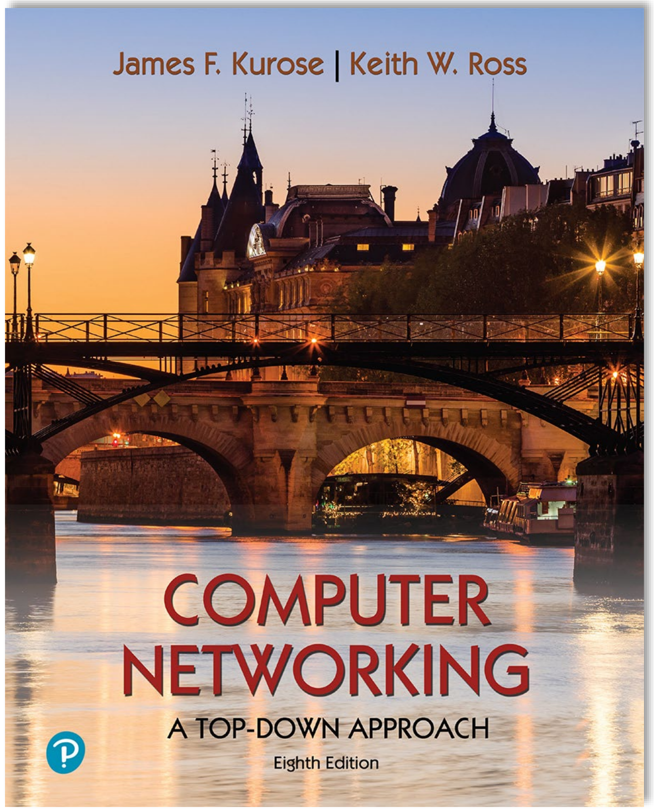
Direct Connection Networks



Computer Networking: A Top-Down Approach
8th edition
Jim Kurose, Keith Ross
Pearson, 2020

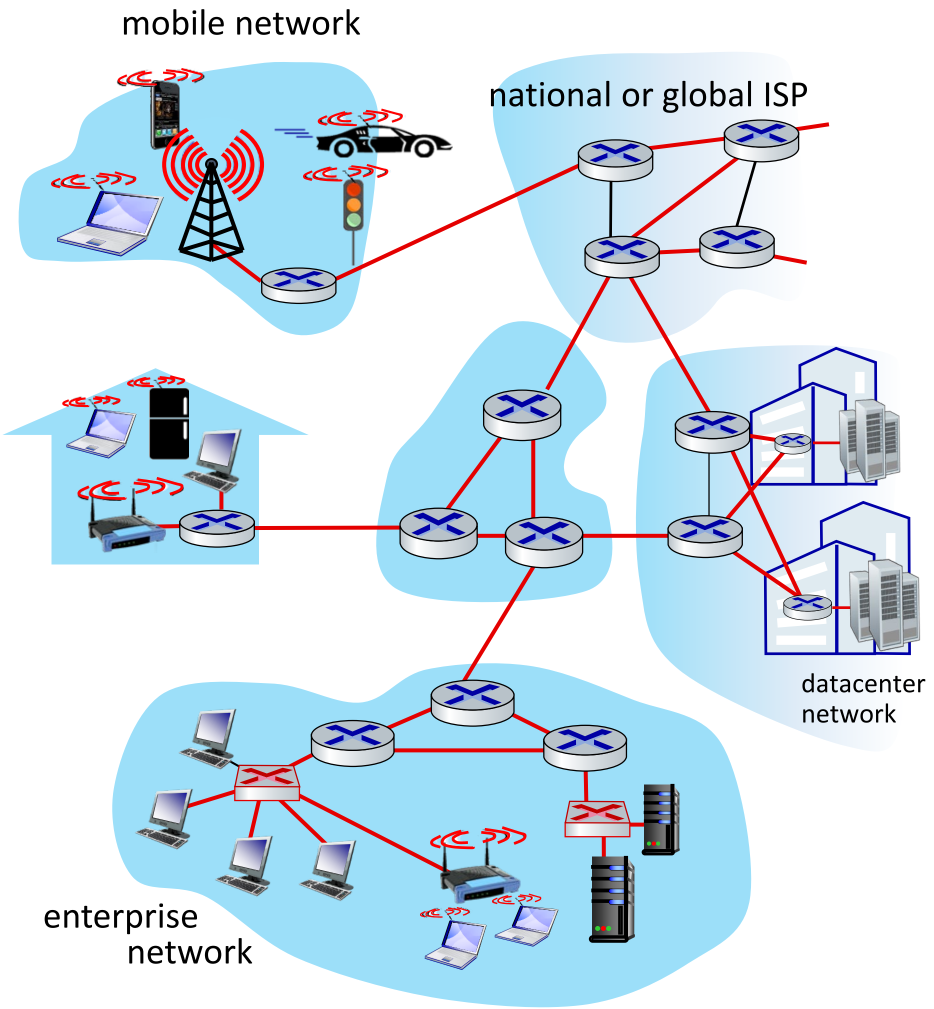
Acknowledgements
These Slides have been adapted from the originals made available by J. Kurose and K. Ross

Context

terminology:

- hosts and routers: nodes
- communication channels that connect adjacent nodes along communication path: links
  - wired
  - wireless

In this part of the course we will look at how data are transferred between adjacent nodes



Goals

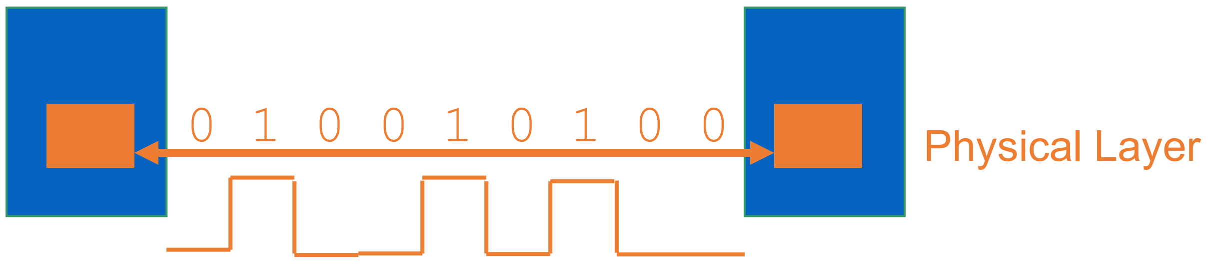
- Introduce Direct Connection Networks
- Understand principles behind link layer services
  - error detection, correction
  - sharing a broadcast channel: multiple access
  - link layer addressing
  - Local Area Networks: Ethernet
- Instantiation, implementation of various link layer technologies

Roadmap

- Introduction
- Reliable communication over unreliable links
  - Framing
  - Error detection, Correction
  - Reliable Data Transfer
- PPP
- Multiple Access protocols
- LANs
  - Addressing
  - Ethernet

Introduction

Physical Link

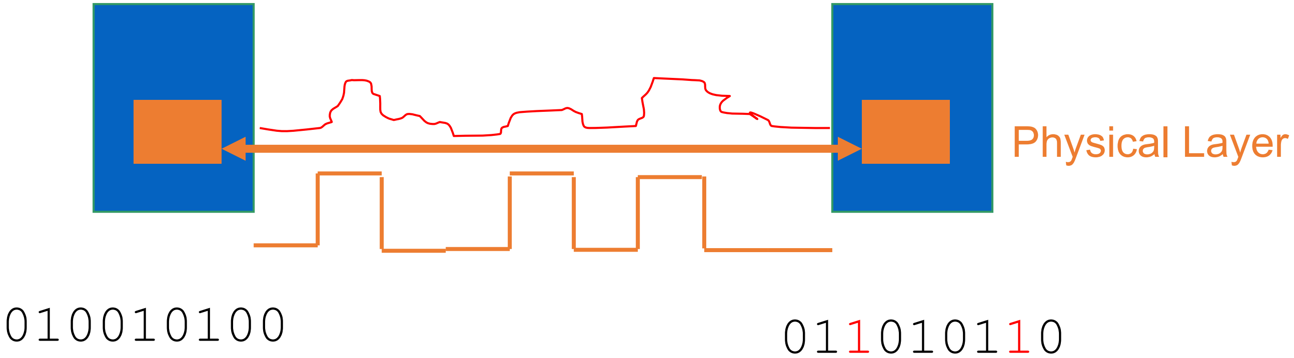


Encoding

Bits are coded through an electric/electro-magnetic/light signal and send over the physical link

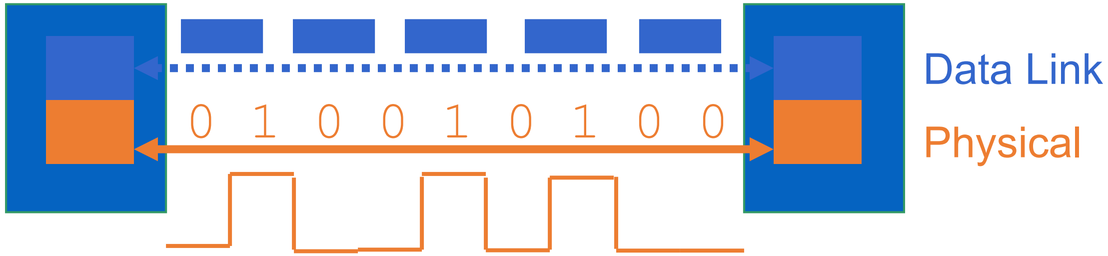
Unreliable Links

- The transmission channel is not ideal
  - Signal attenuation
  - Noise
    - Interferences, fading, …
- The received data sequence may be (very) different from the transmitted one



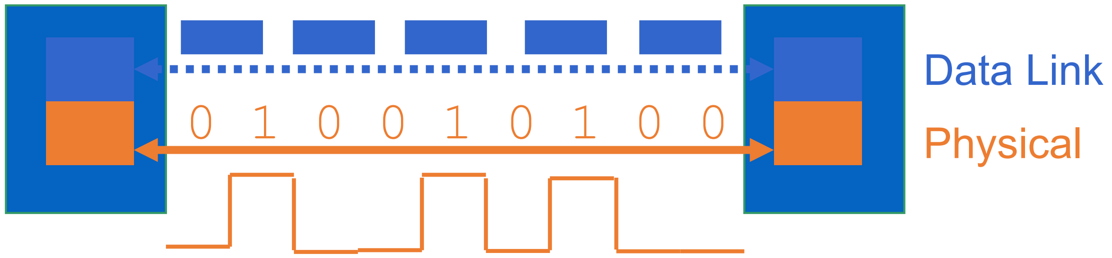
Data Link Layer

- How to achieve reliable communication over unreliable links?



Link layer: services

- Framing
  - encapsulate the datagram into frames
  - adding header and trailer to each frame

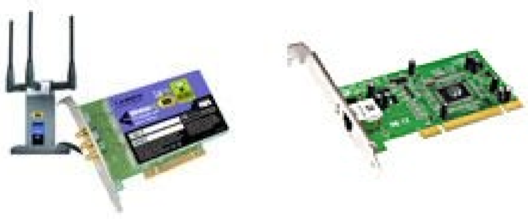


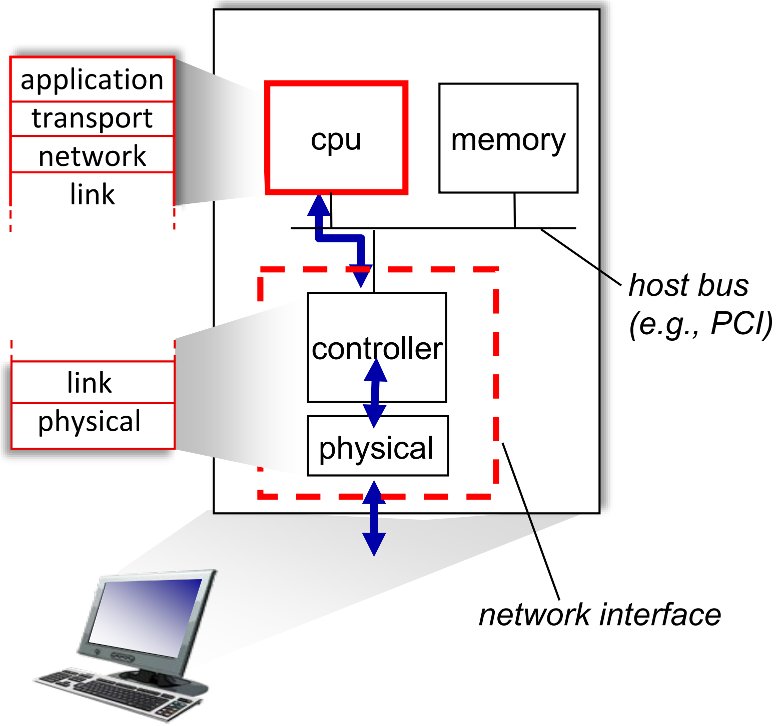
Link layer: services (more)

- Error detection:
  - Errors caused by signal attenuation, noise.
  - Receiver detects errors
- Error correction:
  - Receiver identifies *and corrects* bit error(s)
- Retransmission
  - Receiver signals corrupted frames
  - Transmitter resends corrupted frames
- Flow control:
  - Pacing between adjacent sending and receiving nodes
- Half-duplex and Full-duplex:
  - with half duplex, nodes at both ends of link can transmit, but not at same time

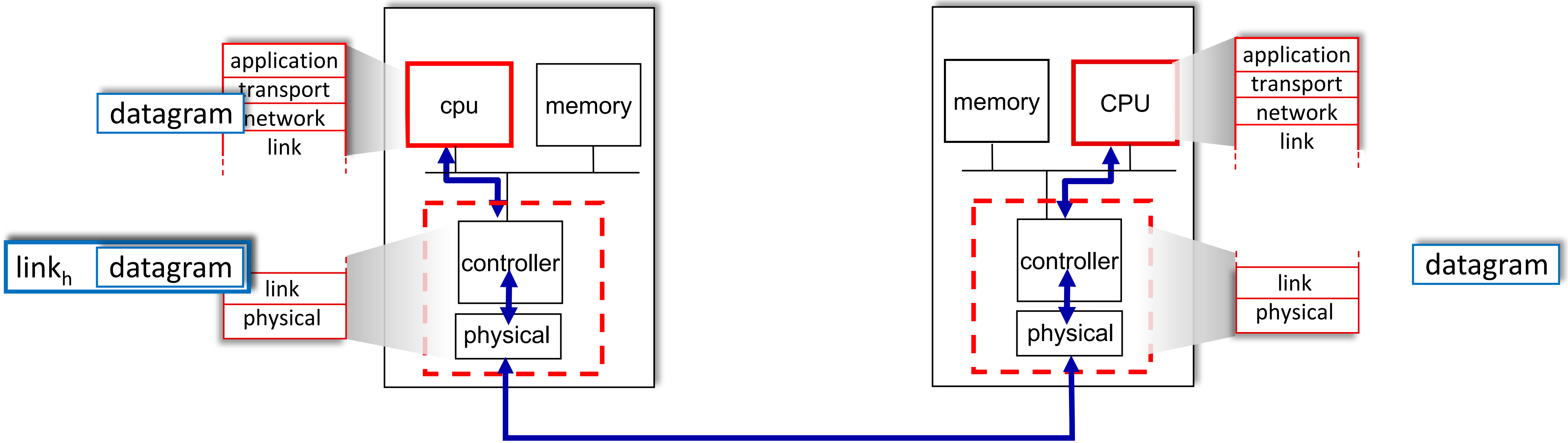
Where is the link layer implemented?

- in each-and-every host
- link layer implemented in *network interface card* (NIC) or on a chip
  - Ethernet, WiFi card or chip
  - implements link, physical layer
- attaches into host's system buses
- combination of hardware, software, firmware





Interfaces communicating



sending side:

- encapsulates datagram in frame
- adds error checking bits, reliable data transfer, flow control, etc.

receiving side:

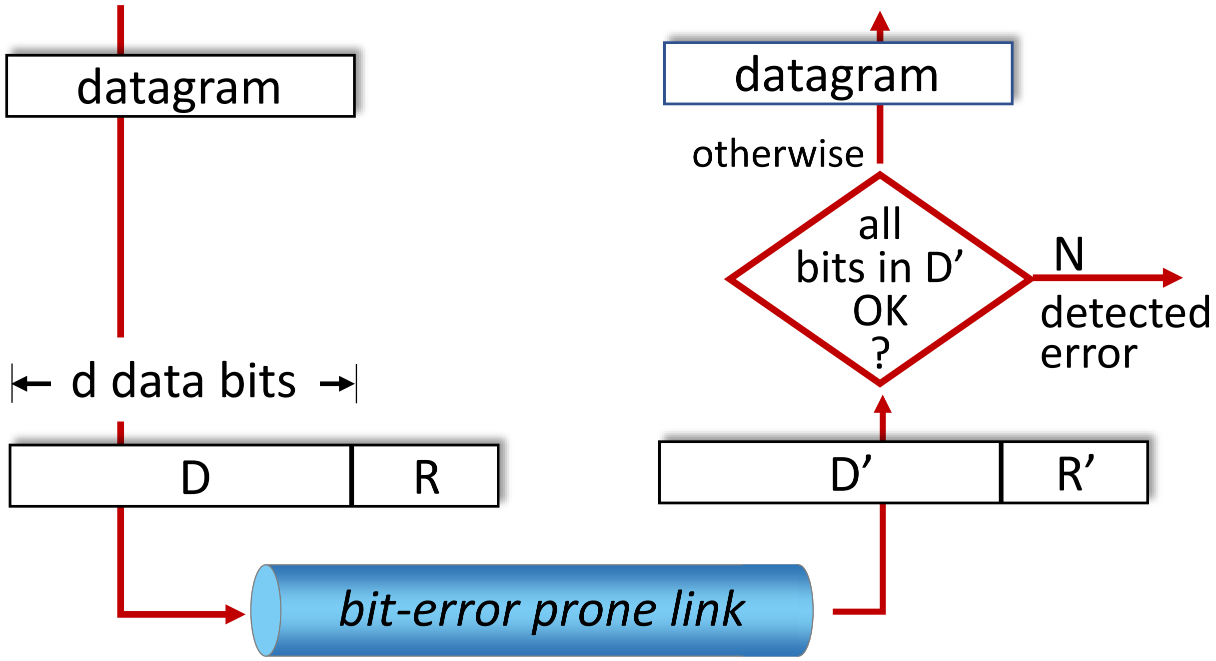
- looks for errors, reliable data transfer, flow control, etc.
- extracts datagram, passes to upper layer at receiving side

Roadmap

- Introduction
- Reliable communication over unreliable links
  - Framing
  - Error detection, correction
  - Reliable Data Transfer
- PPP
- Multiple Access protocols
- LANs
  - Addressing, ARP
  - Ethernet

Error detection

R: error detection bits (redundancy)
D: data protected by error checking, may include header fields

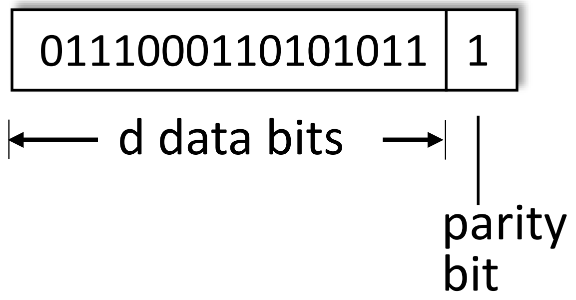


Error detection not 100% reliable!
- protocol may miss some errors, but rarely
- larger R field yields better detection and correction

Parity checking

Single bit parity:

- detect single bit errors



Even parity: set parity bit so there is an even number of 1’s

Odd parity: set parity bit so there is an odd number of 1’s

Internet checksum (review)

Goal: detect errors (*i.e.,* flipped bits) in transmitted segment

sender:

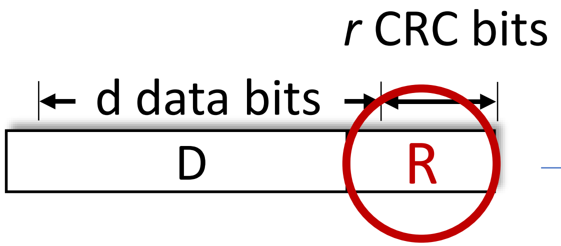
- treat contents of a packet as sequence of 16-bit integers
- checksum: addition (one's complement sum) of packet content
- checksum value put into a specific field of the packet
  - checksum field

receiver:

- compute checksum of received segment
- check if computed checksum equals checksum field value:
  - not equal - error detected
  - equal - no error detected. *But maybe errors nonetheless?* More later ….

