# Introduction to Computer Networks

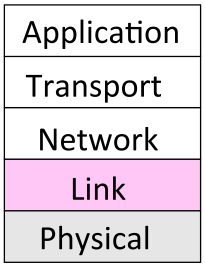
## Overview of the Link Layer

Computer Science & Engineering

UNIVERSITY *of* WASHINGTON

## Where we are in the Course

- Moving on to the Link Layer!



Application

Transport

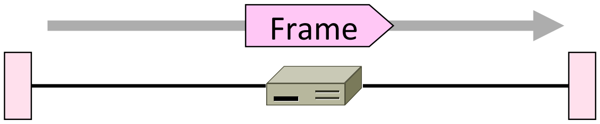
Network

Link

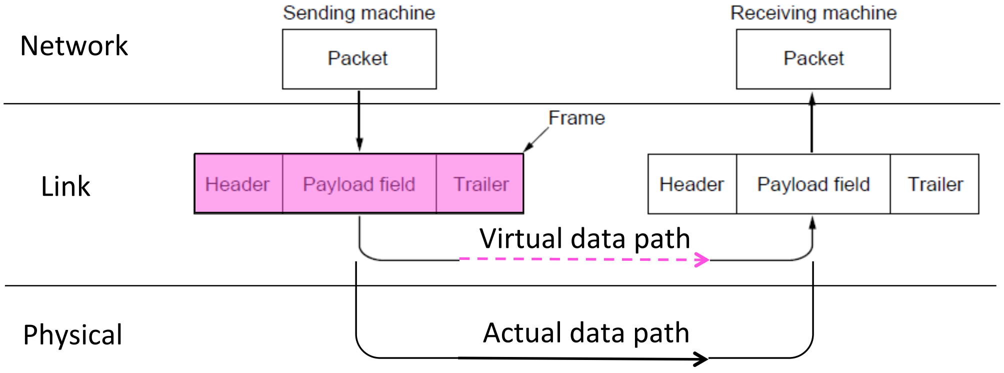
Physical

# Scope of the Link Layer

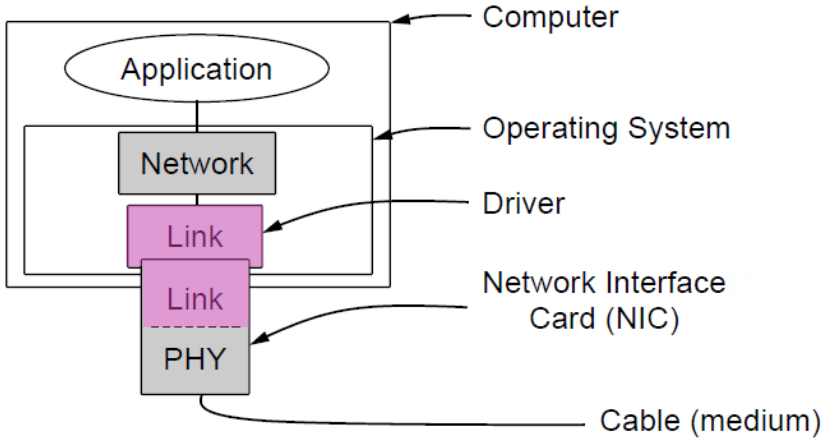
- Concerns how to transfer messages over one or more connected links
  - Messages are <u>frames</u>, of limited size
  - Builds on the physical layer



# In terms of layers

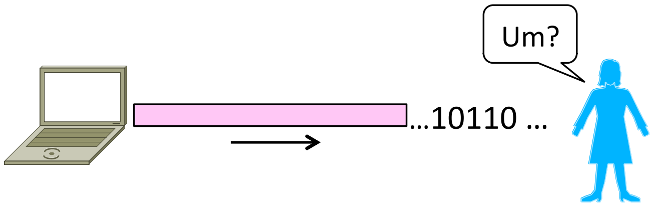


# Typical Implementation of Layers



# Topic

- The Physical layer gives us a stream of bits. How do we interpret it as a sequence of frames?

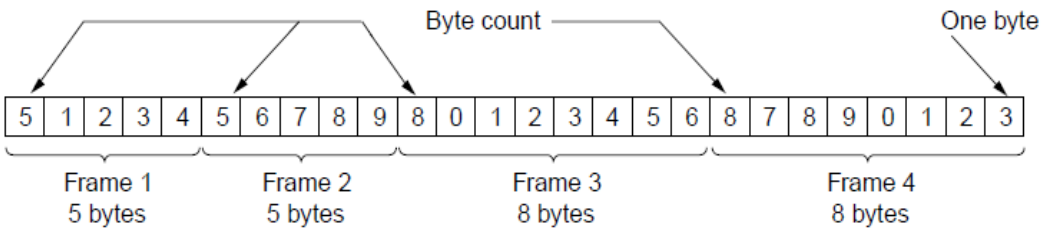


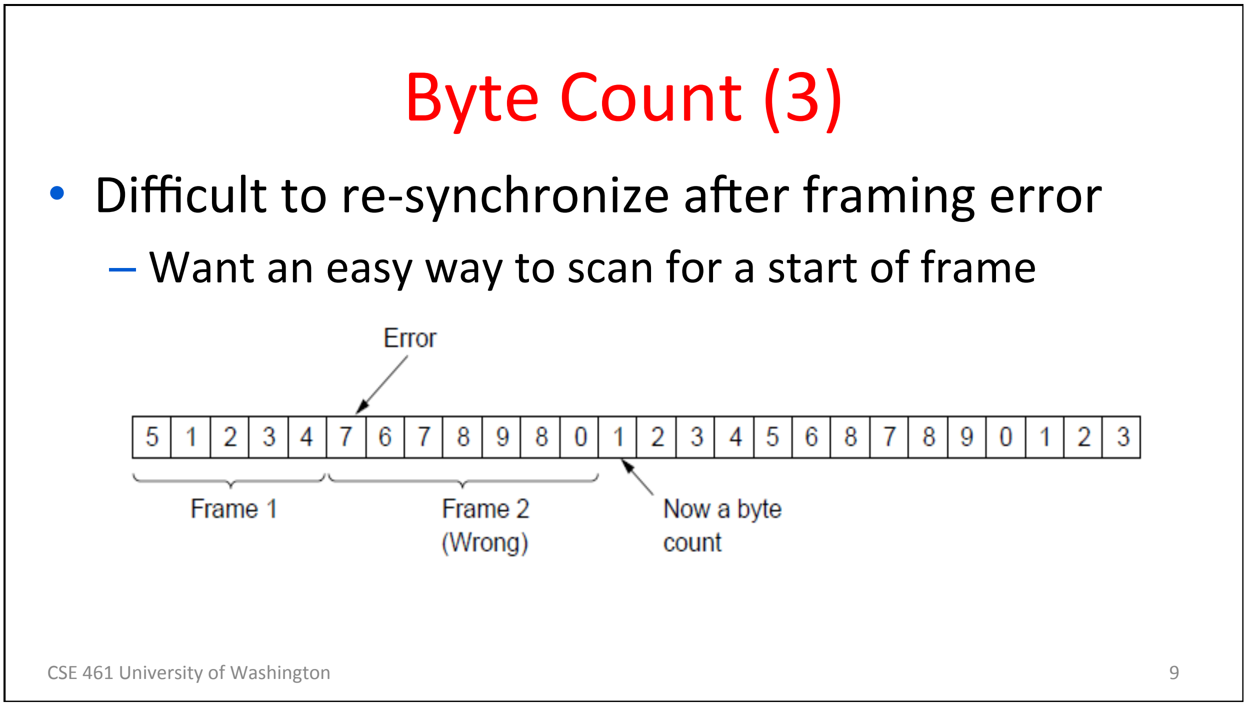
# Byte Count

- First try:
  - Let’s start each frame with a length field!
  - It’s simple, and hopefully good enough …

# Byte Count (2)

- How well do you think it works?





# Byte Count (3)

- Difficult to re-synchronize after framing error
  - Want an easy way to scan for a start of frame

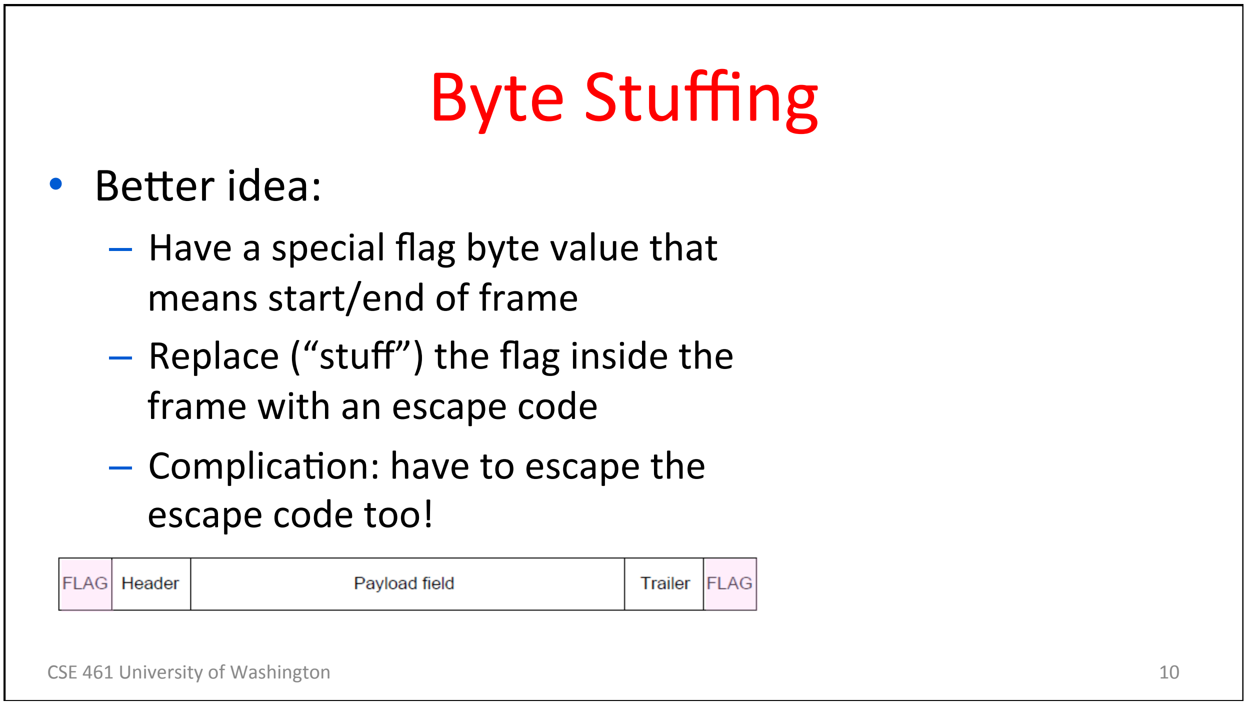
Error

| | | | | | | | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 5 | 1 | 2 | 3 | 4 | 7 | 6 | 7 | 8 | 9 | 8 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 8 | 7 | 8 | 9 | 0 | 1 | 2 | 3 |

Frame 1

Frame 2 (Wrong)

Now a byte count



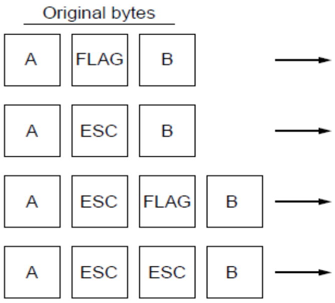
# Byte Stuffing

- Better idea:
  - Have a special flag byte value that means start/end of frame
  - Replace ("stuff") the flag inside the frame with an escape code
  - Complication: have to escape the escape code too!

| | | | | |
|---|---|---|---|---|
| FLAG | Header | Payload field | Trailer | FLAG |

# Byte Stuffing (2)

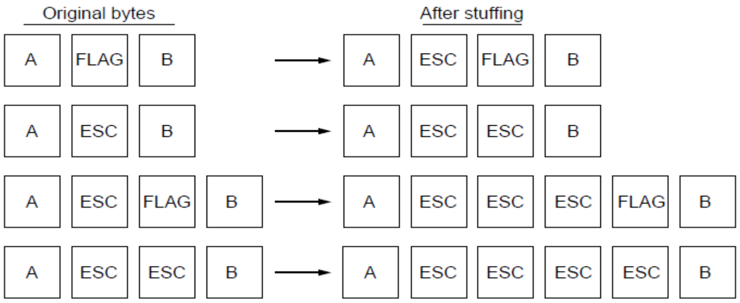
- Rules:
  - Replace each FLAG in data with ESC FLAG
  - Replace each ESC in data with ESC ESC



| Original bytes |
|---|
| A FLAG B → |
| A ESC B → |
| A ESC FLAG B → |
| A ESC ESC B → |

# Byte Stuffing (3)

- Now any unescaped FLAG is the start/end of a frame



| Original bytes | | After stuffing |
|---|---|---|
| A FLAG B | → | A ESC FLAG B |
| A ESC B | → | A ESC ESC B |
| A ESC FLAG B | → | A ESC ESC ESC FLAG B |
| A ESC ESC B | → | A ESC ESC ESC ESC B |

# Bit Stuffing

- Can stuff at the bit level too
  - Assume a flag has six consecutive 1s
  - On transmit, after five 1s in the data, insert a 0
  - On receive, a 0 after five 1s is deleted

# Bit Stuffing (2)

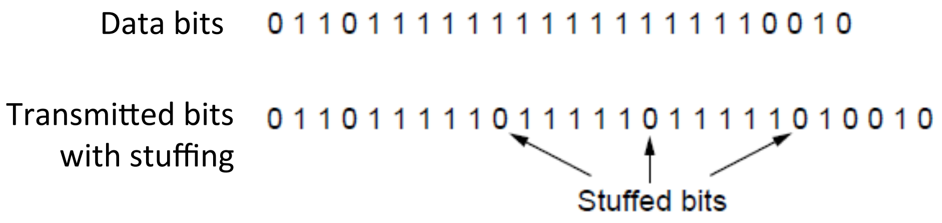
- Example:

Data bits 0 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 1 0

Transmitted bits
with stuffing

# Bit Stuffing (3)

- So how does it compare with byte stuffing?

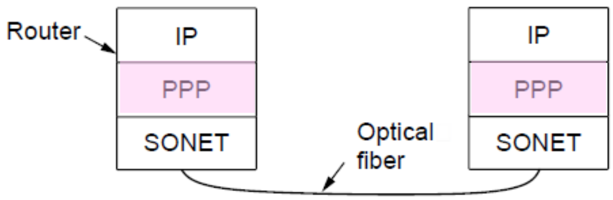


# Link Example: PPP over SONET

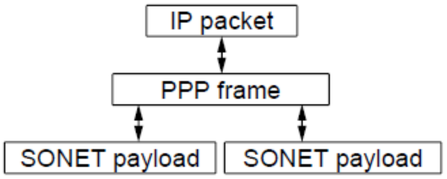
- PPP is Point-to-Point Protocol
- Widely used for link framing
  - E.g., it is used to frame IP packets that are sent over SONET optical links

# Link Example: PPP over SONET (2)

- Think of SONET as a bit stream, and PPP as the framing that carries an IP packet over the link



Protocol stacks



PPP frames may be split over SONET payloads

# Link Example: PPP over SONET (3)

- Framing uses byte stuffing
  - FLAG is 0x7E and ESC is 0x7D. To stuff (unstuff) a byte, add (remove) ESC, and XOR byte with 0x20

| Bytes | 1 | 1 | 1 | 1 or 2 | Variable | 2 or 4 | 1 |
|---|---|---|---|---|---|---|---|
| | Flag<br>01111110 | Address<br>11111111 | Control<br>00000011 | Protocol | Payload | Checksum | Flag<br>01111110 |

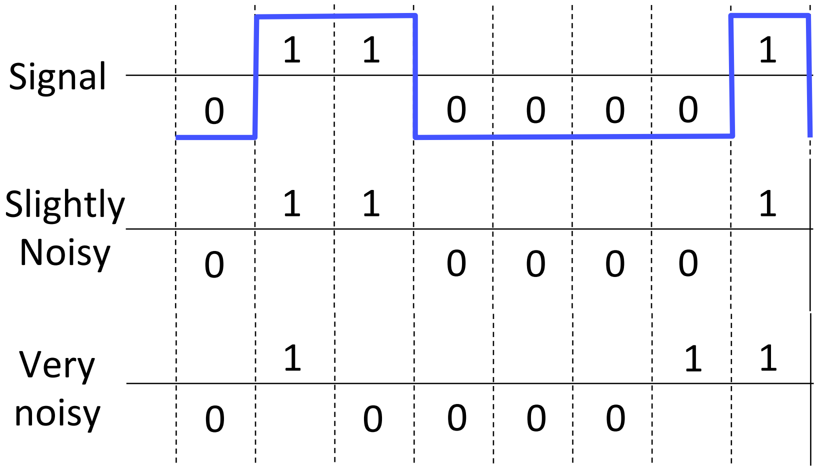
# Topic

- Some bits will be received in error due to noise. What can we do?
  - Detect errors with codes »
  - Correct errors with codes »
  - Retransmit lost frames

Later

- Reliability is a concern that cuts across the layers – we'll see it again

# Problem – Noise may flip received bits



# Approach – Add Redundancy

- Error detection codes
  - Add check bits to the message bits to let some errors be detected
- Error correction codes
  - Add more check bits to let some errors be corrected
- Key issue is now to structure the code to detect many errors with few check bits and modest computation

# Motivating Example

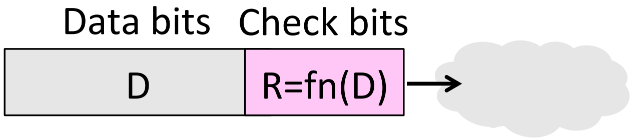
- A simple code to handle errors:
  - Send two copies! Error if different.
- How good is this code?
  - How many errors can it detect/correct?
  - How many errors will make it fail?

# Motivating Example (2)

- We want to handle more errors with less overhead
  - Will look at better codes; they are applied mathematics
  - But, they can't handle all errors
  - And they focus on accidental errors (will look at secure hashes later)

# Using Error Codes

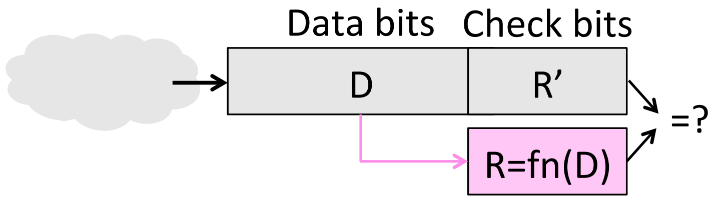
- Codeword consists of D data plus R check bits (=systematic block code)



- Sender:
  - Compute R check bits based on the D data bits; send the codeword of D+R bits

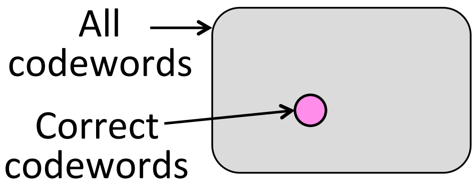
# Using Error Codes (2)

- Receiver:
  - Receive D+R bits with unknown errors
  - Recompute R check bits based on the D data bits; error if R doesn't match R'



# Intuition for Error Codes

- For D data bits, R check bits:



- Randomly chosen codeword is unlikely to be correct; overhead is low

## Hamming Distance

- Distance is the number of bit flips needed to change $D_1$ to $D_2$
- Hamming distance of a code is the minimum distance between any pair of codewords

## Hamming Distance (2)

- Error detection:
  - For a code of distance d+1, up to d errors will always be detected

# Hamming Distance (3)

- Alternatively, error correction:
  - For a code of distance 2d+1, up to d errors can always be corrected

# Introduction to Computer Networks

## Error Detection (§3.2.2)

Computer Science & Engineering

UNIVERSITY *of* WASHINGTON

# Topic

- Some bits may be received in error due to noise. How do we detect this?
  - Parity »
  - Checksums »
  - CRCs »
- Detection will let us fix the error, for example, by retransmission (later).

# Simple Error Detection – Parity Bit

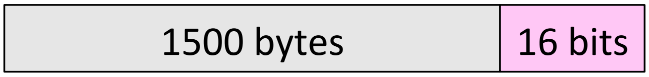
- Take D data bits, add 1 check bit that is the sum of the D bits
  - Sum is modulo 2 or XOR

# Parity Bit (2)

- How well does parity work?
  - What is the distance of the code?
  - How many errors will it detect/ correct?
- What about larger errors?

# Checksums

- Idea: sum up data in N-bit words
  - Widely used in, e.g., TCP/IP/UDP



- Stronger protection than parity

# Internet Checksum

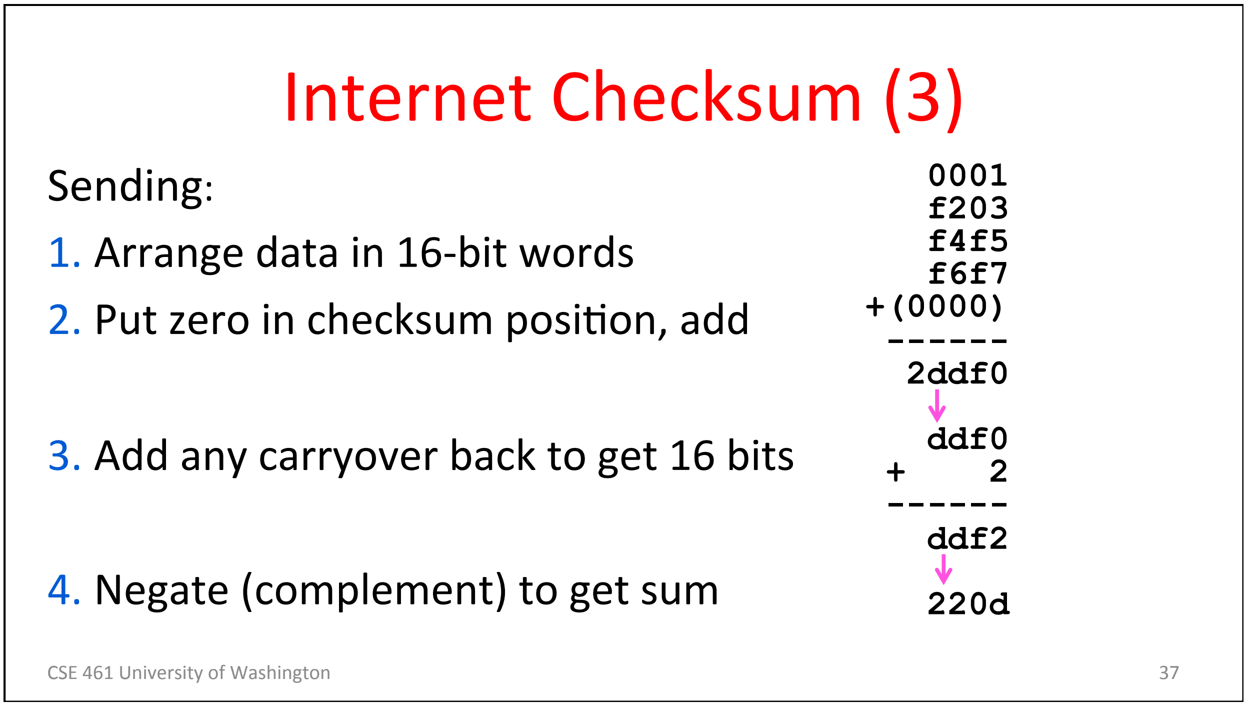
- Sum is defined in 1s complement arithmetic (must add back carries)
  - And it’s the negative sum
- *“The checksum field is the 16 bit one's complement of the one's complement sum of all 16 bit words …”* – RFC 791

# Internet Checksum (2)

Sending:

1. Arrange data in 16-bit words
2. Put zero in checksum position, add
3. Add any carryover back to get 16 bits
4. Negate (complement) to get sum

```
0001
f203
f4f5
f6f7
```

