/Reno Queueing Dynamics



- Consider congestion avoidance only



There is a filling and draining of buffer process for each TCP flow.

20

Outline

- Recap
- TCP/Reno
- *TCP/Reno throughput analysis*

21

Objective

- To understand the throughput of TCP/Reno as a function of RTT (RTT), loss rate (p) and packet size
- We will derive the formula twice, using two setups using two different approaches

22

TCP/Reno Throughput Modeling

- Given mean packet loss rate p , mean round-trip time RTT , packet size S
- Consider only the congestion avoidance mode (long flows such as large files)
- Assume no timeout
- Assume mean window size is W_m segments, each with S bytes sent in one RTT :

$$\text{throughput} \approx \frac{W_m * S}{RTT} \text{ bytes/sec}$$

23

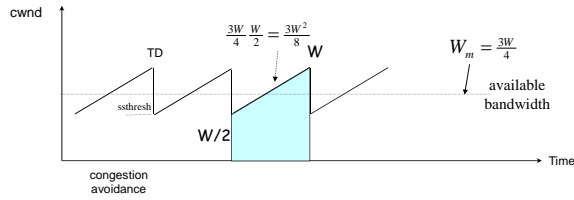
Outline

- Recap
- Linear congestion control law
- TCP/Reno
- TCP/Reno throughput analysis
- *analysis 1: deterministic*

24

TCP/Reno Throughput Modeling: Relating W with Loss Rate p

- Consider congestion avoidance only



Assume one packet loss (loss event) per cycle
 Total packets send per cycle = $(W/2 + W)/2 * W/2 = 3W^2/8$
 Thus $p = 1/(3W^2/8) = 8/(3W^2)$

$$W = \frac{\sqrt{8/p}}{\sqrt{3}} = \frac{1.6}{\sqrt{p}} \Rightarrow \text{throughput} = \frac{S}{RTT} \cdot \frac{3}{4} \cdot \frac{1.6}{\sqrt{p}} = \frac{1.2S}{RTT \sqrt{p}}$$

25

Outline

- Recap
- TCP/Reno
- TCP/Reno throughput analysis
 - analysis 1: deterministic
 - analysis 2: random loss

26

TCP/Reno Throughput Modeling

$$\Delta W = \begin{cases} \frac{1}{W} & \text{if the packet is not lost} \\ -\frac{W}{2} & \text{if packet is lost} \end{cases}$$

$$\text{mean of } \Delta W = (1-p) \frac{1}{W} + p \left(-\frac{W}{2}\right) = 0$$

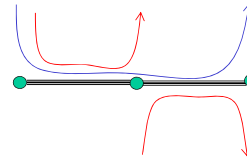
$$\Rightarrow \text{mean of } W = \sqrt{\frac{2(1-p)}{p}} \approx \frac{1.4}{\sqrt{p}}, \text{ when } p \text{ is small}$$

$$\Rightarrow \text{throughput} \approx \frac{1.4S}{RTT \sqrt{p}}, \text{ when } p \text{ is small}$$

27

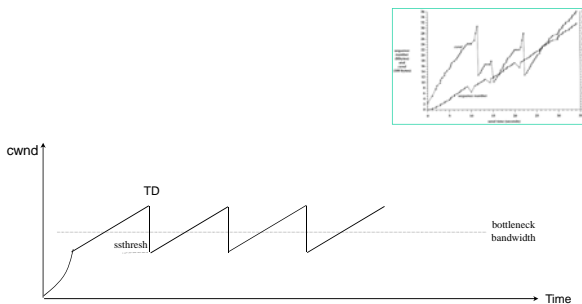
Summary: TCP/Reno Throughput Modeling

- They are all *approximate* modeling
 - the details and the exact numbers are not important
 - the objective is to help us understand TCP better



28

Problem of TCP/Reno Behavior



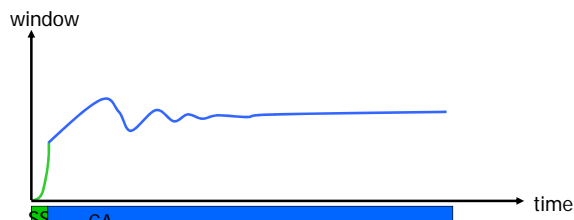
29

Outline

- Recap
- TCP/Reno
- TCP/Reno throughput analysis
 - TCP/Vegas

30

TCP/Vegas (Brakmo & Peterson 1994)

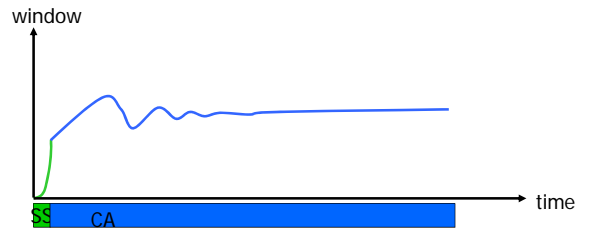


- Idea: try to detect congestion by delay before loss
- Objective: not to overflow the buffer; instead, try to maintain a *constant* number of packets in the bottleneck queue

31

TCP/Vegas: Key Question

- How to estimate the number of packets queued in the bottleneck queue?



32

Background: Little's Law (1961)



- For any system with no or (low) loss.
- Assume
 - mean arrival rate X , mean service time T , and mean number of requests in the system W
- Then relationship between W , X , and T :

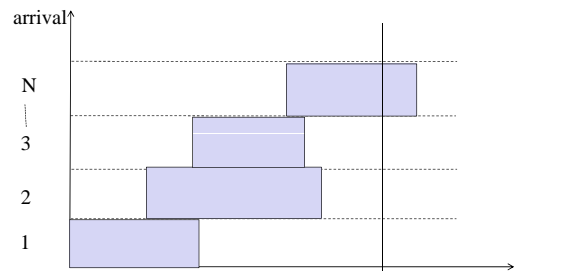
$$W = XT$$

Example: Yale College admits 1500 students each year, and mean time a student stays is 4 years, how many students are enrolled?

33

Little's Law

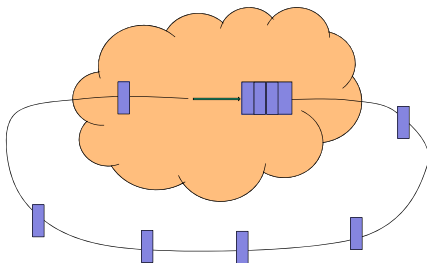
$$W = XT$$



$$X = \frac{N}{t} \quad T = \frac{Area}{N} \quad W = \frac{time \cdot Area}{t}$$

34

Estimating Number of Packets in the Queue



35

TCP/Vegas CA algorithm

$$T = T_{prop} + T_{queueing}$$

Applying Little's Law:

$$x_{vegas} T = x_{vegas} T_{prop} + x_{vegas} T_{queueing},$$

where $x_{vegas} = W/T$ is the sending rate

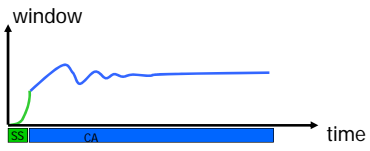
Then number of packets in the queue is

$$\begin{aligned} x_{vegas} T_{queueing} &= x_{vegas} T - x_{vegas} T_{prop} \\ &= W - W/T T_{prop} \end{aligned}$$

36

TCP/Vegas CA algorithm

maintain a *constant* number of packets in the bottleneck buffer



```
for every RTT
{
  if  $W - W/RTT_{min} < \alpha$  then W ++
  if  $W - W/RTT_{min} > \alpha$  then W --
}
for every loss
  W := W/2
```

queue size

37