CISC 7332X T6
Data Link Protocols

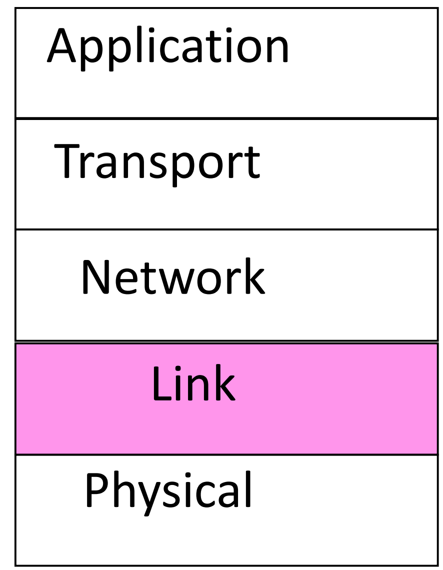
Hui Chen

Department of Computer & Information Science

CUNY Brooklyn College

Data Link Layer

- Responsible for delivering frames of information over a single link
  - Handles transmission errors
  - Regulates the flow of data



Design Issues in Data Link Layer

- Discussed
  - Concept of frames
  - Error control
  - Framing methods
- Possible services
- Data link protocols and flow control

Outline

- Data link protocol (for point-to-point links)
  - A utopian simplex protocol
  - Stop-and-wait protocols
    - Stop-and-wait for an error-free channel
    - Stop-and-wait for a noisy channel
    - Analysis of stop-and-wait protocols
  - Sliding window protocols
    - 1-bit sliding window
    - Go-Back-N
    - Selective repeat
- Data link protocols in practice

Possible Services

- Unacknowledged connectionless service
  - Frame is sent with no connection/error recovery
  - Example: Ethernet
- Acknowledged connectionless service
  - Frame is sent with retransmissions if needed
  - Example: 802.11
- Acknowledged connection-oriented service
  - Connection is set up; rare

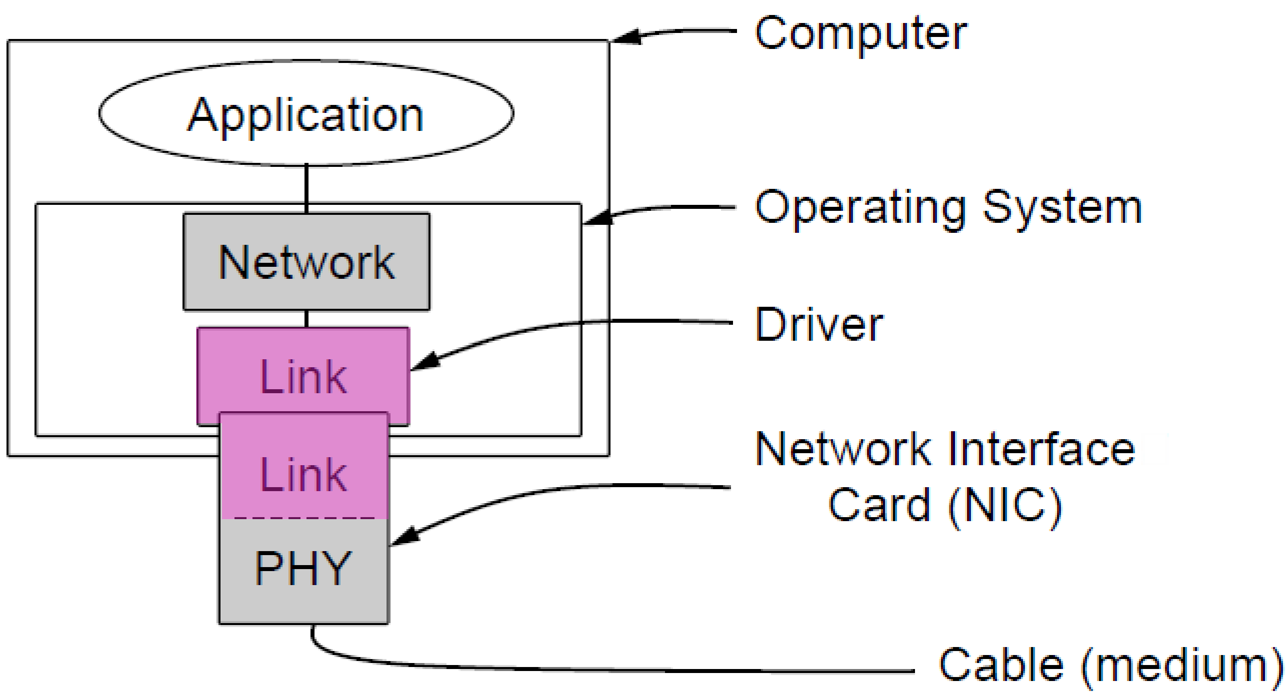
Elementary Data Link Protocols

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

Link Layer Environment

- Commonly implemented as
  - Network Interface Cards (NICs) and Operating Systems (OS) drivers
- Remark
  - Network layer (IP) is often a part of the OS software