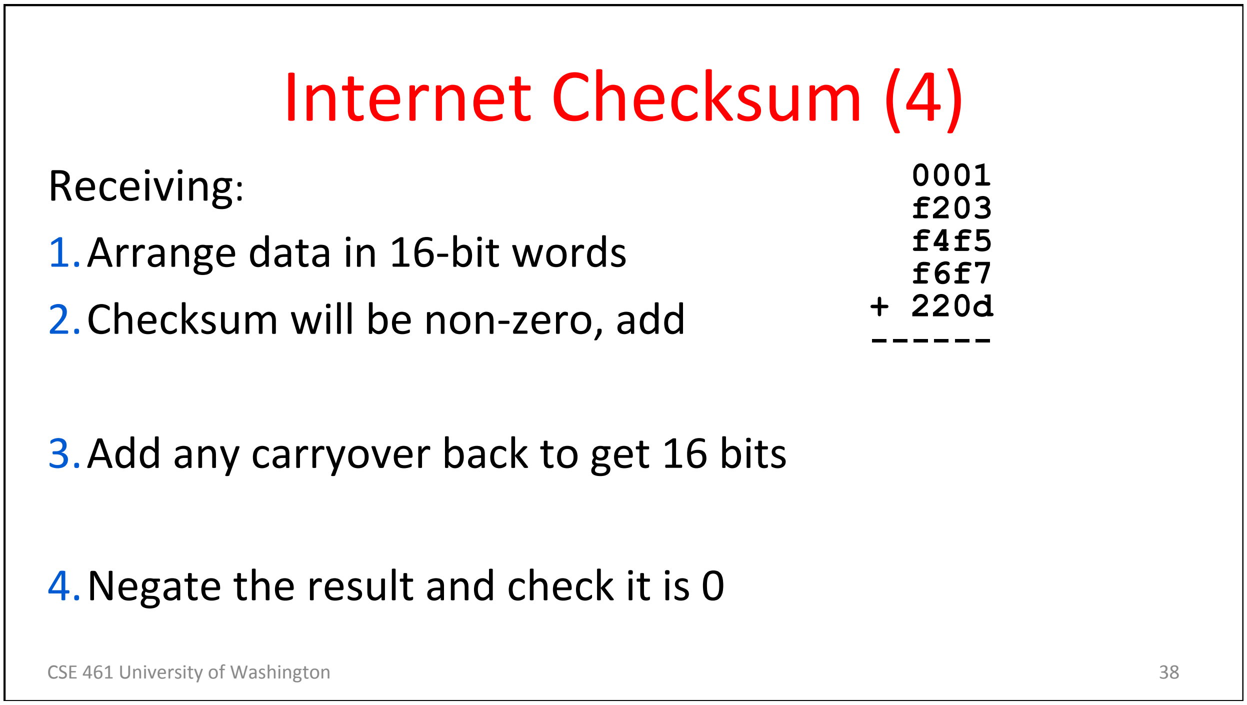


## Internet Checksum (3)

Sending:

1. Arrange data in 16-bit words
2. Put zero in checksum position, add
3. Add any carryover back to get 16 bits
4. Negate (complement) to get sum

```
   0001
   f203
   f4f5
   f6f7
+(0000)
 ------
  2ddf0
   ↓
   ddf0
 +    2
 ------
   ddf2
    ↓
   220d
```



## Internet Checksum (4)

Receiving:

1. Arrange data in 16-bit words
2. Checksum will be non-zero, add
3. Add any carryover back to get 16 bits
4. Negate the result and check it is 0

```
  0001
  f203
  f4f5
  f6f7
+ 220d
------
```

# Internet Checksum (5)

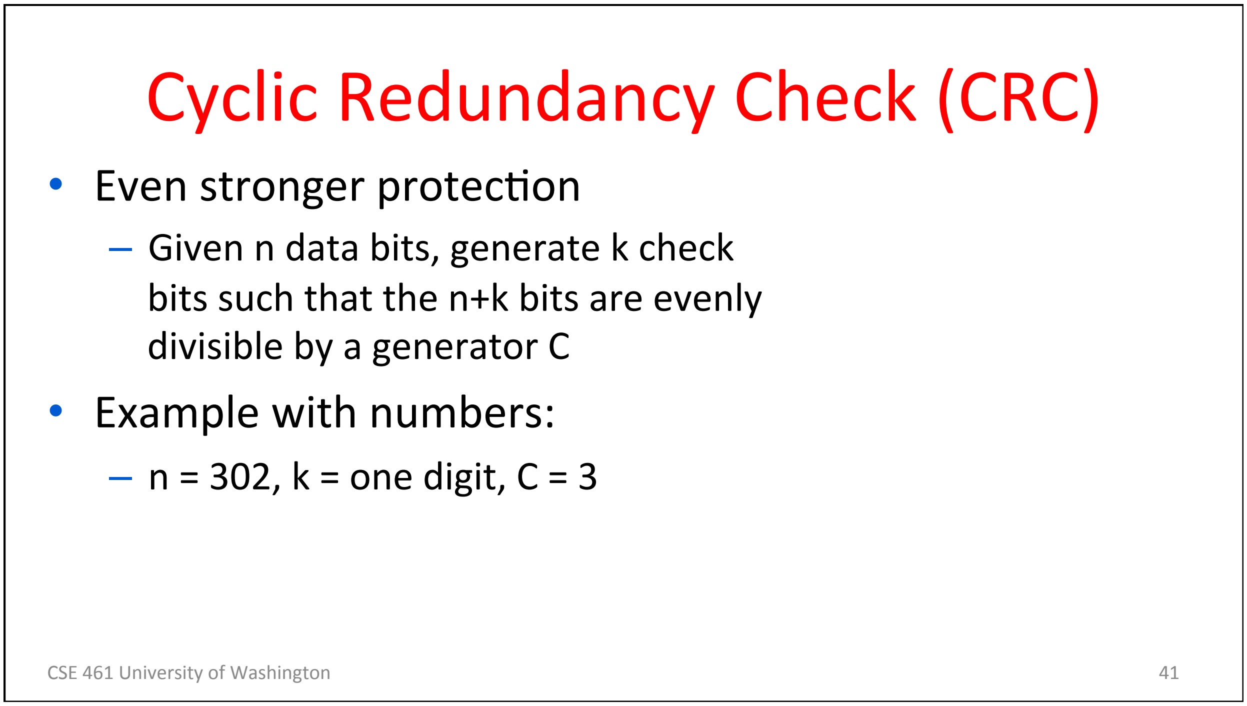
Receiving:

1. Arrange data in 16-bit words
2. Checksum will be non-zero, add
3. Add any carryover back to get 16 bits
4. Negate the result and check it is 0

```
   0001
   f203
   f4f5
   f6f7
 + 220d
 ------
  2fffd
   ↓
   fffd
 +    2
 ------
   ffff
    ↓
   0000
```

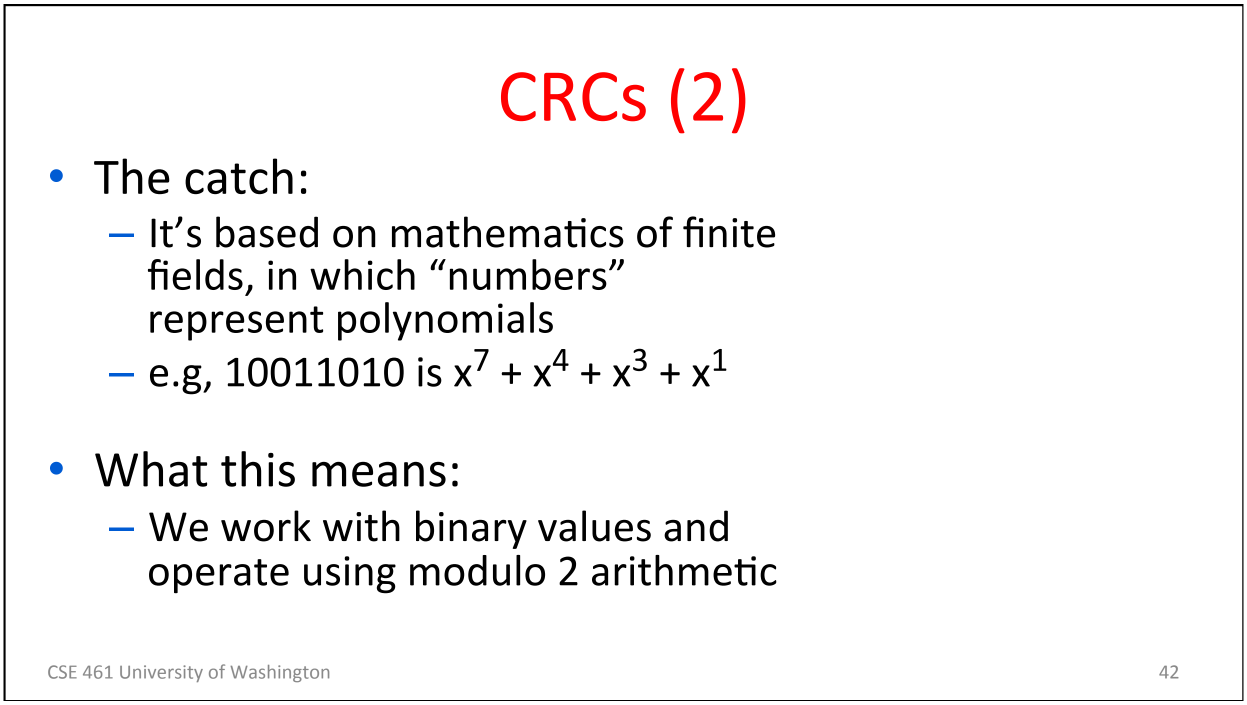
# Internet Checksum (6)

- How well does the checksum work?
  - What is the distance of the code?
  - How many errors will it detect/ correct?
- What about larger errors?



## Cyclic Redundancy Check (CRC)

- Even stronger protection
  - Given n data bits, generate k check bits such that the n+k bits are evenly divisible by a generator C
- Example with numbers:
  - n = 302, k = one digit, C = 3



## CRCs (2)

- The catch:
  - It's based on mathematics of finite fields, in which "numbers" represent polynomials
  - e.g, 10011010 is $x^7 + x^4 + x^3 + x^1$
- What this means:
  - We work with binary values and operate using modulo 2 arithmetic

# CRCs (2)

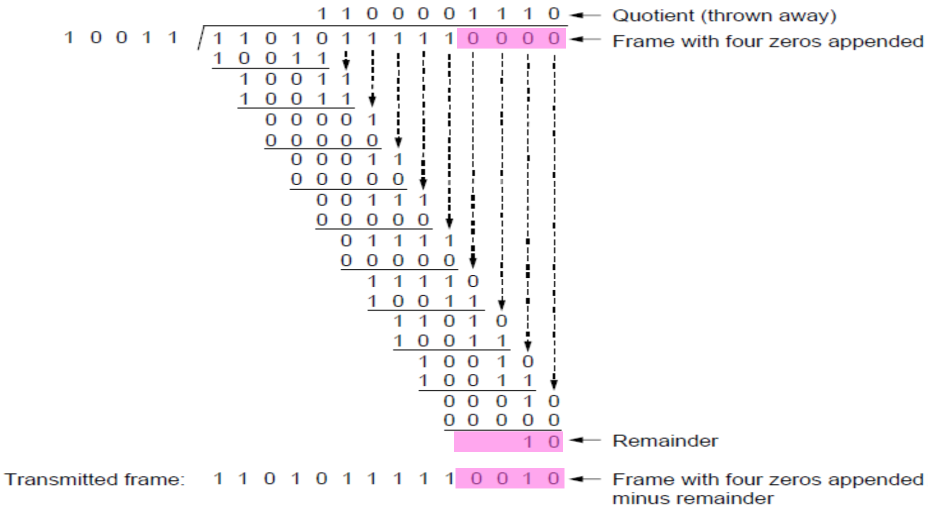
- Send Procedure:

1. Extend the n data bits with k zeros
2. Divide by the generator value C
3. Keep remainder, ignore quotient
4. Adjust k check bits by remainder

- Receive Procedure:

1. Divide and check for zero remainder

# CRCs (3)



```
                    1 1 0 0 0 0 1 1 1 0
1 0 0 1 1 / 1 1 0 1 0 1 1 1 1 1 0 0 0 0
            1 0 0 1 1
              1 0 0 1 1
              1 0 0 1 1
                0 0 0 0 1
                0 0 0 0 0
                  0 0 0 1 1
                  0 0 0 0 0
                    0 0 1 1 1
                    0 0 0 0 0
                      0 1 1 1 1
                      0 0 0 0 0
                        1 1 1 1 0
                        1 0 0 1 1
                          1 1 0 1 0
                          1 0 0 1 1
                            1 0 0 1 0
                            1 0 0 1 1
                              0 0 0 1 0
                              0 0 0 0 0
                                    1 0

Transmitted frame: 1 1 0 1 0 1 1 1 1 1 0 0 1 0
```

Quotient (thrown away)
Frame with four zeros appended
Remainder
Frame with four zeros appended minus remainder

# CRCs (4)

- Protection depend on generator
  - Standard CRC-32 is 10000010 01100000 10001110 110110111
    - »
- Properties:
  - HD=4, detects up to triple bit errors
  - Also odd number of errors
  - And bursts of up to k bits in error
  - Not vulnerable to systematic errors like checksums

# Error Detection in Practice

- CRCs are widely used on links
  - Ethernet, 802.11, ADSL, Cable …
- Checksum used in Internet
  - IP, TCP, UDP … but it is weak
- Parity
  - Is little used

# Introduction to Computer Networks

Error Correction (§3.2.3)

Computer Science & Engineering

UNIVERSITY *of* WASHINGTON

# Topic

- Some bits may be received in error due to noise. How do we fix them?
  - Hamming code »
  - Other codes »
- And why should we use detection when we can use correction?

## Why Error Correction is Hard

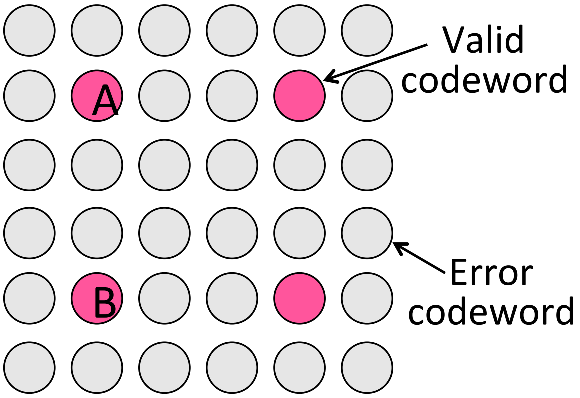
- If we had reliable check bits we could use them to narrow down the position of the error
  - Then correction would be easy
- But error could be in the check bits as well as the data bits!
  - Data might even be correct

## Intuition for Error Correcting Code

- Suppose we construct a code with a Hamming distance of at least 3
  - Need ≥3 bit errors to change one valid codeword into another
  - Single bit errors will be closest to a unique valid codeword
- If we assume errors are only 1 bit, we can correct them by mapping an error to the closest valid codeword
  - Works for d errors if HD ≥ 2d 1

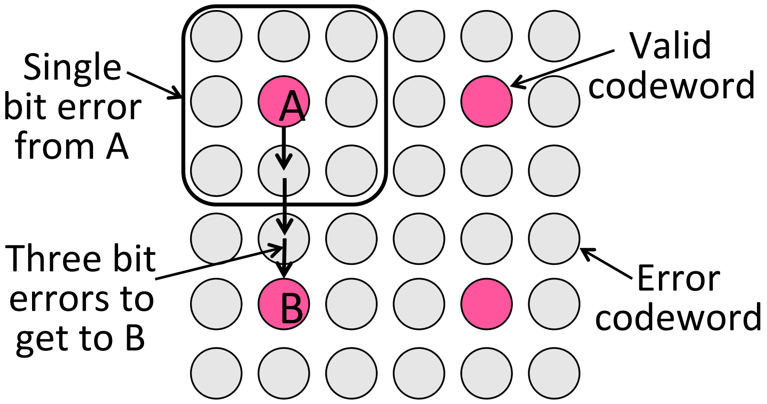
## Intuition (2)

- Visualization of code:



## Intuition (3)

- Visualization of code:



# Hamming Code

- Gives a method for constructing a code with a distance of 3
  - Uses k check bits for $2^{k-1}$ data bits
  - Put check bits in positions p that are powers of 2, starting with position 1
  - Check bit in position p is parity of positions with a p term in their values
- Plus an easy way to correct [soon]

# Hamming Code (2)

- Example: data=0101, 3 check bits
  - 7 bit code, check bit positions 1, 2, 4
  - Check 1 covers positions 1, 3, 5, 7
  - Check 2 covers positions 2, 3, 6, 7
  - Check 4 covers positions 4, 5, 6, 7

_ _ _ _ _ _ _
1 2 3 4 5 6 7

# Hamming Code (3)

- Example: data=0101, 3 check bits
  - 7 bit code, check bit positions 1, 2, 4
  - Check 1 covers positions 1, 3, 5, 7
  - Check 2 covers positions 2, 3, 6, 7
  - Check 4 covers positions 4, 5, 6, 7

$$\begin{array}{ccccccc} \underline{0} & \underline{1} & 0 & \underline{0} & 1 & 0 & 1 \\ 1 & 2 & 3 & 4 & 5 & 6 & 7 \end{array} \longrightarrow$$

$p_1$= 0+1+1 = 0, $p_2$= 0+0+1 = 1, $p_4$= 1+0+1 = 0

# Hamming Code (4)

- To decode:
  - Recompute check bits (with parity sum including the check bit)
  - Arrange as a binary number
  - Value (syndrome) tells error position
  - Value of zero means no error
  - Otherwise, flip bit to correct

# Hamming Code (5)

- Example, continued

$$\longrightarrow \begin{array}{ccccccc} \underline{0} & \underline{1} & 0 & \underline{0} & 1 & 0 & 1 \\ 1 & 2 & 3 & 4 & 5 & 6 & 7 \end{array}$$

$p_1$= $p_2$=

$p_4$=

Syndrome =

Data =

# Hamming Code (6)

- Example, continued

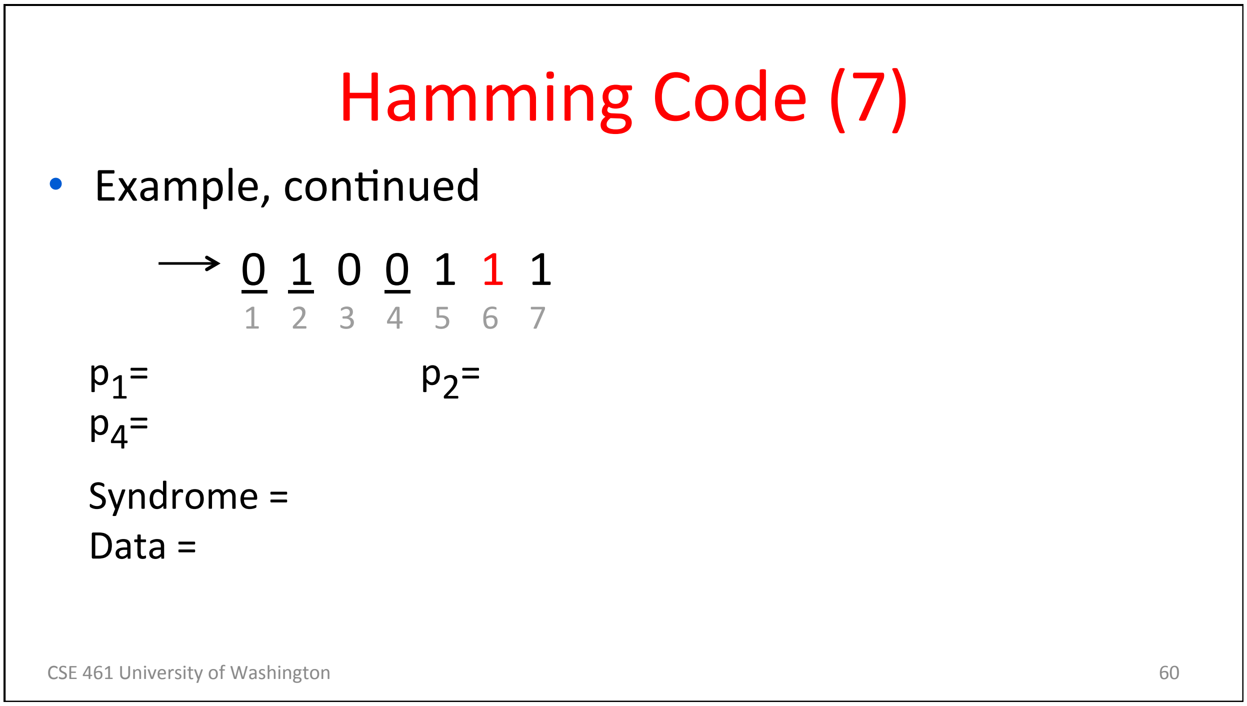
$$\longrightarrow \begin{array}{ccccccc} \underline{0} & \underline{1} & 0 & \underline{0} & 1 & 0 & 1 \\ 1 & 2 & 3 & 4 & 5 & 6 & 7 \end{array}$$

$p_1$= 0+0+1+1 = 0, $p_2$= 1+0+0+1 = 0,

$p_4$= 0+1+0+1 = 0

Syndrome = 000, no error

Data = 0 1 0 1



# Hamming Code (7)

- Example, continued

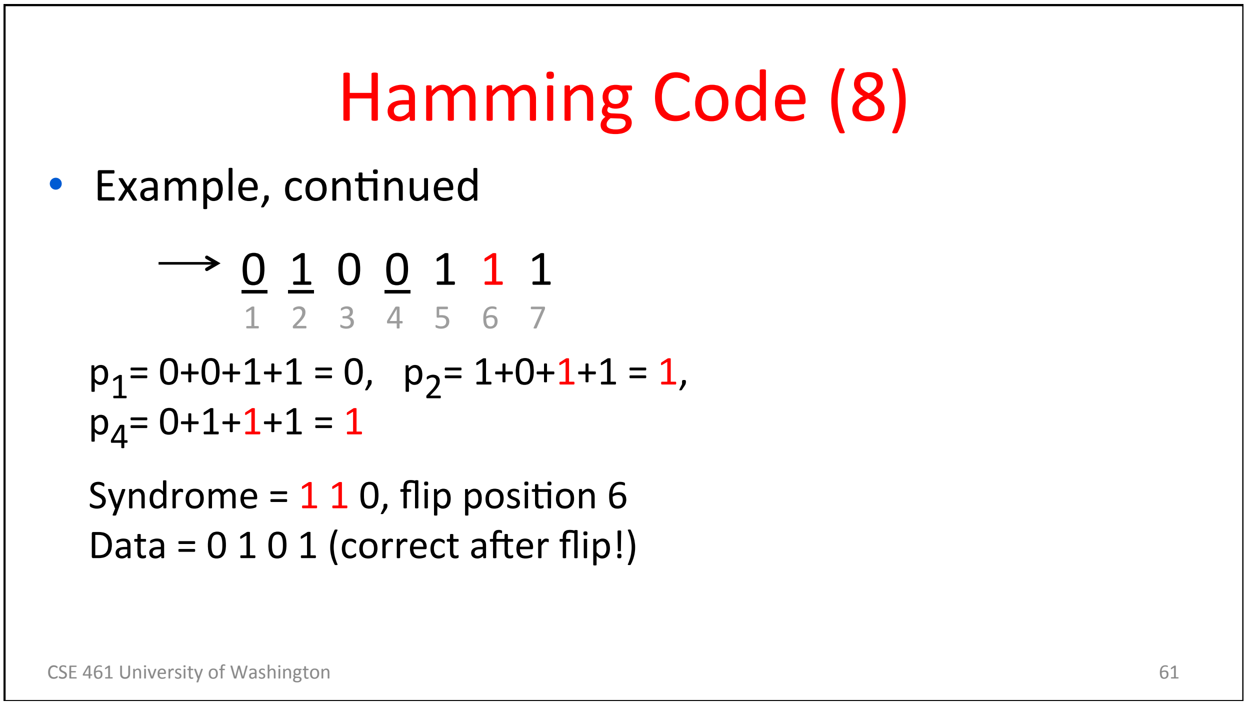
$$\longrightarrow \begin{array}{ccccccc} \underline{0} & \underline{1} & 0 & \underline{0} & 1 & 1 & 1 \\ 1 & 2 & 3 & 4 & 5 & 6 & 7 \end{array}$$

