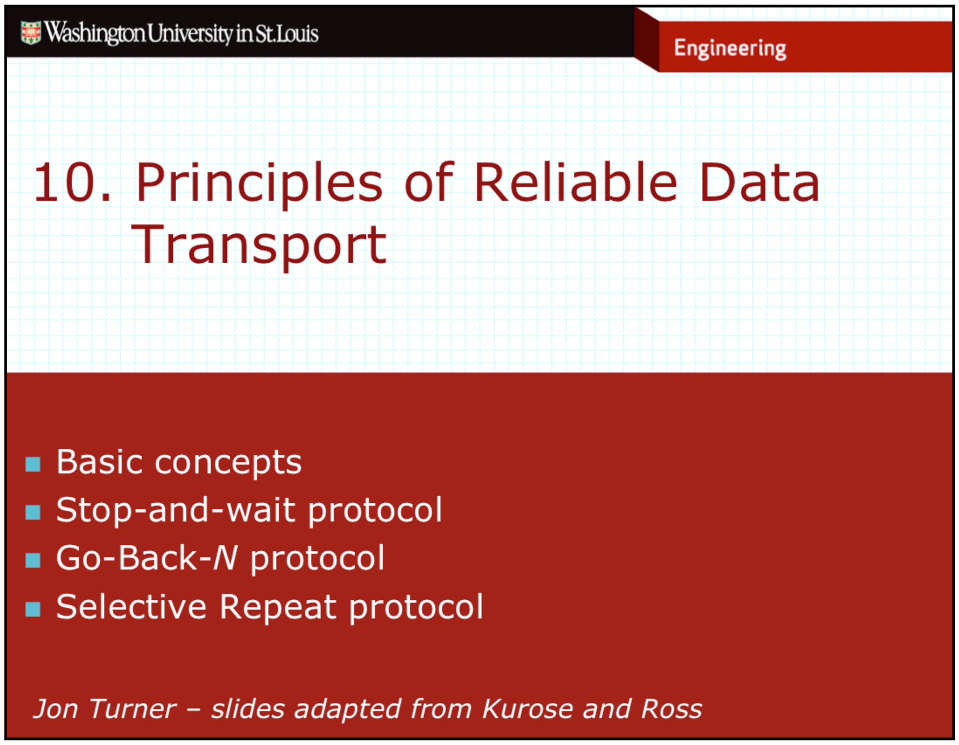


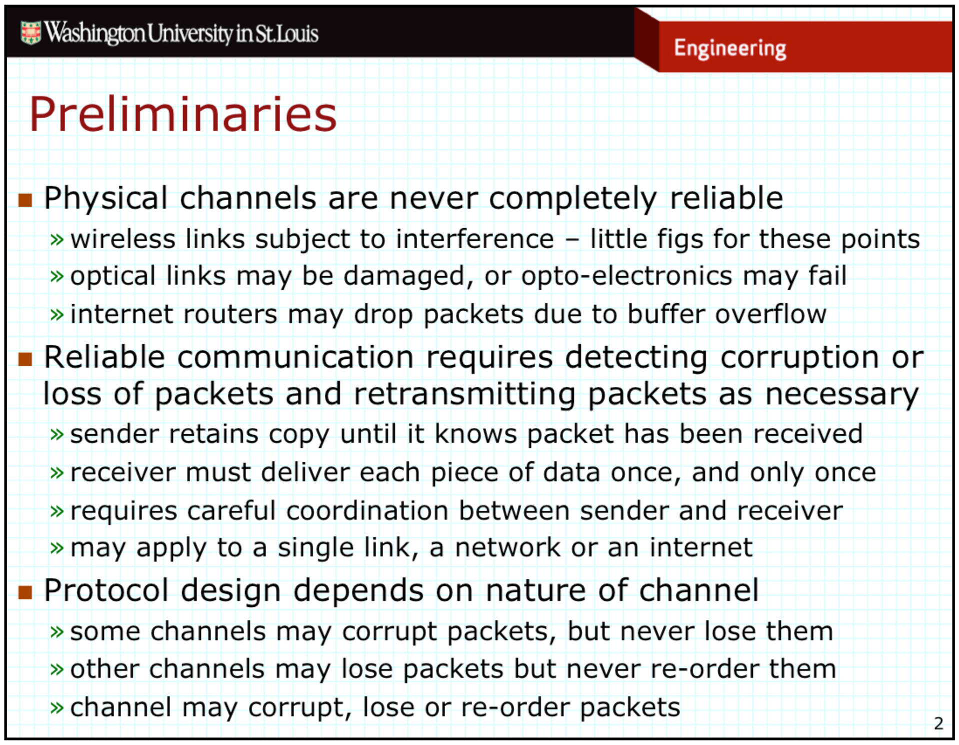
Washington University in St.Louis

Engineering

10. Principles of Reliable Data Transport

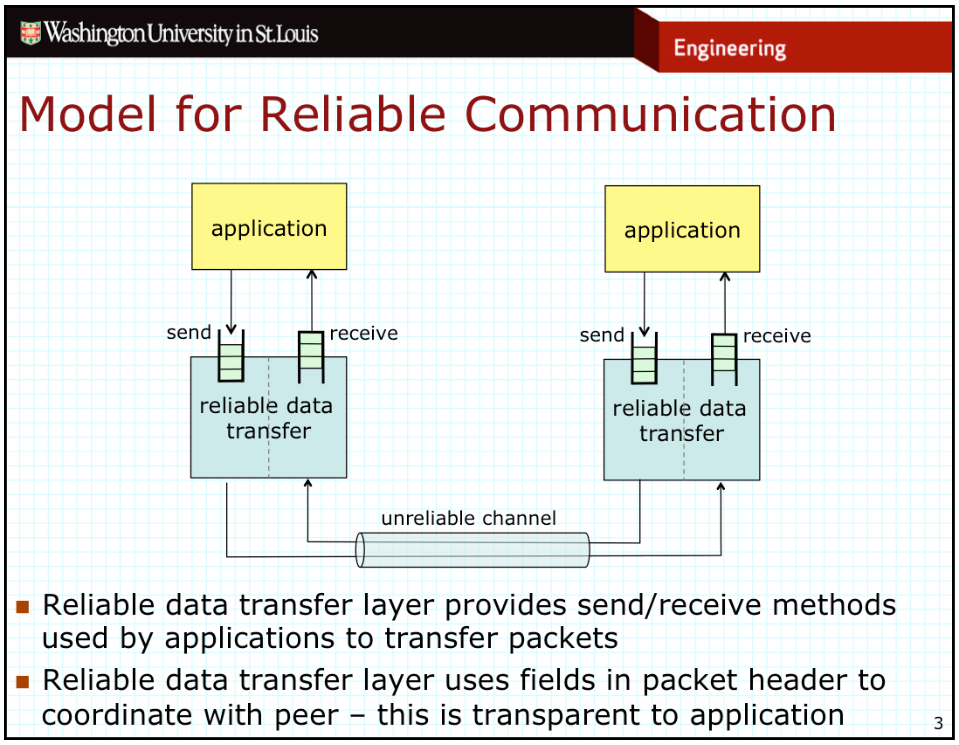
- Basic concepts
- Stop-and-wait protocol
- Go-Back-*N* protocol
- Selective Repeat protocol