Cyclic Redundancy Check (CRC)

- more powerful error-detection coding
- D: data bits (given, think of these as a binary number)
- G: bit pattern (generator), of *r+1* bits (given)



bit pattern

$\langle D,R\rangle = D * 2^r \text{ XOR } R$

formula for bit pattern

goal: choose *r* CRC bits, R, such that <D,R> exactly divisible by G (mod 2)

- receiver knows G, divides <D,R> by G. If non-zero remainder: error detected!
- can detect all burst errors less than r+1 bits
- widely used in practice (Ethernet, 802.11 WiFi)

Cyclic Redundancy Check (CRC): example

We want:

$D \cdot 2^r \;\; XOR \;\; R = nG$

or equivalently:

$D \cdot 2^r = nG \;\; XOR \;\; R$

or equivalently:

if we divide $D \cdot 2^r$ by G, want remainder R to satisfy:

$$R = remainder\left[\frac{D \cdot 2^r}{G}\right]$$

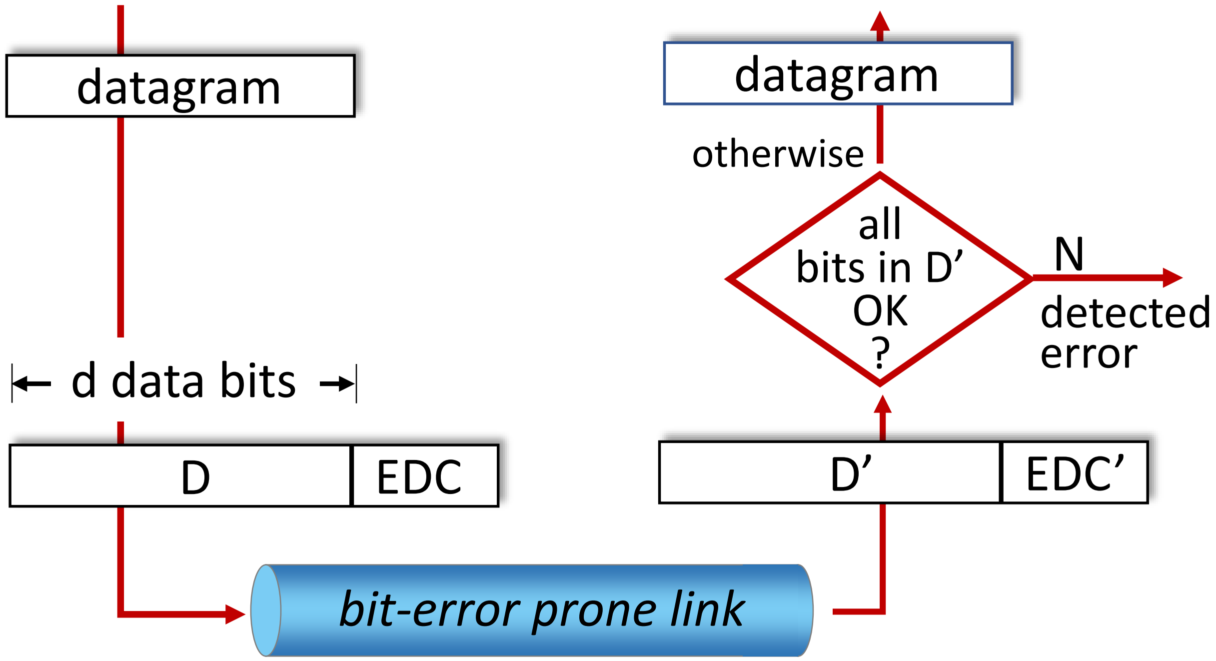
```
   G
                    1 0 1 0 1 1
1 0 0 1 ) 1 0 1 1 1 0 0 0 0      D*2^r
          1 0 0 1
              1 0 1
              0 0 0
              1 0 1 0
              1 0 0 1
                  1 1 0
                  0 0 0
                  1 1 0 0
                  1 0 0 1
                    1 0 1 0
                    1 0 0 1
                      0 1 1
                        R
```

* Check out the online interactive exercises for more examples: http://gaia.cs.umass.edu/kurose_ross/interactive/

Error detection & correction

EDC: error detection and correction bits (e.g., redundancy)

D: data protected by error checking, may include header fields



Error detection not 100% reliable!

- protocol may miss some errors, but rarely
- larger EDC field yields better detection and correction

2-Dimension Parity checking

single bit parity:

- detect single bit errors

0111000110101011	1

← d data bits → | parity bit

two-dimensional bit parity:

- detect *and correct* single bit errors

$$
\begin{array}{cccc|c}
 & & & & \text{row parity} \rightarrow \\
 & d_{1,1} & \cdots & d_{1,j} & d_{1,j+1} \\
 & d_{2,1} & \cdots & d_{2,j} & d_{2,j+1} \\
 & \cdots & \cdots & \cdots & \cdots \\
\text{column parity} \downarrow & d_{i,1} & \cdots & d_{i,j} & d_{i,j+1} \\
\hline
 & d_{i+1,1} & \cdots & d_{i+1,j} & d_{i+1,j+1}
\end{array}
$$

no errors:

$$
\begin{array}{ccccc|c}
1 & 0 & 1 & 0 & 1 & 1 \\
1 & 1 & 1 & 1 & 0 & 0 \\
0 & 1 & 1 & 1 & 0 & 1 \\
\hline
1 & 0 & 1 & 0 & 1 & 0
\end{array}
$$

detected and correctable single-bit error:

$$
\begin{array}{ccccc|c}
1 & 0 & 1 & 0 & 1 & 1 \\
1 & 0 & 1 & 1 & 0 & 0 \\
0 & 1 & 1 & 1 & 0 & 1 \\
\hline
1 & 0 & 1 & 0 & 1 & 0
\end{array}
$$

parity error (row)

parity error (column)

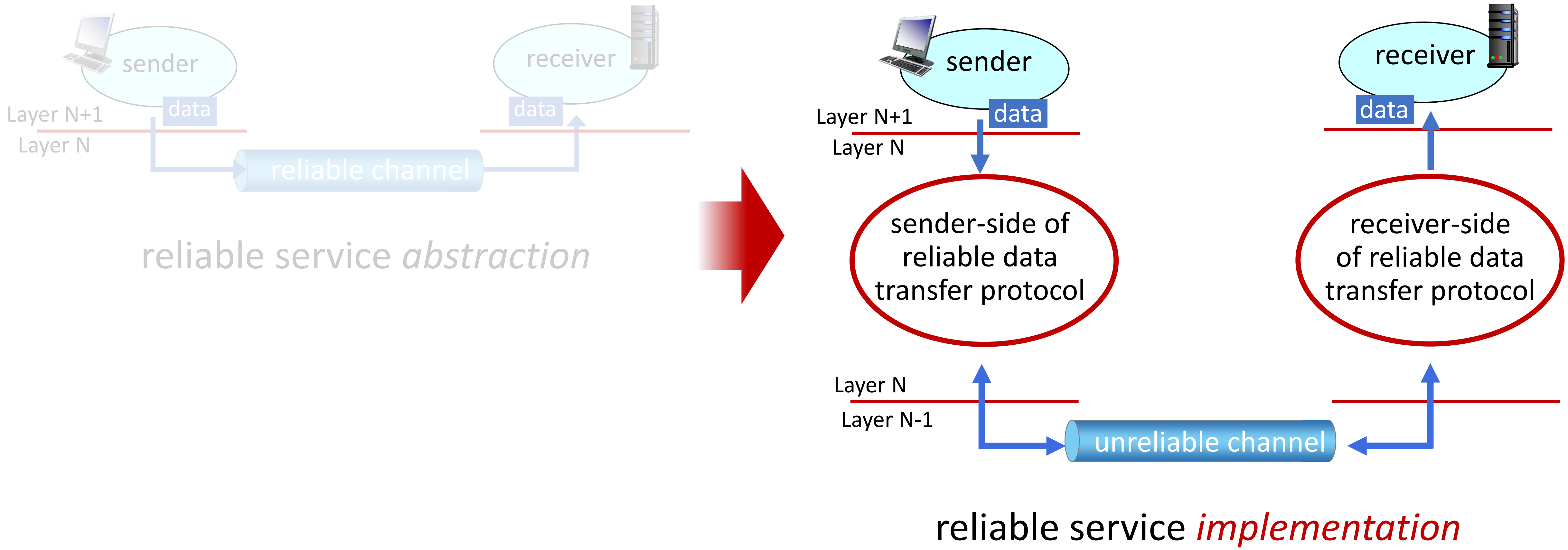
Roadmap

- Introduction
- Reliable communication over unreliable links
  - Framing
  - Error detection, Correction
  - Reliable Data Transfer
- PPP
- Multiple Access protocols
- LANs
  - Addressing, ARP
  - Ethernet

Principles of reliable data transfer

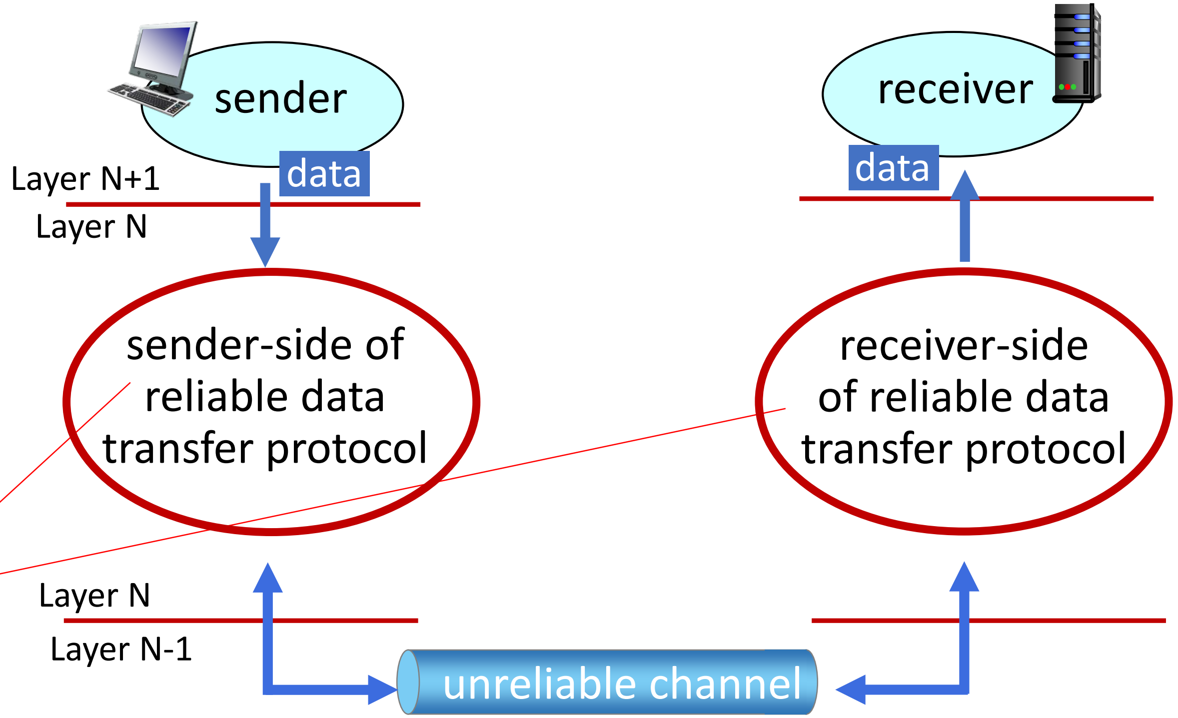
reliable service *abstraction*

Principles of reliable data transfer



Principles of reliable data transfer

Complexity of reliable data transfer protocol will depend (strongly) on characteristics of unreliable channel (lose, corrupt, reorder data?)

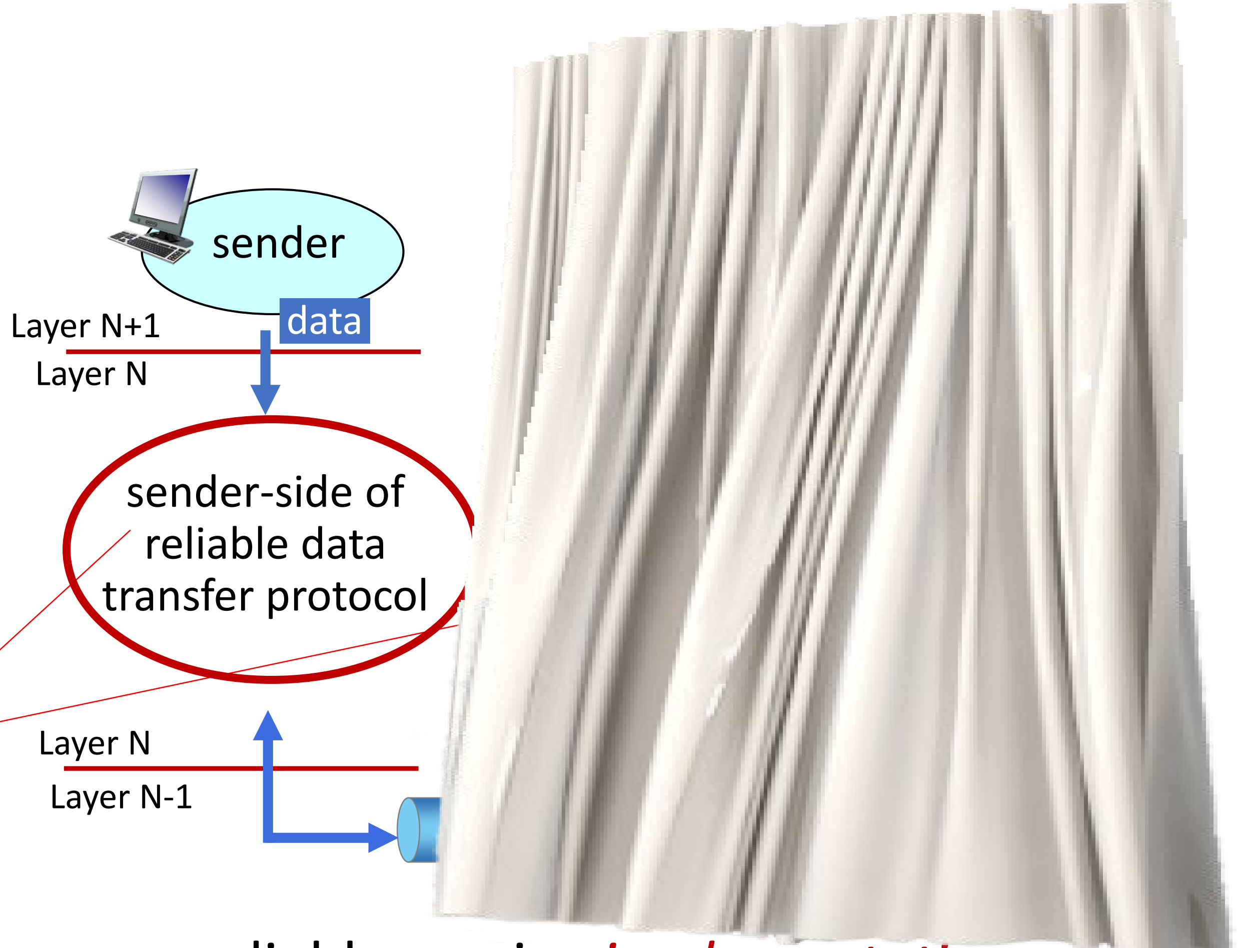


reliable service *implementation*

Principles of reliable data transfer

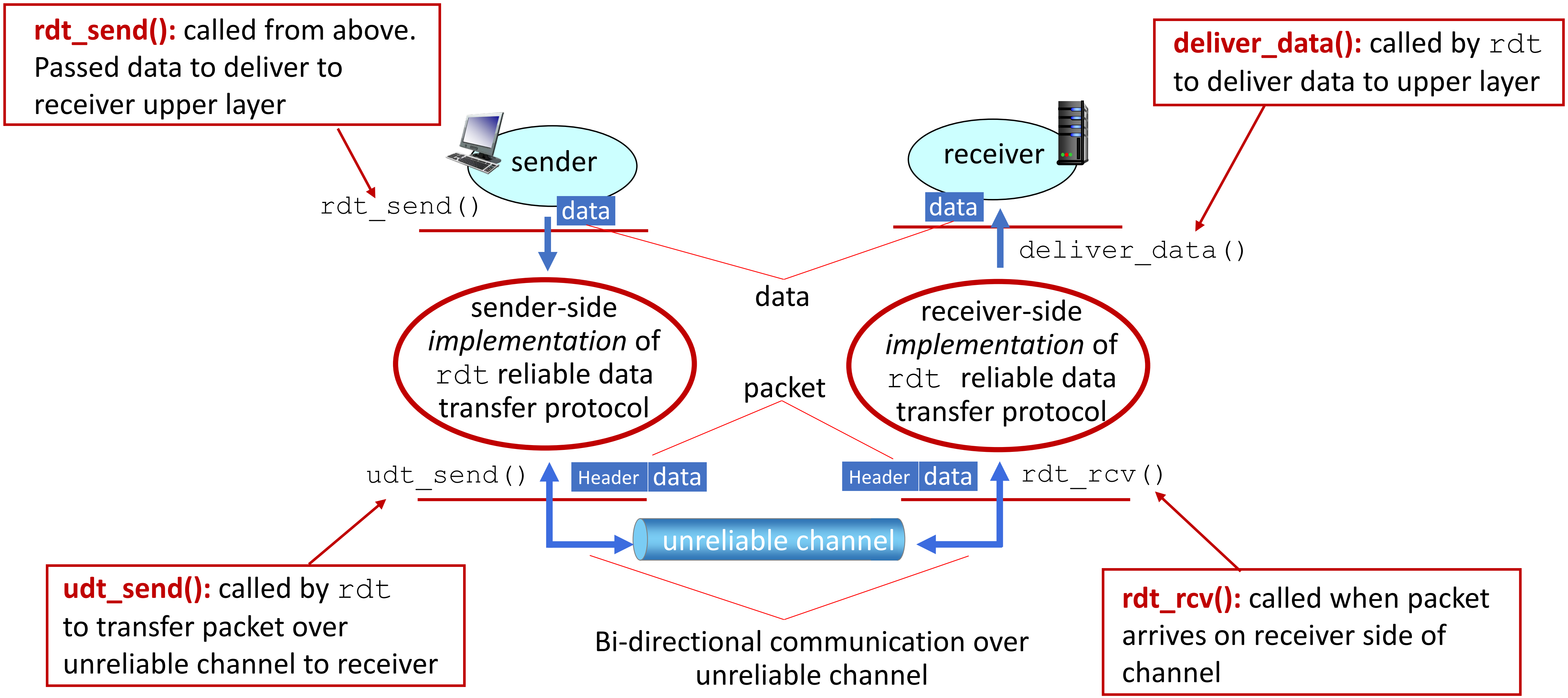
Sender, receiver do *not* know the “state” of each other, e.g., was a message received?

- unless communicated via a message



reliable service *implementation*

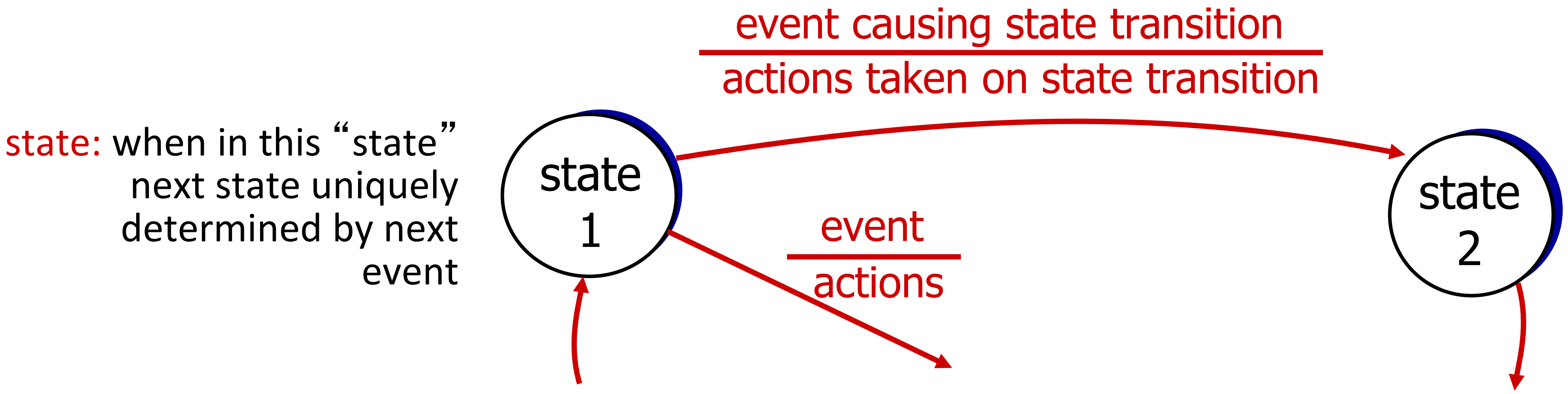
Reliable data transfer protocol (rdt): interfaces



Reliable data transfer: getting started

We will:

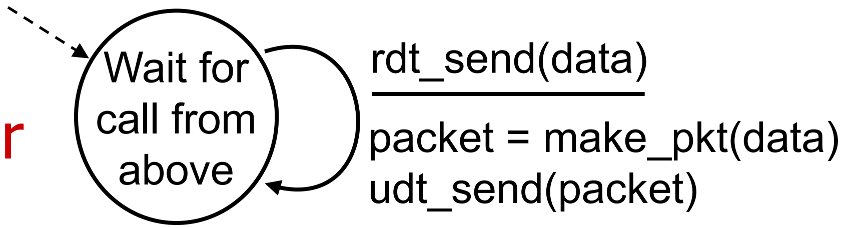
- incrementally develop sender, receiver sides of reliable data transfer protocol (`rdt`)
- consider only unidirectional data transfer
  - but control info will flow in both directions!
- use finite state machines (FSM) to specify sender, receiver



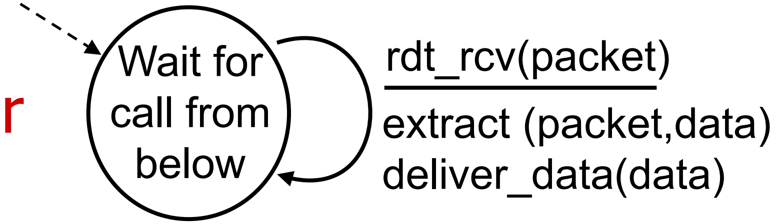
rdt1.0: reliable transfer over a reliable channel

- underlying channel perfectly reliable
  - no bit errors
  - no loss of packets
- *separate* FSMs for sender, receiver:
  - sender sends data into underlying channel
  - receiver reads data from underlying channel

sender



receiver



rdt2.0: channel with bit errors

- underlying channel may flip bits in packet
  - redundancy (e.g., CRC or checksum) to detect bit errors
- *the* question: how to recover from errors?

