Unit-3.1 Error Correction and Detection

Prepared By:
Prof. Vishal A. Polara
Assistant Professor
Information Technology Department
Birla Vishvakarma Mahavidyalaya Engineering College

Outline

- Types of Errors
- Block coding
- Linear block codes
- Cyclic codes
- Checksum

1. Types of errors

- There are two types of errors
  - Single bit errors : Only one bit in data unit has changed
  - multi bit errors: It is also known as burst error where more than two bit in data unit has changed
- Here 0 changes to 1 and 1 changes to 0.
- Single bit errors are normally seen least in serial data transmission.
- Burst error is more likely to occur than a single bit error. It occurs due to noise because duration of noise is longer than the duration of 1 bit. So noise affect more numbers of data.

Figure ***Single-bit error***

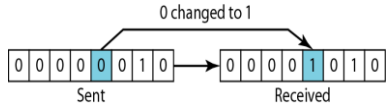
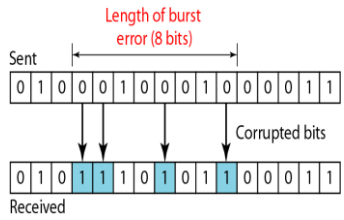


Figure ***Burst error of length 8***



Error correction and detection using Redundancy

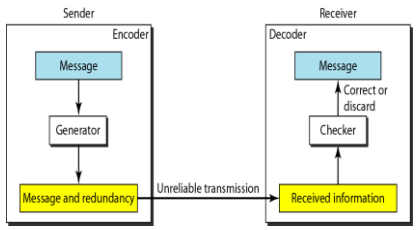
- To detect or correct errors we need to send some extra bits with our data.
- These redundant bits are added by the sender and removed by the receiver. It allows receiver to detect or correct corrupted bits.
- Error detection means we have to change if bit changed or not.
- While in case of error correction we need to know the exact number of bits that are corrupted and their location in the message.
- If we need to correct one single error in an 8-bit data unit we need to consider eight possible error locations, if we need to correct two errors in a data unit of the same size we need to consider 28 possibilities.
- Redundancy achieve using various coding schemes. The sender adds the redundant bits through a process that creates relationship between the redundant bits and the actual data bits.

- Redundancy achieve using various coding schemes. The sender adds the redundant bits through a process that creates relationship between the redundant bits and the actual data bits.
- The receiver checks the relationships between the two sets of bits to detect or correct the errors.
- There are two coding schemes available:
- Block coding
- Convolution coding

Method of correction

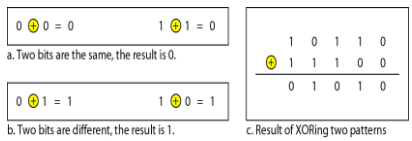
- There are two main methods of error correction.
- Forward error correction
- Retransmission
- Forward error correction is the process in which the receiver tries to guess the message by using redundant bits. If the number of errors is small
- Retransmission is a technique in which the receiver detects the occurrence of an error and asks the sender to resend the message.

Figure *The structure of encoder and decoder*



In modulo-N arithmetic, we use only the integers in the range 0 to N −1, inclusive.

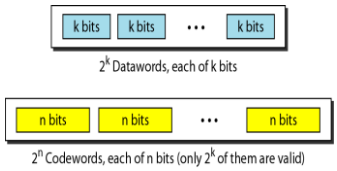
Figure *XORing of two single bits or two words*



2. Block Coding

- In block coding, we divide our message into blocks, each of k bits, called datawords. We add r redundant bits to each block to make the length n = k + r. The resulting n-bit blocks are called codewords.
- With k bits we can create a combination of 2^k data words and with n bits we can create a combination of 2^k codewords.
- In block coding same data word is always encoded using same code words that mean 2^n – 2^k codewords are not used. It is known as invalid or illegal codewords.
- Error Detection
- Error Correction
- Hamming Distance
- Minimum Hamming Distance

Figure *Datawords and codewords in block coding*



13 Prof. Vishal A. Polara

Example

The 4B/5B block coding discussed in previous unit is a good example of this type of coding. In this coding scheme, k = 4 and n = 5. As we saw, we have $2^k = 16$ datawords and $2^n = 32$ codewords. We saw that 16 out of 32 codewords are used for message transfer and the rest are either used for other purposes or unused.

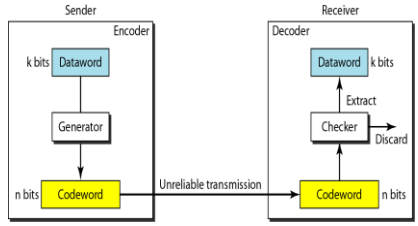
14 Prof. Vishal A. Polara

Error Detection

- In Block coding error detection is possible if following two condition met.
  - The receiver has a list of valid codewords
  - The original codeword has changed to an invalid one
- The sender creates codewords out of datawords by using a generator that applies the rules and procedures of encoding.
- Each codeword sent to the receiver may change during transmission.
- If it is same than accepted else received codeword is not valid, it is discarded.
- If it is corrupted but still matched with a valid codeword the errors remain undetected.

15 Prof. Vishal A. Polara

Figure *Process of error detection in block coding*



16 Prof. Vishal A. Polara

Example

Let us assume that k = 2 and n = 3. Table shows the list of datawords and codewords. Later, we will see how to derive a codeword from a dataword.

Assume the sender encodes the dataword 01 as 011 and sends it to the receiver. Consider the following cases:

1. ***The receiver receives 011. It is a valid codeword. The receiver extracts the dataword 01 from it.***

17 Prof. Vishal A. Polara

Example (continued)

2. ***The codeword is corrupted during transmission, and 111 is received. This is not a valid codeword and is discarded.***

3. ***The codeword is corrupted during transmission, and 000 is received. This is a valid codeword. The receiver incorrectly extracts the dataword 00. Two corrupted bits have made the error undetectable.***

18 Prof. Vishal A. Polara

Table *A code for error detection*

Datawords	Codewords
00	000
01	011
10	101
11	110

19 Prof. Vishal A. Polara

An error-detecting code can detect only the types of errors for which it is designed; other types of errors may remain undetected.