Link Layer Environment: Example Implementation



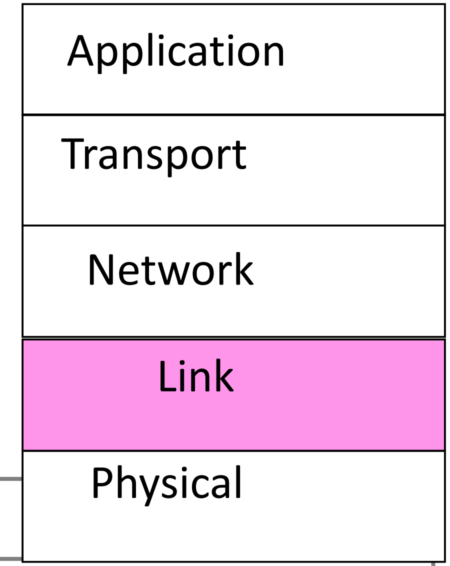
Link Layer: Services

- Link layer protocol implementations use library functions
  - See code (`protocol.h`) in next slide

Example: protocol.h

```
#define MAX_PKT 1024                /* determines packet size in bytes
*/
typedef enum {false, true} boolean;                  /* boolean type */
typedef unsigned int seq_nr;        /* sequence or ack numbers */
typedef struct {unsigned char data[MAX_PKT];} packet;/*packet
definition*/
typedef enum {data, ack, nak} frame_kind; /* frame_kind definition */
typedef struct {   /* frames are transported in this layer */
  frame_kind kind;                  /* what kind of frame is it? */
  seq_nr seq;       /* sequence number */
  seq_nr ack;       /* acknowledgement number */
  packet info;      /* the network layer packet */
} frame;
/* Wait for an event to happen; return its type in event. */
void wait_for_event(event_type *event);
/* Fetch a packet from the network layer for transmission on the
channel. */
void from_network_layer(packet *p);
/* Deliver information from an inbound frame to the network layer. */
void to_network_layer(packet *p);
/* Go get an inbound frame from the physical layer and copy it to r. */
void from_physical_layer(frame *r);
/* Pass the frame to the physical layer for transmission. */
void to_physical_layer(frame *s);
/* Start the clock running and enable the timeout event. */
void start_timer(seq_nr k);
/* Stop the clock and disable the timeout event. */
void stop_timer(seq_nr k);
/* Start an auxiliary timer and enable the ack_timeout event. */
void start_ack_timer(void);
/* Stop the auxiliary timer and disable the ack_timeout event. */
void stop_ack_timer(void);
/* Allow the network layer to cause a network_layer_ready event. */
void enable_network_layer(void);
/* Forbid the network layer from causing a network_layer_ready event.
*/
void disable_network_layer(void);
/* Macro inc is expanded in-line: increment k circularly. */
#define inc(k) if (k < MAX_SEQ) k = k + 1; else k = 0
```

Example: protocol.h: Services



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

Questions?

- Link layer environment
- Link layer services

Data Link Protocols

- Examine three protocols
  - Utopian Simplex Protocol (p1)
  - Stop-and-Wait Protocol in an Error-Free Channel (p2)
  - Stop-and-Wait Protocol in a Noisy Channel (p3)

Utopian Simplex Protocol

- An optimistic protocol (p1) to start
  - Assumes no errors, and receiver as fast as sender
  - Considers one-way data transfer
  - That's it, no error or flow control …
    - Flow control
      - Prevent (fast) sender overwhelms (slow) receiver

Utopian Simplex Protocol: Peer Interface and Implementation

- Unrealistic
  - Error can occur
  - Sender may be faster than receiver

```
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 in Error-free Channel

- Error won’t happen, no error control; but senders may be too fast
  - Adding flow control to protocol p1
- Protocol (p2) ensures sender won’t outpace receiver:
  - Receiver returns a dummy frame called “ack” when ready
  - Stop and wait:
    - Only one frame out from the sender at a time
  - So, added flow control via the stop-and-wait mechanism

Stop-and-Wait: Example Implementation

```
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);
  }
}
```

Wait for Ack

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);
  }
}
```

Send Ack

Receiver sends ack after passing frame to network layer

Stop-and-Wait in 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
  - For stop-and-wait, 2 numbers (1 bit) are sufficient