$p_1$= $p_2$=

$p_4$=

Syndrome =

Data =



# Hamming Code (8)

- Example, continued

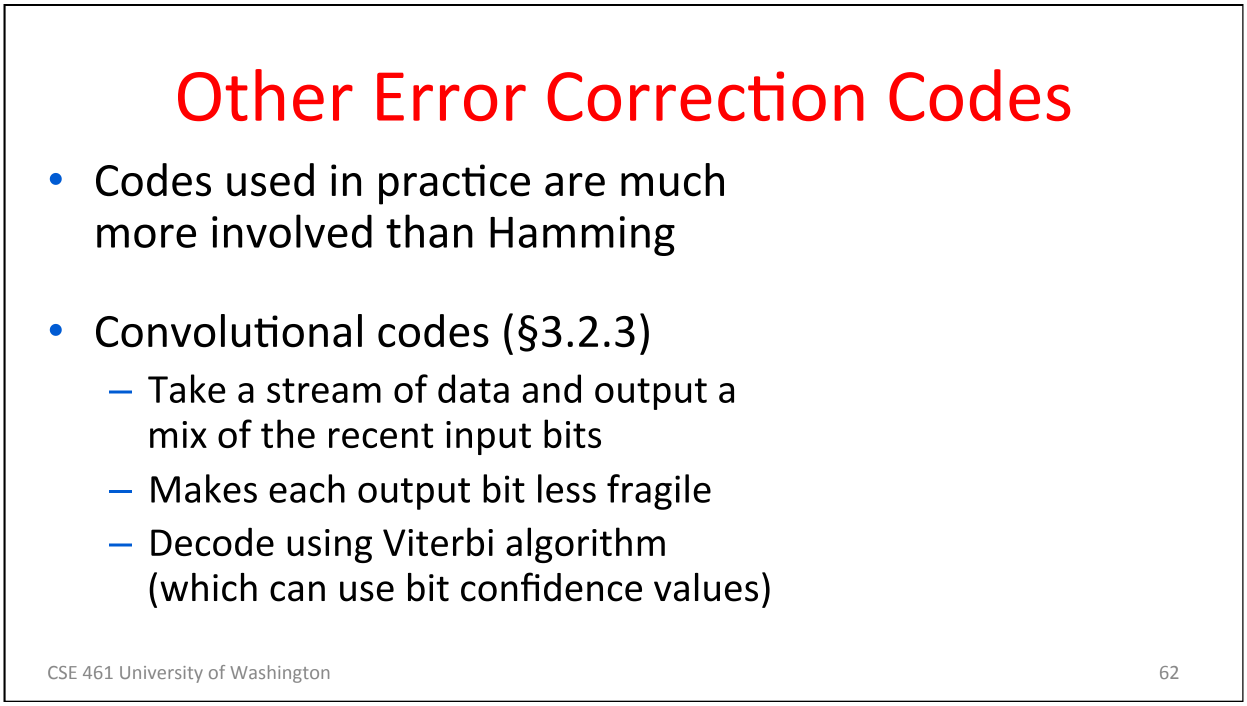
$$\longrightarrow \begin{array}{ccccccc} \underline{0} & \underline{1} & 0 & \underline{0} & 1 & 1 & 1 \\ 1 & 2 & 3 & 4 & 5 & 6 & 7 \end{array}$$

$p_1$= 0+0+1+1 = 0, $p_2$= 1+0+1+1 = 1,

$p_4$= 0+1+1+1 = 1

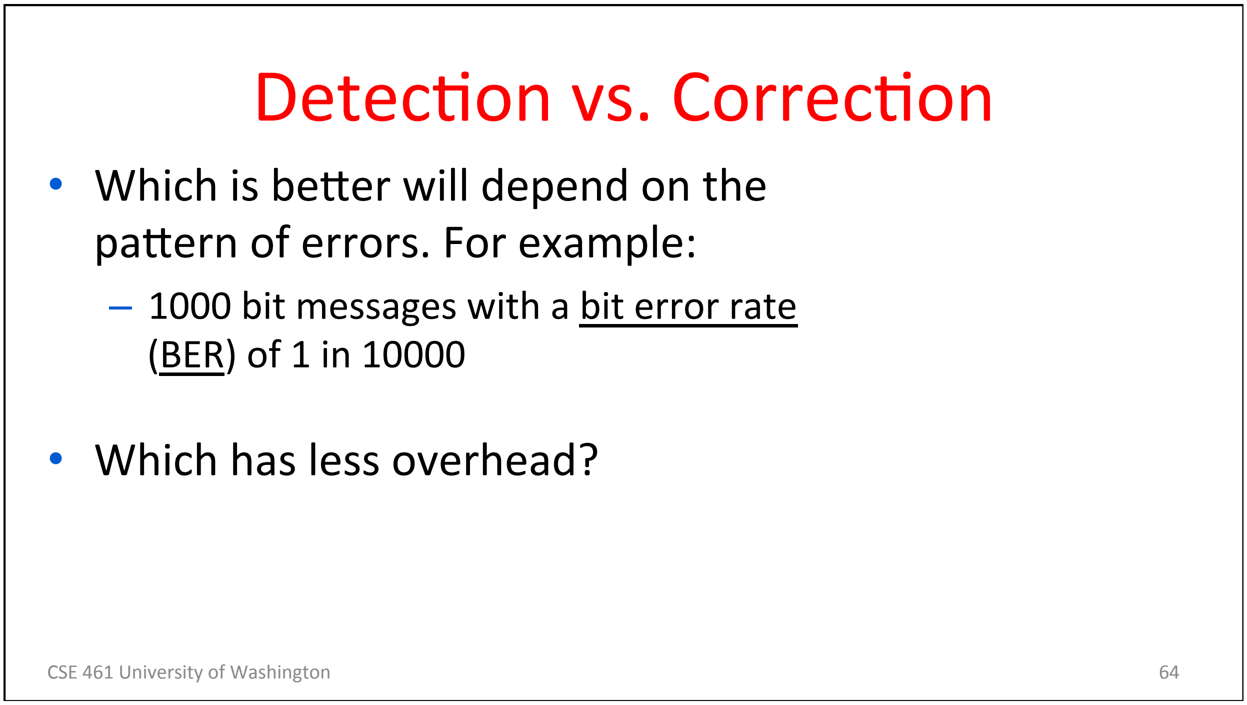
Syndrome = 1 1 0, flip position 6

Data = 0 1 0 1 (correct after flip!)



## Other Error Correction Codes

- Codes used in practice are much more involved than Hamming
- Convolutional codes (§3.2.3)
  - Take a stream of data and output a mix of the recent input bits
  - Makes each output bit less fragile
  - Decode using Viterbi algorithm (which can use bit confidence values)



## Detection vs. Correction

- Which is better will depend on the pattern of errors. For example:
  - 1000 bit messages with a bit error rate (BER) of 1 in 10000
- Which has less overhead?

# Detection vs. Correction (2)

- Assume bit errors are random
  - Messages have 0 or maybe 1 error
- Error correction:
  - Need ~10 check bits per message
  - Overhead:
- Error detection:
  - Need ~1 check bits per message plus 1000 bit retransmission 1/10 of the time
  - Overhead:

# Detection vs. Correction (3)

- Assume errors come in bursts of 100
  - Only 1 or 2 messages in 1000 have errors
- Error correction:
  - Need >>100 check bits per message
  - Overhead:
- Error detection:
  - Need 32? check bits per message plus 1000 bit resend 2/1000 of the time
  - Overhead:

# Detection vs. Correction (4)

- Error correction:
  - Needed when errors are expected
  - Or when no time for retransmission
- Error detection:
  - More efficient when errors are not expected
  - And when errors are large when they do occur

# Error Correction in Practice

- Heavily used in physical layer
  - Convolutional codes widely used in practice
  - LDPC is the future, used for demanding links like 802.11, DVB, WiMAX, LTE, power-line, …
- Error detection (w/ retransmission) is used in the link layer and above for residual errors
- Also used in the application layer
  - With an erasure error model
  - E.g., Reed-Solomon (CDs, DVDs, etc.)

# Introduction to Computer Networks

Retransmissions (ARQ) (§3.3)

Computer Science & Engineering

UNIVERSITY *of* WASHINGTON

# Topic

- Two strategies to handle errors:

1. Detect errors and retransmit frame (Automatic Repeat reQuest, ARQ)
2. Correct errors with an error correcting code

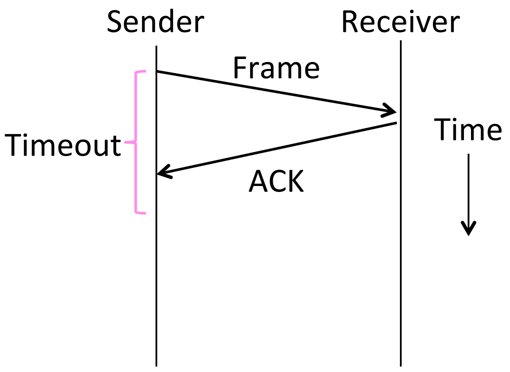
Done this

# ARQ

- ARQ often used when errors are common or must be corrected
  - E.g., WiFi, and TCP (later)
- Rules at sender and receiver:
  - Receiver automatically acknowledges correct frames with an ACK
  - Sender automatically resends after a timeout, until an ACK is received

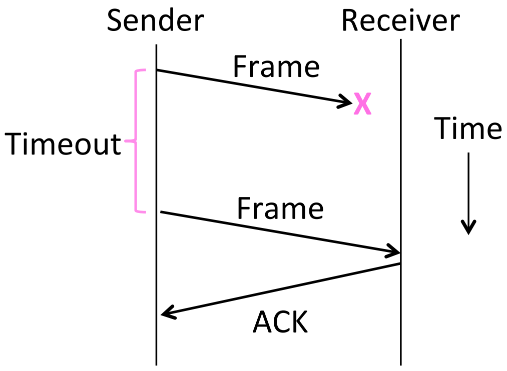
# ARQ (2)

- Normal operation (no loss)



# ARQ (3)

- Loss and retransmission



# So What’s Tricky About ARQ?

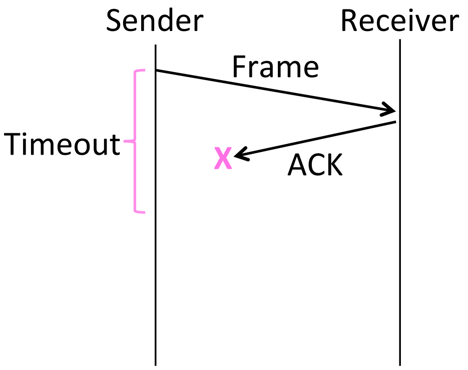
- Two non-trivial issues:
  - How long to set the timeout? »
  - How to avoid accepting duplicate frames as new frames »
- Want performance in the common case and correctness always

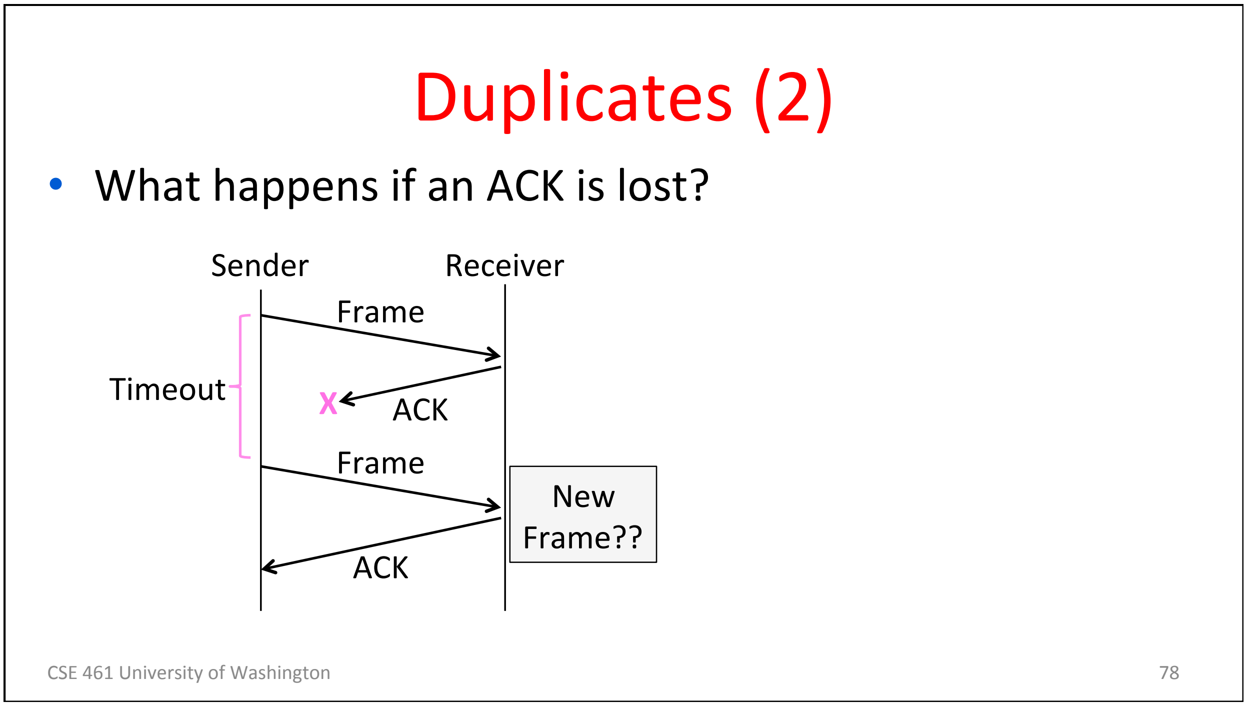
# Timeouts

- Timeout should be:
  - Not too big (link goes idle)
  - Not too small (spurious resend)
- Fairly easy on a LAN
  - Clear worst case, little variation
- Fairly difficult over the Internet
  - Much variation, no obvious bound
  - We'll revisit this with TCP (later)

# Duplicates

- What happens if an ACK is lost?

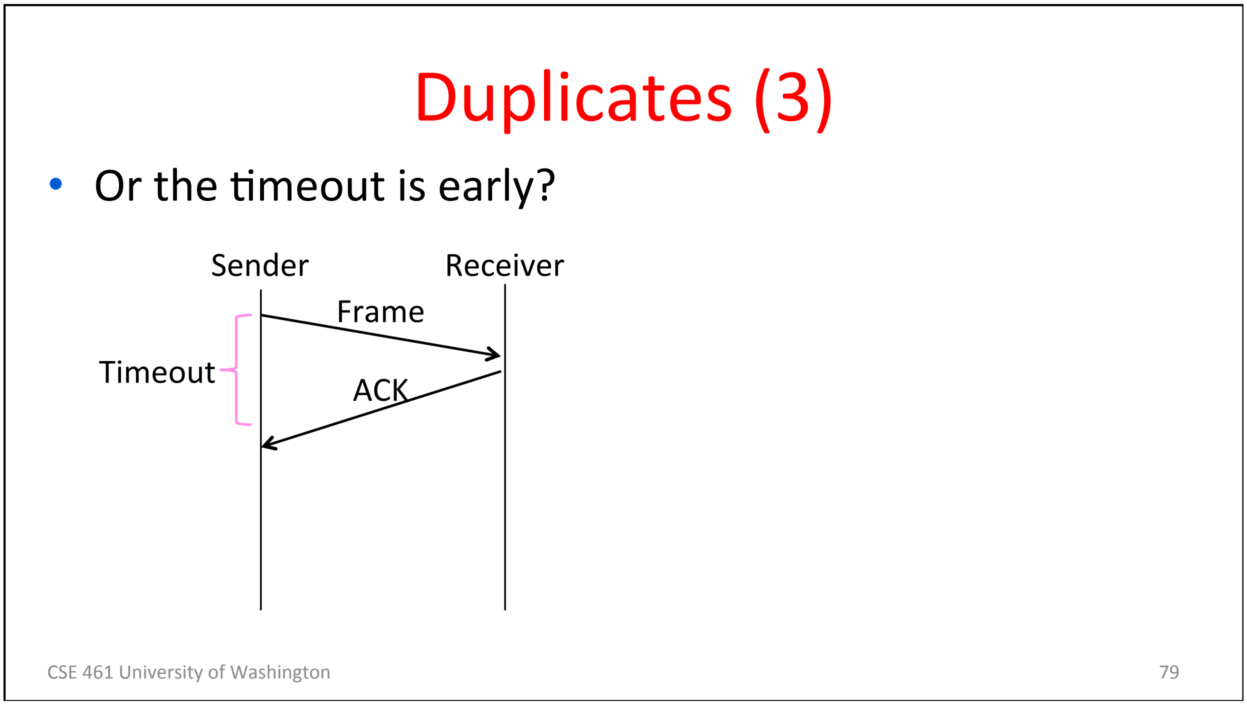




# Duplicates (2)

- What happens if an ACK is lost?

CSE 461 University of Washington 78



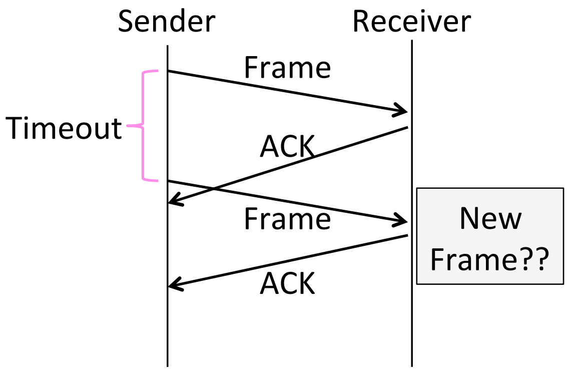
# Duplicates (3)

- Or the timeout is early?

CSE 461 University of Washington 79

# Duplicates (4)

- Or the timeout is early?

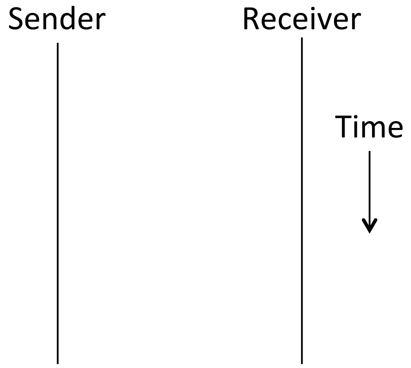


# Sequence Numbers

- Frames and ACKs must both carry sequence numbers for correctness
- To distinguish the current frame from the next one, a single bit (two numbers) is sufficient
  - Called Stop-and-Wait

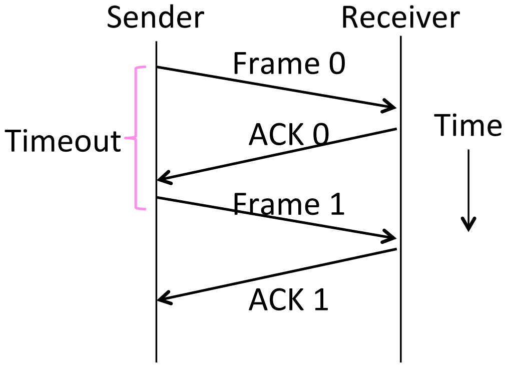
# Stop-and-Wait

- In the normal case:



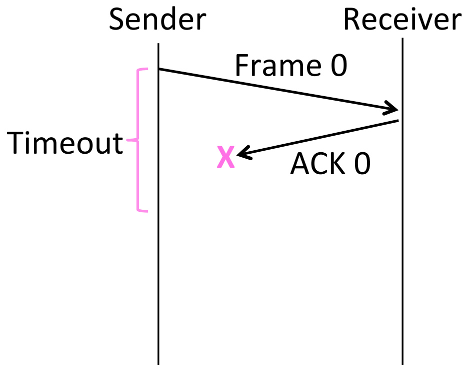
# Stop-and-Wait (2)

- In the normal case:



# Stop-and-Wait (3)

- With ACK loss:



Sender

Receiver

Frame 0

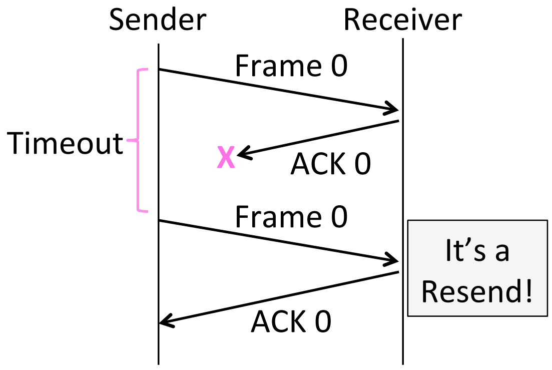
Timeout

X

ACK 0

# Stop-and-Wait (4)

- With ACK loss:



Sender

Receiver

Frame 0

Timeout

X

ACK 0

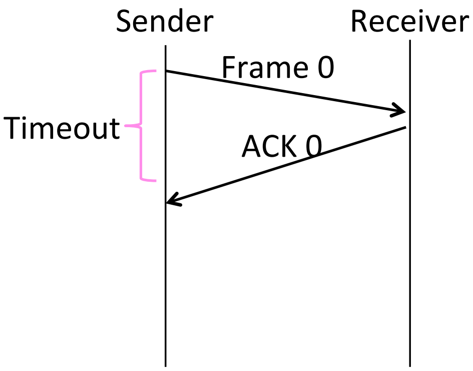
Frame 0

It’s a Resend!

ACK 0

# Stop-and-Wait (5)

- With early timeout:



Sender

Receiver

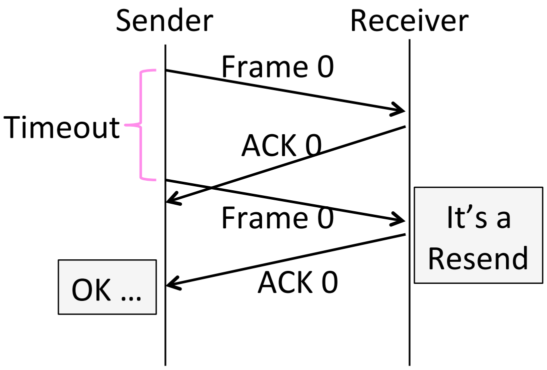
Frame 0

Timeout

ACK 0

# Stop-and-Wait (6)

- With early timeout:



Sender

Receiver

Frame 0

Timeout

ACK 0

Frame 0

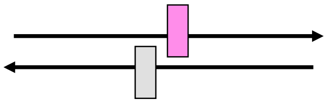
It's a Resend

OK …

ACK 0

# Limitation of Stop-and-Wait

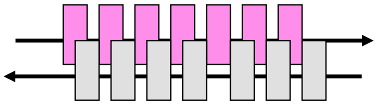
- It allows only a single frame to be outstanding from the sender:
  - Good for LAN, not efficient for high BD



- Ex: R=1 Mbps, D = 50 ms
  - How many frames/sec? If R=10 Mbps?

