CSE 473 – Introduction to Computer Networks *Roch Guérin*

Midterm Exam

Your Name: *10/14/2014*

1) (**5 points**). A user in St. Louis is connected to the Internet via a 4 Mbps ($4x10^6$ bits/sec) DSL link (download speed) and is retrieving a webpage from a server in China. The page is 125 kbytes and contains references to ten (10) images that are each 250 kbytes. The one-way propagation delay is 50 ms and the DSL link is the bandwidth bottleneck for this connection.

   (2 points) Approximately how long does it take for the page (including images) to appear on the user's screen, assuming she uses a single persistent HTTP connection to access the server (queueing and transmission delays on links other than the DSL access link are negligible, and you can ignore the impact of TCP's ramp-up)?

   Total download time is three RTTs (TCP connection, request of main page, requests for images), plus the transmission times of the main page and the 10 images, i.e.,

   3(100) ms + (125,000*8 + 10*250,000*8)/(4,000,000) =300 ms + 5.25s = 5.55 seconds*

   (2 points) How long would it take if the user's browser used instead two persistent HTTP connections (the two connections are opened in parallel; one connection is used to request the page, and the two connections would then each request five images)?

   There would be no difference. Each connection incurs a penalty of 3 RTTs, and the total transmission time on the DSL access link is unchanged.

   (1 point) Approximately how big (in bytes) should the buffer at the access router connected to the user's DSL link be to ensure there are no packet losses (assume that packets arrive from the Internet much faster than the 4 Mbps download speed of the DSL link)?

   *The buffer needs to be able to hold the 10 images arriving from the server (note that the main page must have been fully transmitted and received by the user's browser for requests for images to be sent, and it must therefore have cleared the buffer by the time the images arrive). This translates into a total buffer space of 10*250 kbytes = 2.5 Mbytes.*

2) (**10 points**). A private network that uses the 10.1.0.0/16 private address space sits behind a NAT router (NAT1) that has been allocated the public IP address 45.6.7.1. Assume that a local host in that network with address 10.1.10.1 is accessing a remote web server with public address 194.34.45.67. Headers of local packets originating from host 10.1.10.1 and destined for the web server have the following format in the private network behind NAT1:

<src_addr>	<dst_ addr>	<src_port>	<dest_port>
10.1.10.1	194.34.45.67	4567	80

The NAT (NAT1) already has the following entries:

Internal: <local_addr><local_port>	**External**: <external_port>
<10.1.11.2><4567>	<4567>
<10.1.10.1><3333>	<3333>
<10.1.10.1><4444>	<5555>

(3 points) What is (i) a possible entry in the forwarding table of NAT1 to accommodate the connection from 10.1.10.1 to web server 194.34.45.67, and (ii) what would then be the headers of packets from this connection when forwarded by NAT1 into the public Internet?

Because outgoing port <4567> is already in use, the NAT needs to assign a different unused outgoing port to the connection. A possible new entry in the NAT would be of the form

Internal: <*local_addr*><*local_port*>	***External***: <*external_port*>
<10.1.10.1><4567>	*<5678>*

Correspondingly, the headers of packets forwarded by the NAT in the public Internet would be of the form

<src_addr>	*<dst_ addr>*	*<src_port>*	*<dest_port>*
45.6.7.1	*194.34.45.67*	*5678*	*80*

(2 points) Assume now that host 10.1.10.1 decides to open a second parallel connection to the web server. Packet headers for this connection are of the following form in the private network behind NAT1:

<src_addr>	<dst_ addr>	<src_port>	<dest_port>
10.1.10.1	194.34.45.67	4568	80

Does NAT1 need to create a new entry for this connection, or can it reuse the previous one? Justify your answer.

Because the connection uses a different port number (4568), the NAT is forced to create a new entry in its table.

Host 10.1.10.1 is also engaged in a videoconference with another host in a different private network behind another NAT (NAT2). The private address of that other host is 10.2.2.2. Packet headers for the connection from host 10.1.10.1. to host 10.2.2.2 have the following formats on the internal network of host 10.1.10.1 (internal network 1), in the public Internet, and in the internal network of host 10.2.2.2 (internal network 2):

Packet headers	<src_addr>	<dst_ addr>	<src_port>	<dest_port>
Internal Network 1	10.1.10.1	53.3.4.7	3456	2345
Public Internet	45.6.7.1	53.3.4.7	5889	2345
Internal Network 2	45.6.7.1	10.2.2.2	5889	1345

(3 points) Identify the entries for this connection in the forwarding table of the two NATs?

