15-441 Computer Networks
Homework 3

Due: April 15, 2008, 1:30 PM
Lead TA: Xin Zhang (xzhang1@cs.cmu.edu)

April 8, 2008

A How does TCP update its window

At time t, a TCP connection has a congestion window of 4000 bytes. The maximum segment size used by the connection is 1000 bytes. What is the congestion window after it sends out 4 packets and receives acks for all of them? Suppose there is one ack per packet.

(a) If the connection is in slow-start?

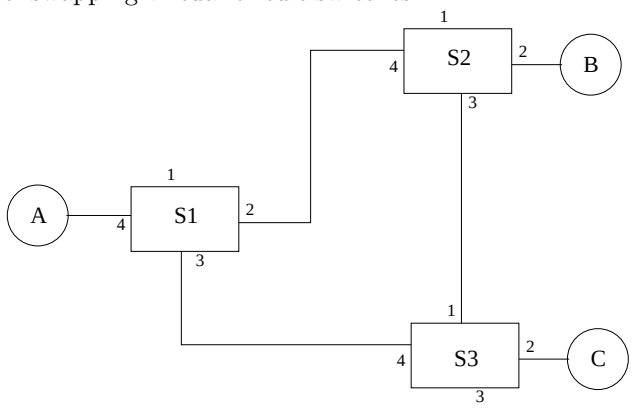
> **Solution:** 8000 bytes. In slow start, the sender increases its window for each byte successfully received.

(b) If the connection is in congestion avoidance (linear mode)?

> **Solution:** 5000 bytes. The sender increases its window by one segment each window.

B Label Switching

You are trying to debug a problem with your company's virtual circuit-based network. A diagram of the network is shown below. A, B, and C are hosts attached to the network. S1, S2, and S3 are switches configured to act as label swapping virtual circuit switches.



The label swapping tables for the switches are configured as follows. Some of the entries are stale and not actually in use right now.

Switch	Input Port	Input Label	Output Port	Output Label
S1	2	2	3	4
S1	4	2	3	1
S1	4	17	2	2
S2	2	19	4	2
S2	3	1	2	19
S2	3	2	2	15
S2	3	5	4	2
S2	4	2	2	1
S2	4	1	4	1
S3	2	1	1	2
S3	2	2	4	5
S3	4	1	1	1
S3	4	4	1	5

Write the sequence of (Switch, Input Port, Input Label) tuples and the destination node and label for each of these packets. We've given you the start node and starting label. The intermediate tuples should look like (S1, 1, 999) [e.g., switch S1, input port 1, label 999].

(a) Start node A, label 17. Switch tuples:

Solution: (S1, 4, 17) (S2, 4, 2)

Dest node and final label:

Solution: B, 1

(b) Start node A, label 2.

Switch tuples:

Solution: (S1,4,2) (S3,4,1) (S2,3,1)

Dest node and final label:

Solution: B,19

(c) Start node C, label 1.

Switch tuples:

Solution: (S3,2,1) (S2,3,2)

Dest node and final label:

Solution: B,15

C Why UDP vs TCP?

(a) Give **one** reason that DNS lookups are run over UDP rather than TCP:

Solution: OK: Connection-setup overhead, short-duration interaction NOT OK: Header overhead

(b) Give **one** reason that streaming multimedia is run over UDP rather than TCP:

Solution: OK: Variable delays from reliability, loss tolerant applications, drastic congestion control NOT OK: connection setup overhead

D Pick the true choices about congestion collapse and backoff

Otto Pilot creates a new network for the 150 PC computers he mounted within his car. Each computer sends indepenent UDP query/response packets to the other computers in the car when it needs to know or do something. Requests are retried after a time out that is a fixed, small multiple of the typial response time. After running the OttoNet for a few days, Otto notices that network congestion occasionally causes a congestion collapse because too many packets are sent into the network, only to be dropped before reaching the eventual destination. These packets consume valuable resources.

Suppose each response or request can be fit into one packet. Which of the following techniques is likely to reduce the likelihood of a congestion collapse? (Circle ALL that apply)

A. Increase the size of the queue in each router from 4 packets to 8 packets. Suppose the timout value is appropriately adjusted accordingly to the queue length.

> **Solution:** Yes. There are two possibilities for the timeout value. First, suppose that Ben used the answer to question 9 to set the timeout. Given a fixed timeout, lengthening queues would increase, not decrease, the chance of congestion collapse. The longer queues may cause clients to time out and resend their request packets, even though a response may already be on its way back. Second, suppose that Ben adjusted the timeout for the longer queues. Doubling queue lengths certainly doesnt prevent congestion collapse, because congestion collapse can occur with queues of any length. There is no a priori reason to believe that it is less likely with 8-packet queues than with 4-packet queues. Increasing the size of the queue to 8 packets might have a positive effect: some packets that would otherwise have been dropped might eventually reach their destination. However, it might also have a negative effect: packets that would otherwise have been dropped remain in the system and may cause congestion elsewhere.

