

Data Link Layer

The Data Link layer can be further subdivided into:

1. Logical Link Control (LLC): error and flow control
2. Media Access Control (MAC): framing and media access

different link protocols may provide different services, e.g., Ethernet doesn't provide reliable delivery (error recovery)

MAC topics:

- framing and MAC address assignment
- LAN forwarding
- IP to MAC address resolution
 - IP to MAC: Address Resolution Protocol (ARP)
 - MAC to IP: Reverse ARP (RARP), BOOTstrap Protocol (BOOTP), Dynamic Host Configuration Protocol (DHCP)
- media access control



Repeaters and Bridges

Each Ethernet segment is limited to 500 m long by signal attenuation

Repeaters: repeat and strengthen signal (physical layer)

Ethernet only allows 4 repeaters: max 2.5 km. Why?

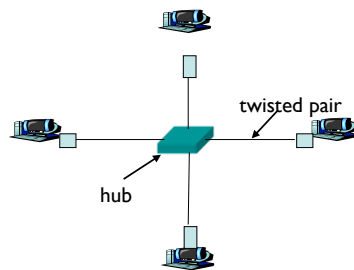
Bridges: equivalence of routers at the data link layer

- forward frames between segments
- unlike routers, only know whether a node is in a segment
- does not propagate interference and collisions (must buffer)
- increase effective/aggregate bandwidth of a LAN by taking advantage of spatial locality
- can connect segments with different MAC protocols

Hubs

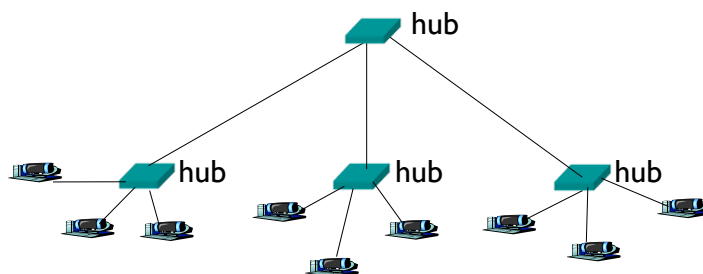
Hubs are essentially physical-layer repeaters:

- bits coming from one link go out all other links
- at the same rate
- no frame buffering
- no CSMA/CD at hub: collision detection left to host adaptors



Interconnecting with Hubs

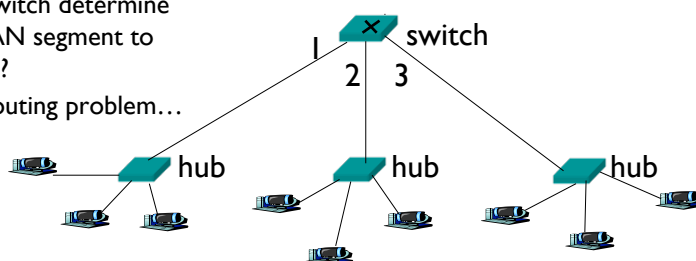
- backbone hub interconnects LAN segments
- extends max distance between nodes
- but individual segment collision domains become one large collision domain
- can't interconnect 10BaseT & 100BaseT



Switches

Link layer router-equivalent:

- stores and forwards Ethernet frames
- examines frame header and selectively forwards frame based on MAC destination address
- when frame is to be forwarded on a segment, uses CSMA/CD to access segment
- transparent: hosts are unaware of presence of switches
- plug-and-play: self-learning, switches do not need to be configured
- How does a switch determine onto which LAN segment to forward frame?
- Looks like a routing problem...



Transparent Bridges/Switches and Backward Learning

How does a bridge know which segment a node is located at?

