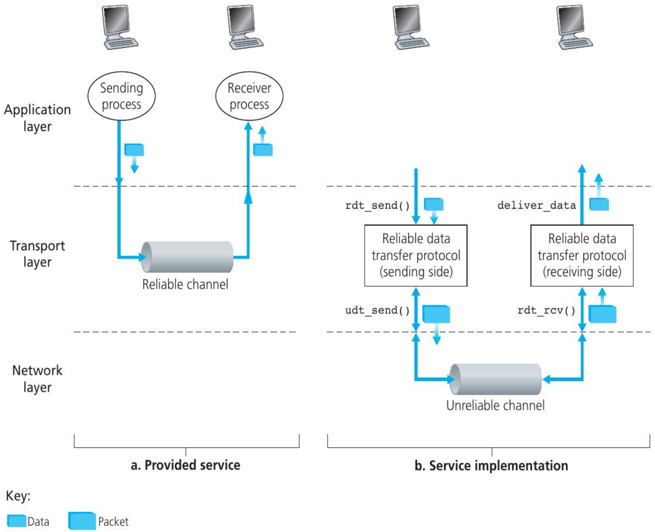
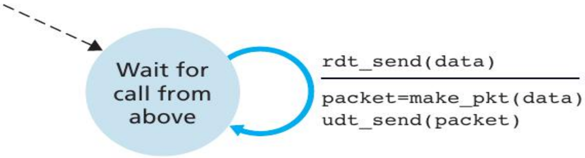
Principles of Reliable Data Transfer

