Error Correction

CSE 461

Ratul Mahajan

Why Error Correction is Harder

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

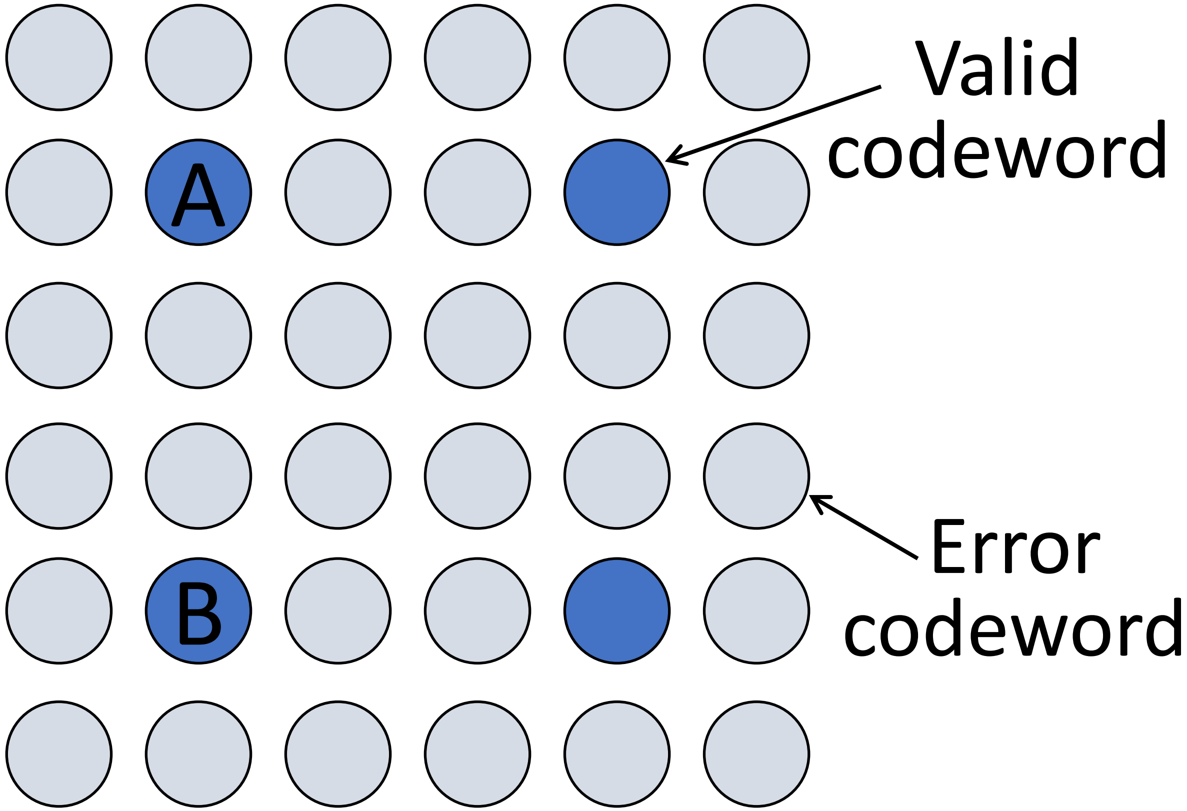
Assume a code with a Hamming distance of at least 3

- Need ≥3 bit errors to change a valid codeword into another
- Single bit errors will be closest to a unique valid codeword

