COMP2001J Computer Networks

Lecture 4 – Data Link Layer
(Flow Control)

Dr. Shen WANG (王燊)

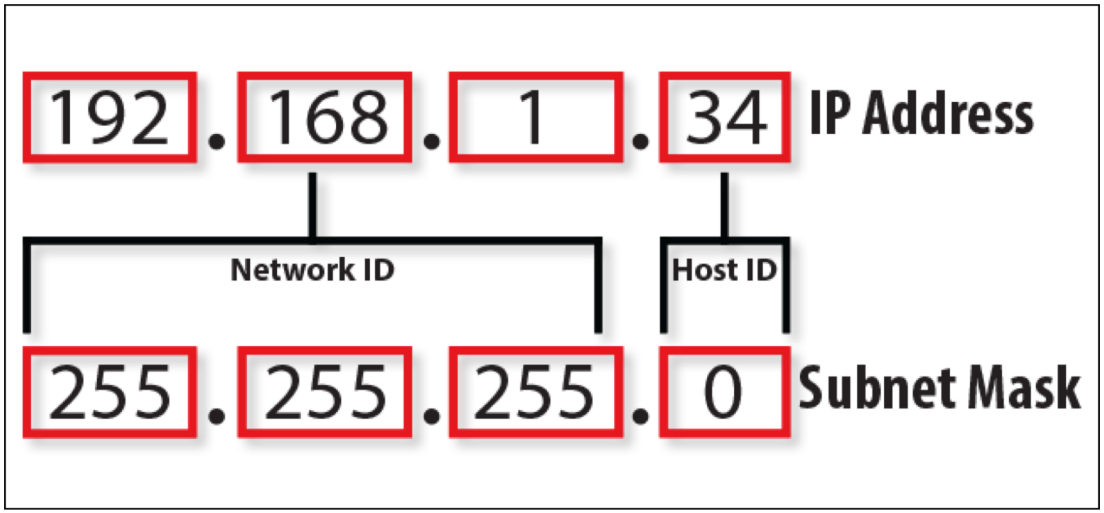
shen.wang@ucd.ie

About Labs

- Q1. Submission?
  - You only need to submit report (with other required files) for Lab Exams.
  - However, you are still highly encouraged to be in regular lab sessions, as lab exams are built based on several previous regular lab sessions step by step.
- Q2. IP address? Gateway? Default gateway?
  - IP address: The address of your device on the internet.
  - Gateway: IP address of your closest router interface.
  - Default gateway: tells your device what is the IP address of the gateway in your subnet

About labs

- Q3. IP address and Subnet Mask?
  - An IP address is composed of a subnet id (prefix) and a host id (suffix)
  - A subnet mask tells you which part of its corresponding IP address is a subnet id, and which part is the other.



Continuous Assessment

- **30% (confirmed! No further change on this!):**
  - Mid-term Exam **10% (as of now)**
  - Quiz 1
  - Quiz 2
  - Packet Tracer Exam
  - Wireshark Exam

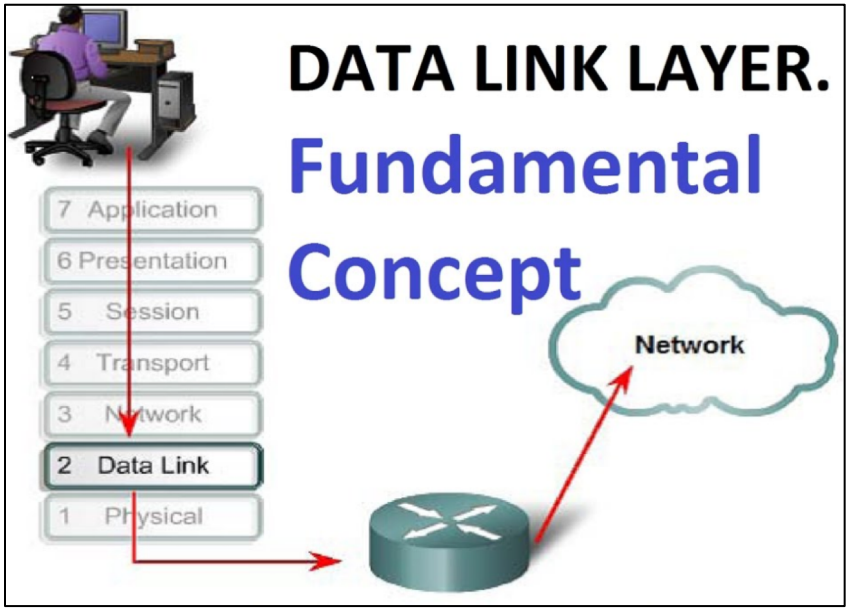
- Each one of them equally important (5%)?
  - Will be adjusted according to your performance
  - **Not confirmed until** you finished all of them!

Outline

- Reliable Communication
  - Acknowledgement
  - Retransmission
- Flow Control
  - Stop-and-Wait
  - Sliding-Window
    - Go-back-N
    - Selective Repeat
  - ARQ extension:
    - Means “flow+error control”

Point-to-Point

Not end-to-end!



Reliable Communication

- The “datagram packet switching” network can not naturally provide reliable communication, for example:
  - Missing Frames:
    - Send “1, 2, 3, 4” -> Receive “1, 2, 4”
  - Out-of-sequenced Frames:
    - Send “1, 2, 3, 4” -> Receive “3, 1, 4, 2”
  - Damaged Frames:
    - Send “1, 2, 3, 4” -> Receive “1, 8, 3, 4”

- In data link layer, **acknowledgement** (ACK) and **retransmission** are used to achieve reliable communication

ACK & Retransmission

- ACK:
  - Sender always has a copy in its buffer when sending each frame
  - Receiver sends ACK or negatively ACK (NAK) for its correctly /incorrectly-received frame, respectively. This ACK does not have to be an independent frame, it could also be along with a regular data frame, which is call **piggybacking**.
  - When sender receives an ACK, it removes the corresponding frame copy from its buffer and ready for the next transmission.
- Retransmission
  - Sender sets a **timer**, when timeout occurs and still no ACK received, the sender will schedule for retransmission.
  - When the sender receives a NAK, it also retransmits corresponding frames

Flow Control

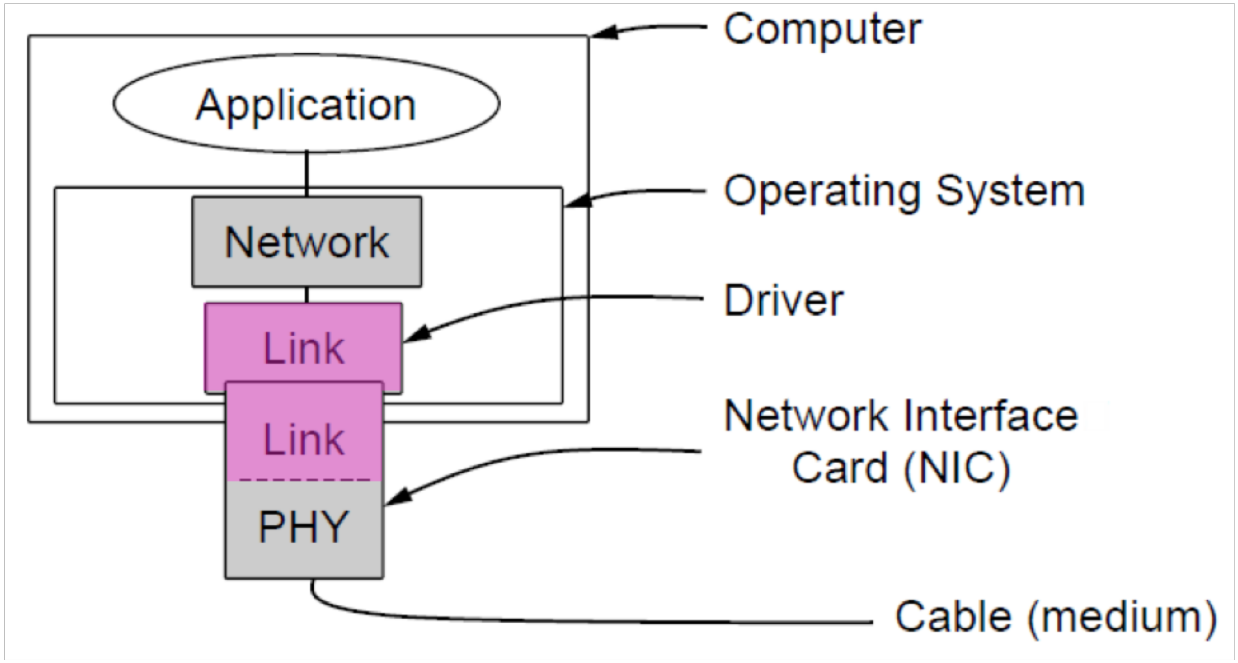
- When the **sending rate** is beyond the **receiving capability**, the frames (even received error-free) on the receiver side will have to be dropped as the receiver buffer is already full.
- Flow control specifies how **receiver** controls the **sender's** transmitting rate so that the receiver will have enough buffer space to receive (process).
  - This control can be explicit (sending ACK) or implicit (timeout)
- Since flow control provides the receiver's *acknowledgement* of correctly-received frames, it is **closely linked to error control**
  - Error control allows the receiver to tell the sender about frames damaged or lost during transmission, and coordinates the *re-transmission* of those frames by the sender
  - Thus, any flow control scheme combined with error control is called **Automatic Repeat Request (ARQ)**