Stop-and-Wait/ARQ: Example: Sender

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/ARQ: Example: Receiver

```
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 (event == frame_arrival) {`

If it's new then take it and advance expected frame → `if (r.seq == frame_expected) { to_network_layer(&r.info); inc(frame_expected);`

Ack current frame → `s.ack = 1 – frame_expected;`

Questions

- ARQ
- Error control via stop-and-wait
- Flow control via stop-and-wait

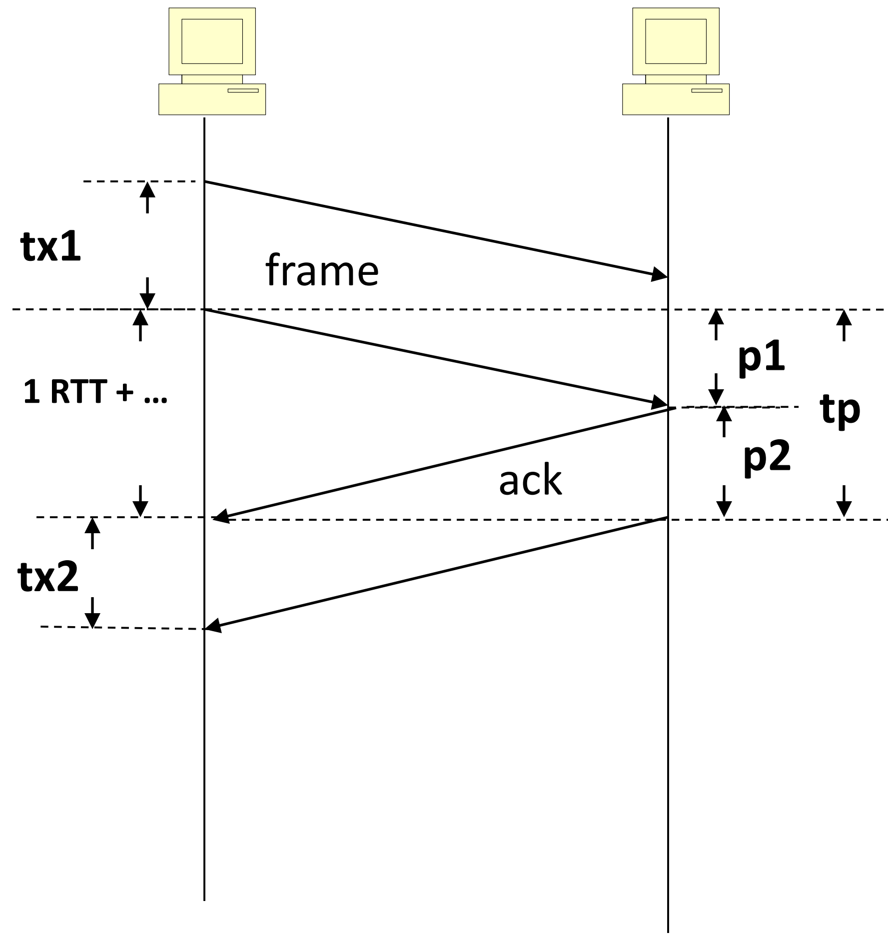
Analysis of Stop-and-Wait

- How well does the stop-and-wait protocols perform?
- Metrics
  - Throughput (effective bandwidth) and link utilization

Throughput

- Q: what is the **maximum** throughput (effective bandwidth)?
- Best case
  - No error, no retransmission
  - Send and receiver are equally fast
- Note: tp = p1 + p2 = 1 RTT
- Transfer time = tx1 + tx2 + tp
- Throughput =

  Transfer size/Transfer time

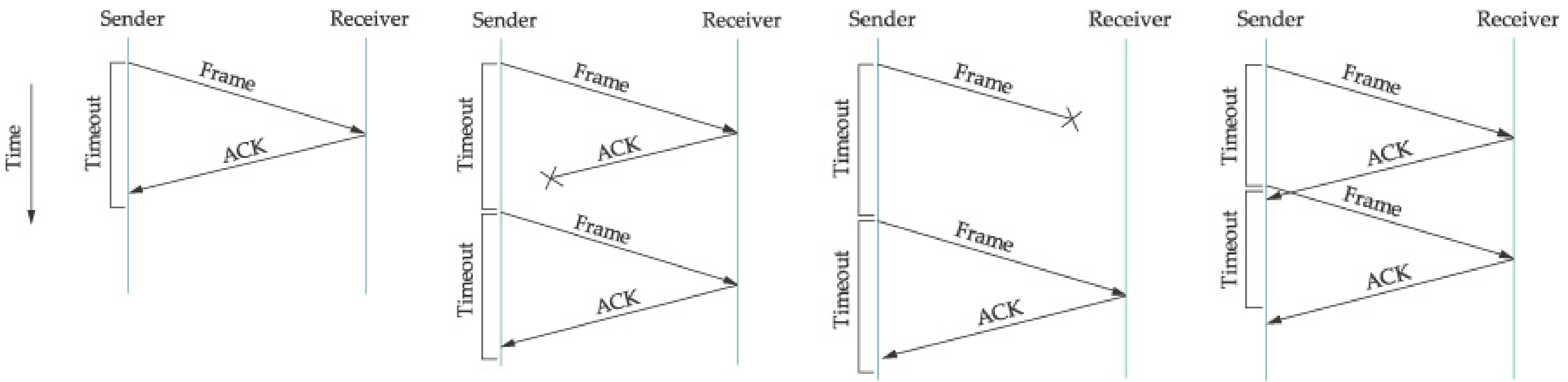


Link Utilization

- How much capacity of a channel is being used?
  - Link utilization
    - Throughput / Max Data Rate of the Channel

Timeout?

- How long should the receiver wait?



- Timeout: 2 x RTT or more …

Exercise 1

- Data frame size (data) = 1500 bytes
- Acknowledgement frame size (ack) = 64 bytes
- Stop-and-Wait protocol: receiver is forced to wait 2 RTT before transmitting acknowledgement frame after having received data frame. No additional processing and queueing delay
- Draw timeline diagram first, and then compute throughputs and link utilization for one of the following,
- Dial-up
  - RTT = 87 μs; Link bandwidth: 56 Kbps
- Satellite
  - RTT = 230 ms; Link bandwidth: 45 Mbps

Questions?

- Estimating link utilization at best-case scenario
- What if the simple stop-and-wait protocol yields poor link utilization ratio?

Sliding Window Protocols

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

Concept of Sliding Window

- 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

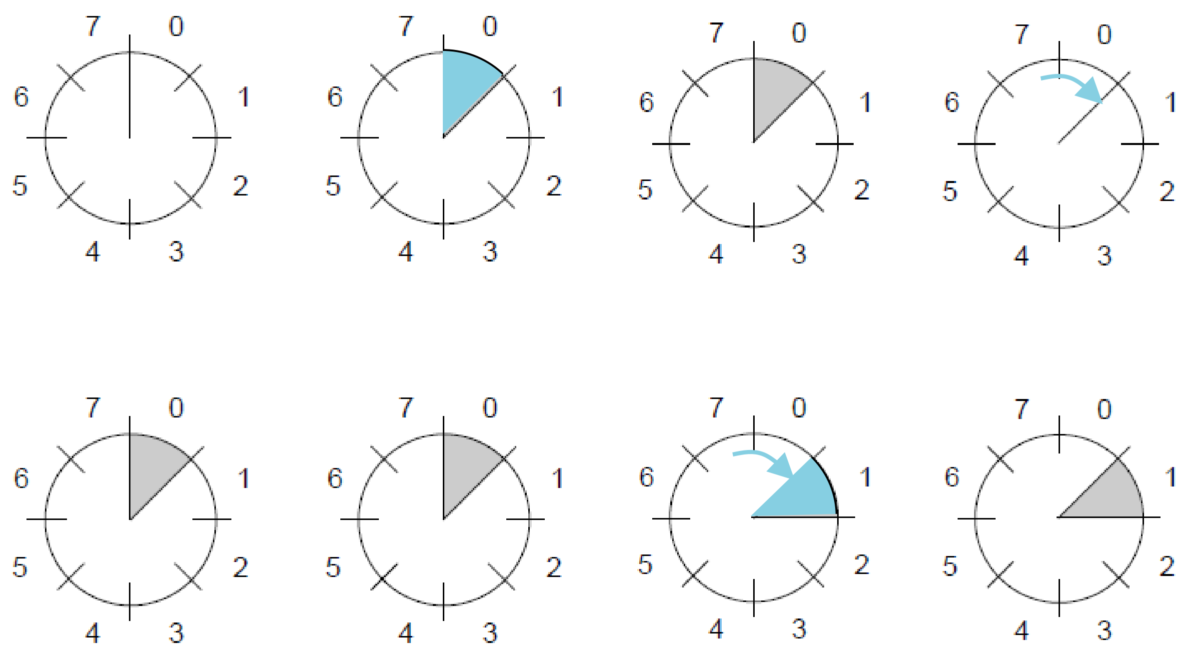
Concept of Sliding Window: Example

- A sliding window advancing at the sender and receiver
  - Ex: window size is 1, with a 3-bit sequence number.

Concept of Sliding Window: Example

Sender

Receiver



At the start

First frame is sent

First frame is received

Sender gets first ack

Sliding Window: Advantage

- 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

Questions?

- Concept of sliding window

One-Bit Sliding Window

- Transfers data in both directions with stop-and-wait
  - Piggybacks acks on reverse data frames for efficiency
  - Handles transmission errors, flow control, early timers

One-bit Sliding Window: Example: Sender