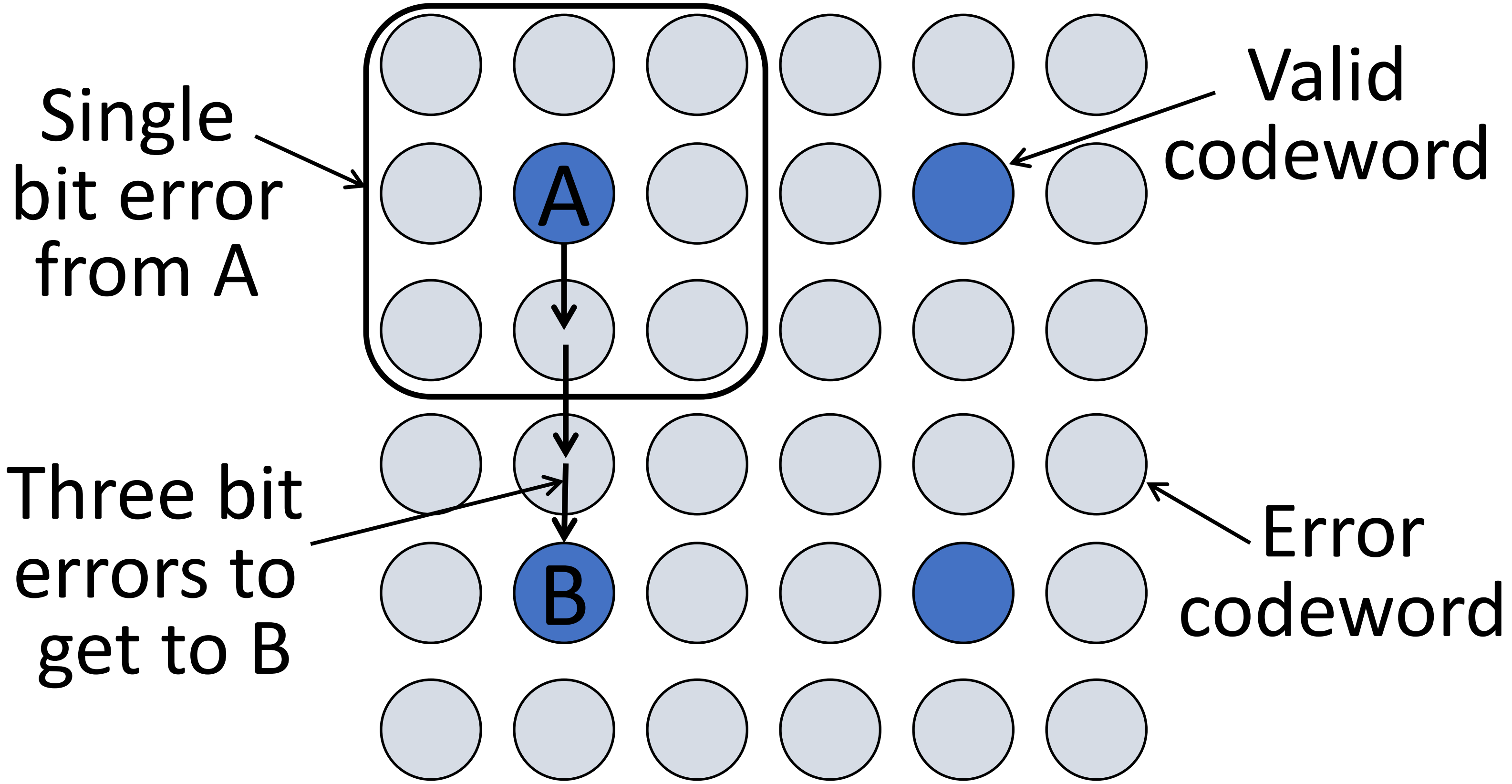
If we assume errors are only 1 bit, we can correct mapping an error to the closest valid codeword

- Works for d errors if HD ≥ 2d + 1

Intuition (2)



Intuition (3)



Single bit error from A
A
Valid codeword
Three bit errors to get to B
B
Error codeword

Hamming Code

Method for constructing a code with a distance of 3

- Uses $n = 2^k - k - 1$, e.g., n=4, k=3
- Put check bits in positions p that are powers of 2, starting with position 1
- N-th check bit is parity of bit positions with n-th LSBit is same as p's

Plus an easy way to correct [soon]

Hamming Code (2)

Cheat sheet

1: 0001

2: 0010

3: 0011

4: 0100

5: 0101

6: 0110

7: 0111

- Example: data=0101, 3 check bits
  - 7 bit code, check bit positions 1, 2, 4
  - Check 1 covers positions 1, 3, 5, 7 (LSB is 1)
  - Check 2 covers positions 2, 3, 6, 7 (2nd LSB is 1)
  - Check 4 covers positions 4, 5, 6, 7 (3rd LSB is 1)

0	1	0	0	1	0	1
1	2	3	4	5	6	7

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

Hamming Code (3)

- 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

$\longrightarrow \underline{0}\ \underline{1}\ 0\ \underline{0}\ 1\ 0\ 1$

1 2 3 4 5 6 7

p_1= p_2=

p_4=

Syndrome =

Data =

Hamming Code (6)

- Example, continued

$$\longrightarrow \underline{0}\ \underline{1}\ 0\ \underline{0}\ 1\ 0\ 1$$

1 2 3 4 5 6 7

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

$\longrightarrow$ $\underline{0}\ \underline{1}\ 0\ \underline{0}\ 1\ {\color{red}1}\ 1$

1 2 3 4 5 6 7

p_1=

p_2=

p_4=

Syndrome =

Data =

Hamming Code (8)

- Example, continued

$\longrightarrow$ $\underline{0}$ $\underline{1}$ 0 $\underline{0}$ 1 1 1

1 2 3 4 5 6 7

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!)

