Homework 2 assignment for ECE374

Posted: 02/21/14

Due: 02/28/14

Note: *In all written assignments, please show as much of your work as you can. Even if you get a wrong answer, you can get partial credit if you show your work*. If you make a mistake, it will also help the grader show you where you made a mistake.

Problem 1: (5 Points)

Suppose within your Web browser you click on a link to obtain a web page. The IP address for the associated URL is not cached in your local host, so a DNS lookup is necessary to obtain the IP address. Suppose that *n* DNS servers are visited before your host receives the IP address from DNS; the successive visits incur an RTT of RTT_1, ..., RTT_n. Further suppose that the web page associated with the link contains exactly one object, consisting of a small amount of HTML text. Let RTT_0 denote the RTT between the local host and the server containing the object.

a. Assuming zero transmission time of the object, how much time elapses from when the client clicks on the link until the client receives the object?

Now assume the HTML file references eight very small objects on the same server. Neglecting transmission times, how much time elapses with

b. Non-persistent HTTP with no parallel TCP connections?
c. Non-persistent HTTP with the browser configured for 5 parallel connections?
d. Persistent HTTP?

Solution:

a. The total amount of time to get the IP address is $RTT_1 + RTT_2 + \cdots + RTT_n$. Once the IP address is known, RTT_O elapses to set up the TCP connection and another RTT_O elapses to request and receive the small object. The total response time is $2RTT_o + RTT_1 + RTT_2 + \cdots + RTT_n$

b. $RTT_1 + \cdots + RTT_n + 2RTT_o + 8 \cdot 2RTT_o = 18RTT_o + RTT_1 + \cdots + RTT_n$.

c. $RTT_1 + \cdots + RTT_n + 2RTT_o + 2 \cdot 2RTT_o = 6RTT_o + RTT_1 + \cdots + RTT_n$.

d. $RTT_1 + \cdots + RTT_n + 2RTT_o + RTT_o = 3RTT_o + RTT_1 + \cdots + RTT_n$.

Problem 2: (20 Points)

Consider the scenario shown in Figure 1 in which a server is connected to a router by a 100Mbps link with a 50ms propagation delay. Initially this router is also connected to two routers, each over a 25Mbps link with a 200ms propagation delay. A 1Gbps link connects a host and a cache (if present) to each of these routers and we assume that this link has 0 propagation delay. All packets in the network are 20,000 bits long.

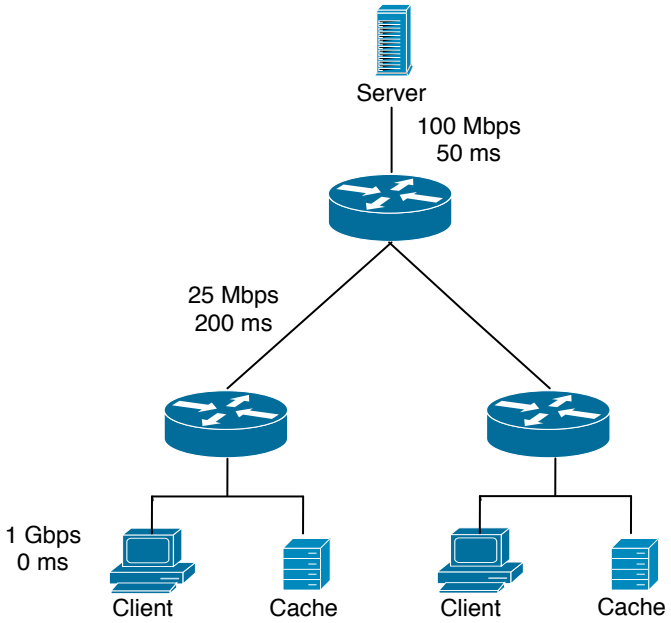


Figure 1

a. What is the end-to-end delay from when a packet is transmitted by the server to when it is received by the client? In this case, we assume there are no caches, there's no queuing delay at the routers, and the packet processing delays at routers and nodes are all 0.
b. Here we assume that client hosts send requests for files directly to the server (caches are not used or off in this case). What is the maximum rate at which the server can deliver data to a single client if we assume no other clients are making requests?
c. Again we assume only one active client but in this case the caches are on and behave like HTTP caches. A client's HTTP GET is always first directed to its local cache. 60% of the requests can be satisfied by the local cache. What is the average rate at which the client can receive data in this case?