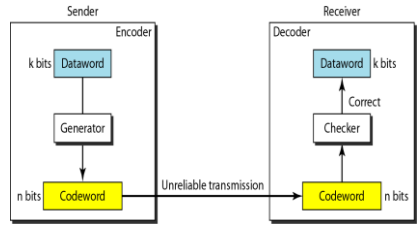
20 Prof. Vishal A. Polara

Error Correction

- In error correction we need more redundant bits.

21 Prof. Vishal A. Polara

Figure *Structure of encoder and decoder in error correction*



22 Prof. Vishal A. Polara

Example

Let us add more redundant bits to previous example to see if the receiver can correct an error without knowing what was actually sent. We add 3 redundant bits to the 2-bit dataword to make 5-bit codewords. Table shows the datawords and codewords. Assume the dataword is 01. The sender creates the codeword 01011. The codeword is corrupted during transmission, and 01001 is received. First, the receiver finds that the received codeword is not in the table. This means an error has occurred. The receiver, assuming that there is only 1 bit corrupted, uses the following strategy to guess the correct dataword.

23 Prof. Vishal A. Polara

Example

1. **Comparing the received codeword with the first codeword in the table (01001 versus 00000), the receiver decides that the first codeword is not the one that was sent because there are two different bits.**

2. **By the same reasoning, the original codeword cannot be the third or fourth one in the table.**

3. **The original codeword must be the second one in the table because this is the only one that differs from the received codeword by 1 bit. The receiver replaces 01001 with 01011 and consults the table to find the dataword 01.**

24 Prof. Vishal A. Polara

Table *A code for error correction*

Dataword	Codeword
00	00000
01	01011
10	10101
11	11110

Hamming Distance

- The Hamming distance between two words is the number of differences between corresponding bits.
- The hamming distance cab be found by applying the XOR operation on the two words and count the number of 1s in the result. It must be greater than zero.
- Minimum hamming distance is the smallest hamming distance between all possible pairs.
- It requires to define minimum hamming distance in coding scheme.
- To decide this we find the hamming distances between all words and select the smallest one.
- Any coding scheme required three parameters codeword size n, data word size k, and minimum hamming distance dmin.
- If our code is to detect upto S errors the minimum hamming distance between the valid codes must be S+1 so that the received codeword does not match the valid codeword

Hamming Distance

- The minimum hamming distance for error correction is dmin = 2t + 1.

Example

Let us find the Hamming distance between two pairs of words.

1. The Hamming distance d(000, 011) is 2 because

$000 \oplus 011$ is 011 (two 1s)

2. The Hamming distance d(10101, 11110) is 3 because

$10101 \oplus 11110$ is 01011 (three 1s)

Example

Find the minimum Hamming distance of the coding scheme

Solution

We first find all Hamming distances.

$d(000, 011) = 2$ $d(000, 101) = 2$ $d(000, 110) = 2$ $d(011, 101) = 2$
$d(011, 110) = 2$ $d(101, 110) = 2$

The d_{min} in this case is 2.

Table *A code for error detection*

Datawords	Codewords
00	000
01	011
10	101
11	110

Table *A code for error correction*

Dataword	Codeword
00	00000
01	01011
10	10101
11	11110

Example

Find the minimum Hamming distance of the coding scheme

Solution

We first find all the Hamming distances.

$d(00000, 01011) = 3$	$d(00000, 10101) = 3$	$d(00000, 11110) = 4$
$d(01011, 10101) = 4$	$d(01011, 11110) = 3$	$d(10101, 11110) = 3$

The d_{min} in this case is 3.

Example

The minimum Hamming distance for our first code scheme is 2. This code guarantees detection of only a single error. For example, if the third codeword (101) is sent and one error occurs, the received codeword does not match any valid codeword. If two errors occur, however, the received codeword may match a valid codeword and the errors are not detected.

Example

A code scheme has a Hamming distance d_{min} = 4. What is the error detection and correction capability of this scheme?

Solution

This code guarantees the detection of up to three errors (s = 3), but it can correct up to one error. In other words, if this code is used for error correction, part of its capability is wasted. Error correction codes need to have an odd minimum distance (3, 5, 7, . . .).

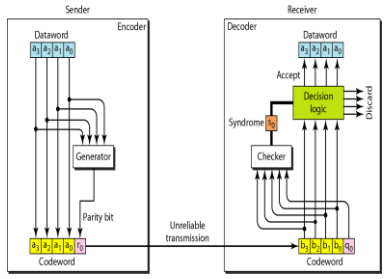
2. Linear Block codes

- The use of nonlinear block codes for error detection and correction is not widespread because of their structure.
- A linear block code is a code in which the exclusive OR of two valid codewords creates another valid codeword.
- The minimum hamming distance is the number of 1s in the nonzero valid codeword with the smallest number of 1s.
- **Simple parity check code:**
  - It is used to detect single bit error.
  - In this technique codeword is get replace by adding extra bit which is parity bit to make total number of 1s in the codeword even.
  - At receiver side addition is done over all 5 bits. It is called syndrome. If it is 0 when the number of 1s in the received codeword is even otherwise it is 1.

