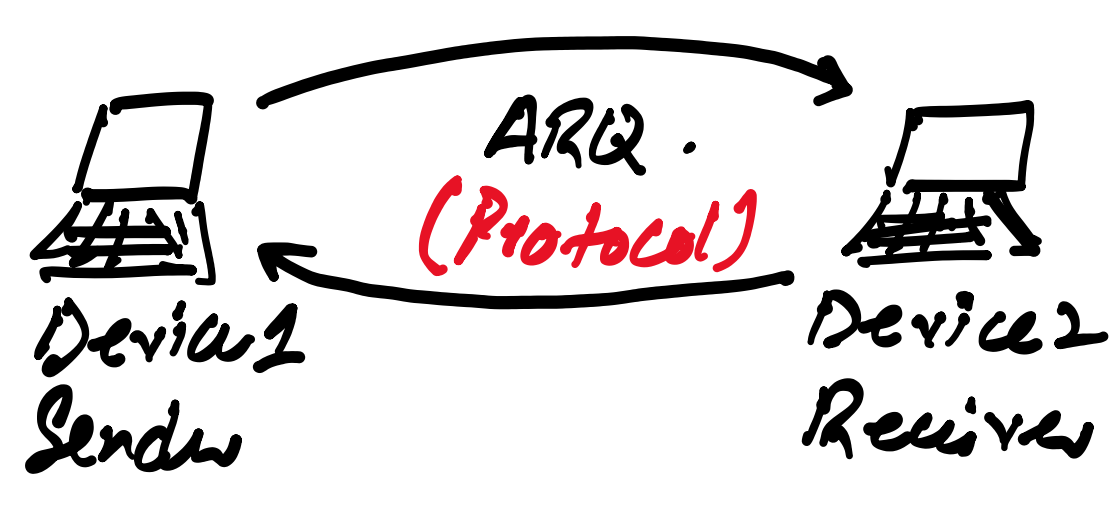
ARQ, CHECK DIGIT, PARITY, CHECKSUM. Errors in data transfer.

Saturday, 31 July 2021 5:31 PM

Key terms:

- ARQ ✓
- Parity ✓
- Check Digit
- Check Sum ✓



- Parity
- Checksum

Protocols: Rules set forth between device before actual data transfer occurs.

Such as: HTTP, www, FTP, ARQ.

Automatic Repeat ReQuest (ARQ): It is a protocol (method) used to check whether data transferred is correct or not.

Acknowledgement:

Receiver of the message (data) sends an acknowledgement to the sender upon receiving the msg.

Ack. is sent for both correct and incorrect receipt of data.

If the data received is "corrupted" then sender when receives ack. re-sends the data/msg.

If the data received is "correct data" then next data due is sent.

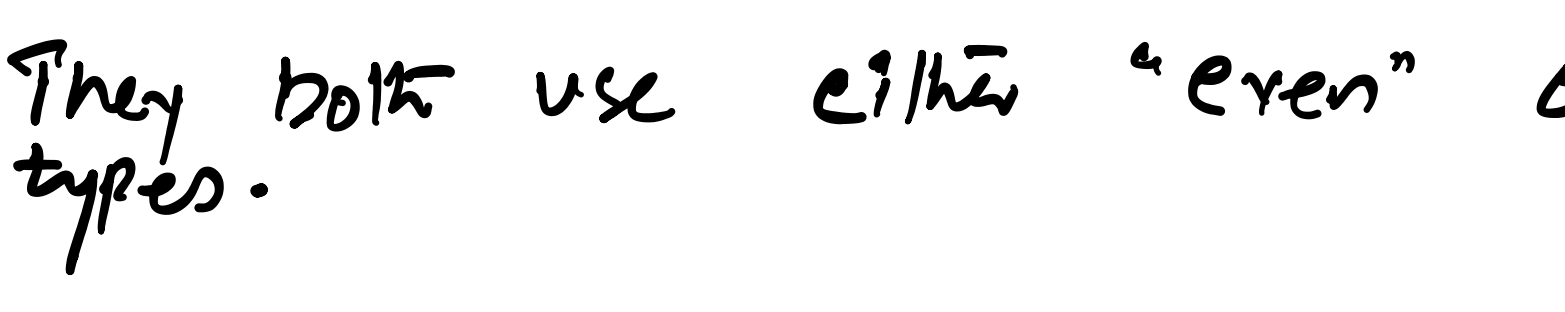
Time out: When two devices start data transfer, they set an agreed (pre-defined) time for the receipt of ack.