Jon Turner – slides adapted from Kurose and Ross



Preliminaries

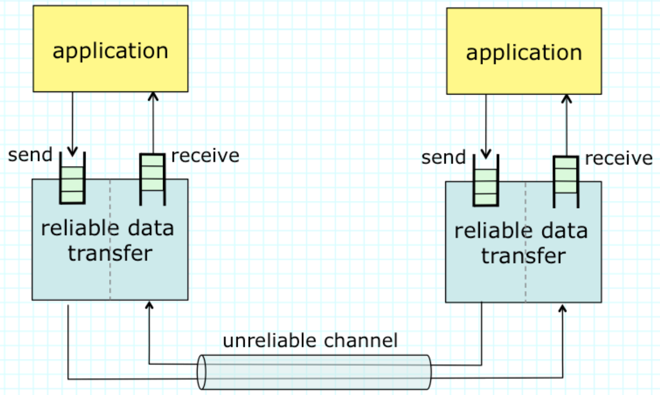
- Physical channels are never completely reliable
  - » wireless links subject to interference – little figs for these points
  - » optical links may be damaged, or opto-electronics may fail
  - » internet routers may drop packets due to buffer overflow
- Reliable communication requires detecting corruption or loss of packets and retransmitting packets as necessary
  - » sender retains copy until it knows packet has been received
  - » receiver must deliver each piece of data once, and only once
  - » requires careful coordination between sender and receiver
  - » may apply to a single link, a network or an internet
- Protocol design depends on nature of channel
  - » some channels may corrupt packets, but never lose them
  - » other channels may lose packets but never re-order them
  - » channel may corrupt, lose or re-order packets



Washington University in St. Louis

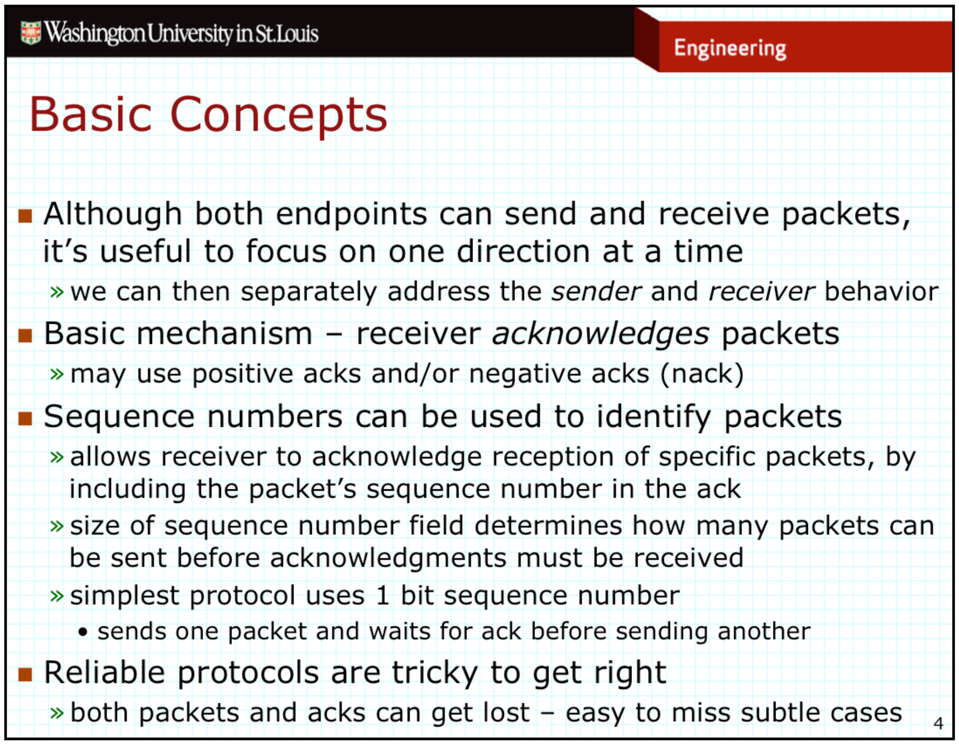
Engineering

Model for Reliable Communication



- Reliable data transfer layer provides send/receive methods used by applications to transfer packets
- Reliable data transfer layer uses fields in packet header to coordinate with peer – this is transparent to application