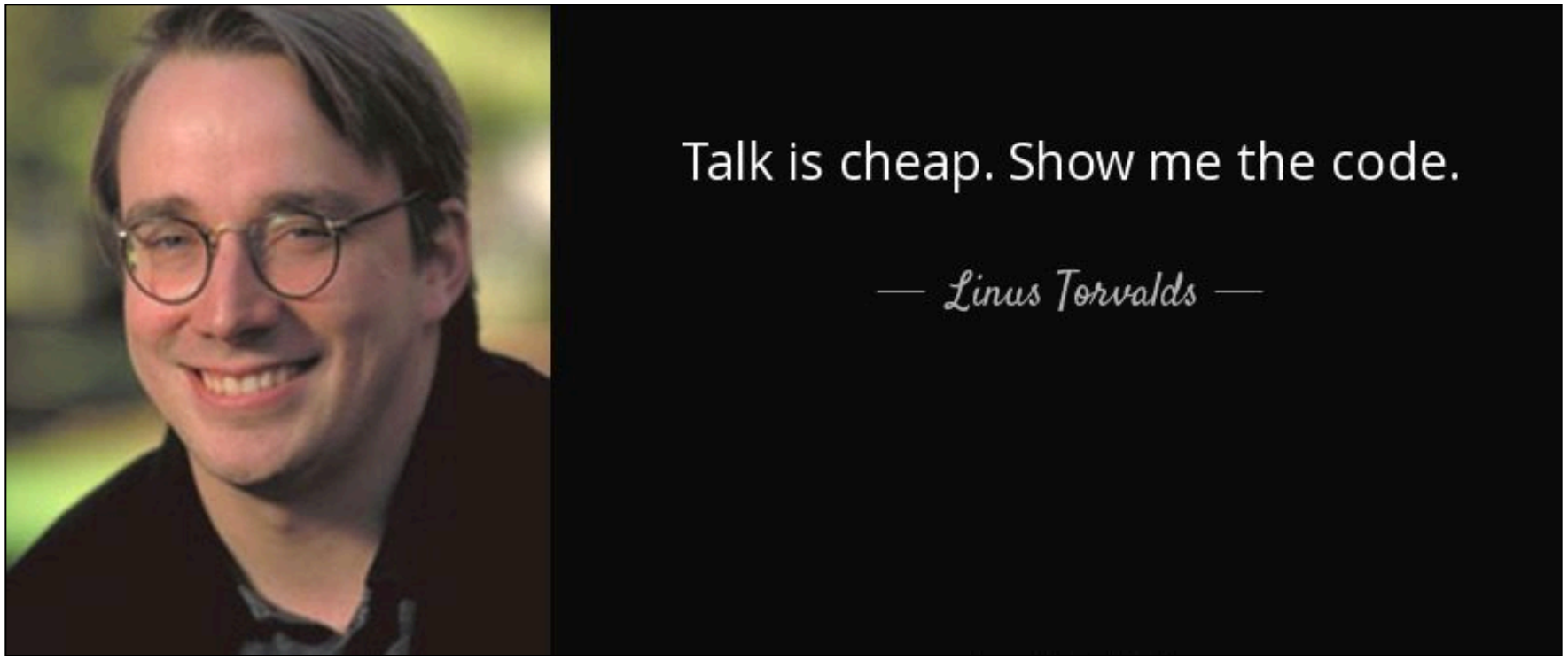
Automatic Repeat Request (ARQ)

- ***Stop-and-wait ARQ*** – extension of Stop-and-wait flow control
- Sliding window ARQ – extension of sliding window flow control:
  - ***Go-back-n ARQ*** – Receiver must get Frames in correct order
  - ***Selective repeat ARQ*** – correctly-received out-of-order Frames are stored at Receiver until they can be re-assembled into correct order

Protocols' Implementation

- For protocols on data link layer, commonly implemented as NICs and OS drivers
- For protocols on network layer (IP) is often OS software




Talk is cheap. Show me the code.
— Linus Torvalds —

Branch: master ▾ linux / include / net /

Create new file | Find file | History

koverstreet and torvalds sctp: convert to genradix ··· Latest commit 2075e50 20 hours ago

..

9p	9p: Add refcount to p9_req_t	6 months ago
bluetooth	Bluetooth: Add quirk for reading BD_ADDR from fwnode property	15 days ago
caif	caif: reduce stack size with KASAN	a year ago
iucv	net/af_iucv: locate IUCV header via skb_network_header()	6 months ago
netfilter	netfilter: nf_tables: nat: merge nft_masq protocol specific modules	12 days ago
netns	ipv6: icmp: use percpu allocation	16 days ago
nfc	NFC: Fix the number of pipes	6 months ago
phonet	phonet: fix building with clang	19 days ago
sctp	sctp: convert to genradix	10 hours ago
tc_act	net: Change TCA_ACT_* to TCA_ID_* to match that of TCA_ID_POLICE	a month ago
6lowpan.h	6lowpan: Fix IID format for Bluetooth	2 years ago
Space.h	net/mac89x0: Convert to platform_driver	a year ago
act_api.h	net: Change TCA_ACT_* to TCA_ID_* to match that of TCA_ID_POLICE	a month ago
addrconf.h	ipv6_stub: add ipv6_route_input stub/proxy.	27 days ago

Library Functions in Simulator

- Check Linux Source Code if you are interested, but its too complicated.
- In this course, we use a simulated implementation.
- There is a public implementation, although not official, to refer:
  - https://github.com/joskuijpers/ti2406pa/blob/master/protocols/protocol.h

Library Functions in Simulator

Group	Library Function	Description
Network layer	from_network_layer(&packet) to_network_layer(&packet) enable_network_layer() disable_network_layer()	Take a packet from network layer to send Deliver a received packet to network layer Let network cause “ready” events Prevent network “ready” events
Physical layer	from_physical_layer(&frame) to_physical_layer(&frame)	Get an incoming frame from physical layer Pass an outgoing frame to physical layer
Events & timers	wait_for_event(&event) start_timer(seq_nr) stop_timer(seq_nr) start_ack_timer() stop_ack_timer()	Wait for a packet / frame / timer event Start a countdown timer running Stop a countdown timer from running Start the ACK countdown timer Stop the ACK countdown timer

Utopian Simplex Protocol

- An optimistic protocol (p1) to get us started
  - Assumes no errors, and receiver as fast as sender
  - Considers one-way data transfer
  - That's it, no error or flow control …
  - This is essentially what **Ethernet** does – just blast packets and receive as quickly as possible

```
void sender1(void)
{
  frame s;
  packet buffer;

  while (true) {
       from_network_layer(&buffer);
       s.info = buffer;
       to_physical_layer(&s);
  }
```

Sender loops blasting frames

```
void receiver1(void)
{
  frame r;
  event_type event;

  while (true) {
       wait_for_event(&event);
       from_physical_layer(&r);
       to_network_layer(&r.info);
  }
}
```

Receiver loops eating frames

Stop-and-Wait: Error-free channel

- Protocol (p2) ensures sender can't outpace receiver:
  - Receiver returns a dummy frame (ack) when ready
  - Only one frame out at a time – called stop-and-wait
  - We added flow control!

```
void sender2(void)
{
  frame s;
  packet buffer;
  event_type event;

  while (true) {
       from_network_layer(&buffer);
       s.info = buffer;
       to_physical_layer(&s);
       wait_for_event(&event);
  }
}
```

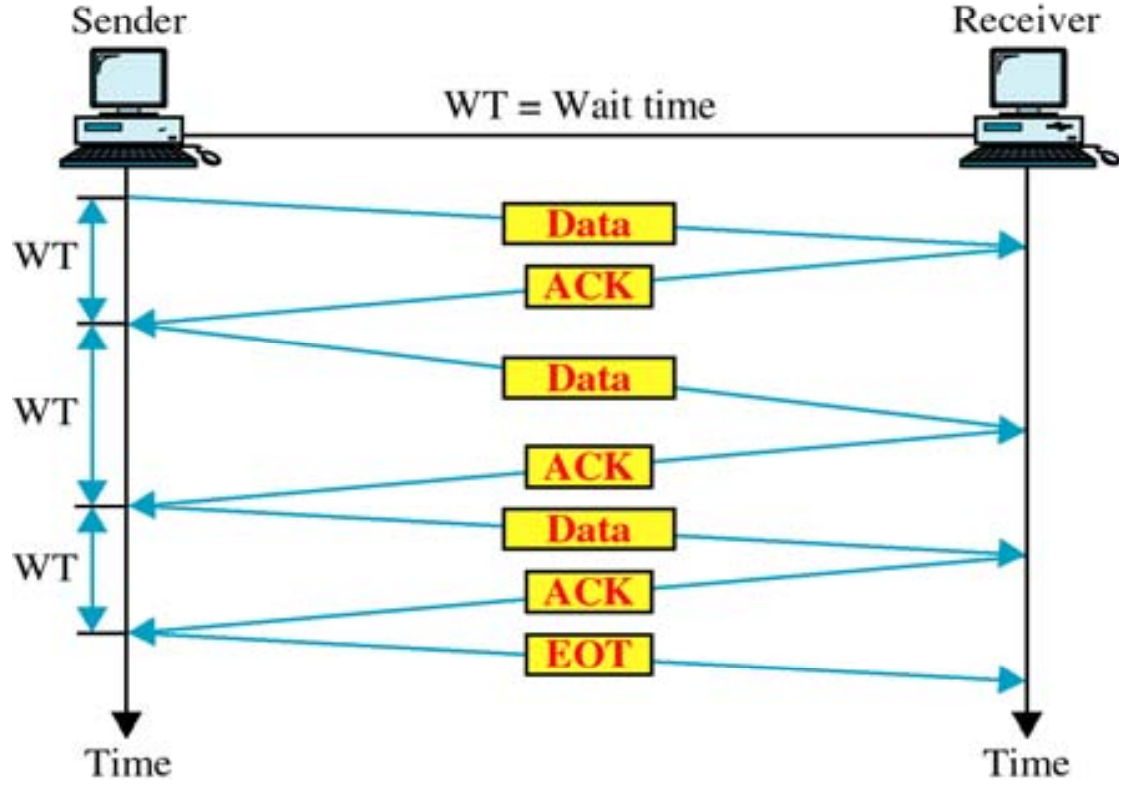
Sender **waits** to for ack after passing frame to physical layer

```
void receiver2(void)
{
  frame r, s;
  event_type event;
  while (true) {
       wait_for_event(&event);
       from_physical_layer(&r);
       to_network_layer(&r.info);
       to_physical_layer(&s);
  }
}
```

Receiver sends ack after passing frame to network layer

Stop-and-Wait: Error-free channel

- The sender **waits** for ACK after **each frame** is transmitted. It is very simple but also inefficient.