Hamming Code (9)

- Example: bad message 0100111
  - 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

0 1 0 0 1 1 1 ⟶

1 2 3 4 5 6 7

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

Hamming Code (10)

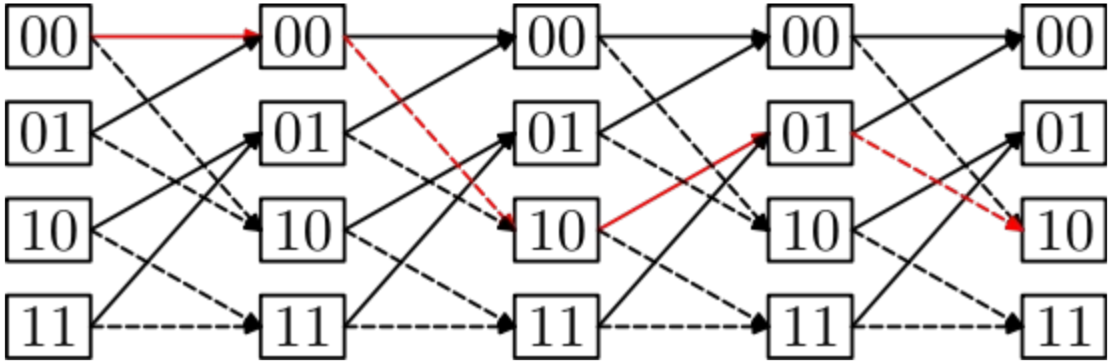
- Example: bad message 0100111
  - 7 bit code, check bit positions 1, 2, 4
  - Check 1 covers positions 1, 3, 5, 7
  - Check 2 covers positions 2, 3, 6, 7
  - Check 4 covers positions 4, 5, 6, 7

0	1	0	0	1	1	1	→
1	2	3	4	5	6	7	

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

Other Error Correction Codes

- Real codes are more involved than Hamming
- E.g., Convolutional codes (§3.2.3)
  - Take a stream of data and output a mix of the input bits
  - Makes each output bit less fragile
  - Decode using Viterbi algorithm (uses bit confidence values)



Detection vs. Correction

Example:

- 1000 bit messages with a bit error rate (BER) of 1 in 10000

Which is better will depend on the pattern of errors

Detection vs. Correction (2)

Assume bit errors are random

- Messages have 0 or maybe 1 error (1/10 of the time)

Error correction:

- Need 10 check bits per message ($1000 <= 2^{10} – 10 – 1$)
- Overhead: 10 bits per message

Error detection:

- Need 1 check bits per message plus 1000 bit retransmission
- Overhead: 101 bits per message

Detection vs. Correction (3)

Assume errors come in bursts of 100

- Only 1 or 2 messages in 1000 have significant (multi-bit) errors

Error correction:

- Need >>100 check bits per message
- Overhead: >> 100 bits per message

Error detection:

- Need 32 check bits per message (say, CRC-32) plus 1000 bit resend 2/1000 of the time
- Overhead: 34 bits per message

Detection vs. Correction (4)

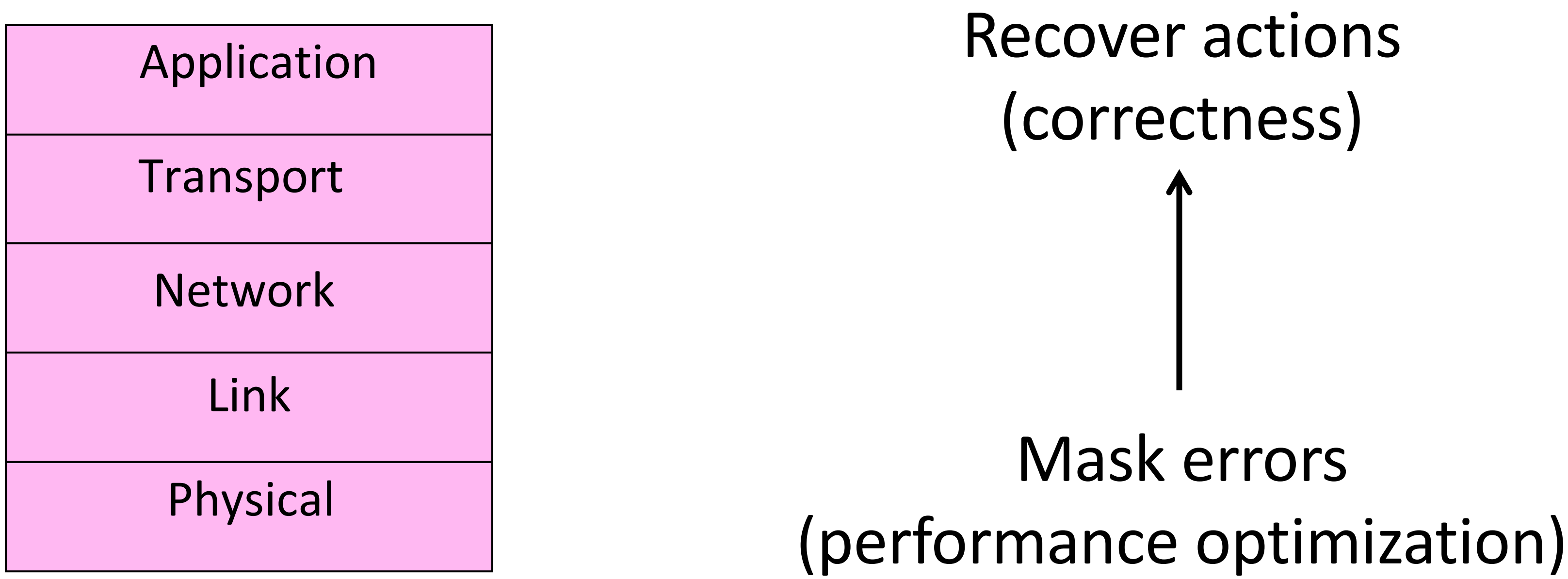
- Error correction:
  - Needed when errors are expected
  - Or when no time for retransmission
- Error detection:
  - More efficient when errors are not expected
  - And when errors are large when they do occur