Dr. A Krishna Chaitanya,
Indian Institute of Information Technology Sri City

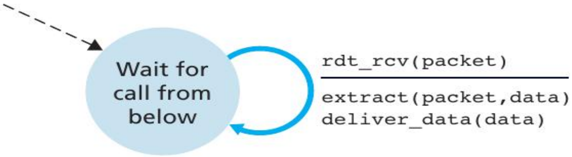
Reliable Data Transfer



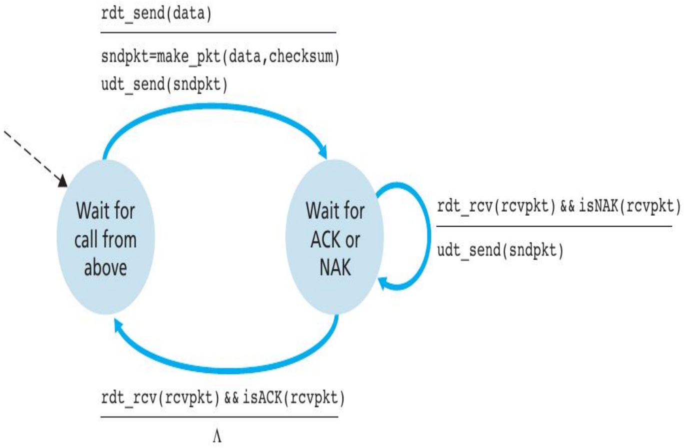
RDT1.0: Perfectly Reliable Channel



a. rdt1.0: sending side

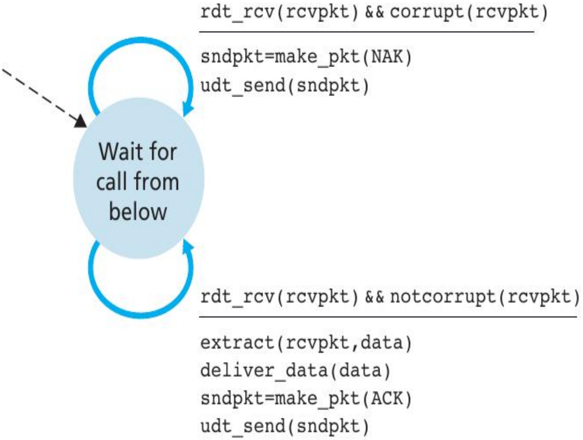


b. rdt1.0: receiving side



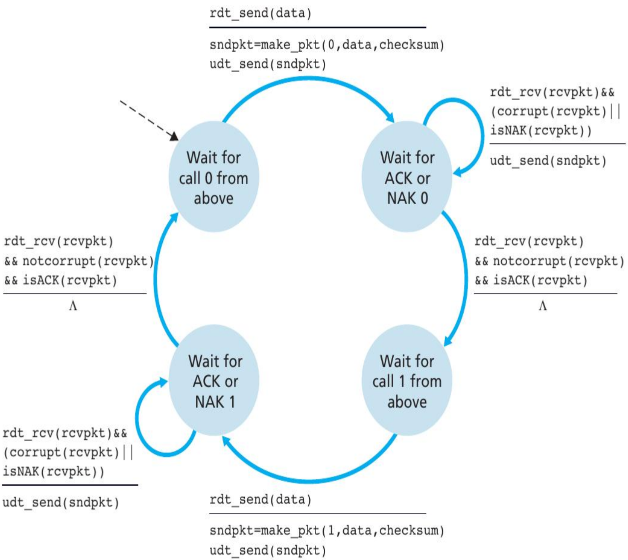
a. rdt2.0: sending side

RDT Over a Channel with Bit Errors: rdt 2.0 receiver

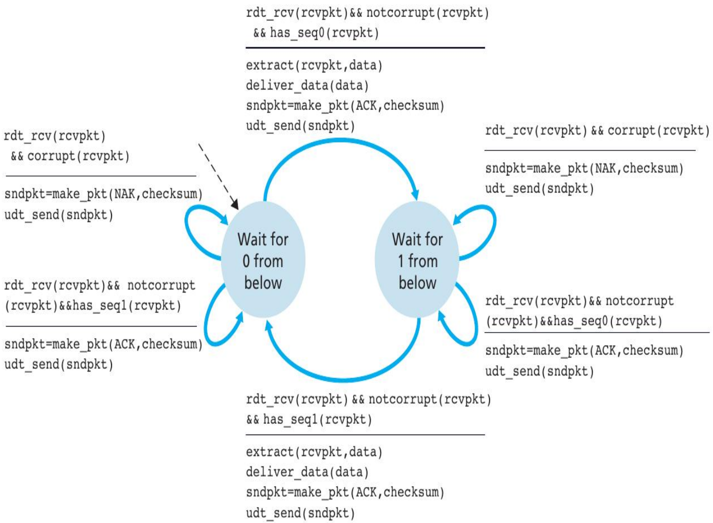


b. rdt2.0: receiving side

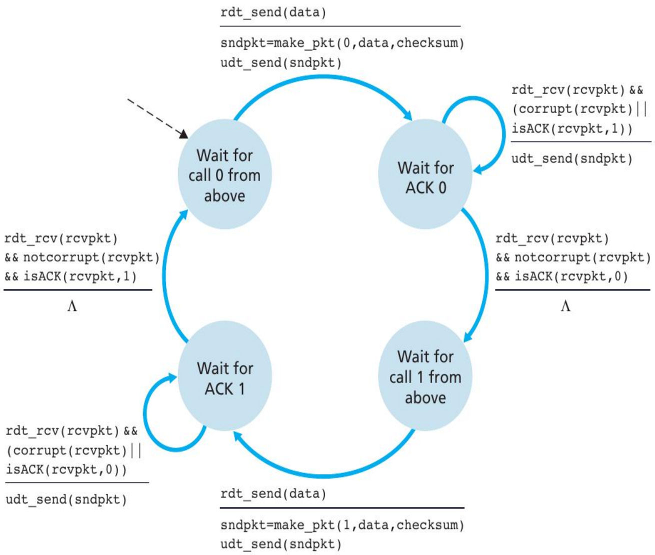
RDT 2.1 Sender



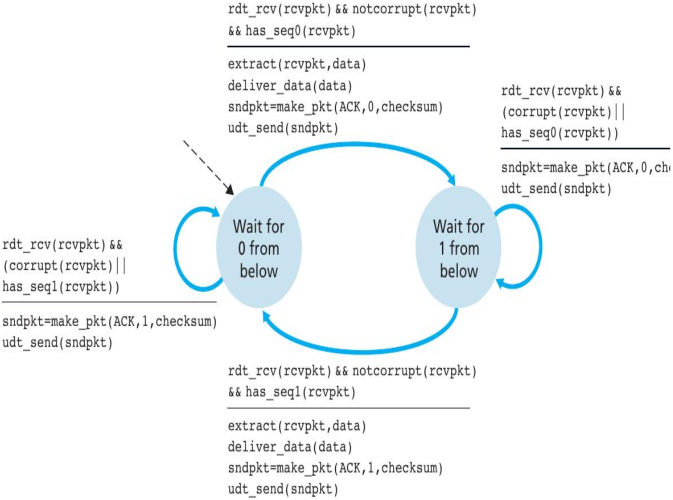
RDT 2.1 Receiver



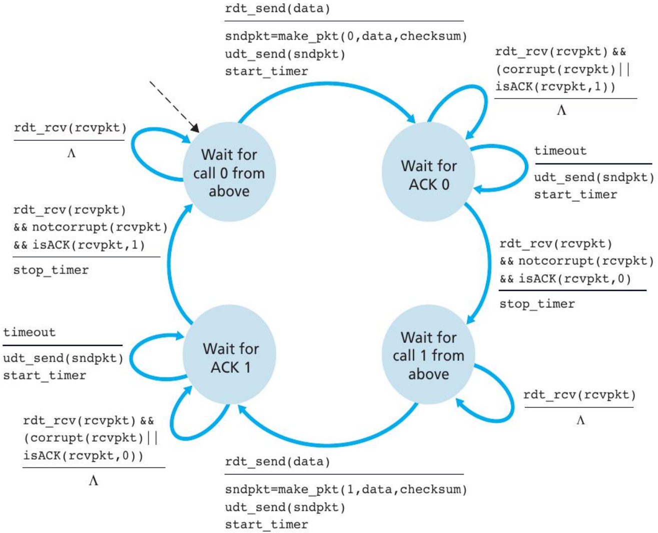
RDT Over a Lossy Channel with Bit Errors: rdt 2.2 sender