Stop-and-Wait: Noisy channel

- ARQ (Automatic Repeat reQuest) **adds error control**
  - Receiver acks frames that are correctly delivered
  - Sender sets timer and resends frame if no ack

- For correctness, frames and acks **must be numbered**
  - Else receiver can't tell retransmission (due to lost ack or early timer) from new frame (i.e. new ? old?)
  - For stop-and-wait, 2 numbers (1 bit) are sufficient

Stop-and-Wait: Noisy channel

- **Sender** loop (p3):
  - Used in 802.11 in principle
  - but 802.11 uses a larger sequence number (12 bits) and gives up after a limited number of retransmissions (retries)

Send frame (or retransmission)
Set timer for retransmission
Wait for ack or timeout

If a good ack then set up for the next frame to send (else the old frame will be retransmitted)

```
void sender3(void) {
  seq_nr next_frame_to_send;
  frame s;
  packet buffer;
  event_type event;

  next_frame_to_send = 0;
  from_network_layer(&buffer);
  while (true) {
      s.info = buffer;
      s.seq = next_frame_to_send;
      to_physical_layer(&s);
      start_timer(s.seq);
      wait_for_event(&event);
      if (event == frame_arrival) {
          from_physical_layer(&s);
          if (s.ack == next_frame_to_send) {
              stop_timer(s.ack);
              from_network_layer(&buffer);
              inc(next_frame_to_send);
          }
      }
  }
}
```

Increment “next_frame_to_send”

Stop-and-Wait: Noisy channel

- **Receiver** loop (p3):

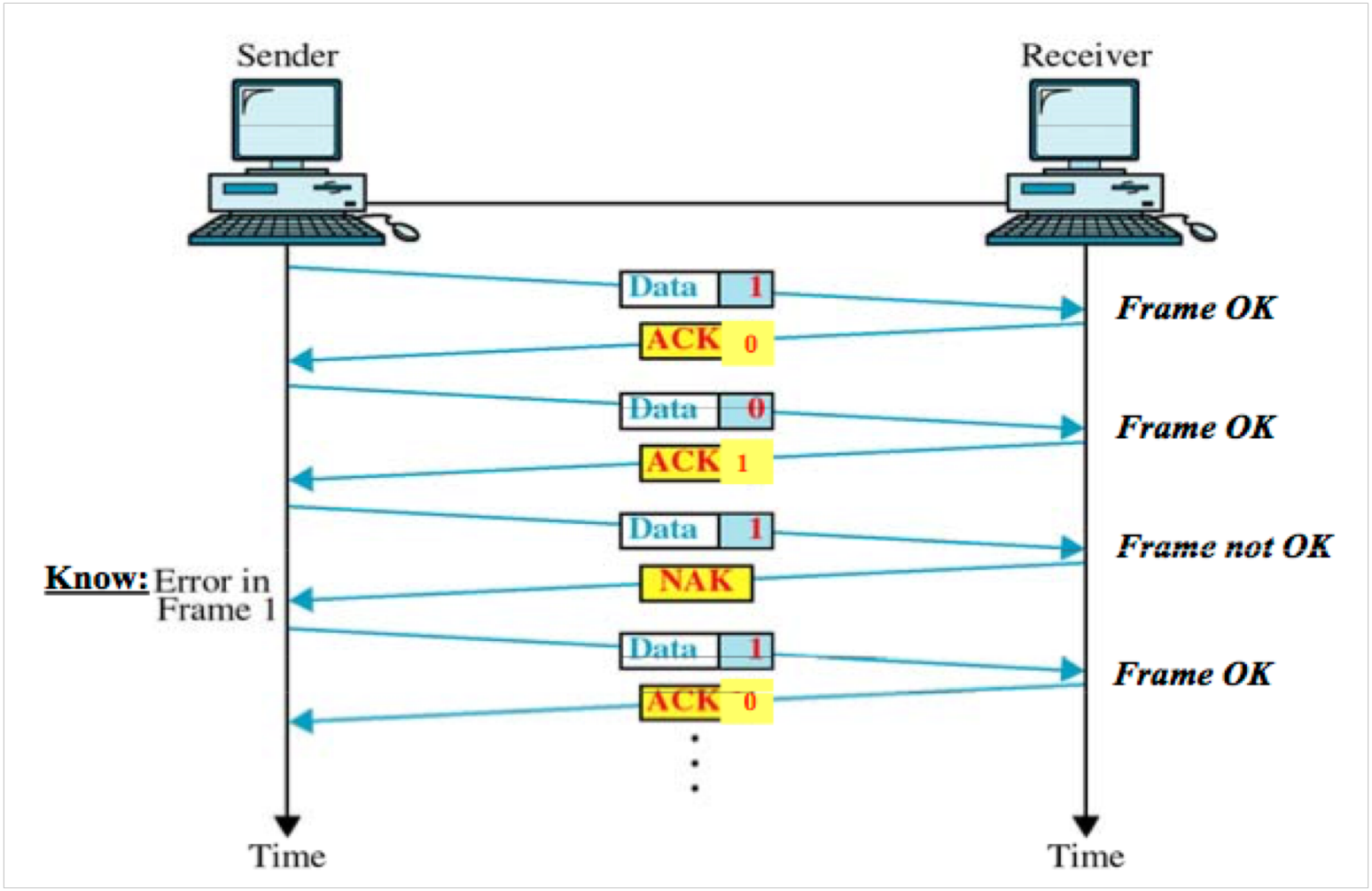
```
void receiver3(void)
{
  seq_nr frame_expected;
  frame r, s;
  event_type event;

  frame_expected = 0;
  while (true) {
      wait_for_event(&event);
      if (event == frame_arrival) {
          from_physical_layer(&r);
          if (r.seq == frame_expected) {
              to_network_layer(&r.info);
              inc(frame_expected);
          }
          s.ack = 1 – frame_expected;
          to_physical_layer(&s);
      }
  }
}
```

Wait for a frame

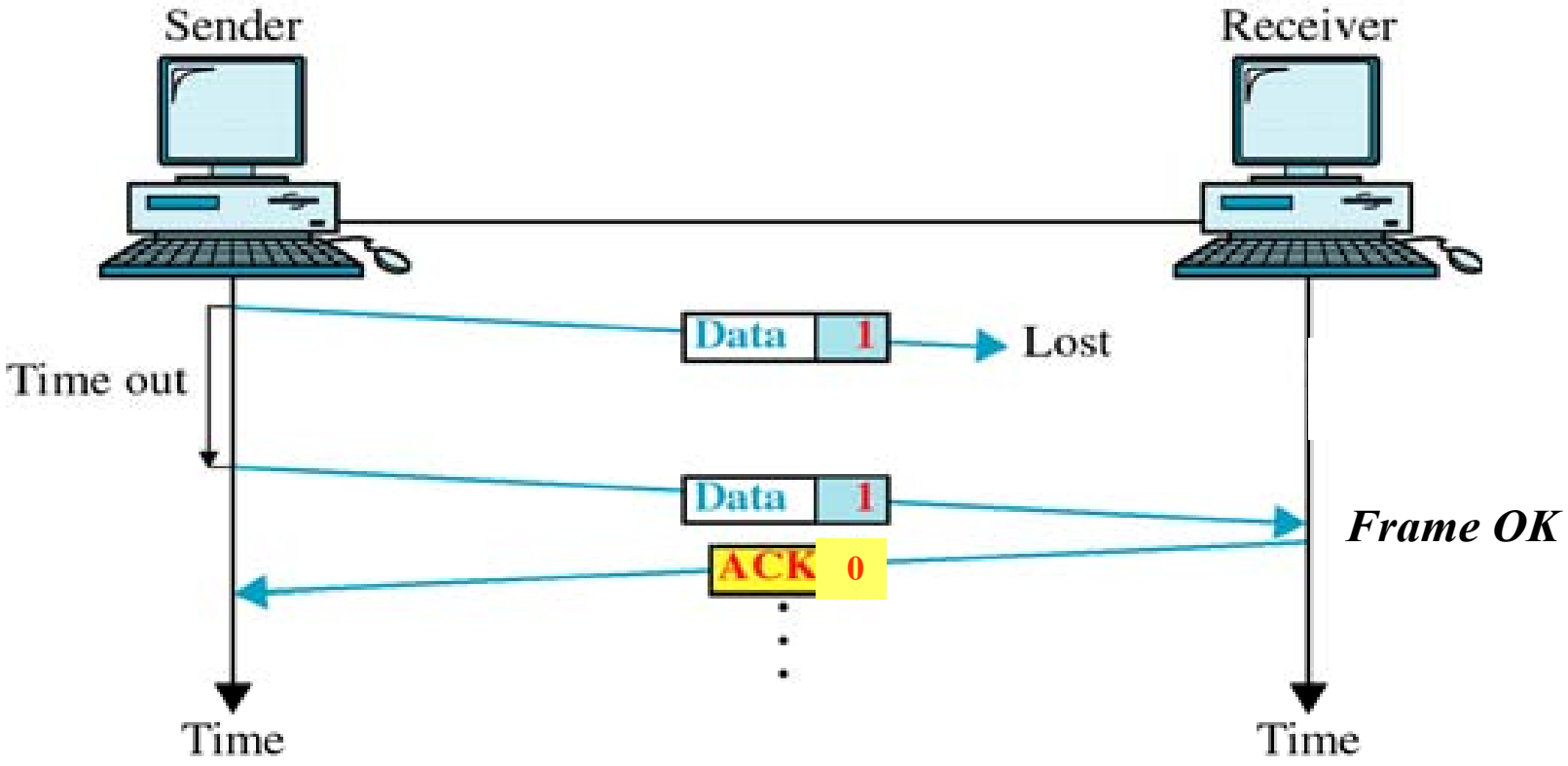
If it's new then take it and advance expected frame

