Error Control Data Compression

End-to-End Data

Data Compression

Introduction to Data Compression

- Data Compression is the process of encoding information in a way that takes up less space than the original, raw data, usually with the goal that all or most of that information can be unencoded.
- Data compression reduces the size of data frames to be transmitted over a network link. Reducing the size of a frame reduces the time required to transmit the frame across the network

Introduction to Data Compression

Data compression is the function of presentation layer in OSI reference model. Compression is often used to maximize the use of bandwidth across a network or to optimize disk space when saving data

- There are two methods of compression
- – lossy and lossless.
- Lossy reduces file size by permanently removing some of the original data.
- Lossless reduces file size by removing unnecessary metadata

Why Data Compression?

- Data storage and transmission cost money. This cost increases with the amount of data available.
- This cost can be reduced by processing the data so that it takes less memory and less
  transmission time.
- Data transmission is faster by using better transmission media or by compressing the data.
- Data compression algorithms reduce the size of a given data without affecting its content.

  Examples
- . Huffman coding
- . Run-Length coding
- . Lempel-Ziv coding

Need of Compression

- **Why Data Compression?**
  - There are two practical motivations for compression:
    - Make optimal use of limited storage space (Reduction of storage requirements)
    - Save time and help to optimize resources
      - If compression and decompression are done in I/O processor, less time is required to move data to or from storage subsystem, freeing I/O bus for other work
      - In sending data over communication line: less time to transmit and less storage to host

Lossless Compression

- • Lossless compression compresses the data in such a way that when data is decompressed it is exactly the same as it was before compression i.e. there is no loss of data.
- A lossless compression is used to compress file data such as executable code, text files, and numeric data, because programs that process such file data cannot tolerate mistakes in the data.
- • Lossless compression will typically not compress file as much as lossy compression techniques and may take more processing power to accomplish the compression.

Lossless Compression Algorithms

1. Run length encoding

This method replaces the consecutive occurrences of a given symbol with only one copy of the symbol along with a count of how many times that symbol occurs. Hence the names 'run length'.

• For example, the string AAABBCDDDD would be encoded as 3A2BIC4D

2. Differential pulse code modulation

In this method first a reference symbol is placed. Then for each symbol in the data, we place the difference between that symbol and the reference symbol used.

• For example, using symbol A as reference symbol, the string AAABBC DDDD would be encoded as AOOOl123333, since A is the same as reference symbol, B has a difference of 1 from the reference symbol and so on

Lossless Compression Algorithms

3. Dictionary based encoding

One of the best known dictionary based encoding algorithms is Lempel-Ziv (LZ) compression algorithm.

• This method is also known as substitution coder. •

In this method, a dictionary (table) of variable length strings (common phrases) is built.

• This dictionary contains almost every string that is expected to occur in data.

• When any of these strings occur in the data, then they are replaced with the corresponding index to the dictionary

Lossless Compression Algorithms

- In this method, instead of working with individual characters in text data, we treat each word as a string and output the index in the dictionary for that word.
- • For example, let us say that the word "compression" has the index 4978 in one particular dictionary; it is the 4978th word is usr/share/dict/words. To compress a body of text, each time the string "compression" appears, it would be replaced by 4978

Lossy Compression

- Lossy compression is the one that does not promise that the data received is exactly the same as data send i.e. the data may be lost.
- This is because a lossy algorithm removes information that it cannot later restore.
- Lossy algorithms are used to compress still images, video and audio.
- Lossy algorithms typically achieve much better compression ratios than the lossless algorithms

Lossy vs. Lossless

- Lossless compression: when data is decompressed, it is restored in the exact same state or without loss of information
  - Huffman Encoding
  - LZ77
  - PNG, BMP, WAV

Lossy Compression: upon decompression, some elements are lost or unable to be restored, some loss of fidelity.

- MP3
- JPG/JPEG

Huffman Encoding

A quick review of Huffman Encoding:

- Variable-length prefix encoding - the more frequent characters take up less space

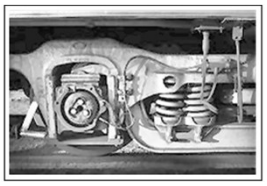
- Build a tree with the most frequent characters at the top with each child receiving an additional 1 or 0 in their encoding

JPEG

The JPEG algorithm is a lossy compression algorithm designed for images.

