# ITM 1010
## Computer and Communication Technologies

## Lecture #16

**Part II** **Introduction to Communication Technologies:**

## Multiple Access Control; Data Compression

# CSMA/CD (Collision Detection)

- Unlike CSMA, CSMA/CD aborts a transmission immediately if collision is detected
  - then start a backoff algorithm
  - the first station to detect the collision sends out a special jamming pulse to alert all stations

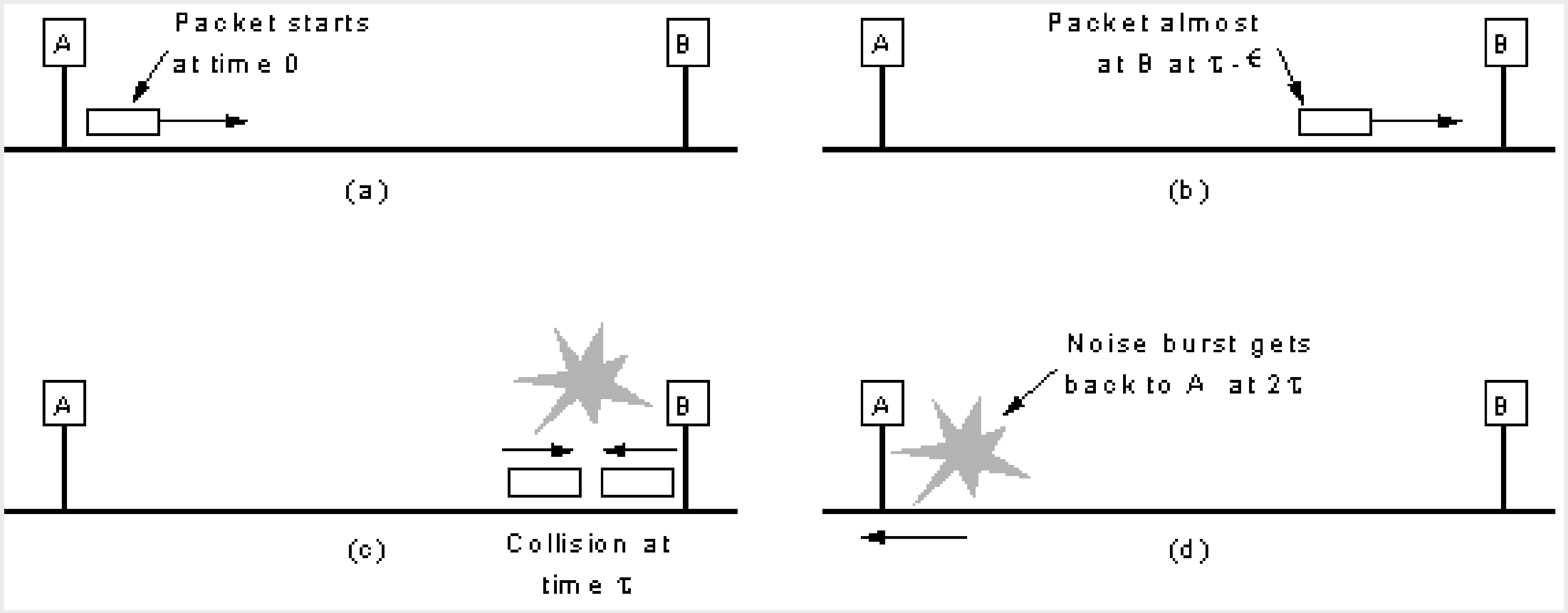
- binary exponential backoff algorithm
  - each station picks a random number out of N and then waits for that random number of slot time
  - if N is large, then long average waiting time (N/2 slots)
  - if N is small, in case the number of re-transmitting stations is large, then the chance that data will collide again will be very high

# CSMA/CD

- Binary exponential backoff
  - start with a small N  (N=2)
    - station choose between 0 (no delay), or 1 (wait 1 slot)
  - if collide again, double N
  - after 10 collision, N is fixed at the maximum value at 1024, the chance that stations will collide again is very small
  - after 16 collision, the algorithm gives up and sends alert message to sys admin
    - the chance that data collides 16 times is so small that more likely there is something wrong with the system

# CSMA

- Collision problem - even with collision avoidance, collision can still happen for two reasons:
    1. propagation delay in cable
    2. stations that avoid sending data during the busy period will transmit immediately after the channel is free