Ack current frame

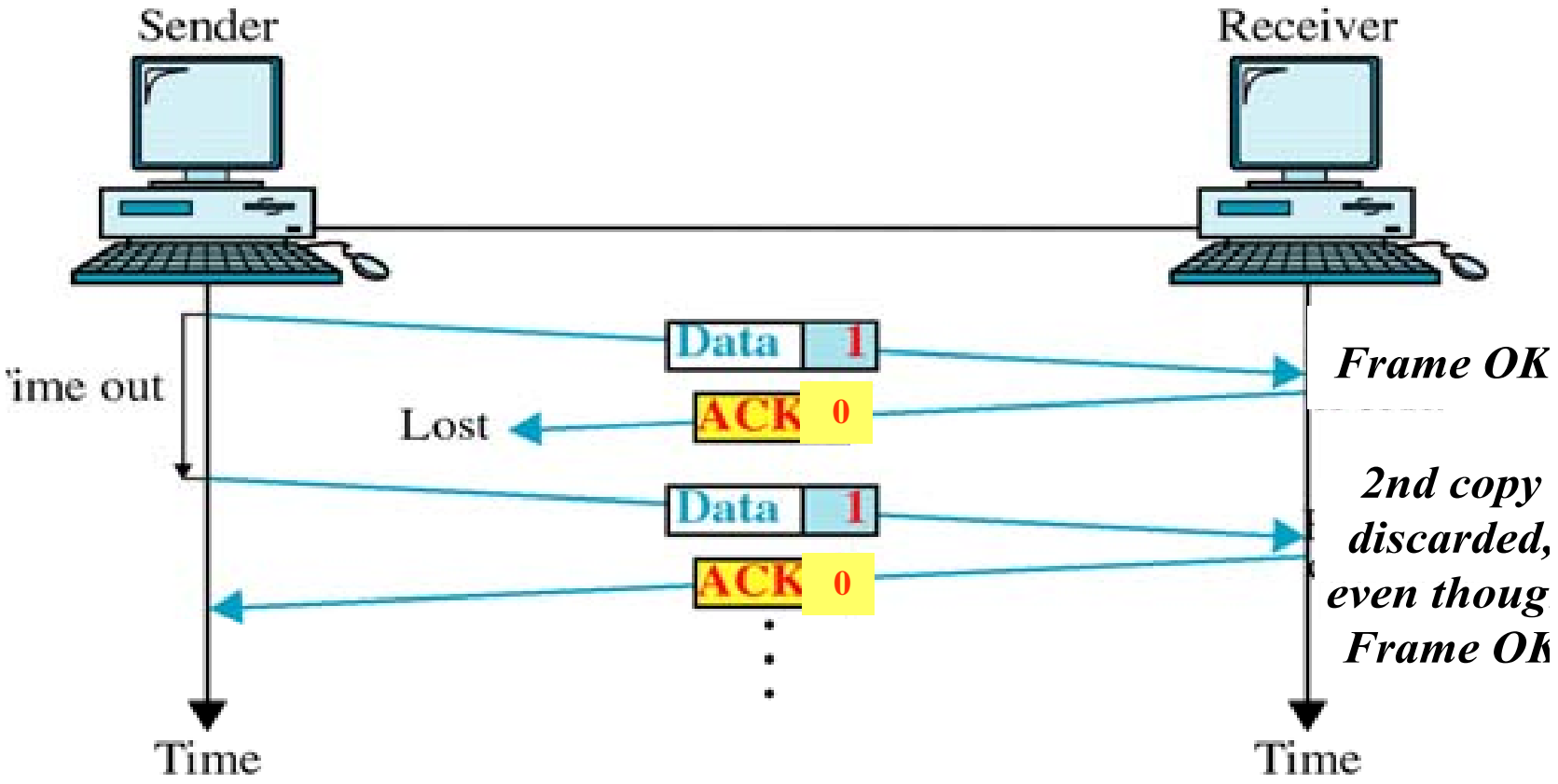
Stop-and-Wait ARQ: Damaged Frame



Stop-and-Wait ARQ Lost Frame



Stop-and-Wait ARQ Lost ACK

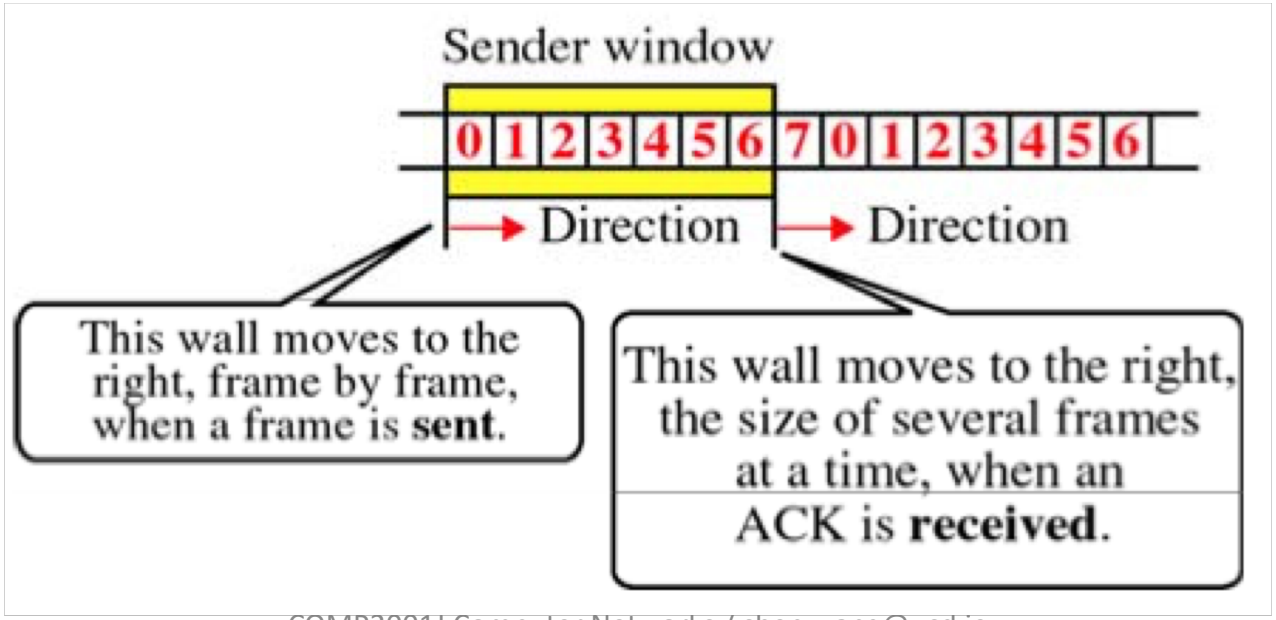


Sliding Window Protocols

- Sliding Window Concept
- One-bit Sliding Window
- Go-Back-N
- Selective Repeat

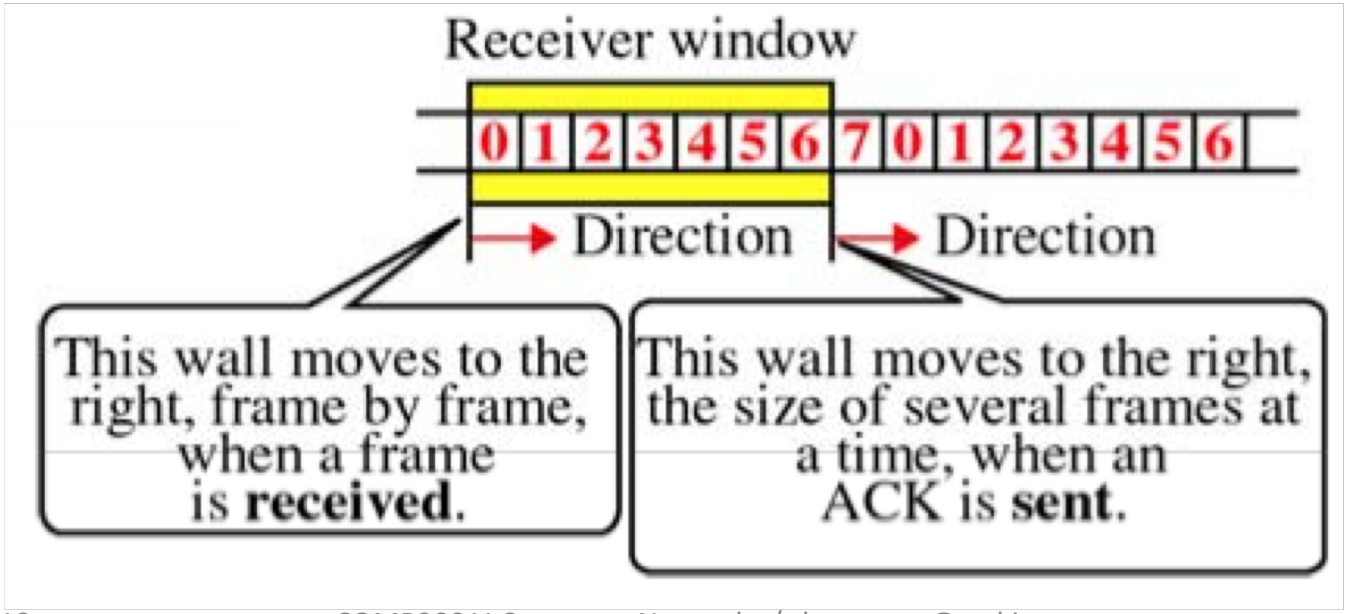
Sliding Window Concept

- **Sender** maintains window of continuously numbered frames it can send
  - Needs to buffer them for possible retransmission
  - Window advances with next acknowledgements
  - Window size means the maximum number of frames it can send without receiving any corresponding ACK



Sliding Window Concept

- **Receiver** maintains window of continuously numbered frames it can receive
  - Needs to keep buffer space for arrivals
  - Frames can only be received, if their sequence numbers are within the window range.
  - Window advances with **in-order** arrivals

