Lecture 9

Chapter 2:

P30

Advantage of byte-oriented communication over preserving message boundaries: many answers are possible. Examples are: when sending a large amount of data, the applications do not need to manage the division of the data into fragments and their reassembly; reordering is not an issue, since the receiving buffer will use the bytes sequence numbers to deliver them in the correct order to the application layer.

Advantage of communication preserving message boundaries over byte-oriented communication: many answers are possible. Examples are: more control on when the receiving application layer receives the data; never two "sends" will correspond to one "receive"; never one "send" will correspond to two "receives".

Chapter 3:

R8

For each persistent connection, the Web server creates a separate "connection socket". Each connection socket is identified with a four-tuple: (source IP address, source port number, destination IP address, destination port number). When host C receives and IP datagram, it examines these four fields in the datagram/segment to determine to which socket it should pass the payload of the TCP segment. Thus, the requests from A and B pass through different sockets. The identifier for both of these sockets has 80 for the destination port; however, the identifiers for these sockets have different values for source IP addresses. Unlike UDP, when the transport layer passes a TCP segment's payload to the application process, it does not specify the source IP address, as this is implicitly specified by the socket identifier.

R14

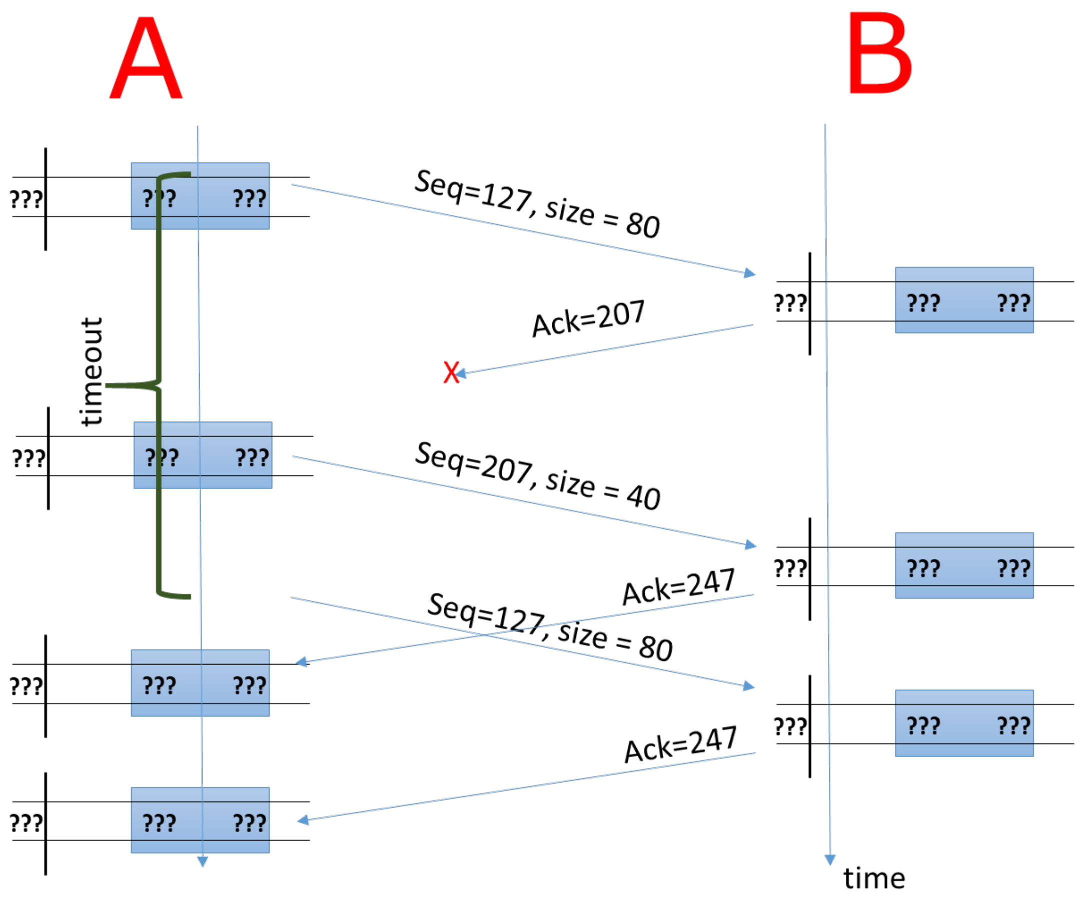
a) False. Host B can send acknowledgment fragments without data.
b) False.
c) True.
d) False. A fragment can contain more than one byte of payload.
e) True.
f) True.
g) False. It could be the ACK number for the packet B -> A, if everything goes fine.

R15

a) 20 bytes.
b) 90.

P27

a) Sequence number: 207. Source port: 302. Destination port: 80.
b) ACK number: 207. Source port: 80. Destination port: 302.
c) ACK number: 127.
d) Solution in the diagram. The buffers are provided only to have a uniform appearance with respect to the examples discussed in the lecture, but they are not carrying any information.



P44

a) 6 RTTs
b) (6+12) MSS / RTT

Practice problem 1: the server

```
const net = require('net');

htmldocument_with_http_headers = `HTTP/1.1 200 OK
Content-Type: text/html
Connection: close
Content-Length: 41

<html><body><h1>hello</h1></body></html>
`;

const server = net.createServer((socket) => {
  console.log('Connection from', socket.remoteAddress, 'port',
socket.remotePort);

socket.on('data', (buffer) => {
  console.log(buffer.toString('utf-8'));
  socket.end(htmldocument_with_http_headers);
  });
  socket.on('end', () => {
    console.log('Closed', socket.remoteAddress, 'port',
socket.remotePort);
  });
});

server.maxConnections = 1;
server.listen(9999)
```

Practice problem 2: the client.

```
const net = require('net');

http_request = `GET /index.html HTTP/1.1



`;


if (process.argv.length < 4) {console.log("I need host and 80, for
example 'google.com 80'"); process.exit(0);}


const client = new net.Socket();

client.connect({ port: process.argv[3], host: process.argv[2] });

client.on('data', (data) => {

  console.log("Received:\n" + data.toString('utf-8'));

});

client.end(http_request);
```

Usage: `node filename.js localhost 9999`