Tannenbaum's Protocols as explained in the 5^{th} Ed

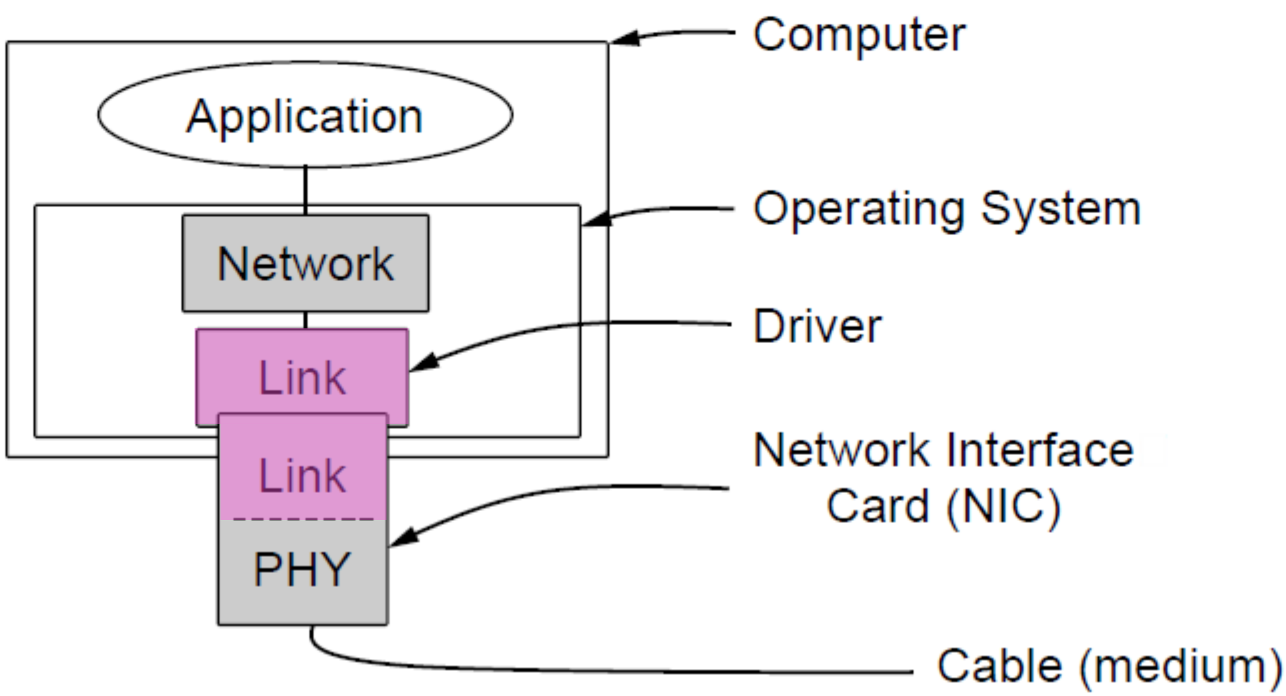
2011

Elementary Data Link Protocols

- Link layer environment »
- Utopian Simplex Protocol »
- Stop-and-Wait Protocol for Error-free channel »
- Stop-and-Wait Protocol for Noisy channel »

Link layer environment (1)

Commonly implemented as NICs and OS drivers; network layer (IP) is often OS software



Link layer environment (2)

Link layer protocol implementations use library functions

- See code (`protocol.h`) for more details

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

```
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

- That's it, no error or flow control …

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 – Noisy channel (1)

<u>ARQ</u> (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
- For stop-and-wait, 2 numbers (1 bit) are sufficient

Stop-and-Wait – Noisy channel (2)

Sender loop (p3):

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);
          }
      }
  }
}
```

Stop-and-Wait – Noisy channel (3)

Receiver loop (p3):

Wait for a frame

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

Ack current frame

```
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);
      }
  }
}
```

Sliding Window Protocols

- Sliding Window concept »
- One-bit Sliding Window »
- Go-Back-N »
- Selective Repeat »

Sliding Window concept (1)

Sender maintains window of frames it can send

- Needs to buffer them for possible retransmission
- Window advances with next acknowledgements

