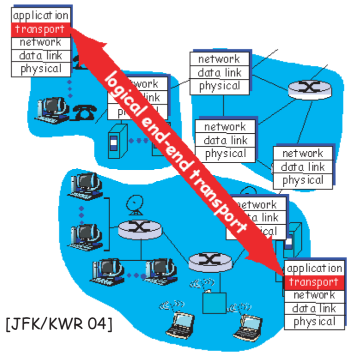
Slide 1

The Transport Layer



Outline

1. Transport layer services
2. Principles of reliable data transfer protocols
3. TCP flow control
4. TCP congestion control

Slide 2

1. Transport layer services (to the application layer)

- Multiplexing and demultiplexing (from/to different processes)
  - See the slides on the Sockets API.

Transport Layer Concept 1

Demultiplexing packets at any host uses transport layer addresses (e.g., TCP port numbers) and network layer addresses (e.g., IP numbers).

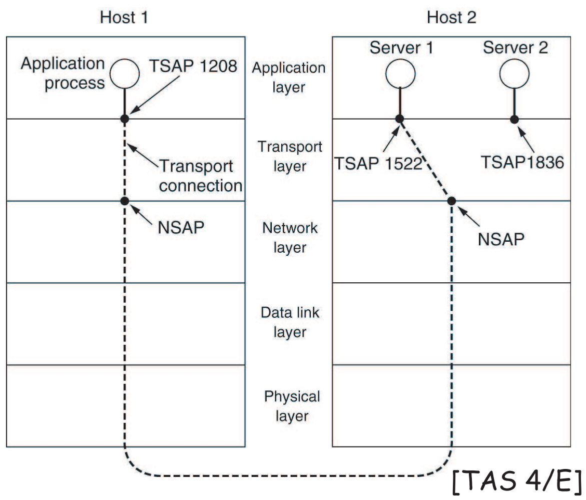
Slide 3

Remark: OSI terminology:

网络服务访问点

- NSAPs (network service access points) : for IP numbers

  传输服务访问点
- TSAPs (transport service access points) : for port numbers of the Internet TCP protocol



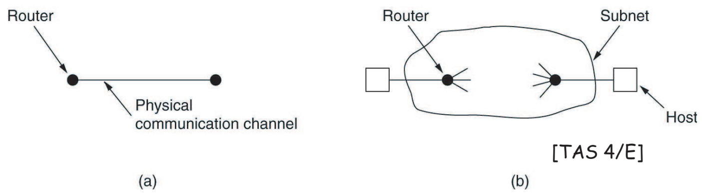
Slide 4

- Reliable data transfer (for connection-oriented transport)
- Flow control
- Congestion control

2. Principles of reliable data transfer (rdt) protocols

2.1 Getting started

- Designing rdt protocols is a common issue between the data link layer and the transport layer

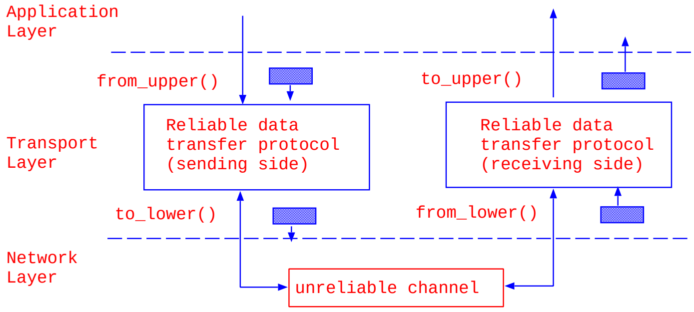


- In both layers, we have

Slide 5

- bit erros,
- packet losses, and
- end-to-end delays

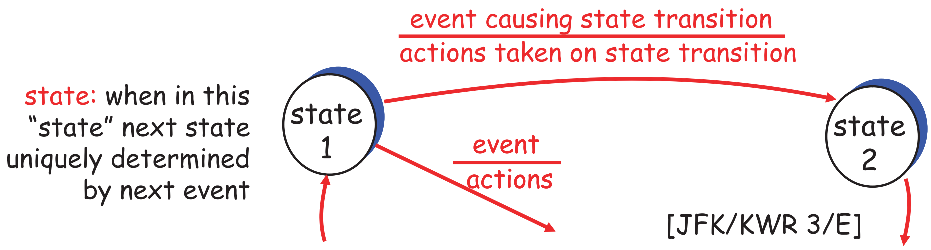
- Service model:



- In textbook:
  - from_upper() ≡ rdt_send(), to_lower() ≡ udt_send()
  - to_upper() ≡ deliver_data(), from_lower() ≡ rdt_rcv()

Slide 6

- Our objective: incrementally develop sender, receiver sides of rdt protocol for various types of channel impairments
- Use of 有限状态机 finite state machines(FSM) :



Slide 7

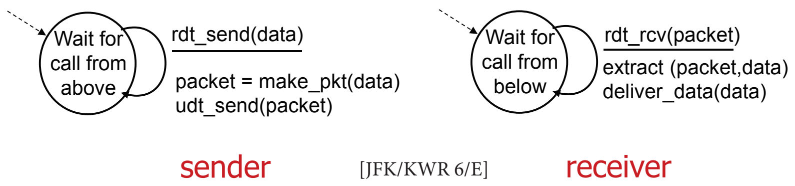
Transport Layer Concept 2

(next few slides)

If a communication channel does not **re-order** *packets then reliable transmission against channel impairments (delays, losses, and errors) can be achieved using packet* **re-transmission** *and* **sequence numbers***.*

Slide 8

- Rdt 1.0: Reliable transfer over a reliable channel
  - Unidirectional data transfer, no bit errors, no loss of packets