# Sliding Window

- Generalization of stop-and-wait
  - Allows W frames to be outstanding
  - Can send W frames per RTT



  - Various options for numbering frames/ACKs and handling loss
    - Will look at along with TCP (later)

# Introduction to Computer Networks

Multiplexing(§2.5.3, 2.5.4)

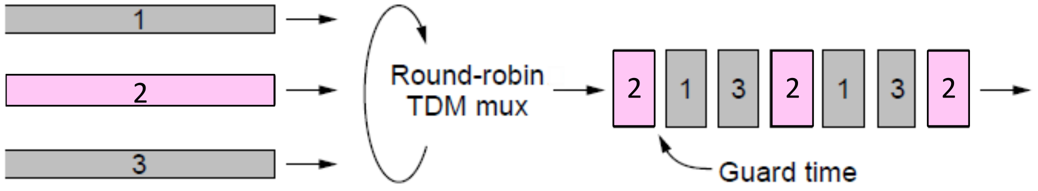
Computer Science & Engineering
UNIVERSITY *of* WASHINGTON

## Topic

- Multiplexing is the network word for the sharing of a resource
- Classic scenario is sharing a link among different users
  - Time Division Multiplexing (TDM) »
  - Frequency Division Multiplexing (FDM) »

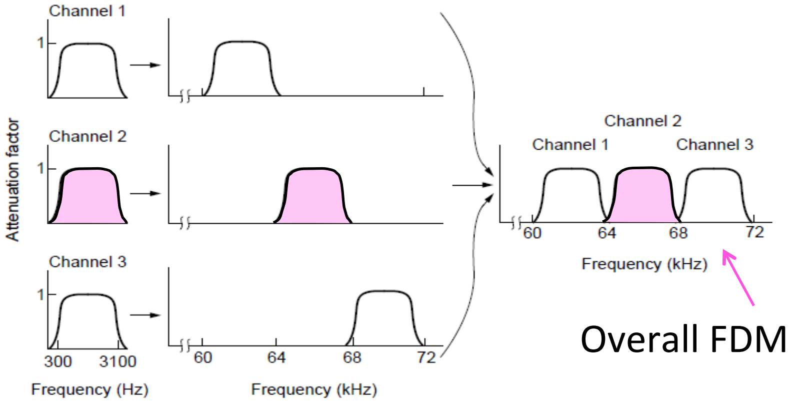
# Time Division Multiplexing (TDM)

- Users take turns on a fixed schedule



# Frequency Division Multiplexing (FDM)

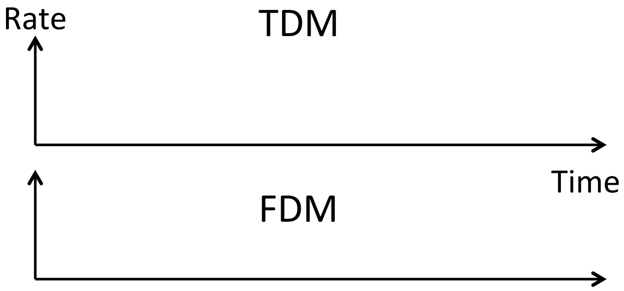
- Put different users on different frequency bands



Overall FDM channel

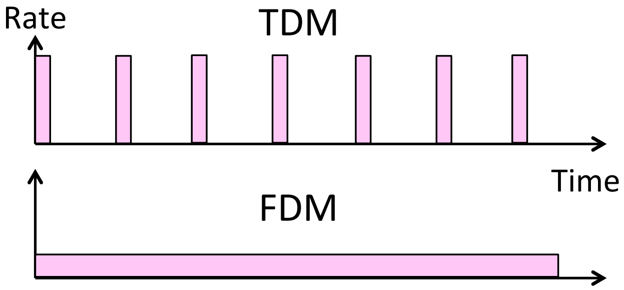
# TDM versus FDM

- In TDM a user sends at a high rate a fraction of the time; in FDM, a user sends at a low rate all the time



# TDM versus FDM (2)

- In TDM a user sends at a high rate a fraction of the time; in FDM, a user sends at a low rate all the time

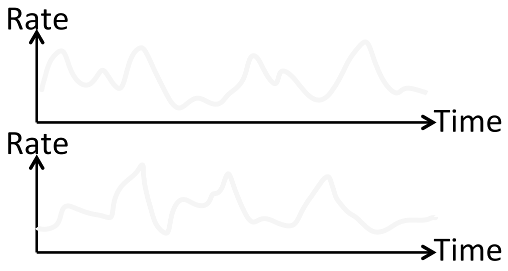


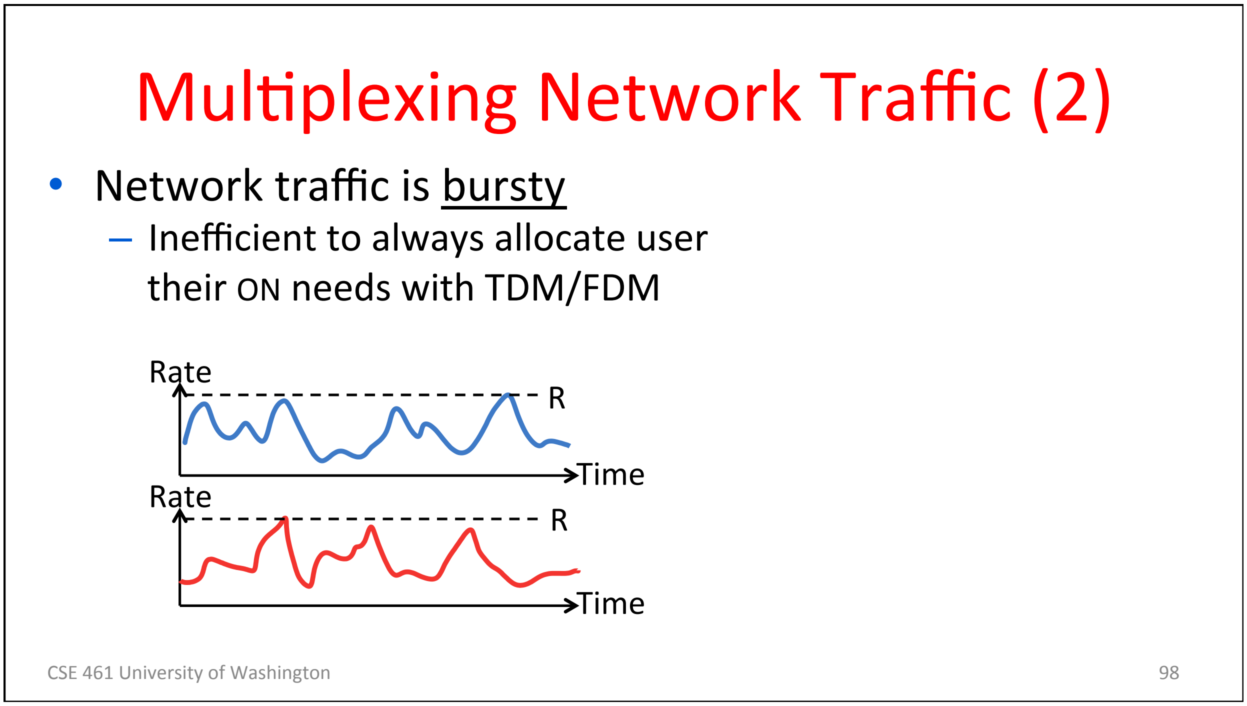
# TDM/FDM Usage

- Statically divide a resource
  - Suited for continuous traffic, fixed number of users
- Widely used in telecommunications
  - TV and radio stations (FDM)
  - GSM (2G cellular) allocates calls using TDM within FDM

# Multiplexing Network Traffic

- Network traffic is <u>bursty</u>
  - ON/OFF sources
  - Load varies greatly over time

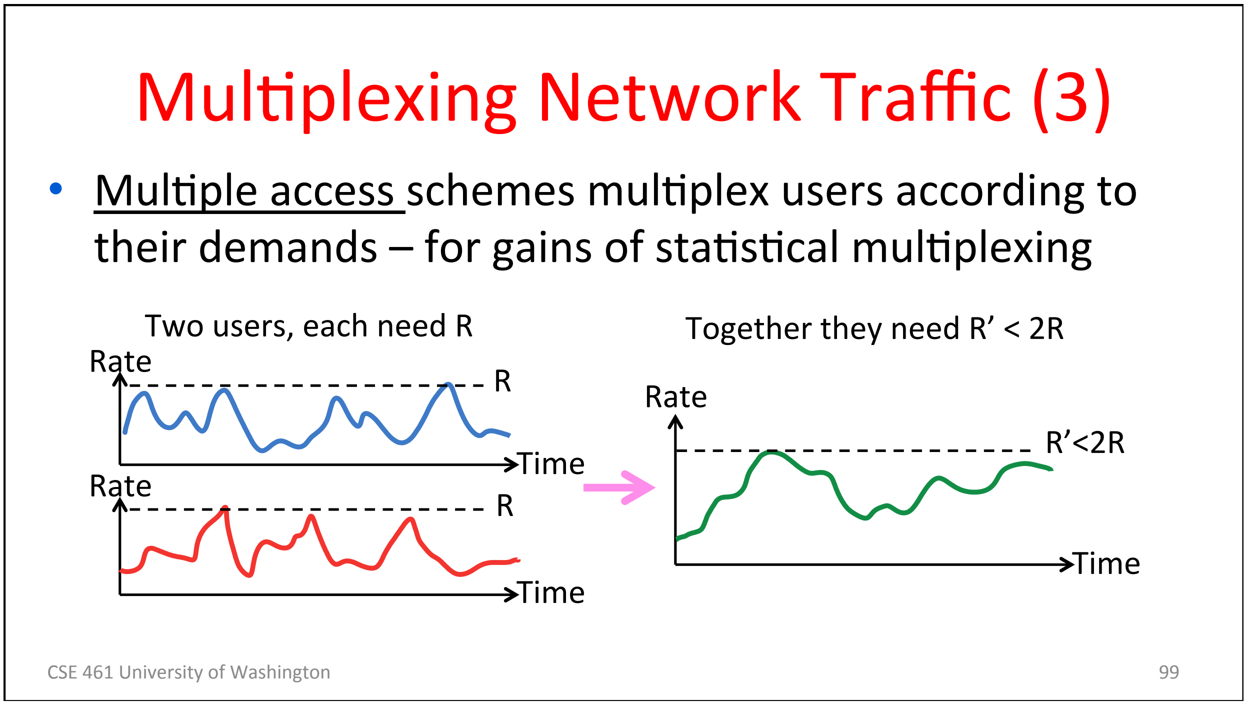




# Multiplexing Network Traffic (2)

- Network traffic is bursty
  - Inefficient to always allocate user their ON needs with TDM/FDM

Rate
R
Time
Rate
R
Time



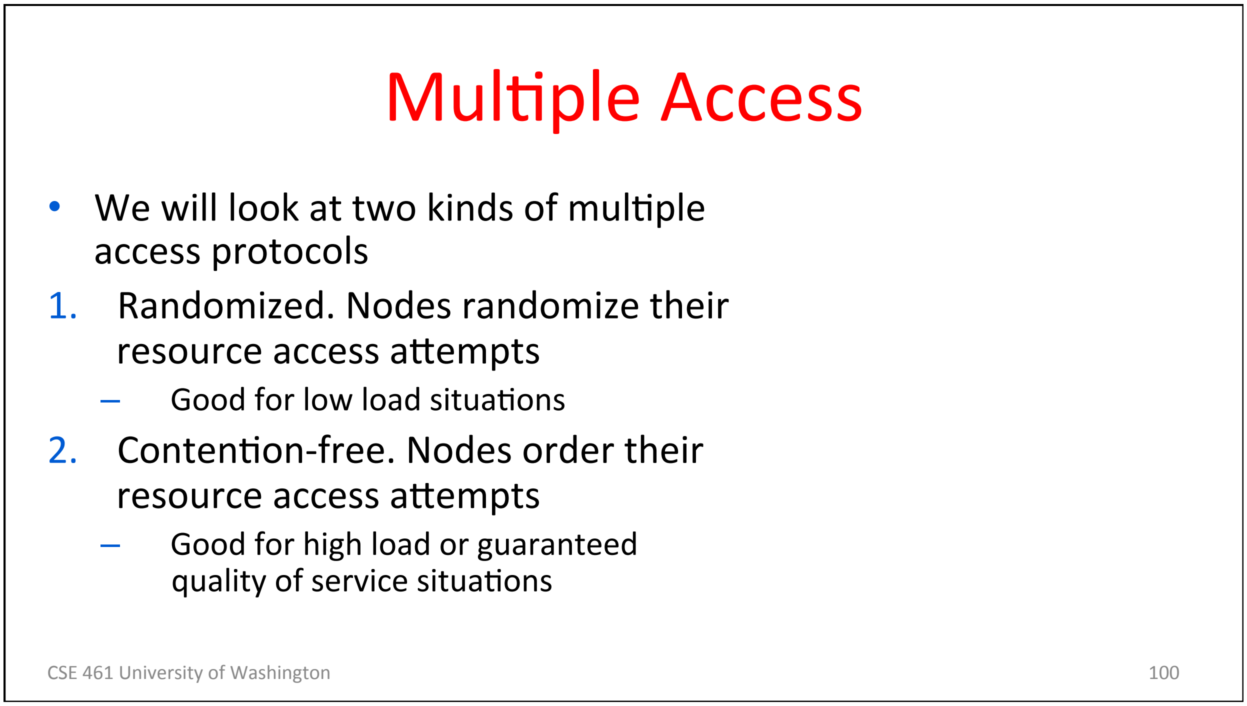
# Multiplexing Network Traffic (3)

- Multiple access schemes multiplex users according to their demands – for gains of statistical multiplexing

Two users, each need R

Together they need R’ < 2R

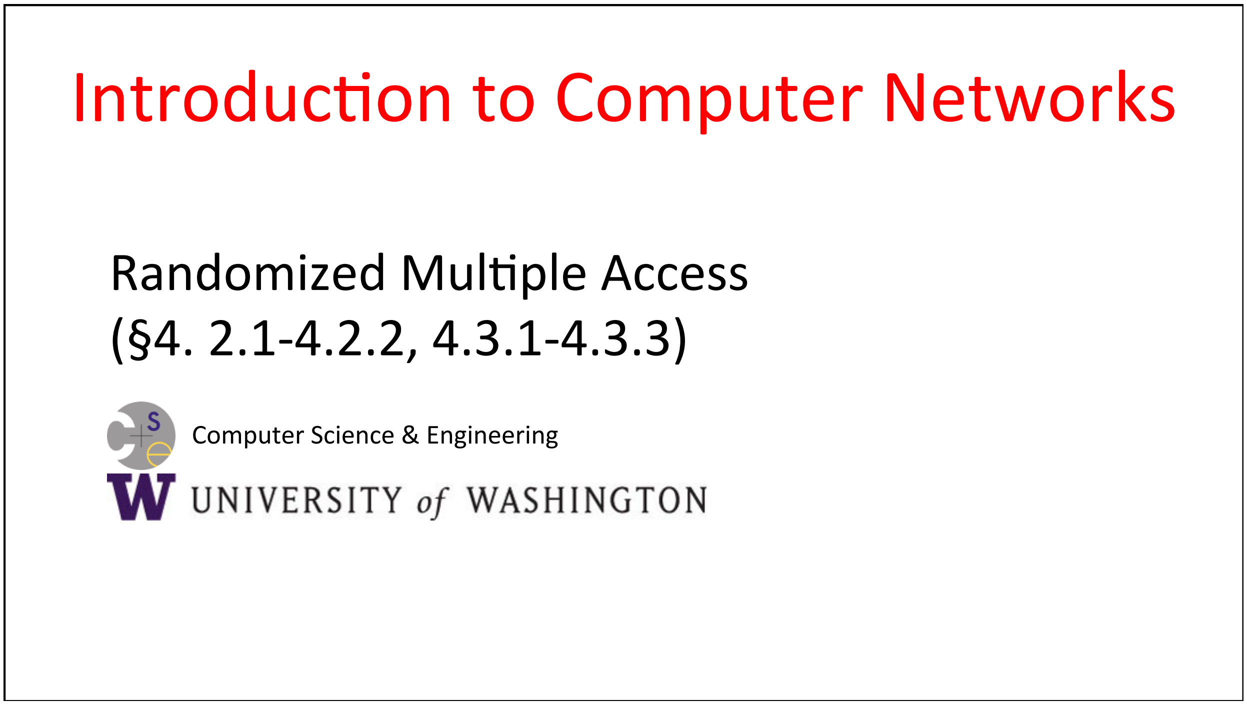
Rate
R
Time
Rate
R
Time
Rate
R’<2R
Time



# Multiple Access

- We will look at two kinds of multiple access protocols

1. Randomized. Nodes randomize their resource access attempts
   – Good for low load situations
2. Contention-free. Nodes order their resource access attempts
   – Good for high load or guaranteed quality of service situations



# Introduction to Computer Networks

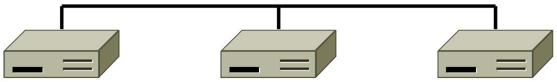
Randomized Multiple Access
(§4. 2.1-4.2.2, 4.3.1-4.3.3)

Computer Science & Engineering

UNIVERSITY *of* WASHINGTON

# Topic

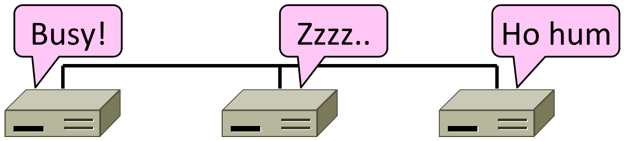
- How do nodes share a single link? Who sends when, e.g., in WiFI?
  - Explore with a simple model



- Assume no-one is in charge; this is a distributed system

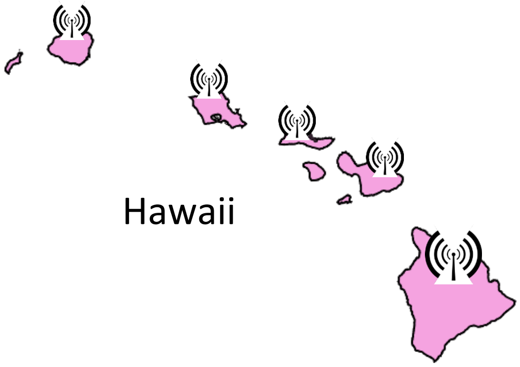
# Topic (2)

- We will explore random multiple access control (MAC) protocols
  - This is the basis for classic Ethernet
  - Remember: data traffic is bursty



# ALOHA Network

- Seminal computer network connecting the Hawaiian islands in the late 1960s
  - When should nodes send?
  - A new protocol was devised by Norm Abramson …

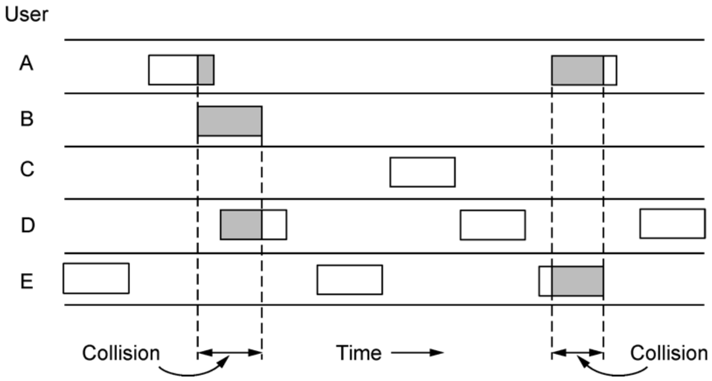


# ALOHA Protocol

- Simple idea:
  - Node just sends when it has traffic.
  - If there was a collision (no ACK received) then wait a random time and resend
- That's it!

# ALOHA Protocol (2)

- Some frames will be lost, but many may get through…
- Good idea?

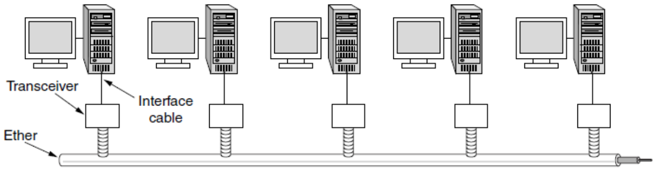


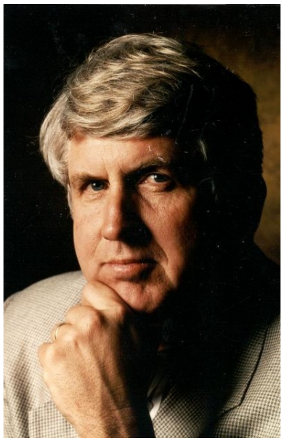
# ALOHA Protocol (3)

- Simple, decentralized protocol that works well under low load!
- Not efficient under high load
  - Analysis shows at most 18% efficiency
  - Improvement: divide time into slots and efficiency goes up to 36%
- We'll look at other improvements

# Classic Ethernet

- ALOHA inspired Bob Metcalfe to invent Ethernet for LANs in 1973
  - Nodes share 10 Mbps coaxial cable
  - Hugely popular in 1980s, 1990s

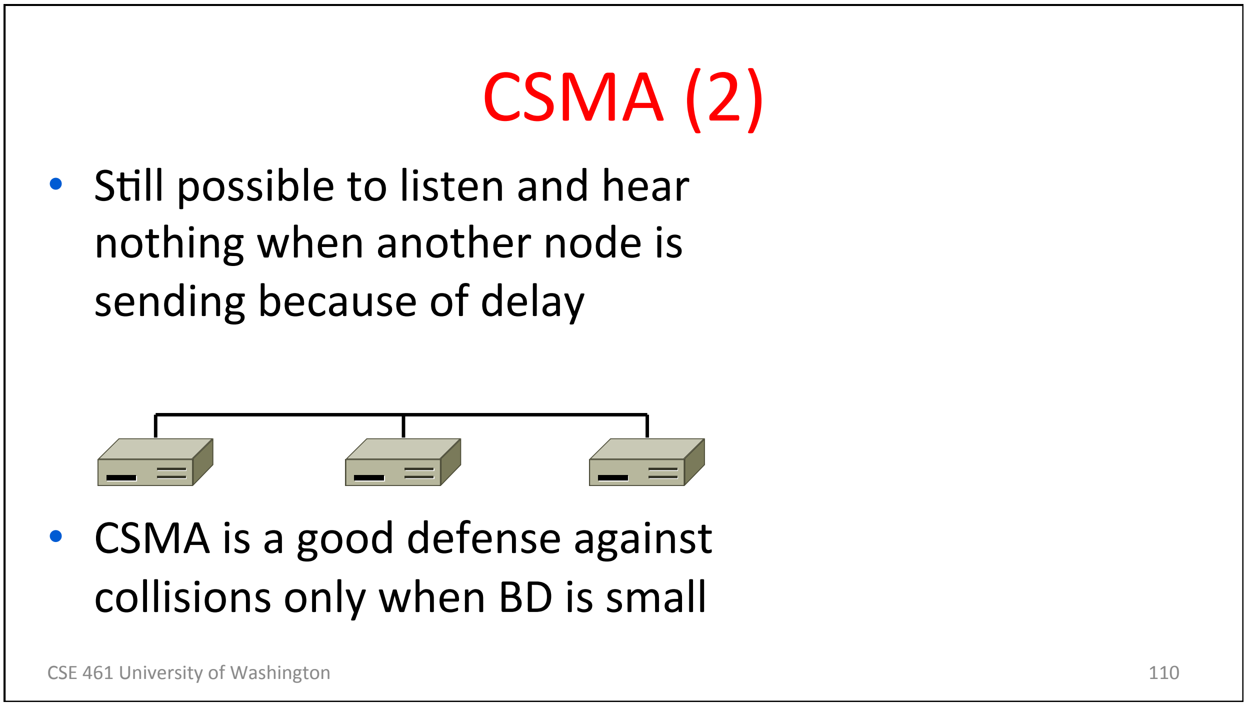




: © 2009 IEEE

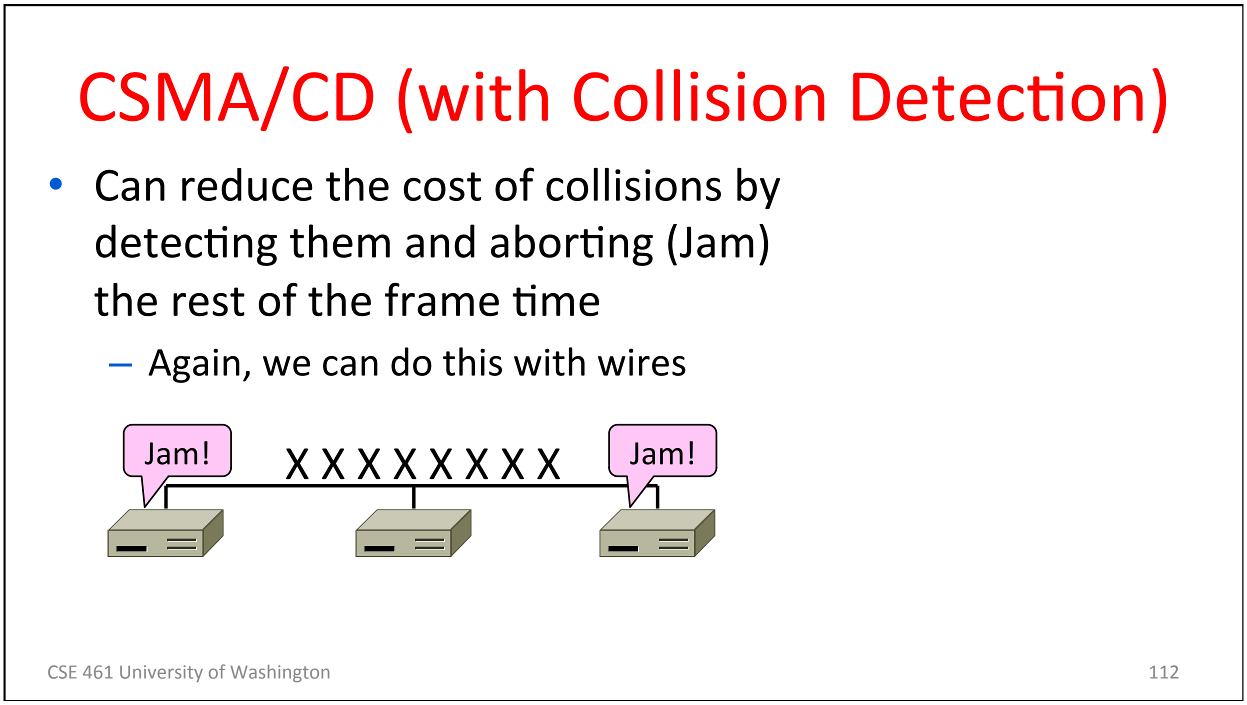
# CSMA (Carrier Sense Multiple Access)

- Improve ALOHA by listening for activity before we send (Doh!)
  - Can do easily with wires, not wireless
- So does this eliminate collisions?
  - Why or why not?



# CSMA (2)

- Still possible to listen and hear nothing when another node is sending because of delay
- CSMA is a good defense against collisions only when BD is small

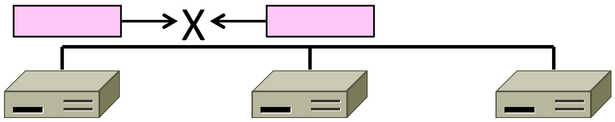


# CSMA/CD (with Collision Detection)

- Can reduce the cost of collisions by detecting them and aborting (Jam) the rest of the frame time
  – Again, we can do this with wires

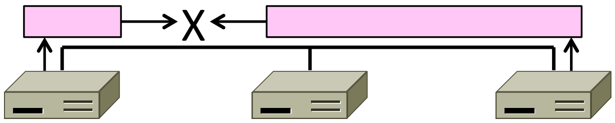
# CSMA/CD Complications

- Want everyone who collides to know that it happened
  - Time window in which a node may hear of a collision is 2D seconds



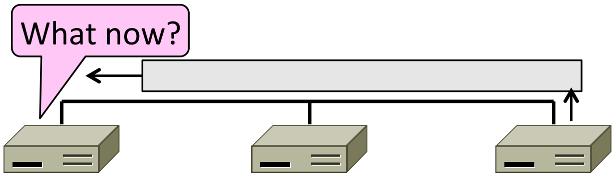
# CSMA/CD Complications (2)

- Impose a minimum frame size that lasts for 2D seconds
  - So node can’t finish before collision
  - Ethernet minimum frame is 64 bytes



# CSMA "Persistence"

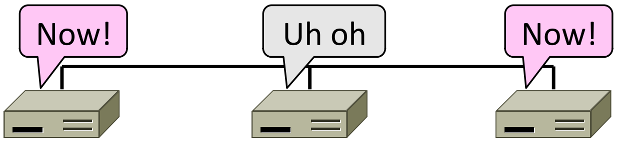
- What should a node do if another node is sending?