The corresponding entry in NAT1 is of the form

Internal: <*local_addr*><*local_port*>	***External***: <*external_port*>
<*10.1.10.1*><*3456*>	<*5889*>

and conversely, the entry in NAT2 is of the form

Internal: <*local _addr*><*local_port*>	***External***: <*external_port*>
<*10.2.2.2*><*1345*>	<*2345*>

(2 points) How would the header of a packet for the return connection (from 10.2.2.2 to 10.1.10.1) look like when traversing the public Internet?

The header of packets for the return connection would be of the following form when traversing the public Internet

<*src_addr*>	<*dst_ addr*>	<*src_port*>	<*dest_port*>
53.3.4.7	*45.6.7.1*	*2345*	*5889*

3) (**10 points**) Consider a circular DHT with 64 nodes numbered 0, 1,..., 63. Node *i* handles keys with hash values in the range $i \times 2^{28}$ to $(i+1) \times 2^{28}-1$, and has routes to nodes *i*+1, *i*+4, *i*+8 and *i*+32 (where addition is modulo 64).

(5 points) What is the maximum number of hops required to get from one server to another in this DHT?

Each node proceeds greedily jumping to the next node that brings it closest to the target without overshooting, under the constraint that it can only progress in jumps of 1, 4, 8 or 32 hops. It is easy to enumerate possible options for this scenario given a client query that originally arrives at node i.

If the key is located in node i+3, 3 one-hop jumps are required to node i+1, i+2 and i+3. If the key is in node i+7, 4 jumps are required, first to i+4 and then three one-hop jumps as in the previous case. If the key is in node i+31, the query proceeds to nodes i+8, i+16, i+24, i+28, i+29, i+30 and i+31 for a total of 7 hops. Finally, if the key is in node i+63, the query proceeds to node i+32, i+40, i+48, i+56, i+60, i+61, i+62 and i+63 for a total of 8 hops, which is therefore the maximum number of hops a query will require.

(2 points) Suppose that node 32 receives a *get* request with a key string of "Led Zeppelin" that *hashes* to 8,321,499,137. How many hops will this request go through?

$8,321,499,137=(31*2^{28})+1$, which implies that the key hashes to a value in the range belonging to node 31, the immediate predecessor of node 32 that receives the query so that this belongs to the scenario where the key is in node 31=32+63 mod 64. Hence, according to the reasoning of the previous question, the query will be resolved in 8 hops.

(3 points) Assume now that nodes can cache key-value pairs. Suppose node 4 cached the key-value pair for the key "Led Zeppelin" but is the only node to have done so, *i.e.,* the only other node that has the key-value pair is the node that has the hash value of the key in its range. Assuming that the next query for key "Led Zeppelin" arrives at a randomly selected node, what are the odds that node 4 will be handling it, *i.e.,* what are the odds that the next query for "Led Zeppelin" is handled by node 4?

To answer the question, we need to identify the starting nodes for which node 4 is on the "shortest path" to node 31. Given that nodes greedily forward the query towards node 31, node 4 is on the path only if the query originally arrives at either node 4 or node 36. Hence, the odds that node 4 handles the next query is 2/64 = 0.03125.

4) (**10 points**) Two hosts, A and B, are connected by a direct point-to-point link of speed 100 Mbps (10^8 bits/sec), and rely on a basic go-back-N protocol to ensure reliable transmissions between them. The link can lose/corrupt packets, but not reorder them. A and B use 1,250 bytes packets, including header and payload, and as usual the protocol uses a single shared timer. The window size is **61 packets**.

(3 points) What is the maximum possible RTT between A and B that will allow communication at the maximum rate of 100 Mbps in the absence of packet errors or losses? For simplicity, assume that the RTT measures the time between the transmission of the **last** bit of a packet until the ACK for that packet is received.