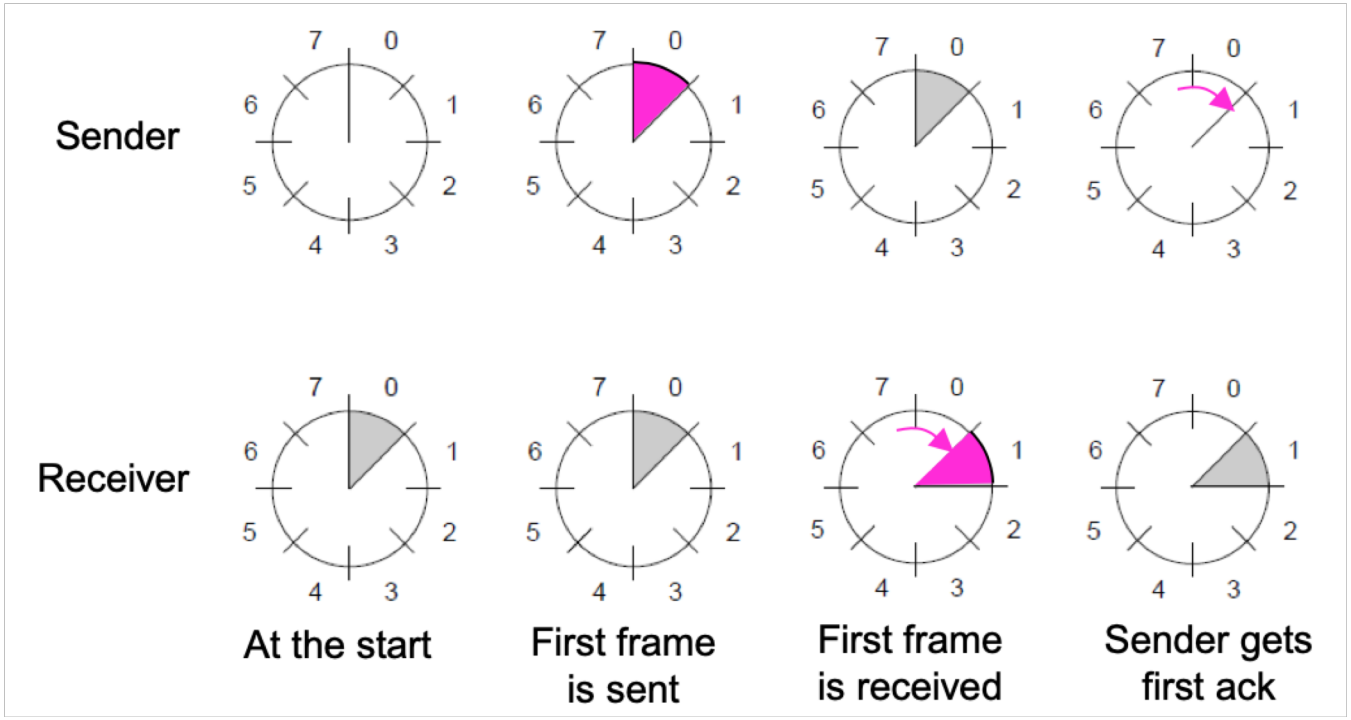


Sliding Window Concept

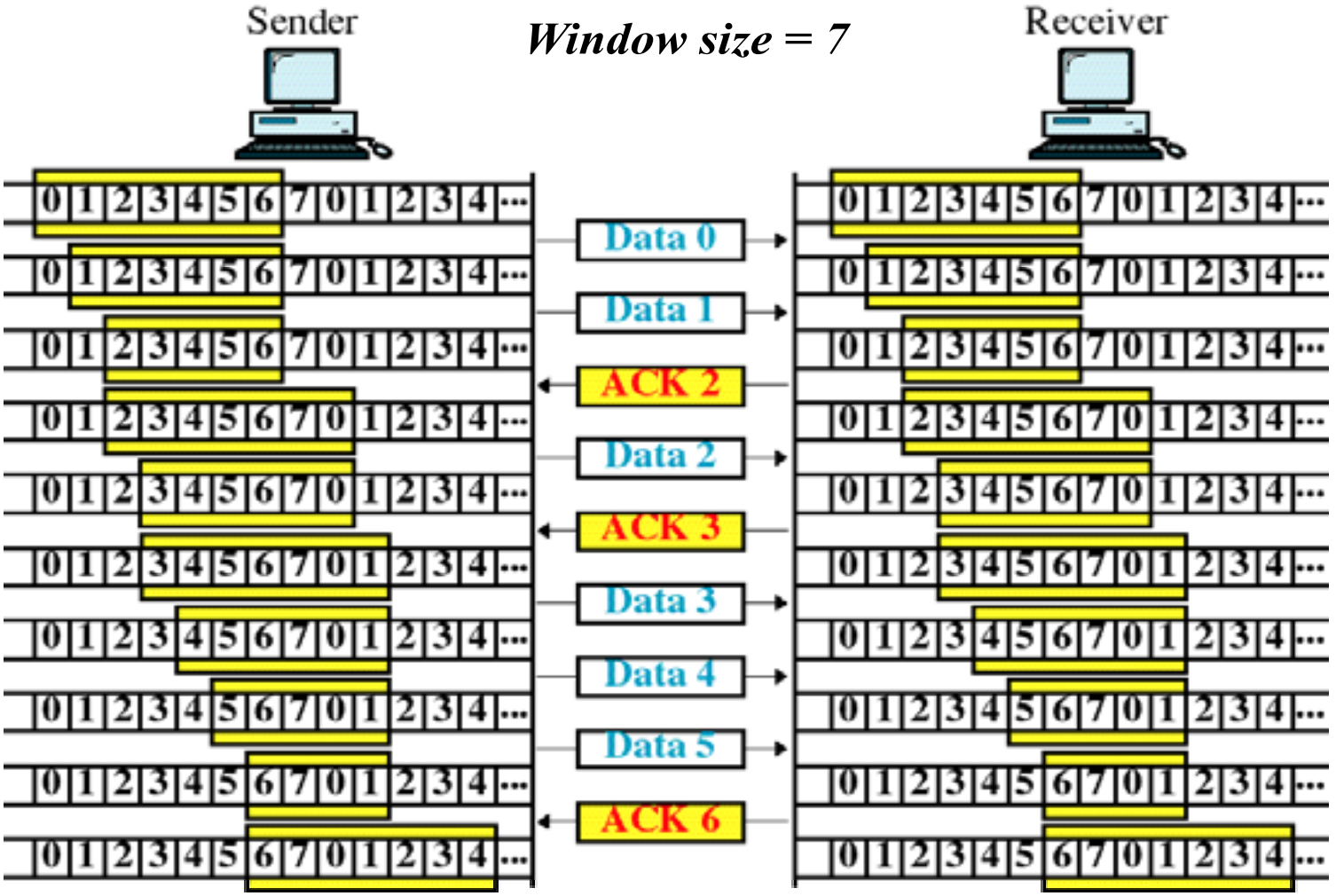
- The window size of both sender $\boldsymbol{w_s}$ and receiver $\boldsymbol{w_r}$ is **fixed** over the communication duration.
  - Obviously, we have $w_s \geq w_r$
- When $w_r > 1$, **Cumulative ack** is applied:
  - Instead of replying ACK for every received frame, the receiver only replies **one single** ACK after continuously received multiple frames
  - For sender, it means all frames that numbered up to the ACK number, are successfully received.

Sliding Window Concept

- A sliding window advancing at the sender and receiver
  - Ex: $w_s = w_r = 1$ (stop-and-wait), with a 3-bit sequence number.



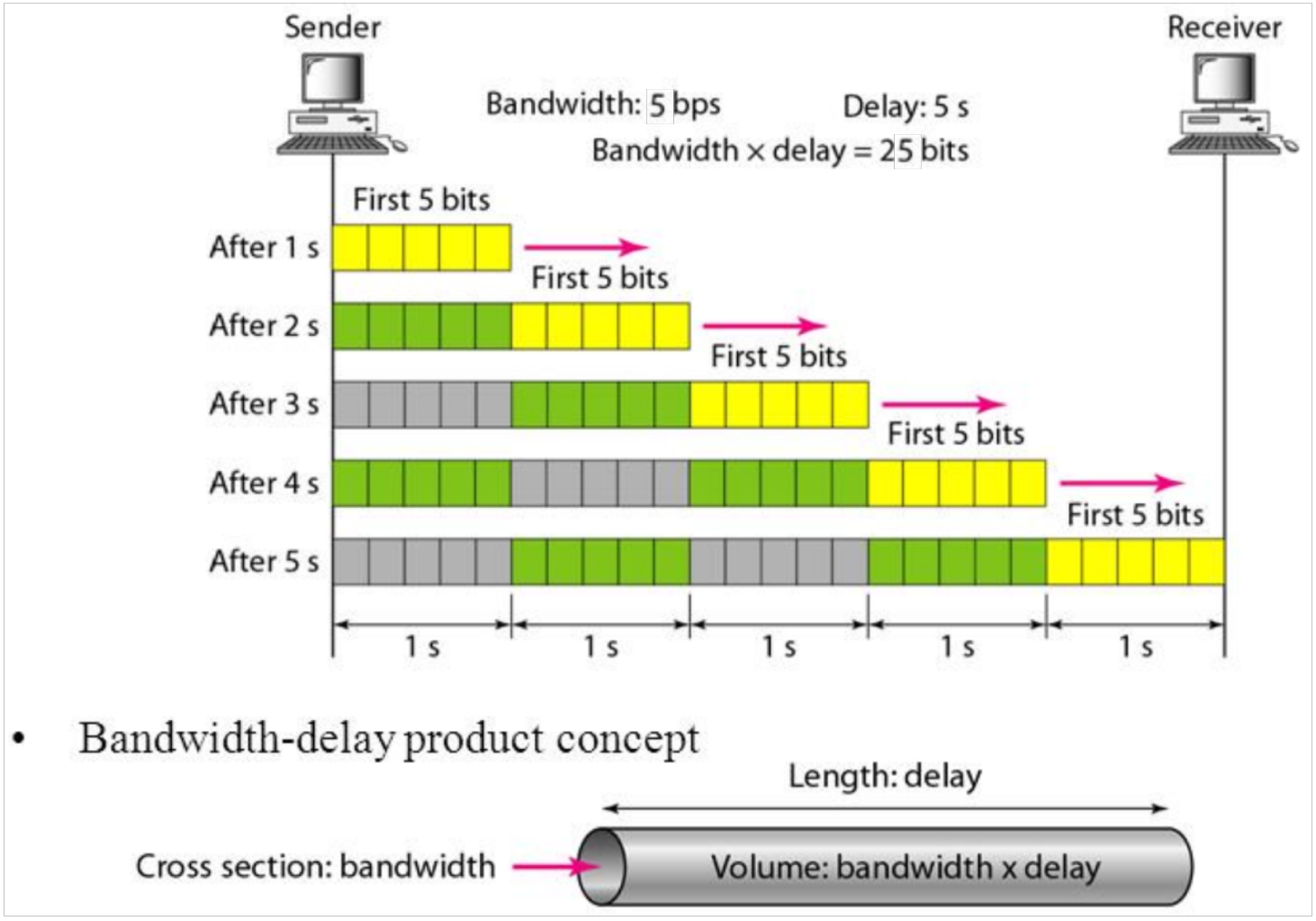
Sliding Window Flow Control Example



Sender window size w_s?

- How do we set the sender's window size w_s?
- An equivalent question is, what is the maximum number of frames a sender can transmit without receiving any corresponding ACK?
- The sender wants to "Fill a pipe" full of frames, before it expects an ACK form the receiver!