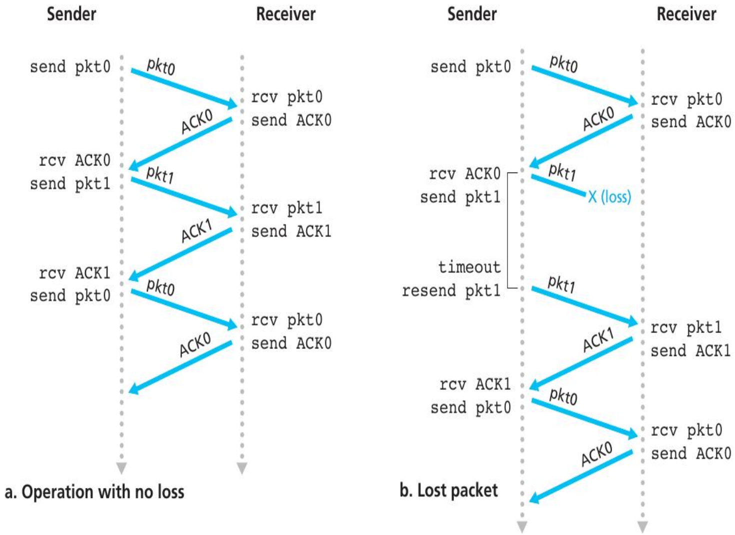
RDT Over a Lossy Channel with Bit Errors: rdt 2.2 receiver



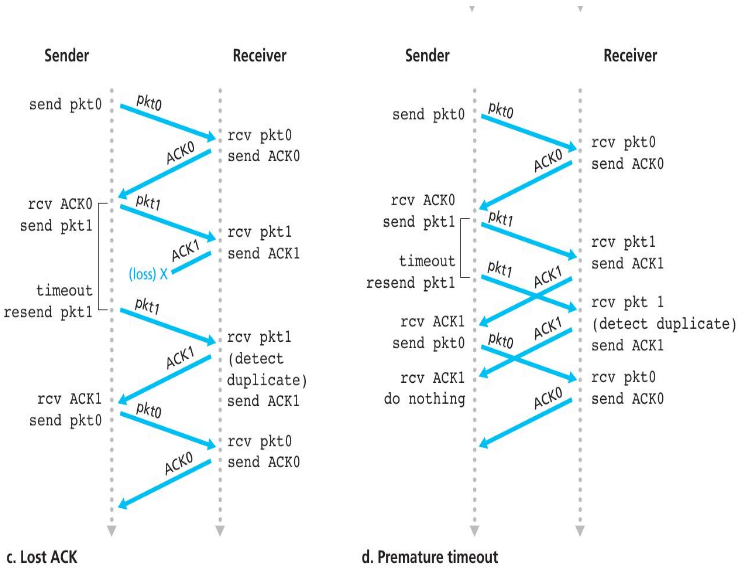
RDT 3.0: NAK-Free



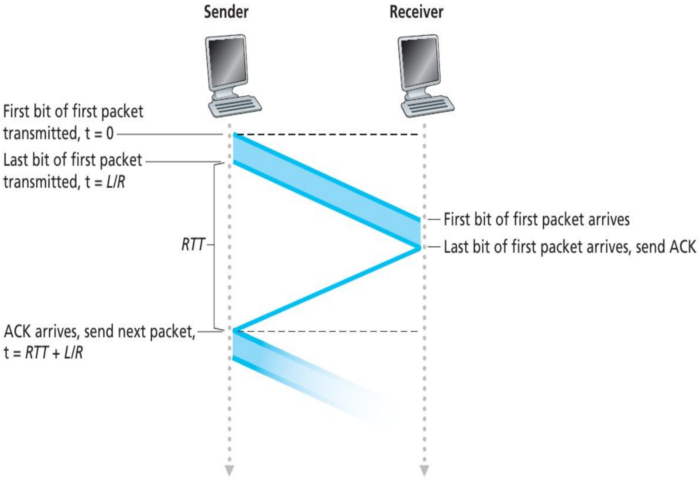
RDT 3.0-Alternating-bit Protocol: Operation