Each switch has a switch table, entry in switch table:

- <MAC Address, interface, timestamp>
- stale entries in table dropped (TTL can be 60 min)

switch *learns* which hosts can be reached through which interfaces

- when a frame is received, switch “learns” location of sender: incoming interface connects to the LAN segment through which a sender may be reached
- records sender/interface pair in switch table
- called “backward learning”

Frame Filtering/Forwarding

When a switch receives a frame:

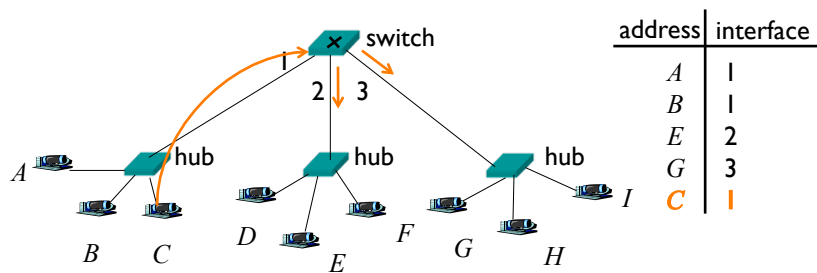
```

look for MAC destination address in switch table
if entry found for destination {
  if destination on segment from which frame arrived {
    drop the frame
  } else {
    forward the frame on interface indicated
  }
} else {
  flood // forward on all but the interface on which the frame arrived
}

```

Switch Example

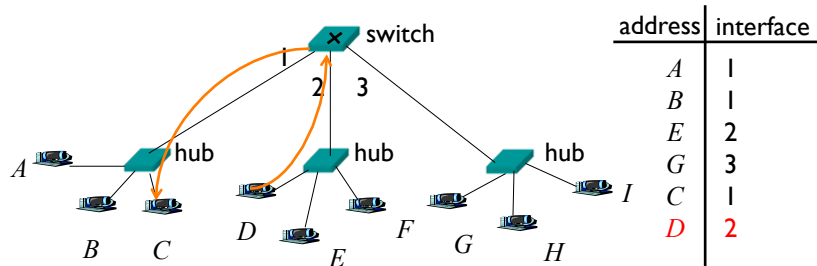
Suppose *C* sends a frame to *D*



Switch receives frame from *C*
 records in switch table that *C* is on interface 1
 because *D* is not in table, switch forwards frame to
 interfaces 2 and 3
 frame received by *D*

Switch Example

Suppose *D* now sends a frame to *C*



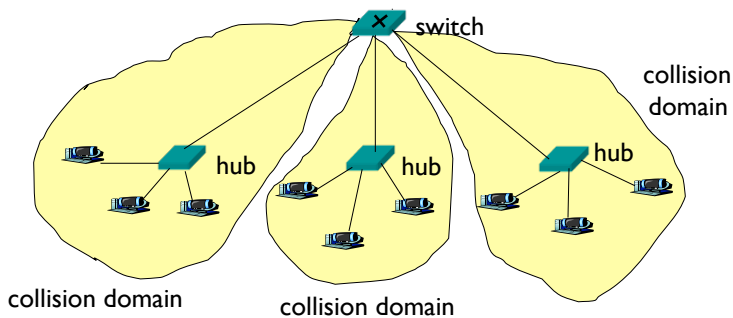
Switch receives frame from *D*
 records in switch table that *D* is on interface 2
 because *C* is in table, switch forwards frame only to
 interface 1
 frame received by *C*

Switch: Traffic Isolation

switch installation breaks subnet into LAN segments

switch filters packets:

- same-LAN-segment frames are not usually forwarded onto other LAN segments
- segments become separate collision domains