- Idea: Wait until it is done, and send

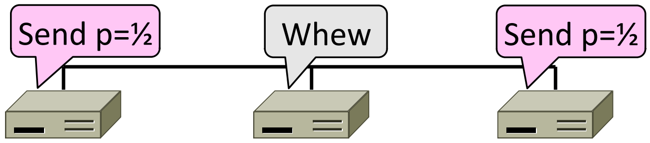
# CSMA "Persistence" (2)

- Problem is that multiple waiting nodes will queue up then collide
  - More load, more of a problem



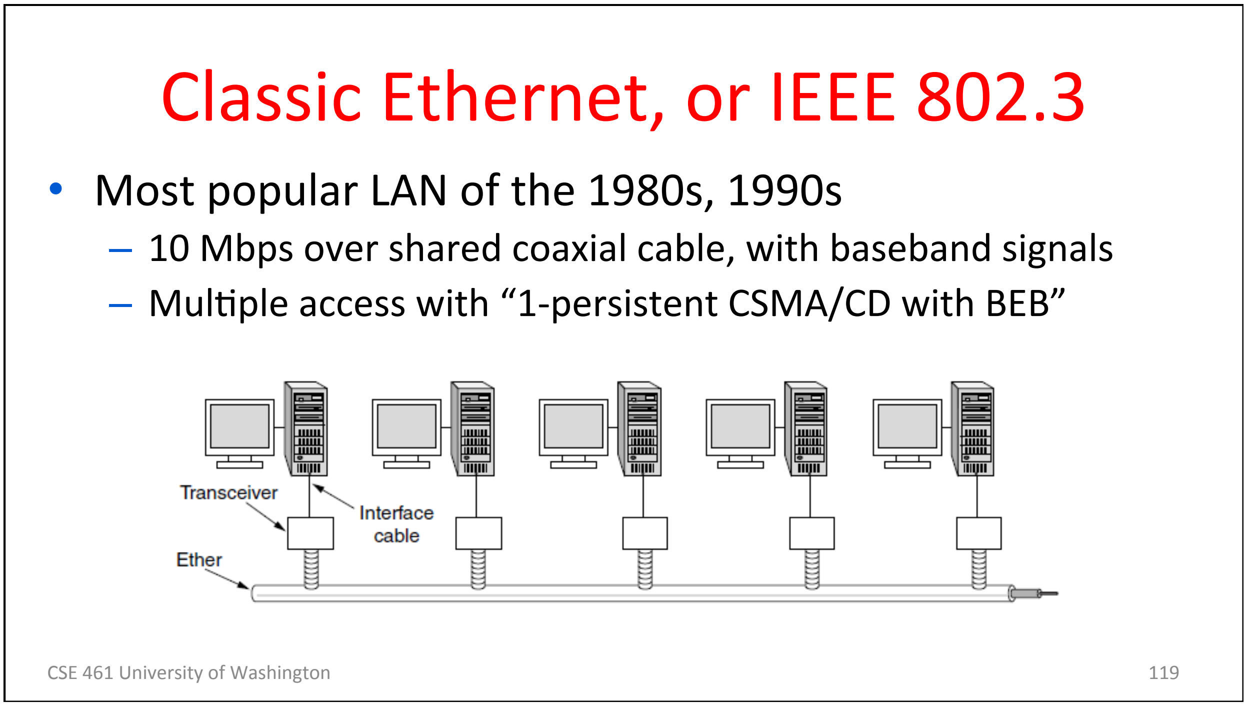
# CSMA "Persistence" (3)

- Intuition for a better solution
  - If there are N queued senders, we want each to send next with probability 1/N



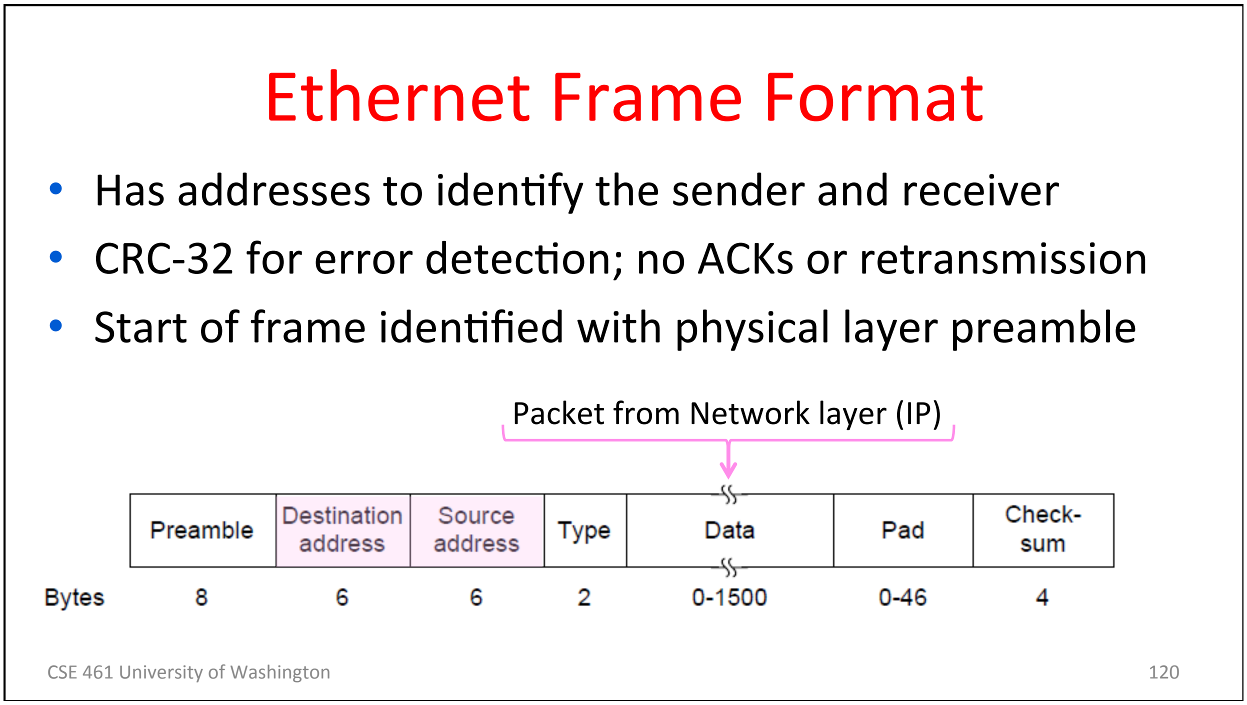
# Binary Exponential Backoff (BEB)

- Cleverly estimates the probability
  - 1st collision, wait 0 or 1 frame times
  - 2nd collision, wait from 0 to 3 times
  - 3rd collision, wait from 0 to 7 times …
- BEB doubles interval for each successive collision
  - Quickly gets large enough to work
  - Very efficient in practice



# Classic Ethernet, or IEEE 802.3

- Most popular LAN of the 1980s, 1990s
  - 10 Mbps over shared coaxial cable, with baseband signals
  - Multiple access with “1-persistent CSMA/CD with BEB”

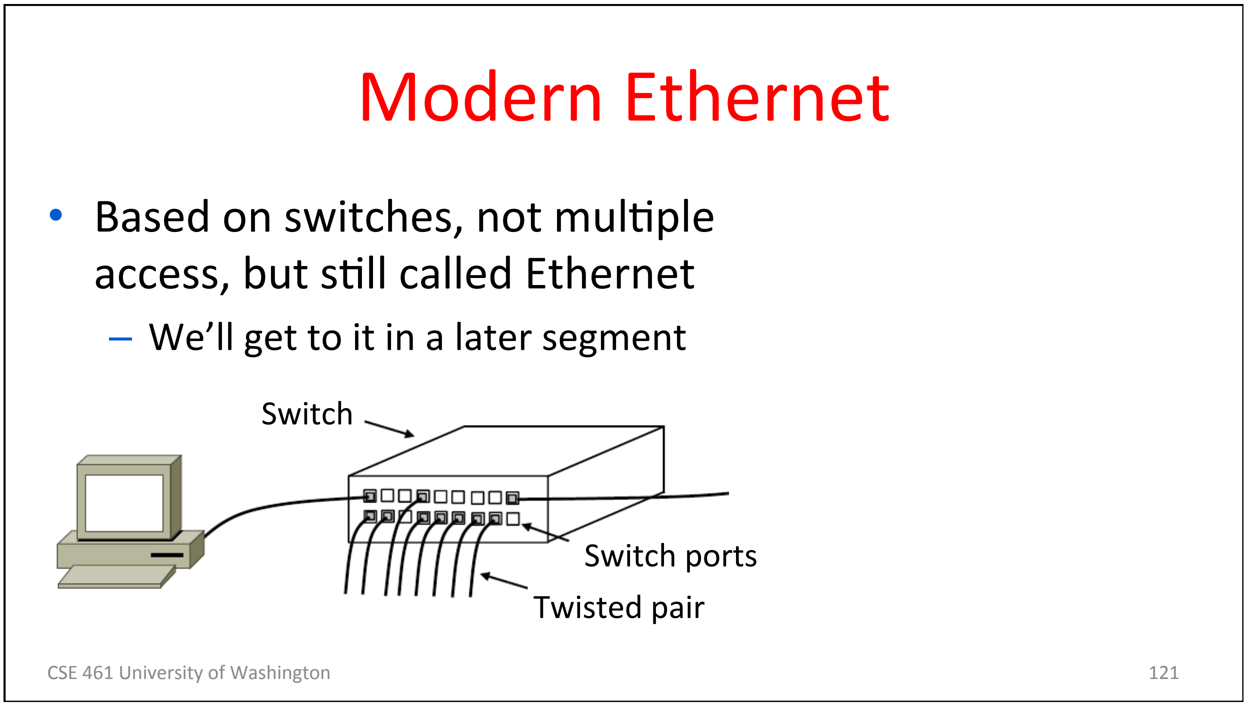


# Ethernet Frame Format

- Has addresses to identify the sender and receiver
- CRC-32 for error detection; no ACKs or retransmission
- Start of frame identified with physical layer preamble

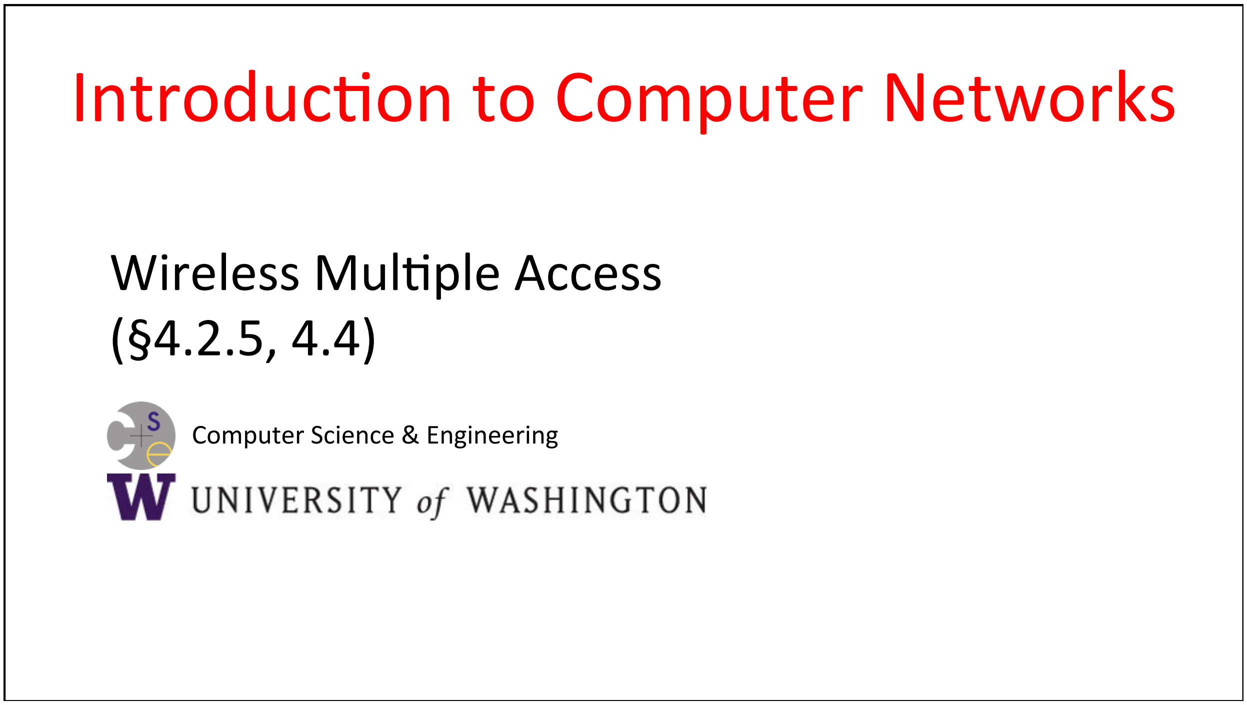
Packet from Network layer (IP)

| | Preamble | Destination address | Source address | Type | Data | Pad | Check-sum |
|---|---|---|---|---|---|---|---|
| Bytes | 8 | 6 | 6 | 2 | 0-1500 | 0-46 | 4 |



# Modern Ethernet

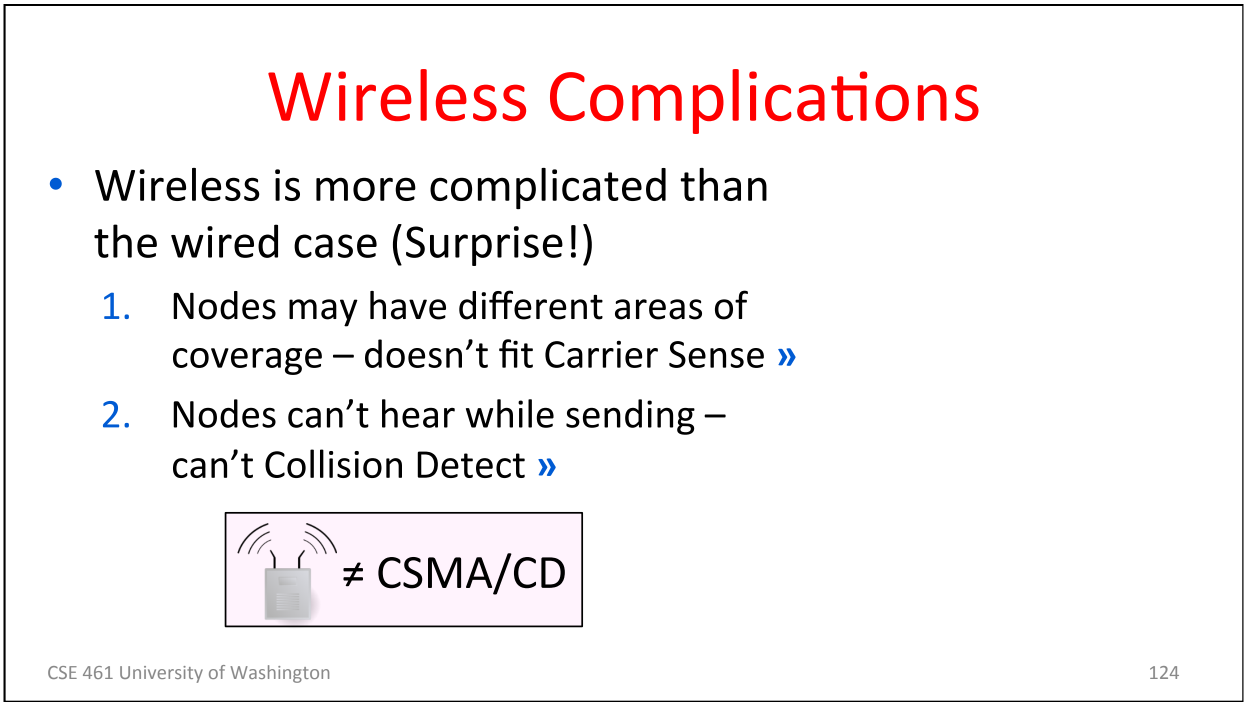
- Based on switches, not multiple access, but still called Ethernet
  - We'll get to it in a later segment



# Introduction to Computer Networks

Wireless Multiple Access
(§4.2.5, 4.4)

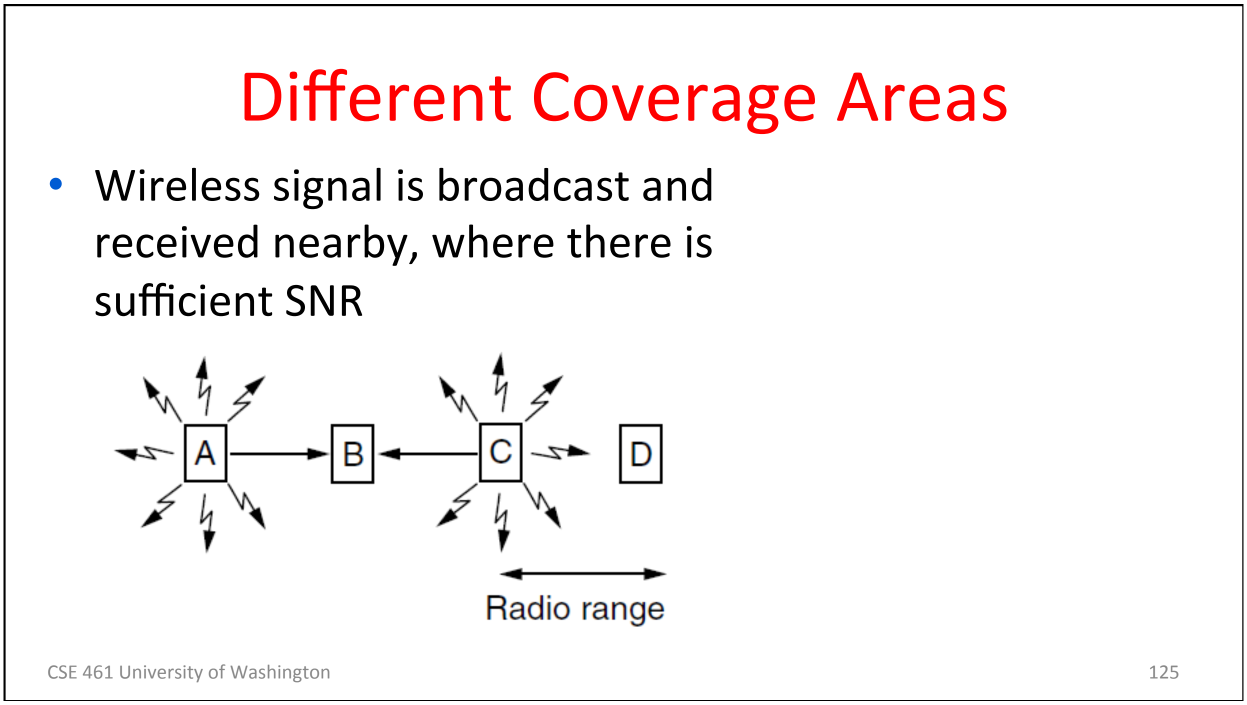
Computer Science & Engineering
UNIVERSITY *of* WASHINGTON



# Wireless Complications

- Wireless is more complicated than the wired case (Surprise!)
  1. Nodes may have different areas of coverage – doesn’t fit Carrier Sense »
  2. Nodes can’t hear while sending – can’t Collision Detect »

≠ CSMA/CD

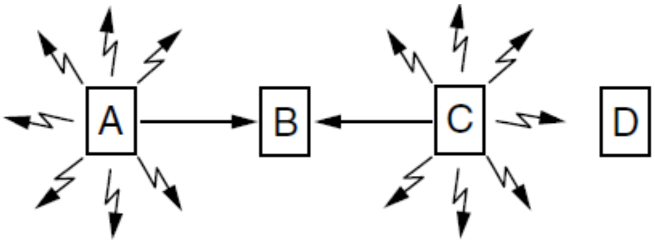


# Different Coverage Areas

- Wireless signal is broadcast and received nearby, where there is sufficient SNR

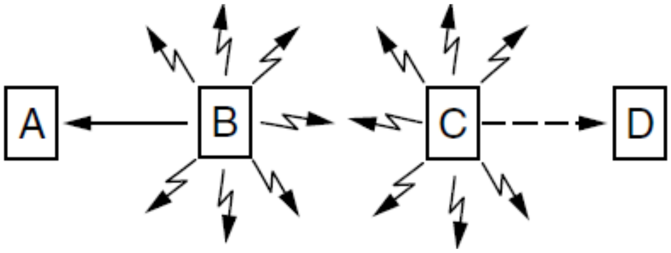
# Hidden Terminals

- Nodes A and C are hidden terminals when sending to B
  - Can’t hear each other (to coordinate) yet collide at B
  - We want to avoid the inefficiency of collisions



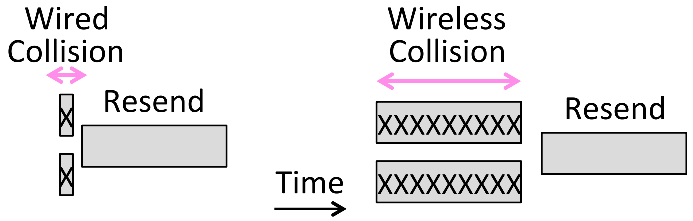
# Exposed Terminals

- B and C are exposed terminals when sending to A and D
  - Can hear each other yet don’t collide at receivers A and D
  - We want to send concurrently to increase performance



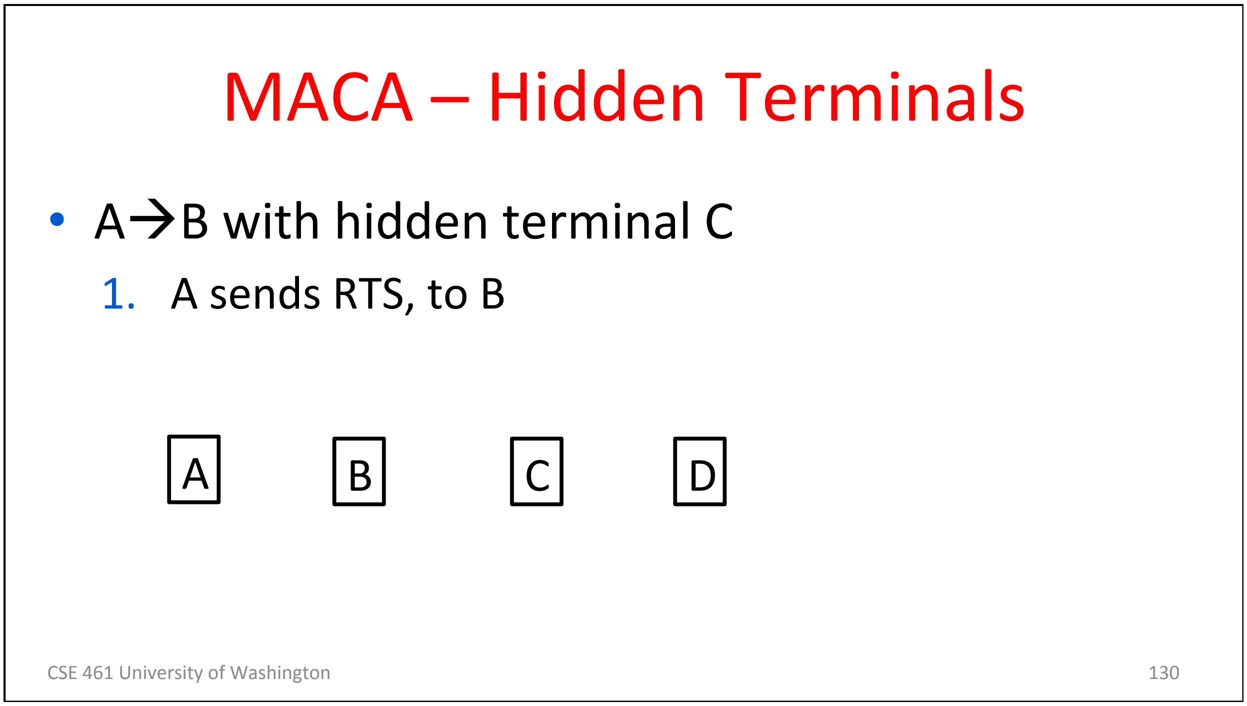
# Nodes Can't Hear While Sending

- With wires, detecting collisions (and aborting) lowers their cost
- More wasted time with wireless