*It takes 60*1,250*8/10^8 = 6 ms to transmit the sixty 1,000 bytes packets (packets 2 to 61) allowed after the transmission of packet 1. In order for transmission to be able to continue uninterrupted, the ACK for packet 1 must, therefore, have been received by that time. Hence, 6 ms is the maximum RTT that the connection can tolerate and still be able to communicate at 100 Mbps in the absence of packet errors or losses.*

(4 points) Assume from now on an RTT of 10 ms (5 ms each way) and a time-out value of 11 ms. Consider a scenario where at time t = 0, A starts transmitting packets 1 to 50, packet 1 is lost, so that A eventually retransmits packets 1 to 50. Packet 50 is then lost during this retransmission. When will packet 50 be eventually received at B (assume that no ACKs are lost)?

*The loss of packet 1 is detected at t = 11 ms because of a time-out. This triggers the retransmission of all 50 packets. The ACKs for packets 1 to 49 arrive 10 ms after A transmits their last bit and are each 0.1 ms apart. Hence, the ACK for packet 49 arrives at t=11+10+49*0.1=25.9 ms. The receipt of the ACK for packet 49 restarts the time-out timer that will, therefore, expire 11 ms later at t=36.9 ms. This triggers the retransmission of packet 50 with the packet's last bit leaving A at time t=37 ms to finally arrive at B 5 ms later at t = 42 ms.*

(3 points) Assume next that at time t the state at A is as follows: sendBase = 25, nextSeqNum = 63, timerExpiration = 5 ms, where sendBase is the sequence number of the oldest unacknowledged packet, nextSeqNum is the sequence number to be used in the next packet transmission, and timerExpiration tracks how much time is left before the time-out timer expires. An ACK with sequence number 32 is received at t + 1 ms, and at t + 7 ms the application at A writes five (5) packets worth of new payload. What is the state at A, *i.e.*, values of sendBase, nextSeqNum, and timerExpiration, at time t + 9 ms, and what packets are still in A's re-send buffer?

The receipt of an ACK with sequence number 32 at t + 1 ms shifts sendBase to 33 and restarts the time-out timer, so that its value is now timerExpiration = 11 ms. Given that the window size is 60 and that sendBase = 33, the legal sendWindow of A extends from sequence number 33 to sequence number 92. Since nextSeqNum is 63, A is able to send another 30 packets, which is sufficient to allow it to transmit the five packets needed to carry the new payload written by the application. Hence at time t + 9 ms, the state at A is as follows: sendBase = 33, nextSeqNum = 68, and timerExpiration = 3 ms.

Additionally, packets 33 to 67 are still in A's resend buffer.

5) (**15 points**) Consider a TCP sender that sends MSS size packets with MSS = 1000 bytes. The sender needs to upload a file whose size corresponds to the aggregate payload of a total of 2048 MSS size packets. The sender starts packet transmissions at $t = 0$ (after the TCP handshakes completes) with *ssthresh = rcvWindow = 64 MSS*. The sender is connected to the Internet through a 10 Gbps (10^{10} bits/sec) link, but the bottleneck link on the connection's path is 100 Mbps (10^8 bits/sec). The connection's RTT is mostly constant at 40 ms, *i.e.*, *EstimatedRTT* ≈ 40 ms and *DevRTT* ≈ 0 ms. Assume that the TCP connection does not use delayed ACKs.

(5 points) How long approximately will it take the sender to upload half the file, *i.e.*, transmit 1024 MSS size packets, in the absence of packet losses?

*The sender enters slow-start with cwnd = 1 MSS at t = 0, and the slow-start period lasts until cwnd = ssthresh with cwnd increasing by one packet for each ACK received. cwnd therefore increases as follows during successive cycles of duration 1 RTT each: 1, 2, 4, 8, 16, 32, 64. Hence, the sender exits slow-start approximately six (6) RTTs later, and during that time transmitted 127 packets. It subsequently transmits 64 MSS size packets in each RTT. As a result, after 20 RTTs it will have transmitted 127+14*64 = 1023 MSS size packets. The sender receives another ACK at the start of the 21st RTT cycle, at which point it is able to transmit the 1024th packet. So it takes approximately 20 RTTs (plus one packet transmission time) or 800 ms for the sender to upload half the file.*

(6 points) The sender uses TCP Reno. Assume that the connection experienced no losses until packet number 1984 that is lost at time *t*. At that time, *cwnd* had long reached it maximum value. How long approximately will it take from *t* onward before the sender knows that the entire file has been successfully uploaded to the receiver, *i.e.*, receives an ACK acknowledging the last packet? For simplicity, assume that the lost packet 1984 is the first to be transmitted in a new batch (window) of *cwnd* packets sent by the sender.