Solution:

a. *Answer: If all packets are 20,000 bits long it takes 200 usec to send the packet over the 100Mbps link, 800 usec to send over the 25Mbps link, and 20 usec to send over the 1Gbps link. Sum of the three-link transmission is 1020 usec. Thus, the total end-to-end delay is 251.02 msec.*
b. *Answer: Server can send at the max of the bottleneck link: 25Mbps.*
c. *Answer: We assume that requests are serially satisfied. 40% of the requests can be delivered at 25Mbps and 60% at 1Gbps. So the average rate is 610Mbps.*

Problem 3: (15 Points)

In this problem, we use the useful dig tool available on Unix and Linux hosts to explore the hierarchy of DNS servers. Recall that in slide 67 in Chapter 2, a DNS server higher in the DNS hierarchy delegates a DNS query to a DNS server lower in the hierarchy, by sending back to the DNS client the name of that lower-level DNS

server. First read the man page for dig (e.g., http://linux.die.net/man/1/dig) , and then answer the following questions.

a. Starting with a root DNS server (from one of the root servers [a-m].root-servers.net], initiate a sequence of queries for the IP address for your department's Web server (www.ecs.umass.edu) by using *dig*. Show the list of the names of DNS servers in the delegation chain in answering your query.
b. Repeat part a) for several popular Web sites, such as google.com, yahoo.com, or amazon.com.

Back up your answers with screen shots that show the results of your *dig* queries.

Solution:

a)
The following delegation chain is used for gaia.cs.umass.edu
a.root-servers.net
E.GTLD-SERVERS.NET
ns1.umass.edu(authoritative)

First command:
dig +norecurse @a.root-servers.net any gaia.cs.umass.edu

;; AUTHORITY SECTION:

edu.	172800	IN	NS	E.GTLD-SERVERS.NET.
edu.	172800	IN	NS	A.GTLD-SERVERS.NET.
edu.	172800	IN	NS	G3.NSTLD.COM.
edu.	172800	IN	NS	D.GTLD-SERVERS.NET.
edu.	172800	IN	NS	H3.NSTLD.COM.
edu.	172800	IN	NS	L3.NSTLD.COM.
edu.	172800	IN	NS	M3.NSTLD.COM.
edu.	172800	IN	NS	C.GTLD-SERVERS.NET.

Among all returned edu DNS servers, we send a query to the first one.
dig +norecurse @E.GTLD-SERVERS.NET any gaia.cs.umass.edu

umass.edu.	172800	IN	NS	ns1.umass.edu.
umass.edu.	172800	IN	NS	ns2.umass.edu.
umass.edu.	172800	IN	NS	ns3.umass.edu.

Among all three returned authoritative DNS servers, we send a query to the first one.
dig +norecurse @ns1.umass.edu any gaia.cs.umass.edu

gaia.cs.umass.edu.	21600	IN	A	128.119.245.12

b) The answer for google.com could be:

a.root-servers.net
E.GTLD-SERVERS.NET
ns1.google.com(authoritative)

Problem 4: (20 Points)
Consider a short, 10-meter link, over which a sender can transmit at a rate of 150 bits/sec in both directions. Suppose that packets containing data are 100,000 bits long, and packets containing control (e.g., ACK or handshaking) are 200 bits long. Assume that *N* parallel connections each get *1/N* of the link bandwidth. Now consider the HTTP protocol, and suppose that each downloaded object is 100 Kbits long, and that the initial downloaded object contains 10 referenced objects from the same sender. Would parallel downloads via parallel instances of non-persistent HTTP make sense in this case? Now consider persistent HTTP. Do you expect significant gains over the non-persistent case? Justify and explain your answer.

Solution:
Note that each downloaded object can be completely put into one data packet. Let Tp denote the one-way propagation delay between the client and the server.

First consider parallel downloads using non-persistent connections. Parallel downloads would allow 10 connections to share the 150 bits/sec bandwidth, giving each just 15 bits/sec. Thus, the total time needed to receive all objects is given by:

$$(200/150+Tp + 200/150 +Tp + 200/150+Tp + 100{,}000/150+ Tp)$$
$$+ (200/(150/10)+Tp + 200/(150/10) +Tp + 200/(150/10)+Tp + 100{,}000/(150/10)+ Tp)$$
$$= 7377 + 8*Tp \text{ (seconds)}$$

Now consider a persistent HTTP connection. The total time needed is given by:

$$(200/150+Tp + 200/150 +Tp + 200/150+Tp + 100{,}000/150+ Tp)$$
$$+ 10*(200/150+Tp + 100{,}000/150+ Tp)$$
$$=7351 + 24*Tp \text{ (seconds)}$$

Assuming the speed of light is $300*10^6$ m/sec, then $Tp=10/(300*10^6)=0.03$ microsec. Tp is therefore negligible compared with transmission delay.