```
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: Example: Receiver

```
while (true) {
```

Wait for frame or timeout

```
    wait_for_event(&event);
    if (event == frame_arrival) {
        from_physical_layer(&r);
```

If a frame with new data then deliver it

```
        if (r.seq == frame_expected) {
            to_network_layer(&r.info);
            inc(frame_expected);
        }
```

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

```
        if (r.ack == next_frame_to_send) {
            stop_timer(r.ack);
            from_network_layer(&buffer);
            inc(next_frame_to_send);
        }
    }
```

(Otherwise it was a timeout)

```
    s.info = buffer;
    s.seq = next_frame_to_send;
    s.ack = 1 – frame_expected;
```

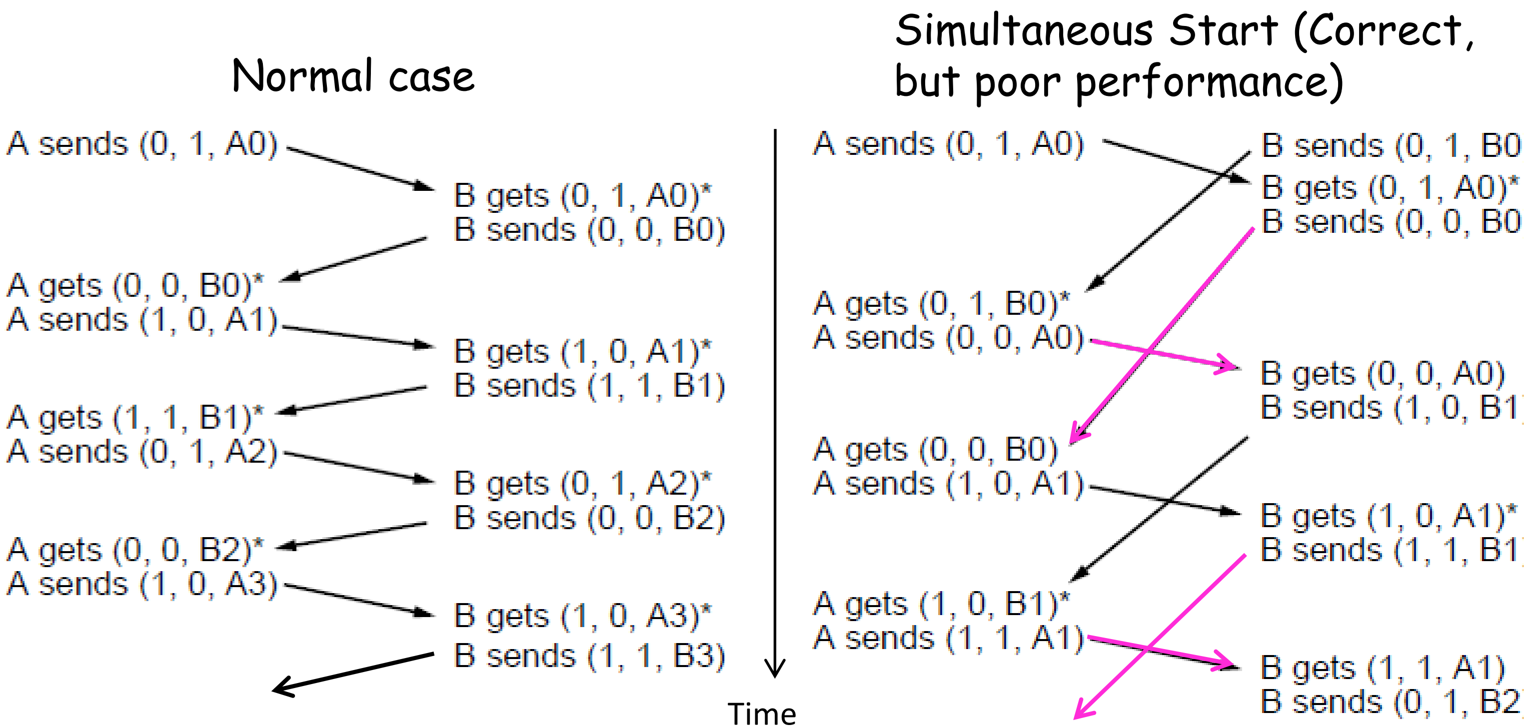
Send next data frame or retransmit old one; ack the last data we received

```
    to_physical_layer(&s);
    start_timer(s.seq);
  }
}
```