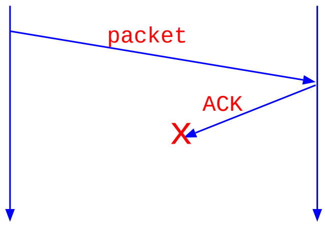


- Rdt 2.0: Channel with bit errors
  - Assumes a hypothetical *asymmetric* channel with bit errors in one direction (from sender to receiver), but reliable in the other direction
  - a stop-and-wait protocol with positive acknowledgments (ACKs) and negative acknowledgments (NAKs)
  - ACKs: receiver explicitly tells sender that pkt received OK (and receiver is ready to receive more)

Slide 9

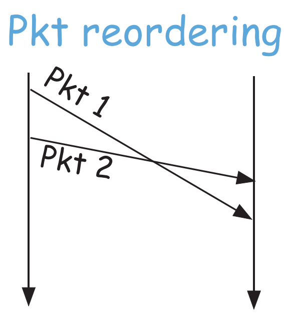
- sender retransmits pkt on receipt of NAK

■ Question: Now, assume the reverse channel is also unreliable (so, we can get garbled ACKs/NAKs, or lose them completely). How to fix the protocol?



- Answer: retransmit a duplicate pkt after timeout.
- But , the transport layer should detect and discard duplicates
- Fix (handling duplicates): sender adds *sequence number* to each pkt, sender retransmits current pkt if ACK/NAK garbled, receiver discards duplicate pkt
- Q: How large should the range of sequence numbers [0,MAX_SEQ] be?
  - ∗ Assume channel does not *re-order* packets:

Slide 10

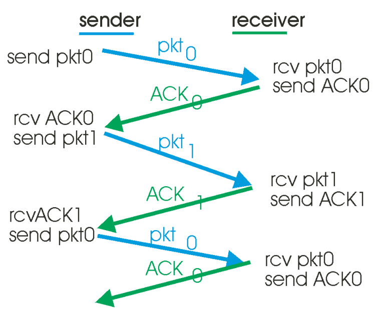


  - ∗ MAX_SEQ= 1 is enough (if the channel does not *re-order* packets)
  - ∗ the protocol is also called alternating-bit protocol.
- Two implementations:
  - ∗ rdt 2.1: a protocol using ACK and NAK
  - ∗ rdt 2.2: a protocol using ACK0 and ACK1 (no NAK)

Slide 11

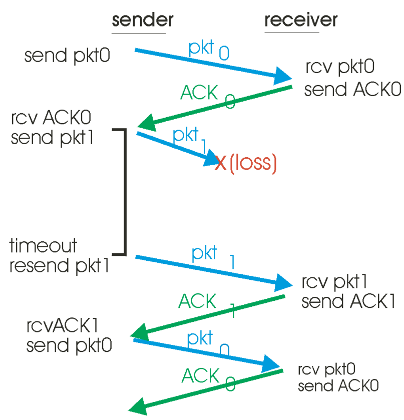
2.2 Rdt 3.0: channels with errors and loss

- Rdt 3.0 is like rdt 2.2 (no NAKs), but rdt 3.0 has timeouts
- Rdt 3.0 in action:



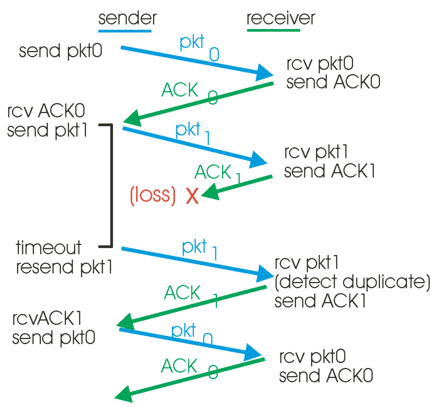
(a) operation with no loss

[JFK/KWR 2/E]

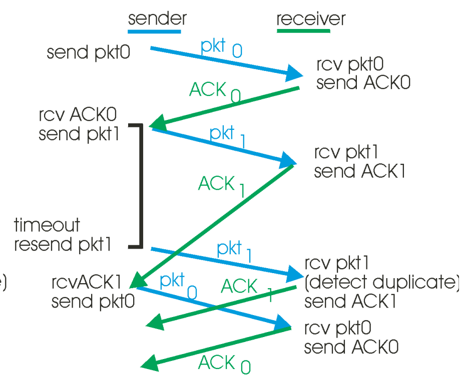


(b) lost packet

Slide 12



(c) lost ACK



(d) premature timeout

[JFK/KWR 2/E]

- Exercise. Show that if the network connection between the sender and receiver can *reorder* messages then the above alternating-bit protocol will not work correctly.

Transport Layer Concept 3

(next few slides)

The throughput (or efficiency) of the stop-and-wait protocol depends on

Slide 13

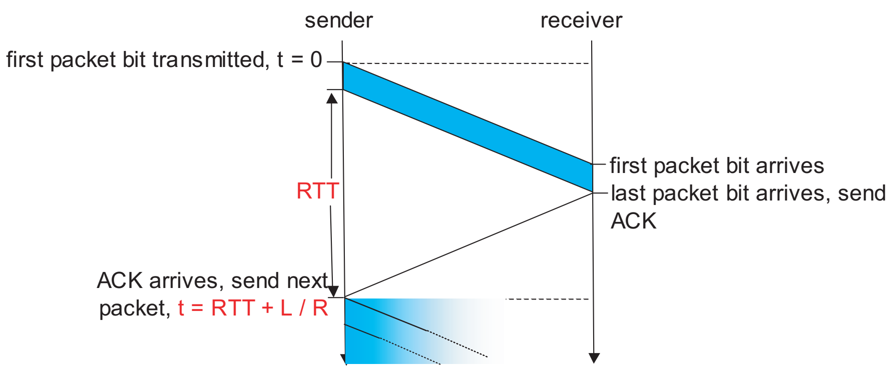
- *the channel's bit error rate (BER),*
- *the average packet length,*
- *the channel's data rate (bits/sec), and*
- *the channel's round trip time (RTT).*

Maximizing the average throughput of a protocol requires careful analysis.