RDT 3.0-Alternating-bit Protocol: Operation

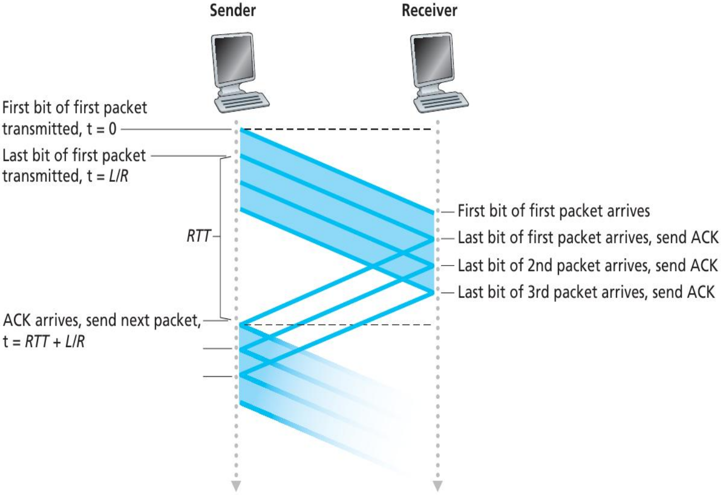


Stop-and-Wait Operation



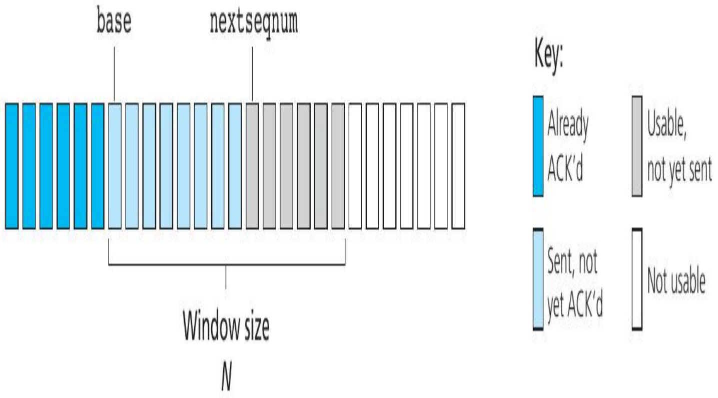
a. Stop-and-wait operation

Pipelining

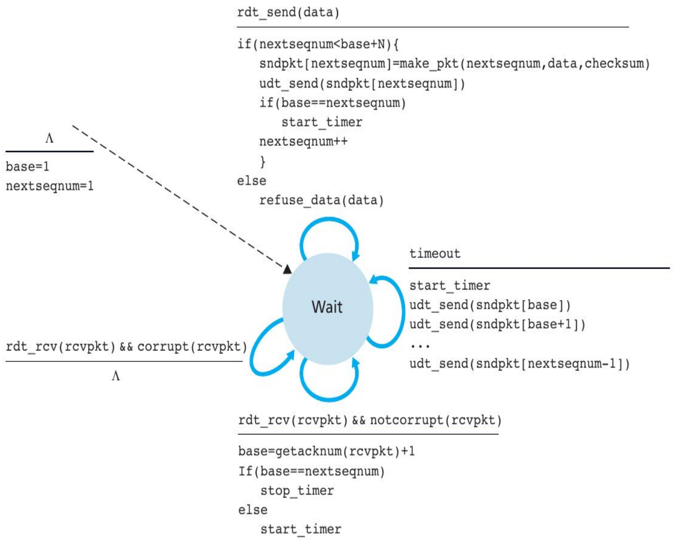


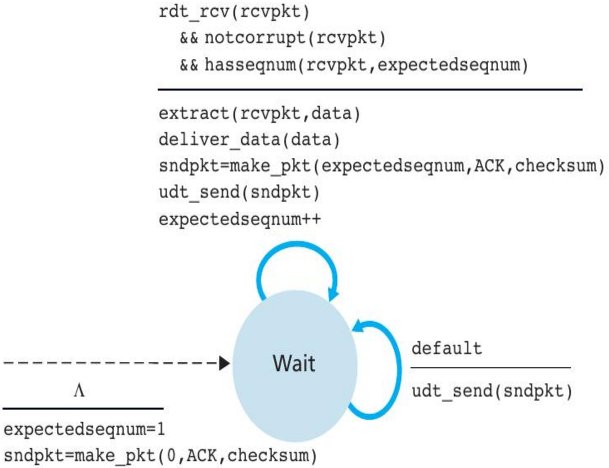
b. Pipelined operation

Go-Back-N

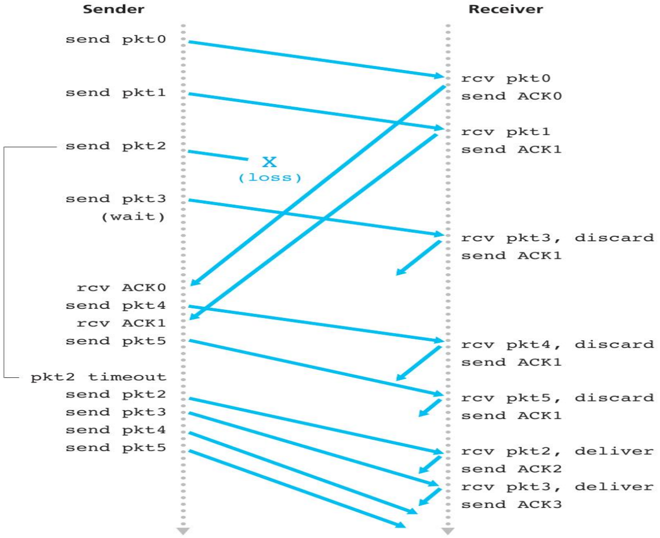

base
nextseqnum
Window size
N
Key:
Already ACK'd
Usable, not yet sent
Sent, not yet ACK'd
Not usable