One-Bit Sliding Window: Interactions

- Two scenarios show subtle interactions exist in p4:
  - Simultaneous start [right] causes correct but slow operation compared to normal [left] due to duplicate transmissions.

Simultaneous Start

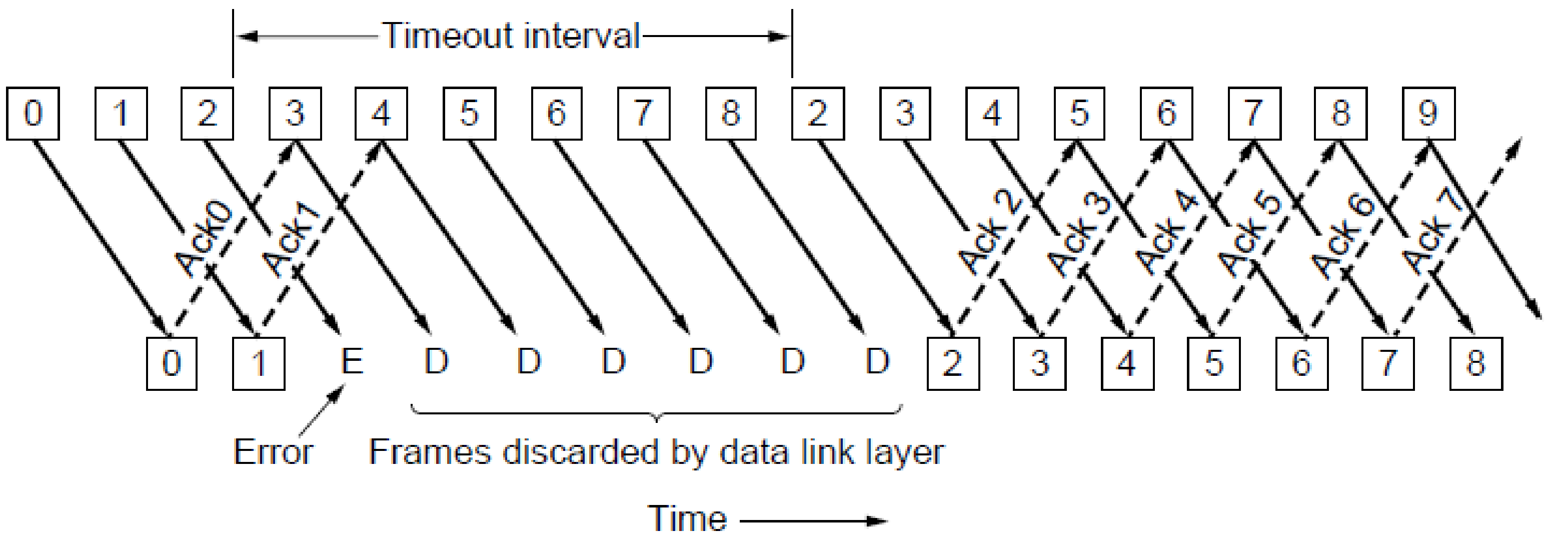


Notation is (seq, ack, frame number). Asterisk indicates frame accepted by network layer .