Switches: Dedicated Access

Hosts have direct connection to switch

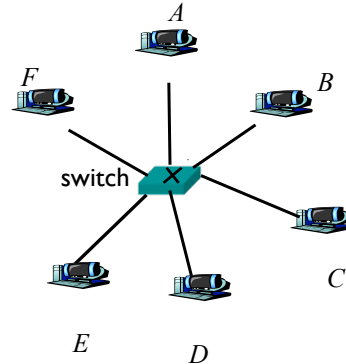
No collisions; full duplex

Switching: *A-to-D* and *B-to-E* simultaneously, no collisions

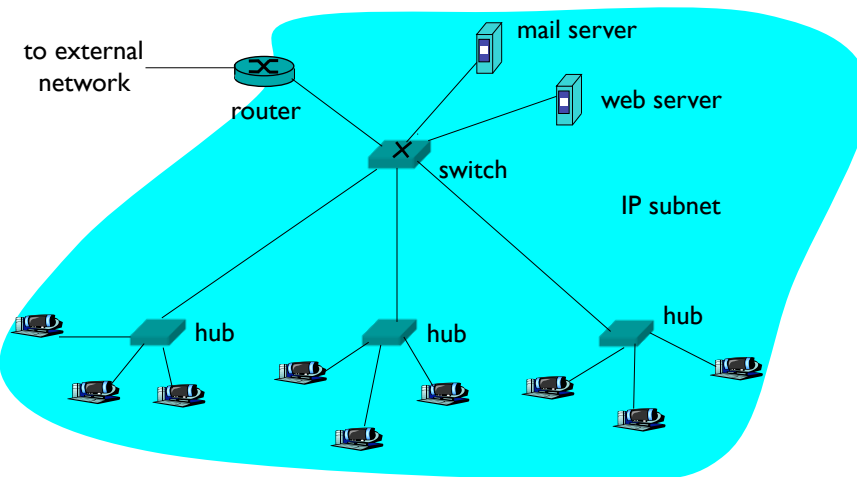
Cut-through switching: frame forwarded from input to output port without storing

- slight reduction in latency

switches can support combinations of shared/dedicated and 10/100/1000 Mbps interfaces



Example Enterprise Network Switch/Hub Installment

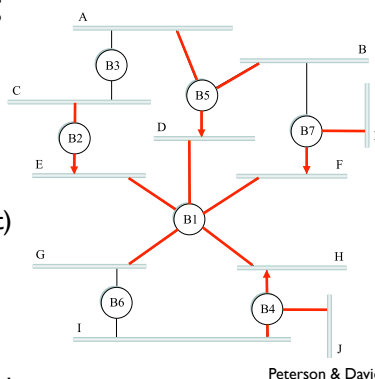


Switches and Spanning Tree

LANs may form cycles, causing broadcast storm

Bridges/switches detect cycles by doing distributed spanning tree computation:

- all bridges broadcast serial #, root ID, cost to root
- bridge with lowest serial # becomes root of tree
- all bridges determine root port (port to root)
- the spanning tree consists of bridges (nodes) and root port (links)



Peterson & Davie

Forwarding on the tree:

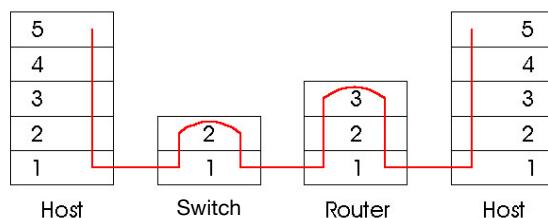
- each LAN determines a designated bridge by lowest cost to root, break tie by serial #
- forward frames only on links that are part of the tree

Switches vs. Routers

Both store-and-forward devices

Given bridges, why do we still need routers?

- routers are network layer devices (what does this mean?)
 - routers maintain routing tables, implement routing algorithms
- switches are link layer devices
 - switches maintain switch tables, implement filtering, backward learning algorithms



Data Link Layer

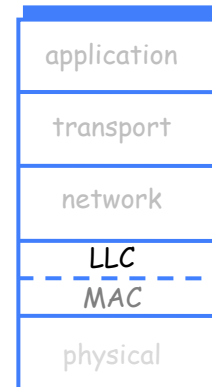
The Data Link layer can be further subdivided into:

1. Logical Link Control (LLC): error and flow control
2. Media Access Control (MAC): framing and media access

different link protocols may provide different services, e.g., Ethernet doesn't provide reliable delivery (error recovery)

MAC topics:

- framing and MAC address assignment
- LAN forwarding
- IP to MAC address resolution
 - IP to MAC: Address Resolution Protocol (ARP)
 - MAC to IP: Reverse ARP (RARP), BOOTstrap Protocol (BOOTP), Dynamic Host Configuration Protocol (DHCP)
- media access control