2.3 Performance of Stop-and-Wait Protocols

Slide 14

- Consider the following scenario:
  - L : packet length $= 1$ KByte
  - R : the link speed $= 1$ Gbps
  - RTT : 30 msec



[JFK/KWR 2/E]

Slide 15

- So, packet transmission time

$$T_{pkt} = \frac{L}{R} = \frac{8 \text{ kb/pkt}}{10^9 \text{ bps}} = 8 \text{ microsec}$$

- i.e., roughly, we send 1 KB every 30 msec (33 KByte/sec) over a 1Gbps link (not a good link utilization)

- What about sender's utilization , i.e., the fraction of time the sender is busy sending?

$$utilization = \frac{T_{pkt}}{T_{pkt} + RTT}$$

$$= \frac{.008}{30.008} = 0.00027$$

Note: utilization is a dimensionless quantity.

Definition of Two Channel Parameters

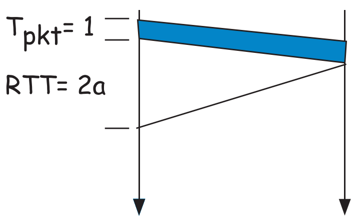
- The following parameters arise frequently in protocol analysis.

Slide 16

- The ratio

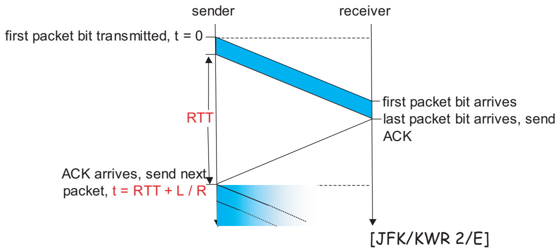
$$a = \frac{\text{Propagation time}}{\text{Pkt transmission time}} = \frac{RTT/2}{T_{pkt}}$$

can be interpreted as the channel propagation delay assuming $T_{pkt} = 1$ time unit.



- The bandwidth-delay product ($R \times RTT$): number of bits that can be sent before the first bit from the receiver is received

Slide 17



- Remark 1. A large bandwidth-delay product encourages sending large packets (to increase channel's utilization). However, as packet size increases, its error probability increases as well, and performance suffers.
- Remark 2. The ratio a is related to the bandwidth-delay product.

Stop-and-Wait Protocol Throughput

- Not in the textbook
- Average Throughput :

Slide 18

- throughput is a productivity measure
- the average throughput is defined as the average number of original pkts transmitted (i.e., excluding retransmissions) per unit time:

$$\overline{Thr} = \frac{\text{number of original pkts send over a long period } T_{sim}}{T_{sim}}.$$

- Error-Free Throughput

$$\overline{Thr}_{error\_free} = \frac{1}{T_{pkt} + RTT} \quad \text{pkts/sec} \tag{1}$$

- Average Throughput with Errors
  - Let's define

$\overline{N}_r$:	the average number of times a pkt is transmitted by the sender

  - A key result then is

Slide 19

$$\overline{Thr} = \frac{\overline{Thr}_{error\_free}}{\overline{N}_r} \quad \text{pkts/sec} \qquad (2)$$

- Example. If $\overline{Thr}_{error\_free} = 1000$ pkts/sec, and the protocol re-transmits each original packet $\overline{N}_r = 5$ times on the average then

$$\overline{Thr} = \frac{\overline{Thr}_{error\_free}}{\overline{N}_r} = \frac{1000}{5} = 200 \text{ pkts/sec.}$$

- Remark on the Error Model. If each transmission is *independently* corrupted with probability q, then N_r is a Geometric random variable, and $\overline{N}_r = 1/(1-q)$.

Slide 20

Transport Layer Concept 4

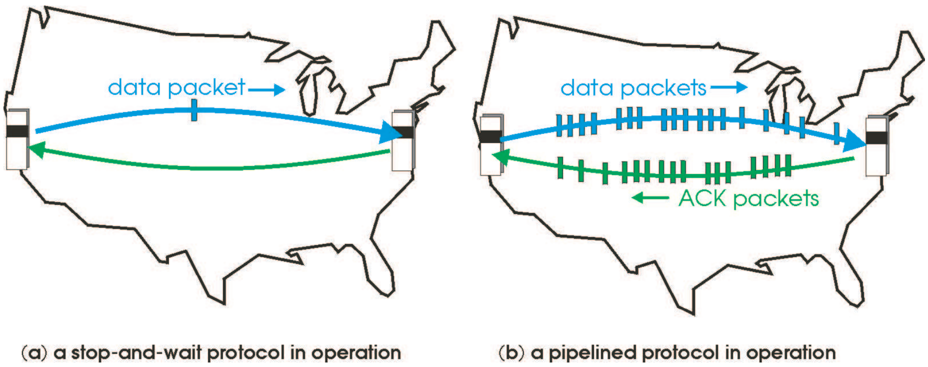
(next few slides)

Sliding window protocols utilize pipelining of packet transmission to increase throughput and channel utilization.

Slide 21

2.4 Pipelined protocols

- Pipelining: sender allows multiple in-flight (yet to be ACK'ed) packets (sender's window size N):

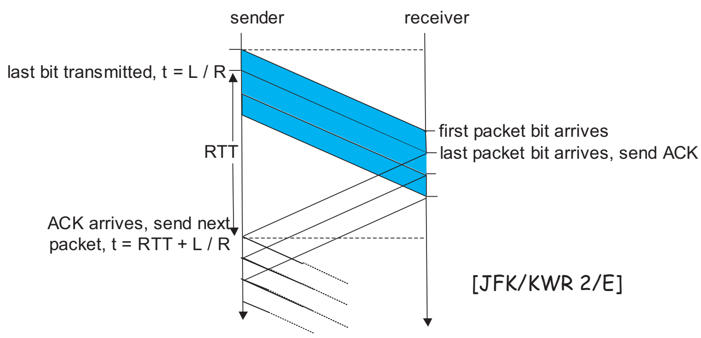


(a) a stop-and-wait protocol in operation (b) a pipelined protocol in operation

[JFK/KWR 2/E]

- Intuitively, pipelining increases utilization: e.g., take $N = 3$ to be the sender's window size then

Slide 22



- $$utilization = \frac{3 * L/R}{RTT + L/R}$$

  (increase by a factor of 3)

- However, pipelining raises a number of questions:
  - What if a frame in the middle of a long stream is damaged or lost?
  - How large should MAX_SEQ be?
  - Should sender timeout for each in-flight packet?