## CSMA/CD and Ethernet

- Collision detection
  - A and B are two stations located at the two ends

    $\tau$ is the propagation time for signal to travel from A to B
  - it takes time $\tau$ for station A to seize the channel
    - because after time $\tau$, the signal will reach all stations on the LAN, and stop other stations in sending data
  - but it may take as long as time $2\tau$ <u>for A to know</u> it has seized the channel
  - if A has completed the transmission before $2\tau$, it will not be able to detect the collision if it happens
- 10Mb/s Ethernet (IEEE802.3)
    - maximum length of 10Mb/s LAN is specified as 2.5km

      $\Rightarrow \tau = 2.5\text{km}/(2\text{x}10^8\text{m/s}) = 12.5\mu\text{s} \Rightarrow 2\tau = 25\mu\text{s}$
    - therefore $2\tau$ is equivalent to 250 bits time
    - minimum packet size must be greater than 250 bits, The IEEE803.2 defines minimum packet to be 512bits (64 bytes)

# Ethernet

- What happen if we extend the network beyond 2.5km?
  - If we extend the maximum distance, $\tau$ will be increased too
    - problem 1: chance of collision increase with t
    - problem 2: we need to increase the min frame size
  - because of these problems (particularly the first), CSMA/CD (and CSMA as well) cannot be used for WAN

- What happen if we increase the speed of the network (100Mb/s ethernet)?
  - If we increase the speed by a factor of 10, the max propagation time $\tau$ remains unchanged, but packet transmission is completed 10 times faster i.e. may complete the transmission before $2\tau$ period
  - Therefore either the minimum frame size must be increased by the same factor (10 times) or the transmission distance must be reduced (IEEE802.3u specifies a max link of only 100m for 100Mb/s ethernet).

## Summary – Multiple Access Control

- Aloha Network
- Slotted Aloha – reduce the vulnerable period of frame collision
- Listen before Talking – If the channel is busy, not to send data.
  - 1-persistent CSMA
  - 0-persistent CSMA
  - P-persistent CSMA
  - CSMA/CD (Ethernet standard)
    - Maximum distance of Ethernet link is determined by frame size and bitrate (minimum packet transmission time > $2\tau$)

## Data Compression

How to minimize the number of bits (binary digits) per bit of information?

## Lossless and Lossy Data Compression

- Bandwidth is a limited resource
- Data compression algorithms seek to reduce the number of binary digits needed to transmit a message and improve efficiency in usage of bandwidth (and data storage capacity)
  - Lossless compression: no information is lost in the compression.  Examples include run length encoding, Huffman, and Lempel-Ziv Welch (LZW) codes.
  - Lossy compression:  some information in the original message is lost. Lossy compression can provide much greater compression ratios and is used for images and audio information.   Examples include the JPEG standards for compressing still images, and MPEG standards for compressing moving pictures and sound.

## Run Length Encoding

- ❑ Run-length Encoding (RLE) codes a sequence of symbols by replacing a consecutive run of the same symbol with the number of repetitions

- ❑ It is useful when large runs of 0s are expected eg. in scanned documents, faxes, weather maps etc., where the probability of a 0 is close to unity.

- ❑ Example - RLE of a binary sequence:

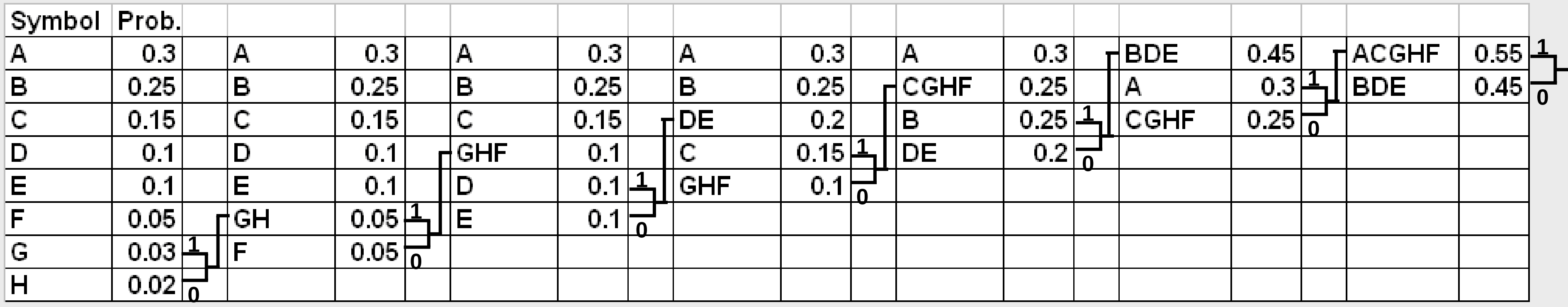
Input : 0, 5, 0, 0, 0, 30, 61, 127, 0, 0, 0, 0, 0, 0, 0, 9
16 symbols, 7 bits/symbol = 112 bits