Table *Simple parity-check code C(5, 4)*

Datawords	Codewords	Datawords	Codewords
0000	00000	1000	10001
0001	00011	1001	10010
0010	00101	1010	10100
0011	00110	1011	10111
0100	01001	1100	11000
0101	01010	1101	11011
0110	01100	1110	11101
0111	01111	1111	11110

Figure *Encoder and decoder for simple parity-check code*



Example

Let us look at some transmission scenarios. Assume the sender sends the dataword 1011. The codeword created from this dataword is 10111, which is sent to the receiver. We examine five cases:

1. **No error occurs; the received codeword is 10111. The syndrome is 0. The dataword 1011 is created.**
2. **One single-bit error changes a_1 . The received codeword is 10011. The syndrome is 1. No dataword is created.**
3. **One single-bit error changes r_0 . The received codeword is 10110. The syndrome is 1. No dataword is created.**

Example

4. **An error changes r_0 and a second error changes a_3 .The received codeword is 00110. The syndrome is 0. The dataword 0011 is created at the receiver. Note that here the dataword is wrongly created due to the syndrome value.**
5. **Three bits—a_3, a_2, and a_1—are changed by errors. The received codeword is 01011. The syndrome is 1. The dataword is not created. This shows that the simple parity check, guaranteed to detect one single error, can also find any odd number of errors.**

2. Linear Block codes

- **Hamming Codes:**
- It is used for error correction with dmin = 3 which means it can detect upto two errors or correct one single error.
- It requires to chose m>=3. the value of n and k are then calculated from m as $n=2^m-1$ and k=n-m. the number of check bits r=m.
- For example if m=3 then n=7 and k=4.
- A copy of a 4-bit dataword is fed into the generator that creates three parity checks r0.r1,r2
- r0= a2 +a1+a0 modulo-2
- r1= a3+a2+a1 modulo-2
- r2= a1+a0+a3 modulo-2
- The checker in the decoder creates a 3-bit syndrome(s2s1s0) in which each bit is the parity check for 4 out of the 7 bits in the received codeword.

Hamming code

- The checker in the decoder creates a 3-bit syndrome(s2s1s0) in which each bit is the parity check for 4 out of the 7 bits in the received codeword.
- s0= b2+b1+b0+q0 modulo-2
- s1=b3+b2+ b1+q1 modulo-2
- s2=b1+b0+b3+q2 modulo-2
- The equation used by checker is same as used by generator with the parity check bits added to the right hand side of the equation.
- The 3 bit syndrome creates eight different bit patterns than can represent eight different conditions.
- The condition define a lack of error or an error in 1 of the 7 bits of the received codeword.

Hamming code

- **Performance:**
- It can only correct a single error or detect a double error. It can also detect burst error.
- The key is to split a burst error between several codewords one error for each codeword.

Table *Hamming code C(7, 4)*

Datawords	Codewords	Datawords	Codewords
0000	0000000	1000	1000110
0001	0001101	1001	1001011
0010	0010111	1010	1010001
0011	0011010	1011	1011100
0100	0100011	1100	1100101
0101	0101110	1101	1101000
0110	0110100	1110	1110010
0111	0111001	1111	1111111

Figure *The structure of the encoder and decoder for a Hamming code*

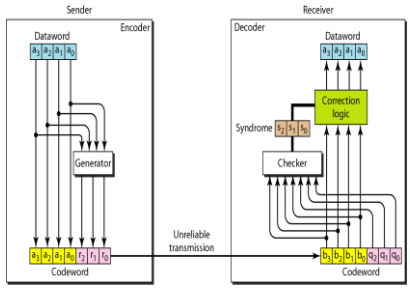


Table *Logical decision made by the correction logic analyzer*

Syndrome	000	001	010	011	100	101	110	111
Error	None	q_0	q_1	b_2	q_2	b_0	b_3	b_1

Example

Let us trace the path of three datawords from the sender to the destination:

1. **The dataword 0100 becomes the codeword 0100011. The codeword 0100011 is received. The syndrome is 000, the final dataword is 0100.**
2. **The dataword 0111 becomes the codeword 0111001. The syndrome is 011. After flipping b_2 (changing the 1 to 0), the final dataword is 0111.**
3. **The dataword 1101 becomes the codeword 1101000. The syndrome is 101. After flipping b_0, we get 0000, the wrong dataword. This shows that our code cannot correct two errors.**

Example

We need a dataword of at least 7 bits. Calculate values of k and n that satisfy this requirement.

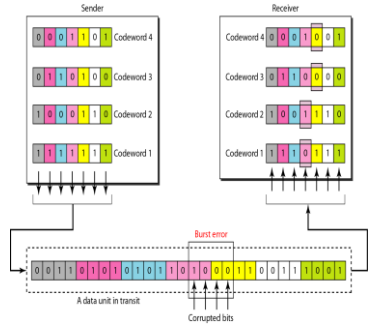
Solution

We need to make $k = n - m$ greater than or equal to 7, or $2m - 1 - m \geq 7$.

1. **If we set $m = 3$, the result is $n = 23 - 1$ and $k = 7 - 3$, or 4, which is not acceptable.**
2. **If we set $m = 4$, then $n = 24 - 1 = 15$ and $k = 15 - 4 = 11$, which satisfies the condition. So the code is**

C(15, 11)

Figure *Burst error correction using Hamming code*



3. Cyclic code

- Cyclic codes are special linear block codes with one extra property. In a cyclic code, if a codeword is cyclically shifted (rotated), the result is another codeword.
- It is used to correct error. It is used in LAN and WAN.
- In the encoder the dataword has k bits, the code word has n bits.
- The size of the dataword is augmented by adding n-k 0s to the right hand side of the word.
- The n-bit result is fed into the generator. The generator uses a divisor of size=n-k+1(4 bits), which is predefined and agreed upon.
- The generator divides the augmented dataword by the divisor(modulo-2 division).
- The remainder(r2r1r0) is appended to the dataword to create the codeword.

49 Prof. Vishal A. Polara

3. Cyclic code

- Decoder receive the codeword. Checker generate the remainder which is called syndrome of n-k bits, which is fed to the decision logic analyzer.
- The analyzer has a simple function. If the syndrome bits are all 0s, the 4 leftmost bits of the code word are accepted as the data word otherwise the 4 bits are discarded.