Go-Back-N

- 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: Example



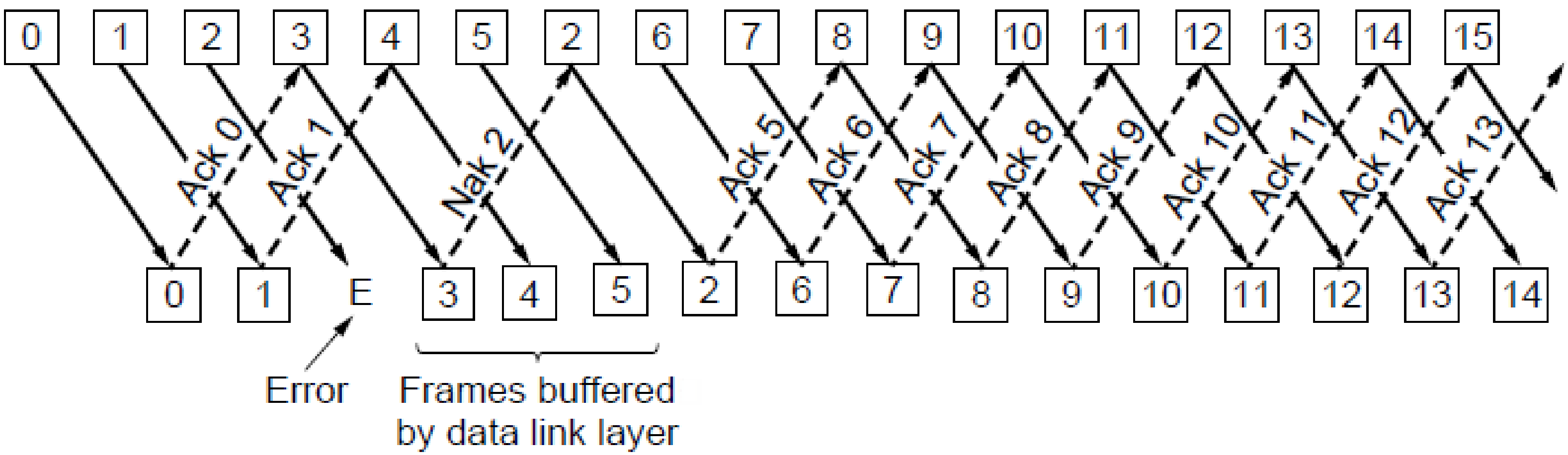
Go-Back-N: Discussion

- 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

Selective Repeat

- 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: Example



Selective Repeat: Discussion

- 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)

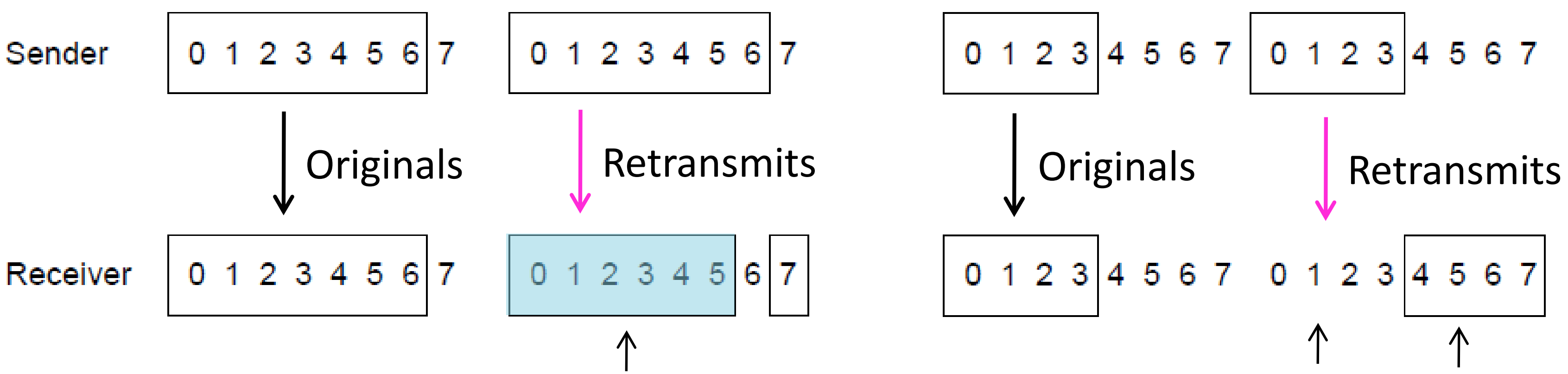
Selective Repeat: Sequence Number

- For correctness, we require:
  - Sequence numbers (s) at least twice the window (w)

Selective Repeat: Sequence Number

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

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



New receive window overlaps old – retransmits ambiguous

New and old receive window don’t overlap – no ambiguity

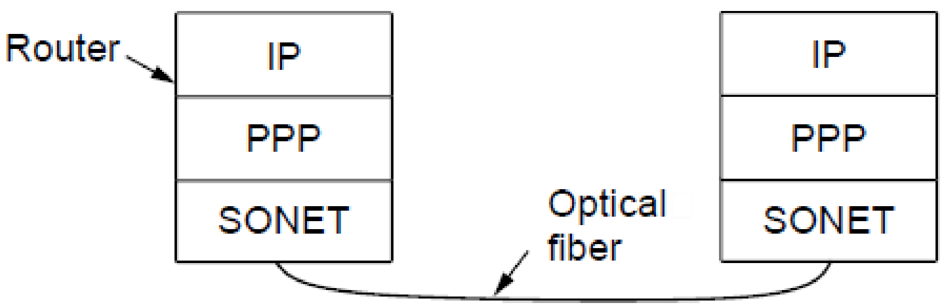
Data Link Protocols: Examples in Practice

- Packet over SONET
- PPP (Point-to-Point Protocol)
- ADSL (Asymmetric Digital Subscriber Loop)