Packet 1984 is actually the first packet to be transmitted in the 36th RTT of the connection. Because packet 1984 is lost, the receiver will generate duplicate ACKs when receiving packets 1985, 1986, and 1987. The sender receives them approximately one RTT after transmitting packet 1984 (one RTT plus three packet transmissions). This then triggers the retransmission of packet 1984 and the sender enters fast recovery. Fast recovery will last for about one RTT (until the receipt of the retransmitted packet 1984 is acked) during which time the sender will be able to send an additional 31 packets beyond those that were sent in the same RTT as packet 1984, i.e., packet 1985 to 2047. Since the file only requires 2048 packets, the sender only needs to send one more packet in addition to retransmitting packet 1984. Duplicate ACKs triggered by packets 1985 to 2047 arrive spaced 80 µsecs apart. The sender can transmit a new packet when the duplicate ACK for packet 1984 + 32 = 2016 arrives, i.e., after 2.56 ms. Hence, the ACK for this last packet arrives 2.56 ms after the ACK for the retransmitted packet 1984. So it takes approximately two (2) RTTs + 2.56 ms, i.e., 82.56 ms, after the loss occurred for the sender to know that the entire file has been successfully uploaded to the receiver.

(4 points) Assume next that the sender is using TCP Tahoe instead of TCP Reno. How does this change the answer?

The main difference between TCP Tahoe and Reno is that Tahoe does not implement fast recovery, i.e., it enters slow-start after detecting a loss. This adds one RTT to the transmission of the next packet after the retransmitted packet (the ACK for the retransmitted packet acknowledges all packets up to packet 2047 and allows the transmission of packet 2048, whose ACK is then received one RTT later), i.e., the sender knows that the file has been fully received 120 ms after the loss occurred.

6) **(15 points)** A sender uses TCP Reno (without delayed ACKs) with *cwnd* = *rcvWindow* = 64 kbytes (2^{16} bytes), an MSS of 1 kbytes (2^{10} bytes), an RTT of 104.858 ms, and is transmitting over a path with a bottleneck link of speed 10 Mbps (10^7 bits/sec). The router connected to that link has a very small buffer, so that it will lose packets as soon as the aggregate incoming transmission rate barely exceeds the link capacity.

(5 points) How many parallel connections can the sender safely open to increase its aggregate transmission rate without incurring losses at the bottleneck link? You can assume that the receiver has enough memory to allocate a rcvBuffer of 64 kbytes to each new connection.

*Assuming cwnd = 64 kbytes and RTT = 104.858 ms, each connection has as rate of 2^{16}*8/0.104858 = 5 Mbits/sec. Hence, the sender can open two parallel connections to realize an aggregate transmission rate of 10 Mbits/sec without incurring losses.*

(10 points) Assume that the sender decides to open 4 parallel connections. What will approximately be the steady-state aggregate transmission rate it manages to realize? Note that if connections experience losses, you need to identify what their overall rate would be, accounting for the up and down pattern of transmission rate the connection would experience

The maximum possible rate that each connection can realize without causing losses is 2.5 Mbits/sec, which correspond to cwnd = 32 kbytes, which they will therefore repeatedly exceed. Since all connections have the same RTT, TCP's fairness property ensures that in steady state the 4 connections achieve the same rate, and alternate through similar patterns of rate increase followed by losses and consequently rate decrease. Since losses occur as soon as the aggregate rate exceeds 10 Mbits/sec, this implies that when each connection just exceeds 2.5 Mbits/sec, it experiences a loss and drops back to half that rate, i.e., cwnd = 16 kbytes. Since the sender is using TCP Reno, the connections do not enter slow-start and progressively increase their rate back up. It takes each connection approximately 16 RTTs to increase their cwnd back up to 32 kbytes, at which point, i.e., in the next RTT, losses occur again and the cycle repeats.

*During one cycle of duration 17 RTTs, each connection transmits 16+17+18+…+31+32+32 =440 MSS or 3,604,480 bits over a period of 18*104.858 ms = 1.887 secs, or a throughput of just under 2 Mbits/sec (1.909 Mbits/sec). This translates into an aggregate throughput of about 7.639 Mbits/sec, which is below the maximum possible rate the sender could have achieved by opening fewer connections.*