3



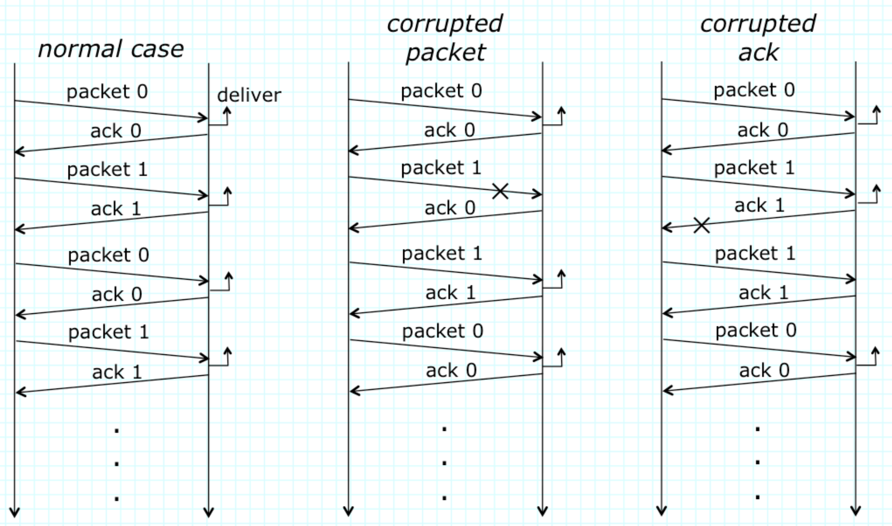
Basic Concepts

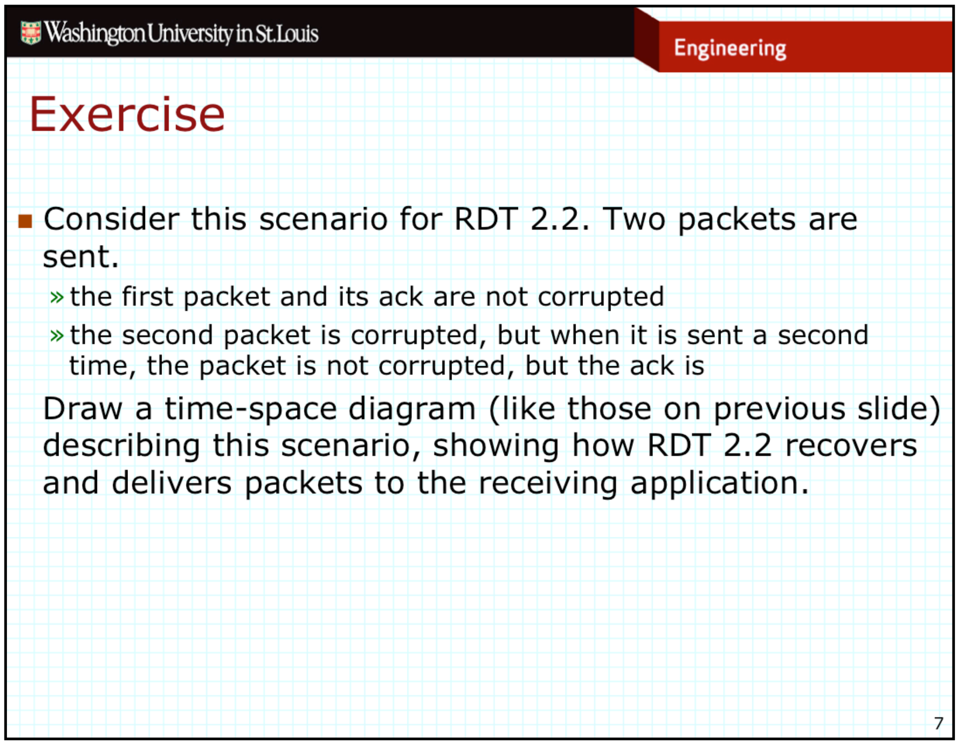
- Although both endpoints can send and receive packets, it's useful to focus on one direction at a time
  - » we can then separately address the *sender* and *receiver* behavior
- Basic mechanism – receiver *acknowledges* packets
  - » may use positive acks and/or negative acks (nack)
- Sequence numbers can be used to identify packets
  - » allows receiver to acknowledge reception of specific packets, by including the packet's sequence number in the ack
  - » size of sequence number field determines how many packets can be sent before acknowledgments must be received
  - » simplest protocol uses 1 bit sequence number
    - sends one packet and waits for ack before sending another
- Reliable protocols are tricky to get right
  - » both packets and acks can get lost – easy to miss subtle cases

Basic Protocol – RDT 2.x

- Assume channel may corrupt but never lose packets
  - » receiver can always detect if packet has been corrupted
- Simple protocol for such a channel uses acks and nacks
  - » sender transmits one packet at a time and waits for ack/nack before sending the next packet
  - » receiver sends ack when it receives packet correctly, sends nack when it receives packet with an error
- What can go wrong?
  - » if ack or nack is corrupted, sender has no way of knowing if receiver got the packet correctly
  - » if sender retransmits the packet, the receiver could deliver the same packet to the application twice
- Solution is to add a 1 bit sequence number
  - » in this case, it is sufficient to use positive acks only

RDT 2.2 Behavior



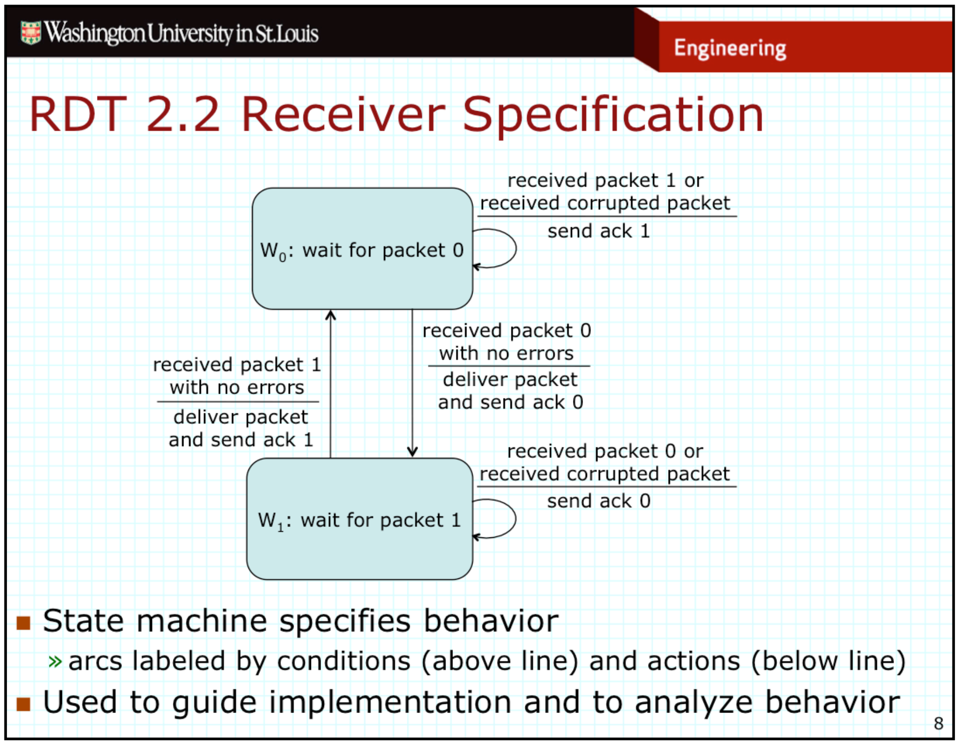


Exercise

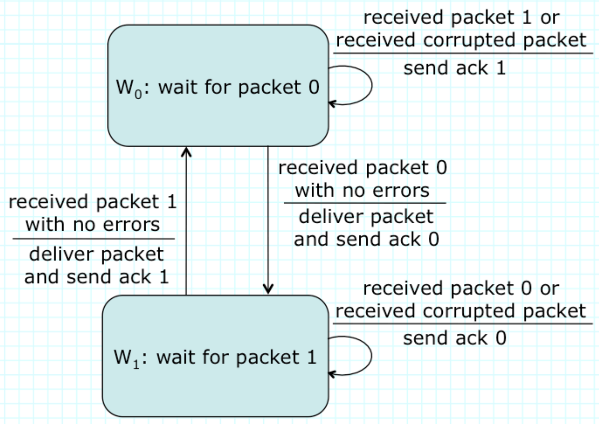
- Consider this scenario for RDT 2.2. Two packets are sent.
  - » the first packet and its ack are not corrupted
  - » the second packet is corrupted, but when it is sent a second time, the packet is not corrupted, but the ack is

  Draw a time-space diagram (like those on previous slide) describing this scenario, showing how RDT 2.2 recovers and delivers packets to the receiving application.

7



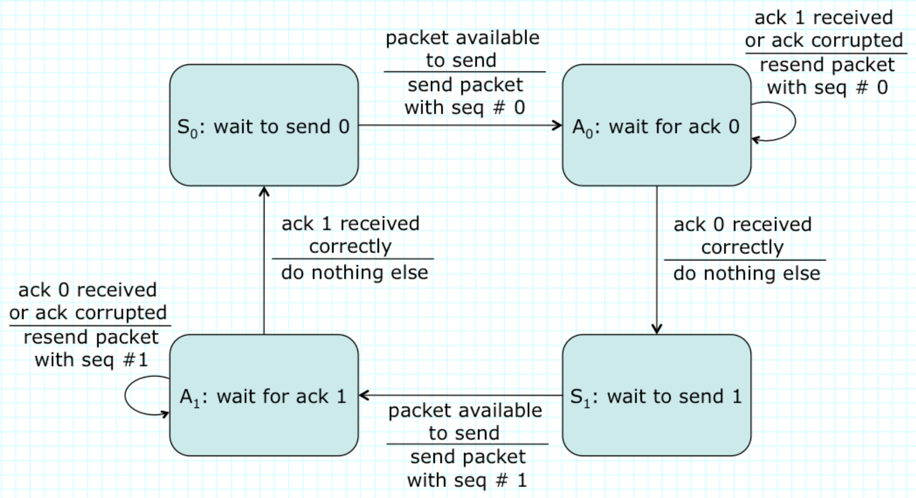
RDT 2.2 Receiver Specification


W$_0$: wait for packet 0
received packet 1 or received corrupted packet
send ack 1
received packet 0 with no errors
deliver packet and send ack 0
received packet 1 with no errors
deliver packet and send ack 1
W$_1$: wait for packet 1
received packet 0 or received corrupted packet
send ack 0