GBN Sender

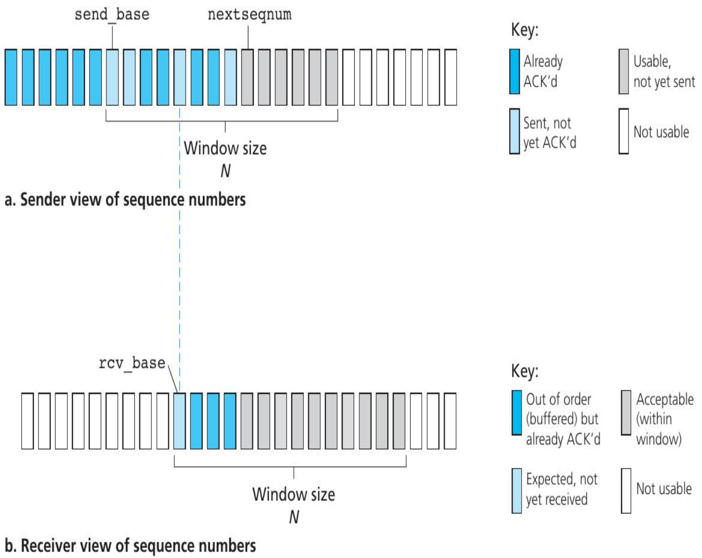

rdt_send(data)
if(nextseqnum<base+N){
sndpkt[nextseqnum]=make_pkt(nextseqnum,data,checksum)
udt_send(sndpkt[nextseqnum])
if(base==nextseqnum)
start_timer
nextseqnum++
}
else
refuse_data(data)
Λ
base=1
nextseqnum=1
Wait
timeout
start_timer
udt_send(sndpkt[base])
udt_send(sndpkt[base+1])
...
udt_send(sndpkt[nextseqnum-1])
rdt_rcv(rcvpkt) && corrupt(rcvpkt)
Λ
rdt_rcv(rcvpkt) && notcorrupt(rcvpkt)
base=getacknum(rcvpkt)+1
If(base==nextseqnum)
stop_timer
else
start_timer


rdt_rcv(rcvpkt)
&& notcorrupt(rcvpkt)
&& hasseqnum(rcvpkt,expectedseqnum)
extract(rcvpkt,data)
deliver_data(data)
sndpkt=make_pkt(expectedseqnum,ACK,checksum)
udt_send(sndpkt)
expectedseqnum++
Wait
default
udt_send(sndpkt)
Λ
expectedseqnum=1
sndpkt=make_pkt(0,ACK,checksum)

