Computer Networks X_400487

Lecture 3

Chapter 3: The Data Link Layer—Part 1

Lecturer: Jesse Donkervliet

Recap of the Physical Layer

Responsible for transferring *bits* over a *wire-like* medium.

Maximum data rate determined by *bandwidth* and *signal-to-noise ratio.* $R = B \times \log_2\left(1 + \frac{S}{N}\right)$

Physical layer responsible for translating bits to and from waves; sharing channel with multiple users

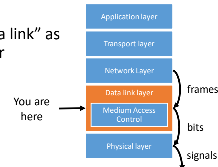
Application layer
Transport layer
Network Layer
Data link layer
Medium Access Control
Physical layer
signals

Where are the frames?

The Data Link layer

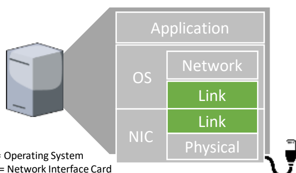
Responsible for transferring *frames* over a single link

1. Groups bits into frames
2. Offers “sending frames over a link” as a *service* to the network layer
3. Handles transmission errors Q: Why needed?
4. Regulates data flow



Link layer environment

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



OS = Operating System
NIC = Network Interface Card

Data Link Layer — Roadmap

Part 1
- **Framing**
- **Flow Control**
- **Guaranteed Delivery**
- **Sliding Window Protocols**

Part 2
- Error detection
- Error correction

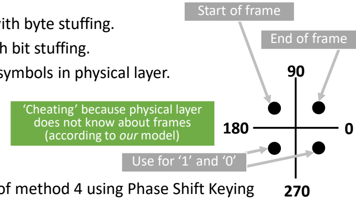
Framing

From Bit Stream to Discrete Units of Information

Copyright Jesse Donkervliet 2024

Framing Methods

1. Byte count.
2. Flag bytes with byte stuffing.
3. Flag bits with bit stuffing.
4. Use special symbols in physical layer.

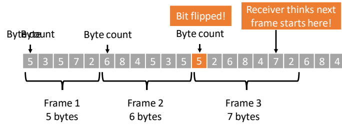


Example of method 4 using Phase Shift Keying

Copyright Jesse Donkervliet 2024

Framing
Byte count

Q: Advantage? Disadvantage?



Copyright Jesse Donkervliet 2024 CNSE by Tanenbaum & Wetherall, © Pearson Education-Prentice Hall and D. Wetherall, 2011

Framing
Byte stuffing

Q: Disadvantage?

Use a 'flag' byte to indicate start and end of frame.

Let's say our flag byte is 00000111_2 (7_{10}).

Copyright Jesse Donkervliet 2024

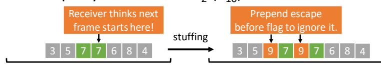
Framing
Byte stuffing

Character	Escape Character
%	%25

What if the data contains a flag byte?

Use an 'escape' byte to ignore certain flag bytes.

Let's say our escape byte is 00001001_2 (9_{10}).



Q: Algorithm on the receiving side?

Q: Are we done?

Copyright Jesse Donkervliet 2024

Framing
Byte stuffing

Q: What is the overhead of this approach?

Escape bytes can also occur in data!

Let's use letters for generality.
Flag byte = F, Escape byte = E.

Escape both 'escape' and 'flag' bytes.

Copyright Jesse Donkervliet 2024

Framing
Bit stuffing

Byte stuffing can be space inefficient.

Byte stuffing		**Bit stuffing**
Flag byte	→	Bit pattern
Escape byte	→	Insert single bit

Example:

Bit pattern: 01111110

Insert bit if pattern in data: 011111010

Add one extra bit

Bit stuffing example
Receiver

Bit pattern: 01111110

```
1100010100011110111100011100110110000001101110001111101000101101111
0100011010100010010100111110100001110010000011001001010000001011111
0101110000111110100110001110011111001100110000011101111001111001010
0001110111110000111011100011001011001100110001111100100111001101 0111
0101001110000111110110101000000010001111100110101110011001100001 0100
0101010000011010010111011010100101001001111100010010100100000001 0010
1101111101001100100111000011101111110111100001010011010001111110 0111
1110000000100100001001011001110101101100000011000110101100000000 0011
0101101001100001100000111101100100101001100110110110011001101101100
0101000111101011101000111110110010101000001010011011100110101101000
0110101111100011100010000010110101010100101111100101100010010010000
0110101110111011100000001000110010000000111110111001001011101101 0010
0010111110010101001011100011001010011001000100011010100100101010011
0111001111000111101000101001010011111000111110001001111101100010000
1010001101010111001011100010010011110101101001111100001111101101001
1011001011111010001010110000001111000111
```


Bit stuffing example
Receiver

Q: Are we done?