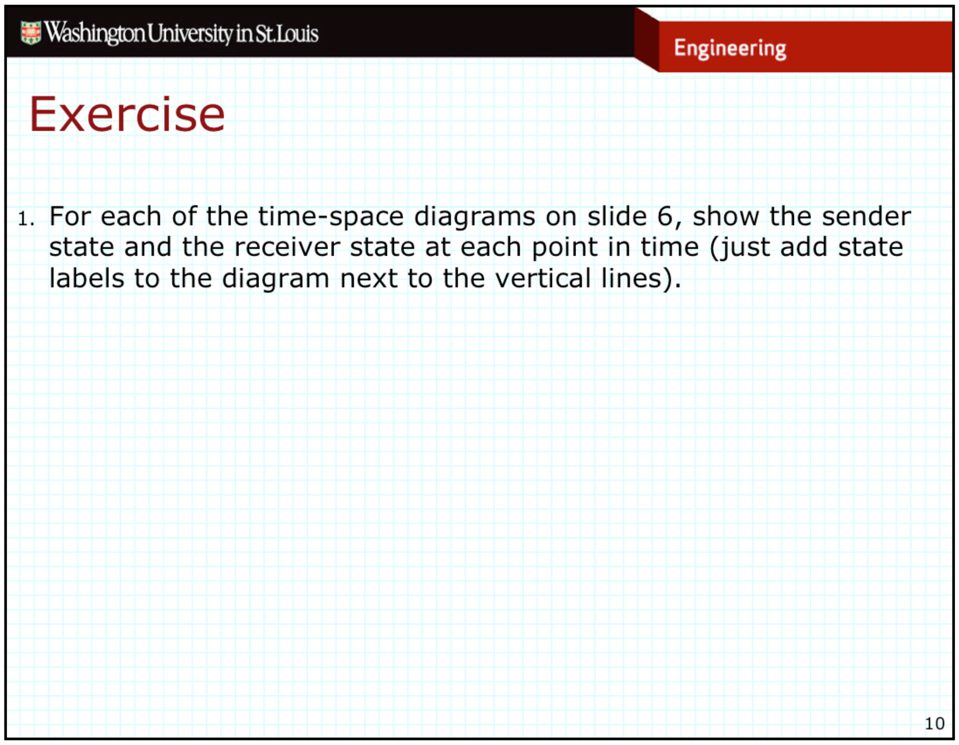
- State machine specifies behavior
  - » arcs labeled by conditions (above line) and actions (below line)
- Used to guide implementation and to analyze behavior

Washington University in St.Louis Engineering

RDT 2.2 Sender Specification

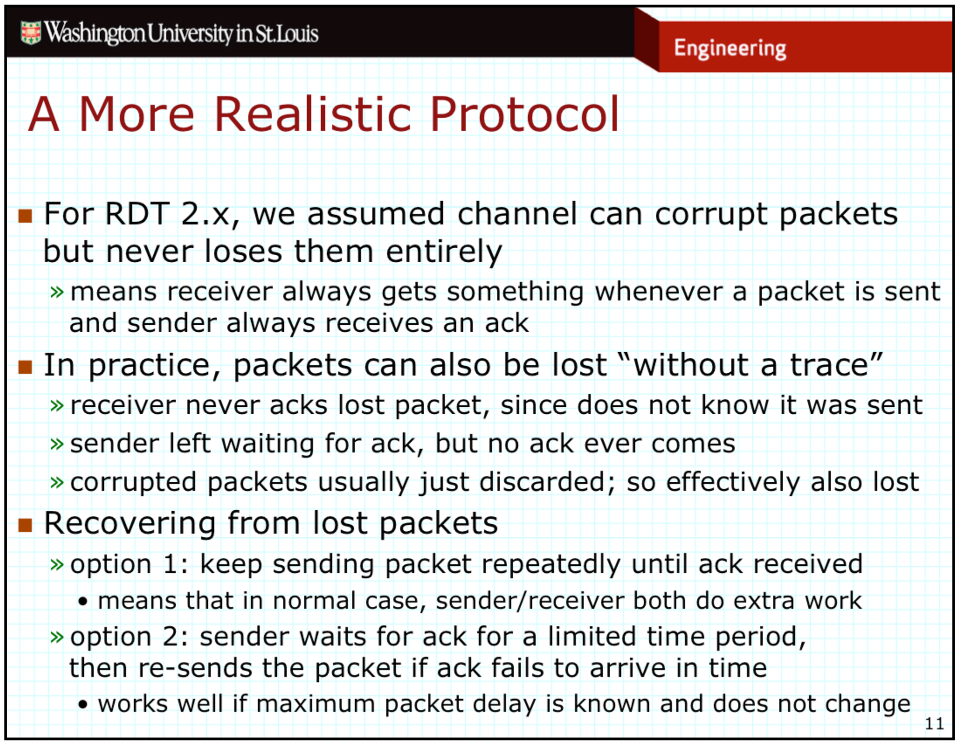

S_0: wait to send 0
packet available to send / send packet with seq # 0
A_0: wait for ack 0
ack 1 received or ack corrupted / resend packet with seq # 0
ack 0 received correctly / do nothing else
S_1: wait to send 1
packet available to send / send packet with seq # 1
A_1: wait for ack 1
ack 0 received or ack corrupted / resend packet with seq #1
ack 1 received correctly / do nothing else