zero-RLE: *1, 5, *3, 30, 61, 127, *7, 9
8 symbols, **8 bits**/symbol = 64 bits
(8 bits instead of 7 bits is needed per symbol in the RLE data-stream; the extra bit distinguishes data from the length of a string of zeros)

## Huffman Codes

- Huffman Codes rely on knowing the probability of occurrence of each symbol to produce a code that approaches the entropy of the source
  - A variable number of binary digits is used for each symbol, and fewer binary digits are assigned for symbols that have a high probability of being sent
  - Decoding of a Huffman code rely on the **no-prefix property** for each symbol : the code for any symbol never occurs in the starting sequence of another symbol.
  - A look-up table is usually needed for decoding
- Huffman codes are a form of entropy encoding

# Huffman Tree



| Symbol | Prob. | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| A | 0.3 | A | 0.3 | A | 0.3 | A | 0.3 | A | 0.3 | BDE | 0.45 | ACGHF | 0.55 |
| B | 0.25 | B | 0.25 | B | 0.25 | B | 0.25 | CGHF | 0.25 | A | 0.3 | BDE | 0.45 |
| C | 0.15 | C | 0.15 | C | 0.15 | DE | 0.2 | B | 0.25 | CGHF | 0.25 | | |
| D | 0.1 | D | 0.1 | GHF | 0.1 | C | 0.15 | DE | 0.2 | | | | |
| E | 0.1 | E | 0.1 | D | 0.1 | GHF | 0.1 | | | | | | |
| F | 0.05 | GH | 0.05 | E | 0.1 | | | | | | | | |
| G | 0.03 | F | 0.05 | | | | | | | | | | |
| H | 0.02 | | | | | | | | | | | | |

- Huffman code may be generated using the following algorithm:
  - Write list of all possible symbol in order of their probabilities
  - Combine the two lowest elements in list and re-list in order of probabilities
  - Repeat above steps until all messages have been combined
  - Assign a 1 and 0 to each of the two possible branches in each “split” working backwards from the end
  - Huffman code for each symbol is the sequence of ones and zeros in tracing back from the end of the table to the single symbol entry in the table
- Code table produced from above example is: A=11, B= 01, C= 101, D= 001, E=000, F=1000, G= 10011, H=10010

## Efficiency of Huffman codes

- In the previous example, we can calculate the minimum number of binary digits needed from the entropy of the source:

$$H = -\sum_{j=1}^{N} p_j \log_2 p_j = 2.57676 \text{ Bits per message}$$

- Actual average number of binary digits needed per symbol achieved with the Huffman code in the example is

0.3x2 + 0.25x2 + 0.15x3 + 0.1x3 + 0.1x3 + 0.05x4 + 0.03x5 + 0.01x5 = 2.55

- Huffman codes achieve optimum compression ratios based only on individual symbol statistics. It does not use higher order redundancy in the data (For example, with English messages Huffman code uses probability of individual letters but not the probability of letter groups or words) For English text a Huffman code can reduce the number of bits needed by about 43%.

