TCP
Flow Control
TCP flow control

- Receiver informs sender of spare room
  - Recall TCP is full-duplex, so while “sender” transmits data to “receiver”, the latter is posting its vacancy rate, \( RcvWindow \), to “sender” in every segment.
  - The sending host makes sure \( \text{LastByteSent} - \text{LastByteAcked} \leq \text{RcvWindow} \).
Avoiding gridlock

- There is one minor technical problem with this scheme: Suppose the B’s receive buffer fills, and RcvWindow = 0 is sent to “sender” A.
  - Suppose further, that the B has no additional data to the “sender”.
  - So what’s the problem?

Problem …

- TCP sends a segment to the sender only if
  - it has data to send, or
  - it has an acknowledgement to send
- The sender is never informed when space becomes available in the receive buffer.

Solution

- The solution is TCP specs require host A to continue to send segments with one data byte when B’s receive window is zero.
  - These segments will be acknowledge by the receiver.
  - Eventually the buffer will begin to empty and the acknowledgements will contain a nonzero RcvWindow value.
TCP
Connection management

Connection Management

Before exchanging data, sender/receiver “handshake”:
• agree to establish connection (each knowing the other willing to establish connection)
• agree on connection parameters

Socket clientSocket = newSocket("hostname","port number");

Socket connectionSocket = welcomeSocket.accept();
Agreeing to establish a connection

2-way handshake:

Let’s talk OK

ESTAB

choose x req_conn(x)

ESTAB acc_conn(x)

Q: Will 2-way handshake always work in network?

- variable delays
- retransmitted messages (e.g. req_conn(x)) due to message loss
- message reordering
- can’t “see” other side

Agreeing to establish a connection

2-way handshake failure scenarios:

choose x req_conn(x)

ESTAB

retransmit req_conn(x)

ESTAB

req_conn(x)

connection x completes

client terminates

half open connection! (no client!)

choose x req_conn(x)

ESTAB

retransmit req_conn(x)

ESTAB

req_conn(x)

connection x completes

client terminates

server forgets x

choose x req_conn(x)

ESTAB

retransmit data(x+1)

ESTAB

req_conn(x)

data(x+1)

client completes

server forgets x

choose x req_conn(x)

ESTAB

accept data(x+1)
TCP 3-way handshake

client state

LISTEN

SYNSENT

ESTAB

choose init seq num, x
send TCP SYN msg

SYNbit=1, Seq=x

received SYNACK(x) indicates server is live; send ACK for SYNACK;
this segment may contain client-to-server data

SYNbit=1, Seq=y

ACKbit=1; ACKnum=x+1

ACKbit=1, ACKnum=y+1

received ACK(y) indicates client is live

server state

LISTEN

SYN RCVD

ESTAB

choose init seq num, y
send TCP SYNACK msg, acking SYN

SYNbit=1, Seq=y

Received SYNACK(x) indicates server is live;
send ACK for SYNACK;
this segment may contain client-to-server data

SYNSENT

ESTABLISHED

1. Client requests TCP connection: Hi

Client host sends TCP SYN segment to server specifying initial sequence number ...

...and setting the SYN bit,

but sends no application-layer data.
2. The server replies: Hello!

Server host receives **SYN**, allocates the TCP buffers and variables to the client TCP, and replies with **SYNACK** by choosing its initial sequence number, setting its Acknowledgment field to the client_isn+1, and setting the ACK and SYN bits (still no data).

3. Client responds: Can we talk?

Client host receives **ACKSYN**, allocates its own TCP buffers and variables to the server TCP, replies with SYN bit set to zero since connection is established, and may include data.
TCP: closing a connection

- Client, server each close their side of connection
  - send TCP segment with FIN bit = 1

- Respond to received FIN with ACK
  - on receiving FIN, ACK can be combined with own FIN

- Simultaneous FIN exchanges can be handled