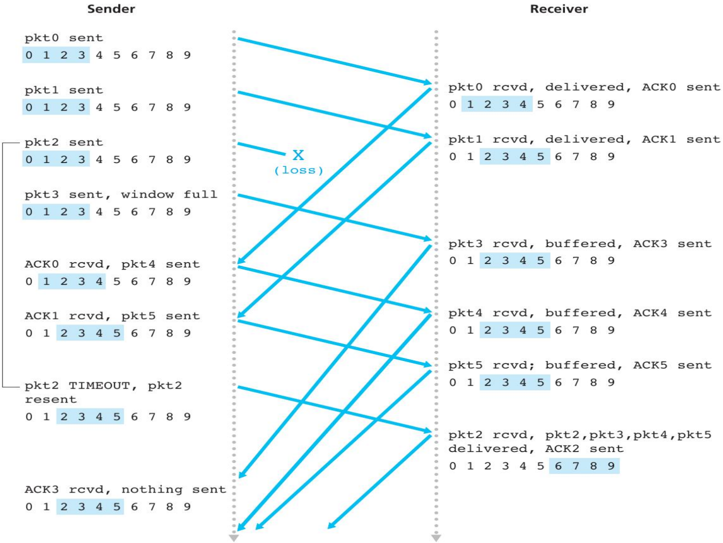
GBN Operation



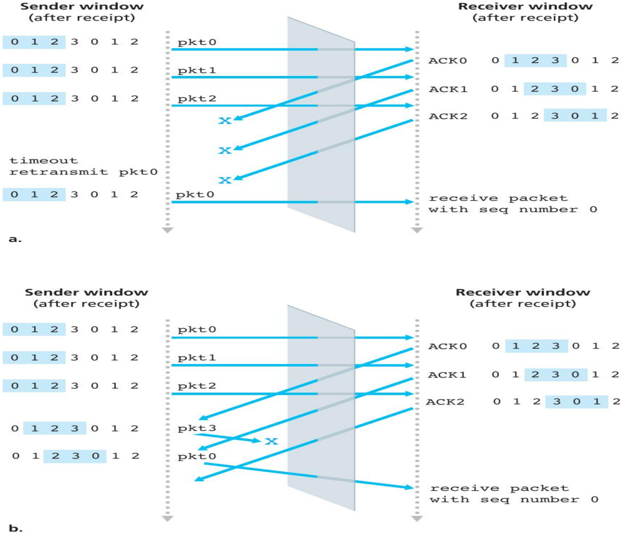
Selective-Repeat



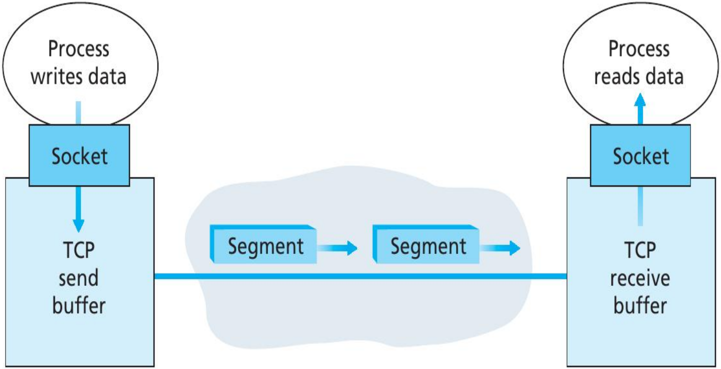
SR Operation



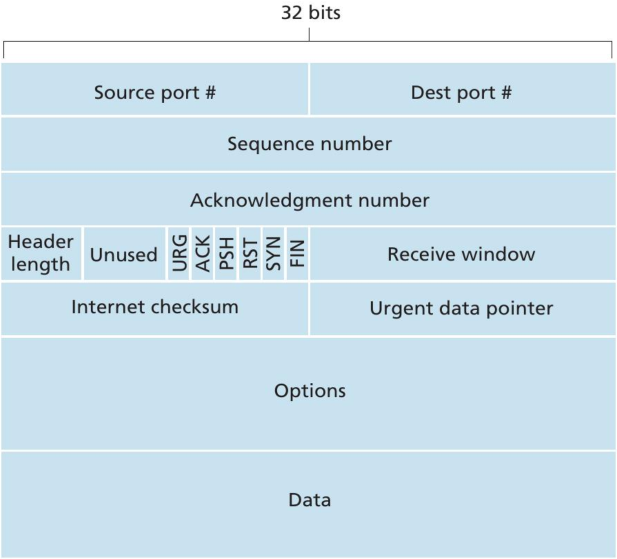
Window Size in SR



- TCP is a full duplex service
- No multicasting
- Maximum segment size (MSS) is the maximum amount of data that a TCP segment can contain.



TCP Segment


32 bits
Source port #
Dest port #
Sequence number
Acknowledgment number
Header length
Unused
URG
ACK
PSH
RST
SYN
FIN
Receive window
Internet checksum
Urgent data pointer
Options
Data

TCP Segment

- The 16-bit receive window indicates the number of bytes that a receiver is willing to accept

TCP Segment

- The 16-bit receive window indicates the number of bytes that a receiver is willing to accept
- Header length filed is 4-bytes, specifies the length of the TCP header in 32-bit words.

TCP Segment

- The 16-bit receive window indicates the number of bytes that a receiver is willing to accept
- Header length filed is 4-bytes, specifies the length of the TCP header in 32-bit words.
- Options are used to negotiate MSS, include time-stamping, etc.

TCP Segment

- The 16-bit receive window indicates the number of bytes that a receiver is willing to accept
- Header length filed is 4-bytes, specifies the length of the TCP header in 32-bit words.
- Options are used to negotiate MSS, include time-stamping, etc.
- The flag field contains 6 bits, RST, SYN, FIN are used for connection setup and teardown.

TCP Segment

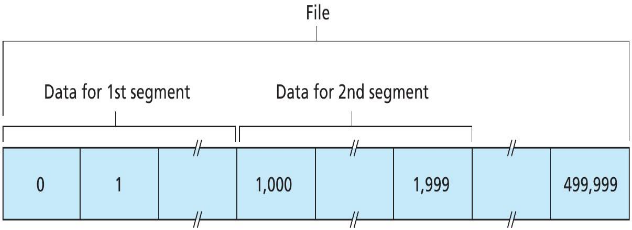
- The 16-bit receive window indicates the number of bytes that a receiver is willing to accept
- Header length filed is 4-bytes, specifies the length of the TCP header in 32-bit words.
- Options are used to negotiate MSS, include time-stamping, etc.
- The flag field contains 6 bits, RST, SYN, FIN are used for connection setup and teardown.
- PSH indicates that data has to be sent to upper layers immediately.

TCP Segment

- The 16-bit receive window indicates the number of bytes that a receiver is willing to accept
- Header length filed is 4-bytes, specifies the length of the TCP header in 32-bit words.
- Options are used to negotiate MSS, include time-stamping, etc.
- The flag field contains 6 bits, RST, SYN, FIN are used for connection setup and teardown.
- PSH indicates that data has to be sent to upper layers immediately.
- URG is used to mark the segment as urgent, when it is on there will be a 16-bit urgent data pointer filed at the end of urgent data.

TCP Sequence Numbers

- The sequence number of a segment is the byte-stream number of the first byte of data.
- The acknowledge number is the sequence number of the next byte that is receiver is expecting from source.
- TCP provides cumulative acknowledgments; Out-of-order segements?
- Seqence numbers may not always start from '0'.



TCP Timeout

- SampleRTT: RTT of a freshly transmitted packet. Computed for each RTT.

TCP Timeout

- SampleRTT: RTT of a freshly transmitted packet. Computed for each RTT.
- Exponentially weighted moving average: EstimatedRTT $=$ $(1-\alpha)$EstmiatedRTT $+ \alpha$ SampleRTT

TCP Timeout

- SampleRTT: RTT of a freshly transmitted packet. Computed for each RTT.
- Exponentially weighted moving average: EstimatedRTT $=$ $(1-\alpha)$EstmiatedRTT $+ \alpha$ SampleRTT
- $\alpha = 0.125$

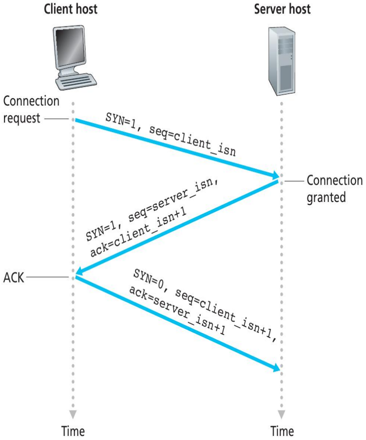
TCP Timeout

- SampleRTT: RTT of a freshly transmitted packet. Computed for each RTT.
- Exponentially weighted moving average: EstimatedRTT $=$ $(1-\alpha)$EstmiatedRTT $+ \alpha$ SampleRTT
- $\alpha = 0.125$
- DevRTT $=$ $(1-\beta)$ DevRTT $+ \beta$ $|$ SampleRTT - EstimatedRTT$|$
- $\beta = 0.25$