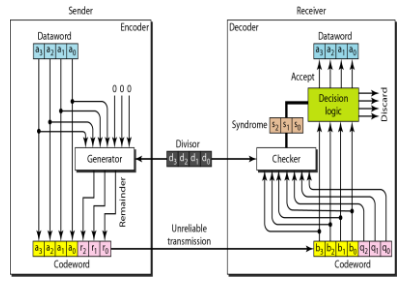
50 Prof. Vishal A. Polara

Table ***A CRC code with C(7, 4)***

Dataword	*Codeword*	*Dataword*	*Codeword*
0000	0000000	1000	1000101
0001	0001011	1001	1001110
0010	0010110	1010	1010011
0011	0011101	1011	1011000
0100	0100111	1100	1100010
0101	0101100	1101	1101001
0110	0110001	1110	1110100
0111	0111010	1111	1111111

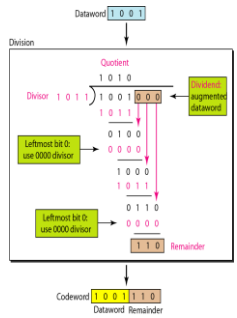
51 Prof. Vishal A. Polara

Figure ***CRC encoder and decoder***



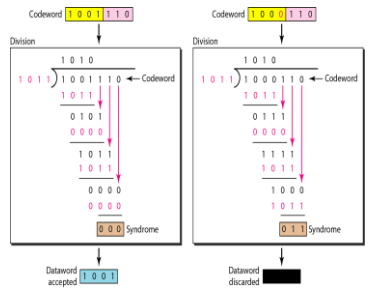
52 Prof. Vishal A. Polara

Figure ***Division in CRC encoder***



53 Prof. Vishal A. Polara

Figure ***Division in the CRC decoder for two cases***



54 Prof. Vishal A. Polara

- It is normally used for serial communication. In case of VRC odd parity bit scheme is used while for LRC even parity scheme is used.

	F	r	e	d	d	y	LRC
b0	0	0	1	0	0	1	0
b1	1	1	0	0	0	0	0
b2	1	0	1	1	1	0	0
b3	0	0	0	0	0	1	1
b4	0	1	0	0	0	1	0
b5	0	1	1	1	1	1	1
b6	1	1	1	1	1	1	0
VRC	0	1	1	0	0	0	1

3. Cyclic code in polynomial form

- A pattern of 0s and 1s can be represented as a polynomial with coefficients of 0 and 1.
- The power of each term shows the position of the bit, the coefficient shows the value of the bit.
- The degree of the polynomial is 1 less than the number of bits in the pattern.
- In example to find the augmented dataword we have left shifted the dataword 3 bits by multiplying by x3.

Figure ***polynomial to represent a binary word***

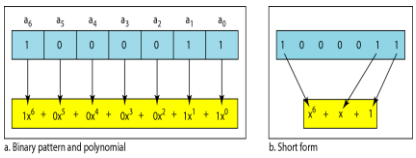


Figure ***CRC division using polynomials***

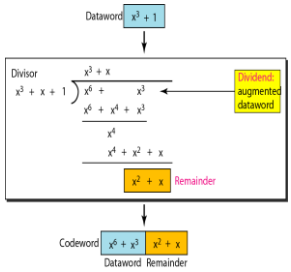


Table ***Standard polynomials***

Name	*Polynomial*	*Application*
CRC-8	x^8+x^2+x+1	ATM header
CRC-10	$x^{10}+x^9+x^5+x^4+x^2+1$	ATM AAL
CRC-16	$x^{16}+x^{12}+x^5+1$	HDLC
CRC-32	$x^{32}+x^{26}+x^{23}+x^{22}+x^{16}+x^{12}+x^{11}+x^{10}+x^8+x^7+x^5+x^4+x^2+x+1$	LANs

4. Check sum

- In checksum receiver send addition of all number with actual number
- Here addition is send as negative value.
- At receiver side again addition of actual value is done if some is 0 that means data are correct else there is an error in data.
- **One's complement:**
- In this technique the number has more than n bits, the extra leftmost bits need to be added to the n rightmost bits(wrapping).

Example

Suppose our data is a list of five 4-bit numbers that we want to send to a destination. In addition to sending these numbers, we send the sum of the numbers. For example, if the set of numbers is (7, 11, 12, 0, 6), we send (7, 11, 12, 0, 6, 36), where 36 is the sum of the original numbers. The receiver adds the five numbers and compares the result with the sum. If the two are the same, the receiver assumes no error, accepts the five numbers, and discards the sum. Otherwise, there is an error somewhere and the data are not accepted.

Example

We can make the job of the receiver easier if we send the negative (complement) of the sum, called the checksum. In this case, we send (7, 11, 12, 0, 6, −36). The receiver can add all the numbers received (including the checksum). If the result is 0, it assumes no error; otherwise, there is an error.

Example

How can we represent the number 21 in one's complement arithmetic using only four bits?

Solution

The number 21 in binary is 10101 (it needs five bits). We can wrap the leftmost bit and add it to the four rightmost bits. We have (0101 + 1) = 0110 or 6.

Example

How can we represent the number −6 in one's complement arithmetic using only four bits?

Solution

In one's complement arithmetic, the negative or complement of a number is found by inverting all bits. Positive 6 is 0110; negative 6 is 1001. If we consider only unsigned numbers, this is 9. In other words, the complement of 6 is 9. Another way to find the complement of a number in one's complement arithmetic is to subtract the number from $2^n - 1$ (16 − 1 in this case).

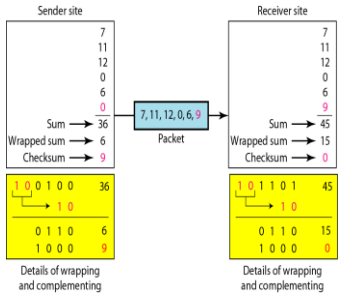
Example

Let us redo previous exercise using one's complement arithmetic. Figure shows the process at the sender and at the receiver. The sender initializes the checksum to 0 and adds all data items and the checksum (the checksum is considered as one data item and is shown in color). The result is 36. However, 36 cannot be expressed in 4 bits. The extra two bits are wrapped and added with the sum to create the wrapped sum value 6. In the figure, we have shown the details in binary. The sum is then complemented, resulting in the checksum value 9 (15 − 6 = 9). The sender now sends six data items to the receiver including the checksum 9.