How do humans recover from "errors" during conversation?

rdt2.0: channel with bit errors

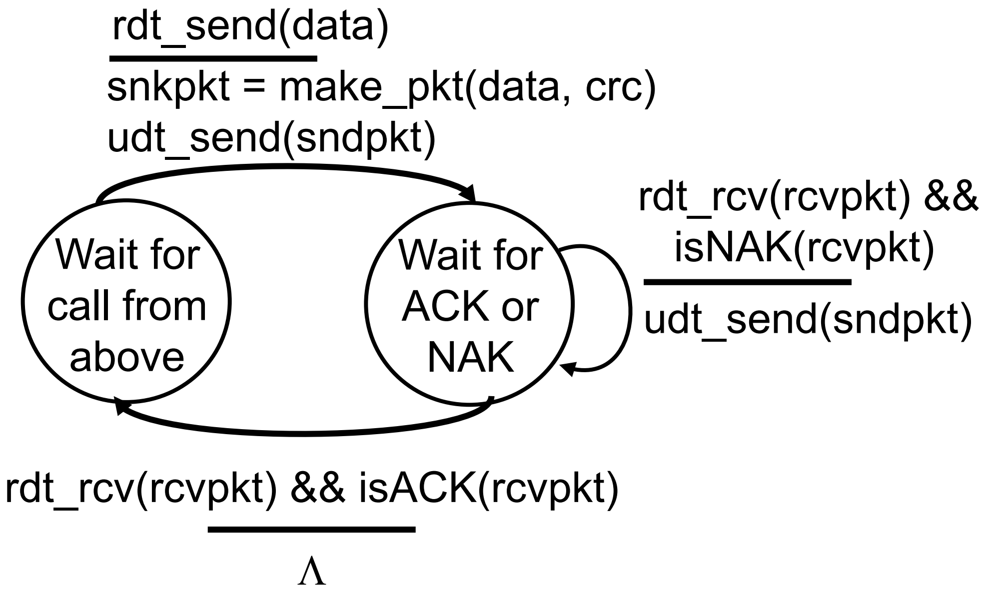
- underlying channel may flip bits in packet
  - checksum to detect bit errors
- *the* question: how to recover from errors?
  - *acknowledgements (ACKs):* receiver explicitly tells sender that pkt received OK
  - *negative acknowledgements (NAKs):* receiver explicitly tells sender that pkt had errors
  - sender *retransmits* pkt on receipt of NAK

stop and wait

sender sends one packet, then waits for receiver response

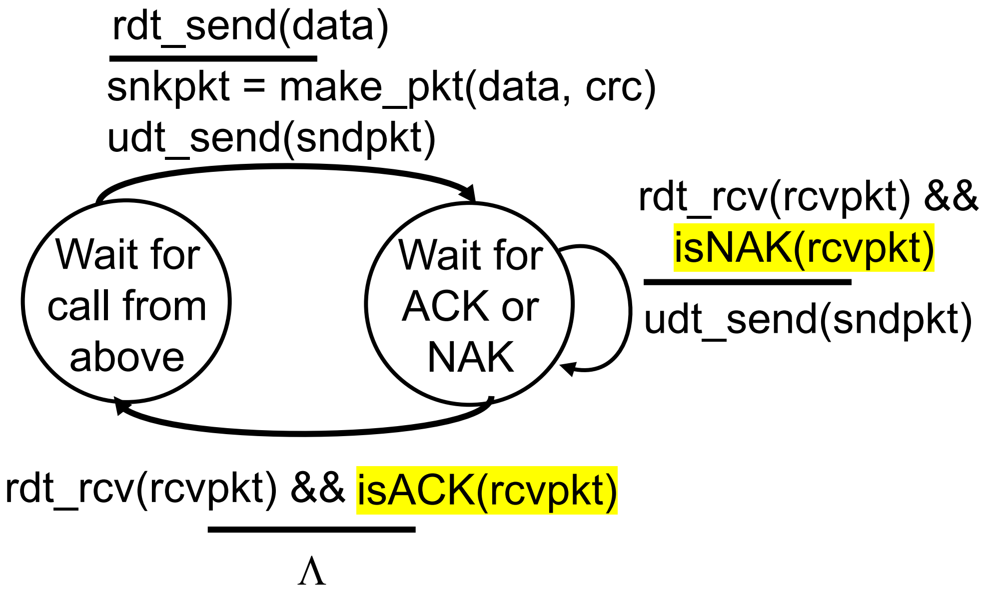
rdt2.0: FSM specifications

sender



rdt2.0: FSM specification

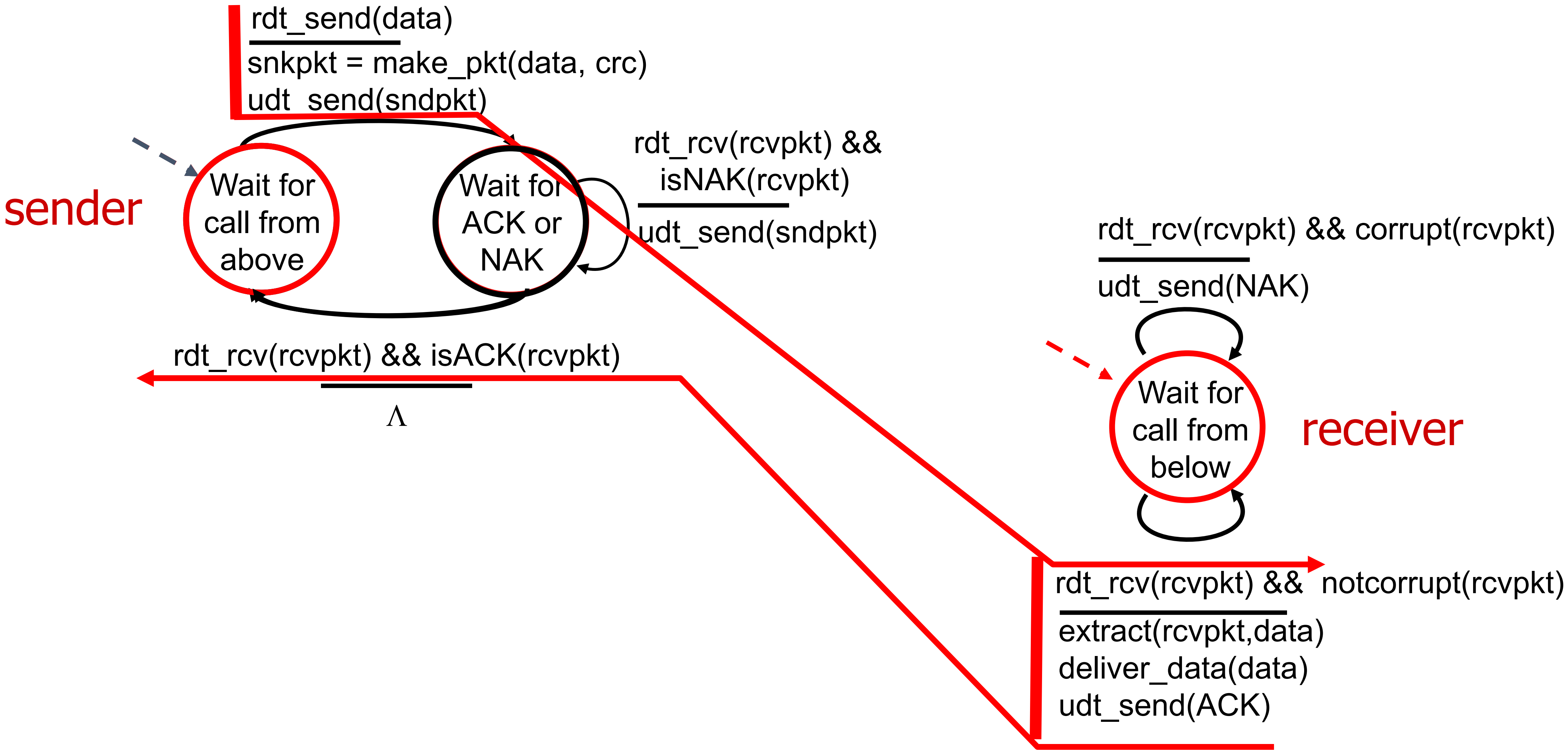
sender



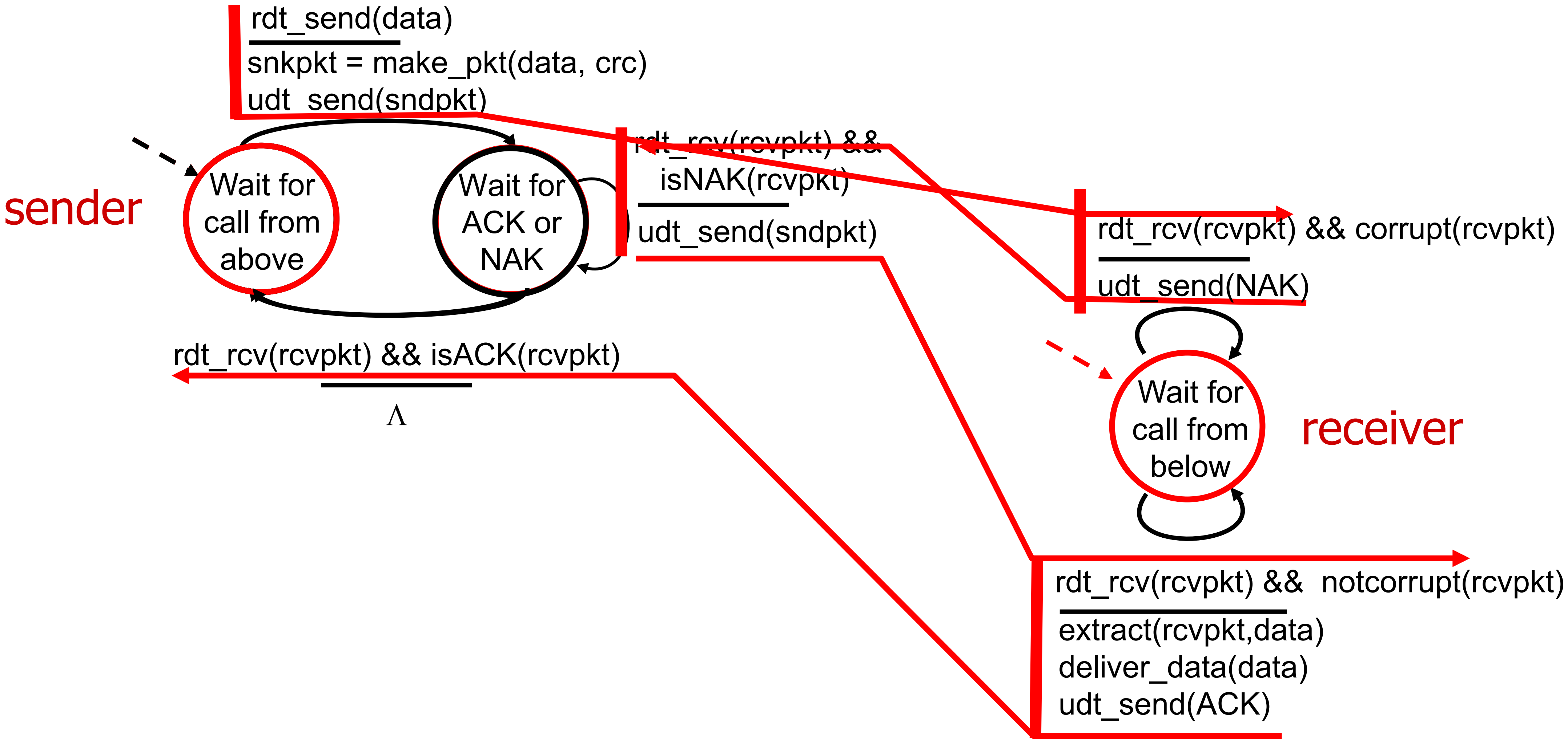
Note: “state” of receiver (did the receiver get my message correctly?) isn’t known to sender unless somehow communicated from receiver to sender

- that’s why we need a protocol!

rdt2.0: operation with no errors



rdt2.0: corrupted packet scenario



rdt2.0 has a fatal flaw!

what happens if ACK/NAK corrupted?

- sender doesn't know what happened at receiver!
- can't just retransmit: possible duplicate

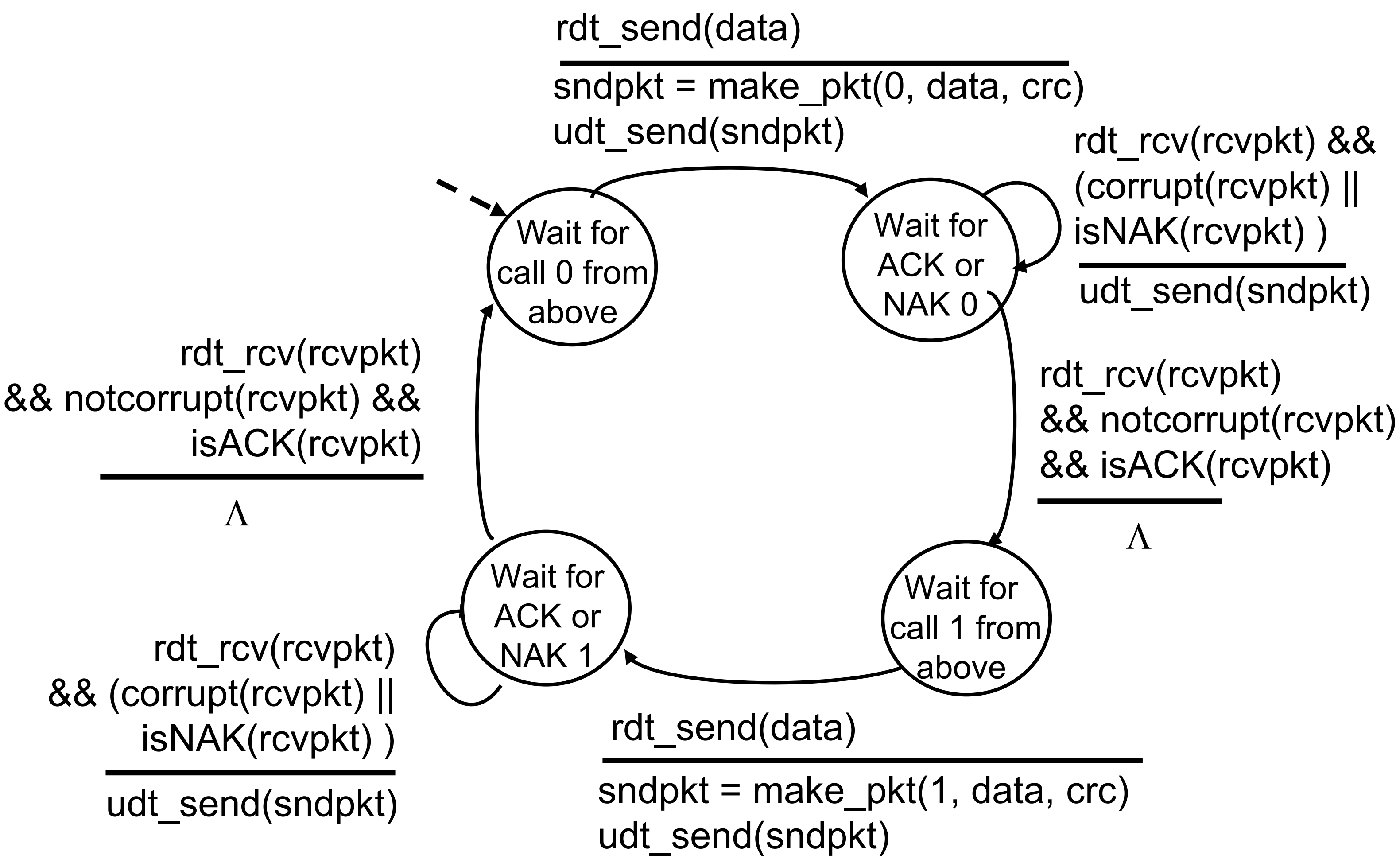
handling duplicates:

- sender retransmits current pkt if ACK/NAK corrupted
- sender adds *sequence number* to each pkt
- receiver discards (doesn't deliver up) duplicate pkt

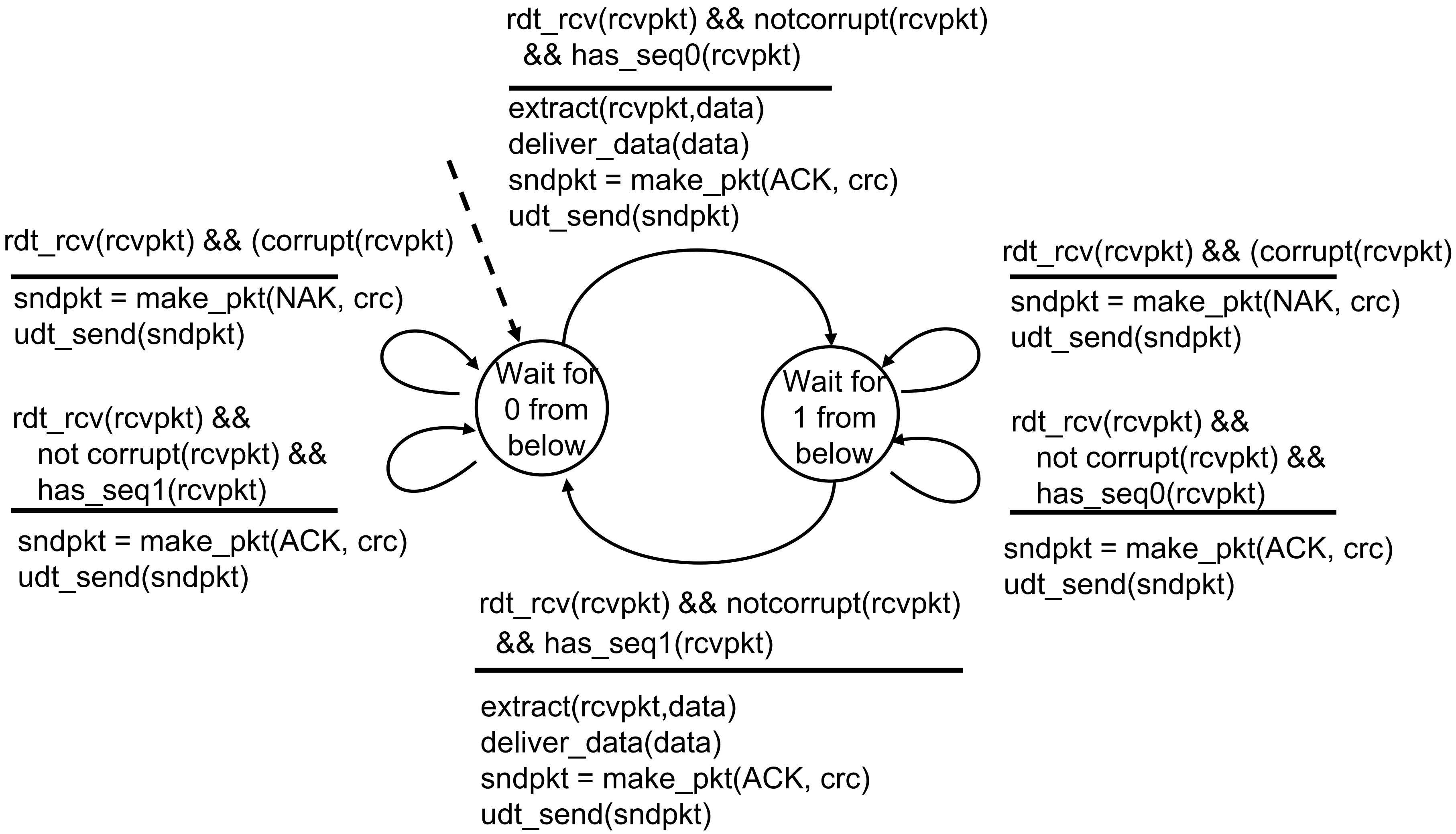
stop and wait

sender sends one packet, then waits for receiver response

rdt2.1: sender, handling garbled ACK/NAKs



rdt2.1: receiver, handling garbled ACK/NAKs



rdt2.1: discussion

sender:

- seq # added to pkt
- two seq. #s (0,1) will suffice. Why?
- must check if received ACK/NAK corrupted
- twice as many states
  - state must "remember" whether "expected" pkt should have seq # of 0 or 1

receiver:

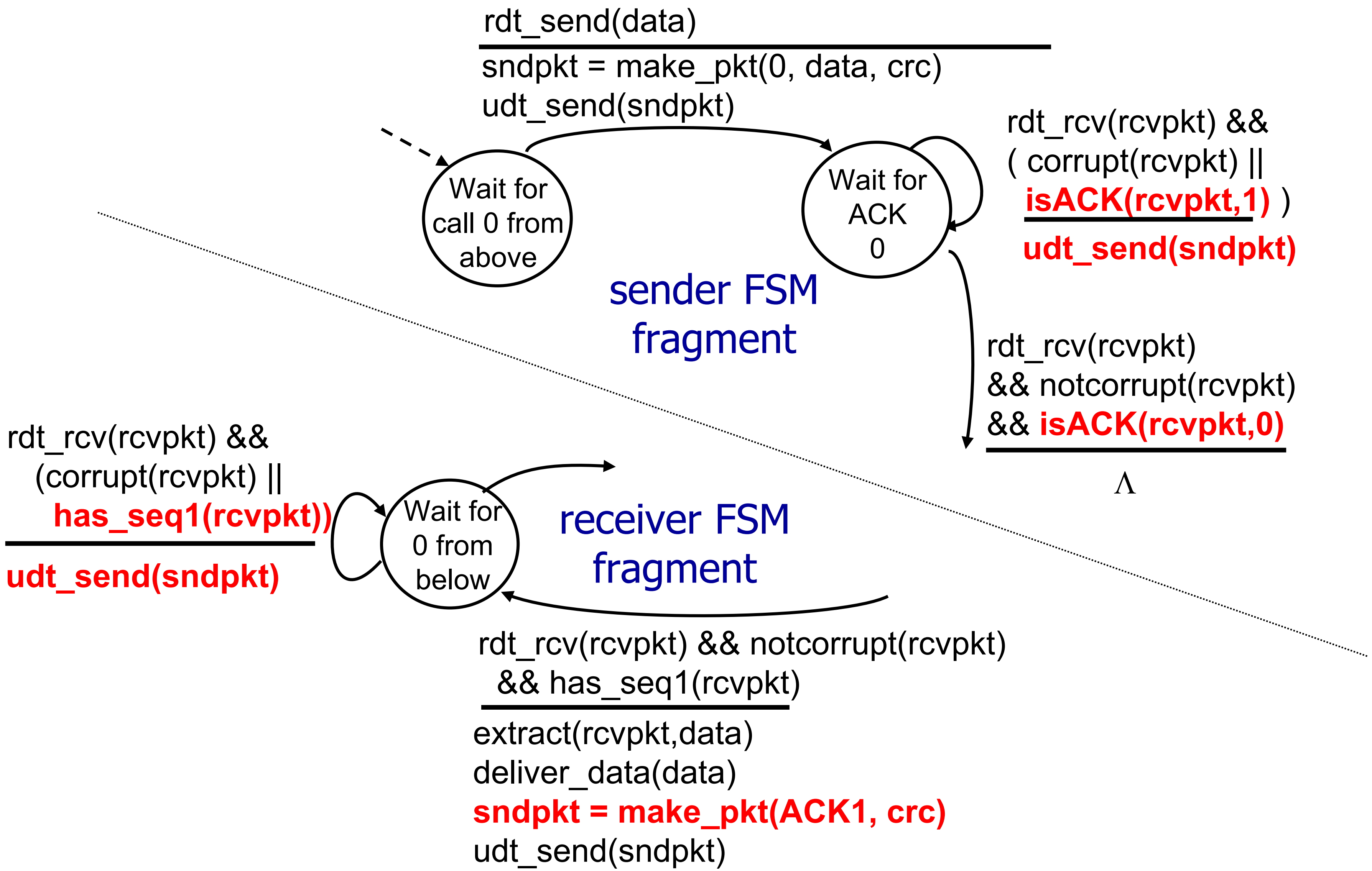
- must check if received packet is duplicate
  - state indicates whether 0 or 1 is expected pkt seq #
- note: receiver can *not* know if its last ACK/NAK received OK at sender

rdt2.2: a NAK-free protocol

- same functionality as rdt2.1, using ACKs only
- instead of NAK, receiver sends ACK for last pkt received OK
  - receiver must *explicitly* include seq # of pkt being ACKed
- duplicate ACK at sender results in same action as NAK: *retransmit current pkt*

Many protocols, including TCP, use this approach to be NAK-free

rdt2.2: sender, receiver fragments



rdt3.0: channels with errors *and* loss

New channel assumption: underlying channel can also *lose* packets (data, ACKs)

- checksum, sequence #s, ACKs, retransmissions will be of help … but not quite enough

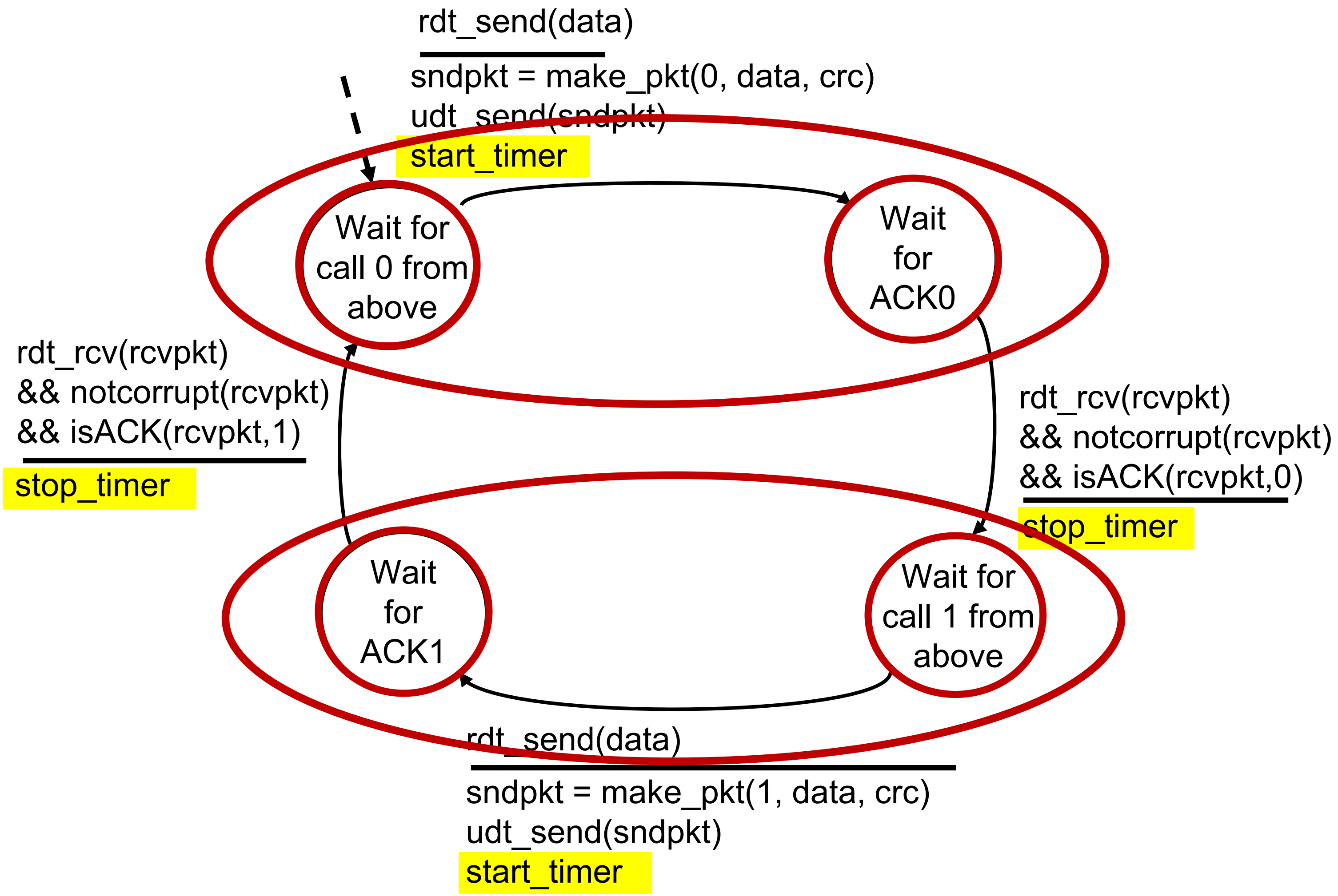
Q: How do *humans* handle lost sender-to-receiver words in conversation?

rdt3.0: channels with errors *and* loss

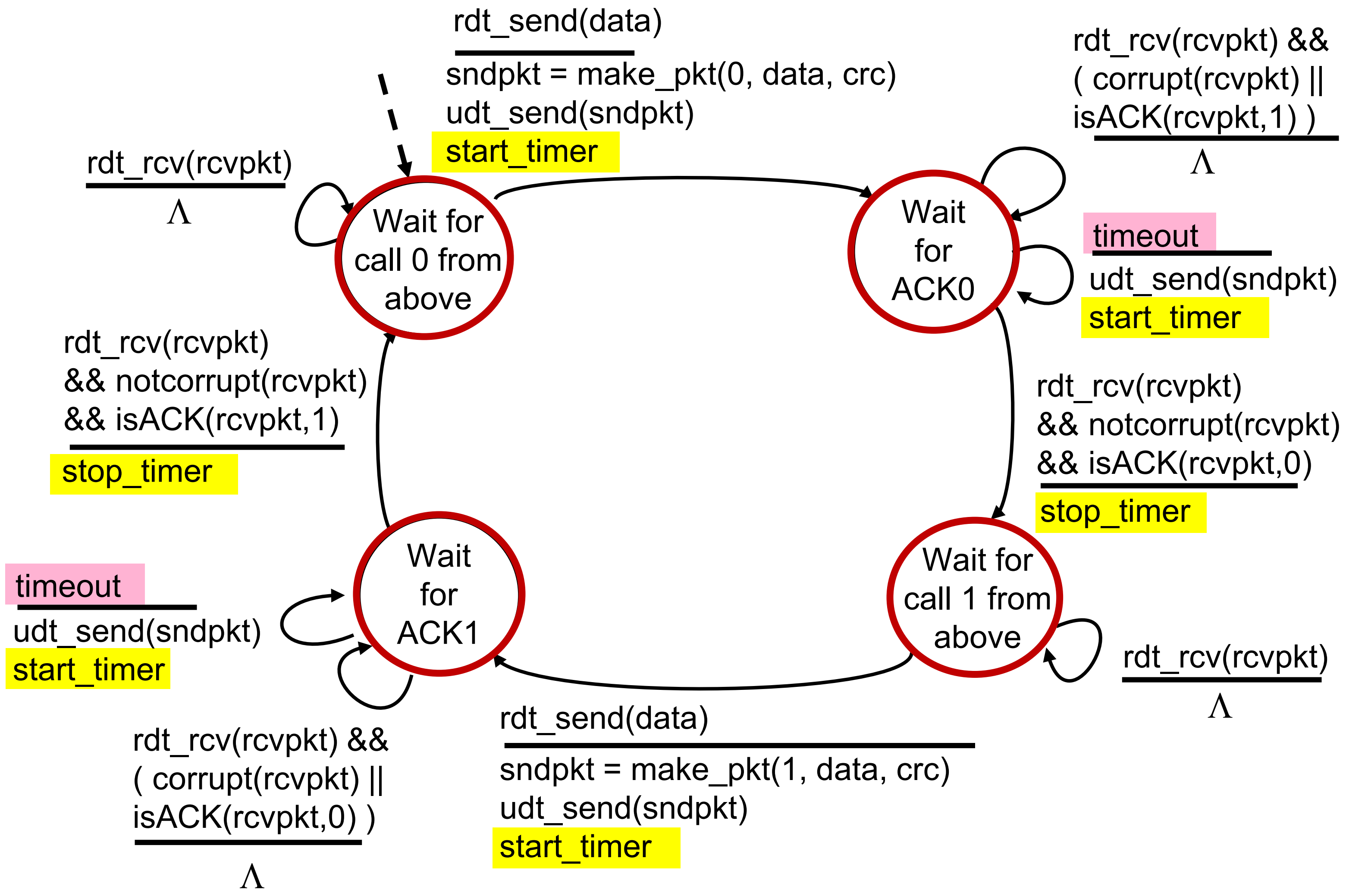
Approach: sender waits "reasonable" amount of time for ACK

- retransmits if no ACK received in this time
- if pkt (or ACK) just delayed (not lost):
  - retransmission will be duplicate, but seq #s already handles this!
  - receiver must specify seq # of packet being ACKed
- use countdown timer to interrupt after "reasonable" amount of time