It uses a number of mathematical principles to shrink file sizes while trying to maintain the overall visual integrity of an image.

We'll use this image as a quick example to demonstrate the JPEG algorithm.



Lossless Compression & Huffman Encoding

- Now, what if we had a way to not just encode single characters, but entire patterns and strings?

- This could potentially result in much smaller or more compressed file sizes than what we were able to achieve before with simple Huffman Encoding.

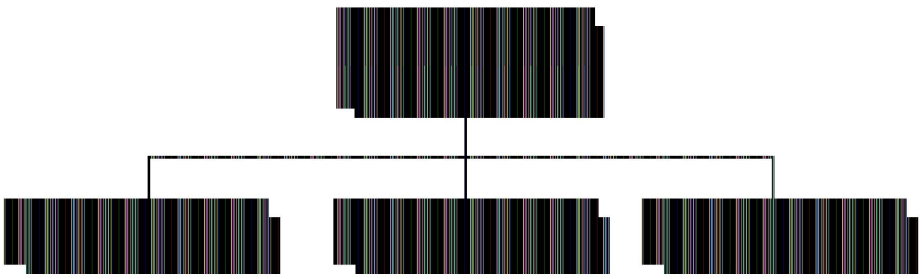
- This idea is the topic of the Lempel-Ziv Paper in 1977, which resulted in the LZ77 algorithm.

Error Detection and Correction

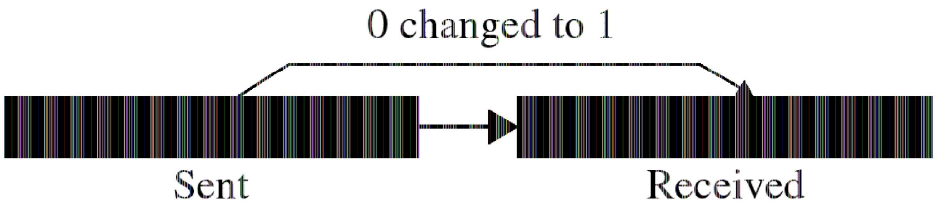
Basic concepts

- ★ Networks must be able to transfer data from one device to another with complete accuracy.
- ★ Data can be corrupted during transmission.
- ★ For reliable communication, errors must be detected and corrected.
- ★ **Error detection and correction** are implemented either at the **data link layer** or the **transport layer** of the OSI model.

Types of Errors



Single-bit error

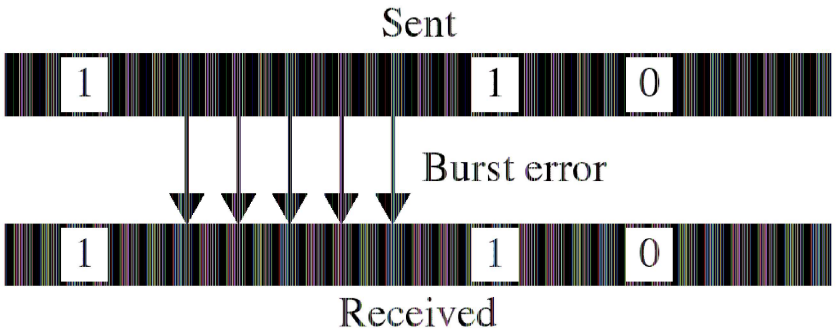


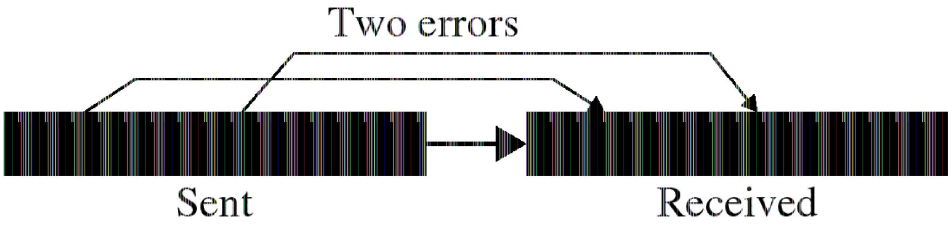
Single bit errors are the **least likely** type of errors in serial data transmission because the noise must have a very short duration which is very rare. However this kind of errors can happen in parallel transmission.