Example

The receiver follows the same procedure as the sender. It adds all data items (including the checksum); the result is 45. The sum is wrapped and becomes 15. The wrapped sum is complemented and becomes 0. Since the value of the checksum is 0, this means that the data is not corrupted. The receiver drops the checksum and keeps the other data items. If the checksum is not zero, the entire packet is dropped.

Figure



5. Flow Control

- Flow control refers to a set of procedures used to restrict the amount of data that the sender can send before waiting for acknowledgment.
- Error control in the data link layer is based on automatic repeat request, which is the retransmission of data.
- Data link layer is responsible for creating frame which contain address of sender and receiver respectively.
- Two types of framing is possible:
- Fixed size framing: no need to define boundaries
- Variable size framing: it requires to define beginning and end of frame.
- Two way we can define boundary using character oriented protocols and bit oriented protocols.

Formatting Data for Transmission

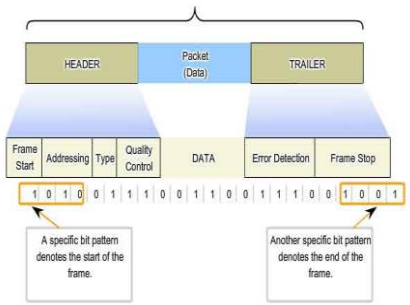
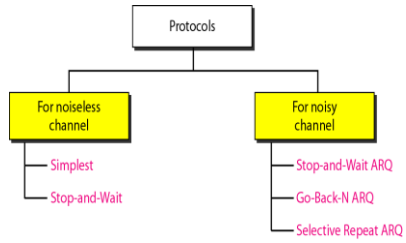


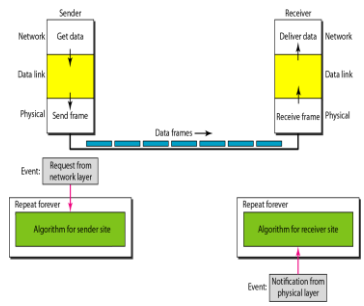
Figure *Taxonomy of protocols*



6. Noiseless Channel

- It can support two protocols.
- **Simplest Protocol:**
- It is unidirectional protocol in which data frames are traveling in only one direction from sender to receiver.
- There is no need for flow control in this scheme.
- In this protocol sender side cannot send a frame until its network layer has a data packet to send.
- The receiver site cannot deliver a data packet to its network layer until a frame arrives.
- If the protocol is implemented as procedure we need to introduce the idea of events in the protocol.

Figure *The design of the simplest protocol with no flow or error control*



Algorithm *Sender-site algorithm for the simplest protocol*

```
1  while(true)                    // Repeat forever
2  {
3   WaitForEvent();               // Sleep until an event occurs
4   if(Event(RequestToSend))      //There is a packet to send
5   {
6      GetData();
7      MakeFrame();
8      SendFrame();               //Send the frame
9   }
10 }
```

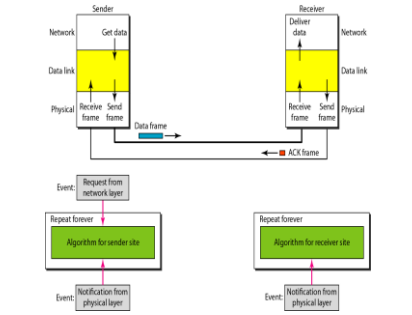

Algorithm *Receiver-site algorithm for the simplest protocol*

```
1  while(true)                    // Repeat forever
2  {
3   WaitForEvent();               // Sleep until an event occurs
4   if(Event(ArrivalNotification)) //Data frame arrived
5   {
6      ReceiveFrame();
7      ExtractData();
8      DeliverData();             //Deliver data to network layer
9   }
10 }
```


6. Noiseless Channel

- **Stop and wait Protocol:**
- It is used to overcome loss of frame when receiver is not capable to handle to much frame so frame get lost.
- In this protocol sender sends one frame, stops until it receives confirmation from the receiver and then sends the next frame.
- It also has unidirectional communication for data frames but auxiliary ACK frames from the other direction.
- In this protocol flow control is added.

Figure *Design of Stop-and-Wait Protocol*



Algorithm *Sender-site algorithm for Stop-and-Wait Protocol*

```
1  while(true)                    //Repeat forever
2  canSend = true                 //Allow the first frame to go
3  {
4   WaitForEvent();               // Sleep until an event occurs
5   if(Event(RequestToSend) AND canSend)
6   {
7      GetData();
8      MakeFrame();
9      SendFrame();               //Send the data frame
10     canSend = false;           //Cannot send until ACK arrives
11  }
12  WaitForEvent();               // Sleep until an event occurs
13  if(Event(ArrivalNotification) // An ACK has arrived
14   {
15     ReceiveFrame();            //Receive the ACK frame
16     canSend = true;
17  }
18 }
```


Algorithm *Receiver-site algorithm for Stop-and-Wait Protocol*

```
1  while(true)                    //Repeat forever
2  {
3   WaitForEvent();               // Sleep until an event occurs
4   if(Event(ArrivalNotification)) //Data frame arrives
5   {
6      ReceiveFrame();
7      ExtractData();
8      Deliver(data);             //Deliver data to network layer
9      SendFrame();               //Send an ACK frame
10  }
11 }
```

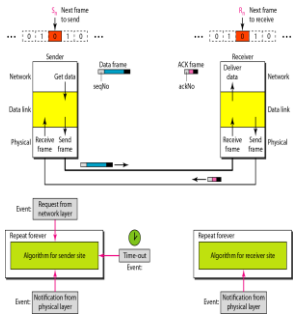
7. Noisy Channel

- It has three protocol
- **Stop-and-Wait Automatic Repeat Request**
- In this protocol flow control and error control is added.
- To detect and correct corrupted frames redundancy is added.
- In this protocol sender sends the frame and waits for the acknowledgement if he will not receive and ACK timer is set for particular frame so sender assumes that frame is discarded and he will resend it if ACK is not received.
- Sequence number is assigned to each frame.
- The sequence numbers start from 0 go to 2^m-1 and then are repeated.
- Receiver send an acknowledgement based on next frame. If 0 is arrive safe it send 1 means he require next frame.

