Sliding Window Protocol and Quiz 3 Problems
Quiz 3 Example

• Write a C code segment for a server that creates a socket, binds the socket and waits for connection requests. The server should accept connections on port 30000 on any IP address associated with a machine on which the server process runs
int sockfd1, sockfd2;
struct sockaddr_in server;
struct sockaddr_in client;

sockfd1 = socket(AF_INET, SOCK_STREAM, 0);
memset(&server, 0, sizeof(struct sockaddr_in));
server.sin_family = AF_INET;
server.sin_addr.s_addr = INADDR_ANY;
server.sin_port = htons(30000);

bind(sockfd1, (struct sockaddr *) &server, sizeof(struct sockaddr_in));
listen(sockfd1, 5);

sockfd2 = accept(sockfd1, (struct sockaddr *) &client, &clen);
Quiz 3 Example

- Suppose a server has been created that accepts pairs of integers from a client and returns the largest of the two integers. The server is listening on port 20000 at IP address 138.49.99.99. The integers are transmitted in binary format (not text). Write a C code segment for a client that creates a connection to the server, reads two integers from standard input, sends the integers to the server and prints to standard output the integer returned from the server.
Quiz 3 Example

```c
int sockfd;
struct sockaddr_in server;
int nums[2];
int max;

sockfd = socket(AF_INET, SOCK_STREAM, 0);

memset(&server, 0, sizeof(struct sockaddr_in));
server.sin_family = AF_INET;
server.sin_addr.s_addr = inet_addr("138.49.99.99");
server.sin_port = htons(20000);

connect(sockfd, (struct sockaddr *) &server, sizeof(struct sockaddr_in));
scanf("%d %d", &nums[0], &nums[1]);
nums[0] = htonl(nums[0]);
nums[1] = htonl(nums[1]);
send(sockfd, nums, 2*sizeof(int),0);
int len = recv(sockfd, &max, sizeof(int), 0);
printf("%d\n", ntohl(max));
```
Quiz 3 Example

• See utilization problems at the end of these slides
Sliding Window Protocol

• A communication protocol to create a reliable data transfer service on top of an unreliable data transfer service.

• Used at both the link layer and the transport layer

• We will look at the general behavior of the protocol and then look at the specifics of TCP
Sliding Window Terminology

- Stop-and-wait
- Go-back-N
- Selective repeat
- Sender Window Size (SWS)
- Received Window Size (RWS)
-Acknowledgement (ACK)
- Negative Acknowledgement (NAK)
Sliding Window Terminology

• Sequence number
• Timeout
• Retransmission
• Round trip time (RTT)
• Utilization
• Delay X Bandwidth
Sliding Window Terminology

- Last Frame Sent (LFS)
- Last ACK Received (LAR)
- Largest Acceptable Frame (LAF)
- Last Frame Received (LFR)
rdt1.0: reliable transfer over a reliable channel

- underlying channel perfectly reliable
  - no bit errors
  - no loss of packets
- separate FSMs for sender, receiver:
  - sender sends data into underlying channel
  - receiver reads data from underlying channel

sender

receiver
**rdt2.0: FSM specification**

```
\( rdt\_send(data) \)
\( sndpkt = make\_pkt(data, checksum) \)
\( udt\_send(sndpkt) \)

\( rdt\_rcv(rcvpkt) && isNAK(rcvpkt) \)
\( udt\_send(sndpkt) \)

\( rdt\_rcv(rcvpkt) && isACK(rcvpkt) \)
\( \land \)

Wait for call from above  
Wait for ACK or NAK  
Wait for call from below  
Wait for ACK or NAK
```

**sender**

**receiver**

```
\( rdt\_rcv(rcvpkt) && corrupt(rcvpkt) \)
\( udt\_send(NAK) \)

\( rdt\_rcv(rcvpkt) && notcorr upt(rcvpkt) \)
\( extract(rcvpkt, data) \)
\( deliver\_data(data) \)
\( udt\_send(ACK) \)
```
rdt2.1: sender, handles garbled ACK/NAKs

rdt_send(data)

sndpkt = make_pkt(0, data, checksum)
udt_send(sndpkt)

Wait for call 0 from above

rdt_rcv(rcvpkt) && notcorrupt(rcvpkt) && isACK(rcvpkt)

Λ

Wait for ACK or NAK 0

rdt_send(data)

sndpkt = make_pkt(1, data, checksum)
udt_send(sndpkt)

Wait for call 1 from above

rdt_rcv(rcvpkt) && notcorrupt(rcvpkt) && isACK(rcvpkt)

Λ

Wait for ACK or NAK 1

rdt_rcv(rcvpkt) && (corrupt(rcvpkt) || isNAK(rcvpkt))

μ
rdt2.1: receiver, handles garbled ACK/NAKs

\[
\text{rdt\_rcv(rcvpkt) \\ \\ && \text{nottcorrupt(rcvpkt)} \\ && \text{has\_seq0(rcvpkt)} \\ \text{extract(rcvpkt,}\text{data)} \\ \text{deliver\_data(data)} \\ \text{sndpkt = make\_pkt(ACK, chksum)} \\ \text{udt\_send(sndpkt)} \\
\]

\[
\text{rdt\_rcv(rcvpkt) \\ && (\text{corrupt(rcvpkt)} \\ \text{sndpkt = make\_pkt(NAK, chksum)} \\ \text{udt\_send(sndpkt)} \\
\]

\[
\text{rdt\_rcv(rcvpkt) \\ && \text{nottcorrupt(rcvpkt)} \\ && \text{has\_seq1(rcvpkt)} \\ \text{sndpkt = make\_pkt(ACK, chksum)} \\ \text{udt\_send(sndpkt)} \\
\]

\[
\text{rdt\_rcv(rcvpkt) \\ && \text{nottcorrupt(rcvpkt)} \\ && \text{has\_seq1(rcvpkt)} \\ \text{extract(rcvpkt,}\text{data)} \\ \text{deliver\_data(data)} \\ \text{sndpkt = make\_pkt(ACK, chksum)} \\ \text{udt\_send(sndpkt)} \\
\]
rdt3.0 sender

Transport Layer
Utilization Problems

Answer parts a, b and c based in the following information. Assume no packets are lost or corrupted.

Packet size 2000 bytes
Data rate 100 Mbps
RTT 0.02 seconds

a. What is the utilization of the sender (or of the network connection) when the sender uses a stop-and-wait protocol.

b. What is the utilization of the sender (or of the network connection) when the sender has a window size of 10 (i.e. the sender can send 10 packets before waiting for an ACK).

c. How big should the sender's window size be in order for the utilization of the sender (or of the network connection) to be 1?