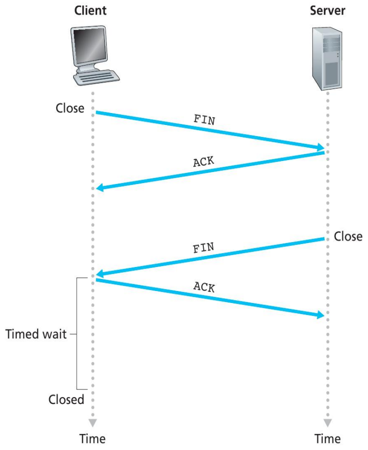
TCP Timeout

- SampleRTT: RTT of a freshly transmitted packet. Computed for each RTT.
- Exponentially weighted moving average: EstimatedRTT $=$ $(1-\alpha)$EstmiatedRTT $+ \alpha$ SampleRTT
- $\alpha = 0.125$
- DevRTT $=$ $(1-\beta)$ DevRTT $+ \beta$ | SampleRTT - EstimatedRTT|
- $\beta = 0.25$
- Timeout $=$ EstimatedRTT $+ 4.$ DevRTT

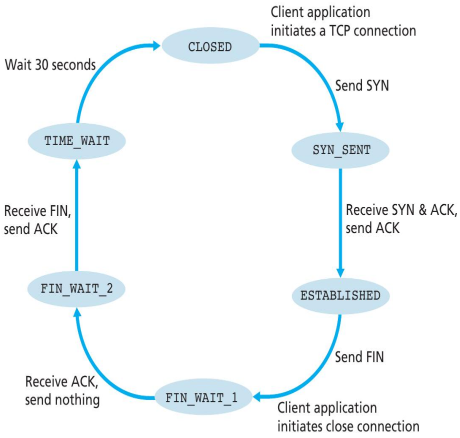
TCP Connection Establishment



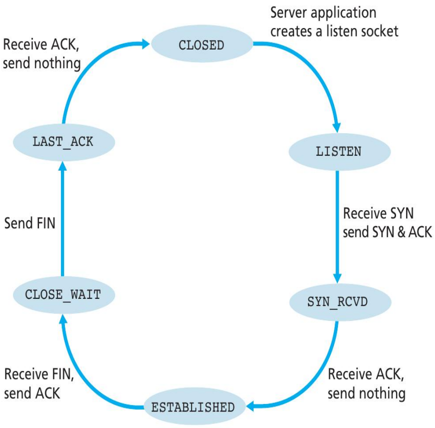
TCP Connection Termination



TCP States at Cleint



TCP States at Server



Congestion

- Why does Congestion occur?

Congestion

- Why does Congestion occur?
- Packet arrival rate at a router is near or higher than the output link capacity.

Congestion

- Why does Congestion occur?
- Packet arrival rate at a router is near or higher than the output link capacity.
- Consequences?

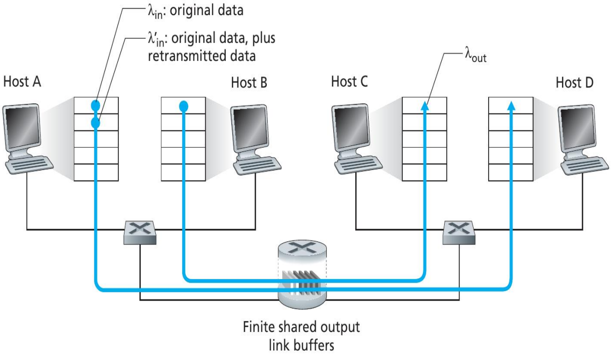
Congestion

- Why does Congestion occur?
- Packet arrival rate at a router is near or higher than the output link capacity.
- Consequences?
- Buffer overflows, retransmissions to compensate for lost packets

Congestion

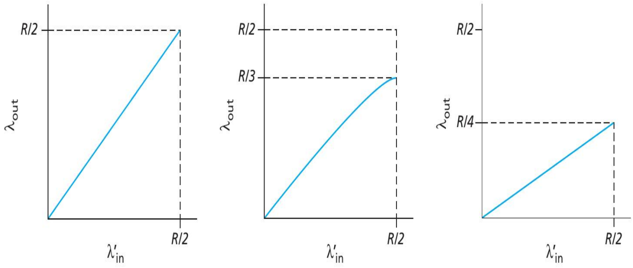
- Why does Congestion occur?
- Packet arrival rate at a router is near or higher than the output link capacity.
- Consequences?
- Buffer overflows, retransmissions to compensate for lost packets
- Unneeded retransmissions