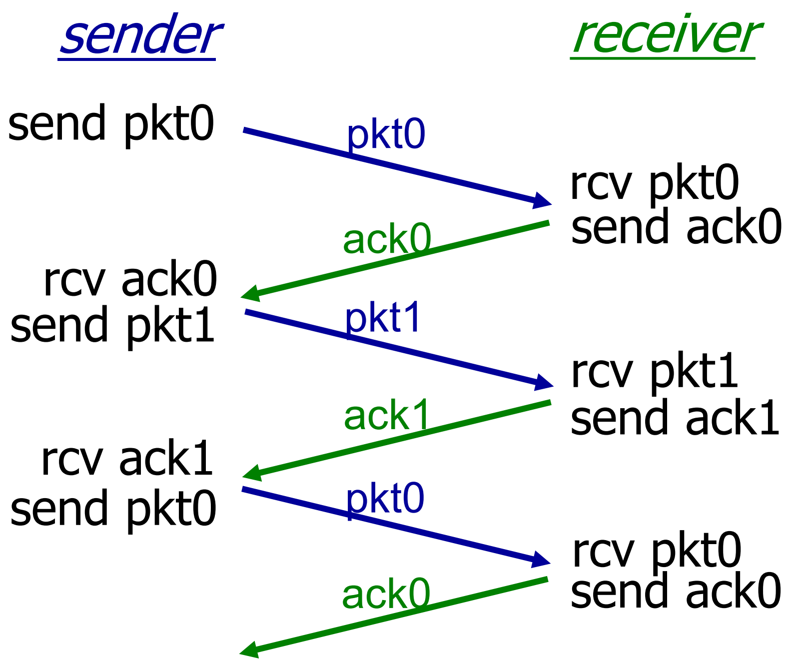
rdt3.0 sender



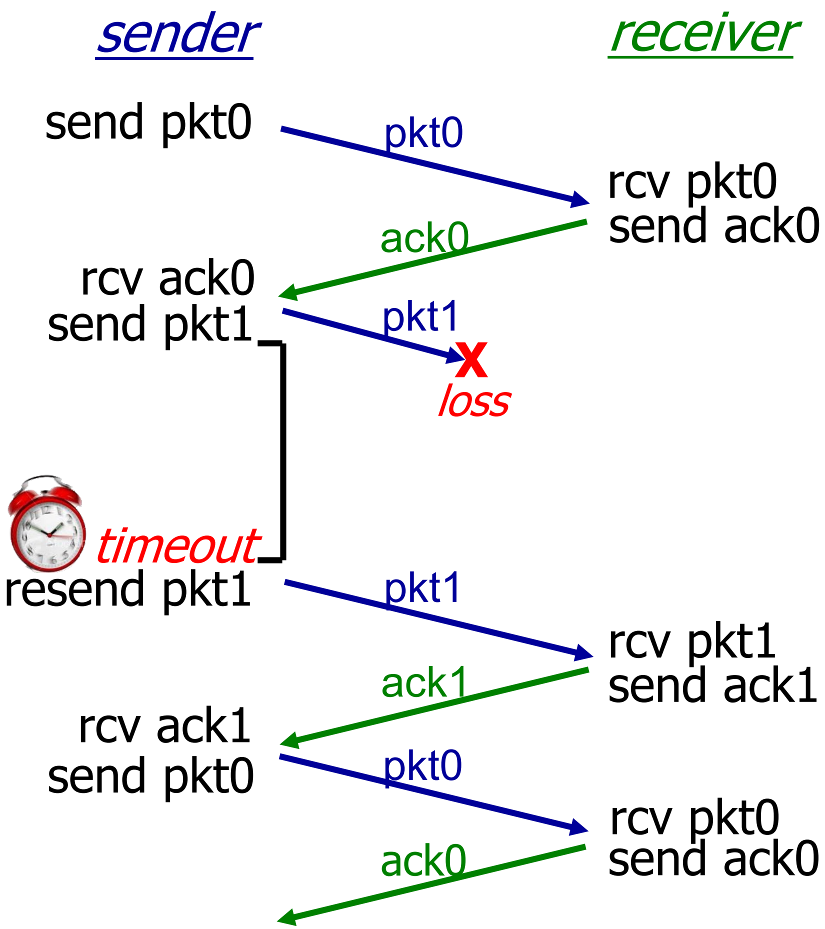
rdt3.0 sender



rdt3.0 in action

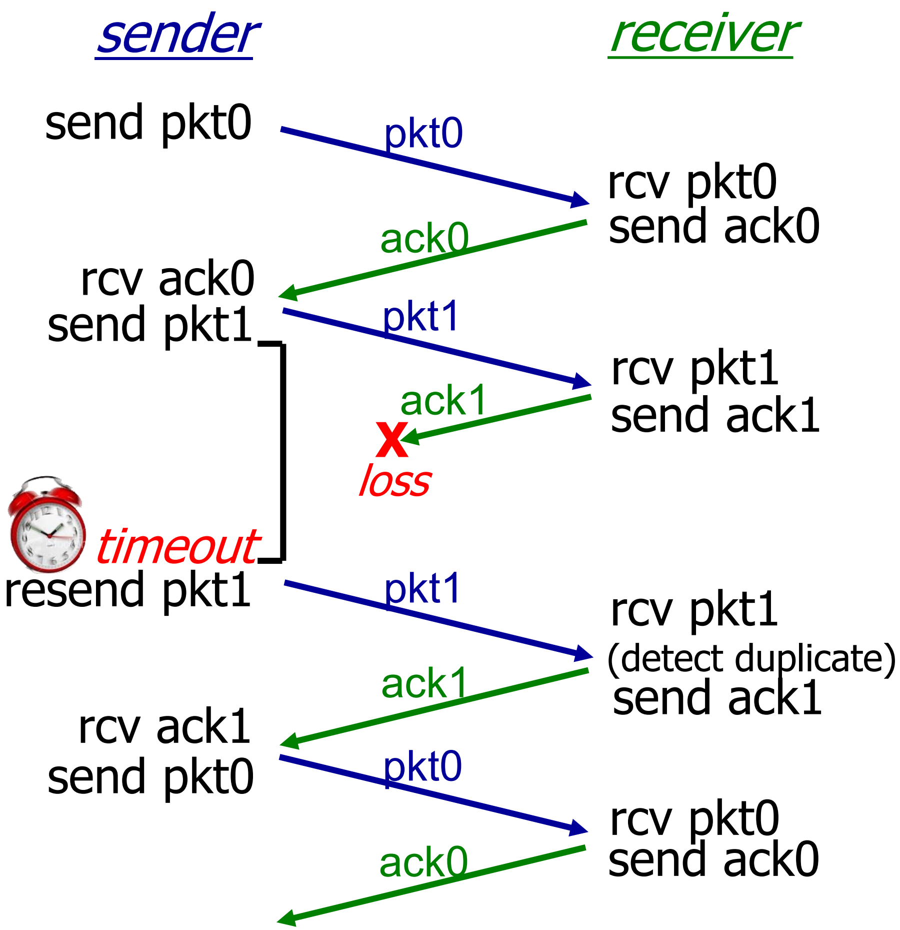


(a) no loss

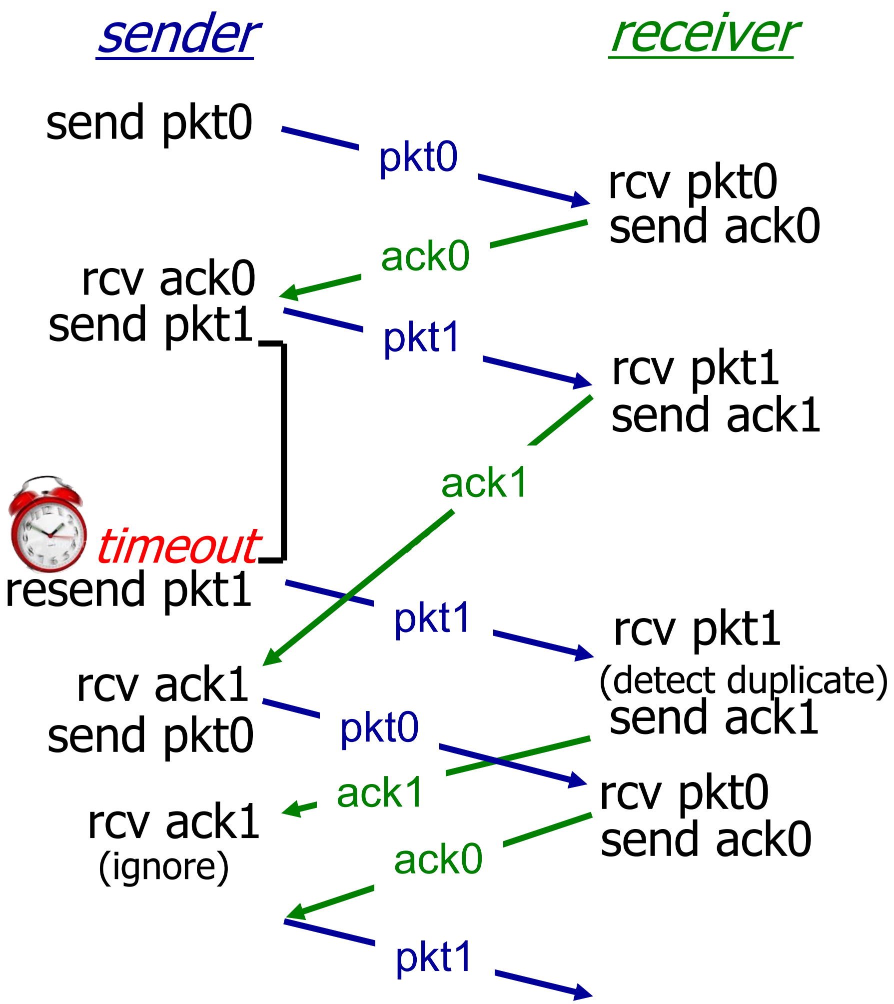


(b) packet loss

rdt3.0 in action



(c) ACK loss



(d) premature timeout/ delayed ACK

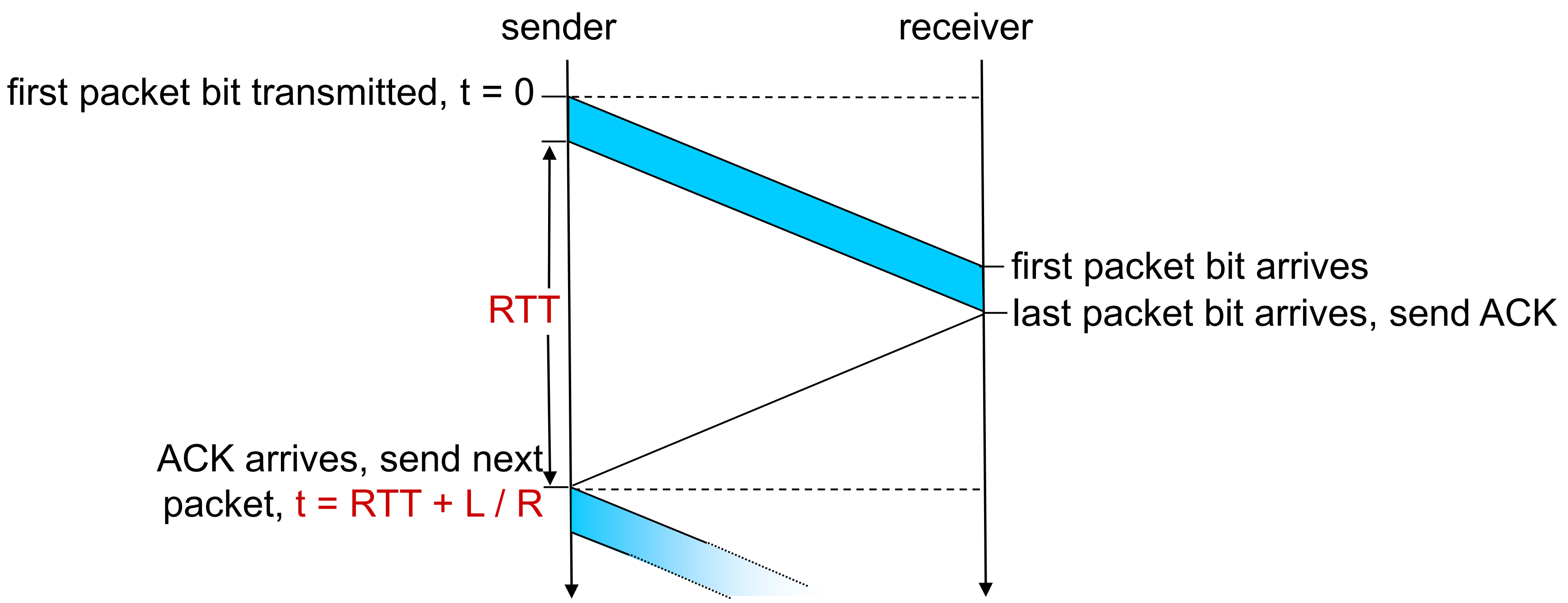
Performance of rdt3.0 (stop-and-wait)

- U_{sender}: *utilization* – fraction of time sender busy sending

- example: 1 Gbps link, 15 ms prop. delay, 8000 bit packet
  - time to transmit packet into channel:

$$D_{trans} = \frac{L}{R} = \frac{8000 \text{ bits}}{10^9 \text{ bits/sec}} = 8 \text{ microsecs}$$

rdt3.0: stop-and-wait operation

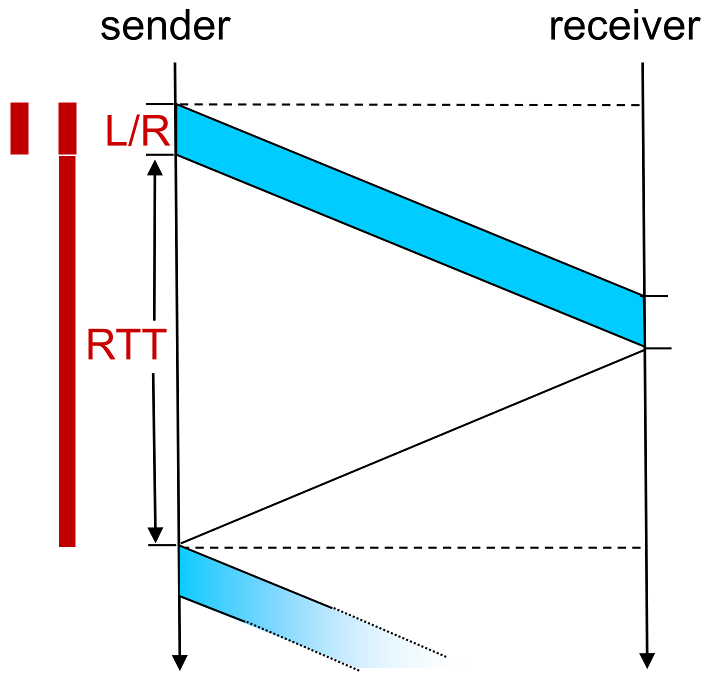


rdt3.0: stop-and-wait operation

$$U_{sender} = \frac{L / R}{RTT + L / R}$$

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

$$= 0.00027$$

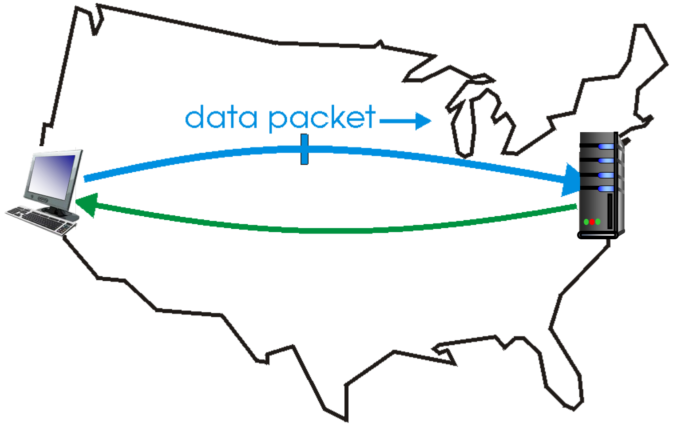


- rdt 3.0 protocol performance stinks!
- Protocol limits performance of underlying infrastructure (channel)

rdt3.0: pipelined protocols operation

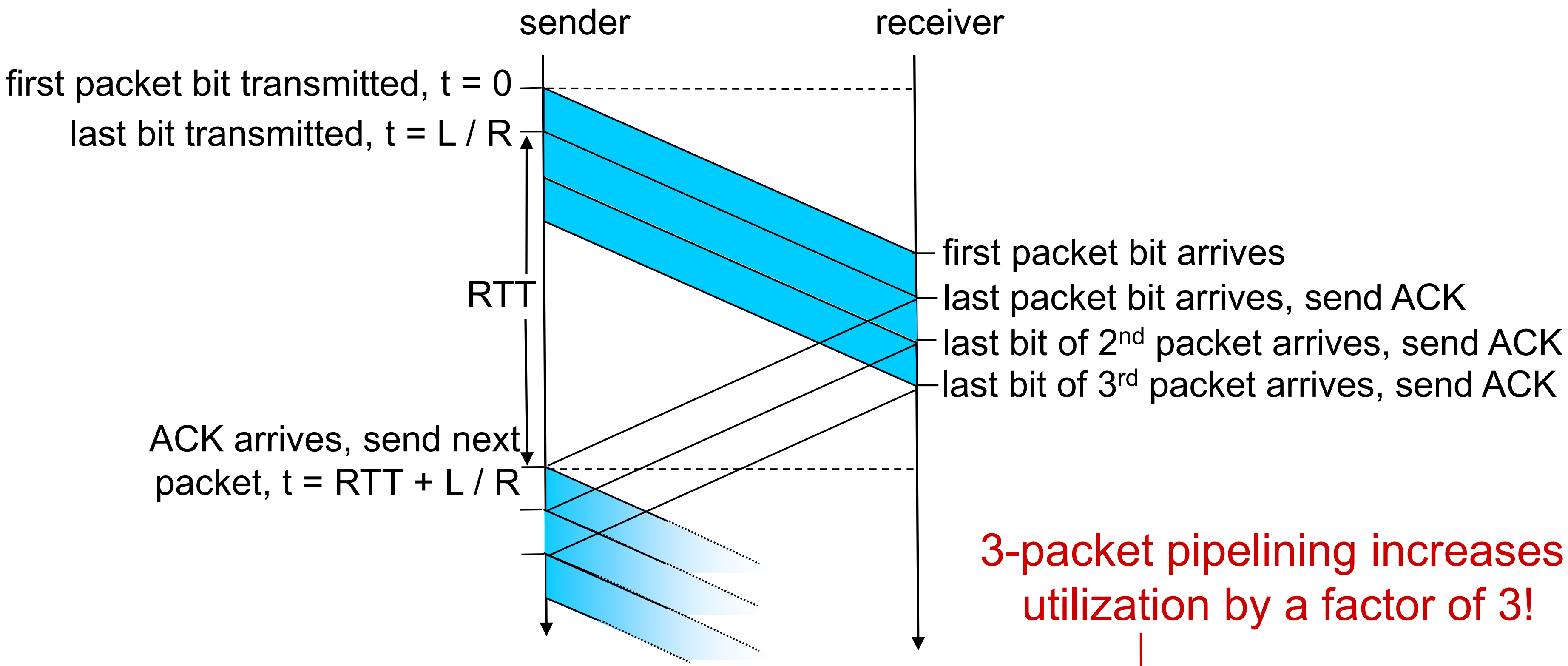
pipelining: sender allows multiple, “in-flight”, yet-to-be-acknowledged packets

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



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

Pipelining: increased utilization



3-packet pipelining increases utilization by a factor of 3!

$$U_{sender} = \frac{3L / R}{RTT + L / R} = \frac{.0024}{30.008} = 0.00081$$

Pipelining: additional mechanims

- Range of sequence numbers must be increased
- Buffering at sender and/or receiver
  - The sender must buffer all packets not yet acknowledged
  - The receiver may buffer out-of-order packets
- The range of seq numbers and buffer size depend on how the protocol manages lost, corrupted, and delayed packets
- Error recovery strategies
  - Go-back-N
  - Selective Repeat

Pipelining protocols

Go-back-N

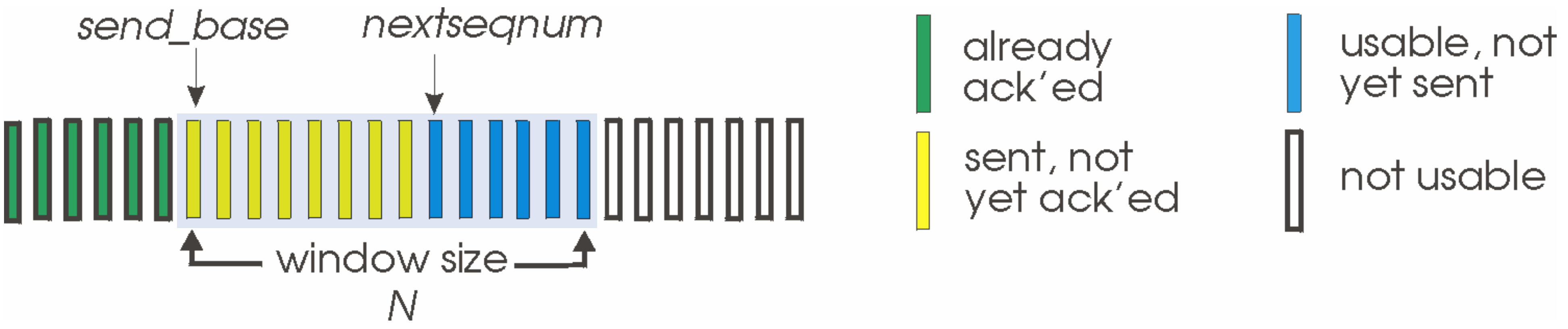
- *sender:* up to N unACKed pkts in pipeline
- *receiver:* only sends cumulative ACKs
  - doesn't ACK pkt if there's a gap
- *sender:* has timer for oldest unACKed pkt
  - if timer expires: retransmit all unACKed packets

Selective Repeat

- *sender:* up to N unACKed packets in pipeline
- *receiver:* ACKs individual pkts
- *sender:* maintains timer for each unACKed pkt
  - if timer expires: retransmit only unACKed packets

Go-Back-N: sender

- sender: "window" of up to N, consecutive transmitted but unACKed pkts
  - k-bit seq # in pkt header



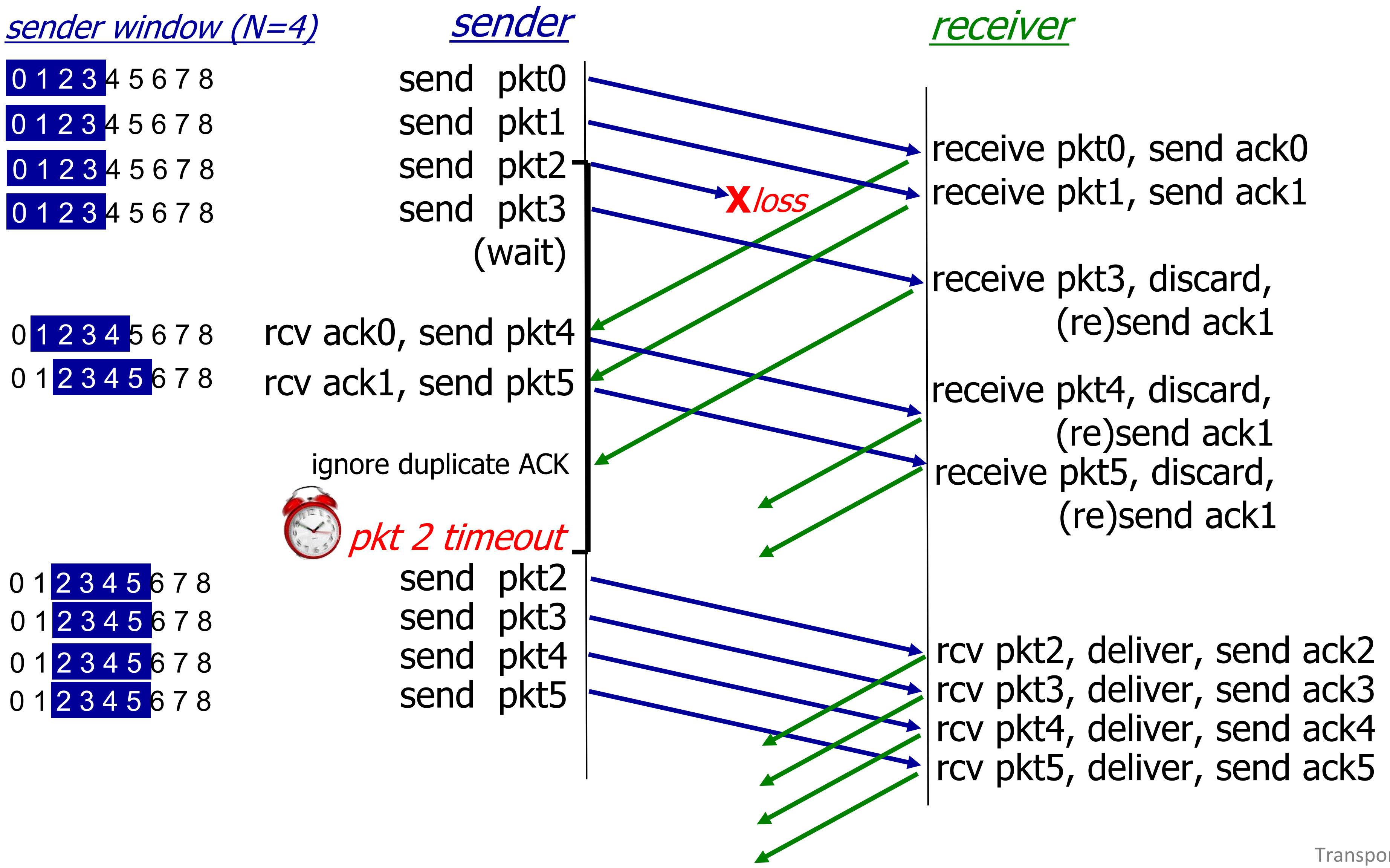
- *cumulative ACK:* ACK(*n*): ACKs all packets up to, including seq # *n*
  - on receiving ACK(*n*): move window forward to begin at *n+1*
- timer for oldest in-flight packet
- *timeout(n):* retransmit packet n and all higher seq # packets in window

Go-Back-N: receiver

- ACK-only: always send ACK for correctly-received packet so far, with highest *in-order* seq #
  - may generate duplicate ACKs
  - need only remember `rcv_base`
- on receipt of out-of-order packet:
  - can discard (don't buffer) or buffer: an implementation decision
  - re-ACK pkt with highest in-order seq #

Receiver view of sequence number space:

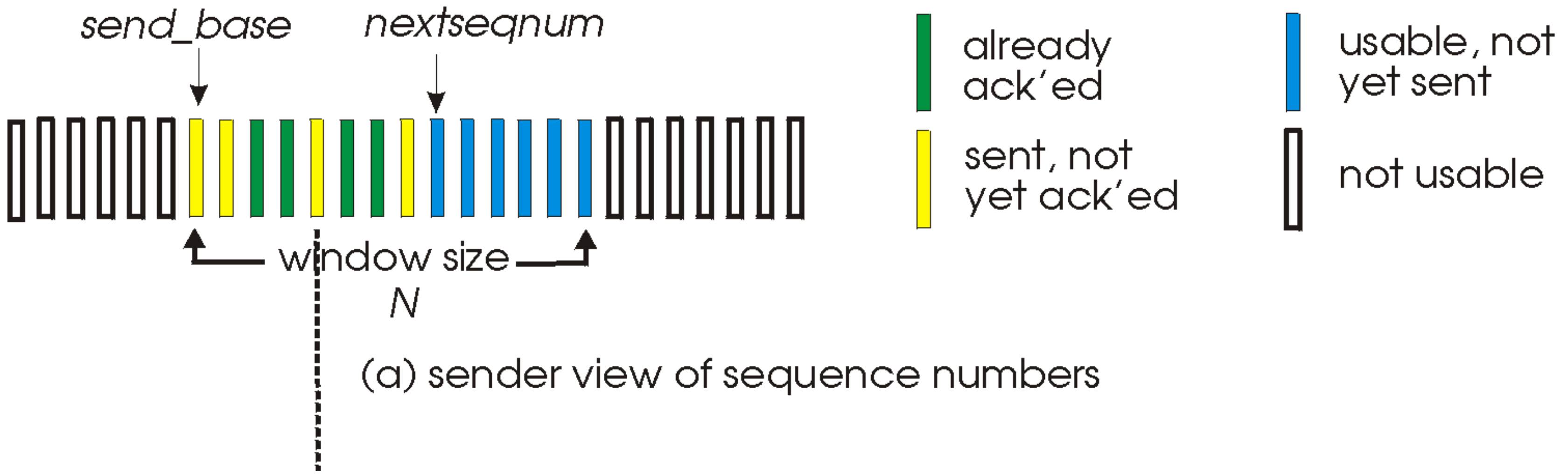
Go-Back-N in action



Selective repeat

- receiver *individually* acknowledges all correctly received packets
  - buffers packets, as needed, for eventual in-order delivery to upper layer
- sender times-out/retransmits individually for unACKed packets
  - sender maintains timer for each unACKed pkt
- sender window
  - *N* consecutive seq #s
  - limits seq #s of sent, unACKed packets