Error Correction in Practice

- Heavily used in physical layer
  - Used for demanding links like 802.11, DVB, WiMAX, power-line, …
  - Convolutional codes widely used in practice
- Error detection (w/ retransmission) is used in the link layer and above for residual errors
- Correction also used in the application layer
  - Called Forward Error Correction (FEC)
  - Normally with an erasure error model
  - E.g., Reed-Solomon (CDs, DVDs, etc.)

Error Correction in Practice (2)

- Everywhere! It is a key issue
  - Different layers contribute differently



Multiple Access

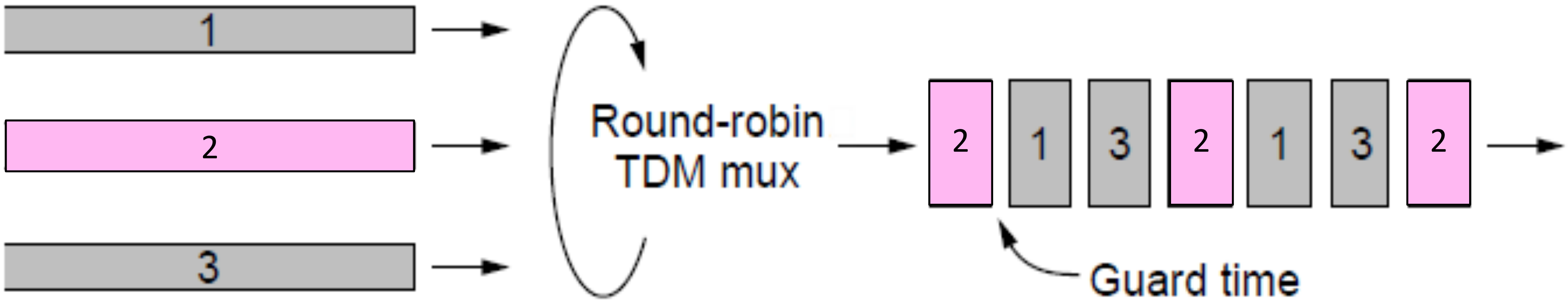
Topic

- 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)

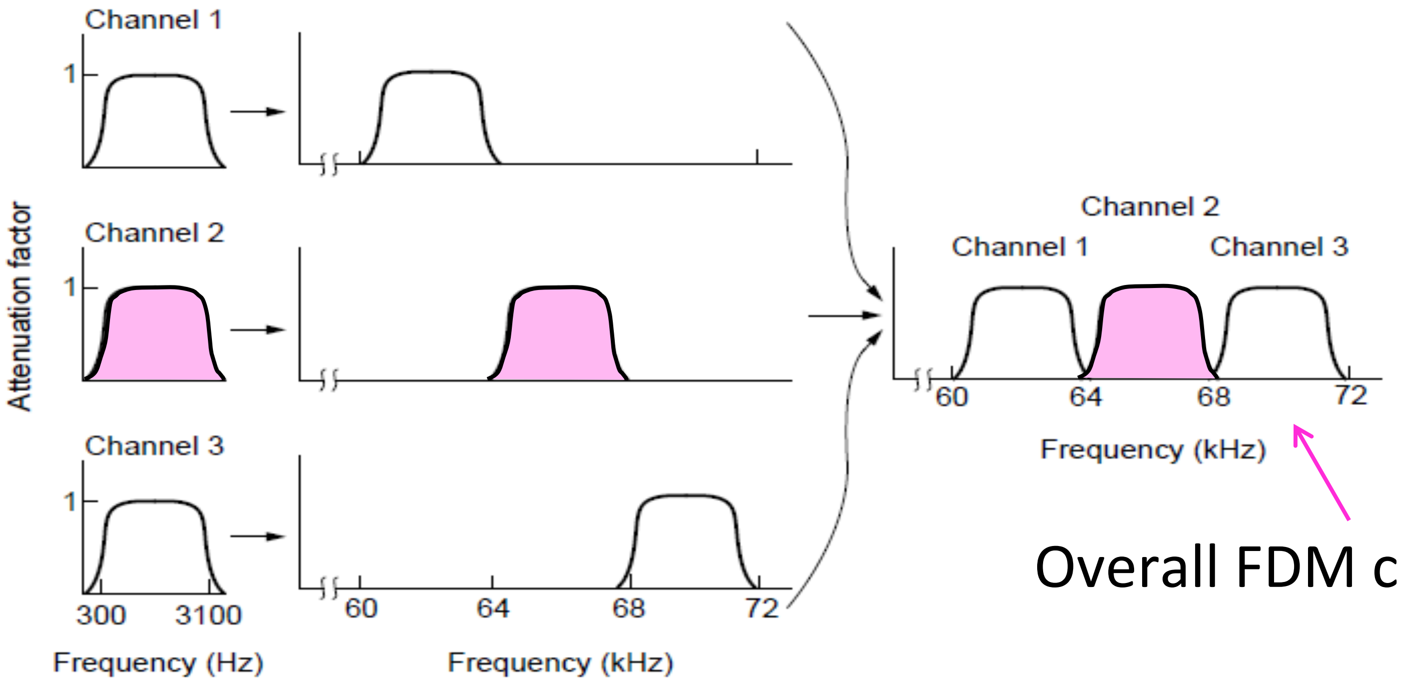
Time Division Multiplexing (TDM)