Stop-and-Wait ARQ Protocol
OR One-bit Sliding Window Protocol

- Error correction in Stop-and-Wait ARQ is done by keeping a copy of the sent frame and retransmitting of the frame when the timer expires.
- In Stop-and-Wait ARQ, we use sequence numbers to number the frames.
- The sequence numbers are based on modulo-2 arithmetic (0,1,0,1,…).
- In Stop-and-Wait ARQ, the acknowledgment number always announces in modulo-2 arithmetic the sequence number of the next frame expected.

Figure *Design of the Stop-and-Wait ARQ Protocol*



Algorithm *Sender-site algorithm for Stop-and-Wait ARQ*

```
1  Sn = 0;                          // Frame 0 should be sent first
2  canSend = true;                  // Allow the first request to go
3  while(true)                      // Repeat forever
4  {
5    WaitForEvent();                // Sleep until an event occurs
6    if(Event(RequestToSend) AND canSend)
7    {
8      GetData();
9      MakeFrame(Sn);               //The seqNo is Sn
10     StoreFrame(Sn);              //Keep copy
11     SendFrame(Sn);
12     StartTimer();
13     Sn = Sn + 1;
14     canSend = false;
15   }
16   WaitForEvent();                // Sleep
```

(continued)

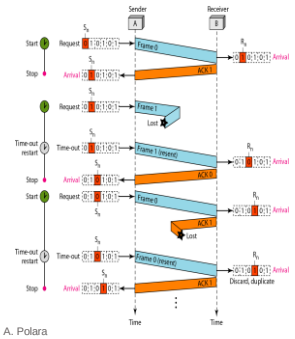
Algorithm *Sender-site algorithm for Stop-and-Wait ARQ* *(continued)*

```
17   if(Event(ArrivalNotification)      // An ACK has arrived
18   {
19     ReceiveFrame(ackNo);             //Receive the ACK frame
20     if(not corrupted AND ackNo == Sn) //Valid ACK
21       {
22         Stoptimer();
23         PurgeFrame(Sn-1);            //Copy is not needed
24         canSend = true;
25       }
26    }
27
28   if(Event(TimeOut)                  // The timer expired
29   {
30    StartTimer();
31    ResendFrame(Sn-1);                //Resend a copy check
32   }
33 }
```


Algorithm *Receiver-site algorithm for Stop-and-Wait ARQ Protocol*

```
1  Rn = 0;                          // Frame 0 expected to arrive first
2  while(true)
3  {
4    WaitForEvent();                // Sleep until an event occurs
5    if(Event(ArrivalNotification))  //Data frame arrives
6    {
7      ReceiveFrame();
8      if(corrupted(frame));
9        sleep();
10     if(seqNo == Rn)              //Valid data frame
11     {
12      ExtractData();
13       DeliverData();             //Deliver data
14       Rn = Rn + 1;
15     }
16      SendFrame(Rn);              //Send an ACK
17   }
18 }
```

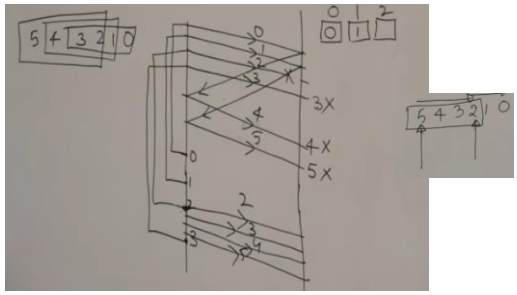
Figure *Flow diagram*



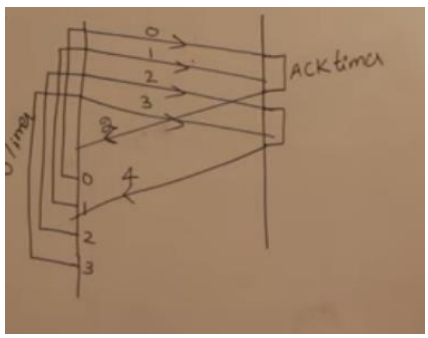
- There is a relationship between sequence number and window size is important.
- Due to lost of acknowledgment there is a chance of receiving duplicate packet.
- If sender window size is 4 , receiver window size is 1, we normally need 5 sequence number.

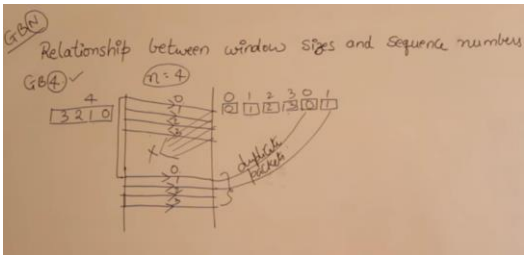
Go-Back-N Protocol

- In the Go-Back-N Protocol, the sequence numbers are modulo 2^m, where m is the size of the sequence number field in bits.
- The send window is an *abstract concept* defining an imaginary box of size $2^m - 1$ with three variables: Sf, Sn, and S_{size}.
- The send window can slide one or more slots when a valid acknowledgment arrives.
- The receive window is an abstract concept defining an imaginary box of size 1 with one single variable Rn. The window slides when a correct frame has arrived; sliding occurs one slot at a time.



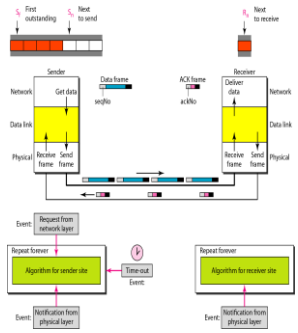
- Sliding window is theoretical protocol. It can be implemented using GBN (Go back N)and SR (selective repeat).
- In GBN N>1 is always greater than 1 else it is stop and wait.
- Receiver window size is always 1. because it will wait for one packet only.
- It will accept packet in order only packet out of order will be discarded.
- 1. If packet lost in Go back N in that case all packet from last to unacknowledged will re transmitted if the time period expires.
- 2. acknowledgement lost : it normally send acknowledgment for many packet at a time (It works on acknowledgment timer once timer expire all packet received within ack time for that packet ack will send).
- Time out timer of data should be larger than ack timer.





