EEE 432
Introduction to Data Communications

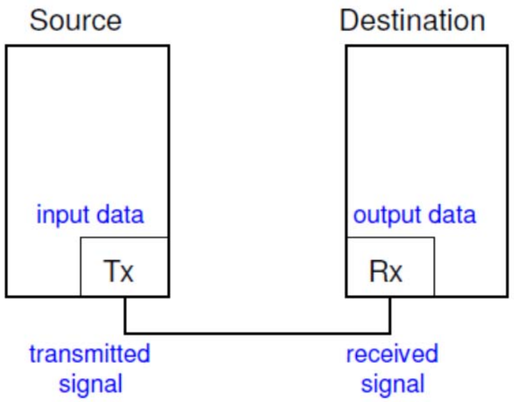
Asst. Prof. Dr. Mahmut AYKAÇ

DIGITAL DATA COMMUNICATION TECHNIQUES

Course Information

1. Data Communications and Networks
2. Data Transmission
3. Transmission Media
4. Signal Encoding Techniques
5. **Digital Data Communication Techniques**
6. Multiplexing
7. Networking and Protocol Architectures
8. Switching
9. Routing in Switched Networks
10. LANs and WANs
11. Ethernet
12. The Internet

Physical and Data Link Layer → Challenges with Link Communications



- How to convert information into transmittable signals?
- What are the characteristics of signals?
- What transmission media to use?
- How to efficiently encode data as signals?
- **How to know who is at other end?**
- **How to deal with errors?**
- **How to share media amongst two or more transmitters?**

Physical and Data Link Layer

- Researchers, designers, standards, implementations often separate functionality into **layers.**

Physical Converting data (e.g. bits) into signals to be sent across the link

Data Link Ensuring link is ready for data transmission, reliable/efficient transmission of data.

Digital Data

- Many communication systems today carry digital data
  - ➢Analog data often converted to digital: voice, video
  - ➢Analog or digital signals
- Challenges for digital data communications:
  - ➢How to split data up?
  - ➢How to deal with errors?
  - ➢How to deal with different types of devices?
- Solutions are often independent of how **physical** signals transmitted: **Data Link** layer

Framing

- Communication protocols group data into separate pieces

➢ What is a protocol? Rules to define how two or more entities communicate, including format of messages

➢ Why group into pieces? faster recovery from errors, fairer sharing of medium amongst multiple users, . . .

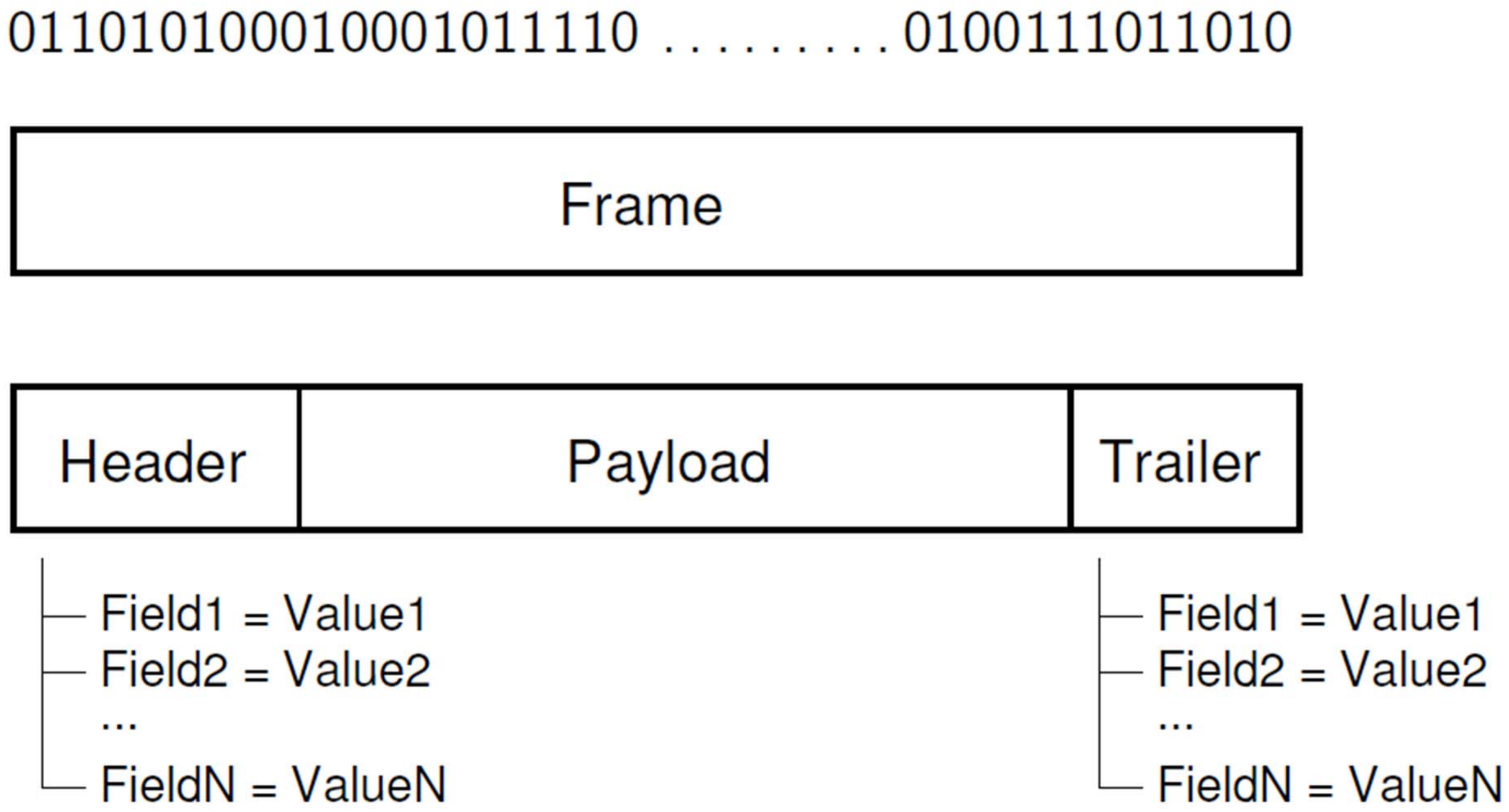
- At the data link layer the pieces commonly called **frames**
- Information in a frame often separated into parts:
  - ➢ **Header** control information at start of frame; used to support protocol operation
  - ➢ **Payload** actual data
  - ➢ **Trailer** control information at end of frame; used to support protocol operation
- Not all parts in all frame, e.g. Header + Payload; Header + Payload + Trailer; Header only