9



Exercise

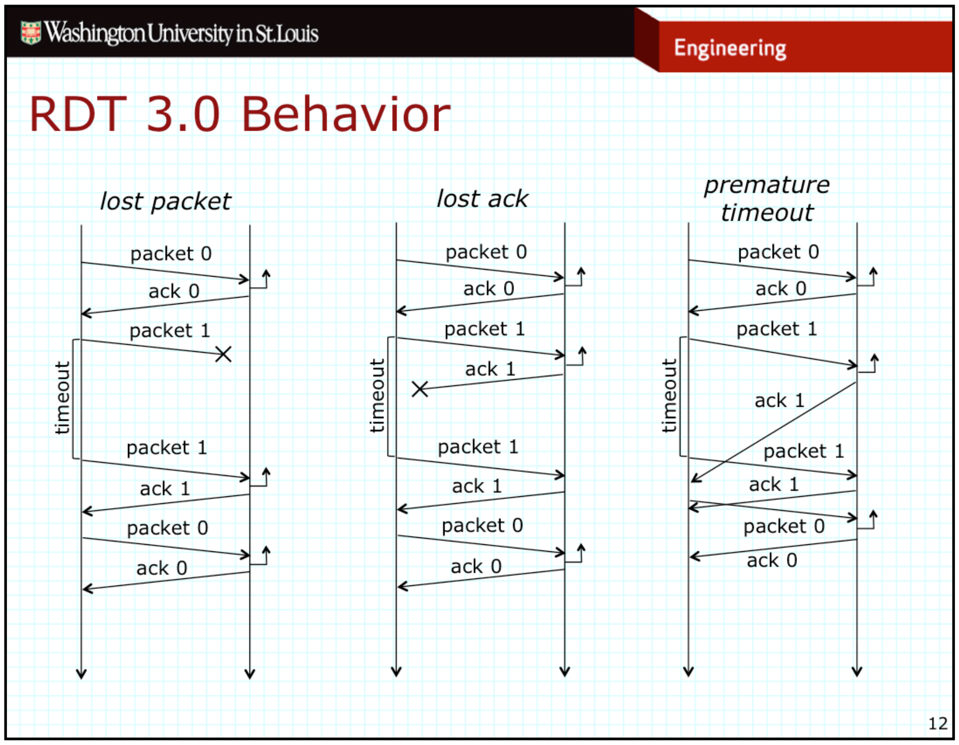
1. For each of the time-space diagrams on slide 6, show the sender state and the receiver state at each point in time (just add state labels to the diagram next to the vertical lines).


Washington University in St. Louis Engineering

A More Realistic Protocol

- For RDT 2.x, we assumed channel can corrupt packets but never loses them entirely
  - » means receiver always gets something whenever a packet is sent and sender always receives an ack
- In practice, packets can also be lost "without a trace"
  - » receiver never acks lost packet, since does not know it was sent
  - » sender left waiting for ack, but no ack ever comes
  - » corrupted packets usually just discarded; so effectively also lost
- Recovering from lost packets
  - » option 1: keep sending packet repeatedly until ack received
    - • means that in normal case, sender/receiver both do extra work
  - » option 2: sender waits for ack for a limited time period, then re-sends the packet if ack fails to arrive in time
    - • works well if maximum packet delay is known and does not change