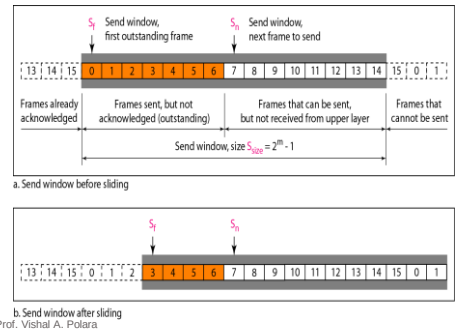
95 Prof. Vishal A. Polara

Figure *Design of Go-Back-N ARQ*



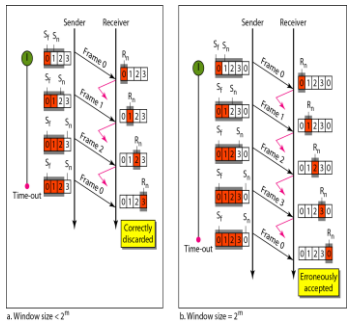
98 Prof. Vishal A. Polara

Figure *Send window for Go-Back-N ARQ*



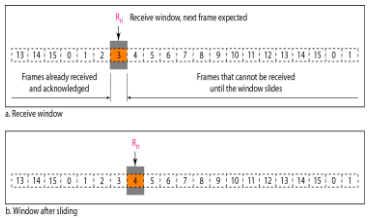
96 Prof. Vishal A. Polara

Figure *Window size for Go-Back-N ARQ*



99 Prof. Vishal A. Polara

Figure *Receive window for Go-Back-N ARQ*



97 Prof. Vishal A. Polara

In Go-Back-N ARQ, the size of the send window must be less than 2^m; the size of the receiver window is always 1.

100 Prof. Vishal A. Polara

Algorithm *Go-Back-N sender algorithm*

```
1  Sw = 2^m - 1;
2  Sf = 0;
3  Sn = 0;
4
5  while (true)                    //Repeat forever
6  {
7   WaitForEvent();
8   if(Event(RequestToSend))       //A packet to send
9   {
10     if(Sn-Sf >= Sw)             //If window is full
11          Sleep();
12     GetData();
13     MakeFrame(Sn);
14     StoreFrame(Sn);
15     SendFrame(Sn);
16     Sn = Sn + 1;
17     if(timer not running)
18          StartTimer();
19  }
20
```

(continued)

Algorithm *Go-Back-N sender algorithm* *(continued)*

```
21  if(Event(ArrivalNotification)) //ACK arrives
22  {
23     Receive(ACK);
24     if(corrupted(ACK))
25          Sleep();
26     if((ackNo>Sf)&&(ackNo<=Sn)) //If a valid ACK
27     While(Sf <= ackNo)
28      {
29       PurgeFrame(Sf);
30       Sf = Sf + 1;
31      }
32      StopTimer();
33  }
34
35  if(Event(TimeOut))             //The timer expires
36  {
37   StartTimer();
38   Temp = Sf;
39   while(Temp < Sn);
40    {
41     SendFrame(Sf);
42     Sf = Sf + 1;
43    }
44  }
45 }
```


Algorithm 11.8 *Go-Back-N receiver algorithm*

```
1  Rn = 0;
2
3  while (true)                    //Repeat forever
4  {
5   WaitForEvent();
6
7   if(Event(ArrivalNotification)) /Data frame arrives
8   {
9      Receive(Frame);
10     if(corrupted(Frame))
11          Sleep();
12     if(seqNo == Rn)             //If expected frame
13     {
14       DeliverData();            //Deliver data
15       Rn = Rn + 1;              //Slide window
16       SendACK(Rn);
17     }
18  }
19 }
```


Figure *Flow diagram*

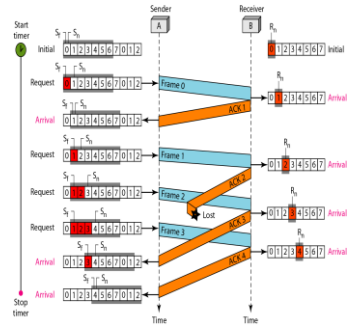
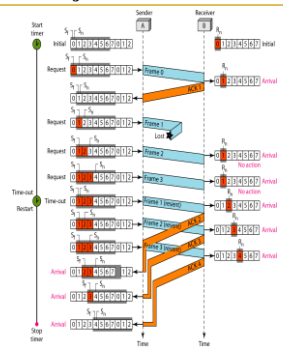


Figure *Flow diagram*



Stop-and-Wait ARQ is a special case of Go-Back-N ARQ in which the size of the send window is 1.

Selective Repeat Protocol

- It is also sliding window protocol. Here also sender window size is >1.
- Here **sender window size and receiver window size is same.** In go back N receiver window size is 1.
- In selective repeat **no order delivery** of packet is required. The packet which arrive as per the window will accepted.
- If packet is lost from 0 to 3 like 0 is lost remaining will accepted and after time out of 0 packet it will be retransmitted. Same with the loss of ack.
- Here retransmission is less compare to go back N.
- If in receive packet bit is corrupted it will send negative acknowledgment.
- It require to do sorting so cpu requirement is high.