- Users take turns on a fixed schedule



Frequency Division Multiplexing (FDM)

- Put different users on different frequency bands



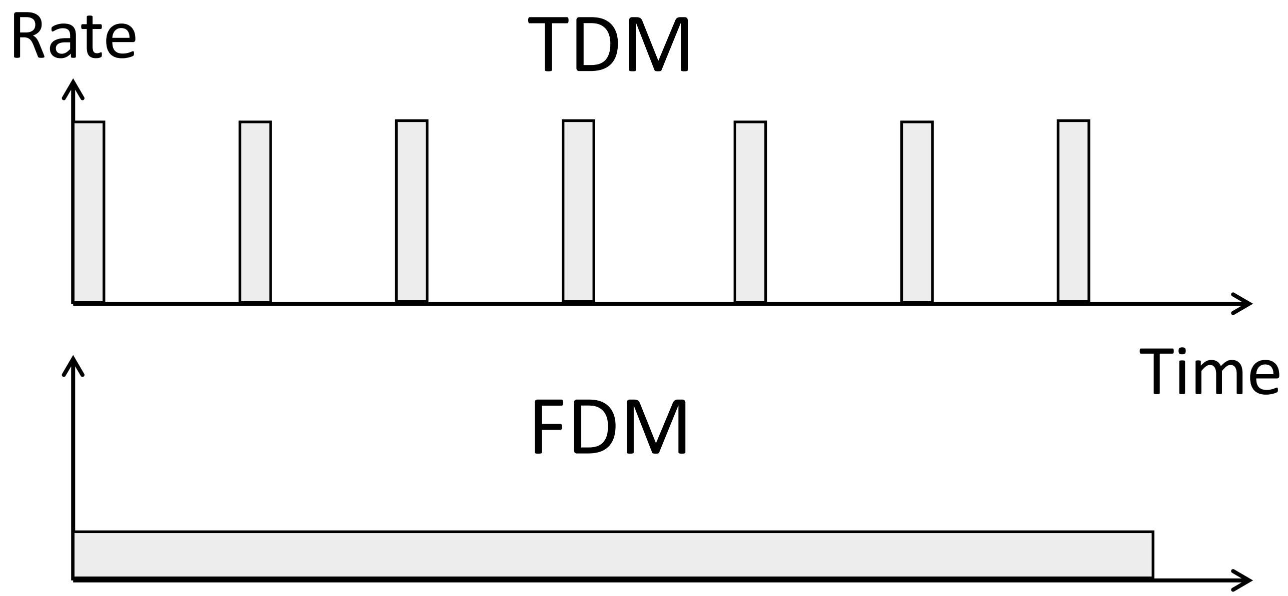
Overall FDM channel

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

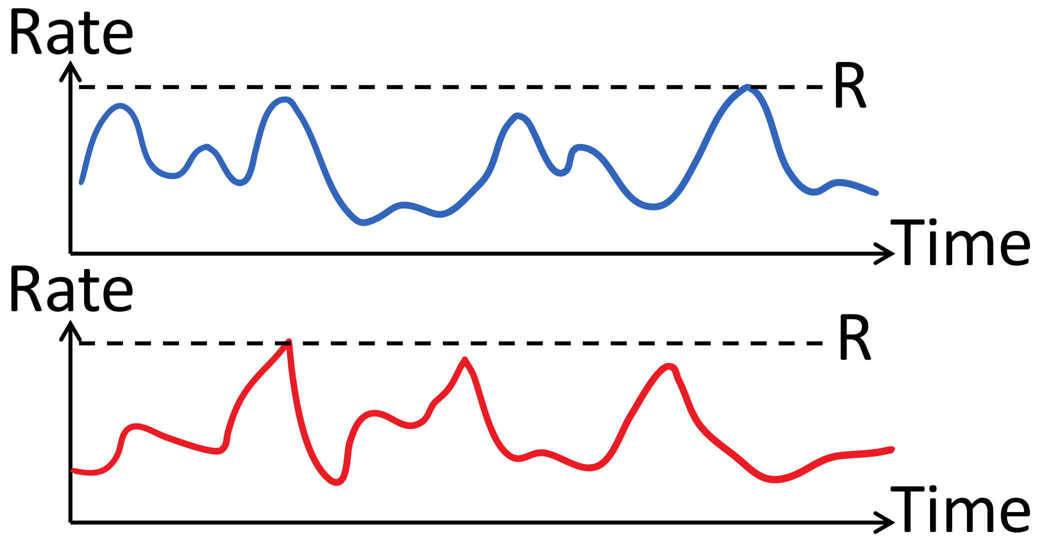