Selective repeat: sender, receiver windows



Selective repeat: sender and receiver

sender

data from above:

- if next available seq # in window, send packet

timeout(*n*):

- resend packet *n*, restart timer

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

- mark packet *n* as received
- if n smallest unACKed packet, advance window base to next unACKed seq #

receiver

packet *n* in [rcvbase, rcvbase+N-1]

- send ACK(*n*)
- out-of-order: buffer
- in-order: deliver (also deliver buffered, in-order packets), advance window to next not-yet-received packet

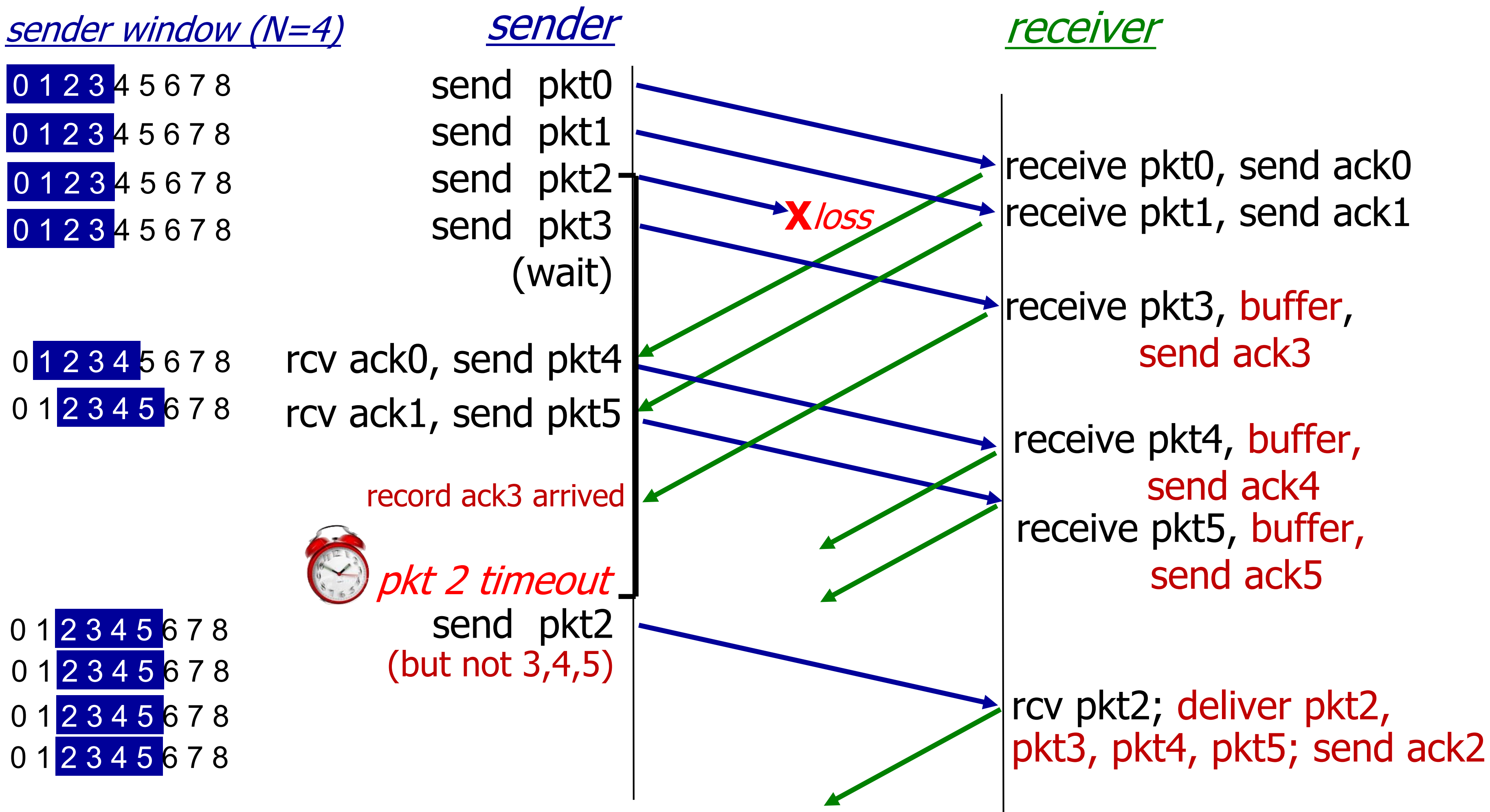
packet *n* in [rcvbase-N,rcvbase-1]

- ACK(*n*)

otherwise:

- ignore

Selective Repeat in action

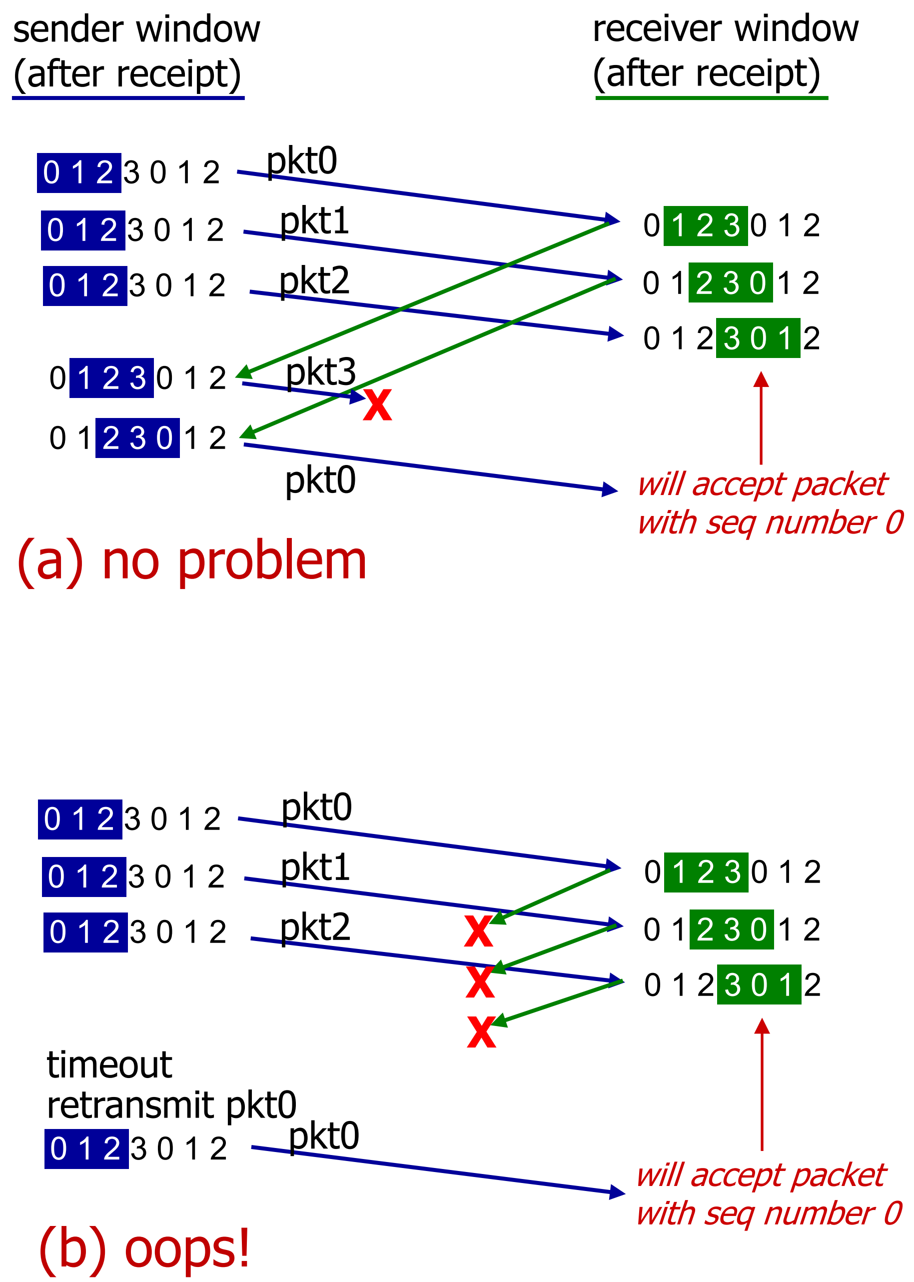


Q: what happens when ack2 arrives?

Selective repeat: a dilemma!

example:

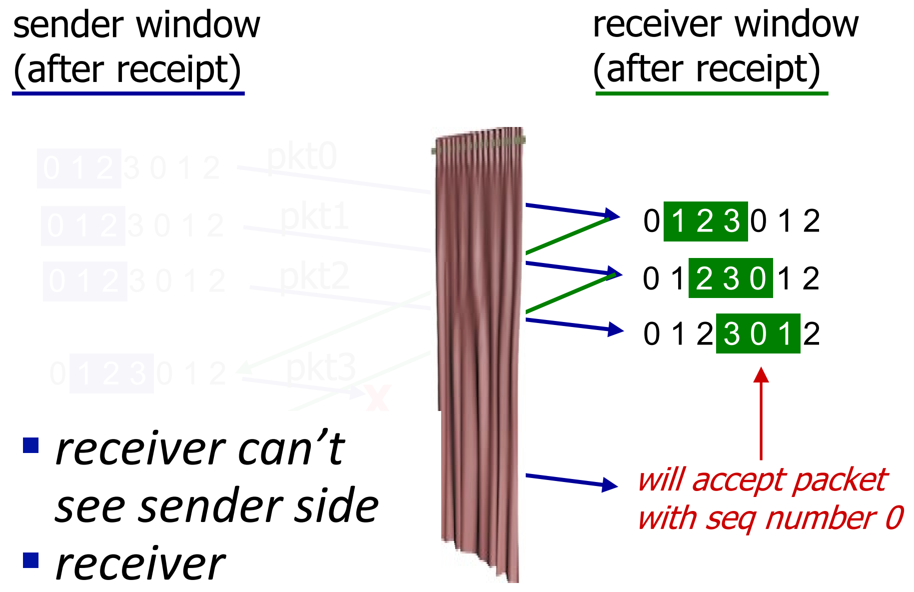
- seq #s: 0, 1, 2, 3 (base 4 counting)
- window size=3



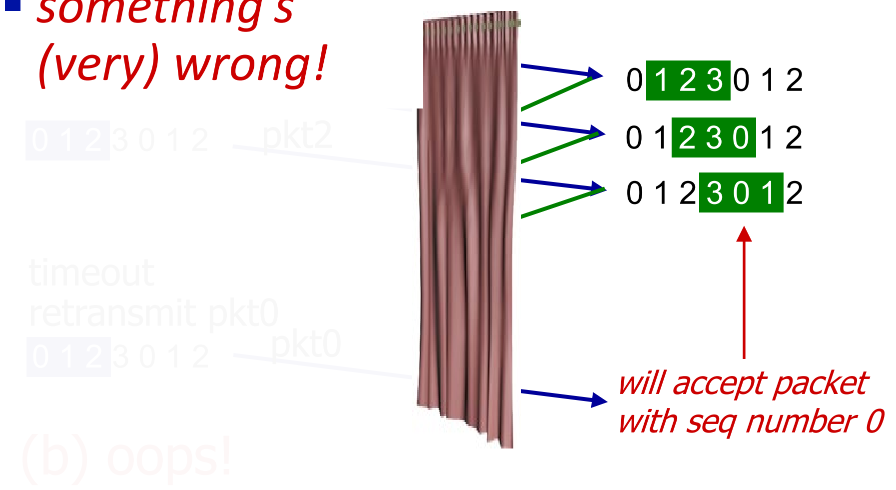
Selective repeat: a dilemma!

example:

- seq #s: 0, 1, 2, 3 (base 4 counting)
- window size=3



- *receiver can't see sender side*
- *receiver behavior identical in both cases!*
- *something's (very) wrong!*



Q: what relationship is needed between sequence # size and window size to avoid problem in scenario (b)?

Roadmap

- Introduction
- Reliable communication over unreliable links
  - Framing
  - Error detection, Correction
  - Reliable Data Transfer
- PPP
- Multiple Access protocols
- LANs
  - Addressing
  - Ethernet

Point-to-Point Protocol (PPP)

- packet framing: encapsulation of network-layer datagram in Data Link frame
  - carry network layer data of any network layer protocol (not just IP)
  - ability to demultiplex upwards
- bit transparency: must carry any bit pattern in the data field
- error detection (no correction)
- connection liveness: detect and signal link failure to network layer
- network layer address negotiation: endpoint can learn/configure each other's network address

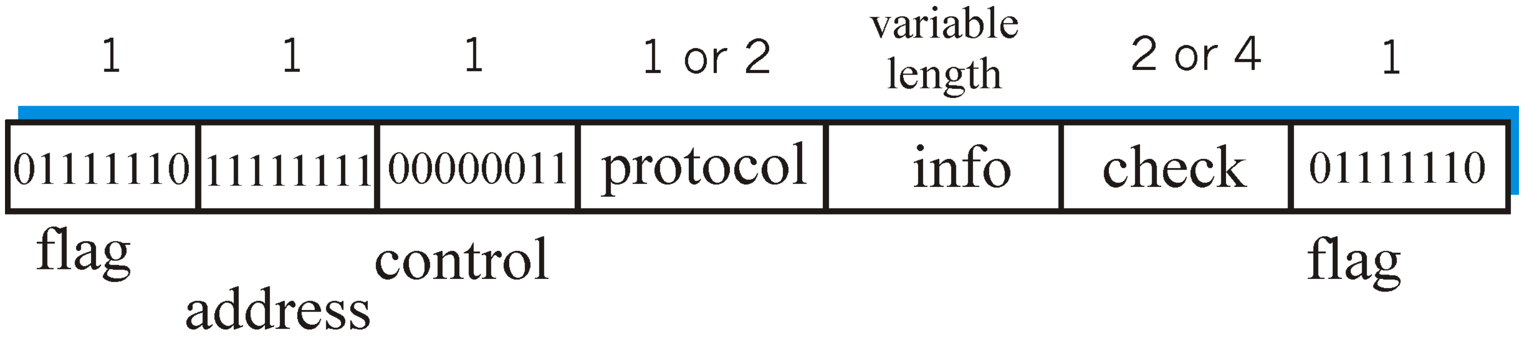
Point-to-Point Protocol (PPP)

- No error correction
- No error recovery
- No flow control
- Possible out of order delivery
- No point-to-multipoint comm
  - Other DL protocols supports this feature (e.g., HDLC)

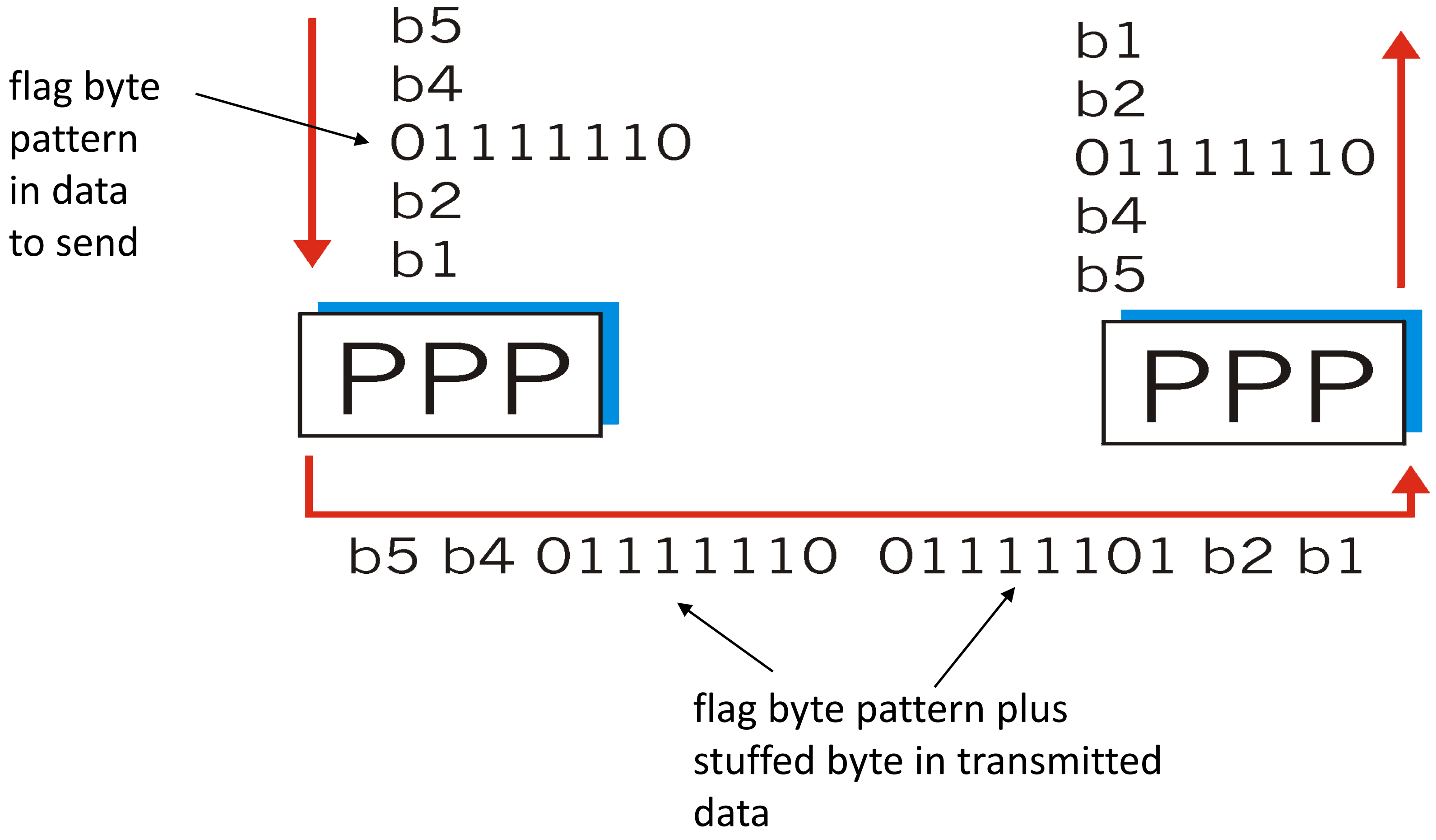
Error recovery, flow control, data re-ordering all delegated to higher layers (e.g., TCP)!