Ethernet: Connectionless Service

No handshaking between sending and receiving adaptor

Receiving adaptor doesn't send ACKs or NACKs to sending adaptor

- stream of datagrams passed up to network layer can have gaps
- gaps will be filled if application uses reliable transport layer
- otherwise, application will see the gaps

Other data link protocols may provide error correction and flow control

Transmission Errors

Three kinds of transmission errors:

1. sent signal changed (received wrong data)
2. sent signal destroyed (doesn't receive data)
3. spurious signal created (received random data)

Caused by noise on the channel:
interference, cosmic rays

Error Control

Ways to detect errors, general idea:

- sender computes some info from data
- sender sends this info along with data
- receiver does the same computation and compares it with the sent info

Not often used for largely reliable links,
but useful for unreliable links such as wireless

Used at the transport layer also
(the Internet is an unreliable "link")

Field: Information Theory

Error Control

Two types of error control:

1. error detecting code
2. error correcting code (ECC),
a.k.a. forward error correction/control (FEC)

Examples error detecting code:

- parity check
- checksum
- cyclic redundancy check (CRC)

Error Control

Trade-offs between alternate methods:

- complexity of info computation,
- bandwidth transmission overhead, and
- degree of protection (# of bit errors that can be detected)

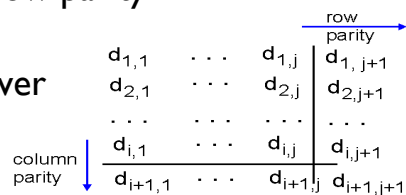
No error detection method is fool-proof

Parity Check

- uses an extra bit (**parity bit**) for error checking
- even parity: total # of 1 bits (incl. the parity bit) is an even number
- odd parity: total # of 1 bits is odd
- single-bit parity examples:
 - 0100101, even-parity bit =
 - 0101101, even-parity bit =
- what happens when an error is detected?
 - discard data and if reliability is required, have sender retransmit
- problem: can not detect even # of flipped bits

2D Parity Check as ECC

- generates both a horizontal/row parity and a vertical/column parity
- both parity info sent to receiver
- receiver can detect *and correct* single-bit errors
- problem: can not detect even # of flipped bits



101011	101011	
111100	101100	→ parity error
011101	011101	
101010	101010	
no errors	parity error	
	correctable single bit error	

Error Correction vs. Detection

ECC generally requires more redundant bits than just detection

It is generally cheaper to retransmit data only when error has been detected than to transmit redundant data *all the time*

FEC is most useful when:

1. link is very noisy, e.g., wireless link
2. retransmission will take too long, e.g.,
 - satellite communication
 - deep space probe transmission
 - real-time audio/video streaming

Checksum

Used also by TCP and UDP

Sender treats data as a sequence of integers and computes their (1's complement) sum

Example: 16-bit checksum

- the string "Hello world." has an ASCII representation of [48 65 6C 6C 6F 20 77 6F 72 6C 64 2E]
- checksum: $4865 + 6C6C + 6f20 + 776F + 726C + 642E + \text{carry} = 71FC$

Advantages:

- ease of computation (only requires addition)
- small amount of additional info to carry: one additional 16-bit or 32-bit integer

Checksum

Disadvantage:

- with 16-bit checksum, 1 in 64K corrupted packet will not be detected (probability of a random 16-bit number matching the checksum of a corrupted packet is $1/2^{16}$)

Data Item In Binary	Checksum Value	Data Item In Binary	Checksum Value
00001	1	00011	3
00010	2	00000	0
00011	3	00001	1
00001	1	00011	3
totals	7		7

⇒ under current Internet conditions (error rate etc.), 1 in every 300M packet accepted corrupted!