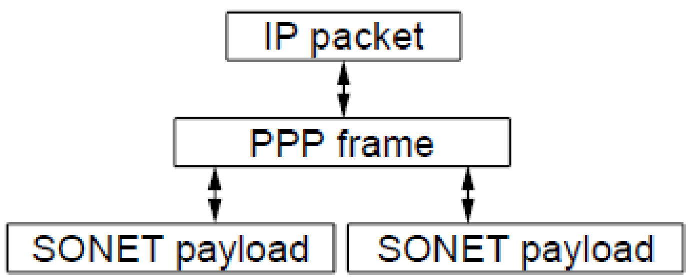
Packet over SONET

- Packet over SONET is the method used to carry IP packets over SONET optical fiber links
  - Uses PPP (Point-to-Point Protocol) for framing

Packet over SONET



Protocol stacks

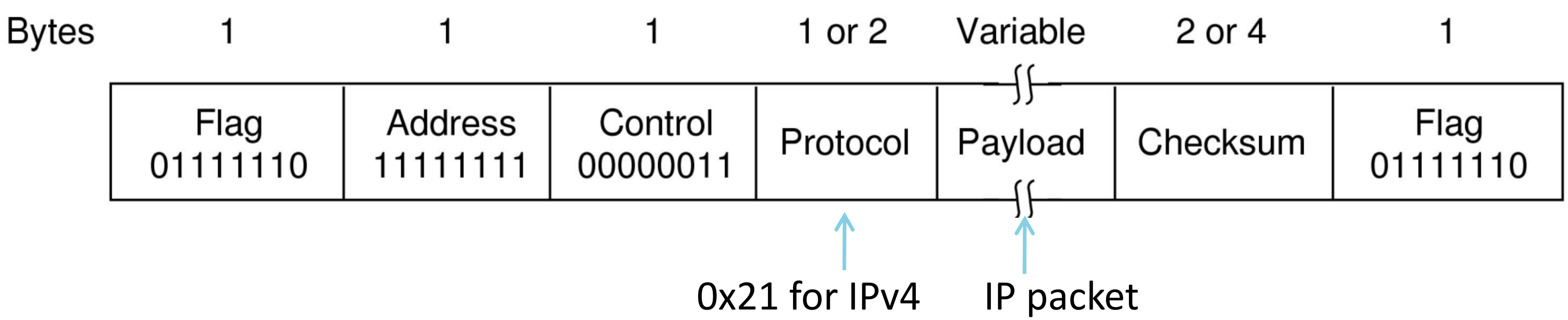


PPP frames may be split over SONET payloads

PPP

- PPP (Point-to-Point Protocol) is a general method for delivering packets across links
    - Framing uses a flag (0x7E) and byte stuffing
    - "Unnumbered mode" (connectionless unacknowledged service) is used to carry IP packets
    - Errors are detected with a checksum

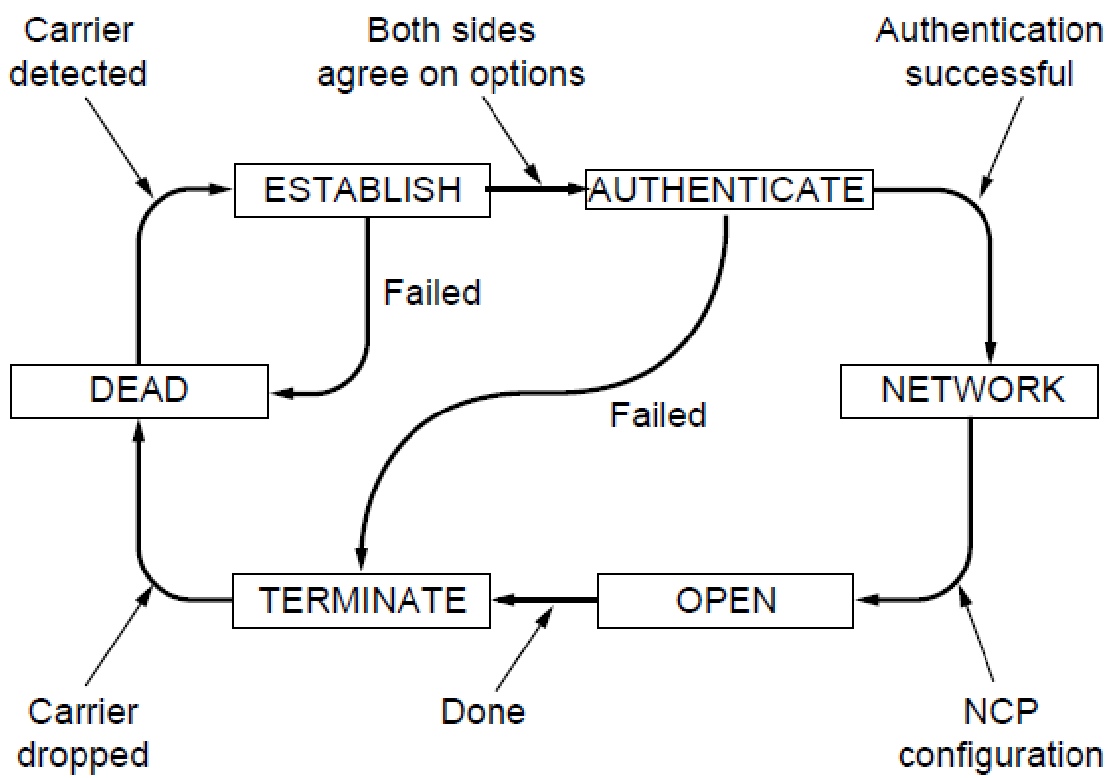
PPP Frame


Bytes
1
1
1
1 or 2
Variable
2 or 4
1
Flag 01111110
Address 11111111
Control 00000011
Protocol
Payload
Checksum
Flag 01111110
0x21 for IPv4
IP packet