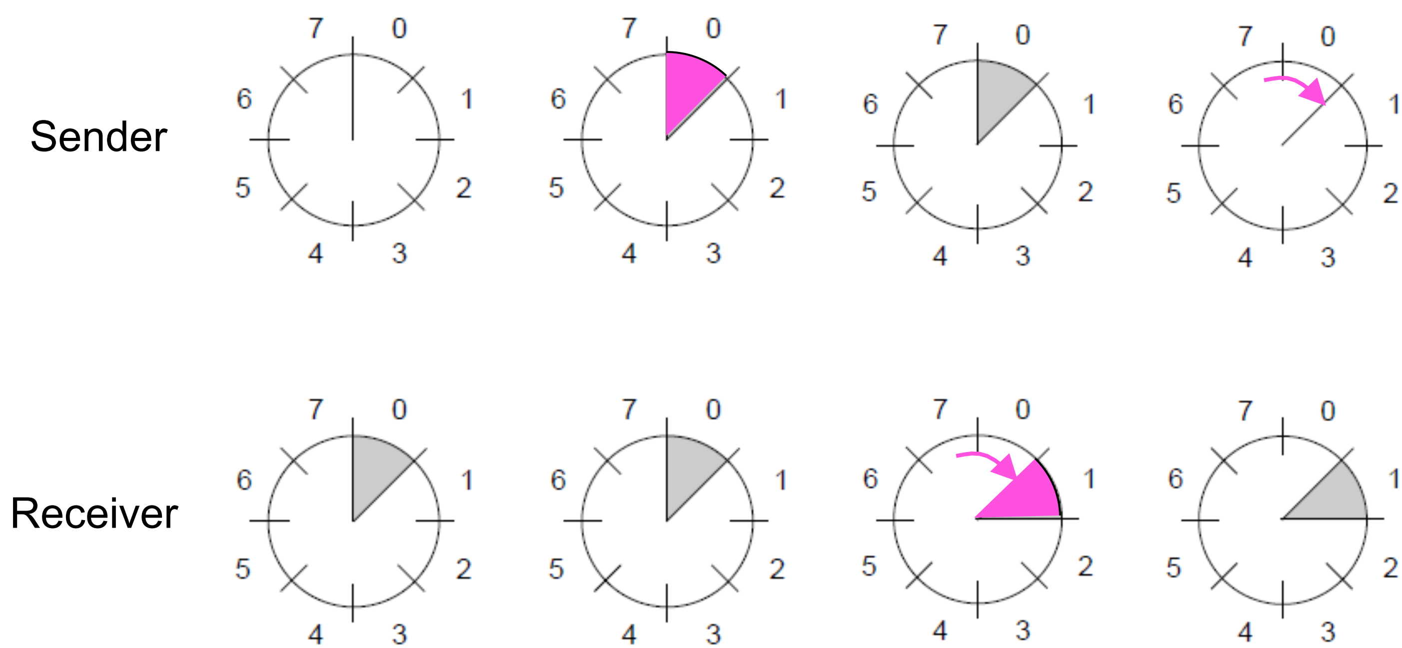
Receiver maintains window of frames it can receive

- Needs to keep buffer space for arrivals
- Window advances with in-order arrivals

Sliding Window concept (2)

A sliding window advancing at the sender and receiver

- Ex: window size is 1, with a 3-bit sequence number.



At the start | First frame is sent | First frame is received | Sender gets first ack

Sliding Window concept (3)

Larger windows enable pipelining for efficient link use

- Stop-and-wait ($w=1$) is inefficient for long links
- Best window (w) depends on bandwidth-delay (BD)
- Want $w \geq 2BD+1$ 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 (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):

Prepare first frame

Launch it, and set timer

```
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);

. . .
```

One-Bit Sliding Window (2)

Wait for frame or timeout

If a frame with new data then deliver it

If an ack for last send then prepare for next data frame

(Otherwise it was a timeout)