Frame Header (and Trailer)

What is Purpose of Header?

- Contains information to support protocol operation
- Sender includes information in header so receiver can correctly process the data and optionally respond
- Information often split into **fields**; each field has a value
- Number, meaning and size of fields defined in standard
  - IEEE 802.11 defines wireless LAN frame header and trailer fields
- Many protocols have default, fixed size header, with optional extra fields
  - IEEE 802.11 MAC Data: typically 24 byte header and 4 byte trailer; other sizes possible

General Frame Structure



01101010001000101110 0100111011010

Frame

Header

Payload

Trailer

Field1 = Value1
Field2 = Value2
...
FieldN = ValueN

Field1 = Value1
Field2 = Value2
...
FieldN = ValueN

Frame Header (and Trailer)

- ***Example Header Fields***
- Source and destination addresses, e.g. MAC address
- Frame, payload, header lengths
- Sequence numbers, e.g. data sequence, ACK number
- Protocol version
- Checksums, error detection codes
- Frame types, e.g. DATA, ACK, Beacon
- Flags
  - ➢ Single bit values
  - ➢ 1: flag is set/true, e.g. feature is on
  - ➢ 0:flag is unset/false, e.g. feature is off

Protocol Performance →Performance Metrics

- Metrics: Ways to measure the performance of communication systems
- How do we use metrics?
  - ➢ Measure the actual performance of real systems
  - ➢ Calculate/estimate to predict performance of planned systems
- Represented using different statistics:
  - ➢ Instantaneous
  - ➢ Average (mean) over some time
  - ➢ Maximum (peak), minimum, standard deviation, variance, . . .
- Some metrics we have seen already: bandwidth (Hz), SNR (dB), data rate/capacity (b/s)
- Following slides show common metrics in digital data communications

Data Rate

Definition

Rate at which data is delivered from one point to another

Other/Related Names

Bit rate, capacity, signalling rate, bandwidth

Units

bits per second (b/s or bps)

Examples

- My computer LAN card can send 100Mb every second; all bits arrive at destination: Data rate = 100Mb/s

Delay

Definition

Time it takes to get data from one point to another

Other/Related Names

Latency; Response time, Round Trip Time (RTT)

Units

seconds

Examples

- I send an email at 10:00am; it arrives at destination at 10:03am: **Delay** = 3 minutes
- At time 1.4s, I click on a webpage link; at time 2.6s the webpage is fully displayed on my browser: **Response Time** = 1.2s
- **Round-trip time (RTT)** is the duration, measured from when a browser sends a request to when it receives a response from a server. Example: ping message.

Error Rate

Definition

Fraction of data sent that doesn't get delivered to destination

Other Names

Bit Error Rate (BER), Frame Error Rate (FER), Packet Error Rate (PER), Loss rate

Units

none (fraction, percentage)

Examples

- I send a copy of an email to 100 students; 5 students do not receive the email: Error rate = 0.05 = 5%
- For every 1000 bits sent across a link, on average 23 bits arrive in error: BER = 0.023 = 2.3%

Overhead

Definition

Amount of additional data needed in order to deliver useful data. (Header+ Trailer)

Other Names

-

Units

bits

Examples

- For every 8 bits of data, a 2-bit parity check is added: Overhead = 2bits
- A packet contains 1000B of data, a 25B header and 25B trailer: Overhead = 50B

Throughput

Definition

Rate at which useful data (payload) is delivered to destination

Other Names

Goodput, Bandwidth

Units

bits per second (b/s or bps)

Example

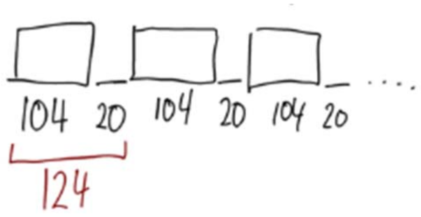
- Downloading a 12MB file from website takes 16 seconds: Throughput = 6Mb/s
- WiFi link has data rate of 54Mb/s. For every 500 Bytes of data sent, there is additional 200 Bytes of overhead plus 20us spent not sending.

Frame=500+200 = 700B,   Transmit time= 700B/(54Mb/s) = 104µs

Total time to transmit 500B is = 104+20 = 104µs

Throughput = 500B/124 µs = 32.3 Mb/s or simply…

Throughput = 54 * $\frac{500}{700} * \frac{104}{124}$ = 32.3 Mb/s



Efficiency (η)

Definition

Fraction of time spent using a resource for intended purpose

Other Names

Utilization

Units

none (fraction, percentage)

Example

- I pay 1000 TL per month for 10Mb/s home Internet. On average, each month I download at 2Mb/s: η = 0.2 = 20%
- WiFi link has data rate of 54Mb/s, but throughput of 20Mb/s: η = 0.37 = 37%
- For every 1000B of data sent, there is an overhead of 200B: η = 0.80 = 80%
- For the example in the previous slide, η = 32.3/54 = 60%

Delay

- Time it takes to get data from one point to another
- Delay is additive
- Four components that contribute to total delay:

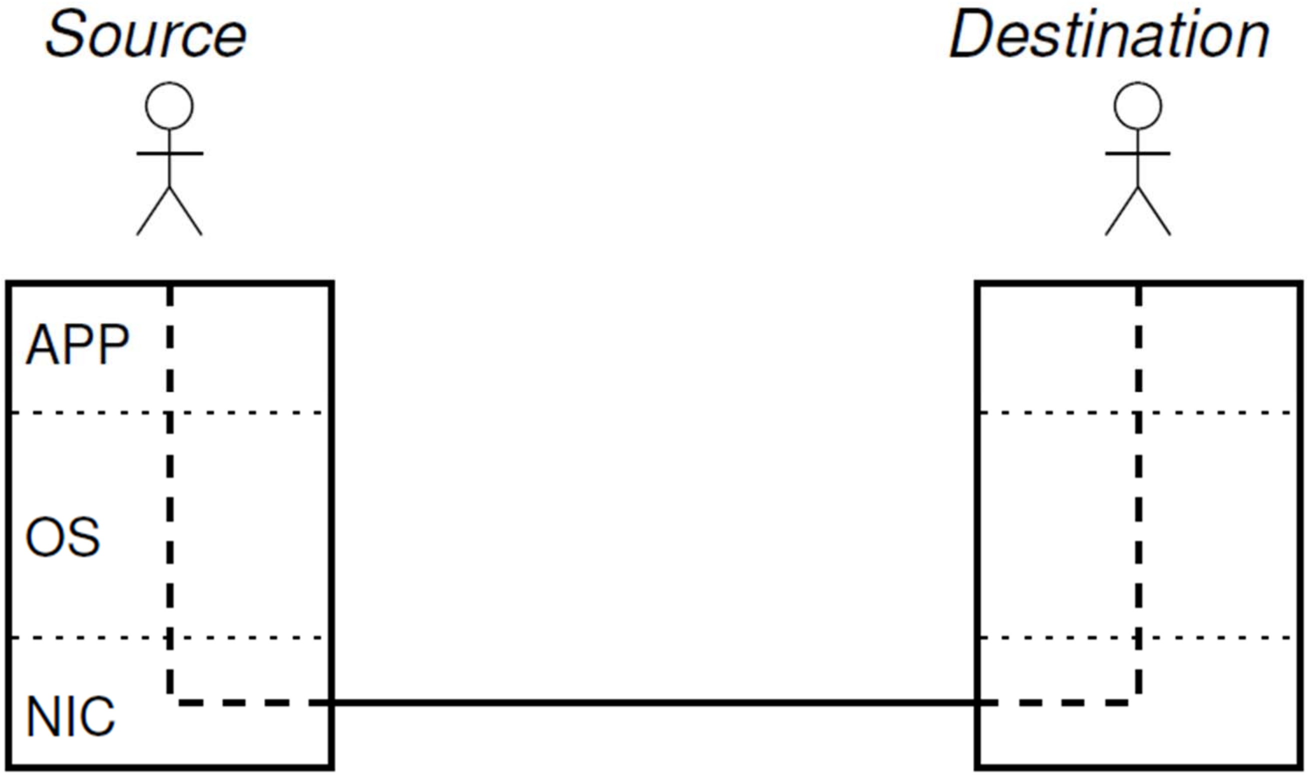
1. Transmission delay: time to transmit data on to link (T_{trans})

2. Propagation delay: time for a signal element (or bit) to propagate across link(T_{prop})

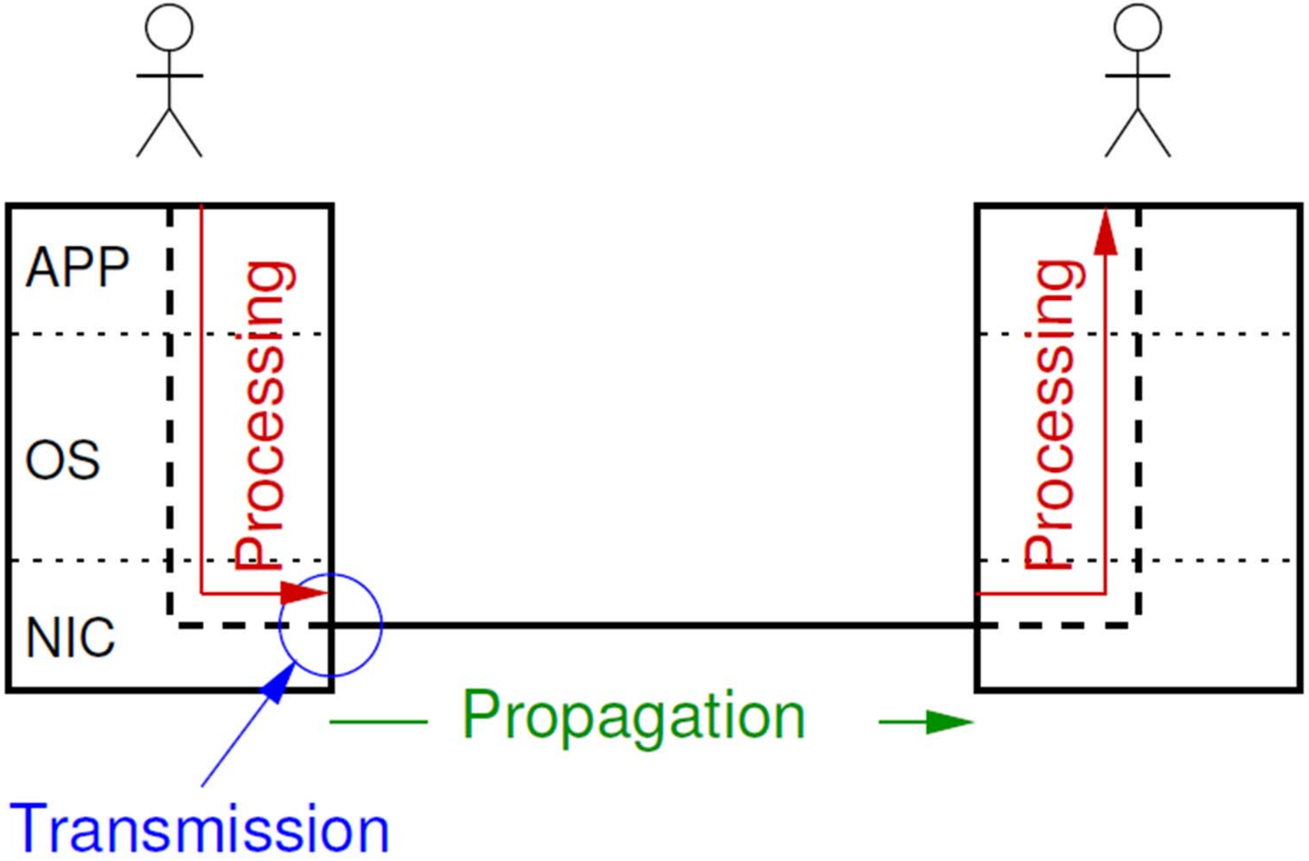
3. Processing delay: time for device to process data (T_{prop})