Point-to-Point Protocol (PPP)

- Flag: delimiter (framing)
- Address: does nothing (only one option)
- Control: does nothing; in the future possible multiple control fields
- Protocol: upper layer protocol to which frame delivered (e.g., IP)
- info: upper-layer data being carried
- check: cyclic redundancy check for error detection



Byte stuffing/unstuffing



Byte stuffing/unstuffing

- Sender (byte stuffing)
  - 01111110 → 01111101 01111110
  - 01111101 → 01111101 01111101
- Receiver (byte unstuffing)
  - 01111101 01111110 → 01111110
  - 01111101 01111101 → 01111101

Roadmap

- Introduction
- Reliable communication over unreliable links
  - Framing
  - Error detection, Correction
  - Reliable Data Transfer
- PPP
- Multiple Access protocols
- LANs
  - Addressing
  - Ethernet

Multiple access links, protocols

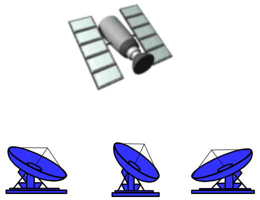
two types of "links":

- point-to-point
  - point-to-point link between Ethernet switch, host
  - PPP for dial-up access
- broadcast (shared wire or medium)
  - old-fashioned Ethernet
  - upstream HFC in cable-based access network
  - 802.11 wireless LAN, 4G/4G. satellite

shared wire (e.g., cabled Ethernet)

shared radio: 4G/5G

shared radio: WiFi



shared radio: satellite

humans at a cocktail party (shared air, acoustical)

Multiple access protocols

- single shared broadcast channel
- two or more simultaneous transmissions by nodes: interference
  - *collision* if node receives two or more signals at the same time

multiple access protocol

- distributed algorithm that determines how nodes share channel, i.e., determine when node can transmit
- communication about channel sharing must use channel itself!
  - no out-of-band channel for coordination

An ideal multiple access protocol

given: multiple access channel (MAC) of rate *R* bps

desiderata:

1. when one node wants to transmit, it can send at rate *R*.
2. when M nodes want to transmit, each can send at average rate *R/M*
3. fully decentralized:
   - no special node to coordinate transmissions
   - no synchronization of clocks, slots
4. simple

MAC protocols: taxonomy

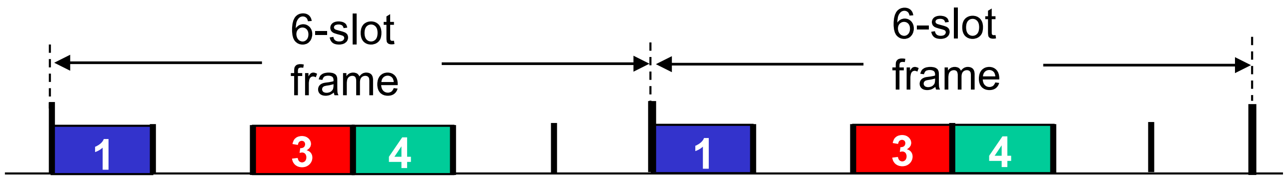
three broad classes:

- channel partitioning
  - divide channel into smaller “pieces” (time slots, frequency, code)
  - allocate piece to node for exclusive use
- *random access*
  - channel not divided, allow collisions
  - “recover” from collisions
- “taking turns”
  - nodes take turns, but nodes with more to send can take longer turns

Channel partitioning MAC protocols: TDMA

TDMA: time division multiple access

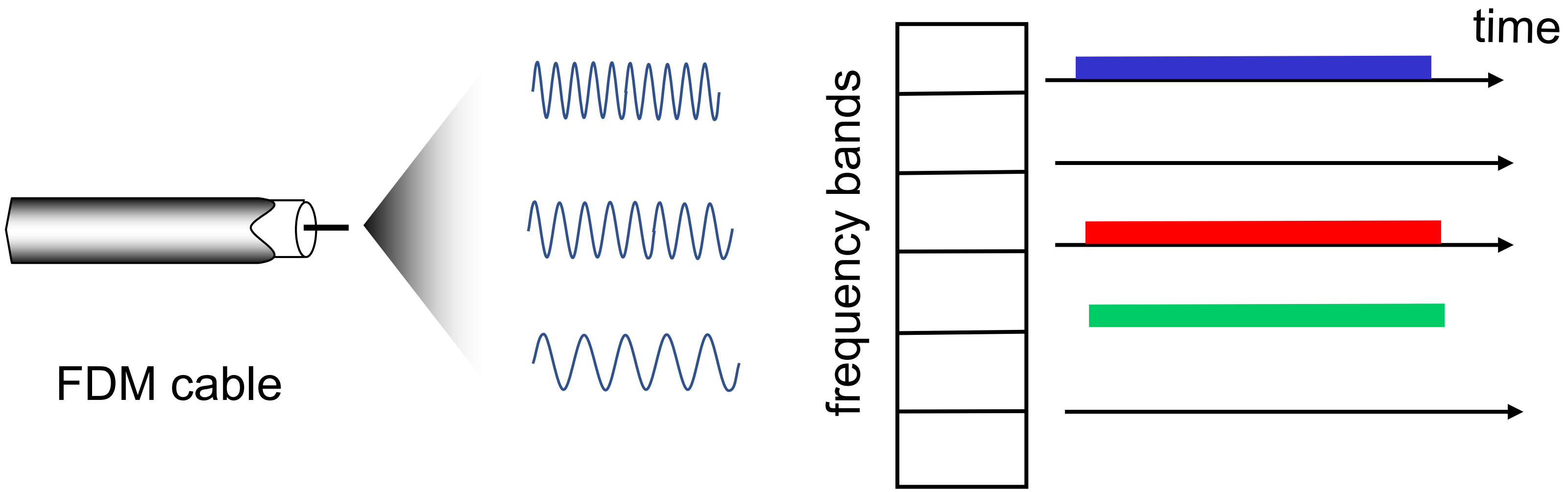
- access to channel in "rounds"
- each station gets fixed length slot (length = packet transmission time) in each round
- unused slots go idle
- example: 6-station LAN, 1,3,4 have packets to send, slots 2,5,6 idle



Channel partitioning MAC protocols: FDMA

FDMA: frequency division multiple access

- channel spectrum divided into frequency bands
- each station assigned fixed frequency band
- unused transmission time in frequency bands go idle
- example: 6-station LAN, 1,3,4 have packet to send, frequency bands 2,5,6 idle



Random access protocols

- when node has packet to send
  - transmit at full channel data rate R.
  - no *a priori* coordination among nodes
- two or more transmitting nodes: "collision"
- random access MAC protocol specifies:
  - how to detect collisions
  - how to recover from collisions (e.g., via delayed retransmissions)
- examples of random access MAC protocols:
  - ALOHA, slotted ALOHA
  - CSMA, CSMA/CD, CSMA/CA

Slotted ALOHA

assumptions:

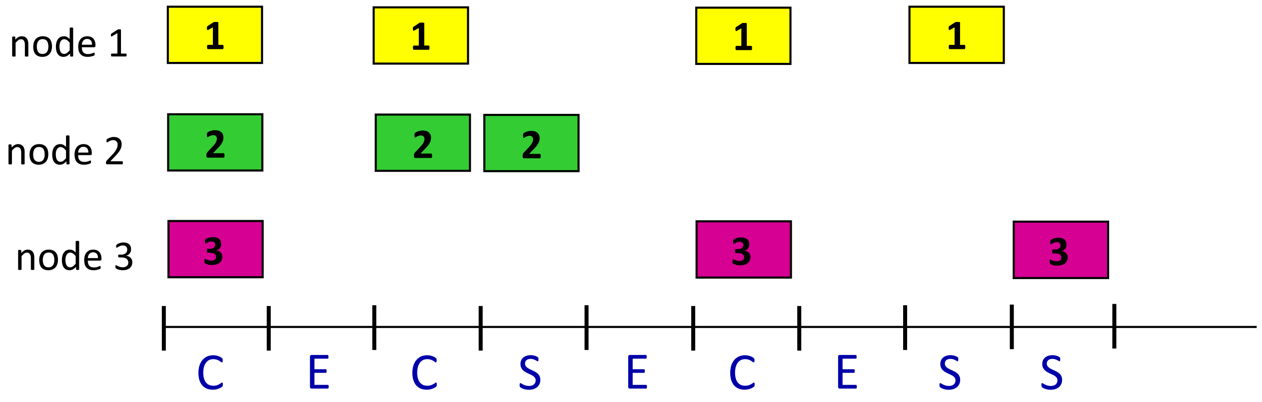
- all frames same size
- time divided into equal size slots (time to transmit 1 frame)
- nodes start to transmit only slot beginning
- nodes are synchronized
- if 2 or more nodes transmit in slot, all nodes detect collision

operation:

- when node obtains fresh frame, transmits in next slot
  - *if no collision:* node can send new frame in next slot
  - *if collision:* node retransmits frame in each subsequent slot with probability *p* until success

randomization – *why*?

Slotted ALOHA



C: collision
S: success
E: empty

Pros:

- single active node can continuously transmit at full rate of channel
- highly decentralized: only slots in nodes need to be in sync
- simple

Cons:

- collisions, wasting slots
- idle slots
- nodes may be able to detect collision in less than time to transmit packet
- clock synchronization

Slotted ALOHA: efficiency

efficiency: long-run fraction of successful slots (many nodes, all with many frames to send)

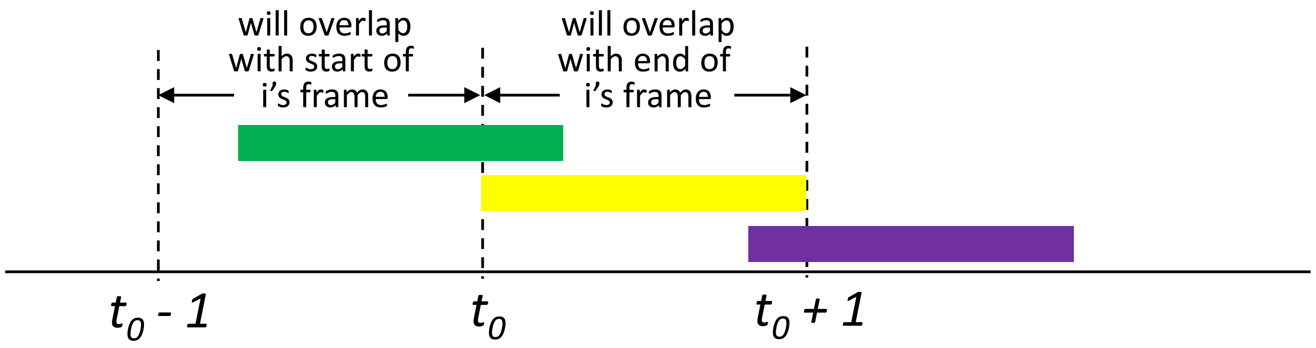
- *suppose:* N nodes with many frames to send, each transmits in slot with probability p
  - prob that given node has success in a slot = $p(1-p)^{N-1}$
  - prob that *any* node has a success = $Np(1-p)^{N-1}$
  - max efficiency: find p^* that maximizes $Np(1-p)^{N-1}$
  - for many nodes, take limit of $Np^*(1-p^*)^{N-1}$ as N goes to infinity, gives:

  max efficiency = 1/e = .37

- *at best:* channel used for useful transmissions 37% of time!

Pure ALOHA

- unslotted Aloha: simpler, no synchronization
  - when frame first arrives: transmit immediately
- collision probability increases with no synchronization:
  - frame sent at t_0 collides with other frames sent in $[t_0-1,t_0+1]$



- pure Aloha efficiency: 18% !

Pure ALOHA efficiency

P(success by given node) = P(node transmits) *

P(no other node transmits in $[t_0-1,t_0]$ * *

P(no other node transmits in $[t_0-1,t_0]$

$= p \cdot (1-p)^{N-1} \cdot (1-p)^{N-1}$

$= p \cdot (1-p)^{2(N-1)}$

… choosing optimum p and then letting $n \rightarrow \infty$

$= 1/(2e) = .18$

even worse than slotted Aloha!

CSMA (carrier sense multiple access)

simple CSMA: listen before transmit:

- if channel sensed idle: transmit entire frame
- if channel sensed busy: defer transmission

- human analogy: don't interrupt others!

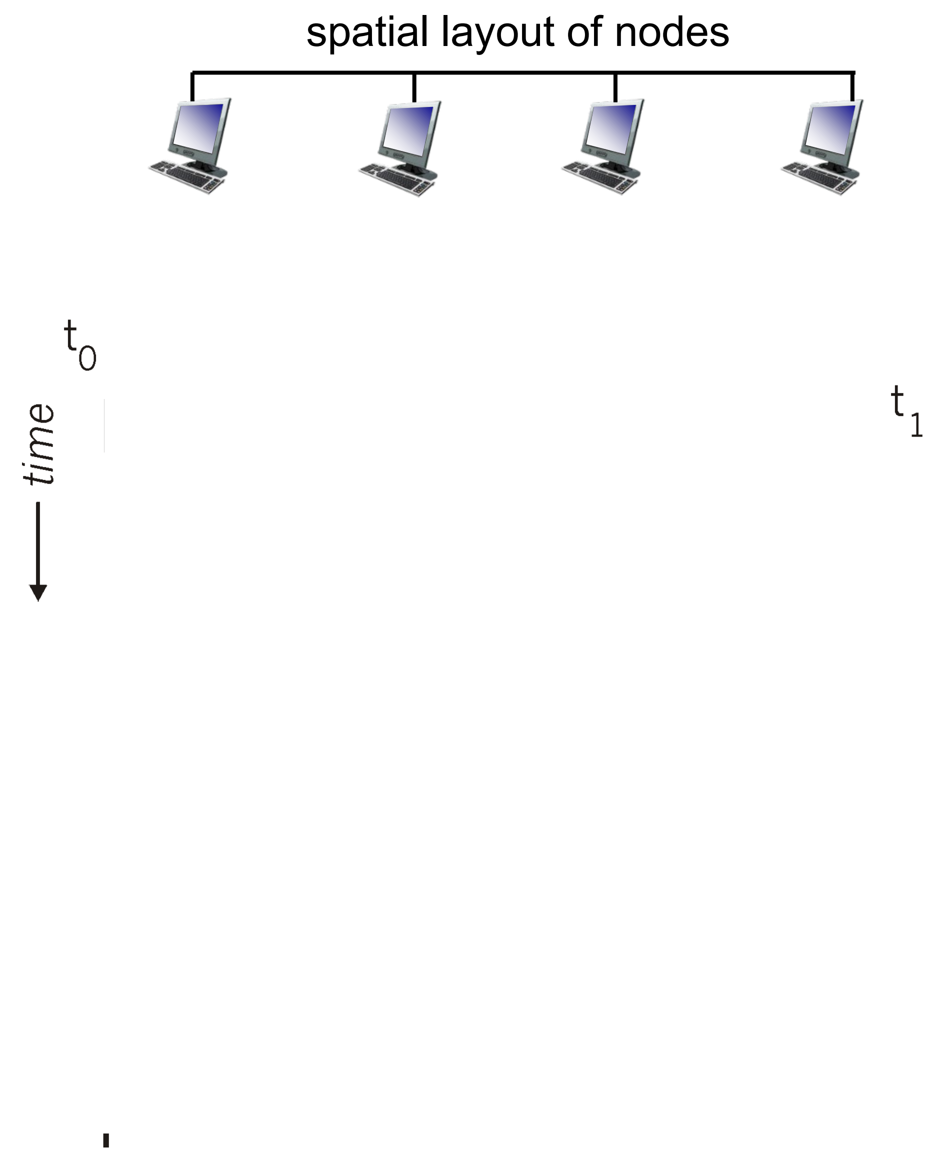
CSMA/CD: CSMA with *collision detection*

- collisions *detected* within short time
- colliding transmissions aborted, reducing channel wastage
- collision detection easy in wired, difficult with wireless

- human analogy: the polite conversationalist

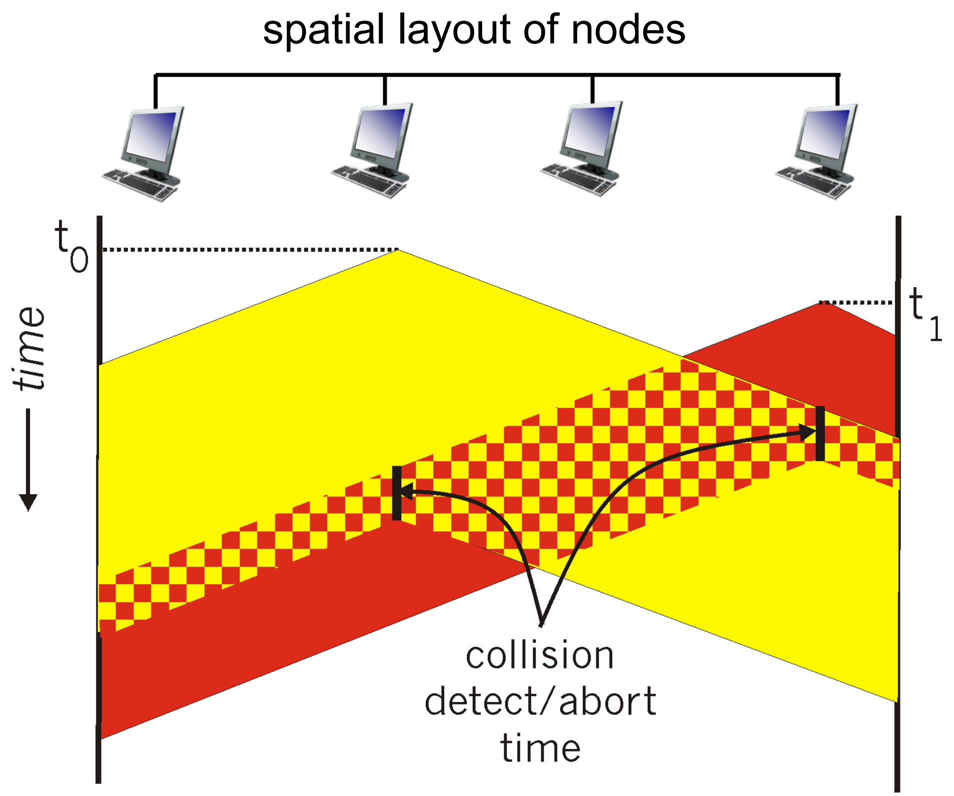
CSMA: collisions

- collisions *can* still occur with carrier sensing:
  - propagation delay means two nodes may not hear each other's just-started transmission
- collision: entire packet transmission time wasted
  - distance & propagation delay play role in in determining collision probability



CSMA/CD:

- CSMA/CS reduces the amount of time wasted in collisions
  - transmission aborted on collision detection



Ethernet CSMA/CD algorithm

1. NIC receives datagram from network layer, creates frame
2. If NIC senses channel:

   if idle: start frame transmission.

   if busy: wait until channel idle, then transmit
3. If NIC transmits entire frame without collision, NIC is done with frame !
4. If NIC detects another transmission while sending: abort, send jam signal
5. After aborting, NIC enters *binary (exponential) backoff:*
   - after *m*th collision, NIC chooses *K* at random from $\{0,1,2, \ldots, 2^m-1\}$. NIC waits $K \cdot 512$ bit times, returns to Step 2
   - more collisions: longer backoff interval

“Taking turns” MAC protocols

channel partitioning MAC protocols:

- share channel *efficiently* and *fairly* at high load
- inefficient at low load: delay in channel access, 1/N bandwidth allocated even if only 1 active node!

random access MAC protocols

- efficient at low load: single node can fully utilize channel
- high load: collision overhead

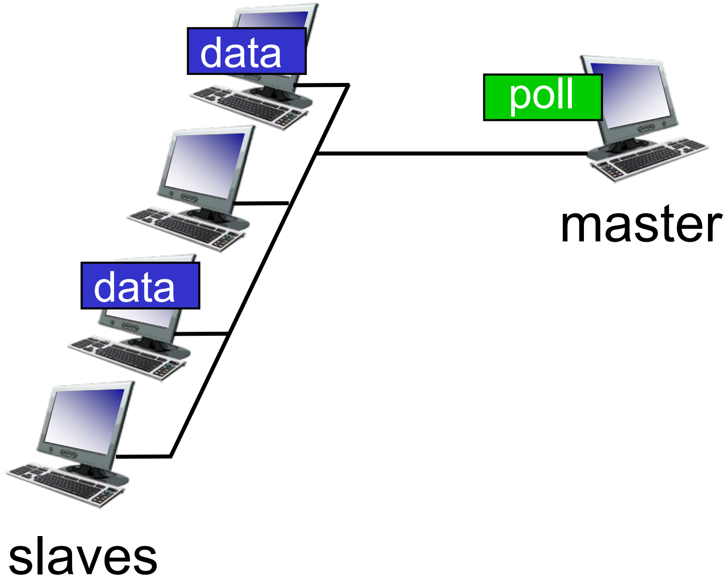
“taking turns” protocols

- look for best of both worlds!