Bit pattern: 01111110

F1 / F2 — the same bit sequence as on slide 14, with the first part marked F1 and the second part marked F2.

Bit stuffing example
Receiver

Bit pattern: 01111110

F1

```
110001010001111011110001110011011000000110111000 11111 1000101101111
0100011010100010010100 11111 100001110010000011001001010000001 11111
 1011100000 11111 10011000111 00 11111 0110011000001110111100111001010
000111011111000011101110001100101100110011000 11111 01001110011010111
0101001110000 11111 110101000000010000 11111 01101011100110011000010100
010101000001101001011101101010010100100 11111 001001010010000000100 10
110 11111 10011001001110000111 0 11111 1111000010100110100
    000000100100001001011001110101101100000011000110101 1000000000 11
0101101001100001100000111101100100101001100110110110011001101101100
0101000111101011101000 11111 1100101010000010100110111001101011 01000
011010 11111 001110001000000101101010101001 0 11111 010110001001001 0000
0110101110111011100000001000110010000000 11111 11100100101110110 10010
0010 11111 0101010010111000110010100110010001000110101001001010 10011
011100111100011110100010100101 00 11111 00 11111 00100 11111 1100010000
1010001101010111001011100010010011110101101000 11111 000 11111 1101001
10110010 11111 1000101011000 0 11111 00111
```

F2

Bit pattern in data

Bit stuffing example
Receiver

Q: What is the overhead of this approach?

Bit pattern: 01111110

Q: What is the algorithm at the sender?

Sender:

1. Change every '11111' into '111110' (stuffing).
2. Add '01111110' to start and end of each frame.

F1 / F2 — the same bit sequence as on slide 16, partly covered by the boxes above.

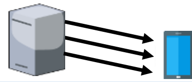
Flow Control

Could you speak more slowly, please?

A *Resource Management* problem

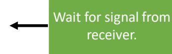
Utopian simplex protocol
The ideal case

```
…
while True:
    packet = from_network_layer()
    frame.payload = packet
    to_physical_layer(frame)
```



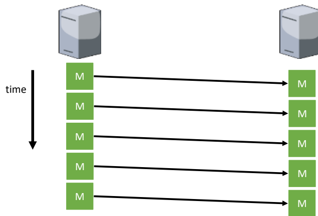
Stop-and-wait
for error-free channel

```
…
while True:
    packet = from_network_layer()
    frame.payload = packet
    to_physical_layer(frame)
    event = wait_for_event()
```



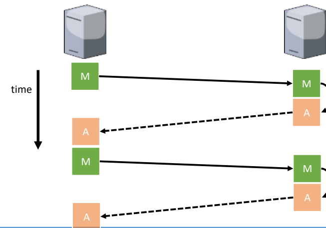
Wait for signal from receiver.

Example of
Utopian Simplex Protocol



Example of
Stop-and-Wait Protocol

Q: What if a frame gets lost?



Guaranteed Delivery

Acknowledgments, Sequence Numbers, and Retransmissions

How Can We Know If a Frame Gets Lost?

Ask a different question

Q: How can we know if a frame *arrives*?

Send a message back: "I got your message!"

Q: When do we want to retransmit data?

Q: What if the acknowledgment gets lost?

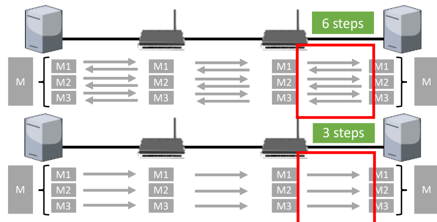
We assume our original message did not arrive

It depends on … the application!

Acknowledgments let the sender know it does ***not*** need to retransmit data.

Many protocols either use or don't use acknowledgments. Different approach: Support acknowledgments, but let the application decide if it needs to use acknowledgments or not.