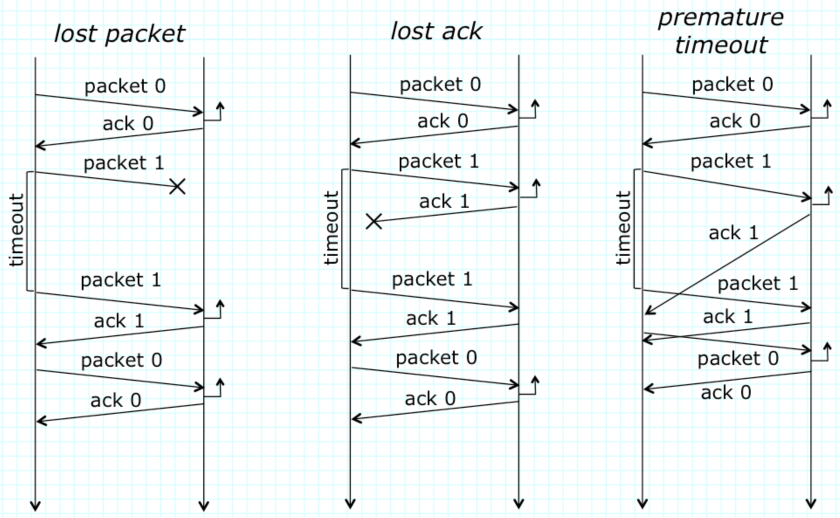
11



Washington University in St.Louis

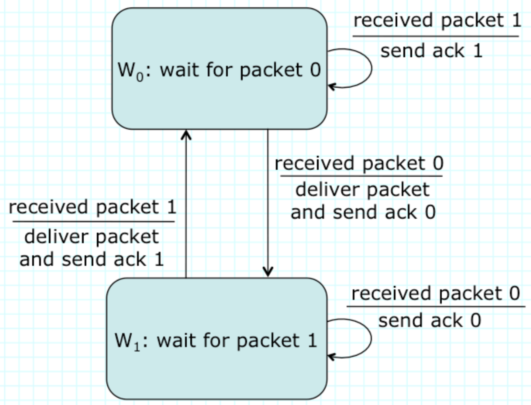
Engineering

RDT 3.0 Behavior

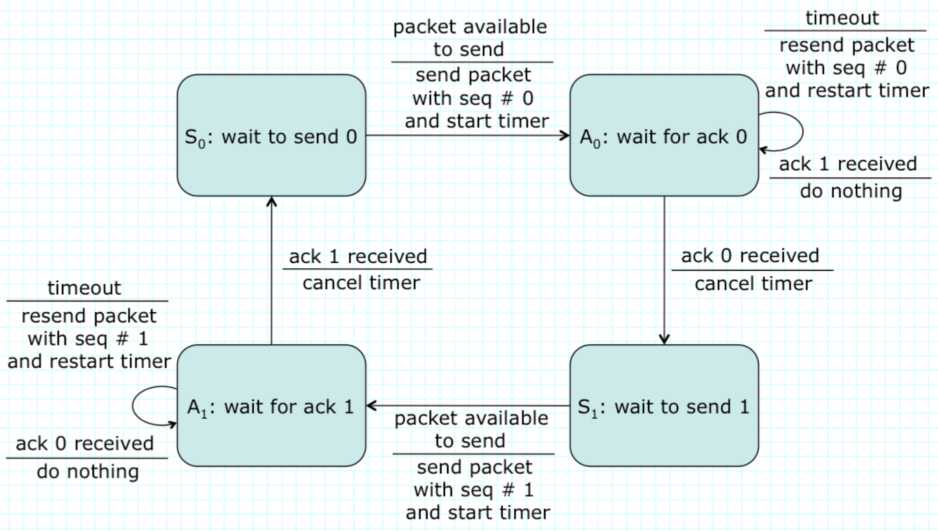


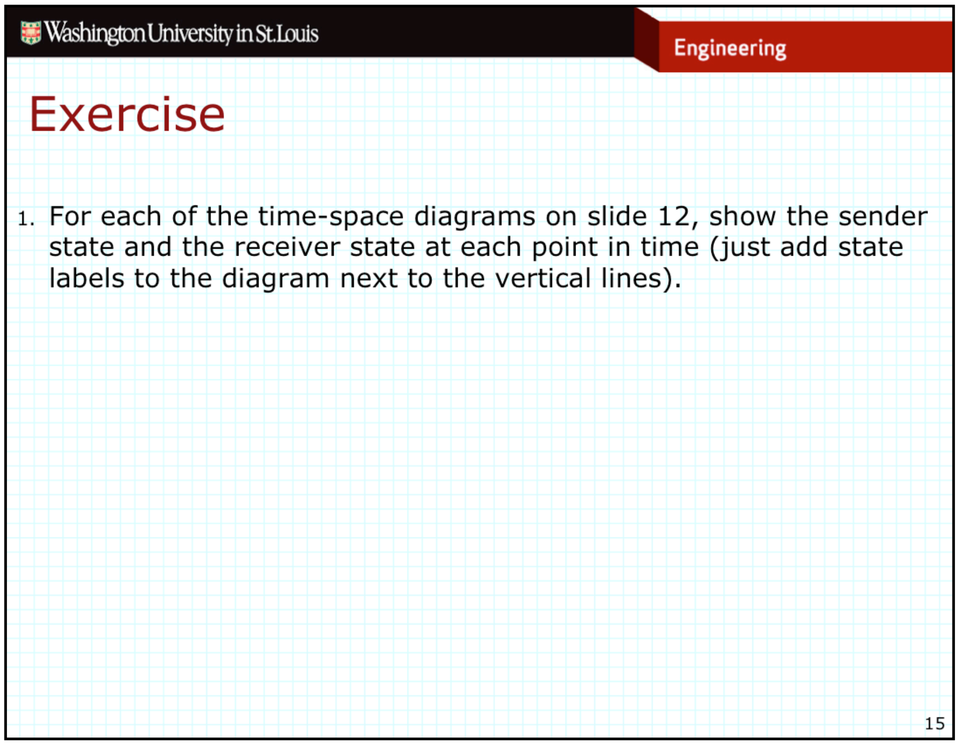
12

RDT 3.0 Receiver Specification



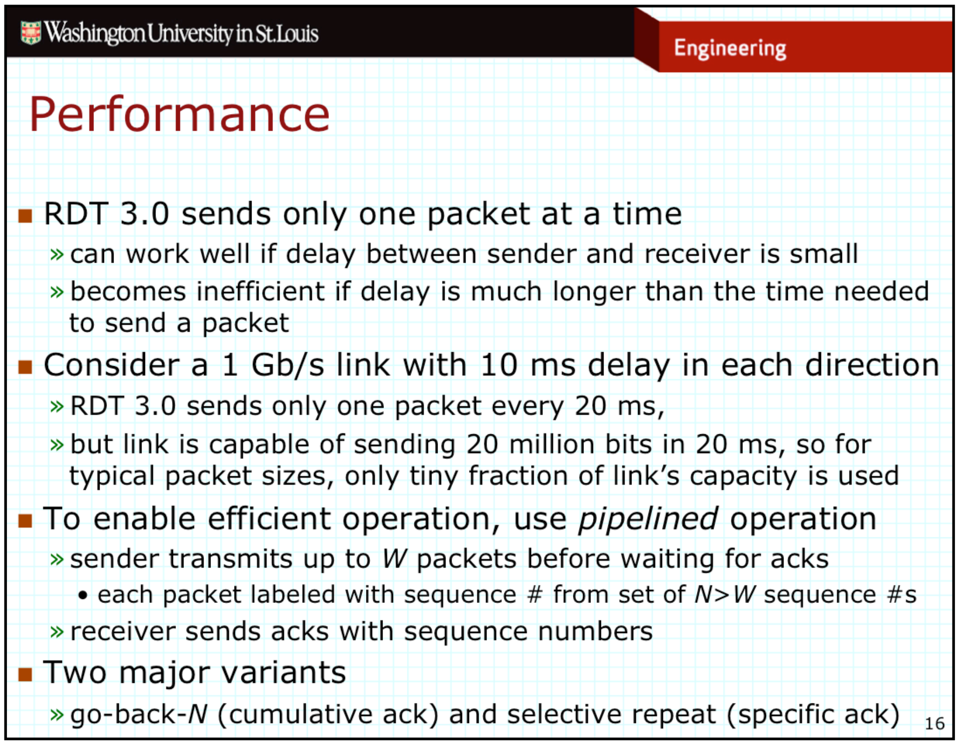
RDT 3.0 Sender Specification





Exercise

1. For each of the time-space diagrams on slide 12, show the sender state and the receiver state at each point in time (just add state labels to the diagram next to the vertical lines).



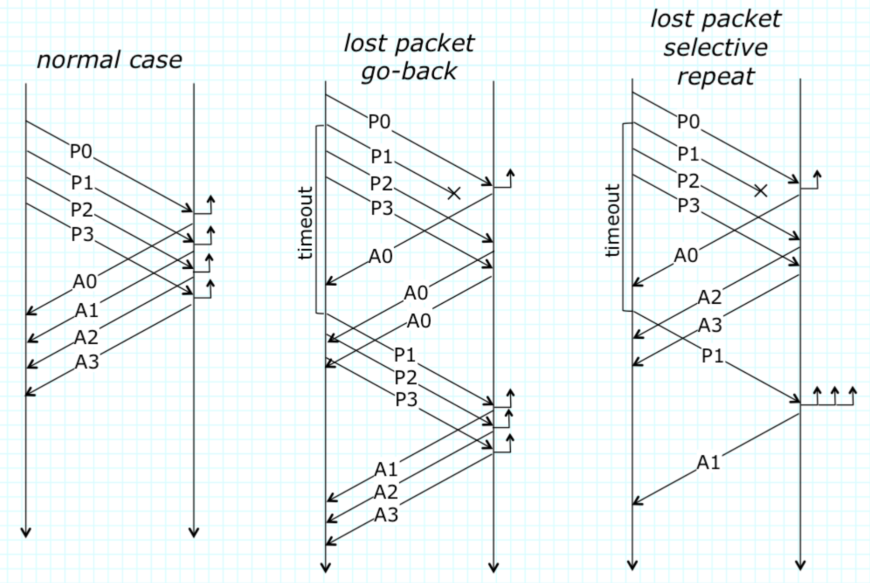
Performance

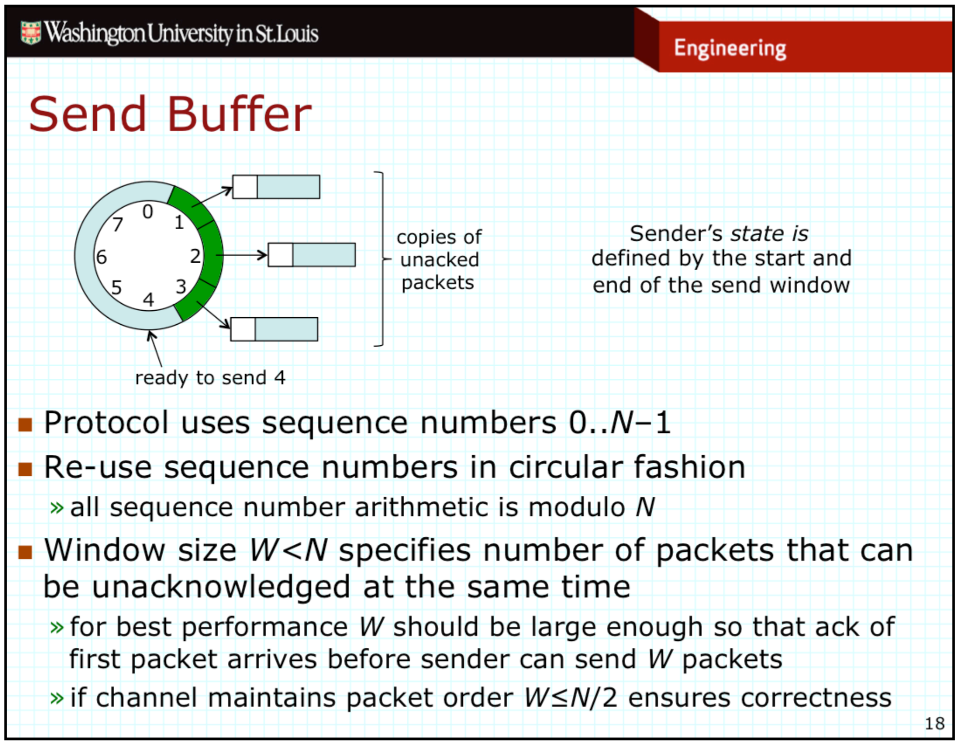
- RDT 3.0 sends only one packet at a time
  - » can work well if delay between sender and receiver is small
  - » becomes inefficient if delay is much longer than the time needed to send a packet
- Consider a 1 Gb/s link with 10 ms delay in each direction
  - » RDT 3.0 sends only one packet every 20 ms,
  - » but link is capable of sending 20 million bits in 20 ms, so for typical packet sizes, only tiny fraction of link's capacity is used
- To enable efficient operation, use *pipelined* operation
  - » sender transmits up to *W* packets before waiting for acks
    - • each packet labeled with sequence # from set of *N*>*W* sequence #s
  - » receiver sends acks with sequence numbers
- Two major variants
  - » go-back-*N* (cumulative ack) and selective repeat (specific ack)