TDM/FDM Usage

- 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

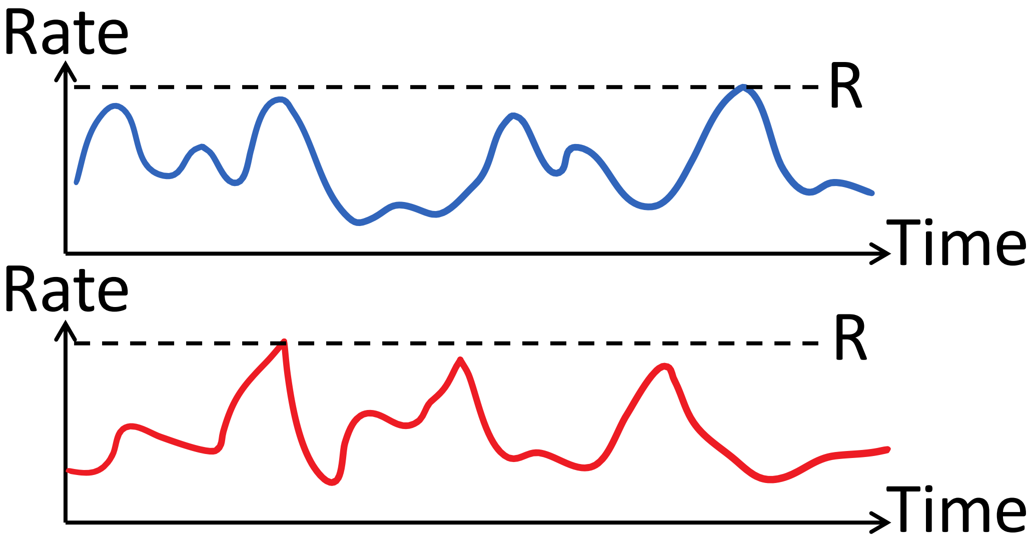
Multiplexing Network Traffic

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



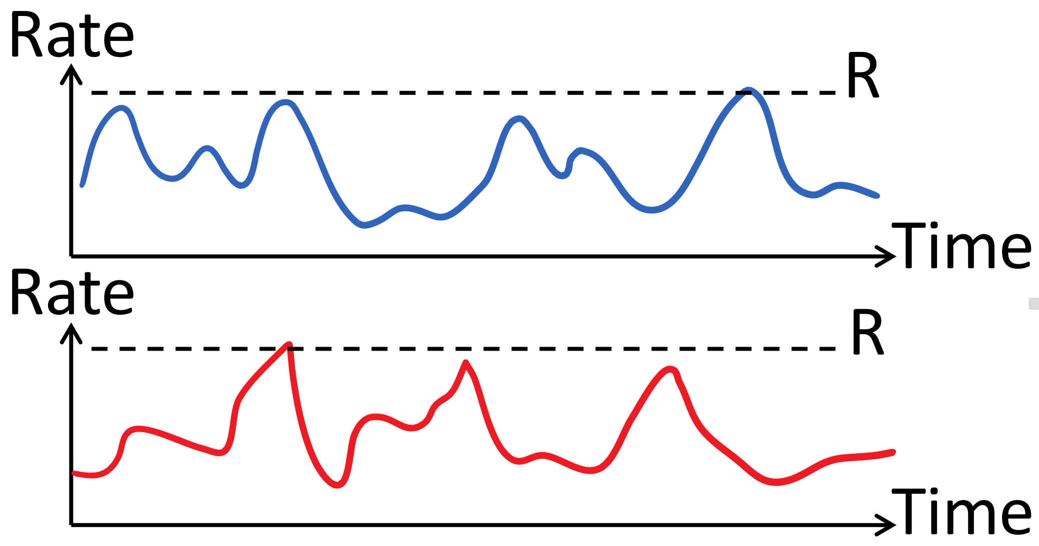
Multiplexing Network Traffic (2)