- How many packets should the receiver buffer to recover from errors?

- Two generic forms of pipelined protocols: go-back-N , selective-repeat
- Both are examples of sliding window protocols

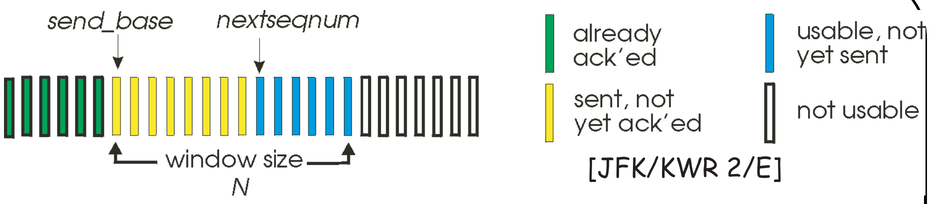
Slide 23

2.5 Go-Back-N

- See Section 3.4.3
- N: maximum number of in-flight packets (i.e., packets sent but not yet ack'ed)
- Some specific features of the implementation:
  - sender uses only one timer
  - receiver buffers only one packet (i.e., ignores out-of-order packets)
- Sender
  - each packet uses a k-bit sequence number (so, $seq\# \in [0, 2^k - 1]$)
  - sender keeps:
    - a "window" of at most N sent but unack'ed packets (say $N = 4$),
    - $base$: sequence number of the oldest unack'ed packet
    - $nextseqnum$: the sequence number of the next packet to be sent

Slide 24

Slide 25



∗ So, at any instant we have 4 sequence number ranges:

· $[0, base - 1]$: for packets already sent and ack'ed

· $[base, nextseqnum - 1]$: for packets sent, but not yet ack'ed

· $[nextseqnum, base + N - 1]$: for packets that can be sent immediately (if available)

· $[base + N, \ldots]$: cannot be used until an unack'ed packet (currently in the pipeline) has been ack'ed

- Also, sender keeps a timer for the oldest transmitted but not yet ack'ed packet.
- On timeout(x) event, sender retransmits pkt x, and all higher seq# pkts in window

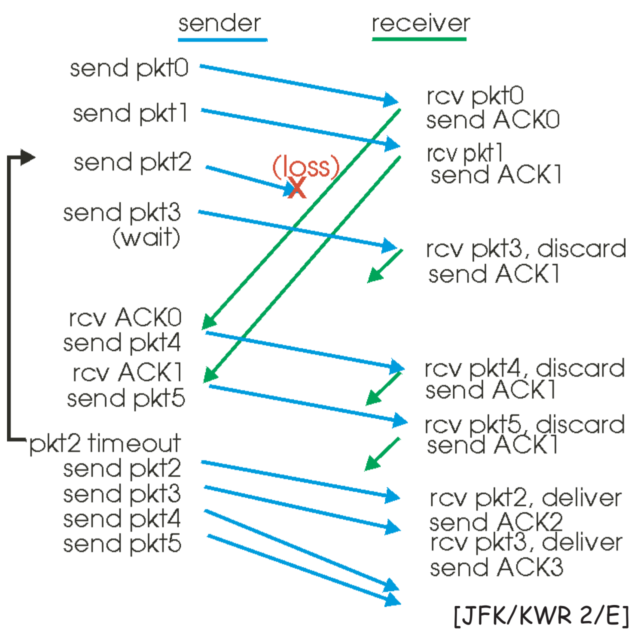
Slide 26

■ Receiver

- has just one buffer, so it discards out-of-order packets
- sends $ACK(x)$ only if x is the highest in-order correctly received pkt
- So, $ACK(x)$ is taken as a cumulative ACK indicating that all packets with a sequence number up to and including x have been correctly received.
- receiver keeps $expectedseqnum$: the sequence number of the next in-order packet

■ Example: $N = 4$

Slide 27



Slide 28

More implementation details

- Let's say, during connection setup sender and receiver agree on some initial conditions: e.g., $base = nextseqnum = 1$ at the sender's side, and $expectedseqnum = 1$ at the receiver's side.
- sender_loop $\{$ /* important events */

  event: receive $ACK(x)$

  (can occur only if the range [base, nextseqnum-1] is not empty)

  - slide window: $base = x + 1$
  - if the range $[base, nextseqnum - 1]$ is empty then stop timer, else restart timer

  event: timeout

  (can occur only if the range $[base, nextseqnum - 1]$ is not empty)

  - send all pkts in the range $[base, nextseqnum - 1]$
  - start timer

event: upper layer requests sending a pkt

- if window is not full (i.e., the range $[nextseqnum, base + N - 1]$ is not empty)
  - $*$ compose and send $pkt[nextseqnum]$
  - $*$ if $(base == nextseqnum)$ start timer
  - $*$ $nextseqnum++$
- else (window is full) block upper layer requests

$\}$

Slide 29

- receiver_loop $\{$ /* important events */

  event: lower layer received $pkt[expectedseqnum]$ uncorrupted

  - send $ACK[expectedseqnum]$
  - $expectedseqnum++$

  event: default (e.g., the received packet is old)

  - send $ACK[expectedseqnum - 1]$
    (equivalently, send last ACK packet)

$\}$

Correctness of Go-Back-N with Modulus $M > N$

- Q: Given a sequence number space $[0, M - 1]$ (e.g., sequence numbers are incremented modulo M), what is the largest allowable sender window N for safe operation?