Thus, we see that persistent HTTP is not significantly faster (less than 1 percent) than the non-persistent case with parallel download.

Problem 5: (20 Points)
In this problem, we consider the performance of HTTP, comparing non-persistent HTTP with persistent HTTP. Suppose the page your browser wants to download is 500K bits long, and contains 5 embedded images (with file names img01.jpg, img02.jpg, …, img05.jpg), each of which is also 100K bits in length. The page and the 5 images are all stored on the same server, which has a 250 msec RTT from the

client that hosts your browser. We will abstract the network path between your browser and the web server as a 100Mbps link. You can assume that the time it takes to transmit a GET message into the path is zero, but you should account for the time it takes to transmit the base file and the embedded objects into the "link." This means that the server-to-client "link" has both a 125 ms one-way propagation delay, as well as a transmission delay associated with it; review slide 40 in Chapter 1 if you are uncertain about the difference between transmission delay and propagation delay. In your answer below, make sure to take into account the time needed to setup up TCP connections.

a. Assuming non-persistent HTTP (and assuming no parallel connections are open between the browser and server). How long is the response time - the time from when the user requests the URL to the point in time when the page and its embedded objects are displayed? Make sure you describe the various components that contribute to this delay.
b. Again assume non-persistent HTTP, but now assume that the browser can open as many parallel TCP connections to the server as it wants. What is the response time in this case?
c. Now assume persistent HTTP (i.e., HTTP1.1). What is the response time, assuming no parallel connections?
d. Now suppose persistent HTTP with parallel connections is used. What is the response time?

Solution:

a. $T_{resp}=2*T_{RTT}+T_{HTML}+2*T_{RTT}+T_{JPG1}+2*T_{RTT}+T_{JPG2}+2*T_{RTT}+T_{JPG3}+2*T_{RTT}+T_{JPG4}+2*T_{RTT}+T_{JPG5}$
$=2*T_{RTT}+T_{HTML}+5*(2*T_{RTT}+T_{JPG})$
$=2*250msec+5msec+5*(2*250msec+1msec)$
$=505msec+2,505msec$
$=3,010msec$

b. $T_{resp}=2*T_{RTT}+T_{HTML}+2*T_{RTT}+T_{JPG}$
$=4* T_{RTT}+T_{HTML}+ T_{JPG}$
$=1000msec+5msec+1msec=1006msec$

c. $T_{resp}=2*T_{RTT}+T_{HTML}+1*T_{RTT}+T_{JPG1}+1*T_{RTT}+T_{JPG2}+1*T_{RTT}+T_{JPG3}+1*T_{RTT}+T_{JPG4}+1*T_{RTT}+T_{JPG5}$
$=2*T_{RTT}+T_{HTML}+5*(1*T_{RTT}+T_{JPG})$
$=2*250msec+5msec+5*(1*250msec+1msec)$
$=505msec+1,255msec$
$=1,760msec$