To acknowledge, or not to acknowledge

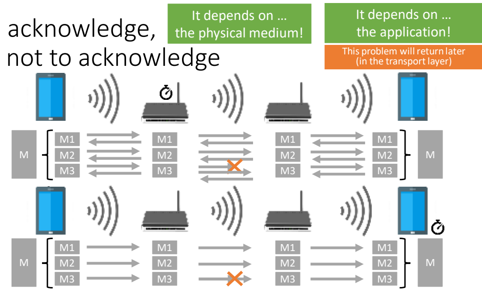


To acknowledge, or not to acknowledge

It depends on ... the physical medium!

It depends on ... the application!

This problem will return later (in the transport layer)

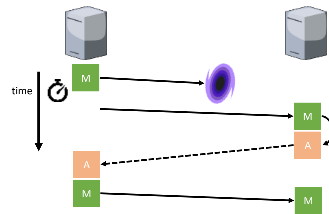


Automatic Repeat ReQuest (ARQ) Guaranteed Delivery over Unreliable Channel

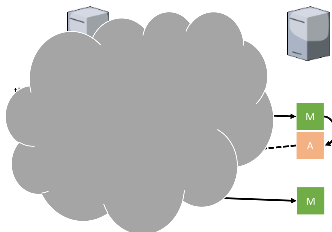
Same as stop-and-wait, except:

1. Keep track of frames using sequence numbers.
2. Wait until previous frame has been accepted.

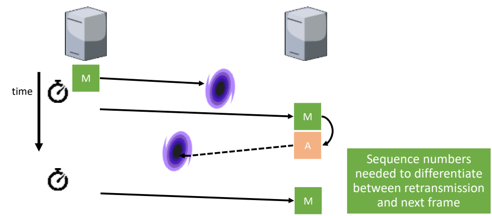
ARQ Example



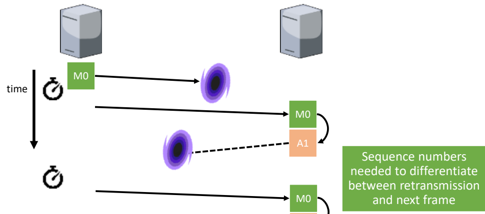
ARQ Example



ARQ Example



ARQ Example



Sequence numbers needed to differentiate between retransmission and next frame

Automatic Repeat ReQuest (ARQ) Guaranteed Delivery over Unreliable Channel

ARQ adds error control

Receiver acks frames that are correctly delivered.

Sender sets timer and resends frame if no ack.

Q: How long should we wait?

Q: What can go wrong?

Frames and acks must be identifiable (e.g., with sequence number)

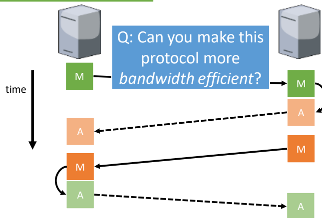
Else receiver cannot tell retransmission (due to lost ack or early timer) from new frame.

For stop-and-wait, 2 numbers (1 bit) are sufficient.

Q: Why sufficient?

Acknowledgements

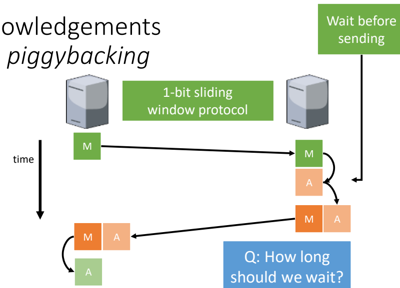
Bidirectional communication



Q: Can you make this protocol more *bandwidth efficient?*

Acknowledgements With *piggybacking*

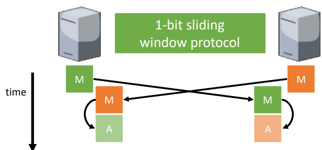
Wait before sending



1-bit sliding window protocol

Q: How long should we wait?

Stop-and-Wait

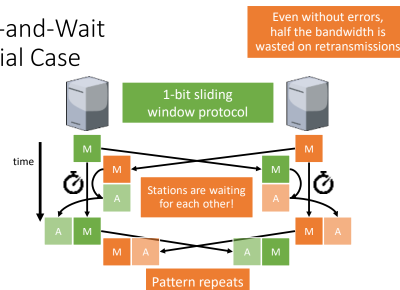


1-bit sliding window protocol

Q: What happens now?

Stop-and-Wait Special Case

Even without errors, half the bandwidth is wasted on retransmissions



1-bit sliding window protocol

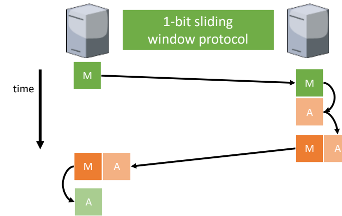
Stations are waiting for each other!