  Assume pkts do not get out-of-order by the channel.

Slide 30

- To get started: is $N = M$ safe?

Slide 31

Transport Layer Concept 5

(next few slides)

The throughput of a sliding window protocol depends on

- *the channel's bit error rate (BER),*
- *the average packet length,*
- *the channel's data rate (bits/sec),*
- *the channel's round trip time (RTT), and*
- *the size of the sender's window and the receiver's window.*

Maximizing the average throughput of a protocol requires careful analysis.

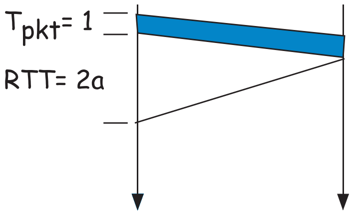
Slide 32

Go-Back-N Throughput

- Remark. A large bandwidth-delay product encourages the use of a large window size N (to increase link utilization).
  - However, when the window size N and the bandwidth-delay product are both large, many packets can be in the link.
  - A single packet error can cause GBN to retransmit a large number of packets (perhaps unnecessarily), and performance suffers.
- We now do a quantitative analysis (not in the textbook) , under the following simplifying assumptions:
  - Forward Channel (sender to receiver): not lossy, but may corrupt packets
  - Reverse Channel : reliable
- Under the above assumptions, the analysis tends to *overestimate* the achieved throughput (by ignoring the effect of timeouts).
- To start, let's define K as the number of pkts sent every $T_{pkt} + RTT$ time.

Slide 33

- Equivalently, if $T_{pkt} = 1$ then K is the number of pkts sent every $1 + 2a$ time units. So,



$$K = \begin{cases} 1 + 2a & \text{if window size } N \geq 1 + 2a \\ N & \text{if window size } N < 1 + 2a \end{cases} \quad (3)$$

- Error-Free Throughput

Since GBN sends K pkts every $T_{pkt} + RTT = T_{pkt}(1 + 2a)$ units then

$$\boxed{\overline{Thr}_{error\_free} = \frac{K}{T_{pkt}(1+2a)} = \begin{cases} \frac{1}{T_{pkt}} & \text{if } N \geq 1 + 2a \\ \frac{N}{T_{pkt}(1+2a)} & \text{otherwise} \end{cases} \quad (4)}$$

- Average Throughput with Errors

Slide 34

- Let's define

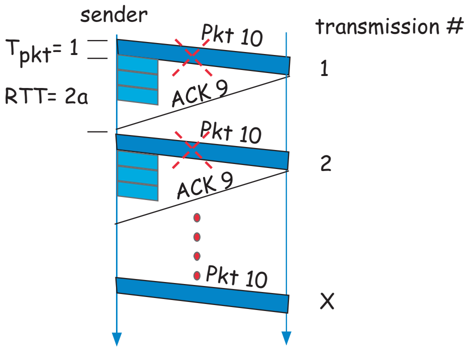
$\overline{N}_x$: avg. number of transmitted pkts to transmit one *original* pkt successfully

- A key result is

$$\boxed{\overline{Thr} = \frac{\overline{Thr}_{error\_free}}{\overline{N}_x} \quad \text{pkts/sec} \quad (5)}$$

- Now, N_x is a derived random variable from X (number of transmissions before a test packet is received successfully):

Slide 35



$$N_x = 1 + (X - 1)K \tag{6}$$

- So,

$$\overline{N}_x = \begin{cases} 1 + (\overline{X} - 1)(1 + 2a) & \text{if } N \geq 1 + 2a \\ 1 + (\overline{X} - 1)N & \text{otherwise} \end{cases} \tag{7}$$

- Normalized Average Throughput: $\overline{Thr}_{norm}$

Slide 36

- Defined as

$$\overline{Thr}_{norm} = \frac{\overline{Thr}}{Thr_{max}}$$

where Thr_{max} is the maximum possible throughput; that is

$$Thr_{max} = \frac{1}{T_{pkt}} \text{ pkts/sec}$$

- Exercise. Verify that if each pkt is *independently* corrupted with probability q then

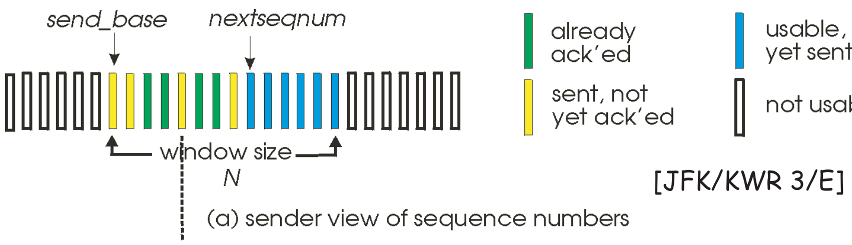
$$\overline{Thr}_{norm} = \begin{cases} \frac{1-q}{1+2aq} & \text{if } N \geq 1 + 2a \\ \frac{N(1-q)}{(1+2a)(1-q+Nq)} & \text{otherwise} \end{cases} \tag{8}$$