If ack. is not received for that particular pre-defined time then it is said to be "TIMEOUT" occured. So, sender re-sends the data.

This may go-on until an ack. arrives at sender's end.

Parity Check: It is a method used to identify data corruption upon receival of the data/msg.

It is mainly used:
- with text data (ASCII)
- in close proximity.



7bits are available to store ASCII Codes. $2^7 = 128$

There are two parity mechanisms at work:
1. Byte parity
2. Block parity.

They both use either "even" or "odd" parity types.

Example: Parity is the count of "1s"

Char	Den	(ASCII) Bin	
A	65	01000001	Even parity.
B	66	01000010	
C	67	11000011	

Char	Den	(ASCII) Bin	
A	65	11000001	Odd parity
B	66	11000010	
C	67	01000011	

In byte parity a single byte is transfered at a time.

There are few weaknesses in byte parity. For example:
- when two bits are swapped
- when two bits are changed (gain, drop) 0→1 1→0

Char	Den	(ASCII) Bin	positions exchanged
A	65	01000001 →	01000100
B	66	01000010	
C	67	11000011 →	11001111

Corrupted; but corruption is unidentified in even byte parity mechanism.

Block parity: It is a way out to cover weaknesses in byte parity. In it instead of a single byte at a time, several bytes, as a block, are sent.

In addition to a parity bit in every byte, for block parity a parity for column is also found and added under every column.

	Block
A	0 1 0 0 0 0 0 1
B	0 1 0 0 0 0 1 0
C	1 1 0 0 0 0 1 1
Parity Byte →	1 1 0 0 0 0 0 0

Even parity Block.

Example of block parity:

Sender's End:

0	1	0	0	0	0	0	1
0	1	0	0	0	0	1	0
1	1	0	0	0	0	1	1
1	1	0	0	0	0	0	0

Even parity

Receiver's End:

0	1	0	0	0	0	0	1
0	1	0	0	0	0	1	0
1	1	0	0	0	0	1	1
1	1	0	0	0	0	0	0

Checking at receiver's end:
- Check all columns as per decided parity
- Check all rows as per decided parity.
- Mark the column and row where there are any discrepancies.
- Intersection of the column and row where there are errors is marked.
- The bit at the intersection will be flipped.
- This action will remove the error from received block of msg at receiver's end without having the msg block called up again from the sender.

Benefits of block parity:

1. Time saving:
   Data is not called again when there is an error; rather data is made correct at receiver's end.
2. Bandwidth saving:
   Since the data is only sent once by the sender, the bandwidth is saved.

Checksum: It is another way for checking of data if it has been corrupted during the data transmission; at receiver's end.

The data is sent in blocks along with an additional value, the checksum.

Checksum works with all types of data.

Example of checksum:

ABC

11000110 01000011 01000010 01000001

Checksum 67 + 66 + 65 = 198 Checksum

~ SENDER'S END ~

↓ Data Transmission (1 bit got changed (gained; 0-1))

11000110 01000011 01000110 01000001

(198) Checksum from sender's end.

67 + 70 + 65 ⇒ (202) Checksum at receiver's

~ RECIEVER'S END ~

Two checksums don't match; it means that the data received is corrupted during data transmission.

Exception in checksum for values greater than 255.

If the checksum found is greater than 255 then following algorithm is used to find "one byte fitting" checksum.

Checksum found is: 465 ⟶ x

Divide x by 256 : 465/256 = 1.81641 → y

Round down y to the nearest whole number: RD(1.81641) = 1 → z

Multiply z with 256: 1 × 256 = 256 → M

Calculate the difference: x − M = Checksum

465 − 256 = 209