"Taking turns" MAC protocols

polling:

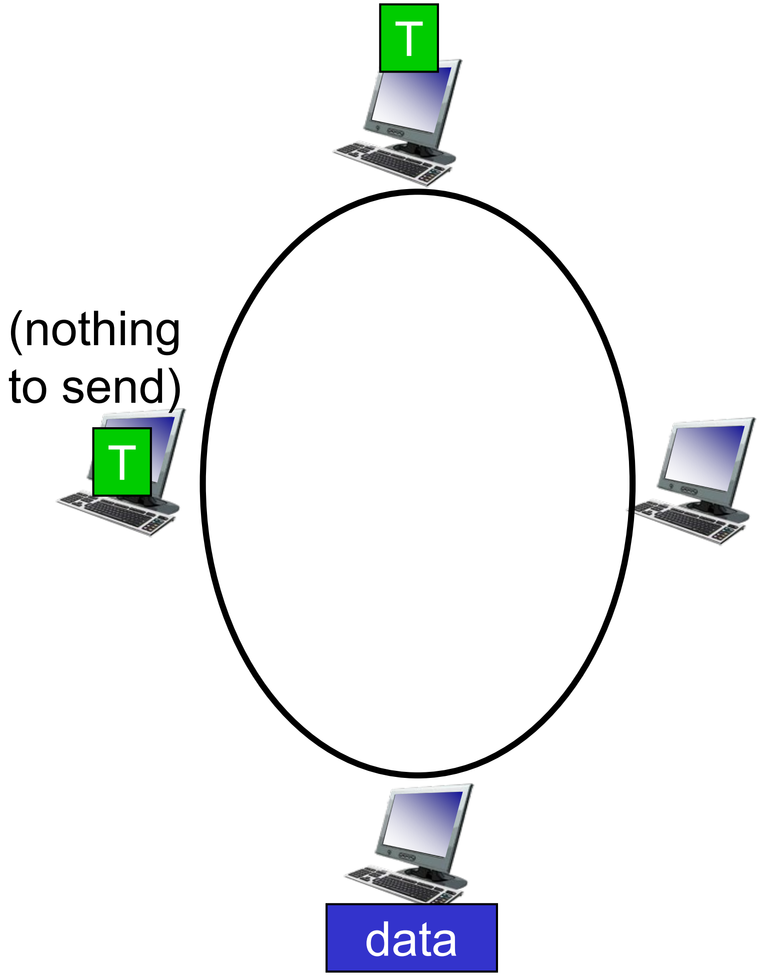
- master node "invites" other nodes to transmit in turn
- typically used with "dumb" devices
- concerns:
  - polling overhead
  - latency
  - single point of failure (master)



"Taking turns" MAC protocols

token passing:

- control *token* passed from one node to next sequentially.
- token message
- concerns:
  - token overhead
  - latency
  - single point of failure (token)



Summary of MAC protocols

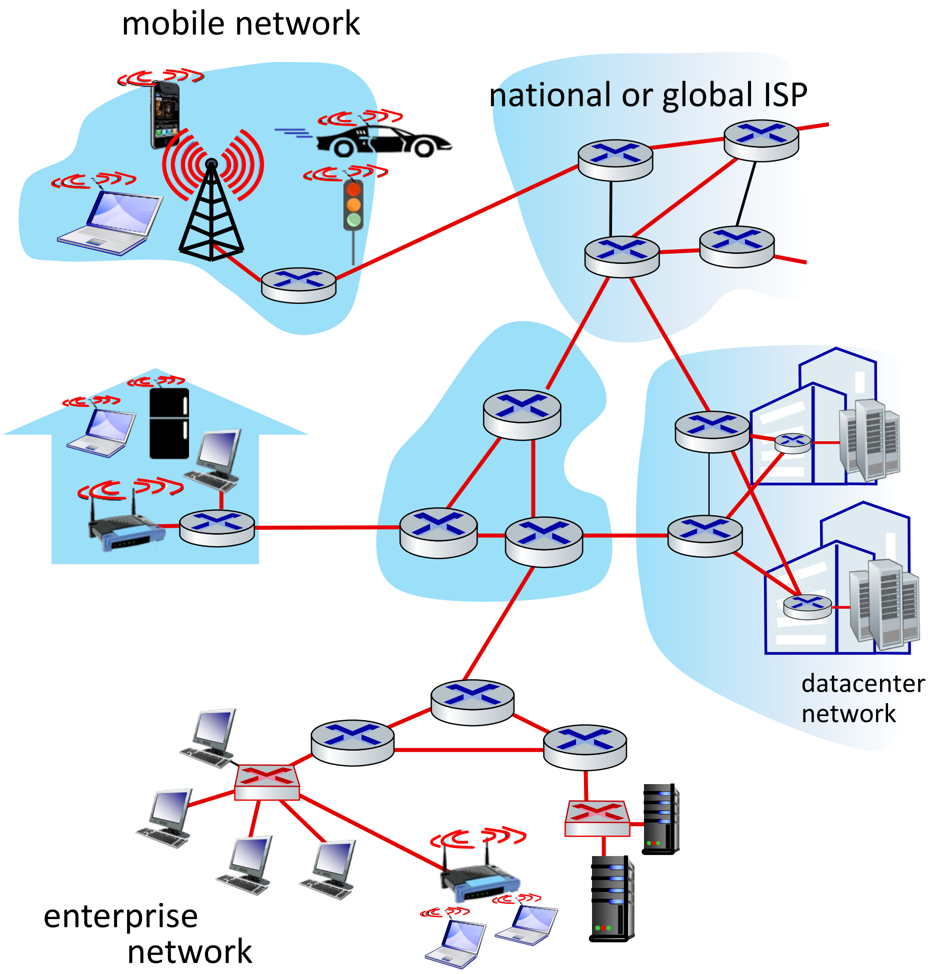
- channel partitioning, by time, frequency or code
  - Time Division, Frequency Division
- random access (dynamic),
  - ALOHA, S-ALOHA, CSMA, CSMA/CD
  - carrier sensing: easy in some technologies (wire), hard in others (wireless)
  - CSMA/CD used in Ethernet
  - CSMA/CA used in 802.11
- taking turns
  - polling from central site, token passing
  - Bluetooth, FDDI, token ring

Roadmap

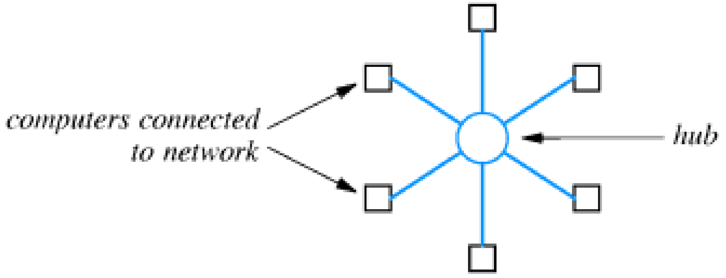
- Introduction
- Reliable communication over unreliable links
  - Framing
  - Error detection, Correction
  - Reliable Data Transfer
- PPP
- Multiple Access protocols
- LANs
  - Addressing
  - Ethernet

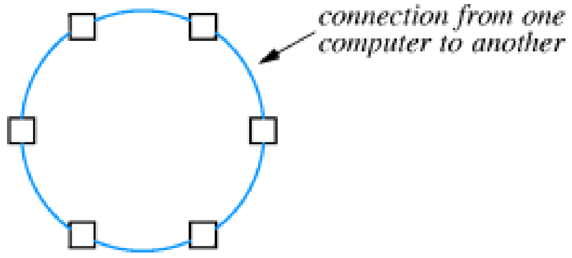
Local Area Networks (LANs)

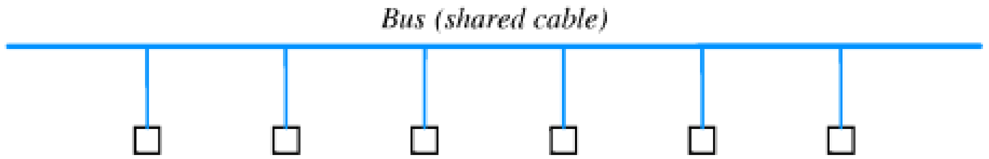
- Broadcast Medium
  - Medium Access Control Protocol
  - MAC addressing
- Limited Coverage Area
  - Home, Building, Campus
- High Bit Rate
  - 100 Mbps – 10 Gbps



Local Area Networks (LANs)







Broadcast Medium

- Medium Access Control (MAC) Protocol for channel access
- MAC addressing

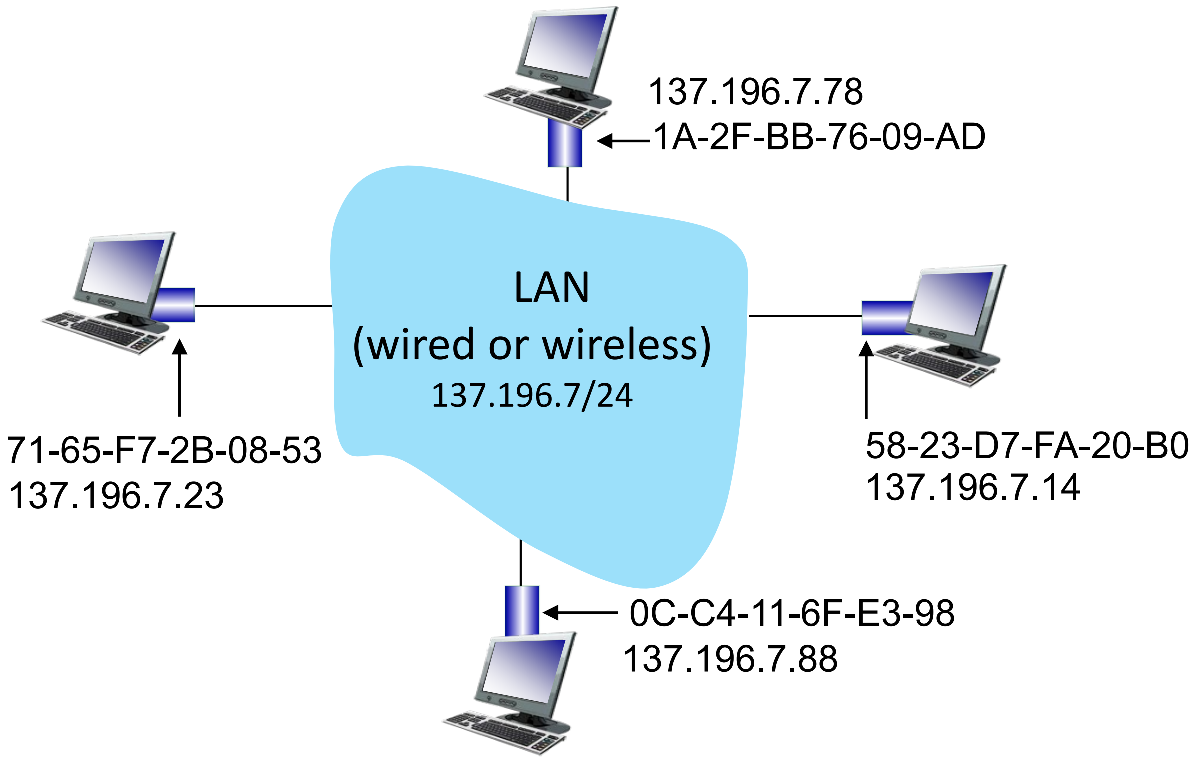
MAC addresses

- MAC (or LAN or physical or Ethernet) address:
  - function: used "locally" to get frame from one interface to another physically-connected interface
  - 48-bit MAC address (for most LANs) burned in NIC ROM, also sometimes software settable
  - e.g.: 1A-2F-BB-76-09-AD
- 32-bit IP address:
  - *network-layer* address for interface
  - used for layer 3 (network layer) forwarding
  - e.g.: 128.119.40.136

MAC addresses

each interface on LAN

- has unique 48-bit MAC address
- has a locally unique 32-bit IP address (as we've seen)



MAC addresses

- MAC address allocation administered by IEEE
- manufacturer buys portion of MAC address space (to assure uniqueness)
- analogy:
  - MAC address: like Social Security Number
  - IP address: like postal address
- MAC flat address: portability
  - can move interface from one LAN to another
  - recall IP address *not* portable: depends on IP subnet to which node is attached

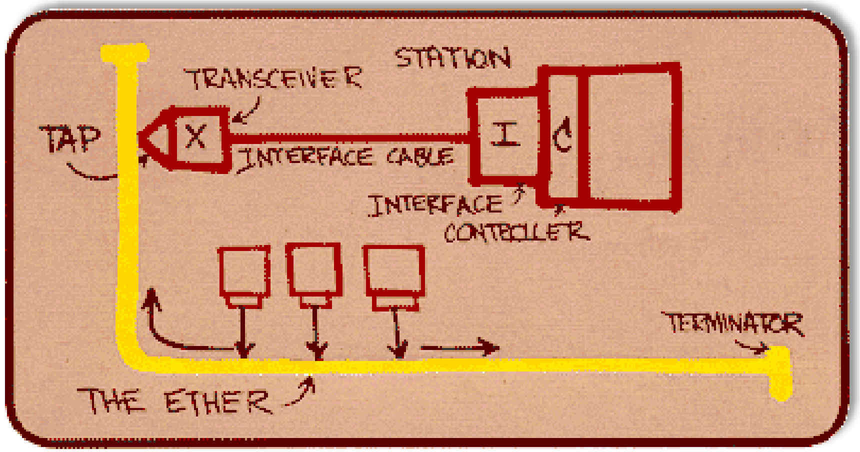
Roadmap

- Introduction
- Reliable communication over unreliable links
  - Framing
  - Error detection, Correction
  - Reliable Data Transfer
- PPP
- Multiple Access protocols
- LANs
  - Addressing
  - Ethernet

Ethernet

"dominant" wired LAN technology:

- first widely used LAN technology
- simpler, cheap
- kept up with speed race: 10 Mbps – 400 Gbps
- single chip, multiple speeds (e.g., Broadcom BCM5761)



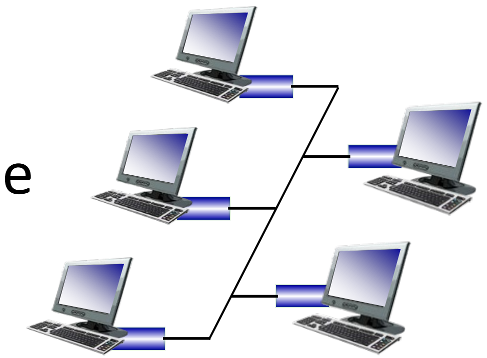
Metcalfe's Ethernet sketch

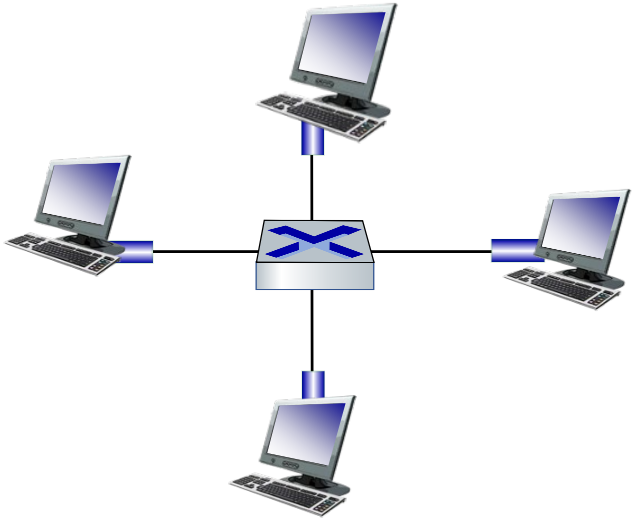
https://www.uspto.gov/learning-and-resources/journeys-innovation/audio-stories/defying-doubters

Ethernet: physical topology

- bus: popular through mid 90s
  - all nodes in same collision domain (can collide with each other)
- switched: prevails today
  - active link-layer 2 *switch* in center
  - each "spoke" runs a (separate) Ethernet protocol (nodes do not collide with each other)

bus: coaxial cable





switched

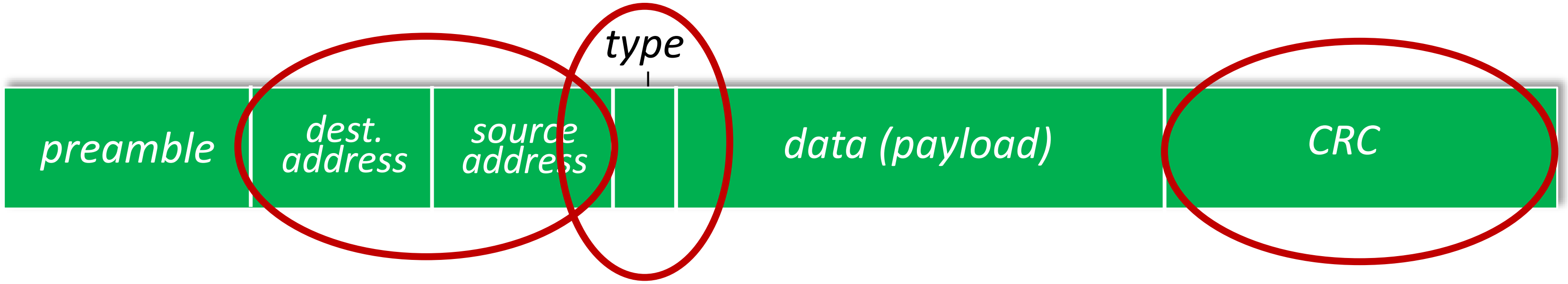
Ethernet frame structure

sending interface encapsulates IP datagram (or other network layer protocol packet) in Ethernet frame

preamble:

- used to synchronize receiver, sender clock rates
- 7 bytes of 10101010 followed by one byte of 10101011

Ethernet frame structure (more)



- addresses: 6 byte source, destination MAC addresses
  - if adapter receives frame with matching destination address, or with broadcast address (e.g., ARP packet), it passes data in frame to network layer protocol
  - otherwise, adapter discards frame
- type: indicates higher layer protocol
  - mostly IP but others possible, e.g., Novell IPX, AppleTalk
  - used to demultiplex up at receiver
- CRC: cyclic redundancy check at receiver
  - error detected: frame is dropped

Ethernet: unreliable, connectionless

- connectionless: no handshaking between sending and receiving NICs
- unreliable: receiving NIC doesn't send ACKs or NAKs to sending NIC
  - data in dropped frames recovered only if initial sender uses higher layer rdt (e.g., TCP), otherwise dropped data lost
- Ethernet's MAC protocol: unslotted CSMA/CD with binary backoff

Ethernet CSMA/CD protocol

1. NIC receives data from network layer, creates frame
2. If NIC senses channel idle for **96 bits**, starts frame transmission

   If NIC senses channel busy, waits until channel idle for **96 bits**, then transmits
3. If NIC transmits entire frame without detecting another transmission, NIC is done with frame !
4. If NIC detects another transmission while transmitting, aborts and sends **48-bit jam signal**
5. After aborting, NIC enters **exponential backoff**: after the *i*-th collision
   - NIC chooses *K* at random from {0,1,2,…,2^m-1}, where m=max (i, 10)
   - Backoff timer ← K*512 bit times
   - Wait until backoff timer expires
   - Returns to step 2

After 17 trials the frame is dropped

Ethernet CSMA/CD protocol

Jam Signal: make sure all other transmitters are aware of collision; 48 bits

Bit time: 0.1 microsec for 10 Mbps Ethernet

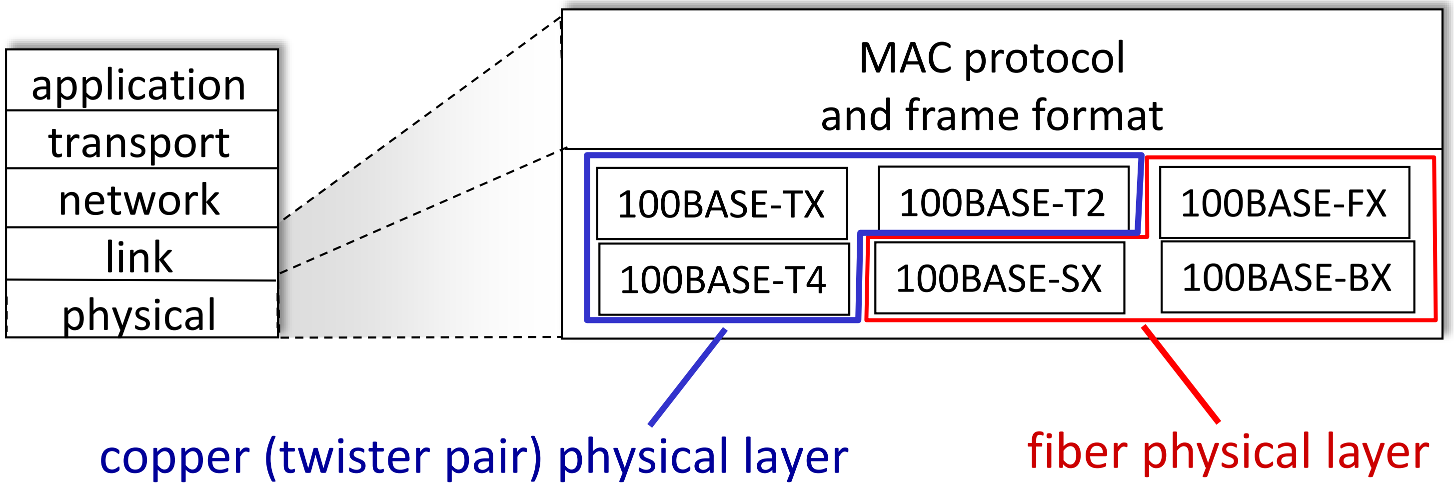
- K=1023
- K*512= 523.776 bit times
- Wait time is about 50 msec

Exponential Backoff

- ***Goal***: adapt retransmission attempts to estimated current load
  - heavy load: random wait will be longer
- first collision: choose K from {0,1}; delay is K· 512 bit transmission times
- after second collision: choose K from {0,1,2,3}…
- after ten collisions, choose K from {0,1,2,3,4,…,1023}

802.3 Ethernet standards: link & physical layers

- *many* different Ethernet standards
  - common MAC protocol and frame format
  - different speeds: 2 Mbps, 10 Mbps, 100 Mbps, 1Gbps, 10 Gbps, 40 Gbps
  - different physical layer media: fiber, cable



Summary

- Direct connection networks
- Principles behind Data Link layer services:
  - error detection, correction
  - reliable data transfer
  - sharing a broadcast channel: multiple access
  - link layer addressing
- Instantiation and implementation of various link layer technologies
  - PPP
  - Ethernet