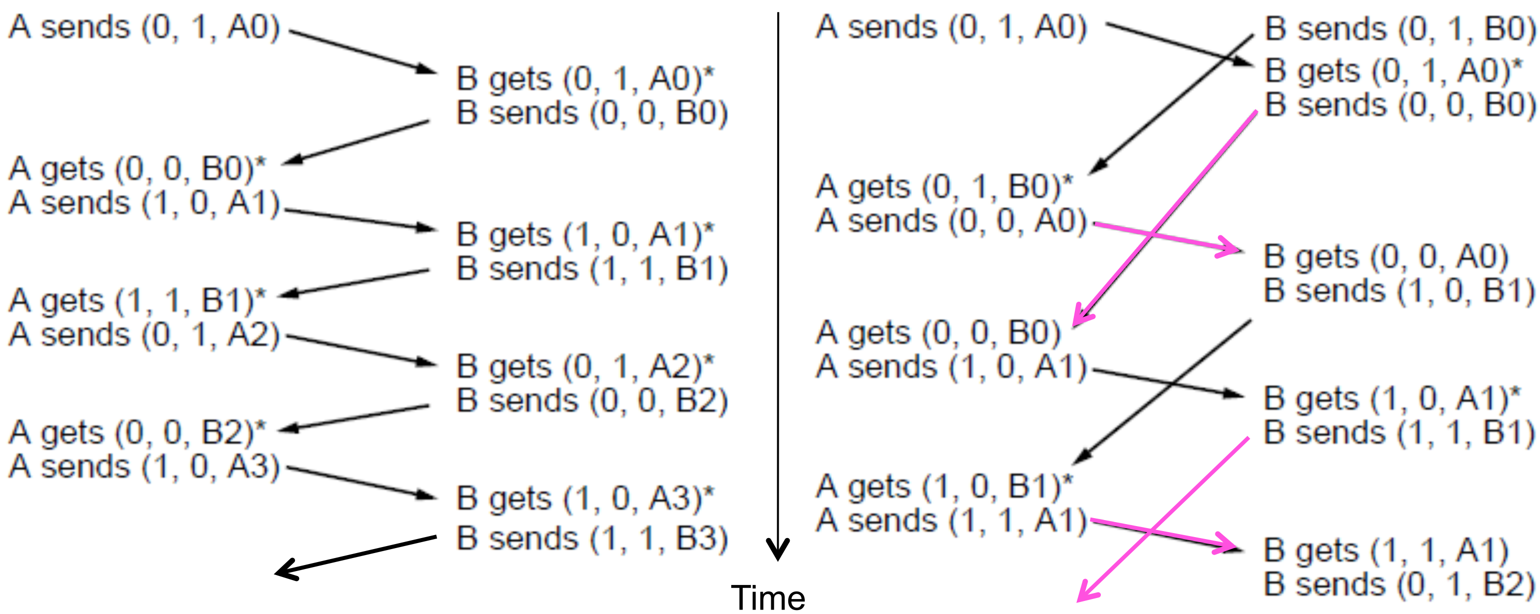
Send next data frame or retransmit old one; ack the last data we received

```
. . .
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);
        }
        if (r.ack == next_frame_to_send) {
            stop_timer(r.ack);
            from_network_layer(&buffer);
            inc(next_frame_to_send);
        }
    }
    s.info = buffer;
    s.seq = next_frame_to_send;
    s.ack = 1 – frame_expected;
    to_physical_layer(&s);
    start_timer(s.seq);
  }
}
```

One-Bit Sliding Window (3)

Two scenarios show subtle interactions exist in p4:

- Simultaneous start [right] causes correct but slow operation compared to normal [left] due to duplicate transmissions.



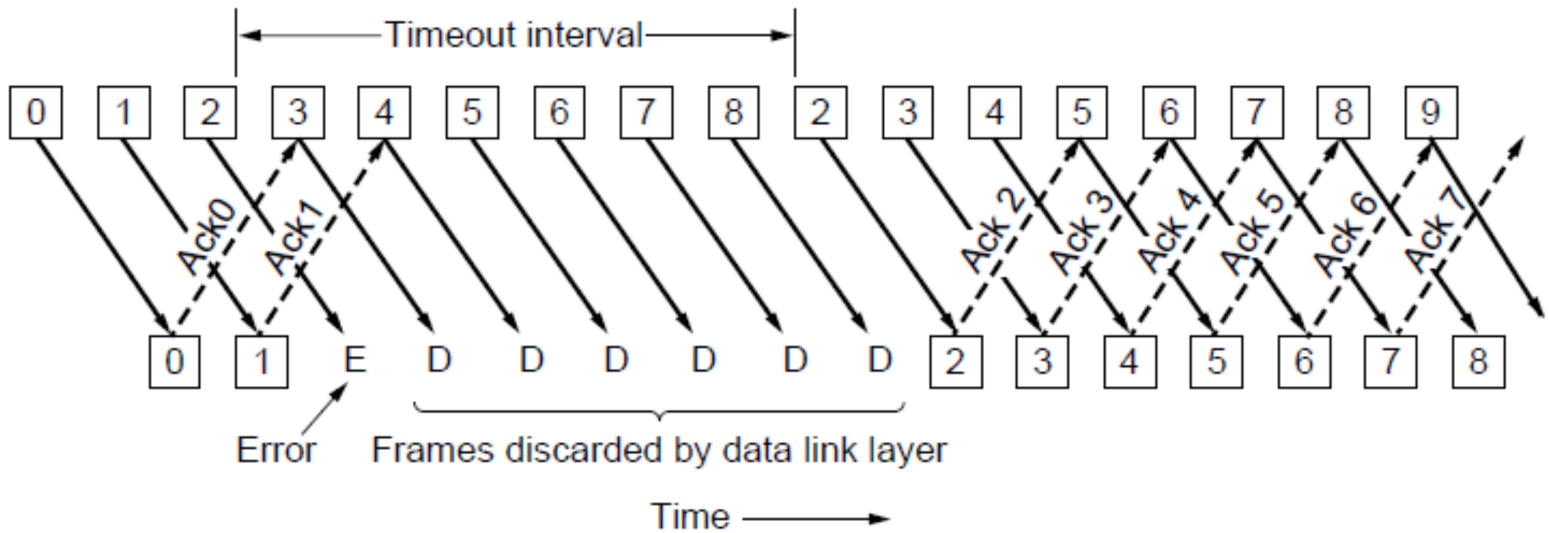
Notation is (seq, ack, frame number). Asterisk indicates frame accepted by network layer .