4. Queuing delay: time data spent waiting in queue (memory) inside device (T_{queue})

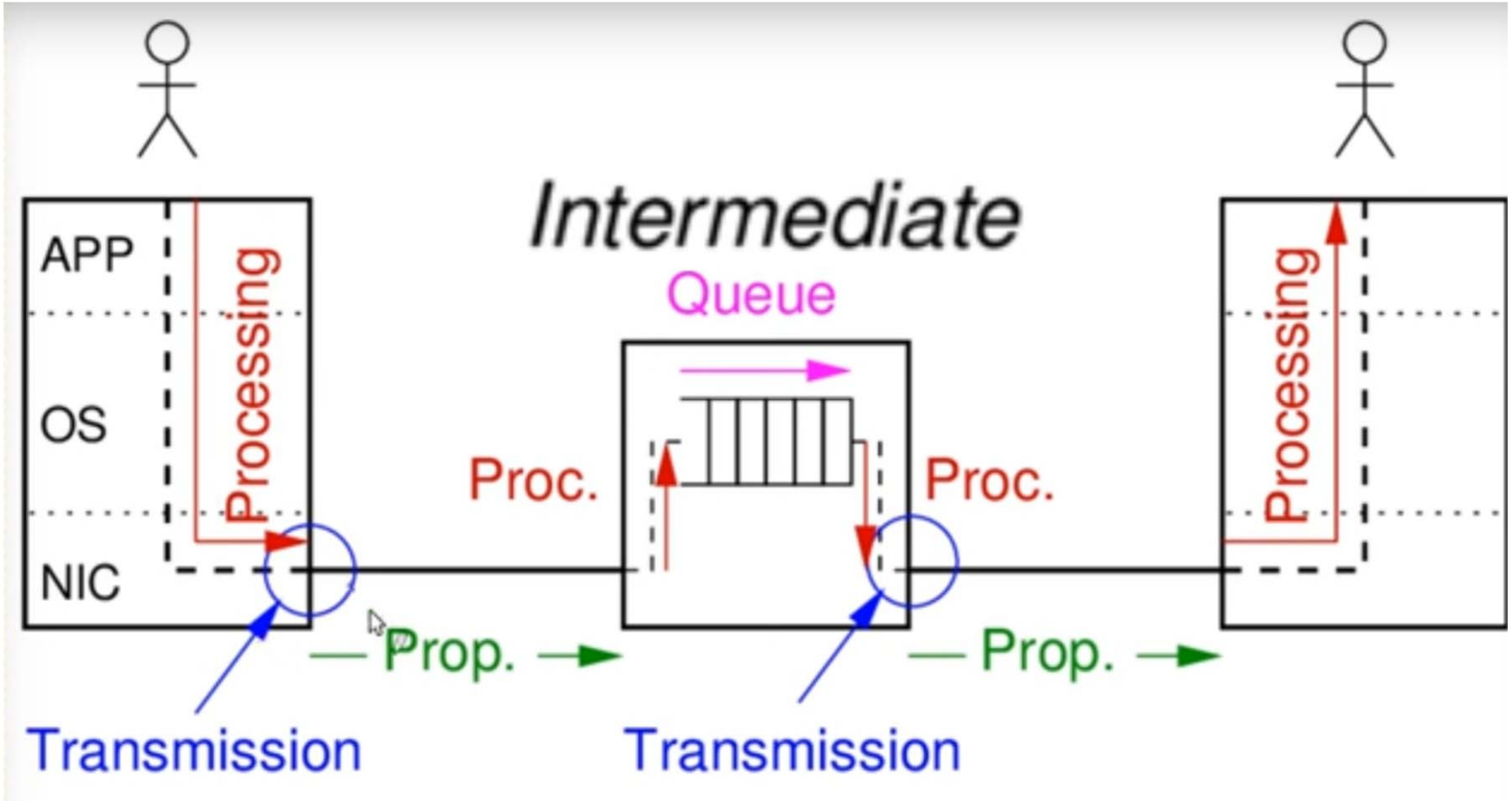
Delay Components in a Link



Delay Components in a Link



Delay Components in a Link (Queue case)



Determining Delay

- **Transmission, Bandwith or Store-and-Forward Delay**

➢ Number of bits to send, b [bits]

➢ Link data rate, r [bits per second]

➢ Transmission delay, $T_{trans} = \frac{b}{r}$

- **Propagation Delay**

➢ Link distance, d (meters)

➢ Speed of signal propagation, s (meters per second, m/s)