Example:

- ★If data is sent at 1Mbps then each bit lasts only 1/1,000,000 sec. or 1 μs.
- ★For a single-bit error to occur, the noise must have a duration of only 1 μs, which is very rare.

Burst error





The term **burst error** means that two or more bits in the data unit have changed from 1 to 0 or from 0 to 1.

Burst errors does not necessarily **mean that the errors occur in consecutive bits**, the length of the burst is measured from the first corrupted bit to the last corrupted bit. Some bits in between may not have been corrupted.

★**Burst error is most likely to happen in serial transmission** since the duration of noise is normally longer than the duration of a bit.

★The number of bits affected depends on the data rate and duration of noise.

Example:

➔If data is sent at rate = 1Kbps then a noise of 1/100 sec can affect 10 bits.(1/100*1000)

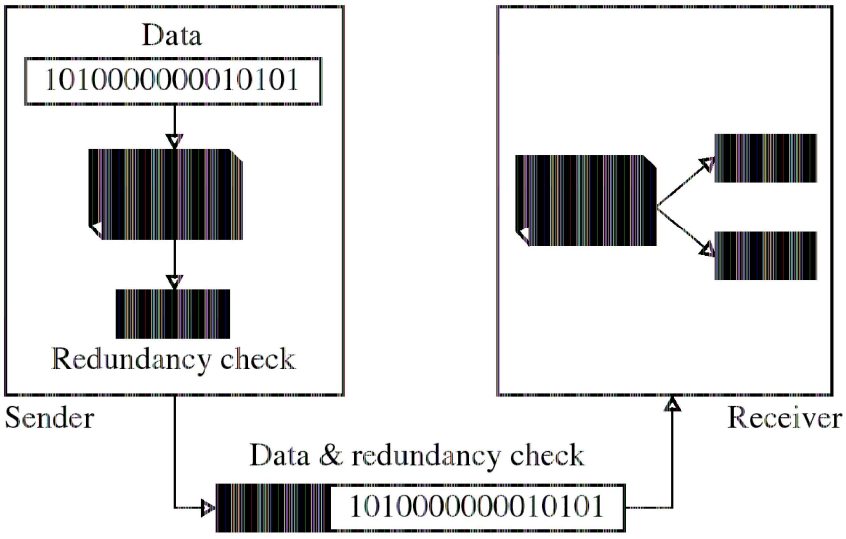
➔If same data is sent at rate = 1Mbps then a noise of 1/100 sec can affect 10,000 bits.($1/100*10^6$)

Error Detection

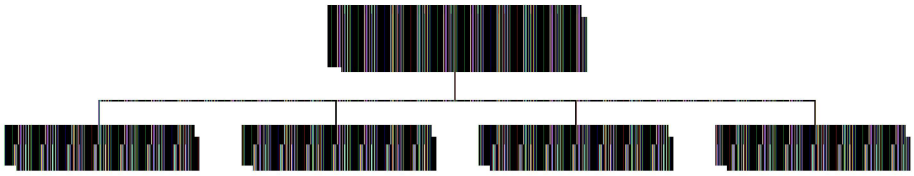
Error detection means to decide whether the received data is correct or not without having a copy of the original message.

Error detection **uses the concept of redundancy**, **which means** adding extra bits for detecting errors at the destination.