Mogul (1992) measured on a busy NFS server that has been up 40 days:

Layer	# checksum errors caught	~#pkts
ethernet	(CRC) 446	1.7×10^8
IP	14	1.7×10^8
UDP	5	1.4×10^8
TCP	350	3×10^7

Cyclic Redundancy Check

Goal of any error detection/correction code: maximize probability of detecting error with minimal redundant info

32-bit CRC protects against most bit errors in messages thousands of bytes long, also used in storage systems (CD, DVD)

CRC is based on finite fields math

Consider a binary message as a representation of an n -degree polynomial, with the coefficient of each term being 1 or 0 depending on the bit in the message, with the most significant (leftmost) bit representing the highest degree term

- For example: 1011 represents $1x^3 + 0x^2 + 1x^1 + 1x^0 = x^3 + x + 1$

An m -bit message represents a polynomial of $m-1$ degree

Polynomial Arithmetic

The math says you can divide one such polynomial by another such polynomial of lower or equal degree by dividing the binary representation of the polynomials, e.g., to divide $x^5+x^3+x^2+x$ by x^3+1 , divide 101110 by 1001

Polynomial arithmetic is done using modulo-2 arithmetic, with no carry and borrow: $1+1 = 0+0 = 0$ and $1+0 = 0+1 = 1$, e.g.,

10011011	11110000	01010101
11001010 +	10100110 -	10101111 -
-----	-----	-----
01010001	01010110	11111010

Note that both addition and subtraction are identical to XOR

Constructing CRC

Let's call the polynomial to be divided T and the divisor/generator polynomial G

Let t be the number of bits in T and $r + 1$ be the number of bits in G , $t \geq r + 1$

Let's call the remainder of T/G , R ; R is of r bits

Want: the polynomial represented by the message to be exactly divisible by G , such that if the receiver divides the message by G and the remainder is not 0, it will know the message has been corrupted

$M = (T-R)$ is exactly divisible by G

Constructing CRC

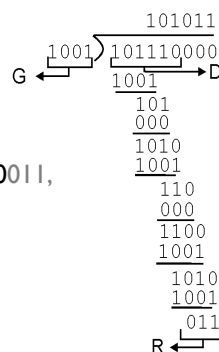
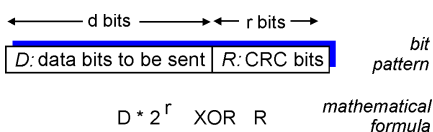
Recall: multiplying a number by 2 is the same as shifting it left by 1 bit

Let D be the message to be sent, e.g., $D = 101110$

Construct T as $D \cdot 2^r$, D shifted left by r bits, e.g., $r = 3$, $T = 101110000$

Let $G = 1001$, compute R , the remainder of T/G , by doing the long-division, with modulo-2 arithmetic, e.g., $R = 011$

Now $M = (T-R) = (D \cdot 2^r - R) = (D \cdot 2^r \text{ XOR } R)$, e.g., 101110011 , is exactly divisible by G



How to Choose G ?

Let the string of bit errors introduced be represented as polynomial E

Error will not be detected only if $T+E$ is exactly divisible by G

Want G that makes this unlikely. What's known:

- if x^r and x^0 terms have non-zero coefficients, G can detect all single-bit errors
- as long as G has a factor with at least 3 terms, it can detect all double-bit errors
- as long as G contains the factor $(x+1)$, it can detect any odd number of errors
- G can detect any burst (sequence of consecutive) errors of length $< r$ bits

Usually, you just look up a commonly used G , e.g., Ethernet uses CRC-32

CRC-32: 100000100110000010001110110110111

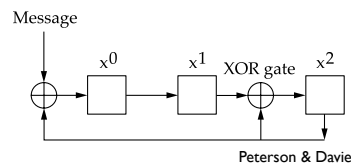
CRC-CCITT: 10001000000100001

CRC Hardware Implementation

CRC can be cheaply implemented in hardware by implementing the long-division to compute R as a combination of linear feedback shift register (LFSR) and XOR gates

The shift registers and XOR gates represents the G :

- the 0-th term of G occupies the leftmost bit of the shift registers
- each XOR gate represents a modulo-2 addition in G
- the message is fed into the circuit most significant (leftmost) bit first
- each bit of the message causes the current content of the shift registers to be shifted right by one bit
- when the message is exhausted, the shift registers contain R
- for example, computing CRC with $G = x^2 + 1$ can be implemented as:



Link Layer Services

Half-duplex and full-duplex

- with half duplex, nodes at both ends of link can transmit, but not at same time

Framing, link access:

- encapsulate datagram into frame, adding header, trailer
- channel access if shared medium
- “MAC” addresses used in frame headers to identify source, dest
 - different from IP address!