# Possible Solution: MACA

- MACA uses a short handshake instead of CSMA (Karn, 1990)
  - 802.11 uses a refinement of MACA (later)
- Protocol rules:
  1. A sender node transmits a RTS (Request-To-Send, with frame length)
  2. The receiver replies with a CTS (Clear-To-Send, with frame length)
  3. Sender transmits the frame while nodes hearing the CTS stay silent
  - Collisions on the RTS/CTS are still possible, but less likely

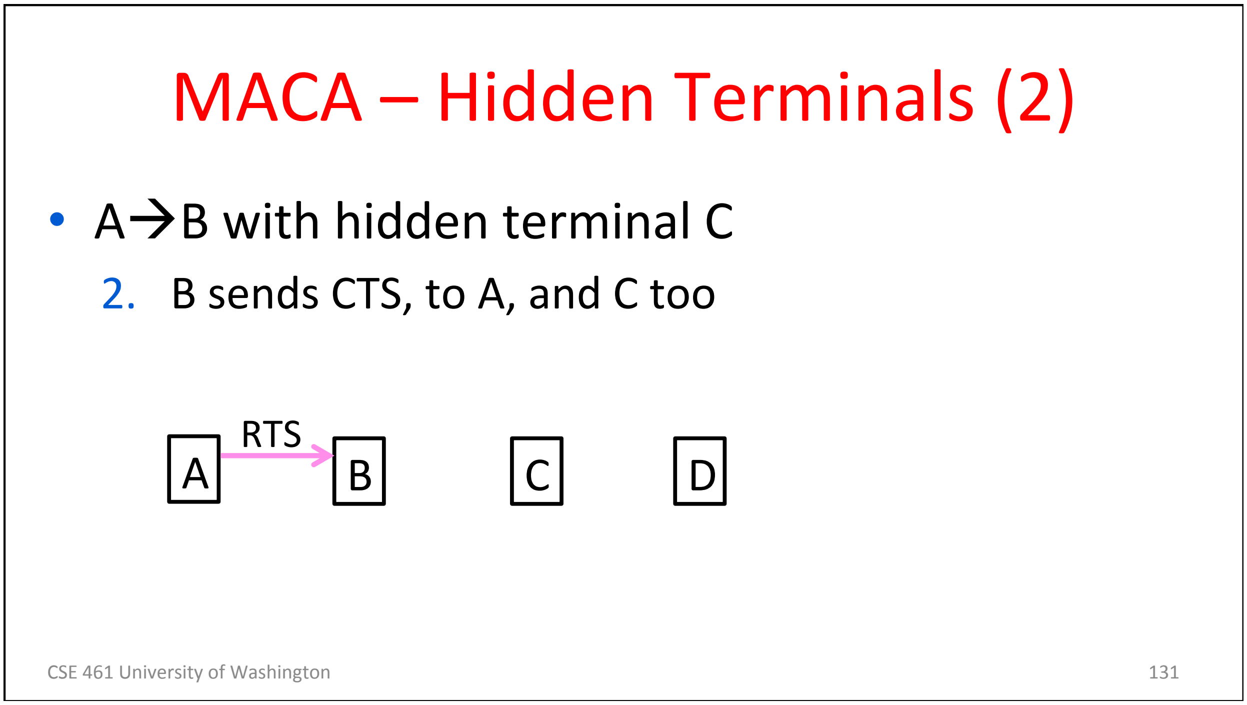


# MACA – Hidden Terminals

- A→B with hidden terminal C
  1. A sends RTS, to B

A B C D

CSE 461 University of Washington 130



# MACA – Hidden Terminals (2)

- A→B with hidden terminal C
  2. B sends CTS, to A, and C too

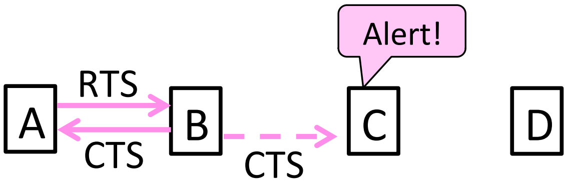
RTS

A B C D

CSE 461 University of Washington 131

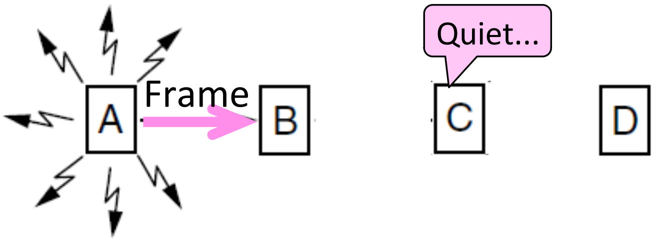
# MACA – Hidden Terminals (3)

- A→B with hidden terminal C
  2. B sends CTS, to A, and C too



# MACA – Hidden Terminals (4)

- A→B with hidden terminal C
  3. A sends frame while C defers

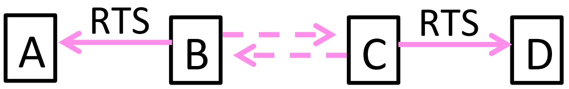


# MACA – Exposed Terminals

- B→A, C→D as exposed terminals
  - B and C send RTS to A and D

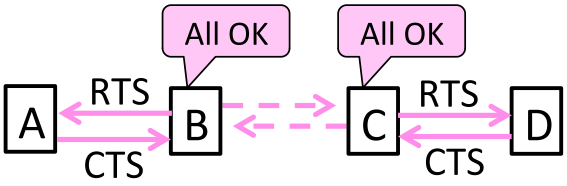
# MACA – Exposed Terminals (2)

- B→A, C→D as exposed terminals
  - A and D send CTS to B and C



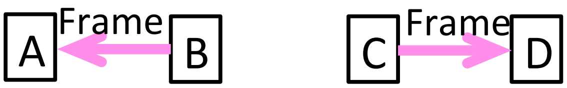
# MACA – Exposed Terminals (3)

- B→A, C→D as exposed terminals
  - A and D send CTS to B and C



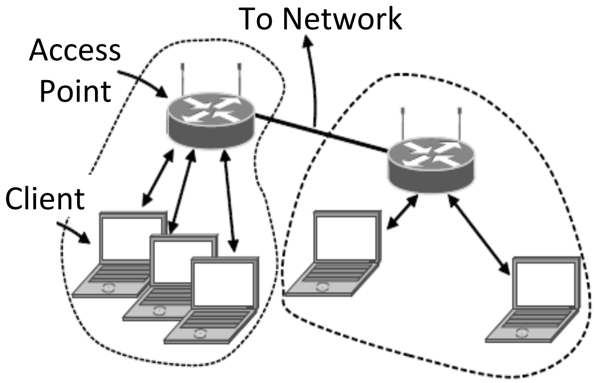
# MACA – Exposed Terminals (4)

- B→A, C→D as exposed terminals
  - A and D send CTS to B and C



# 802.11, or WiFi

- Very popular wireless LAN started in the 1990s
- Clients get connectivity from a (wired) AP (Access Point)
- It's a multi-access problem ☺
- Various flavors have been developed over time
  - Faster, more features



# 802.11 Physical Layer

- Uses 20/40 MHz channels on ISM bands
  - 802.11b/g/n on 2.4 GHz
  - 802.11 a/n on 5 GHz
- OFDM modulation (except legacy 802.11b)
  - Different amplitudes/phases for varying SNRs
  - Rates from 6 to 54 Mbps plus error correction
  - 802.11n uses multiple antennas; see "802.11 with Multiple Antennas for Dummies"

# 802.11 Link Layer

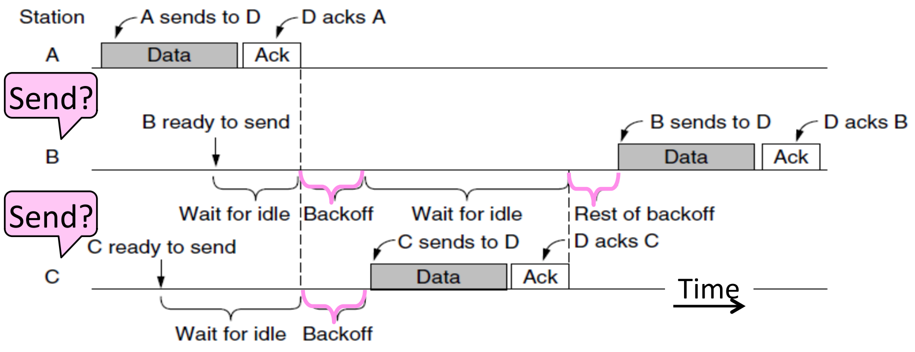
- Multiple access uses CSMA/CA (next); RTS/CTS optional
- Frames are ACKed and retransmitted with ARQ
- Funky addressing (three addresses!) due to AP
- Errors are detected with a 32-bit CRC
- Many, many features (e.g., encryption, power save)

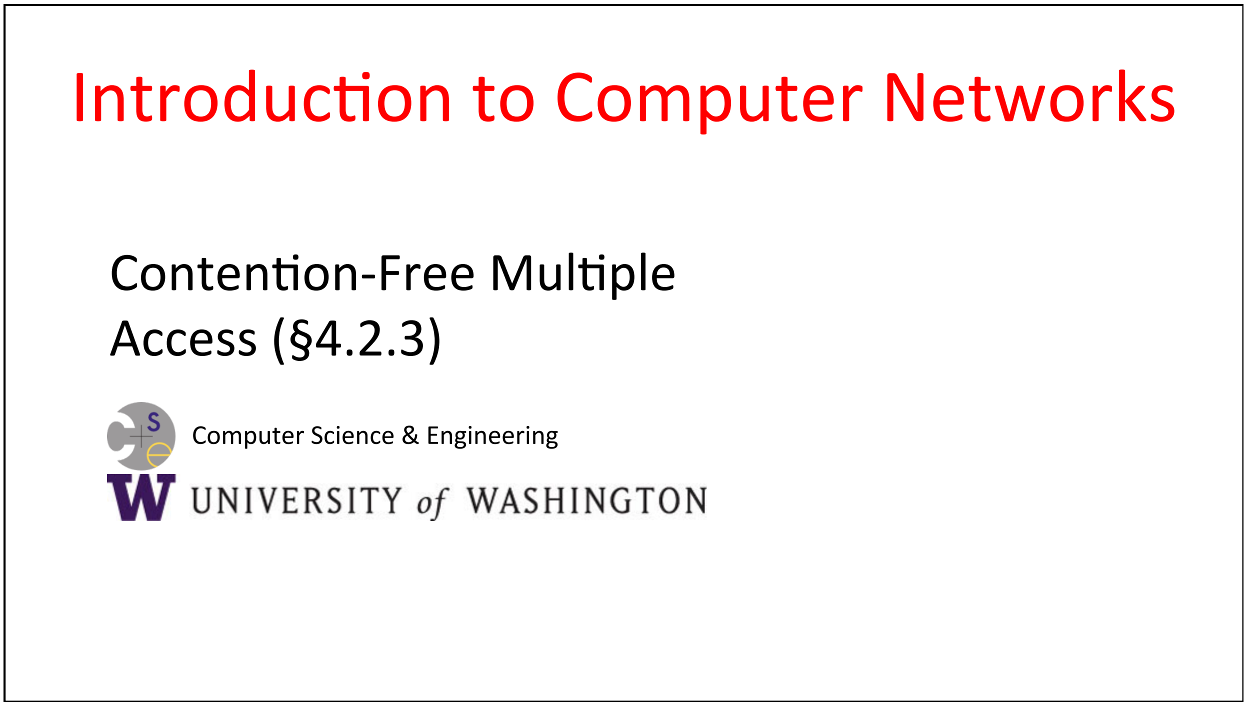
Packet from Network layer (IP)

| | Frame control | Duration | Address 1 (recipient) | Address 2 (transmitter) | Address 3 | Sequence | Data | Check sequence |
|---|---|---|---|---|---|---|---|---|
| Bytes | 2 | 2 | 6 | 6 | 6 | 2 | 0–2312 | 4 |

# 802.11 CSMA/CA for Multiple Access

- Sender avoids collisions by inserting small random gaps
  - E.g., when both B and C send, C picks a smaller gap, goes first



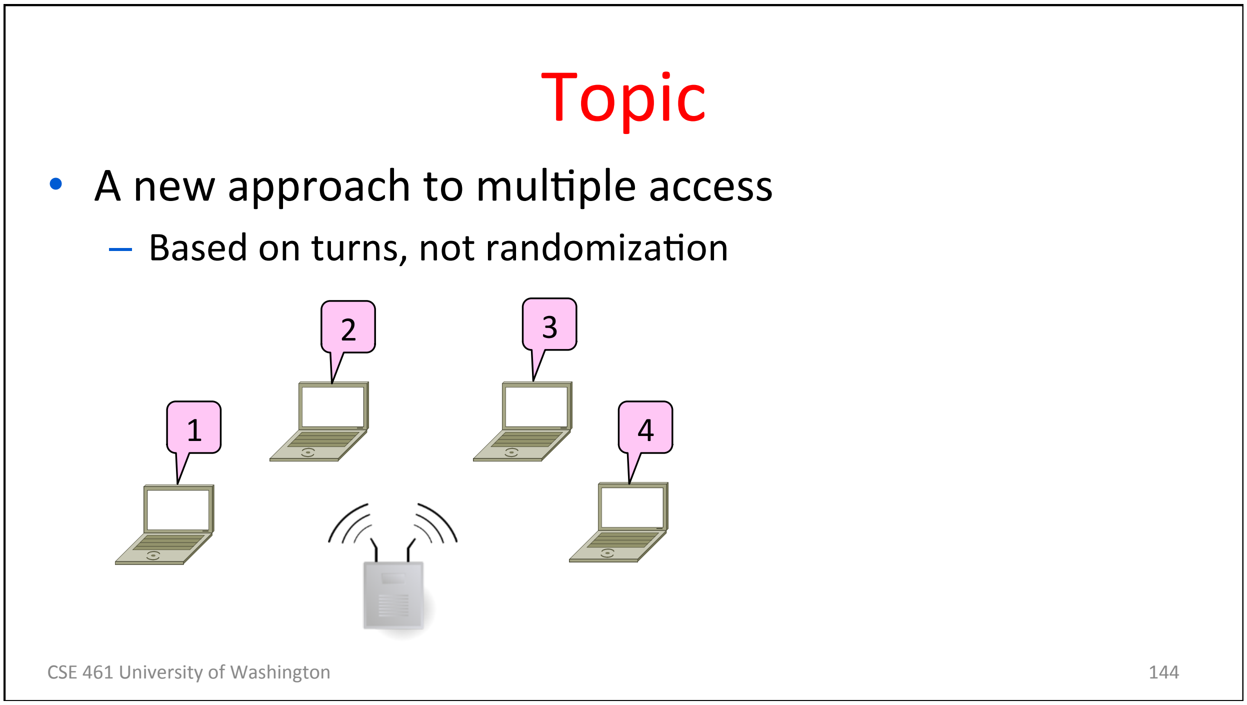


# Introduction to Computer Networks

Contention-Free Multiple Access (§4.2.3)

Computer Science & Engineering

UNIVERSITY *of* WASHINGTON



## Topic

- A new approach to multiple access
  - Based on turns, not randomization

CSE 461 University of Washington 144

# Issues with Random Multiple Access

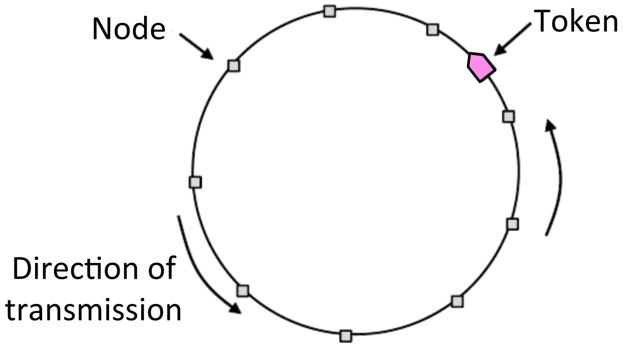
- CSMA is good under low load:
  - Grants immediate access
  - Little overhead (collisions)
- But not so good under high load:
  - High overhead (expect collisions)
  - Access time varies (lucky/unlucky)
- We want to do better under load!

# Turn-Taking Multiple Access Protocols

- They define an order in which nodes get a chance to send
  - Or pass, if no traffic at present
- We just need some ordering …
  - E.g., Token Ring »
  - E.g., node addresses

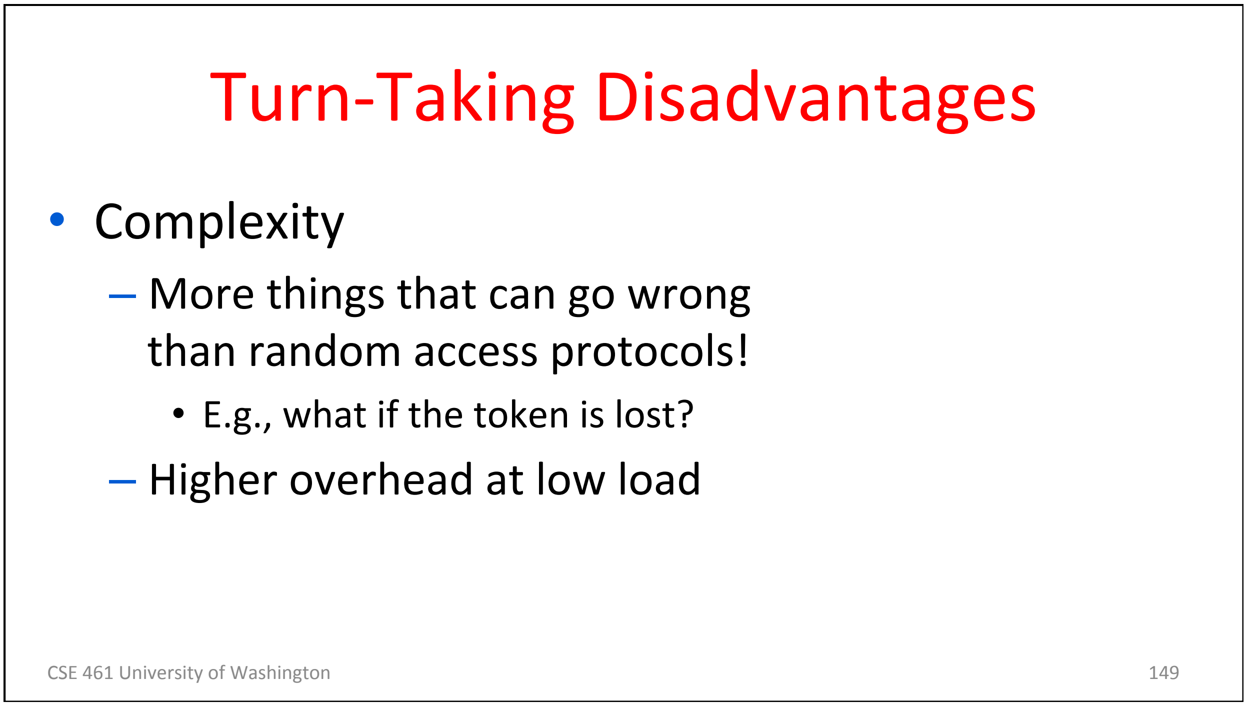
# Token Ring

- Arrange nodes in a ring; token rotates “permission to send” to each node in turn



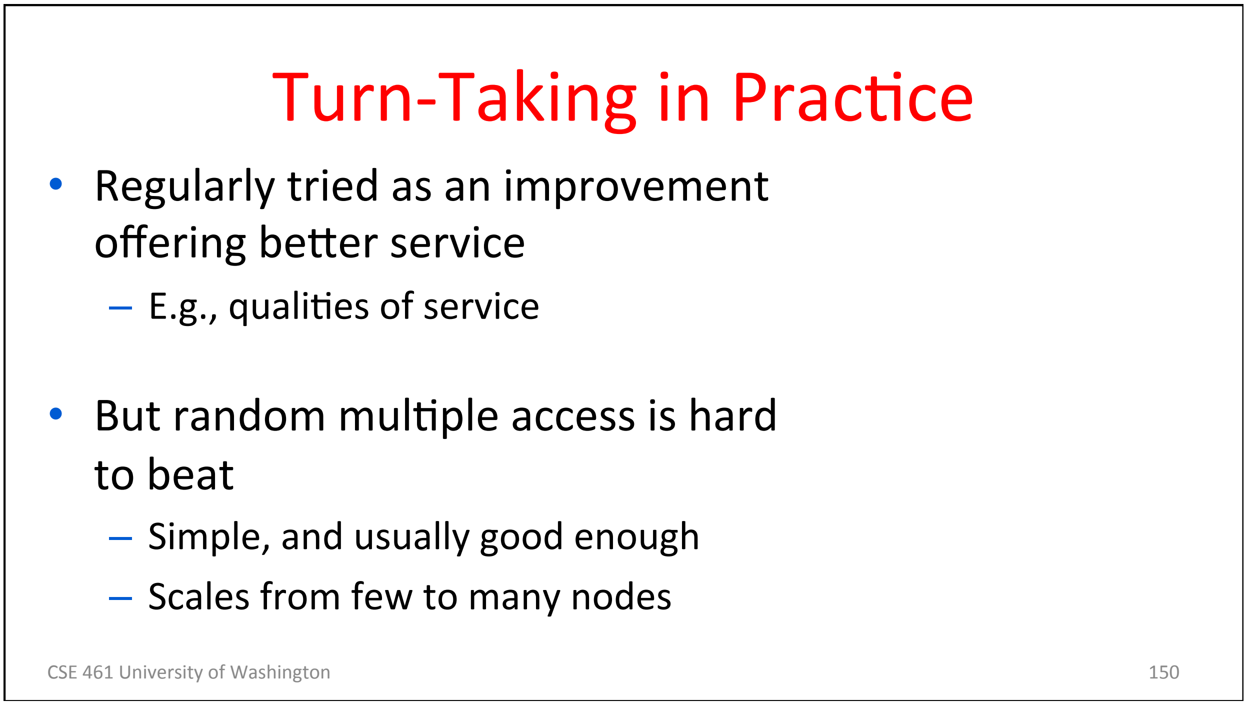
# Turn-Taking Advantages

- Fixed overhead with no collisions
  - More efficient under load
- Regular chance to send with no unlucky nodes
  - Predictable service, easily extended to guaranteed quality of service



# Turn-Taking Disadvantages

- Complexity
  - More things that can go wrong than random access protocols!
    - E.g., what if the token is lost?
  - Higher overhead at low load



# Turn-Taking in Practice

- Regularly tried as an improvement offering better service
  - E.g., qualities of service
- But random multiple access is hard to beat
  - Simple, and usually good enough
  - Scales from few to many nodes

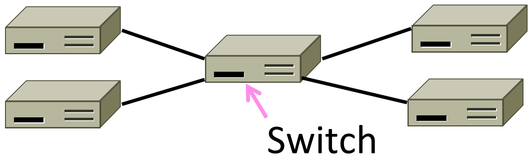
# Introduction to Computer Networks

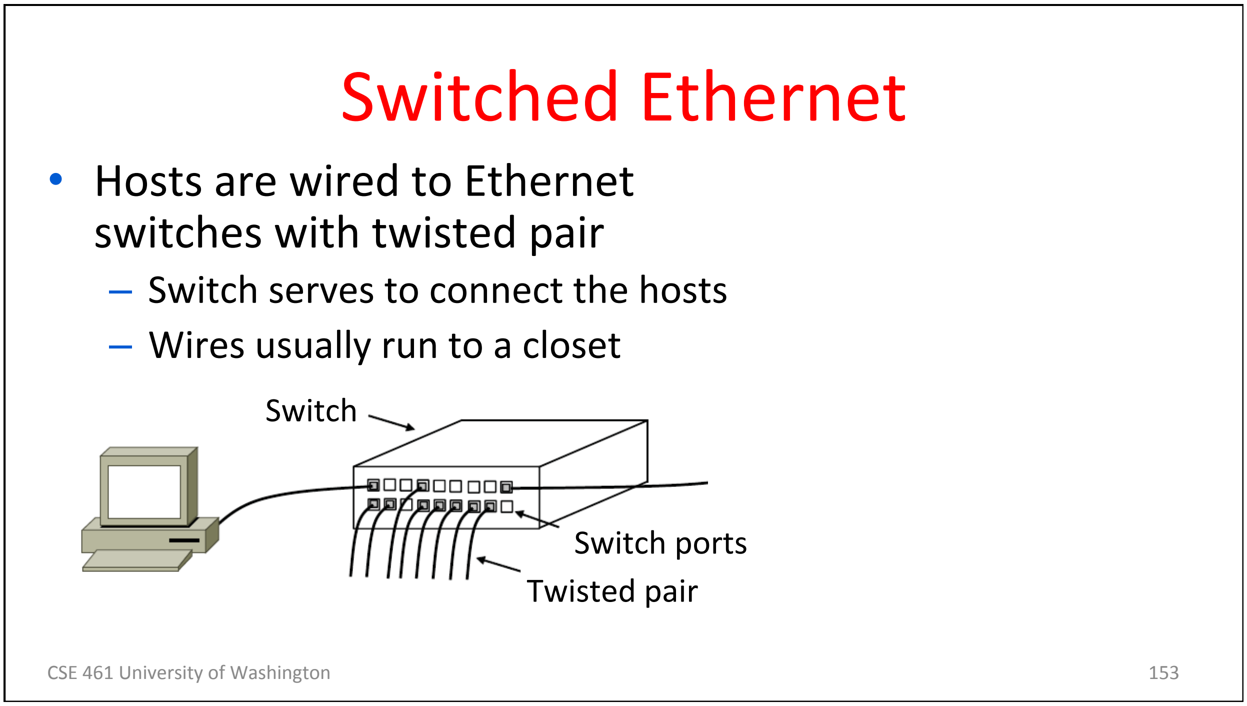
LAN Switches (§4.x)