2.6 Selective Repeat (SR)

Slide 37

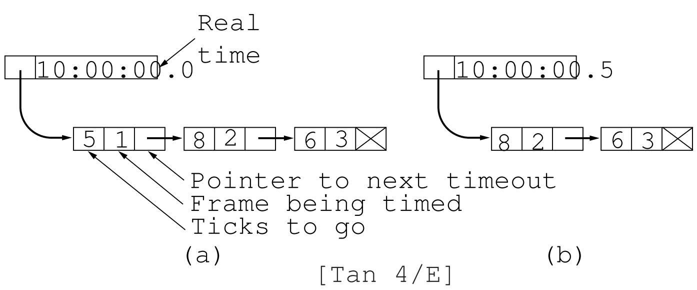
- See Section 3.4.4
- Some features:
  - sender uses a timer for each sent but unack'ed pkt
  - receiver maintains a buffer (window of size N) for out-of-order pkts
  - receiver ACKs each correctly received pkt that can be stored in its buffer (so, ACKs in SR are not cumulative)
  - protocol is NAK-free
- Sender
  - sender keeps:
    - ∗ a "window" of at most N sent but unack'ed packets,
    - ∗ $send\_base$: (as in GBN) sequence number of the oldest unack'ed packet
    - ∗ $nextseqnum$: (as in GBN) the sequence number of the next packet to be sent

Slide 38



- Also, sender keeps a logical timer for each unack'ed pkt

  Remark: simulation of multiple timers in software

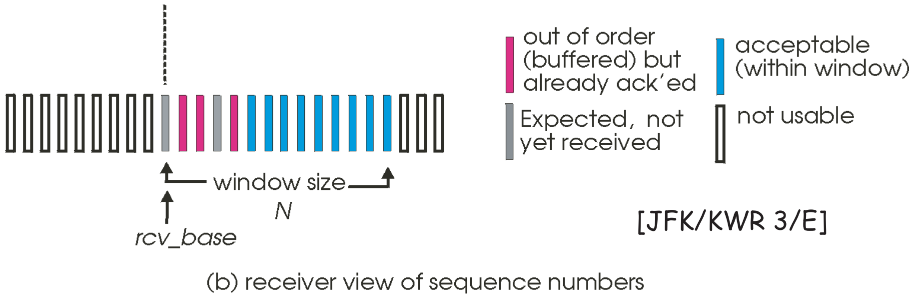


Slide 39

- On timeout(x) event, sender retransmits pkt with sequence number x only

■ Receiver

- Keeps a window (buffer) of some size (may be N as the sender)
- rcv_base : (similar to $expectedseqnum$ in GBN) sequence number of the next in-order packet

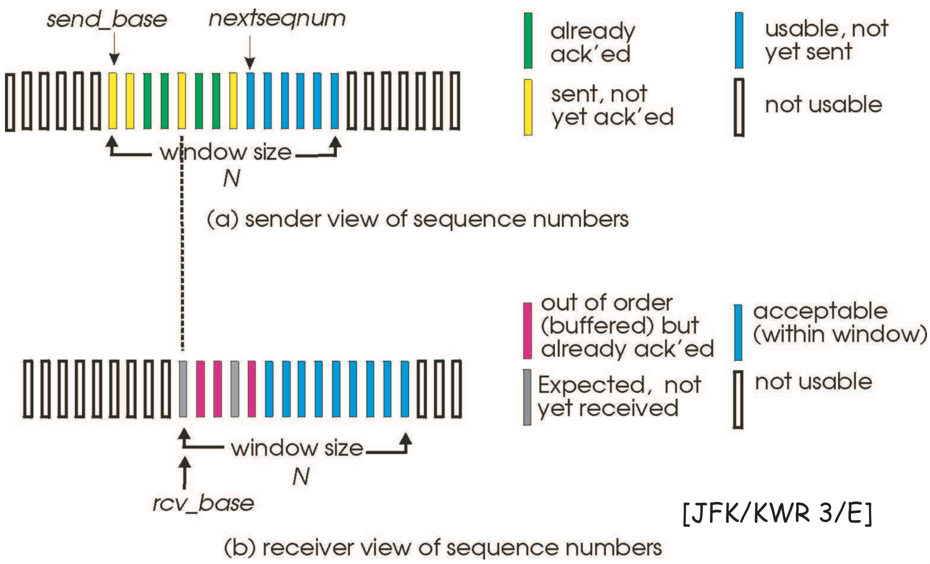


- Receiver buffers out-of-order pkts until any missing pkts (i.e. pkts with lower sequence numbers) are received
- After receiving the missing pkts, a batch of pkts can be delivered in order

Slide 40

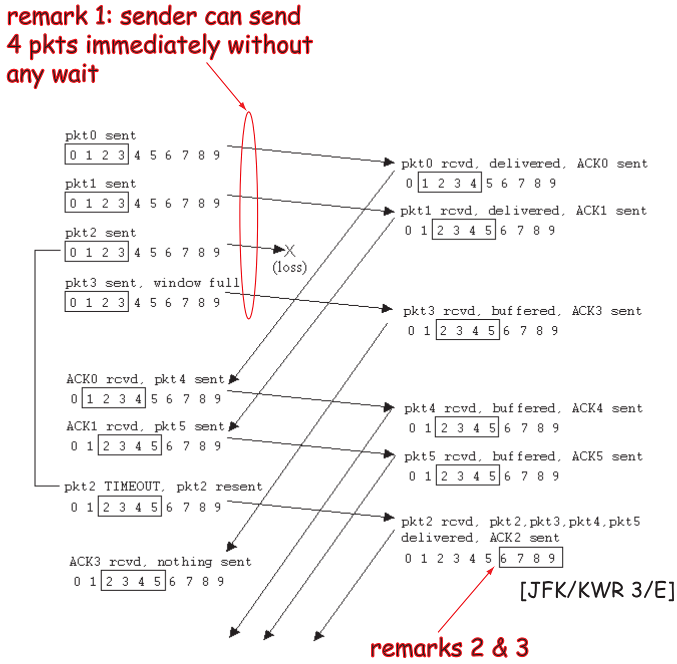
to the upper layer

- Note: in general, sender and receiver may have different views of what has been received correctly, and what has not.



Slide 41

■ Example ($N = 4$): SR operation with a lost pkt



Slide 42

■ Remark 2. Receiver buffers pkts 3, 4, and 5 and delivers them together with pkt 2 to upper layer when pkt 2 is finally received.