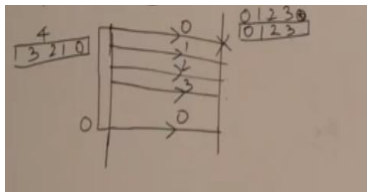


Figure ***Send window for Selective Repeat ARQ***

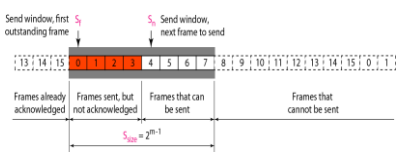


Figure ***Receive window for Selective Repeat ARQ***

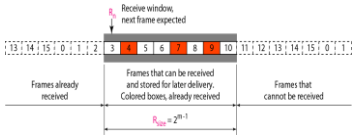


Figure ***Design of Selective Repeat ARQ***

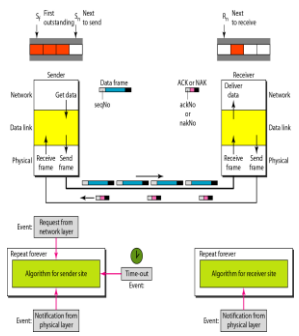
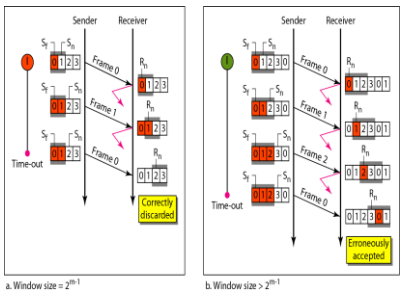


Figure ***Selective Repeat ARQ, window size***



In Selective Repeat ARQ, the size of the sender and receiver window must be at most one-half of 2^m.

Algorithm *Sender-site Selective Repeat algorithm*

```
1  Sw = 2^(m-1) ;
2  Sf = 0;
3  Sn = 0;
4
5  while (true)                        //Repeat forever
6  {
7   WaitForEvent();
8   if(Event(RequestToSend))           //There is a packet to send
9   {
10     if(Sn-Sf >= Sw)                 //If window is full
11          Sleep();
12     GetData();
13     MakeFrame(Sn);
14     StoreFrame(Sn);
15     SendFrame(Sn);
16     Sn = Sn + 1;
17     StartTimer(Sn);
18  }
19
```

(continued)

Algorithm *Sender-site Selective Repeat algorithm* *(continued)*

```
20  if(Event(ArrivalNotification)) //ACK arrives
21  {
22     Receive(frame);             //Receive ACK or NAK
23     if(corrupted(frame))
24          Sleep();
25     if (FrameType == NAK)
26        if (nakNo between Sf and Sn)
27        {
28         resend(nakNo);
29         StartTimer(nakNo);
30        }
31     if (FrameType == ACK)
32        if (ackNo between Sf and Sn)
33        {
34          while(sf < ackNo)
35          {
36           Purge(sf);
37           StopTimer(sf);
38           Sf = Sf + 1;
39          }
40        }
41  }
```

(continued)

Algorithm *Sender-site Selective Repeat algorithm* *(continued)*

```
42
43  if(Event(TimeOut(t)))          //The timer expires
44  {
45   StartTimer(t);
46   SendFrame(t);
47  }
48 }
```


Algorithm *Receiver-site Selective Repeat algorithm*

```
1  Rn = 0;
2  NakSent = false;
3  AckNeeded = false;
4  Repeat(for all slots)
5     Marked(slot) = false;
6
7  while (true)                        //Repeat forever
8  {
9   WaitForEvent();
10
11  if(Event(ArrivalNotification))     /Data frame arrives
12  {
13     Receive(Frame);
14     if(corrupted(Frame))&& (NOT NakSent)
15     {
16      SendNAK(Rn);
17      NakSent = true;
18      Sleep();
19     }
20     if(seqNo <> Rn)&& (NOT NakSent)
21     {
22      SendNAK(Rn);
```


Algorithm *Receiver-site Selective Repeat algorithm*

```
23      NakSent = true;
24      if ((seqNo in window)&&(!Marked(seqNo))
25      {
26       StoreFrame(seqNo)
27       Marked(seqNo)= true;
28       while(Marked(Rn))
29       {
30        DeliverData(Rn);
31        Purge(Rn);
32        Rn = Rn + 1;
33        AckNeeded = true;
34       }
35        if(AckNeeded);
36        {
37        SendAck(Rn);
38        AckNeeded = false;
39        NakSent = false;
40        }
41      }
42     }
43   }
44 }
```

Figure ***Delivery of data in Selective Repeat ARQ***

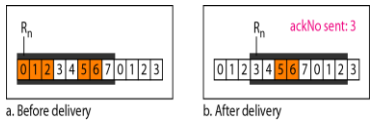


Figure ***Flow diagram***

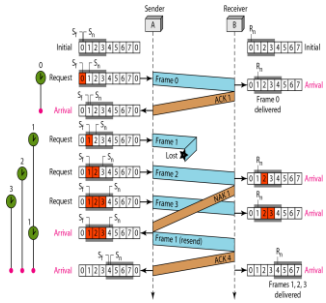
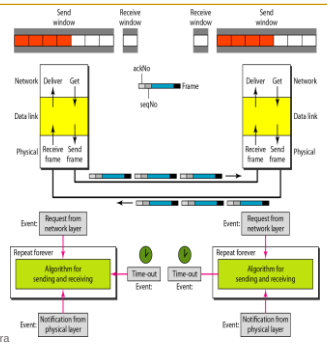


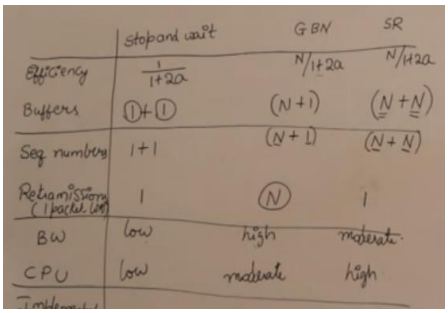
Figure ***Design of piggybacking in Go-Back-N ARQ***



Piggybacking

- In all previous example we have seen unidirectional communication.
- If we wants to do bidirectional communication then there must be some mechanism to do flow control.
- We can do by using two window at both the side sender as well as receiver side.
- Here sender is sending data to receiver and receiver is sending to sender vice versa. But receiver or sender will not send an acknowledgement directly after receiving frame. They will wait for next frame and hooked acknowledgement with next outgoing frame.
- Hooking of acknowledgement with next going frame is called piggybacking.

Comparison of sliding window protocol



Thank You