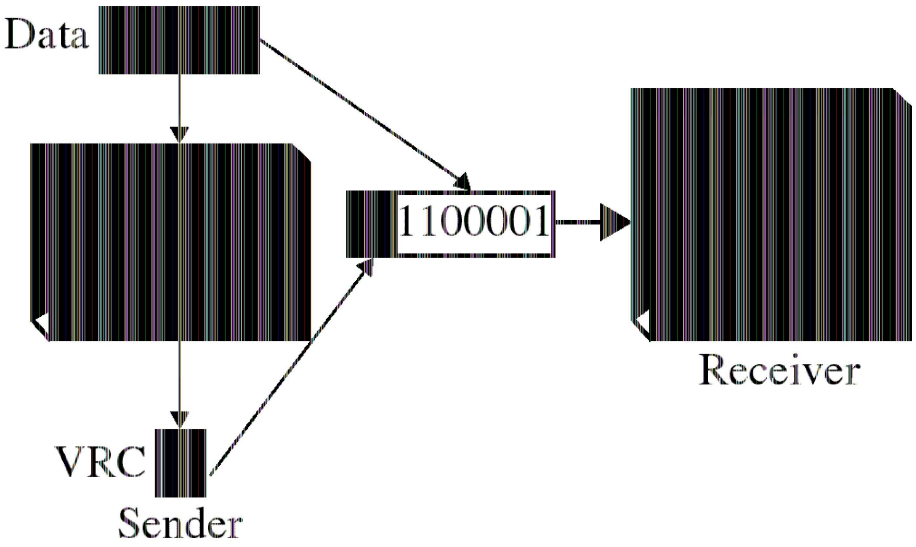
Redundancy



Four types of redundancy checks are used in data communications



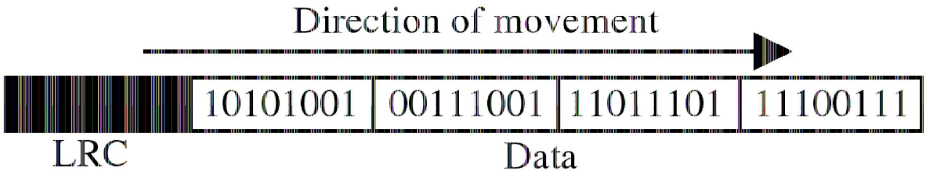
Vertical Redundancy Check
VRC



Performance

- It can detect single bit error
- It can detect burst errors only if the total number of errors is odd.