B. Use exponential backoff in the timeout mechanism while retrying queries.

> **Solution:** YES. Exponential backoff reduces the injection rate of packets to a level that the network can tolerate.

C. If a query is not answered within a timeout interval, multiplicatively reduce the maximum rate at which the client application sends OttoNet query packets.

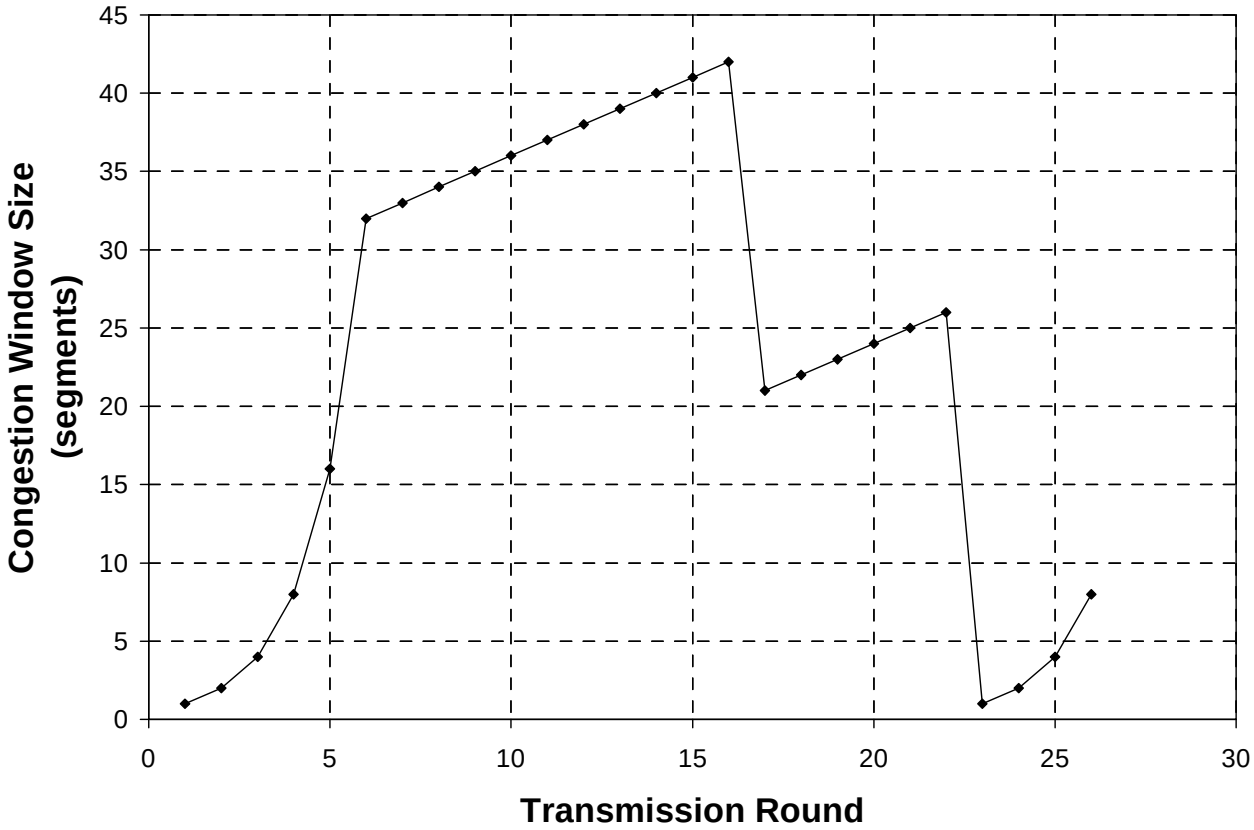
> **Solution:** YES. If this question had said "current" rather than "maximum" rate, it would have exactly been exponential backoff. Reducing the maximum rate eventually produces the same end result.

D. Use a TCP style flow control window (per session) at each receiver to prevent buffer overruns.

> **Solution:** NO. Flow control windows apply to streams of data. OttoNet requests are not streams, they are independent packets, each one of which may be delivered to a different server, so a flow control window is not applicable. Moreover, flow control is an end-to-end mechanism to ensure that a slow receivers buffers dont get overwritten by a fast sender. But the problem states that the server and client processing are both infinitely fast, so adding flow control would not accomplish anything.