Bandwidth-Delay Product



- Bandwidth-delay product concept

Sliding Window Concept

- Larger windows enable pipelining for efficient link use
  - Pipelining means that there are multiple frames **outstanding (sent but not yet ACKed)** in the network at any instant.
  - Stop-and-wait ($w_s = 1$) is inefficient for long links
  - Best window (w_s) depends on bandwidth-delay (BD)
  - Want w_s ≥ 2BD+1 to ensure high link utilization
- Pipelining leads to different choices for errors/buffering
  - We will consider Go-Back-N and Selective Repeat

One-Bit Sliding Window

- Sender window size $w_s = 1$; Receiver window size $w_r = 1$
- Transfers data in both directions with stop-and-wait
  - Piggybacks acks on reverse data frames for efficiency
  - Handles transmission errors, flow control, early timers

Each node is **sender and receiver** (p4):

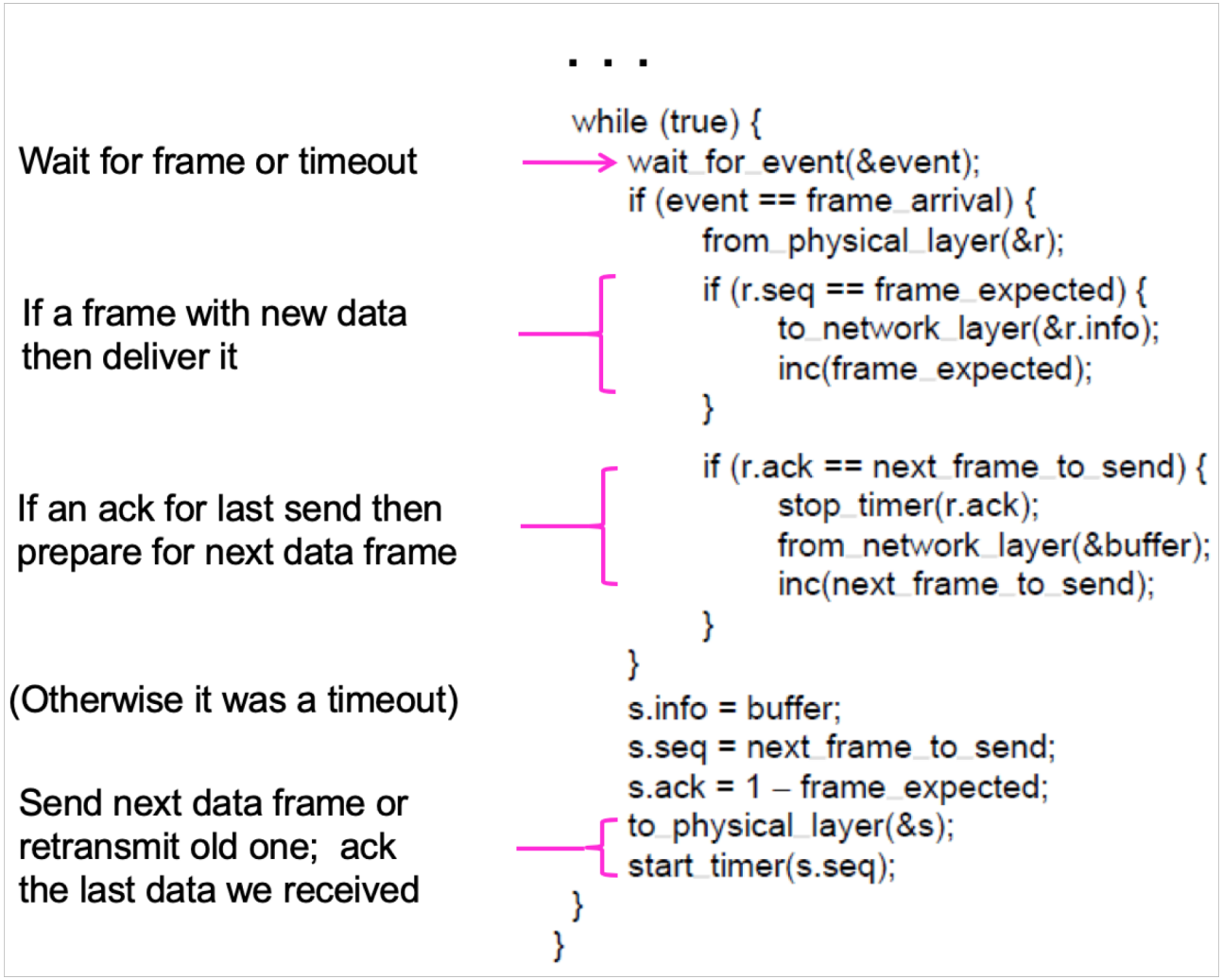
```
void protocol4 (void) {
  seq_nr next_frame_to_send;
  seq_nr frame_expected;
  frame r, s;
  packet buffer;
  event_type event;
  next_frame_to_send = 0;
  frame_expected = 0;
  from_network_layer(&buffer);
  s.info = buffer;
  s.seq = next_frame_to_send;
  s.ack = 1 – frame_expected;
  to_physical_layer(&s);
  start_timer(s.seq);
```

Prepare first frame

Launch it, and set timer

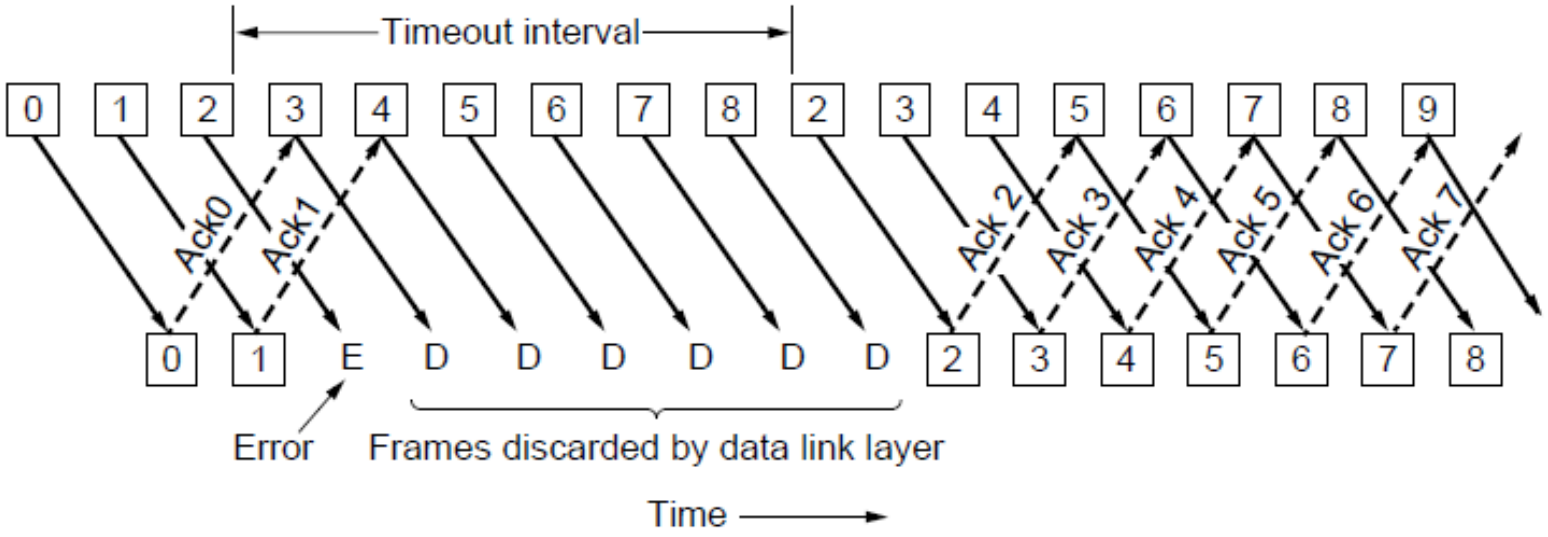
. . .