➢ Propagation delay, $T_{prop} = \frac{d}{s}$

  ➢ Unless otherwise stated, $s = c = 3 \times 10^8$ m/s

Determining Delay

- **Processing Delay**
- ➢ Depends on amount of data to process, software implementation, computer hardware, and other activities of computer
- ➢ Often very small compared to transmission and propagation delay
- ➢ Unless otherwise stated, assume T_{proc} = 0 s

- **Queuing Delay**
- ➢ Depends on amount of data arriving from other users and leaving device, and queuing scheme (e.g. FIFO, priority)
- ➢ Can be significant in large networks, e.g. the Internet
- ➢ Unless otherwise stated, assume T_{queue} = 0 s

Delay Examples

Ex: R is an intermediate point (router, repeater etc.) between A and B. All the connections are made of copper whose transmission speed is 0.7c. If 100B of digital data wants to be transmitted by using 8Mb/s technology. What will be the total time for the data to arrive point B? (Unmentioned delays are assumed to be zero)

A-------------------→R---------------→B $T_{total} = 2\times T_{trans} + 2\times T_{prop}$

100km 100km

$T_{trans} = \frac{100\times8}{8\times10^6} = 100\mu s$ $T_{prop} = \frac{100\times10^3}{2{,}1\times10^8} = 476\mu s$

$T_{total} = 2\times 100\mu s + 2\times 476\mu s = \boldsymbol{1152\mu s}$

Dealing with Errors, Bit Errors

In digital transmission systems errors occur when a bit is altered between transmission and reception (If 0 is transmitted and if 1 is received or vice versa **bit error** occurs)

Single-bit errors

- Only one bit altered, surrounding bits not affected
- Caused by random noise

Error burst

- A group of bits near each other are affected (in error)
- Caused by impulse noise or fading
- Effects of burst errors are greater at higher data rates

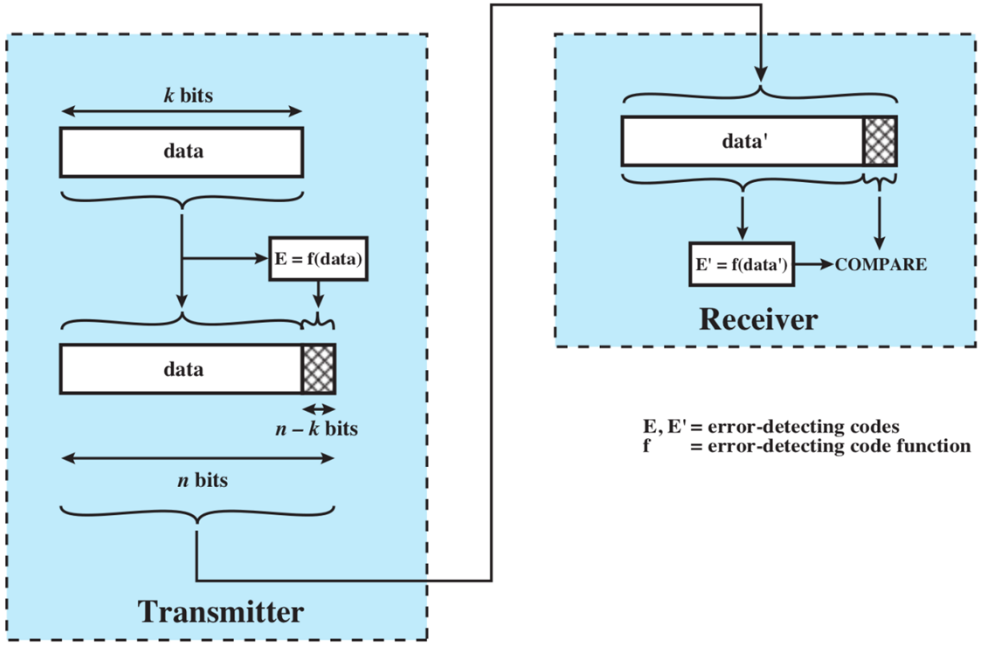
Require methods to detect errors, and correct where possible

Error Detection

Transmitter adds extra information to transmitted data, i.e. an error detecting code

Receiver recalculates the error detecting code from received data, and compares to received error-detecting code

If the same, good. If not, then error (in data or code). Still a chance that an error is not detected



Transmitter has k- bits of data to send. It applies some functions on the data, get error detecting code (E) and it attaches the value of this calculated data to the original data.

Now we have n-k bits of error detecting code. Then n-bits of data is transmitted across our link leading to the receiver.

Receiver receives n-bits but it must know where the split is, which means the first k-bits of data is the original data, it applies the same function that the transmitter applies and obtains E'.

Then, it compares E and E'. If they match, no bit errors. If not, bit error occurs.

Error Detection with Parity Check

Odd-parity check: append parity bit to block of data; resulting set of bits has odd number of ones.

Receiver detects an error if receiver bits has unexpected number of ones (transmitter and receiver both know parity scheme being used)

Parity Check Example

Assume character ‘q’ is to be sent using odd-parity check. What is transmitted? What happens if the last bit is corrupted? What about the last two bits?

Sol’n: ASCII code of q is 1110001 (There are total 4 1s inside so the transmitter adds a 1 at the start of the data and transmits)

Transmitted data: **1**1110001.

- When the receiver takes data with last bit corrupted:**1**1110000. Receiver counts the total number of 1s (4) and finds out it is even, so decides there is an error.
- When the receiver takes data with last 2 bits corrupted:**1**1110010. Receiver counts the total number of 1s (5) and finds out it is odd and assumes there is no error. Error detection does’nt work in this case.

As a result, error detection by parity check for 1 bit error works but for 2 bits error, doesn’t work.