E Congestion Window

Consider the following plot of TCP window size as a function of time:



Assuming TCP Reno is the protocol experiencing the behavior shown above, answer the following questions.

(a) Identify the intervals of time when TCP slow start is operating.

Solution: 1-6, 23-26

(b) Identify the intervals of time when TCP congestion avoidance is operating (AIMD).

Solution: 6-23

(c) After the 16th transmission round, is segment loss detected by a triple duplicate ACK or by a timeout?

Solution: dupack

(d) What is the initial value of ssthreshold at the first transmission round?

Solution: 32

(e) What is the value of ssthreshold at the 18th transmission round?

Solution: 21

(f) What is the value of ssthreshold at the 24th transmission round?

Solution: 13

(g) During what transmission round is the 70th segment sent?

Solution: 7

(h) Assuming a packet loss is detected after the 26th round by the receipt of a triple duplicate ACK, what will be the values of the congesion-window size and of ssthreshold?

Solution: 4,4

F Buffers, Losses, and TCP

Harry Bovik is given the responsibility of configuring the packet queuing component of a new router. The link speed of the router is 100 Mbit/s and he expects the average Internet round-trip time of connections through the router to be 80ms. Harry realizes that he needs to size the buffers appropriately.

You should assume the following:

- You're dealing with exactly one TCP connection.
- The source is a long-running TCP connection implementing additive-increase (increase window size by 1 packet after an entire window has been transmitted) and multiplicative-decrease (factor-of-two window reduction on congestion).
- The advertised window is always much larger than the congestion window.
- The loss recovery is perfect and has no impact on performance.
- The overhead due to headers can be ignored.

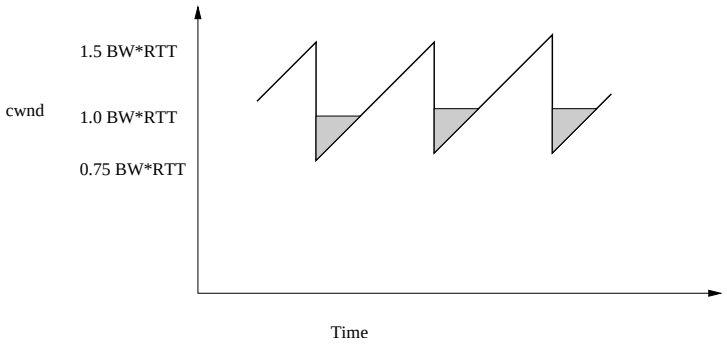
Harry argues that because the average RTT is 80ms, the average one-way delay is 40ms. Therefore, the amount of buffering he needs for high link utilization is 100 Mbit/s * 40 ms or 500 KBytes.

Approximately what bandwidth will TCP achieve with this buffering?

Note: There are two approaches to solving this problem. The easiest way to solve it is to assume that queueing delay does *not* increase the RTT. A more complete analysis will take queueing delay when the router buffers are over-full into account. The non-queueing-delay analysis will receive 80% of the points for this problem, so if you're stuck, it's much better to do the linear analysis than nothing, but we encourage you to do the full analysis. It's got some summations, but doesn't actually require computing an integral

Solution: 10 points.

The buffering is not enough for TCP to fill the link. The window size with this buffering will go between $1.5BW * RTTand0.75BW * RTT$.



The above graph shows the rough behavior of the congestion window over time. The overly simplified analysis:

During $\frac{1}{3}$ of the connection, the window will be too small, using an average of $(0.75+1)/2 = \frac{7}{8}$ of the capacity. During the other two thirds of the time, it will fill the link. The total utilization will be $\frac{23}{24}$ of the capacity, or about 95.8 Mbit/sec.

This analysis is overly simple, because it doesn't take into account the fact that the RTT increases by one packet's worth each RTT when we're sending a larger window than the buffer can handle. Exact answer:

In going from a window of 750 packets to a window of 1500 packets, the source sends:

$$\sum_{i=750}^{1500} i$$

packets. During this same amount of wall-clock time, an optimal scheme could have transmitted

$$\sum_{i=750}^{1500} i + \sum_{i=1}^{249} i$$

packets. (The missing packets from the underutilized periods). Or:

$$\begin{aligned}
\text{throughput} &= \frac{\sum_{i=750}^{1500}}{\sum_{i=750}^{1500} + \sum_{i=1}^{249}} \\
&= \frac{1}{1 + \frac{\sum_{i=1}^{249}}{\sum_{i=750}^{1500}}} \\
&= \frac{1}{1 + \frac{\frac{249*250}{2}}{\frac{1500*1501-749*750}{2}}} \\
&= \frac{1}{1 + \frac{31125}{844875}} = 0.9645 = 96.45 \text{ Mbit/sec}
\end{aligned}$$