One-Bit Sliding Window

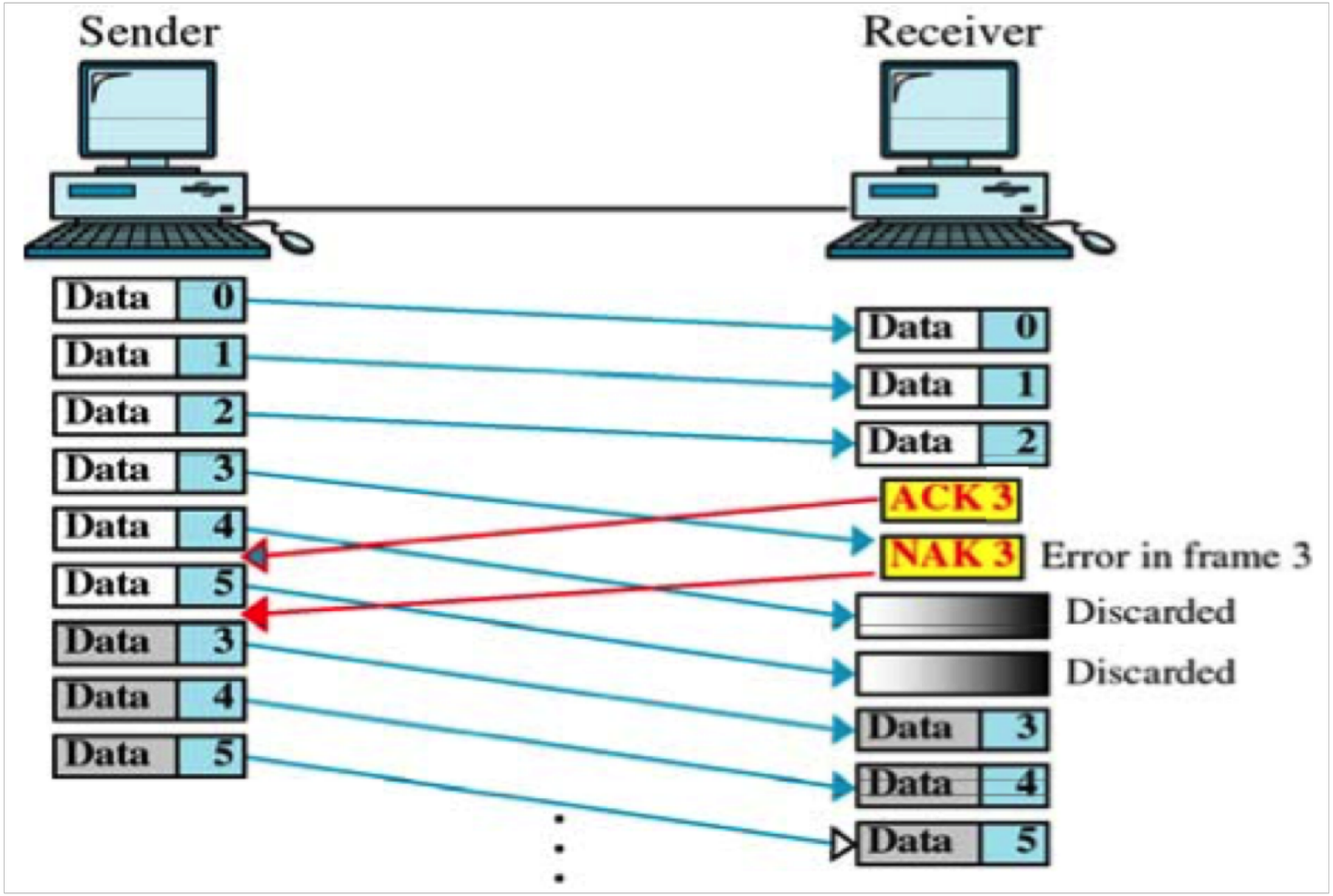


Go-Back-N

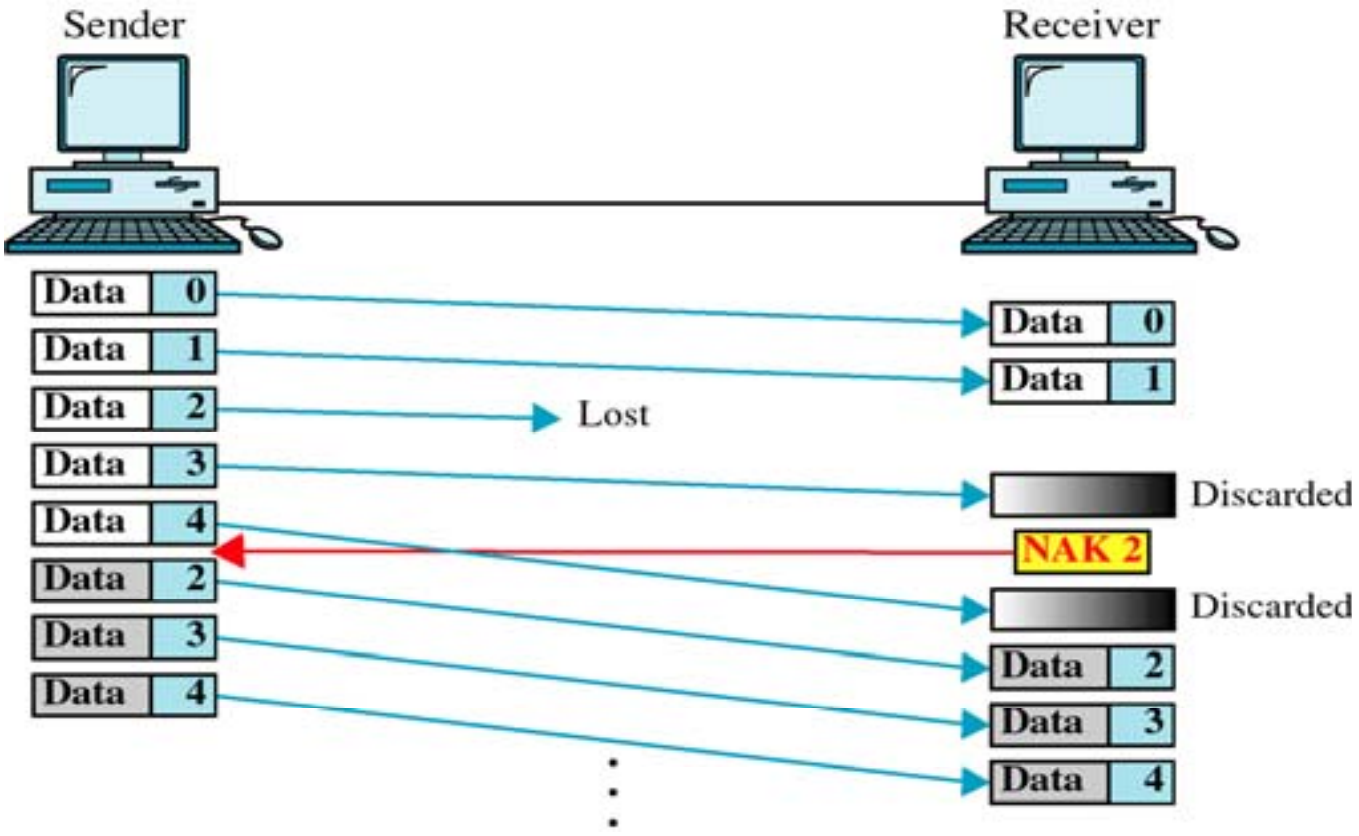
- Sender $w_s > 1$; Receiver $w_r = 1$
- Receiver only accepts/acks frames that arrive **in order**:
  - Discards frames that follow a missing/errored frame
  - Sender times out and resends all outstanding frames



