CS 3251 Homework 2

Wenqi He

October 2, 2018

2.4

a. `gaia.cs.umass.edu/cs453/index.html`. From the `GET` line and the `Host` header.

b. HTTP 1.1. From the `GET` line.

c. Persistent. From the `Connection:keep-alive` header.

d. It's not in the HTTP message.

e. Netscape. From the `User-Agent` header. Because not all Web standards are fully supported by all browsers, and the source files intended for different devices (desktop, tablet, phones) might be maintained separately. The server can use this information to send appropriate files to different browsers.

2.5

a. Yes. (from the status code). 12:39:45 GMT on March 7, 2008 (from the `Date` header).

b. 18:27:46 GMT on December 10, 2005 (from the `Last-Modified` header).

c. 3874 bytes (from the `Content-Length` header).

d. `<!doc` (following the double cr-lf). Yes (from the `Connection:Keep-Alive` header).

2.10

Since the link is short, propagation delay is negligible. However, the link rate is extremely low, so even a small control packet can have a significant impact on transmission delay. For example, a three-way handshake to establish TCP connection would take $3 \cdot 200/150 = 4$ sec. If referenced objects are downloaded in parallel, each instance must establish a separate TCP connection, which is very time-consuming under this network condition. Furthermore, even if we ignore the time to establish connections, parallel instances wouldn't speed up download in this network, because all traffic goes through a single link with fixed finite bandwidth that is shared by everyone. The more instances there are, the lower the transmission rate is for each intance, and the overall transmission rate stays the same. Therefore it would always take the same amount of time to tranmit data, regardless of whether parallel download is used or not. On the other hand, if persistent HTTP is used, the client only needs to establish one TCP connection, which would reduce the transmission delay by as much as $10 \cdot 4 = 40$ sec.

2.19

a

For `cc.gatech.edu`:

```
a.root-servers.net -> f.edu-servers.net -> dns1.gatech.edu
```

b

For `google.com`:

```
b.root-servers.net -> c.gtld-servers.net -> ns2.google.com
```

For `yahoo.com`:

```
b.root-servers.net -> h.gtld-servers.net -> ns1.yahoo.com
```

For `amazon.com`:

```
c.root-servers.net -> f.gtld-servers.net -> pdns1.ultradns.net
```

2.22

	10	100	1000
300 Kbps	7500s	25000s	45454.55s
700 Kbps	7500s	15000s	20547.95s
2 Mbps	7500s	7500s	7500s

3.2

From server to process 1 on host C:
source port: 80, dest. port: 7532
source IP: B, dest. IP: C

From server to process 2 on host C:
source port: 80, dest. port: 26145
source IP: B, dest. IP: C

From server to process on host A:
source port: 80, dest. port: 26145
source IP: B, dest. IP: A

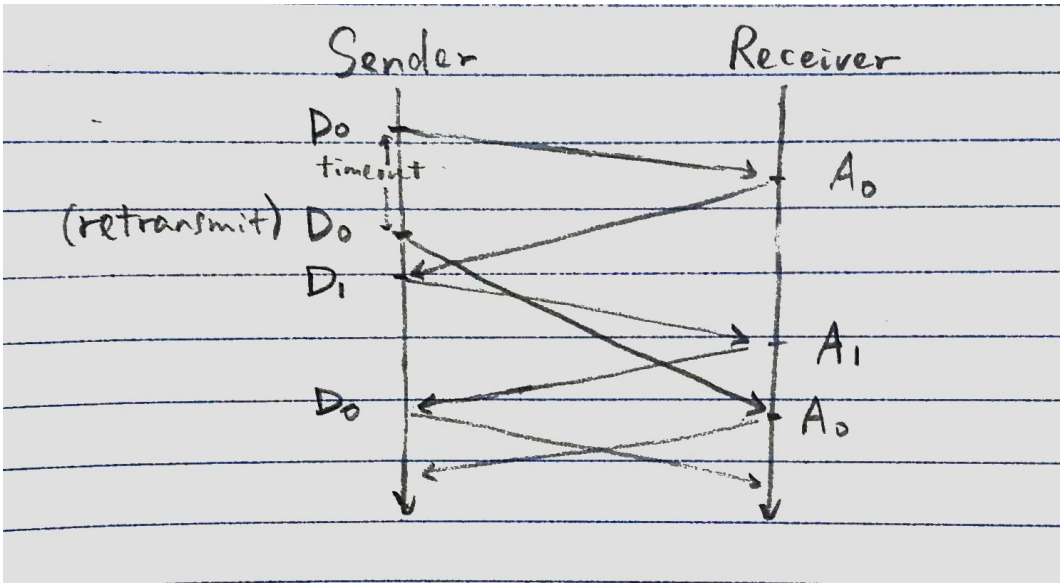
3.6

Suppose the receiver successfully receives a packet with sequence number 0, but the acknowledgement gets corrupted on its way back to the sender. The sender, according to rdt2.1, would resend packet 0, but the receiver is now expecting packet 1. According to Figure 3.57, upon receiving packet 0 it would respond with a NAK packet, which might be successfully received by the sender or also get corrupted. In either situation, the sender would resend packet 0 yet again. And the cycle continues.