Error Detection with Cyclic Redundancy Check (CRC)

- Parity checks are not good when multiple bit errors occur
- CRC is a powerful, commonly used error detection scheme
- Approach:
  - k bits of data to send
  - Constant divisor known by transmitter/receiver, n - k + 1 bits (Pattern, P)
  - Append n - k bits to data such that no remainder when divided by divisor (Frame Check Sequence, FCS)
  - Transmit n bits
  - Receiver divides received n bits by divisor; if remainder is not zero, error detected
  - Division is done through Module 2 arithmetic (XOR operation)
- Length and value of divisor is important for error detection capabilities (e.g. chance that one or more errors go undetected)
- CRC used in: Ethernet, HDLC, SATA, CDMA, PNG images, SD cards, . . .

Example

Ex: Data= 1011010010 (10 bits), Pattern = P= 110010 (6 bits), pattern and its number of bits can be chosen randomly. But remember increasing the number of bits of the overhead in the data frame will decrease efficiency. Therefore, FCS = 5 bits (because number of FCS bits = number of P bits - 1)

Step 1: Append zeros to the data, number of zeros must be equal to the number of bits in FCS. In the end of the process these zeros will edited with real FCS.

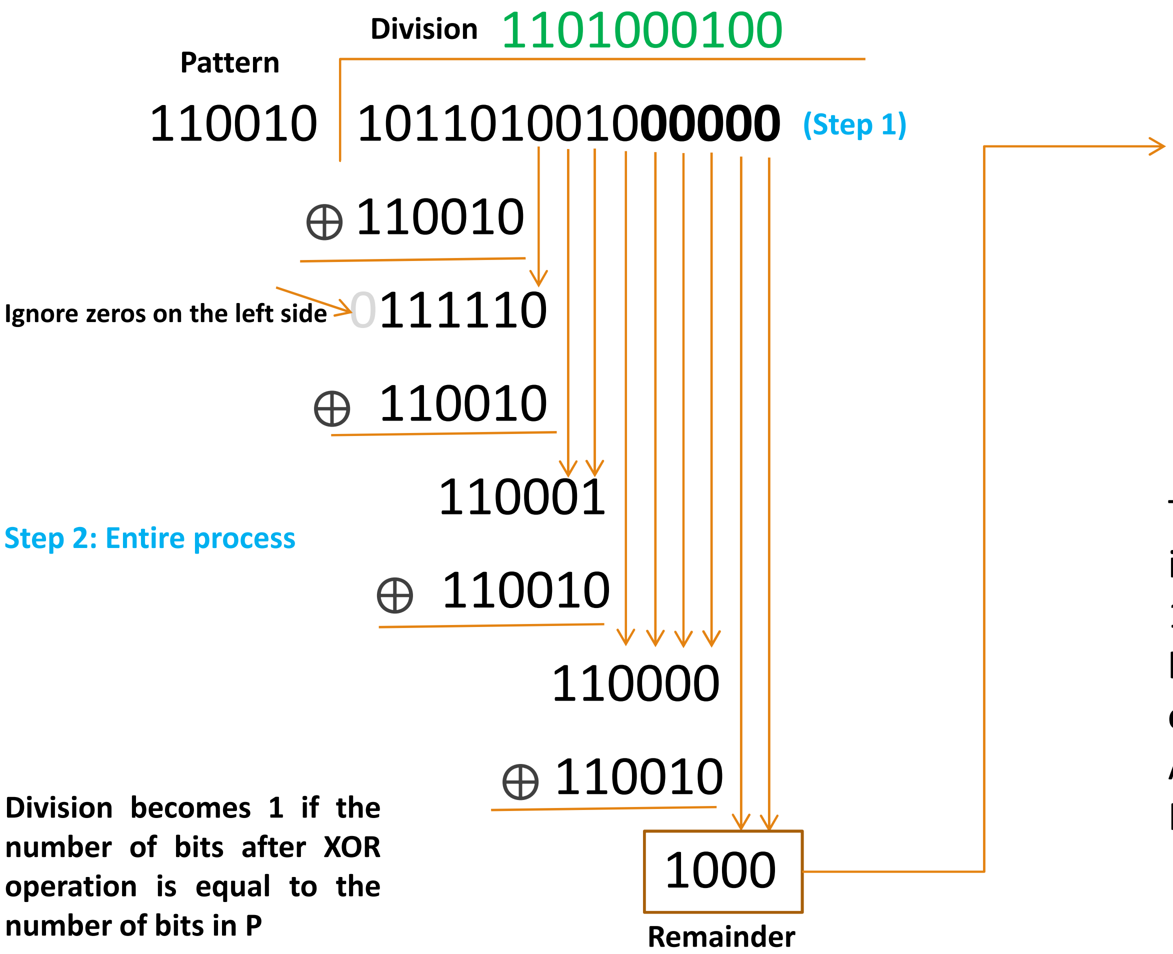
Therefore, new data becomes 1011010010**00000**

Step 2: Do modulo 2 (XOR) operation on the new data until the end of the new data, start from the leftmost bit.

Step 3: Remainder is the real FCS, append it to the data, obtain the data frame and transmit it to the channel. Equalize the number of bits if necessary.

Step 4: Receiver receives the data frame from the channel and do the similar process on it. If the remainder is zero, it means there is no error. Otherwise, there is error and detected.

Transmitter Side of the Example



Division 1101000100

Pattern

110010 | 101101001**00000** (Step 1)

⊕ 110010

Ignore zeros on the left side → 0111110

⊕ 110010

110001

Step 2: Entire process

⊕ 110010

110000

⊕ 110010

Division becomes 1 if the number of bits after XOR operation is equal to the number of bits in P

1000

Remainder

We stop at this point. Because we finish all the bits in the edited data. But the remainder will be used as the FCS. Therefore, we complete remainder to 5 bits to equalize the number of bits in FCS.