## Lempel-Ziv Welch (LZW) Code

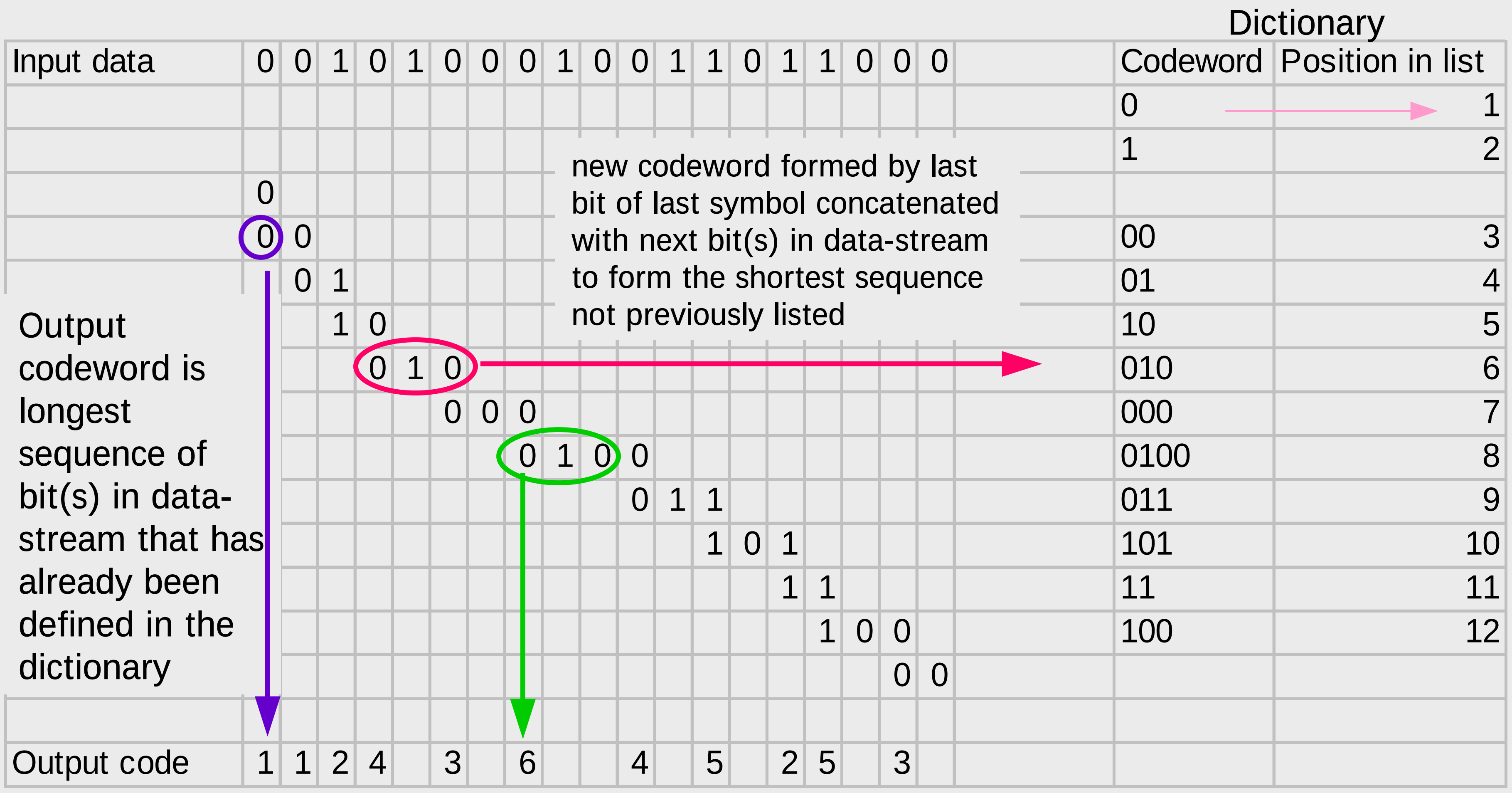
- Drawbacks of Huffman codes:
  - Need *a priori* knowledge of source statistics
  - Does not make use of high order redundancy in the data (e.g. words or phrases being repeated).
- Lempel and Ziv proposed a more efficient algorithm than the Huffman code in 1976. Algorithm was later improved by Welch and became known as LZW compression
- LZW compression is the compression of a file into a smaller file using a table-based lookup algorithm
- LZW codes are now widely used (eg. UNIX compress command, v42bis modem standard) . Typical English text may be compressed by 55% using LZW code (better than the 43% compaction from Huffman codes because LZW uses repetitions of words and phrases in the data)
- LZW coding does not need *a priori* knowledge of the source statistics

## LZW Algorithm

- Basic approach:
  1. A particular LZW compression algorithm takes each input sequence of bits of a given length (for example, 12 bits) and creates an entry in a table for that particular bit pattern, consisting of the pattern itself and a shorter code.
  2. As input is read, any pattern that has been read before results in the substitution of the shorter code, effectively compressing the total amount of input to something smaller.
  3. LZW algorithm does include the look-up table of codes as part of the compressed file. The decoding program that uncompresses the file is able to build the table itself by using the algorithm as it processes the encoded input.