Computer Science & Engineering
UNIVERSITY *of* WASHINGTON

# Topic

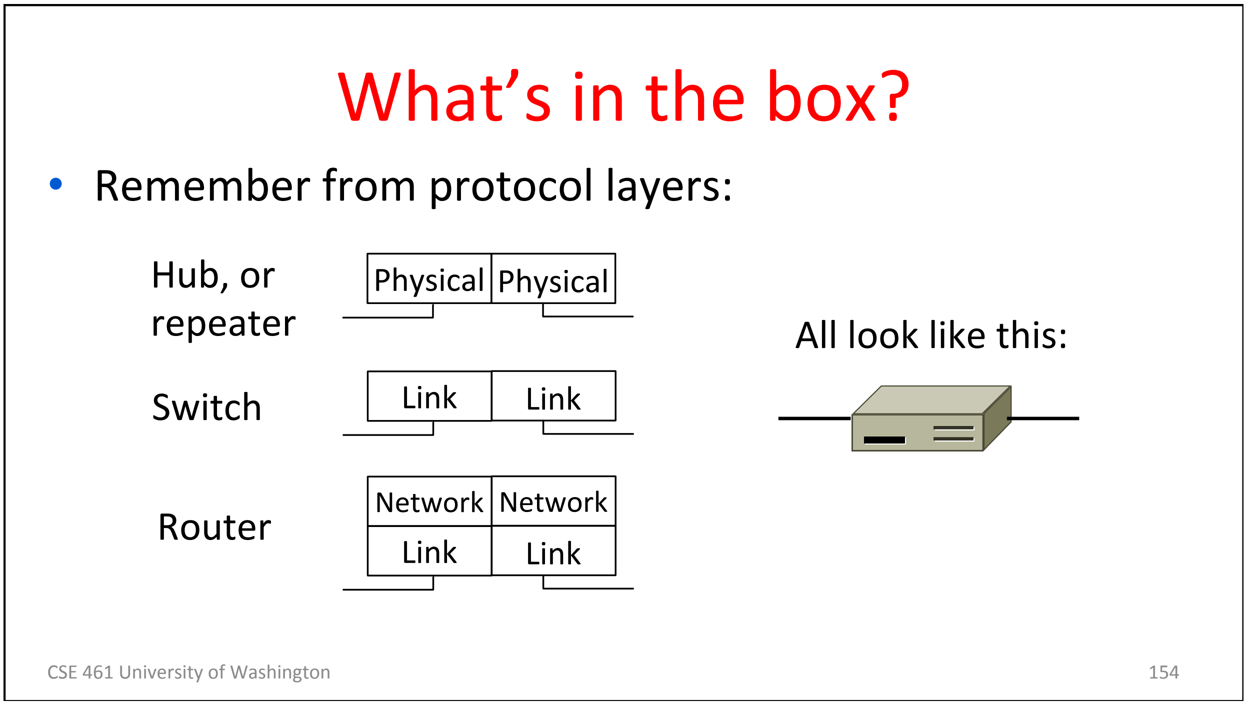
- How do we connect nodes with a switch instead of multiple access
  - Uses multiple links/wires
  - Basis of modern (switched) Ethernet





# Switched Ethernet

- Hosts are wired to Ethernet switches with twisted pair
  - Switch serves to connect the hosts
  - Wires usually run to a closet



# What's in the box?

- Remember from protocol layers:

Hub, or repeater: Physical | Physical

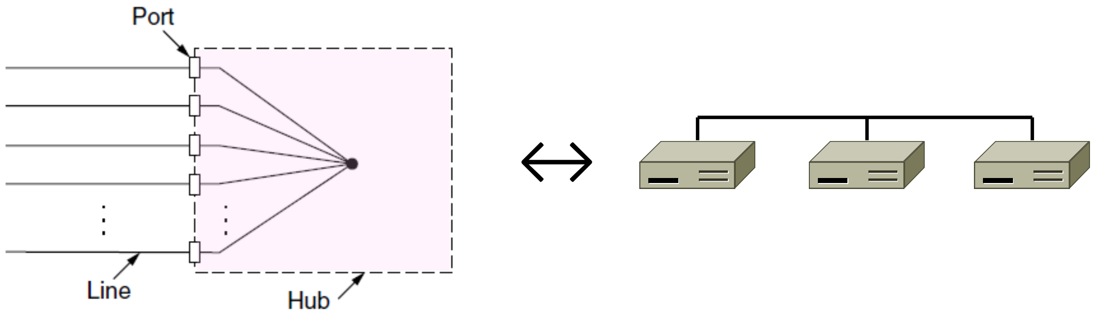
Switch: Link | Link

Router: Network | Network; Link | Link

All look like this:

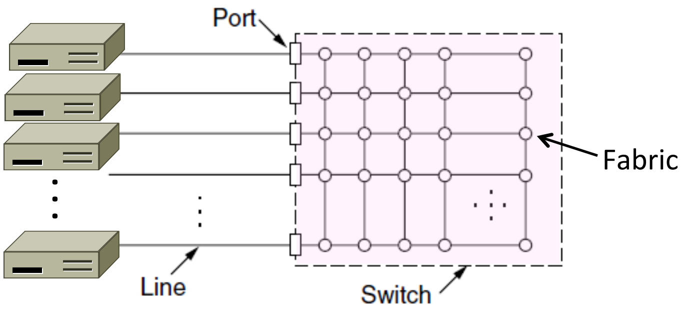
# Inside a Hub

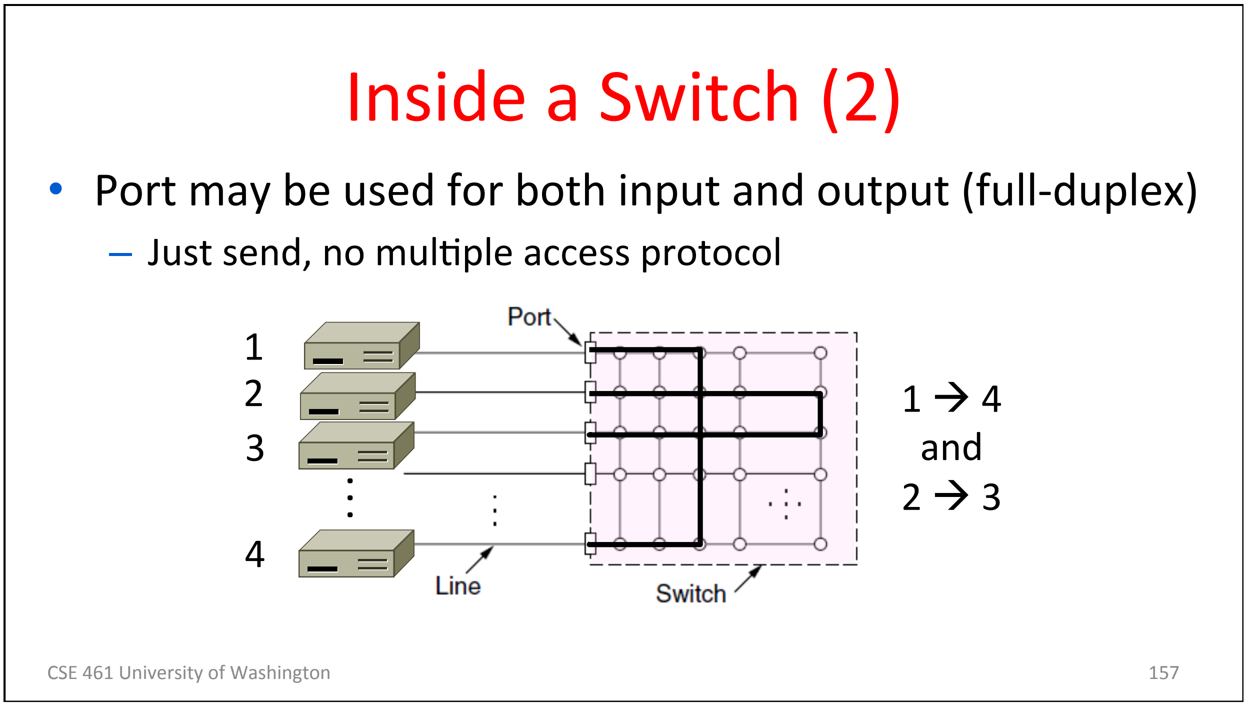
- All ports are wired together; more convenient and reliable than a single shared wire



# Inside a Switch

- Uses frame addresses to connect input port to the right output port; multiple frames may be switched in parallel

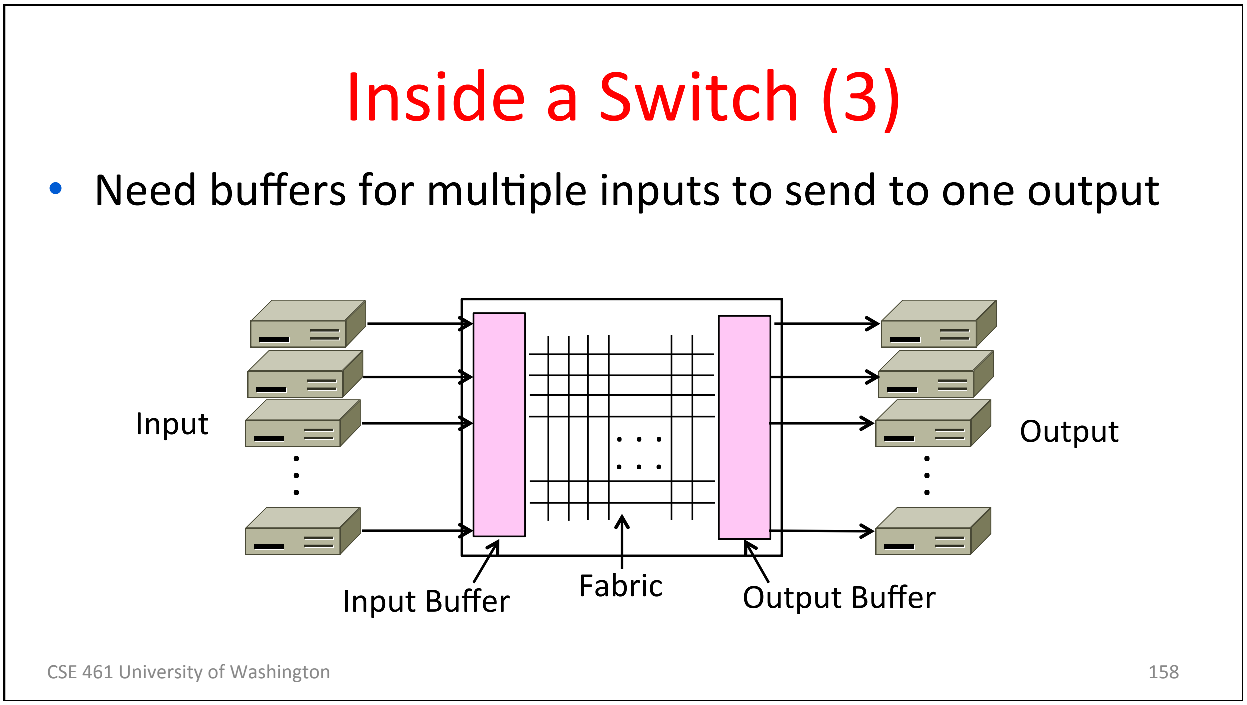




# Inside a Switch (2)

- Port may be used for both input and output (full-duplex)
  - Just send, no multiple access protocol

1 → 4
and
2 → 3

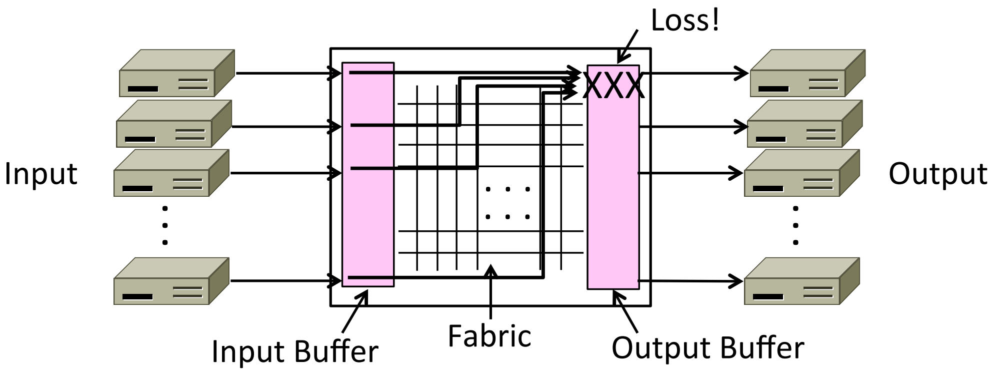


# Inside a Switch (3)

- Need buffers for multiple inputs to send to one output

# Inside a Switch (4)

- Sustained overload will fill buffer and lead to frame loss

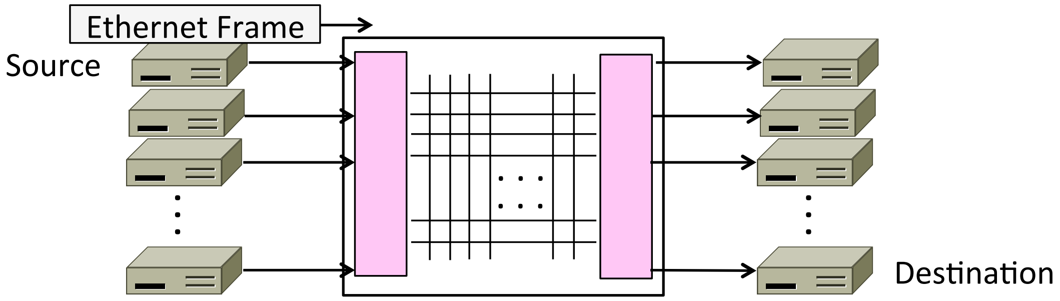


# Advantages of Switches

- Switches and hubs have replaced the shared cable of classic Ethernet
  – Convenient to run wires to one location
  – More reliable; wire cut is not a single point of failure that is hard to find
- Switches offer scalable performance
  – E.g., 100 Mbps per port instead of 100 Mbps for all nodes of shared cable / hub

# Switch Forwarding

- Switch needs to find the right output port for the destination address in the Ethernet frame. How?
  - Want to let hosts be moved around readily; don't look at IP

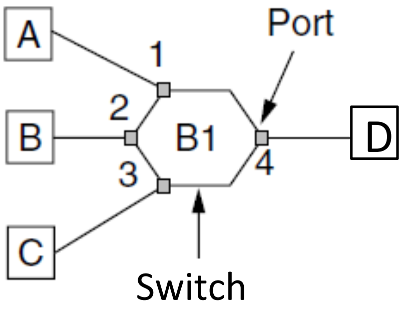


# Backward Learning

- Switch forwards frames with a port/address table as follows:
  1. To fill the table, it looks at the source address of input frames
  2. To forward, it sends to the port, or else broadcasts to all ports

# Backward Learning (2)

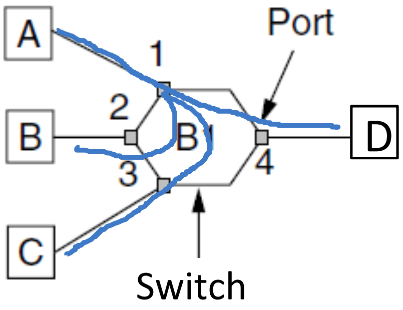
- 1: A sends to D



| Address | Port |
|---|---|
| A | |
| B | |
| C | |
| D | |

# Backward Learning (3)

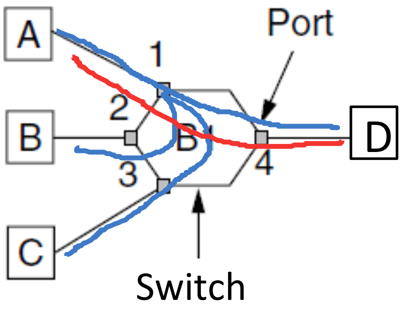
- 2: D sends to A



| Address | Port |
|---|---|
| A | 1 |
| B | |
| C | |
| D | |

# Backward Learning (4)

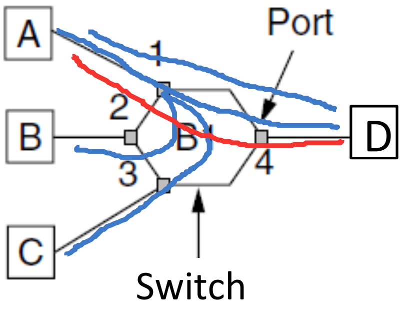
- 3: D sends to A



| Address | Port |
|---|---|
| A | 1 |
| B | |
| C | |
| D | 4 |

# Backward Learning (5)

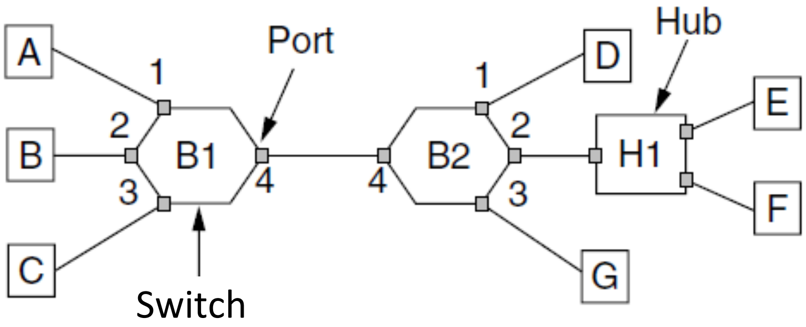
- 3: D sends to A



| Address | Port |
|---|---|
| A | 1 |
| B | |
| C | |
| D | 4 |

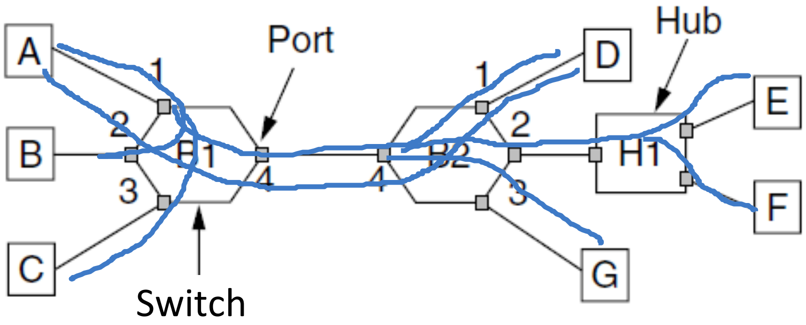
# Learning with Multiple Switches

- Just works with multiple switches and a mix of hubs, *assuming no loops* in the topology, E.g., A sends to D



# Learning with Multiple Switches (2)

- Just works with multiple switches and a mix of hubs *assuming no loops*, e.g., A sends to D then D sends to A



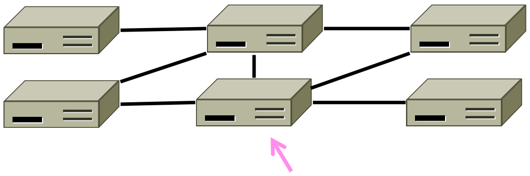
# Introduction to Computer Networks

Switch Spanning Tree (§4.x)

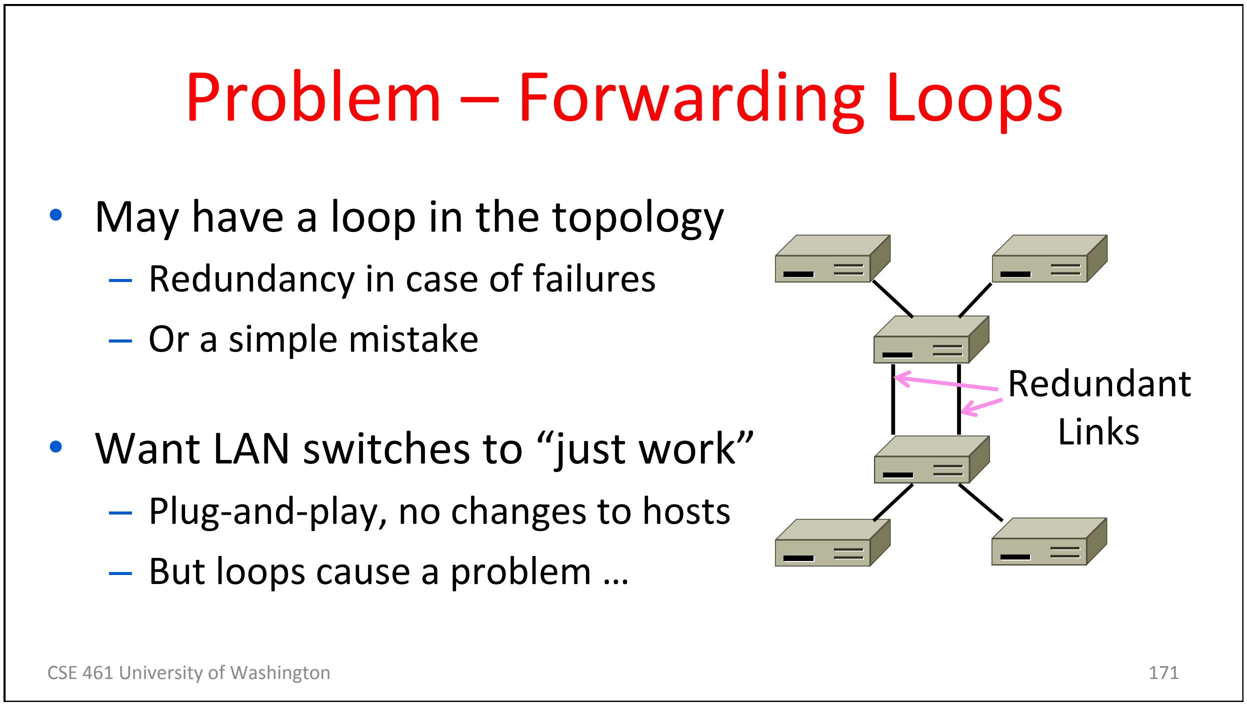
Computer Science & Engineering
UNIVERSITY *of* WASHINGTON

## Topic

- How can we connect switches in any topology so they just work
  - This is part 2 of switched Ethernet

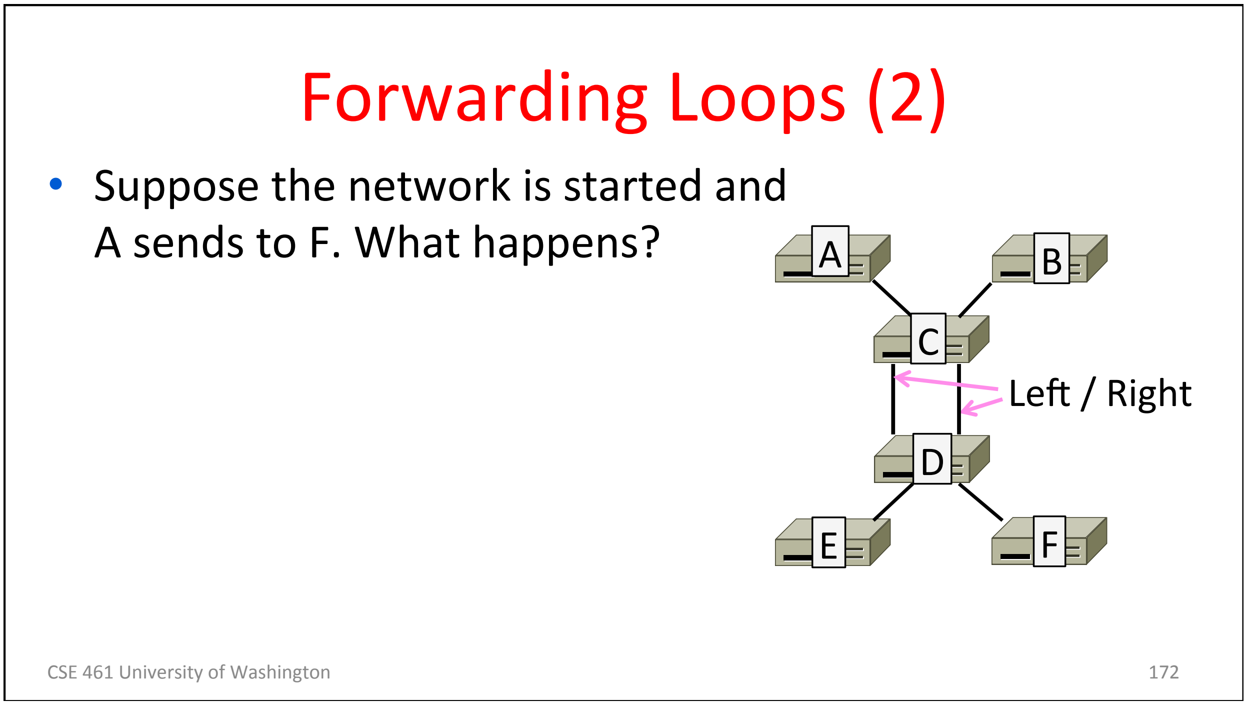


Loops – yikes!