New Remainder = FCS= 01000

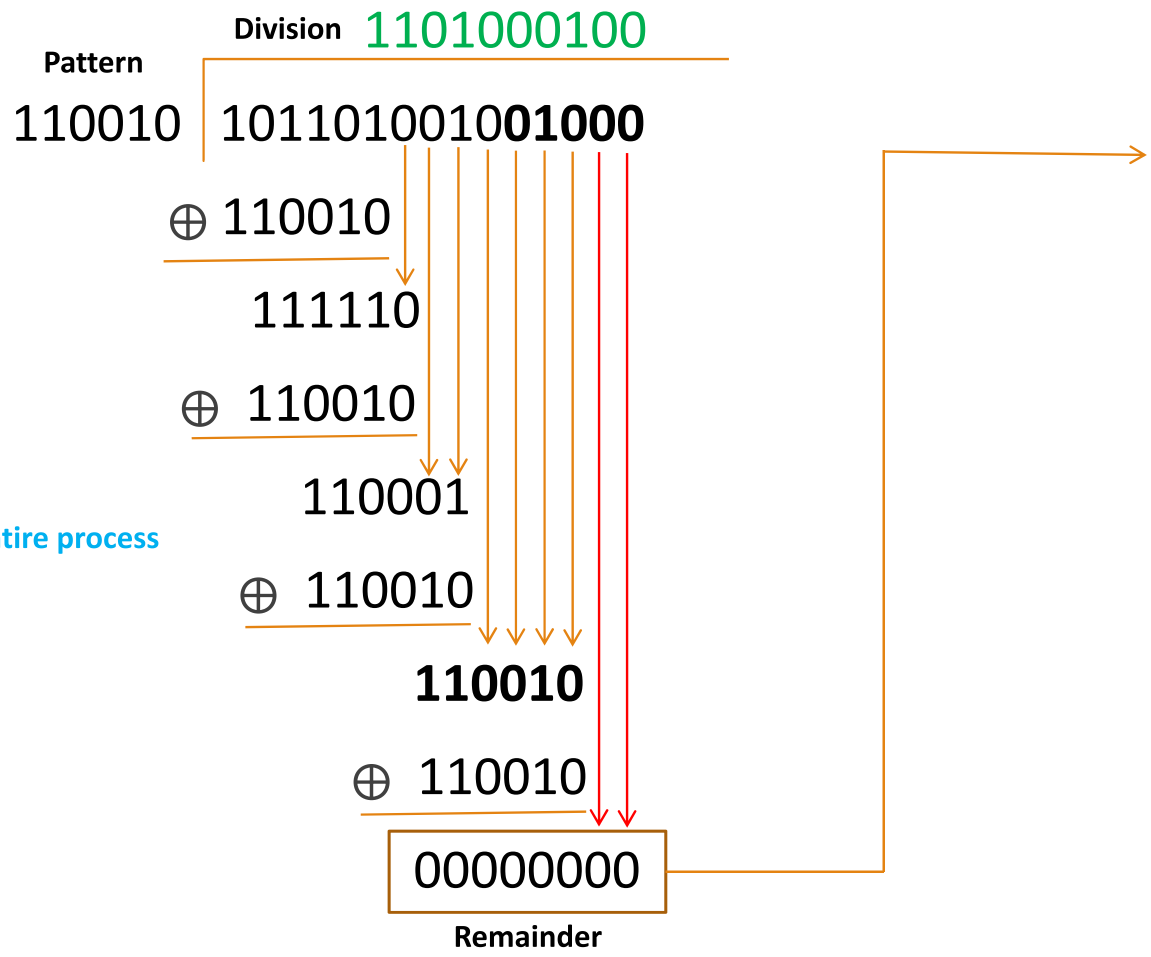
Transmitter appends the real FCS to the data and sends it to the receiver. Our final data becomes 1011010010**01000.** **(Step 3)**

How does the receiver know whether there is an error or not?

Assume the receiver receives the same finaş data

Receiver will also do a similar process…

Receiver Side of the Example



Division 1101000100

Pattern

110010 | 1011010010**01000**

⊕ 110010

111110

⊕ 110010

110001

⊕ 110010

110010

⊕ 110010

00000000

Remainder

When the remainder is all zeros. Receiver can be sure there is no error in transmission. Otherwise, there is an error and detected

Step 4: Entire process

Error Correction

- What to do when error detected at receiver?

- Ask transmitted to send again, i.e. retransmit
  - Can be inadequate if link has high delay or many errors, e.g. wireless/satellite links

- **Forward Error Correction:** sender sends a codeword (instead of data); codeword chosen such that if error detected, receiver can correct the error without retransmission

- Depending on encoding scheme and pattern of errors, receiver may: detect and correct errors; detect, but not correct errors; not detect errors

Forward Error Correction (FEC)

- Sender sends a codeword (instead of data); codeword chosen such that if error detected, receiver can correct the error without retransmission

- Depending on encoding scheme and pattern of errors, receiver may: detect and correct errors; detect, but not correct errors; not detect errors