Error Detection:

- errors caused by signal attenuation, noise.
- receiver detects presence of errors:
 - signals sender for retransmission or drops frame

Link Layer Services

Error Correction:

- receiver identifies and corrects bit error(s) without resorting to retransmission

Flow Control:

- pacing between adjacent sending and receiving nodes

Reliable delivery between adjacent nodes

- seldom used on low bit error link (fiber, some twisted pair)
- wireless links: high error rates
- Q: why both link-level and end-end reliability?

Flow Control

What is flow control?

- receiver telling sender to slow down

Why do you need flow control?

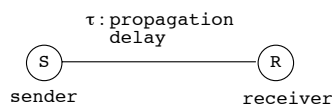
- slow or busy receiver
- don't want to overflow receiver's buffer

Flow control protocols at data link layer (single hop):

- XON/XOFF
- Stop & Wait Protocol (SWP)
- Sliding Window Protocol

Similar issues and mechanisms apply at the transport layer

XON/XOFF



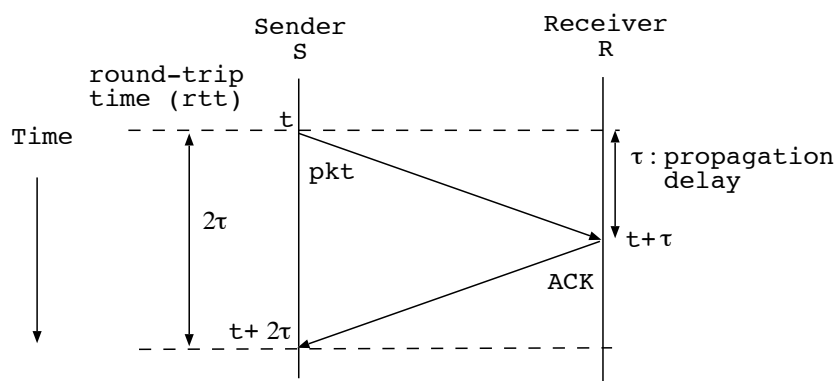
Algorithm:

- S sends stream of data
- R sends XOFF, S stops transmission
- R sends XON, S resumes transmission

Works OK if τ is small, otherwise sender can overrun receiver (Why?)

Stop and Wait (S&W) Protocol

After each pkt, sender must wait for acknowledgment (ACK) before sending the next pkt



Stop & Wait Performance

Disadvantages:

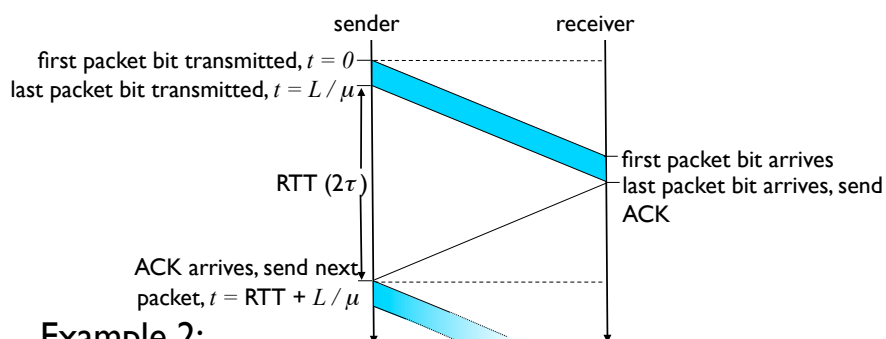
- slow
- must wait for ACK even if no overrun
- max transmission bandwidth 1 pkt/rtt