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

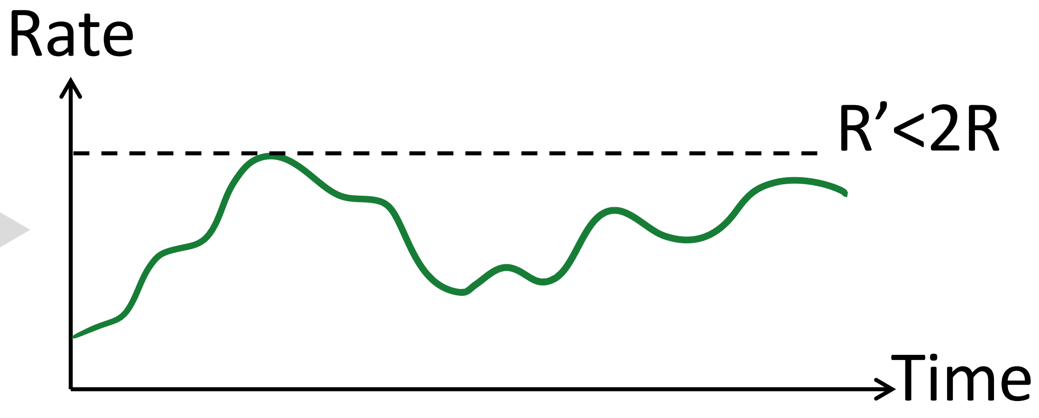


Multiplexing Network Traffic (3)

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



Two users, each need R



Together they need R’ < 2R

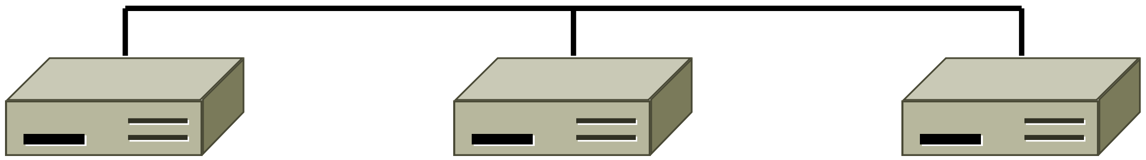
How to control?

Two classes of multiple access algorithms

- **Centralized**: Use a “Scheduler” to pick who transmits and when
  - Scales well and is usually efficient, but requires setup and management
  - Example: Cellular networks (tower coordinates)
- **Distributed**: Have participants “figure it out” via some mechanism
  - Operates well under low load and easy set up but scaling efficiently is hard
  - Example: WiFi networks

Distributed (random) Access

- How do nodes share a single link? Who sends when?
  - Explore with a simple model



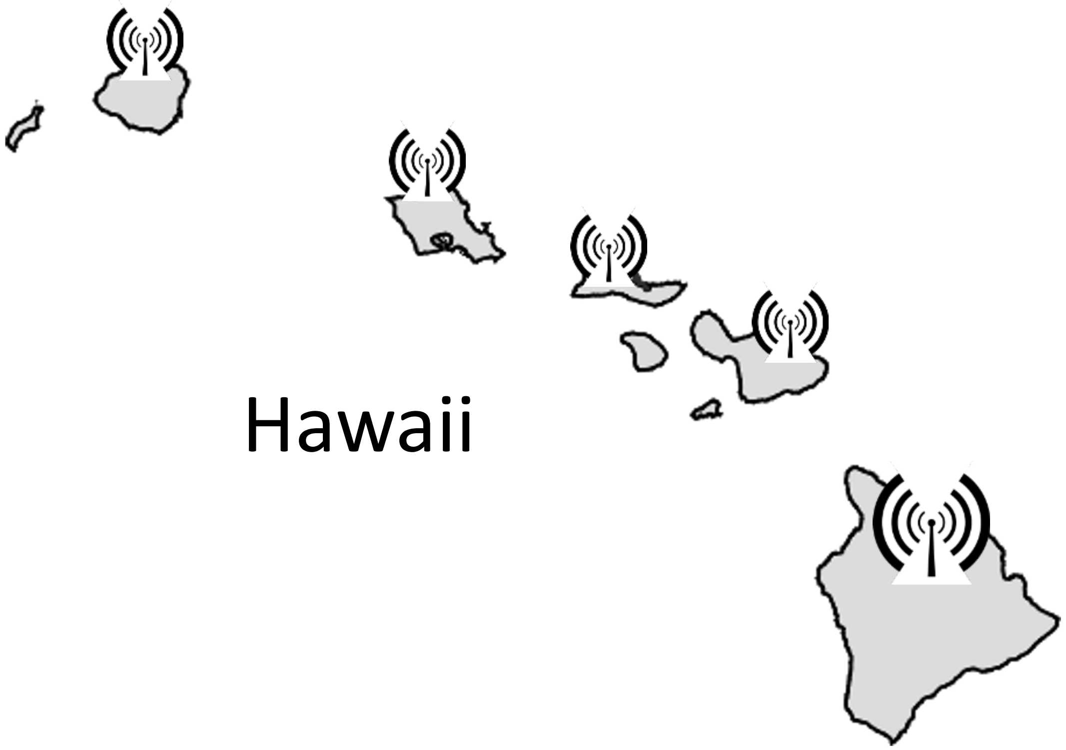
- Assume no-one is in charge
  - Distributed system