Washington University in St. Louis

Engineering

Pipelined Protocol with *W*=4

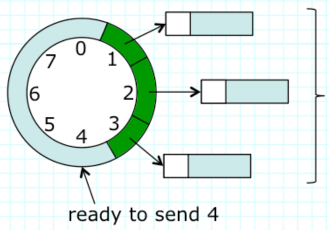




Washington University in St.Louis

Engineering

Send Buffer



copies of unacked packets

Sender's *state is* defined by the start and end of the send window

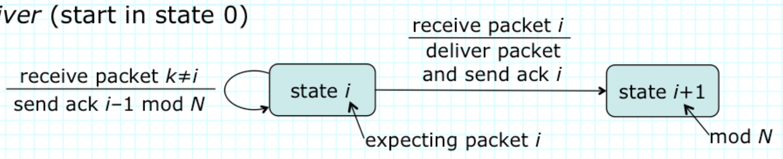
- Protocol uses sequence numbers 0..N–1
- Re-use sequence numbers in circular fashion
  - » all sequence number arithmetic is modulo N
- Window size $W<N$ specifies number of packets that can be unacknowledged at the same time
  - » for best performance W should be large enough so that ack of first packet arrives before sender can send W packets
  - » if channel maintains packet order $W \le N/2$ ensures correctness

18

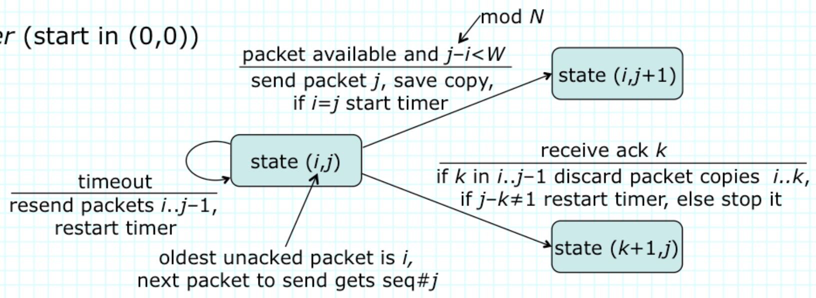
Washington University in St.Louis Engineering

Go-Back-*N* State Machines

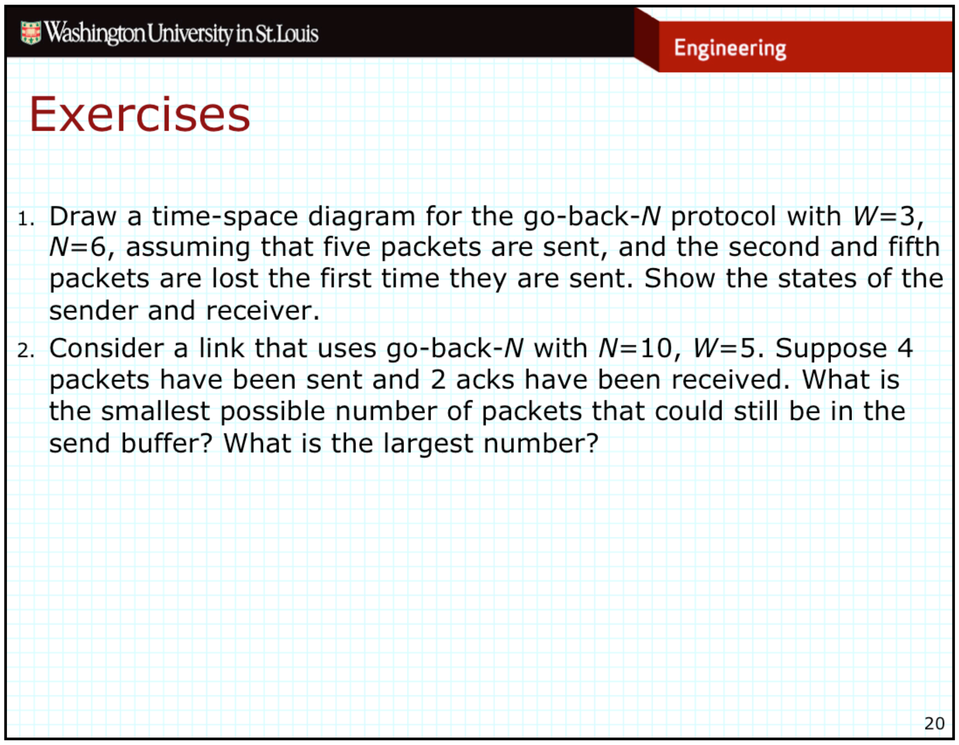
Receiver (start in state 0)



Sender (start in (0,0))


mod N
packet available and $j-i<W$
send packet j, save copy, if $i=j$ start timer
state $(i,j+1)$
timeout
resend packets $i..j-1$, restart timer
state (i,j)
receive ack k
if k in $i..j-1$ discard packet copies $i..k$, if $j-k \neq 1$ restart timer, else stop it
state $(k+1,j)$
oldest unacked packet is i, next packet to send gets seq#j