Normal case | Correct, but poor performance

Go-Back-N (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 (2)

Tradeoff made for Go-Back-N:

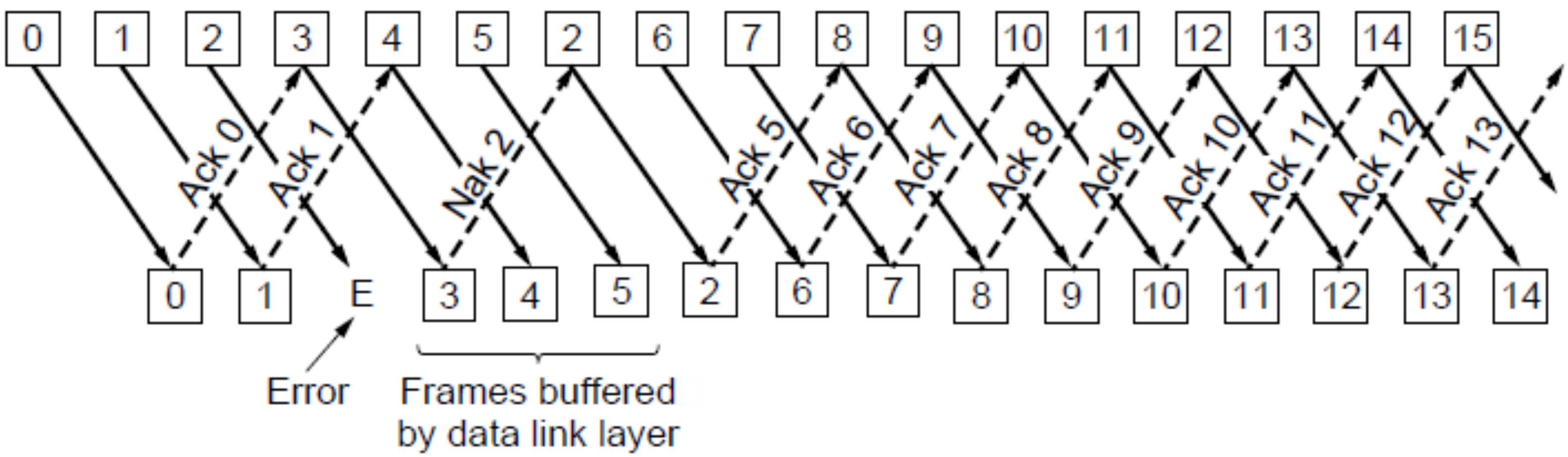
- Simple strategy for receiver; needs only 1 frame
- Wastes link bandwidth for errors with large windows; entire window is retransmitted

Implemented as p5 (see code in book)

Selective Repeat (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 (2)

Tradeoff made for Selective Repeat:

- More complex than Go-Back-N due to buffering at receiver and multiple timers at sender
- More efficient use of link bandwidth as only lost frames are resent (with low error rates)

Implemented as p6 (see code in book)

Selective Repeat (3)

For correctness, we require:

- Sequence numbers (s) at least twice the window (w)

Error case (s=8, w=7) – too few sequence numbers

Correct (s=8, w=4) – enough sequence numbers