Distributed (random) Access (2)

- We will explore random <u>multiple access control</u> (MAC) protocols
  - This is the basis for <u>classic Ethernet</u>
  - 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 …



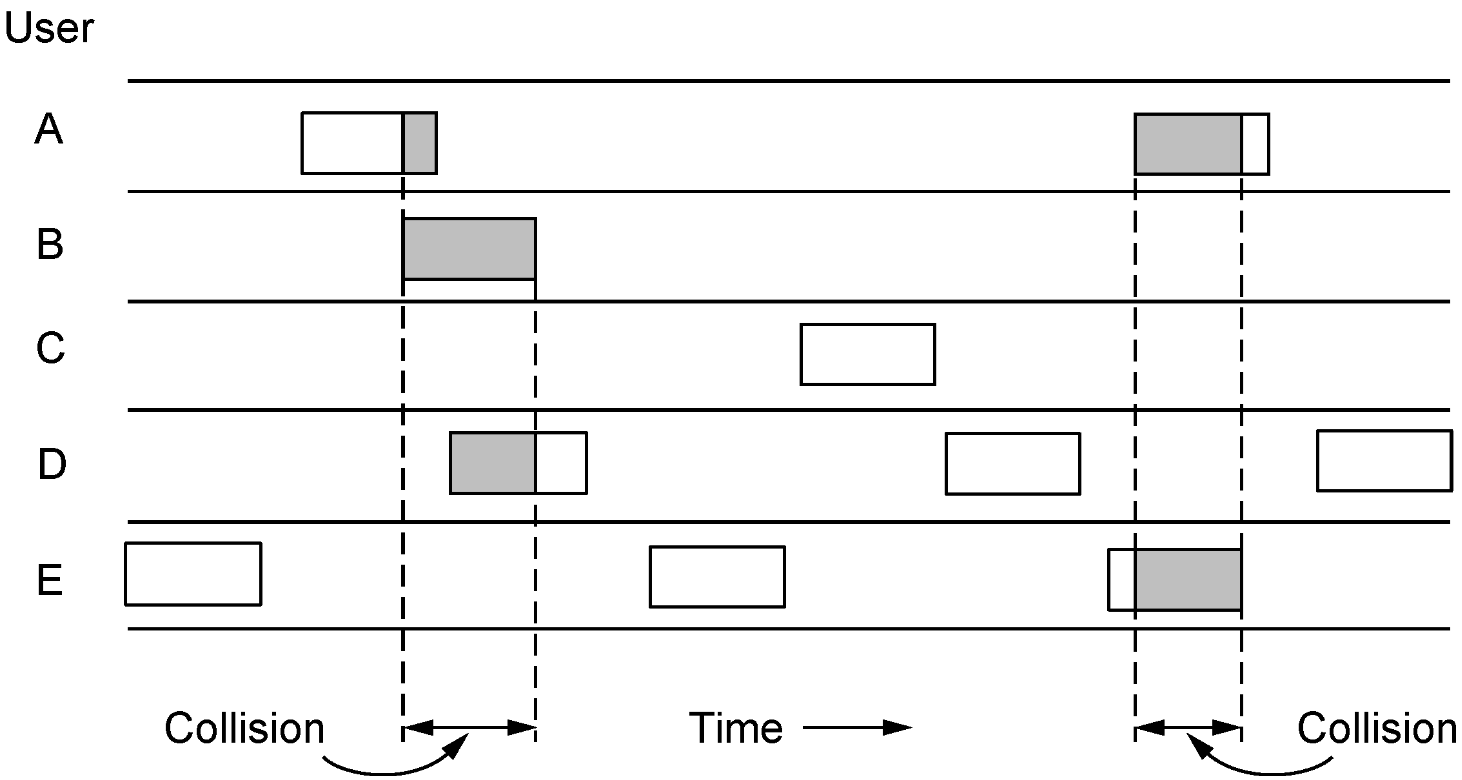
ALOHA Protocol

- 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…

- Limitations?



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