Performance ok if τ is small, else inefficient

Example 1:

- link bandwidth (μ) = 1 Mbps, with pkt size (L) = 1 Kbits, transmission time is $L/\mu = 1$ ms
- if rtt (2τ) = 9 ms, we can send 100 pkts/sec
- the throughput (T_g) is 100 Kbps (10% of capacity)

Stop & Wait Performance



Example 2:

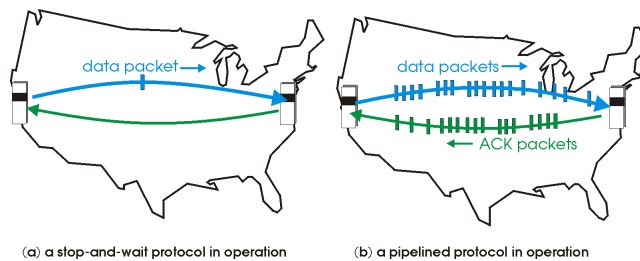
- link bandwidth (μ) = 1 Gbps, with pkt size (L) = 8 Kbits, transmission time is $L/\mu = 8 \mu\text{s}$
- sender utilization (U_s), fraction of time sender is sending:

$$U_s = \frac{L / \mu}{2\tau + L / \mu} = \frac{8 \cdot 10^3 / 10^9}{30 + 8 \cdot 10^{-6}} = 0.00027$$

Sliding Window: Pipelined Flow Control

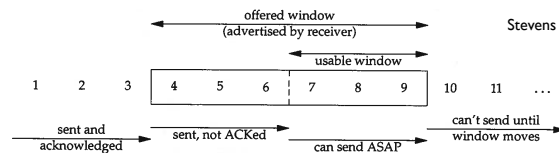
Pipelining: sender allows multiple, “in-flight”, yet-to-be-acknowledged pkts

- range of sequence numbers must be increased
- buffering at sender and/or receiver



Sliding Window

Send w number of pkts before waiting for an ACK (can have w outstanding, i.e., unACKed, pkts)



On receiving an ACK, slide window (over data) by 1 pkt (S&W is sliding window with $w = 1$)

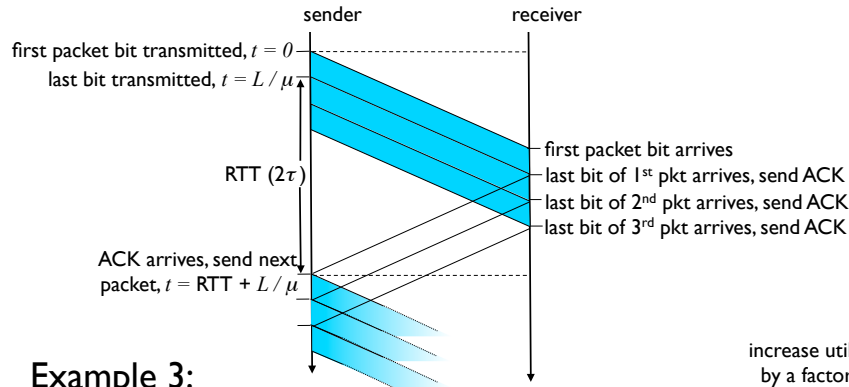
Throughput of the sliding window protocol (T_w):

$$T_w = T_g * w$$

send window size w limited by buffer size at receiver (w_R):

$$T_w = T_g * \text{MIN}(w, w_R)$$

Pipelining: Increased Utilization



Example 3:

- link bandwidth (μ) = 1 Gbps, with pkt size (L) = 8 Kbits, transmission time is $L/\mu = 8 \mu\text{s}$, window size (w) = 3
- sender utilization (U_s), fraction of time sender is sending:

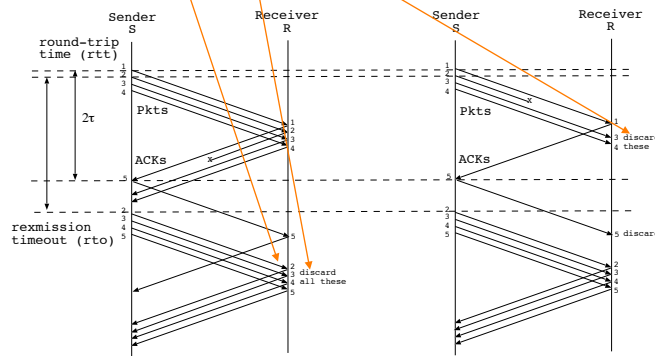
$$U_s = \frac{w * L / \mu}{2\tau + L / \mu} = \frac{3 * 8 \cdot 10^3 / 10^9}{30 + 8 \cdot 10^{-6}} = 0.0008$$

increase utilization by a factor of 3!

Go-Back-N

Receiver:

- remembers next expected seq#
- ACKs and delivers to app in-order pkts
- discards out-of-order pkts \Rightarrow no buffering
- ACKs out-of-order packets if seq# is smaller than next expected number (why?)

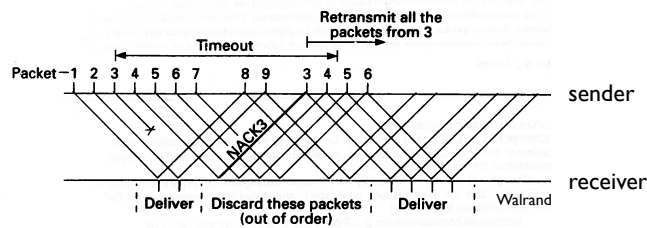


Go-Back-N with Negative ACK (NAK)

Receiver:

- ACKs and delivers in-order packets
- sends NAK for first out of order pkt and discards pkt
- ACKs and discards subsequent out of order packets

Sender: retransmits on receiving NAK or rto



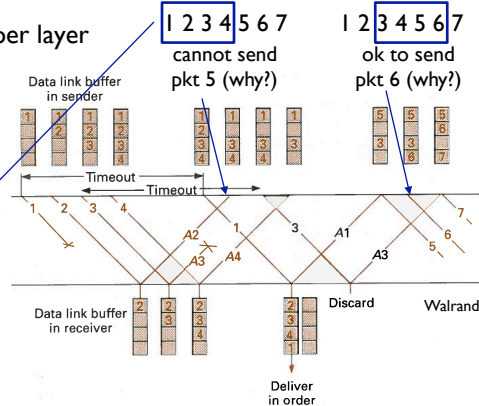
Selective Repeat Protocol (SRP)

Receiver:

- ACKs all correctly received pkts
- buffers out of order pkts (up to w_R), for eventual in-order delivery to upper layer

Sender:

- keeps a retransmit timer for each pkt
- retransmits **only unACKed pkts**
- must keep track of w_R and ensures that $w_R > (\text{largest unACKed} - \text{smallest unACKed})$, in example, $w_R = 4$

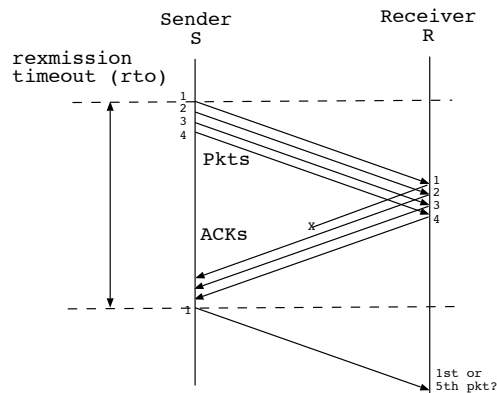


Selective Acknowledgement: Piggy-back NAK with ACK., e.g. [ACK2,NAK1], [ACK4,NAK3]

Simplifying Assumptions

Infinite sequence# space

Suppose you have only a 2-bit sequence space:



Other Issues at Transport Layer

Connectionless network layer means each pkt can:

- take a different path
- experience different congestion

Implications:

- non-deterministic rtt
- out of order pkts must be buffered for Go-Back-N
- complicates computation of w