- LZW exploits the redundancy/repetition in groups of symbols (e.g. frequently used phrases in English) in the data to achieve larger compression ratio than is possible by exploiting just the statistics of individual symbols (Huffman coding)

# LZW Encoding Example



| | | | | | | | | | | | | | | | | | | | | Dictionary | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Input data | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 1 | 0 | 1 | 1 | 0 | 0 | 0 | Codeword | Position in list |
| | | | | | | | | | | | | | | | | | | | | 0 | 1 |
| | | | | | | | | | | | | | | | | | | | | 1 | 2 |
| | 0 | | | | | | | | | | | | | | | | | | | | |
| | 0 | 0 | | | | | | | | | | | | | | | | | | 00 | 3 |
| | | 0 | 1 | | | | | | | | | | | | | | | | | 01 | 4 |
| | | | 1 | 0 | | | | | | | | | | | | | | | | 10 | 5 |
| | | | | 0 | 1 | 0 | | | | | | | | | | | | | | 010 | 6 |
| | | | | | | 0 | 0 | 0 | | | | | | | | | | | | 000 | 7 |
| | | | | | | | | 0 | 1 | 0 | 0 | | | | | | | | | 0100 | 8 |
| | | | | | | | | | | | 0 | 1 | 1 | | | | | | | 011 | 9 |
| | | | | | | | | | | | | | 1 | 0 | 1 | | | | | 101 | 10 |
| | | | | | | | | | | | | | | | 1 | 1 | | | | 11 | 11 |
| | | | | | | | | | | | | | | | | 1 | 0 | 0 | | 100 | 12 |
| | | | | | | | | | | | | | | | | | | 0 | 0 | | |
| | | | | | | | | | | | | | | | | | | | | | |
| Output code | 1 | 1 | 2 | 4 | | 3 | | 6 | | | 4 | | 5 | | 2 | 5 | | 3 | | | |

new codeword formed by last bit of last symbol concatenated with next bit(s) in data-stream to form the shortest sequence not previously listed

Output codeword is longest sequence of bit(s) in data-stream that has already been defined in the dictionary

# LZW Decoding

| | | | | | | | | | | | | | | | | | | | | | Codeword | Position in list |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | | | | | | | | | | | | | | | 0 | 1 |
| | | | | | | | | | | | | | | | | | | | | | 1 | 2 |
| Received data | 1 | 1 | 2 | 4 | | 3 | | 6 | | | 4 | | 5 | | 2 | 5 | | 3 | | | | |
| | 0 | | | | | | | | | | | | | | | | | | | | | |
| | 0 | 0 | | | | | | | | | | | | | | | | | | | 00 | 3 |
| | | 0 | 1 | | | | | | | | | | | | | | | | | | 01 | 4 |
| | | | 1 | 0 | 1 | | | | | | | | | | | | | | | | 10 | 5 |
| | | | | 0 | 1 | 0 | 0 | | | | | | | | | | | | | | 010 | 6 |
| | | | | | | 0 | 0 | 0 | 1 | 0 | | | | | | | | | | | 000 | 7 |
| | | | | | | | | 0 | 1 | 0 | 0 | 1 | | | | | | | | | 0100 | 8 |
| | | | | | | | | | | | 0 | 1 | 1 | 0 | | | | | | | 011 | 9 |
| | | | | | | | | | | | | | 1 | 0 | 1 | | | | | | 101 | 10 |
| | | | | | | | | | | | | | | | 1 | 1 | 0 | | | | 11 | 11 |
| | | | | | | | | | | | | | | | | 1 | 0 | 0 | 0 | | 100 | 12 |
| | | | | | | | | | | | | | | | | | | | | | | |
| Decoded output | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 1 | 0 | 1 | 1 | 0 | 0 | 0 | | | |

## Summary

- Important lossless compression algorithms include: RLE, Huffman Codes and LZW code
- Lossy algorithms can provide high compression ratios but there is a trade-off between image fidelity and compression ratio
- Compression of images and sound can be lossy because of limitations in the human perception of sound and images