Pattern repeats

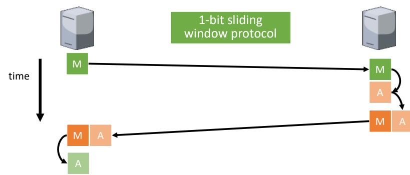
Sliding Window Protocols

Improving Performance using *Pipelining*

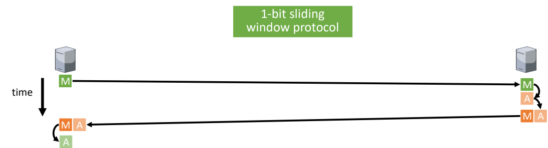
Stop-and-Wait: A 1-Bit Sliding Window Protocol



Stop-and-Wait: A 1-Bit Sliding Window Protocol



Stop-and-Wait: A 1-Bit Sliding Window Protocol



Bandwidth inefficient for high-latency channels

Sliding window protocols

Q: Which properties cause performance to decrease?

When using stop-and-wait, *data rate decreases* when:

- Latency increases
- Frame size decreases

Solution

Send next frame while waiting for acknowledgment of current frame

Sender window specifies how many frames a sender is allowed to send before waiting for an acknowledgement.

Receiver window specifies the range of frames that the receiver is allowed to accept.

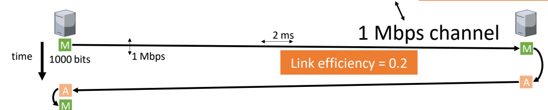
Stop-and-Wait / ARQ: A 1-Bit Sliding Window Protocol

It takes $\frac{1,000}{1,000,000} = 0.001$ seconds to send frame

It takes 2 ms for the frame to arrive at the receiver, takes 2 ms for the (0-bit) acknowledgment to come back at the sender

1 frame per 0.001+0.002+0.002 seconds = 200 kbps

Small window reduces performance



Stop-and-Wait: A 1-Bit Sliding Window Protocol

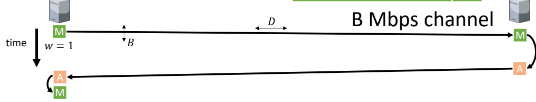
- Frame size (in bits/bytes): f
- Window size (in frames): w
- Bandwidth (max. data rate of physical channel): B_p
- Bandwidth (frames per second): B_f
- Propagation delay (in seconds): D

It takes $\frac{f}{B_p}$ seconds to send frame, $\frac{B_p}{f} = B_f$

It takes D s for the frame to arrive at the receiver, takes D s for the (0-bit) acknowledgment to come back at the sender

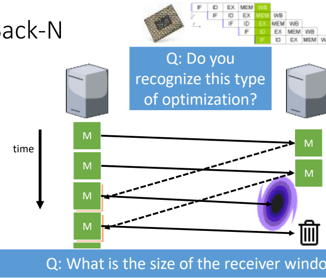
1 frame per $\frac{f}{B_p} + 2 \times D$ seconds

Link utilization $\leq \frac{w}{1+2B_f D}$



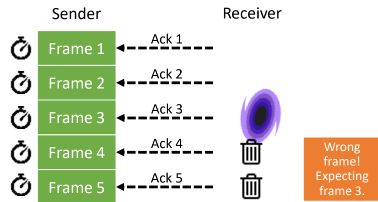
Copyright Jesse Donkervliet 2024

Go-Back-N



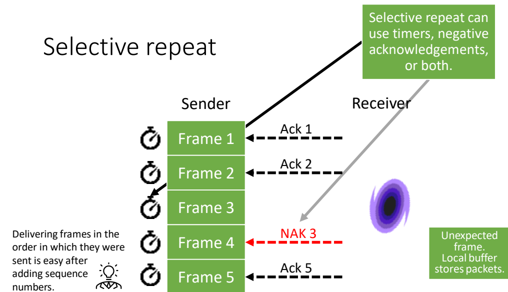
Copyright Jesse Donkervliet 2024

Go-Back-N



Copyright Jesse Donkervliet 2024

Selective repeat



Copyright Jesse Donkervliet 2024

Data Link Layer — Roadmap

Part 1
- **Framing**
- **Flow Control**
- **Guaranteed Delivery**
- **Sliding Window Protocols**

Part 2
- Error detection
- Error correction

Copyright Jesse Donkervliet 2024