■ Remark 3. Receiver sends ACK 2 which has a sequence number below the current window base (rcv_base).

■ Remark 4. receiver should ACK correctly received pkts in two ranges:

- $[rcv\_base, rcv\_base + N - 1]$, and
- $[rcv\_base - N, rcv\_base - 1]$.

In particular, pkts in the second range should not be ignored.

Slide 43

More Implementation Details

sender

data from above :

if next available seq # in window, send pkt

timeout(n):

resend pkt n, restart timer

ACK(n) in [sendbase,sendbase+N]:

mark pkt n as received

if n smallest unACKed pkt, advance window base to next unACKed seq #

receiver

pkt n in [rcvbase, rcvbase+N-1]

send ACK(n)

out-of-order: buffer

in-order: deliver (also deliver buffered, in-order pkts), advance window to next not-yet-received pkt

pkt n in [rcvbase-N,rcvbase-1]

ACK(n)

otherwise:

ignore

[JFK/KWR 2/E]

Slide 44

Transport Layer Concept 6

(next few slides)

Given a sliding window protocol with

N_{sender} = *the sender's window size*

N_{recv} = *the receiver's window size*

M = *the min. sequence number space size required for safe operation*

then M *depends on* N_{sender} *and* N_{recv}.

Correctness of Selective Repeat with Modulus $M > N$

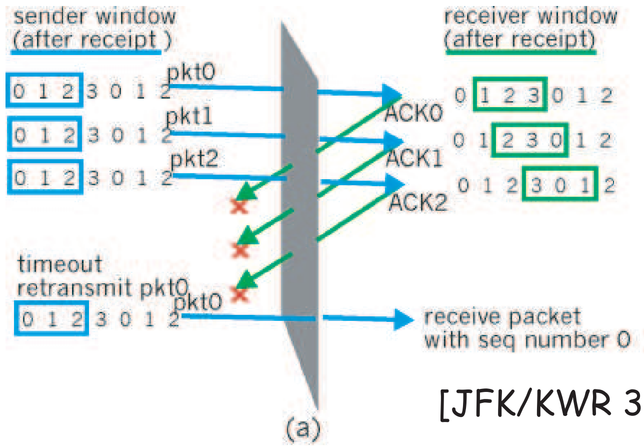
Slide 45

- Q: Given a sequence number space $[0, M-1]$ (e.g., sequence numbers are incremented modulo M), what is the largest allowable sender window N for safe operation?

  Assume pkts do not get out-of-oder by the channel.

- Example showing that $N = M - 1$ is unsafe.
  - Let window size $N = 3$, and
  - let $M = 4$ (so, seq#'s: 0,1,2,3).
  - Case 1: pkt_0 (bottom) is a retransmission of pkt_0 (top).

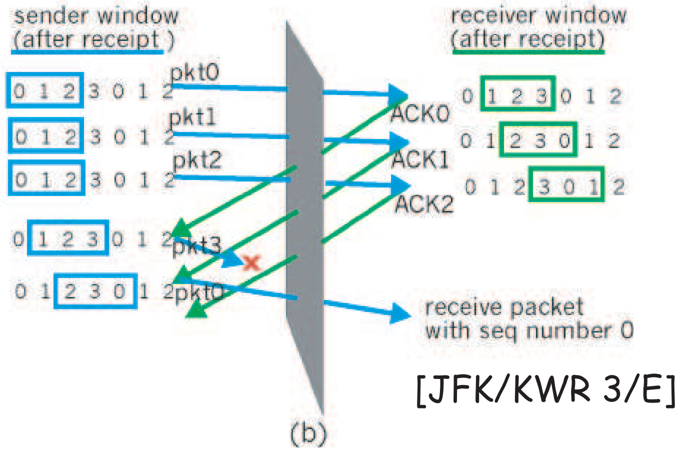
Slide 46



[JFK/KWR 3/E]

- Case 2: pkt_0 (bottom) is different from pkt_0 (top).

Slide 47



- However, the receiver sees no difference in both scenarios!

- Exercise. For SR, Show that choosing a window size $N > \frac{M}{2}$ (e.g., $N = 3$, and $M = 5$) is unsafe, even when the channel does not re-order pkts.

- Note. So far, we have raised and discussed the question: Given the modulus of the seq.#'s M, what is the largest allowable sender window N for safe operation? But we did not obtain final answers.

Slide 48

2.7 Piggybacking ACKs onto Data Pkts

- Both sender and receiver exchange data, so it makes sense to combine ACKs and data in one pkt.

- However, it is important to note that if receiver delays ACKS until a data pkt is ready for transmission then performance suffers (because sender may time out).

Slide 49

Transport Layer Concept 7

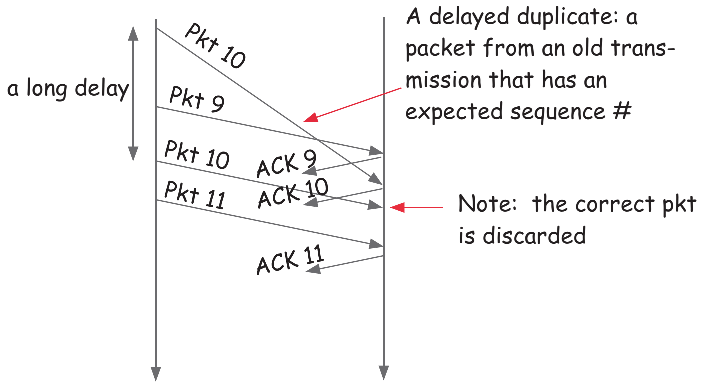
(next few slides)

- *If a communication channel re-orders packets then* delayed duplicate *packets may arise.*
- *Detecting delayed duplicate packets can be achieved by properly limiting packet's lifetime and using a sufficiently large sequence number space.*

Slide 50

2.8 Delayed Duplicate Packets

- Let's define a delayed duplicate pkt as:



- Obviously, the re-ordering caused by delayed duplicates is very problematic.
- So, what can we do?
  We'll examine a solution based on
  - limiting pkt's lifetime in the network, and

Slide 51

- setting the seq #'s space to a sufficiently large range so that no two active packets from the same sender have identical seq #'s.