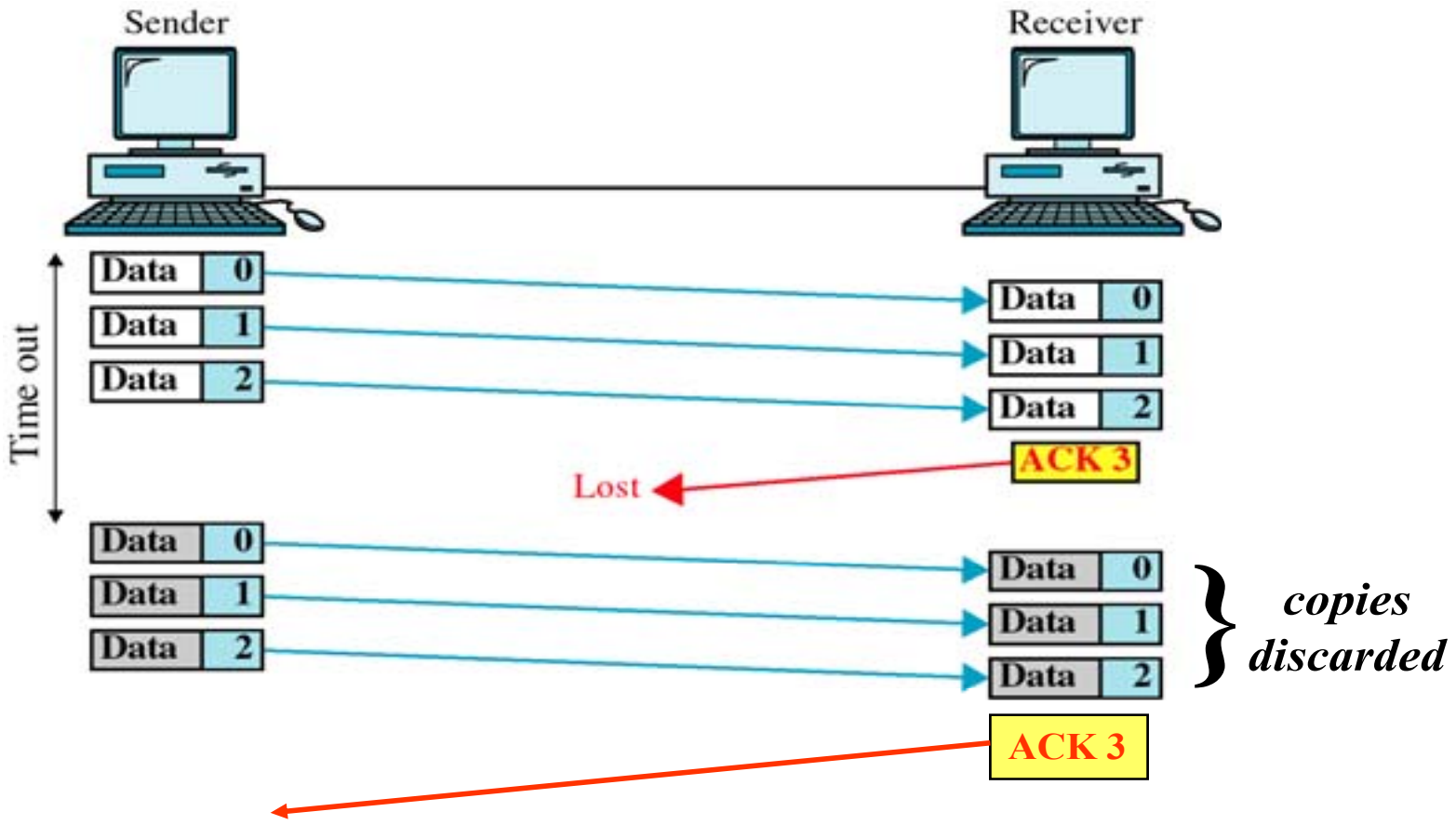
Go-back-N ARQ Damaged Frame



Go-back-N ARQ Lost Frame



Go-back-N ARQ Lost ACK

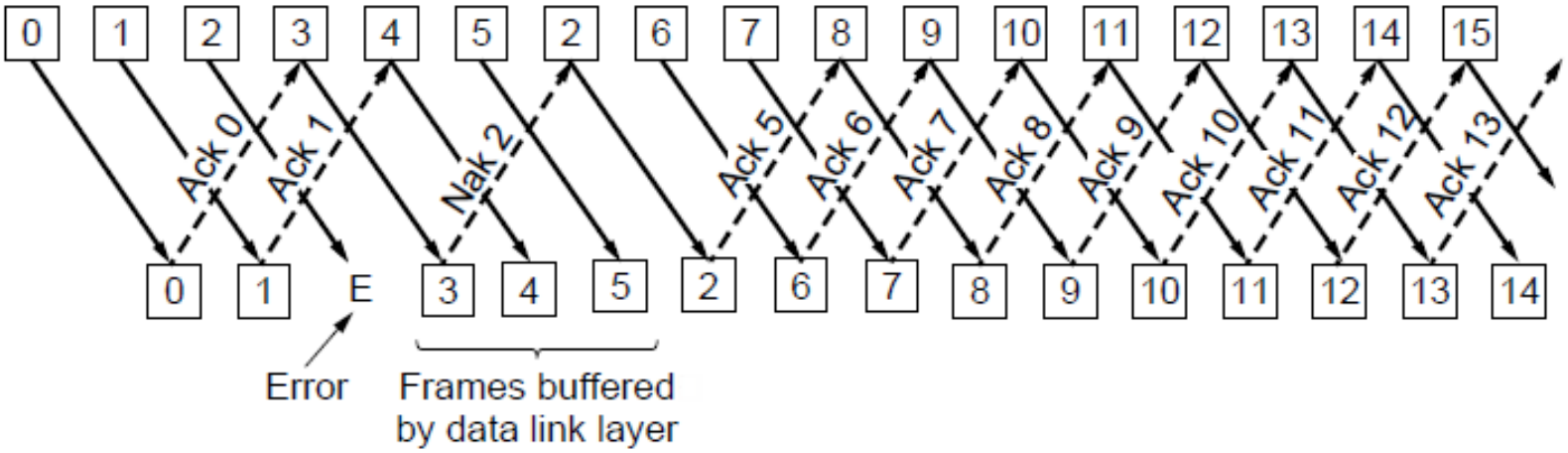


Go-Back-N: Tradeoff

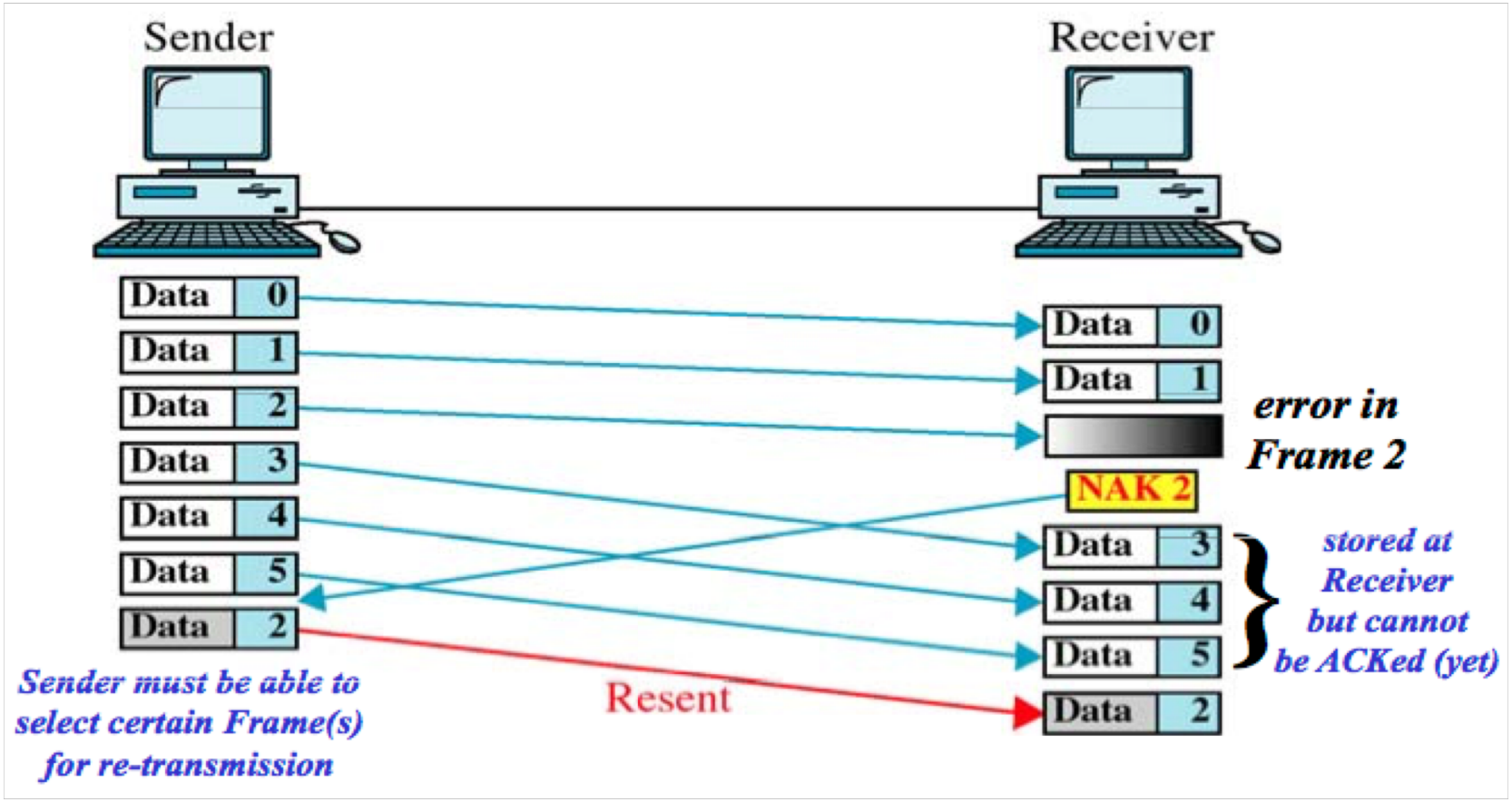
- Pros:
  - Can transmit multiple/N frames continuously without having to wait ACK one after another
  - Simple strategy for receiver
  - Needs only 1 frame for w_r
- Cons:
  - Wastes link bandwidth for errors with large sender windows
  - Entire window might be retransmitted, if only one frame not received correctly

Selective Repeat

- Sender $w_s > 1$; Receiver $w_r > 1$
- Receiver accepts frames anywhere in receive window
  - Cumulative ack indicates highest in-order frame
  - NAK (negative ack) causes sender retransmission of a missing frame before a timeout resends window



Selective Repeat ARQ Damaged Frame



Selective Repeat:

- Tradeoff :
  - More efficient use of link bandwidth as only lost frames are resent (with low error rates)
  - More complex than Go-Back-N due to buffering at receiver and multiple timers at sender
- What happens if the NAK is lost, or if multiple data frames are lost per round-trip time?
  - The retransmission timer goes off and transmission restarts from the acknowledgement point.
  - To be more tolerant of these situations more acknowledgement information needs to be returned (more NAKs, a bit vector of the frames received beyond the cumulative ack, etc.)

Selective Repeat

- We have $w_s \geq w_r > 1$
- Assume we use $\boldsymbol{n}$ bits numbering frames
- To ensure that when ACK is lost, the receiver still can tell the new/old frame
- This has to be satisfied:
  - $\boldsymbol{w_s + w_r \leq 2^n}$

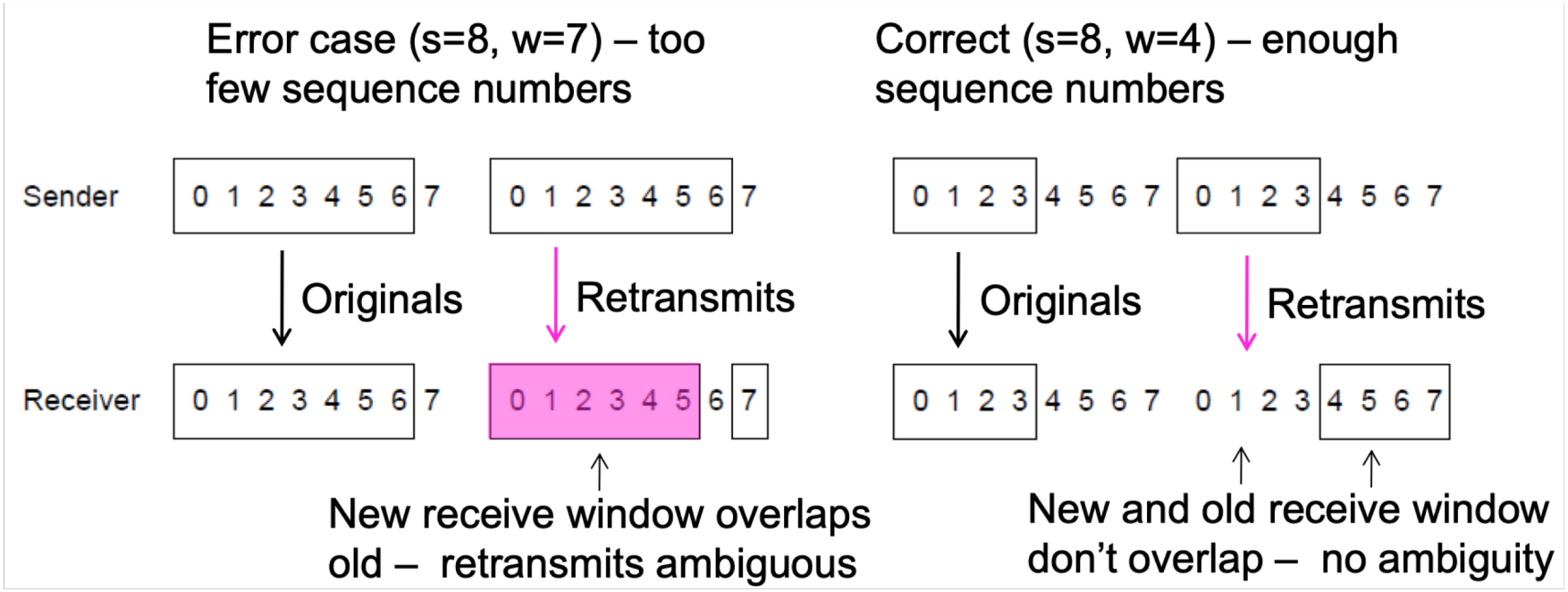
Example

- For a data link layer protocol, it uses selective repeat, if 5 bits are used for numbering frame sequence, what is the maximum receiving window size?

- Since $w_s \geq w_r > 1$, and $w_s + w_r \leq 2^n$
- $w_r \leq 2^{n-1} = 2^{5-1} = 16$ frames

Selective Repeat

- Thus, sequence numbers ($s=2^n$) at least twice the window ($w=w_r=w_s$)



Link Efficiency

- We know that $w_s \geq 2BD + 1$ can ensure high link utilization (100%)
  - "1" here means 1 frame size
- When $w_s < 2BD + 1$, the link utilization (efficiency) is below 100%
- Denote a delivery period as T
  - it refers to the duration that starts from when the sender transmits the frame, to the moment when the sender receives the first ACK.

Link Efficiency

- Denotes RTT as the round trip time for a bit, normally it is the double amount of propagation time T_{prop}: $RTT = 2T_{prop}$, also denotes the transmission time for an ACK frame is T_{ack},
  - $T = T_f + RTT + T_{ack}$
- Assume sender's transmission rate is C bits/sec, if during T, there are L bits sent:
  - If we know a frame is l bits: $L = l * w_s$
  - the transmission time: $T_f = {}^{L}/_{C}$
  - then the **link efficiency**: $\boldsymbol{{}^{T_f}/_{T}}$
  - Link throughput = link efficiency * sending rate

Example

- Host A sending data to host B using stop-and-wait. The sending rate is 3kb/s, the propagation delay is 200ms, ignore the transmission delay for an ACK frame, when link efficiency is 40%, what is the length of the frame?

  - $T_f = x/(3kb/s)$
  - $40\% = T_f/(T_f + 2 * 200ms)$
  - x=800bits