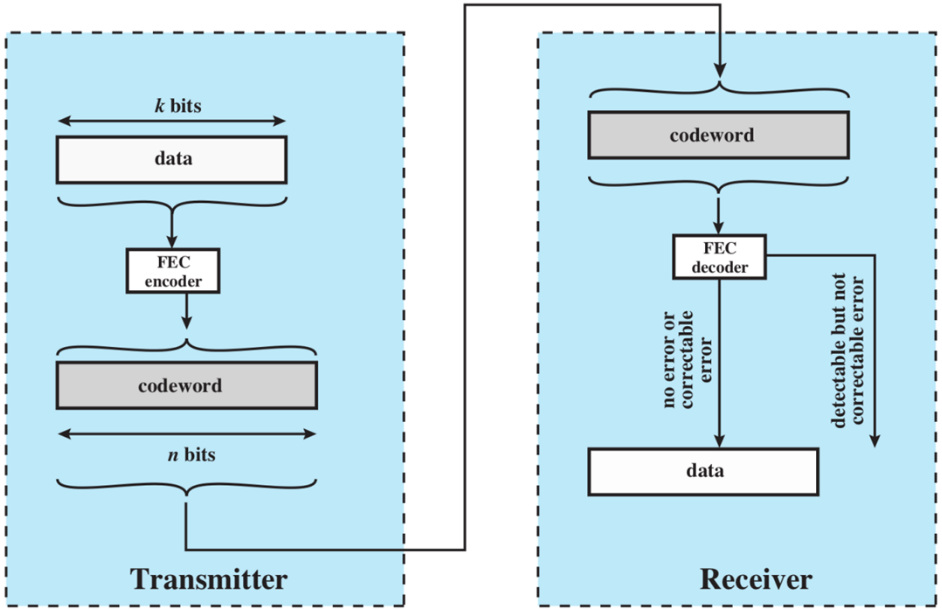


Fig. Forward Error Correction (FEC)

FEC with Hamming Distance

Hamming Distance

Number of bits of two n-bit sequences that differ

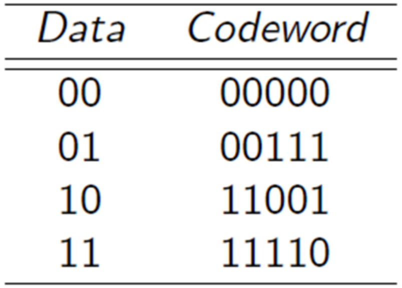
Ex: v1 = **0**1**1**0**1**1, v2 = **1**1**0**0**0**1, d(v1; v2) = 3 (a measure of how close two sequences are)

In FEC, we take the data that we want to transmit and instead of sending that data, we send a codeword which represents that data. So we have a mapping from data to codewords.

We transmit the codeword to receiver, receiver uses that codeword to detect if there are any errors (error detection). If there are errors it will try to correct. **Receiver and transmitter both know the data-to-codeword mapping table.**

Ex: 2-bits of data mapped to 5-bit codeword (k=2, n=5)

If received codeword invalid, assume valid codeword that is unique minimum Hamming distance from received codeword was transmitted.



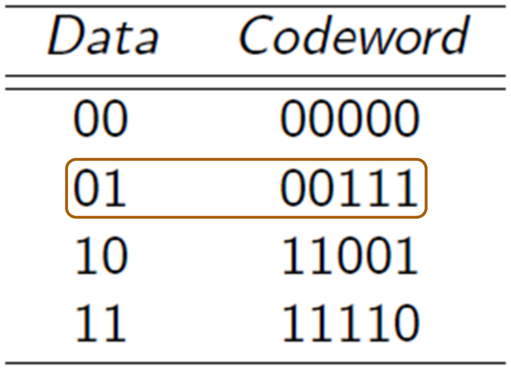
Data	*Codeword*
00	00000
01	00111
10	11001
11	11110

Table. Data-to-codeword table mapping, is randomly done

Error Correction Example 1

- Data to send: 01; no transmission error.

No transmission error means that there is no changing in the data transmission of Tx codeword



Data	*Codeword*
00	00000
01	00111
10	11001
11	11110

No error

Tx Data:01, Tx codeword:00111 → Rx codeword:00111, Rx Data:01

Error Correction Example 2

- Data to send: 01; 3rd bit transmitted is in error

Data	Codeword
00	00000
01	00111
10	11001
11	11110

If Rx codeword is not a valid one (not in the mapping table). Error is detected

Tx Data:01, Tx codeword:00111 **3rd bit in error** → Rx codeword:00**0**11, Corrected:00111, Rx data:01 (✓)

000**00**, d=2

00**1**11, d=1, Correction at this point,

11001, d=3

11110, d=4

We find the Hamming distance of received codeword (Rx codeword) and possible codewords in the mapping table. **The least and unique** Hamming distance will lead us to the correction

Error Correction Example 3

- Data to send: 01; 1st and 4th bit transmitted in error

Data	*Codeword*
00	00000
01	00111
10	11001
11	11110

Tx Data:01, Tx codeword:00111 $\xrightarrow{\textbf{1st\&4th bits in error}}$ Rx codeword:**1**01**0**1, Rx data: ??

00**0**0**0**, d=3

001**1**1, d=2 ←

1**1**0**0**1, d=2 ←

1**1**1**10**, d=3

Hamming distances are **not unique**, which means receiver cannot decide received Rx codeword must be corrected to which available codeword one. There is no correction but detection only in this case. Receiver may ask the transmitter to retransmit the data

Error Correction Example 4

- Data to send: 01; 3rd and 4th bit transmitted in error

Data	*Codeword*
00	00000
01	00111
10	11001
11	11110

Tx Data:01, Tx codeword:00111 $\xrightarrow{\textbf{3rd\&4}^{\textbf{th}}\textbf{ bits in error}}$ Rx codeword:00**00**1, Corrected: 00000, Rx data: 00 (X)

0000**0**, d=1

00**11**1, d=2

11001, d=2

11110, d=5

Hamming distance is the least and unique (d=1). Therefore, the corrected codeword is 00000, which means 00 in the mapping. But the original transmitted data by the transmitter is 01. So, in this case FEC does not work!

Performance of Error Detection/Correction

- Aim to detect/correct as many errors as possible
- But error detection/correction require extra bits to be sent
- ***k*** bits of useful data; ***n*** bits transmitted; efficiency=η= $\frac{k}{n}$
- **Tradeoff:** for a given amount of data, ***k*** bits
  - Increase n, more errors detected/corrected (GOOD)
  - Increase n, lower efficiency of transmission (BAD)

For the examples given before k =2, n=5, η= $\frac{k}{n}$= $\frac{2}{5} = 0.4$ =40%.

Efficiency is not effected from the detection and/or correction cases.