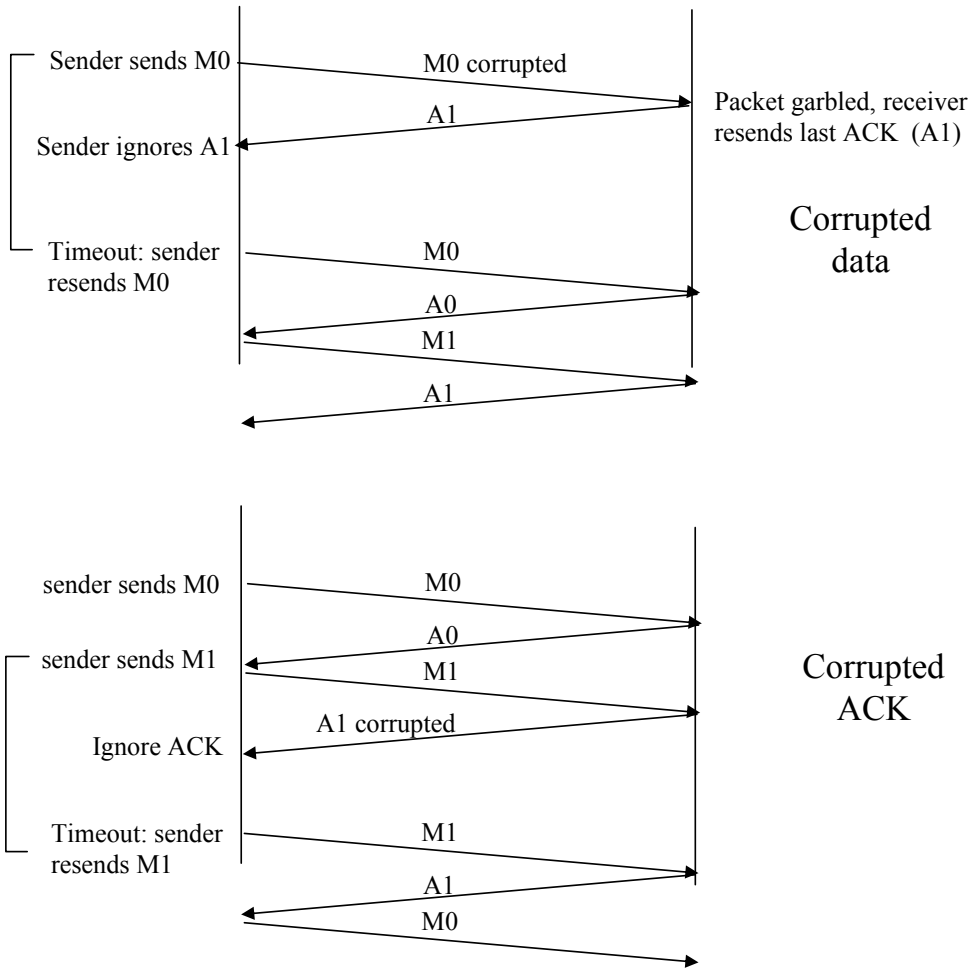
d. $T_{resp}=2*T_{RTT}+T_{HTML}+1*T_{RTT}+T_{JPG}$
$=3* T_{RTT}+T_{HTML}+ T_{JPG}$
$=756msec$

Problem 6: (20 Points)

a. In protocol rdt3.0, tha ACK packets flowing from the receiver to the sender do not have sequence numbers (although they do have an ACK field that contains the sequence number of the packets they are acknowledging). Why is it that our ACK packets do not require sequence numbers?
b. Draw the finite state machine for the receiver side of rdt3.0
c. Give a trace of the operation of protocol rdt3.0 when data packets and acknowledgement packets are garbled. Your trace should be similar to the ones used in Slide 3-39 and 3-40.

Solution:

a. To best answer this question, consider why we needed sequence numbers in the first place. We saw that the sender needs sequence numbers so that the receiver can tell if a data packet is a duplicate of an already received data packet. In the case of ACKs, the sender does not need this info (i.e., a sequence number on an ACK) to tell detect a duplicate ACK. A duplicate ACK is obvious to the rdt3.0 receiver, since when it has received the original ACK it transitioned to the next state. The duplicate ACK is not the ACK that the sender needs and hence is ignored by the rdt3.0 sender.
b. The sender side of protocol rdt3.0 differs from the sender side of protocol 2.2 in that timeouts have been added. We have seen that the introduction of timeouts adds the possibility of duplicate packets into the sender-to-receiver data stream. However, the receiver in protocol rdt.2.2 can already handle duplicate packets. (Receiver-side duplicates in rdt 2.2 would arise if the receiver sent an ACK that was lost, and the sender then retransmitted the old data). Hence the receiver in protocol rdt2.2 will also work as the receiver in protocol rdt 3.0.
c. Suppose the protocol has been in operation for some time. The sender is in state "Wait for call from above" (top left hand corner) and the receiver is in state "Wait for 0 from below". The scenarios for corrupted data and corrupted ACK are shown in Figure 1.


Sender sends M0
M0 corrupted
A1
Packet garbled, receiver resends last ACK (A1)
Sender ignores A1
Corrupted data
Timeout: sender resends M0
M0
A0
M1
A1
sender sends M0
M0
A0
sender sends M1
M1
A1 corrupted
Corrupted ACK
Ignore ACK
Timeout: sender resends M1
M1
A1
M0