Congestion Scenario - 1

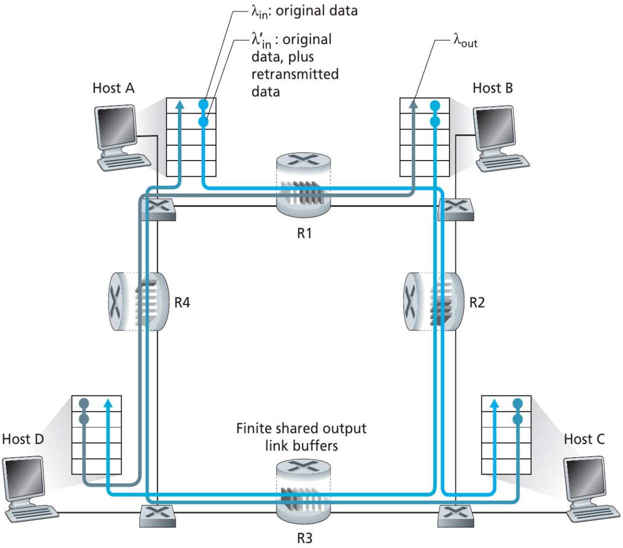


- (a) Host A knows whether buffer in the router has free space or not (Magic!)
- (b) Host A retransmits only if it is sure that packet is lost (Someone has to give this information)
- (c) Host A retransmits on timeouts!

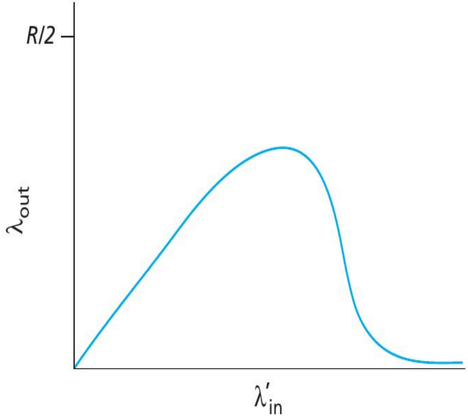
Congestion Scenario - 1



Congestion Scenario - 2



Congestion Scenario - 2



Congestion Control

- End-to-end congestion control: no explicit information about congestion the network
- Network-assisted Congestion Control: Choke packet

Congestion Control

- End-to-end congestion control: no explicit information about congestion the network
- Network-assisted Congestion Control: Choke packet
- How does TCP identify congestion?

Congestion Control

- End-to-end congestion control: no explicit information about congestion the network
- Network-assisted Congestion Control: Choke packet
- How does TCP identify congestion?
- No assistance form IP

Congestion Control

- End-to-end congestion control: no explicit information about congestion the network
- Network-assisted Congestion Control: Choke packet
- How does TCP identify congestion?
- No assistance form IP
- Identify congestion through timeouts and duplicate ACKs

TCP's Congestion Control

- How does TCP control the sending rate?

TCP's Congestion Control

- How does TCP control the sending rate?
- Defines a new variable called cwnd.

TCP's Congestion Control

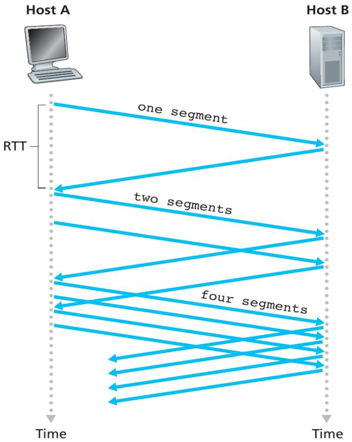
- How does TCP control the sending rate?
- Defines a new variable called cwnd.
- LastByteSent - LastByteAcked $\leq$ min{cwnd,rwnd}

TCP's Congestion Control

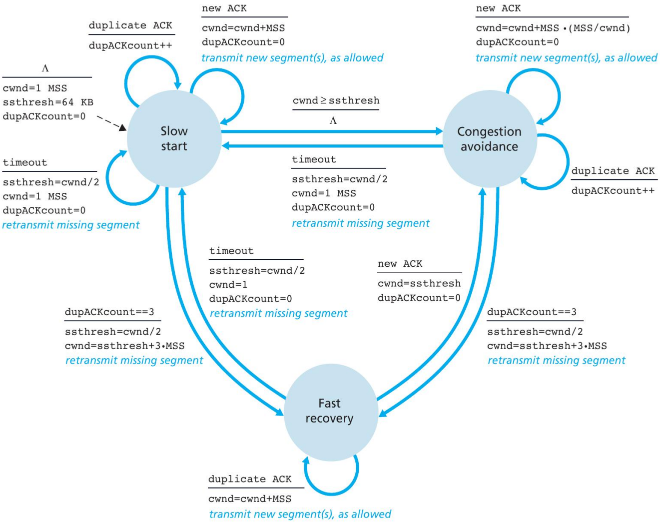
- How does TCP control the sending rate?
- Defines a new variable called cwnd.
- LastByteSent - LastByteAcked $\leq$ min{cwnd,rwnd}
- Sends cwnd bytes of infomration per RTT (approximately)

TCP's Congestion Control

- How does TCP control the sending rate?
- Defines a new variable called cwnd.
- LastByteSent - LastByteAcked $\leq$ min{cwnd,rwnd}
- Sends cwnd bytes of infomration per RTT (approximately)
- Can we adjust the speed? Slef-clocking
  - A lost segment triggers the sender to reduce rate of transmission
  - An acknowledgment indicates all is well! Increase the rate
  - Bandwidth Probing


Host A
Host B
RTT
one segment
two segments
four segments
Time
Time

TCP Congestion Control



TCP Congestion Window