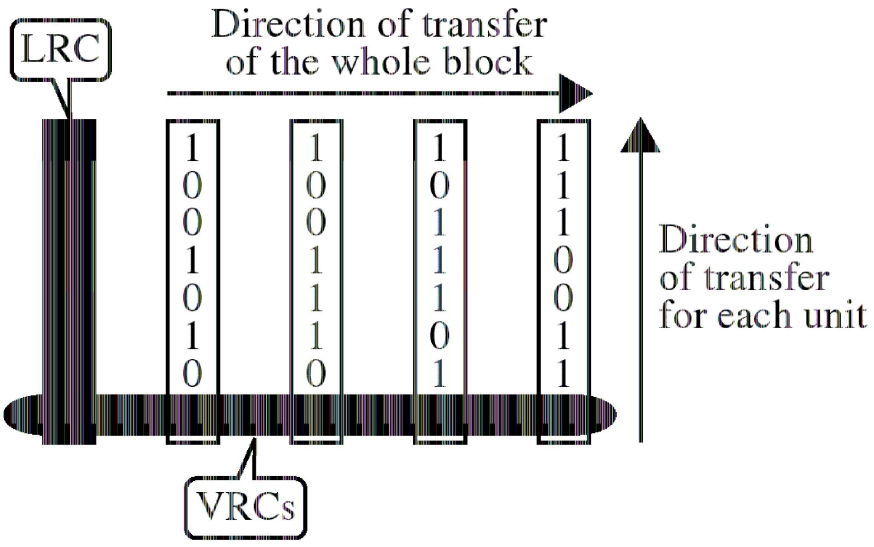
Longitudinal Redundancy Check
LRC



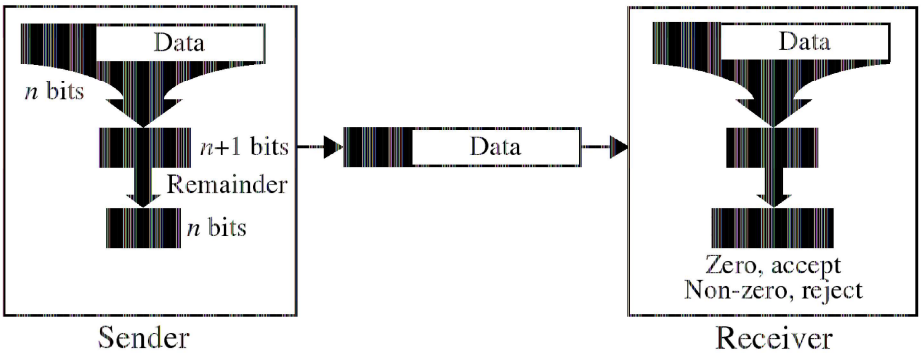
Performance

- LCR increases the likelihood of detecting burst errors.
- If two bits in one data units are damaged and two bits in exactly the same positions in another data unit are also damaged, the LRC checker will not detect an error.

VRC and LRC



Cyclic Redundancy Check
CRC



Cyclic Redundancy Check

- Given a k-bit frame or message, the transmitter generates an n-bit sequence, known as a *frame check sequence (FCS)*, so that the resulting frame, consisting of (k+n) bits, is exactly divisible by some predetermined number.
- The receiver then divides the incoming frame by the same number and, if there is no remainder, assumes that there was no error.

At the sender

- The unit is divided into *k* sections, each of *n* bits.
- All sections are added together using one's complement to get the sum.
- The sum is complemented and becomes the checksum.
- The checksum is sent with the data

Error Correction

It can be handled in two ways:

1) receiver can have the sender retransmit the entire data unit.
2) The receiver can use an error-correcting code, which automatically corrects certain errors.

Single-bit error correction

To correct an error, the receiver reverses the value of the altered bit. To do so, it must know which bit is in error.

Number of redundancy bits needed

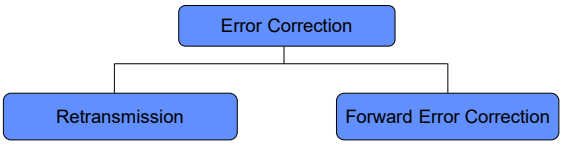
- Let data bits = m
- Redundancy bits = r

$\therefore$ Total message sent = $m+r$

The value of r must satisfy the following relation:

$$2^r \geq m+r+1$$

Error Correction



Error Correction by Retransmission

When an error is discovered, the receiver can have the sender retransmit the entire data unit (related to flow and error control protocols).

Forward Error Correction (FEC)

- In this method, a receiver can use an error-correcting code, which automatically corrects certain errors.

- Error correcting codes are more sophisticated than error detection codes and require more redundancy bits.

- The simplest case is single-bit errors. It can be detected by the addition of a redundant (parity) bit. This additional bit can detect single-bit error because it must distinguish between only two conditions: error or no error. A bit has two states (0 and 1) and these are sufficient for this level of detection.

- For single-bit error correction, two states are enough to detect an error but not to correct it. To correct the error, the receiver simply reverses the value of the altered bit and to do so, it must know which bit is in error – the secret of error correction.

- To correct a single-bit error in an ASCII character, the error correction code must determine which of the 7 bits has changed, so we have eight states : no error, error in position 1, error in position 2 , …., error in position 7

Burst Error Correction

Although the hamming code cannot correct a burst error directly, it is possible to rearrange the data and then apply the code. Instead of sending all the bits in a data unit together, we can organize ***N*** units in a column and then send the first bit of each, followed by the second bit of each, and so on. In this way, if a burst error of ***M*** bits occurs (***M*** < ***N***), then the error does not corrupt ***M*** bits of one single unit; It corrupts only 1 bit of a unit. With the Hamming scheme, we can then correct the corrupted bit in each unit.

Burst Error Correction example

We need to send six data units where each unit is a character with Hamming redundant bits. We organize the bits in columns and rows. We send the first column, then the second column, and so on. Five consecutive bits are corrupted during the actual transmission. When these bits arrive at the destination and are reorganized into data units, each corrupted bit belongs to one unit and is automatically corrected. The trick here is to let the burst error corrupt only 1 bit of each unit.