Link Control Protocol

- A link control protocol brings the PPP link up/down

Link Control

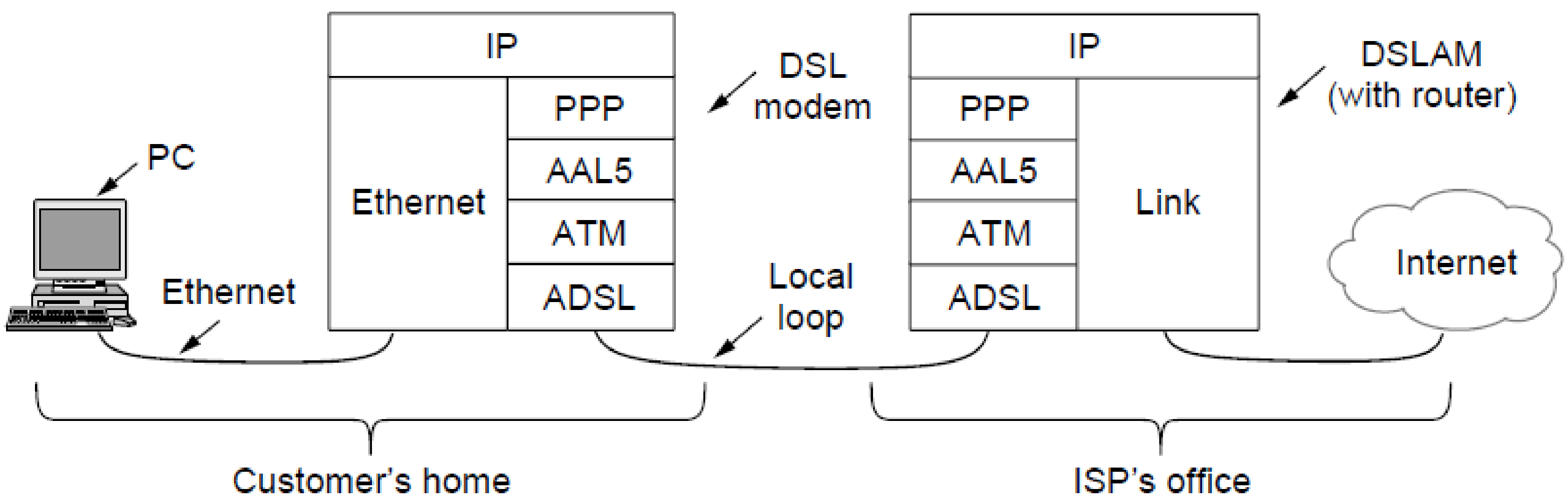


State machine for link control

ADSL

- Widely used for broadband Internet over local loops
  - ADSL runs from modem (customer) to DSLAM (ISP)
  - IP packets are sent over PPP and AAL5/ATM (over)

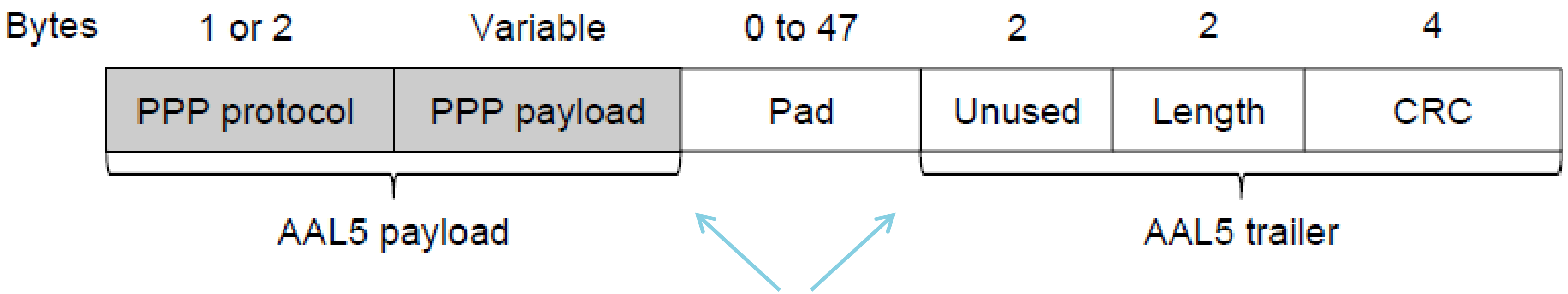
ADSL: Protocol Stack



ADSL and PPP

- PPP data is sent in AAL5 frames over ATM cells:
  - ATM is a link layer that uses short, fixed-size cells (53 bytes); each cell has a virtual circuit identifier
  - AAL5 is a format to send packets over ATM
  - PPP frame is converted to a AAL5 frame (PPPoA)

ADSL Frame



AAL5 frame is divided into 48 byte pieces, each of which goes into one ATM cell with 5 header bytes

Questions

- Data link protocols in practice