19

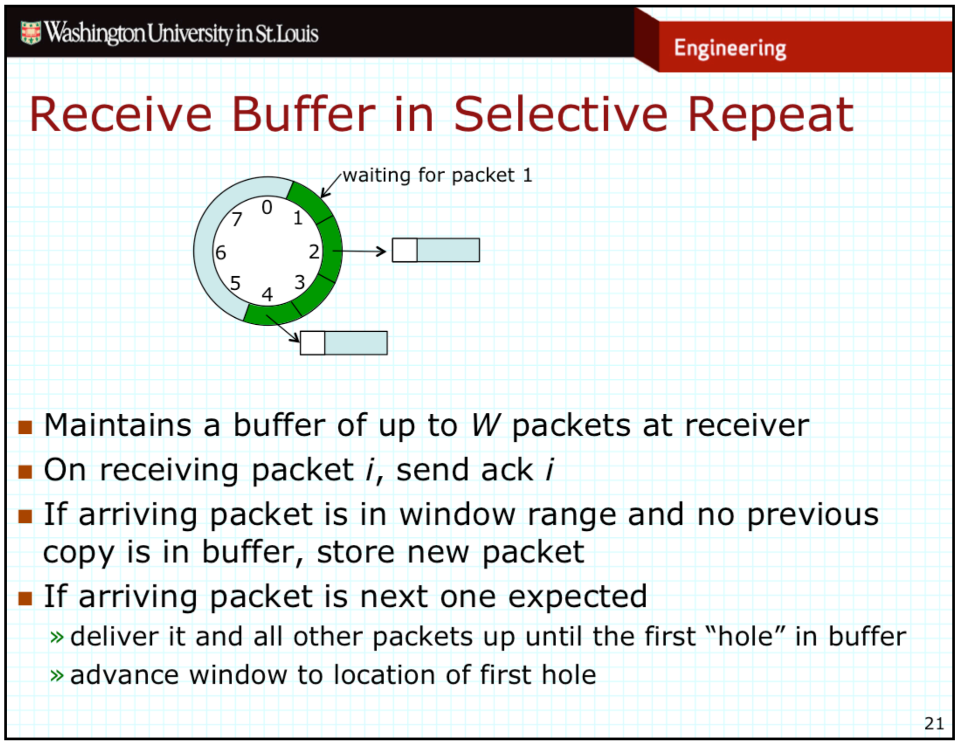

Washington University in St.Louis

Engineering

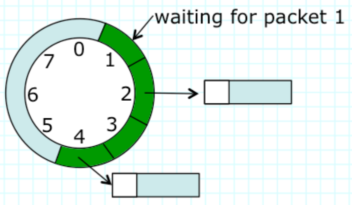
Exercises

1. Draw a time-space diagram for the go-back-*N* protocol with *W*=3, *N*=6, assuming that five packets are sent, and the second and fifth packets are lost the first time they are sent. Show the states of the sender and receiver.
2. Consider a link that uses go-back-*N* with *N*=10, *W*=5. Suppose 4 packets have been sent and 2 acks have been received. What is the smallest possible number of packets that could still be in the send buffer? What is the largest number?

20



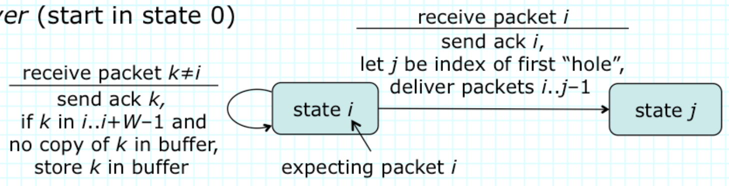
Receive Buffer in Selective Repeat



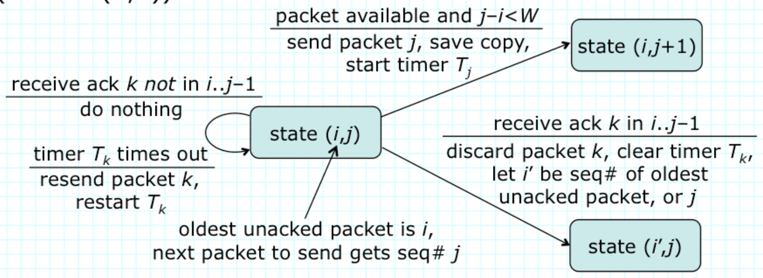
- Maintains a buffer of up to *W* packets at receiver
- On receiving packet *i*, send ack *i*
- If arriving packet is in window range and no previous copy is in buffer, store new packet
- If arriving packet is next one expected
  - » deliver it and all other packets up until the first “hole” in buffer
  - » advance window to location of first hole

Selective Repeat State Machines

Receiver (start in state 0)

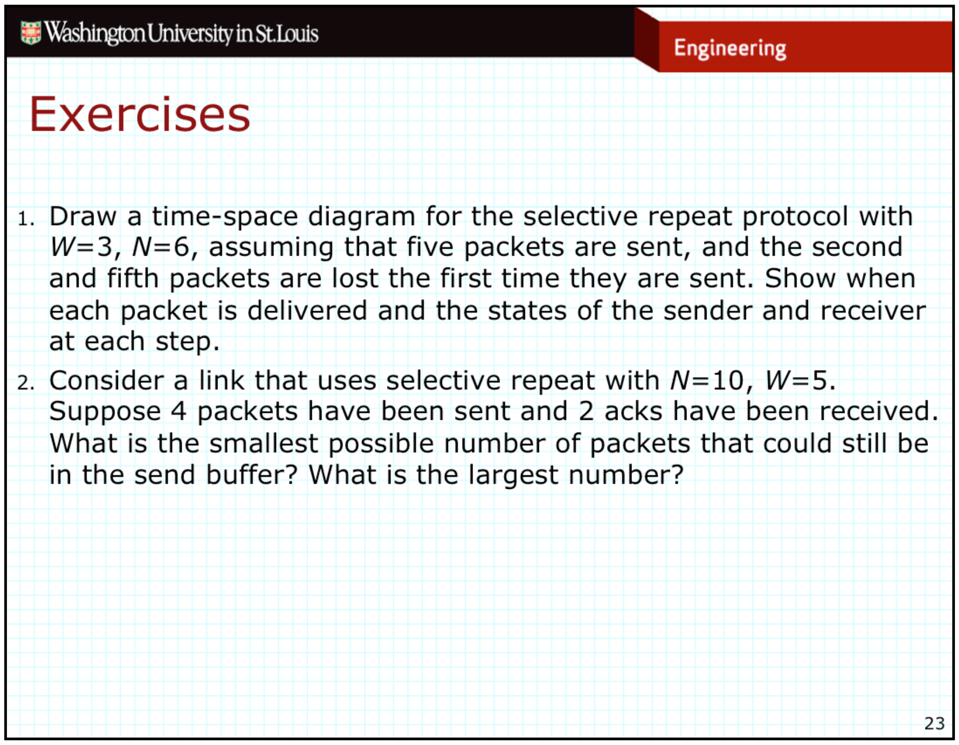


Sender (start in (0,0))



packet available and $j-i<W$
send packet j, save copy,
start timer T_j
state $(i,j+1)$
receive ack k *not* in $i..j-1$
do nothing
state (i,j)
timer T_k times out
resend packet k,
restart T_k
receive ack k in $i..j-1$
discard packet k, clear timer T_k,
let i' be seq# of oldest
unacked packet, or j
oldest unacked packet is i,
next packet to send gets seq# j
state (i',j)

22



Washington University in St.Louis Engineering

Exercises

1. Draw a time-space diagram for the selective repeat protocol with *W*=3, *N*=6, assuming that five packets are sent, and the second and fifth packets are lost the first time they are sent. Show when each packet is delivered and the states of the sender and receiver at each step.
2. Consider a link that uses selective repeat with *N*=10, *W*=5. Suppose 4 packets have been sent and 2 acks have been received. What is the smallest possible number of packets that could still be in the send buffer? What is the largest number?