■ How to limit pkt's lifetime in the network?

Consider the following proposals, based on using:

- t_{gen} : the time a pkt is generated
- TTL : a *time-to-live* value (protocol defined)

Proposals:

1. Sender stores (t_{gen}, TTL) in each transmitted pkt. Routers compute pkt's age. Pkt is discarded it if its age is $\geq TTL$.
2. Sender stores TTL (in secs) in each transmitted pkt. Each router decrements the TTL value by one every second. Routers discard pkts with zero TTL.
3. Sender stores TTL (a small number, e.g., ≤ 255). Each router decrements the TTL by one. Routers discard pkts with zero TTL.

Evaluations:

Slide 52

1. Proposal (1) requires synchronizing router clocks (a difficult problem). In addition, it requires two fields in each pkt.
2. Proposal (2) requires considerable router processing time, and will not be accurate (consider, e.g., cases where a pkt spends a fraction of a sec in a router).
3. Proposal (3) requires the least processing overhead. However, it is a rough method to control the lifetime.

Result: Proposal 3 is the one used in the Internet.

■ Using a sufficiently large Seq.# space: How Large?

- Let's develop a basic framework to answer the question. Let
  - $*$ MPL: the *maximum pkt lifetime* in the network (sec) (note: in TCP, this is called *maximum segment lifetime* (MSL), and is estimated between 30 sec and 2 minutes.)
  - $*$ T_{rcv}: the maximum time a receiver can hold a pkt before sending an ACK (sec)

Slide 53

* R: the maximum transmission rate of the sender (pkts/sec)

- Suppose the sender sends a pkt at time $t = 0$. Then in worst case:
  * At $t_1 = MPL$ the receiver gets the pkt
  * At $t_2 = MPL + t_{rcv}$ the receiver replies
  * At $t_3 = 2MPL + t_{rcv}$ the sender gets the ACK
  * During the interval $[t_0, t_3]$, the sender may have generated as many as $(2MPL + T_{rcv})R$ pkts with new seq #s

  So, the seq # space should be larger than the above value.
- Exercise. Assume that $MPL = 2$ minutes, $T_{rcv} = 500$ msec, and the source transmits at $R = 2$ Mbps. What is the smallest length of the seq # field if the packet size is at least 40 Bytes.
- Exercise. Typically a protocol persists in sending a pkt for sometime, denoted $T_{persist}$ (e.g., 2 minutes). Revise the above framework to take $T_{persist}$ into consideration.

Slide 54

Transport Layer Concept 8

(next few slides)

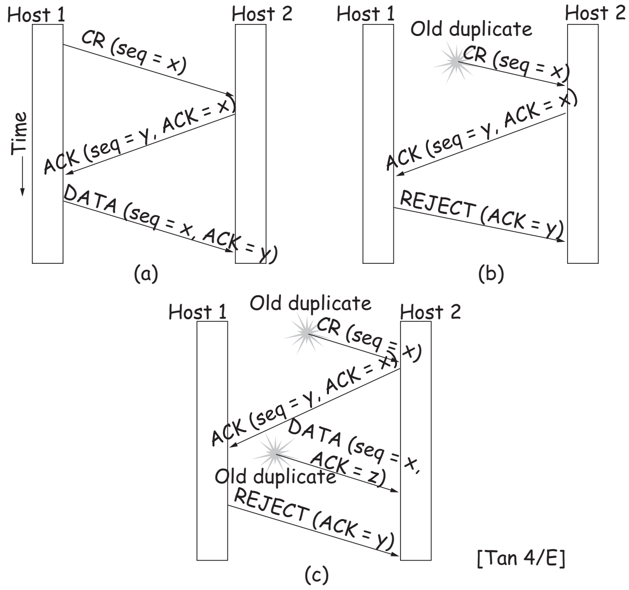
If delayed duplicate packets are handled properly then it is possible to design reliable protocols for:

- *connection setup (the 3-way handshake protocol), and*
- *connection release (the 4-way handshake protocol).*

Slide 55

■ Robustness of the 3-Way Handshake Protocol

- The 3-way handshake protocol is used for connection establishment (exchanging the initial seq #s). It works as in Figure (a):



Slide 56

- To see that the protocol can handle delayed duplicates, let's consider the following cases.
  * Case (b): delayed duplicate is CR (Connection Request)
  * Case (c): delayed duplicates are CR and ACK

2.9 Other Reliability Aspects

Choosing an Initial Seq

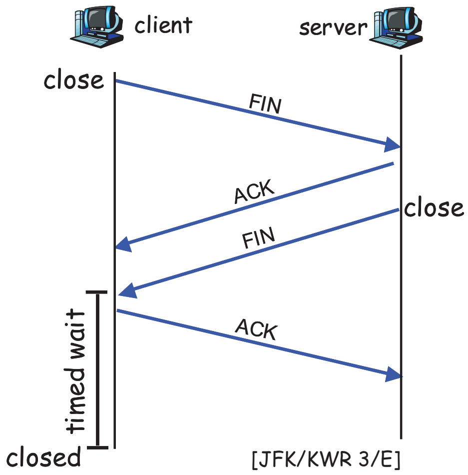
■ Is this an issue? Why?

■ How to fix?

Reliable Connection Release

■ A simple sequence that does not require synchronizing the release time between the two ends goes as follows:

Slide 57



Slide 58

Transport Layer Concept 9

Recovery of a reliable transfer protocol after a host computer crash requires the help of another protocol running in a higher layer.