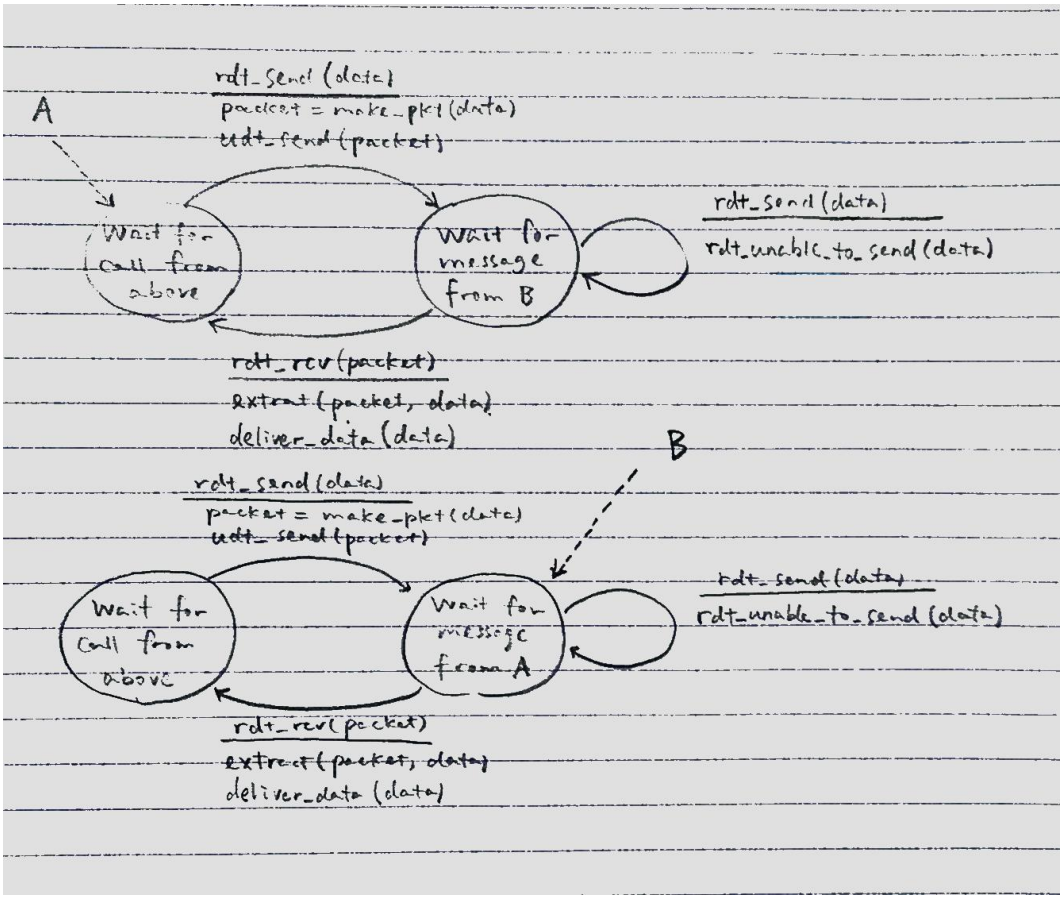
3.10

- Now the sender should start/restart a timer whenever a packet is sent/resent. (Whenever `udt_send(sndpkt)` is called.)
- When the sender is waiting for ACK/NAK, if timeout occurs, it should resend packet (call `udt_send(sndpkt)` again) and keep waiting. If it receives a corrupt packet or a NAK packet, instead of resending the packet immediately, it should simply ignore it, since the packet will eventually be retransmitted if its acknowledgement is not received before timeout.

3.13



3.17



3.23

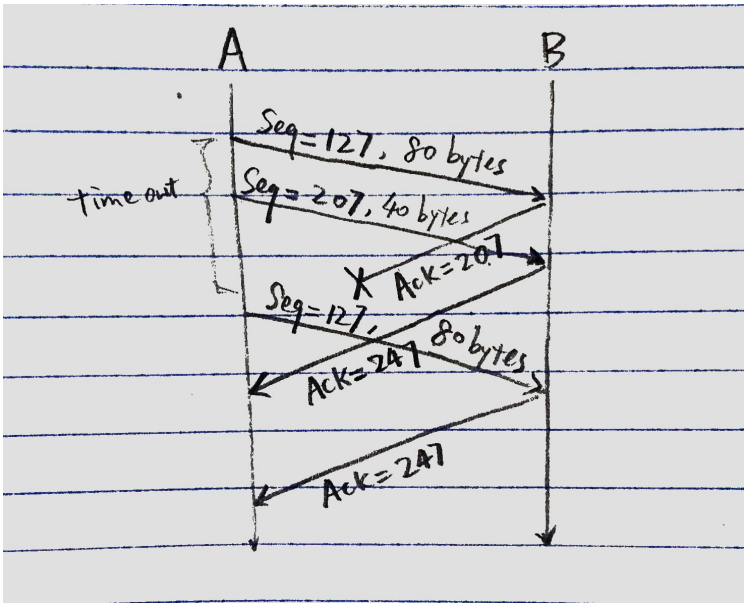
The largest allowable sender window size is $\lfloor \frac{k}{2} \rfloor$.

3.27

a. Sequence number: 207. Source port: 302. Destination port: 80.

b. Acknowledgement number: 207, Source por: 80. Destination port: 302.

c. Acknowledgement number: 127.



3.28

Host B will constantly inform Host A of how much spare room it has in its buffer (`rwnd`) through the receive window header in TCP packets. Meanwhile host A will keep track of `rwnd` and make sure the total size of unacknowledged packets is always less than `rwnd` so that it won't overflow B's buffer. Therefore, even though A has a much higher link rate, its sending speed will be throttled by TCP flow control in order to match B's receiving speed.