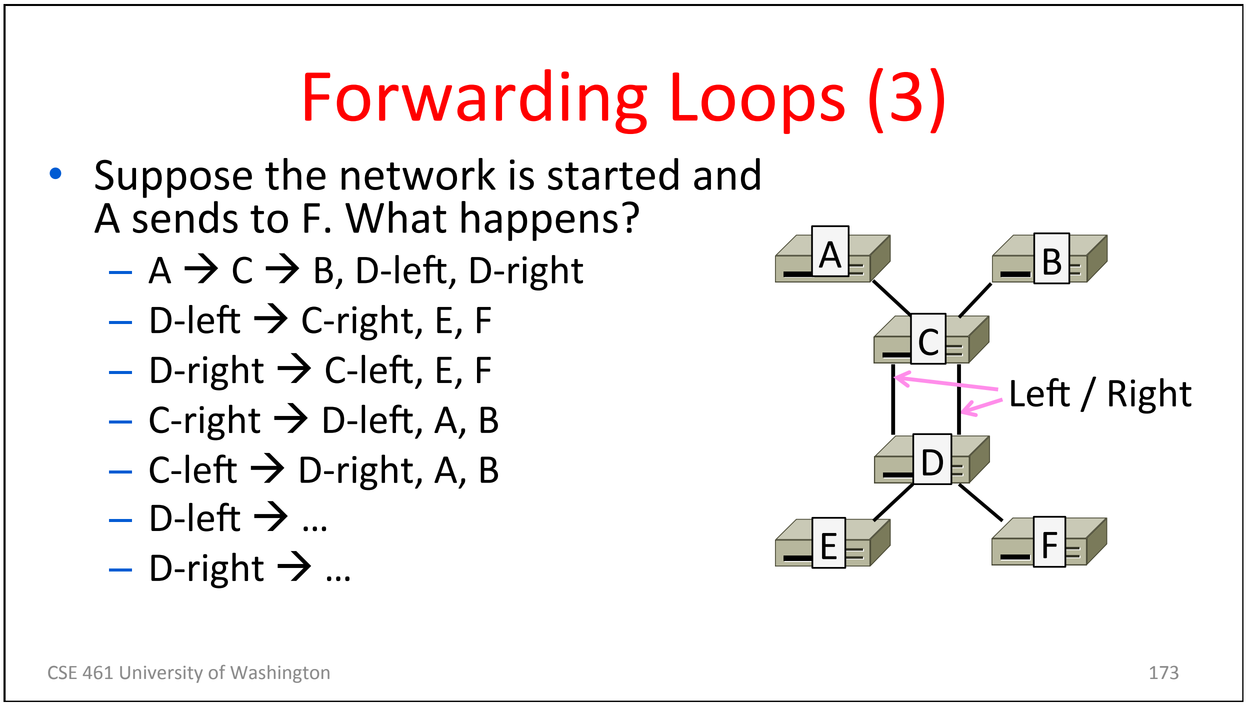
# Problem – Forwarding Loops

- May have a loop in the topology
  - Redundancy in case of failures
  - Or a simple mistake
- Want LAN switches to "just work"
  - Plug-and-play, no changes to hosts
  - But loops cause a problem …



# Forwarding Loops (2)

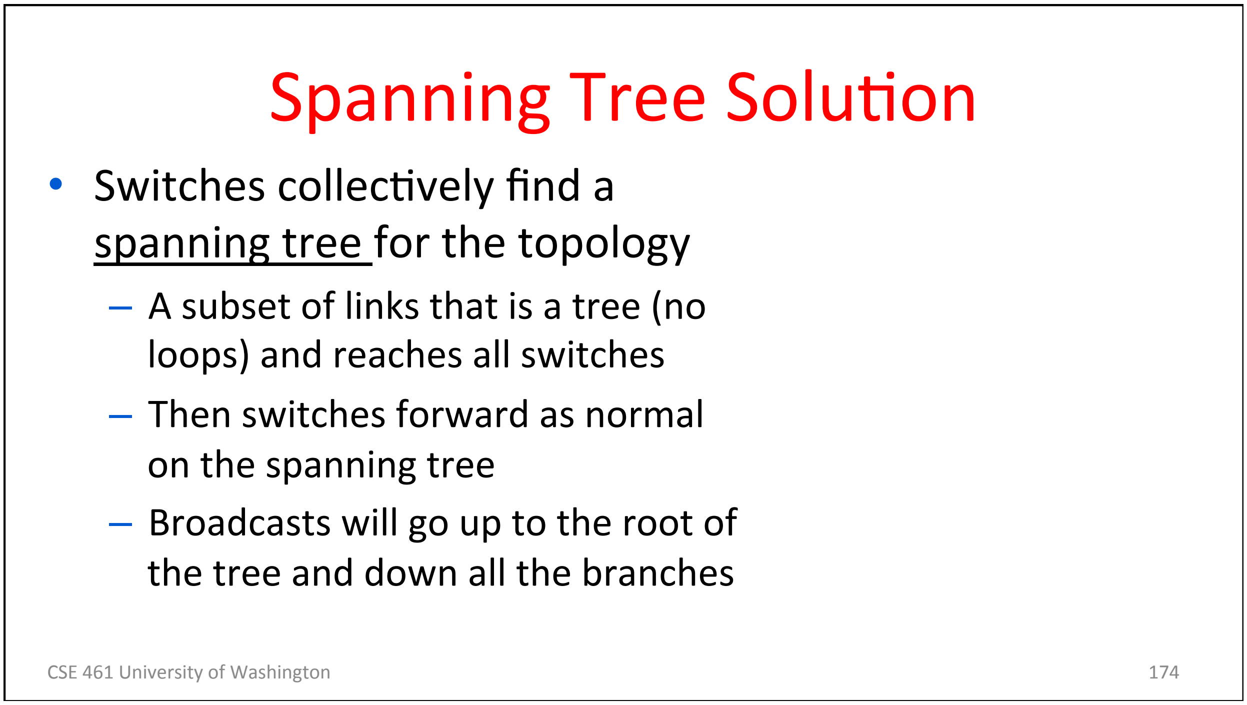
- Suppose the network is started and A sends to F. What happens?



# Forwarding Loops (3)

- Suppose the network is started and A sends to F. What happens?
  - A → C → B, D-left, D-right
  - D-left → C-right, E, F
  - D-right → C-left, E, F
  - C-right → D-left, A, B
  - C-left → D-right, A, B
  - D-left → …
  - D-right → …

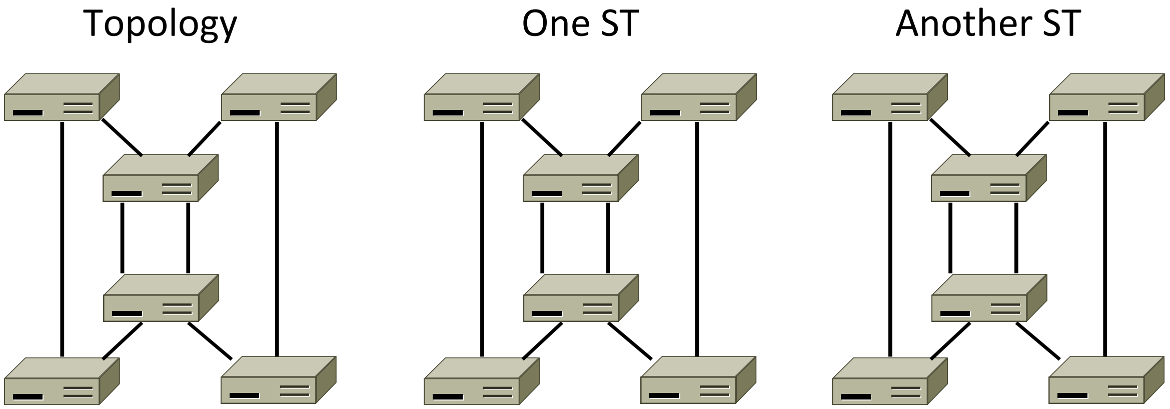
A
B
C
Left / Right
D
E
F



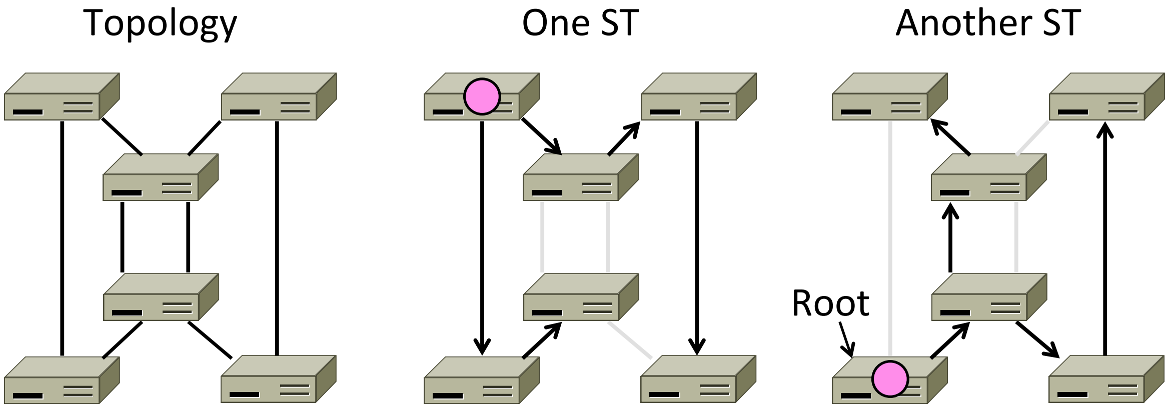
# Spanning Tree Solution

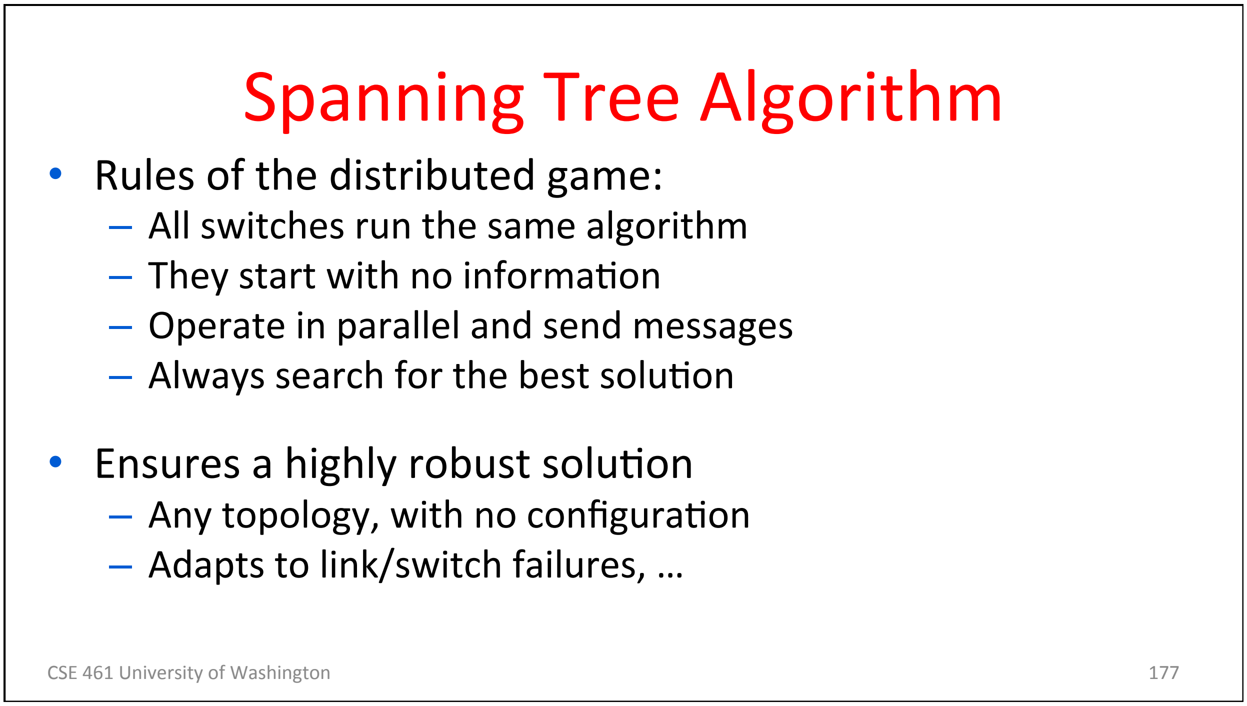
- Switches collectively find a spanning tree for the topology
  - A subset of links that is a tree (no loops) and reaches all switches
  - Then switches forward as normal on the spanning tree
  - Broadcasts will go up to the root of the tree and down all the branches

# Spanning Tree (2)



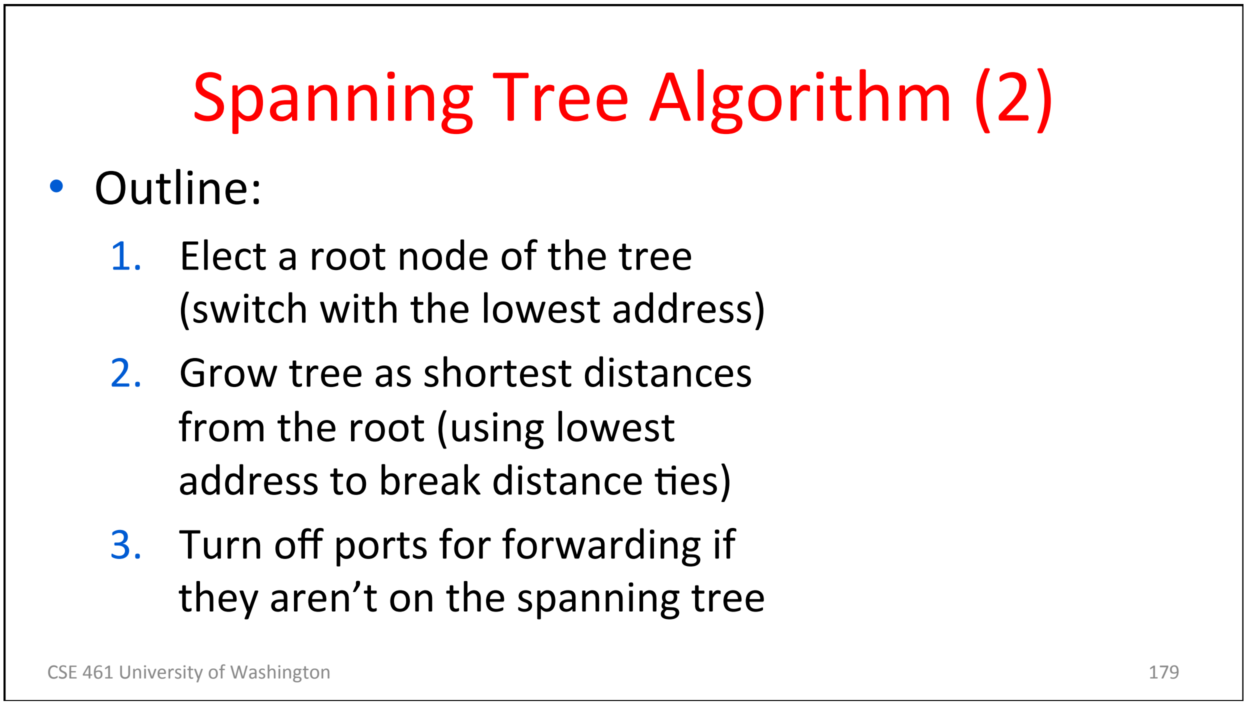
# Spanning Tree (3)





# Spanning Tree Algorithm

- Rules of the distributed game:
  - All switches run the same algorithm
  - They start with no information
  - Operate in parallel and send messages
  - Always search for the best solution
- Ensures a highly robust solution
  - Any topology, with no configuration
  - Adapts to link/switch failures, …



# Spanning Tree Algorithm (2)

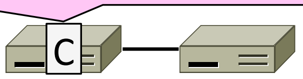
- Outline:
  1. Elect a root node of the tree (switch with the lowest address)
  2. Grow tree as shortest distances from the root (using lowest address to break distance ties)
  3. Turn off ports for forwarding if they aren't on the spanning tree

# Spanning Tree Algorithm (3)

- Details:
  - Each switch initially believes it is the root of the tree
  - Each switch sends periodic updates to neighbors with:
    - Its address, address of the root, and distance (in hops) to root
  - Switches favors ports with shorter distances to lowest root
    - Uses lowest address as a tie for distances

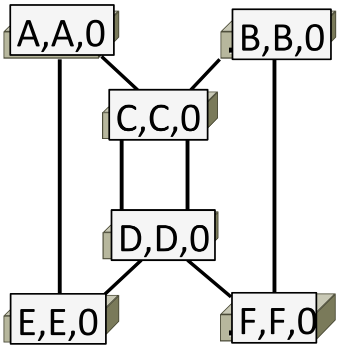
Hi, I'm C, the root is A, it's 2 hops away

or (C, A, 2)



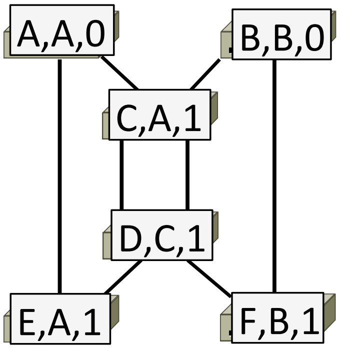
# Spanning Tree Example

- 1st round, sending:
  - A sends (A, A, 0) to say it is root
  - B, C, D, E, and F do likewise
- 1st round, receiving:
  - A still thinks is it (A, A, 0)
  - B still thinks (B, B, 0)
  - C updates to (C, A, 1)
  - D updates to (D, C, 1)
  - E updates to (E, A, 1)
  - F updates to (F, B, 1)



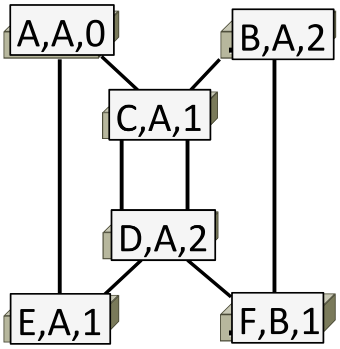
# Spanning Tree Example (2)

- 2nd round, sending
  - Nodes send their updated state
- 2nd round receiving:
  - A remains (A, A, 0)
  - B updates to (B, A, 2) via C
  - C remains (C, A, 1)
  - D updates to (D, A, 2) via C
  - E remains (E, A, 1)
  - F remains (F, B, 1)



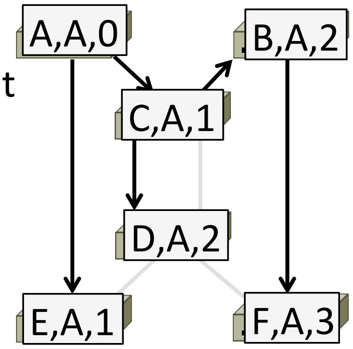
# Spanning Tree Example (3)

- 3rd round, sending
  - Nodes send their updated state
- 3rd round receiving:
  - A remains (A, A, 0)
  - B remains (B, A, 2) via C
  - C remains (C, A, 1)
  - D remains (D, A, 2) via C-left
  - E remains (E, A, 1)
  - F updates to (F, A, 3) via B



# Spanning Tree Example (4)

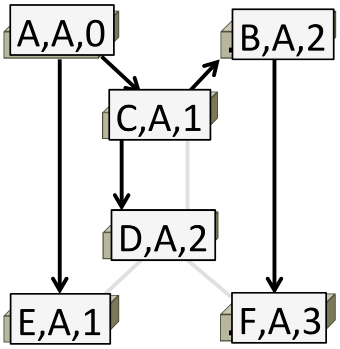
- 4th round
  - Steady-state has been reached
  - Nodes turn off forwarding that  is not on the spanning tree
- Algorithm continues to run
  - Adapts by timing out information
  - E.g., if A fails, other nodes forget it, and B will become the new root

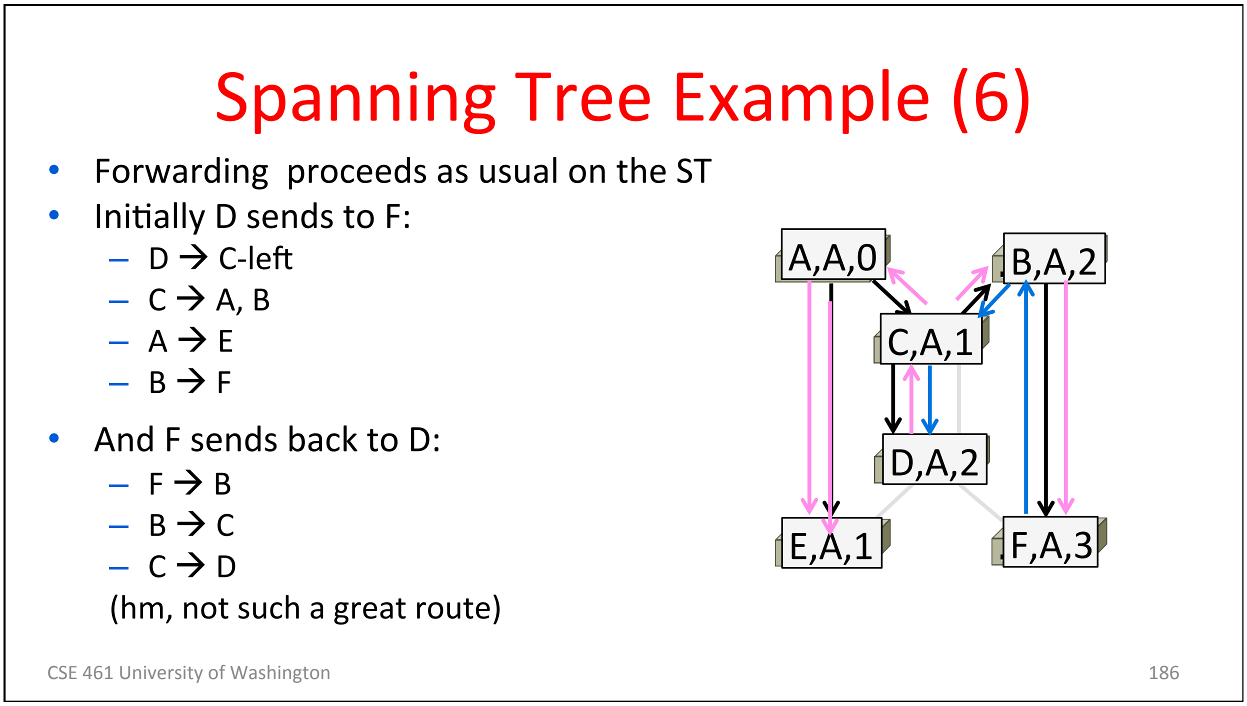


# Spanning Tree Example (5)

- Forwarding  proceeds as usual on the ST
- Initially D sends to F:

- And F sends back to D:





# Spanning Tree Example (6)

- Forwarding proceeds as usual on the ST
- Initially D sends to F:
  - D → C-left
  - C → A, B
  - A → E
  - B → F
- And F sends back to D:
  - F → B
  - B → C
  - C → D

  (hm, not such a great route)