**CS 425: Computer Networks** **Jan-Apr 2020**

# Lecture 9: Computer Networks – January 31, 2020

*Lecturer: Swaprava Nath* *Scribe(s): Aryesh Dubey, Atul Pandey, Bhavita Nalli, Bhavy Khatri*



## 9.0 Recap

**Hamming(n,k) codes**

Hamming codes are linear error-correcting codes which can detect at most two-bit errors or correct one-bit error.

In mathematical terms,
for $r >= 2$ $\quad n = 2^r - 1 \quad k = 2^r - r - 1$

where r,n,k represents redundant bits, block length, message length respectively

rate of hamming codes $\quad R = k/n = 1-(r/(2^r-1))$

The possible (n,k) hamming code pairs are -:

(3,1) , (7,4) , (15,11), ..... , (255,247) where (3,1) triple repetition code is a special case of hamming code

## 9.1 Low Density Parity Check(LDPC) and Convolutional Code

### 9.1.1 Low Density Parity Check (LDPC)

LDPC code is a linear error-correcting code, a method of transmitting a message over a noisy transmission channel. An LDPC is constructed using a sparse Tanner graph (a subclass of the bipartite graph).

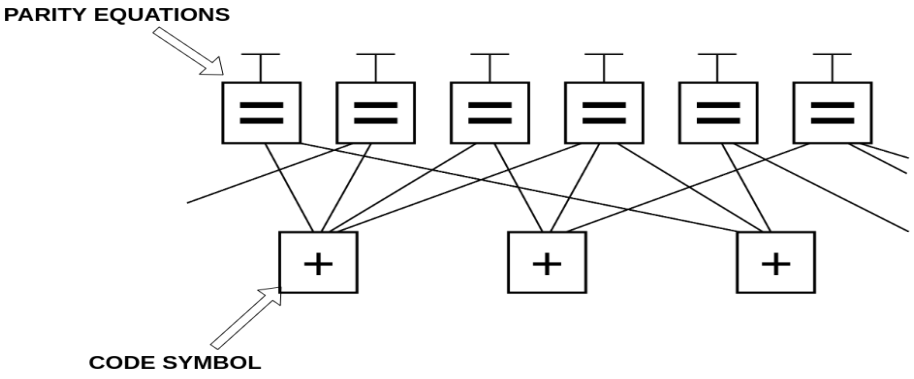


Figure 9.1: Low Density Parity Check(LDPC) Code

The graph is a sparse bipartite graph that contains two sets of nodes, one set representing the parity equations and the other set representing the code symbols. A line connects node in the first set to the second if a code symbol is present in the equation. Decoding is done by passing messages along the lines of the graph. Messages are passed from message nodes to check nodes, and from check nodes back to message nodes and their parity values are calculated probabilistically.
LDPC codes are finding increasing use in applications requiring reliable and highly efficient information transfer over bandwidth-constrained or return-channel-constrained links in the presence of corrupting noise. Besides that, these codes are very well suited for hardware implementation.[Tutorial's Point]

### 9.1.2 Convolutional Code

Convolutional coding is a widely used coding method which is not based on blocks of bits but rather the output code bits are determined by logic operations on the present bit in a stream and a small number of previous bits.
Convolutional codes are used extensively to achieve reliable data transfer in numerous applications, such as digital video, radio, mobile communications, and satellite communications. [Alan Bensky]

## 9.2 Error Detection and Correction for Different Types of Error

**Q. When to use error detection or correction?**

Let us understand it through an example.

**Example-:**

Suppose the data link has Bit Error Rate(BER) of 1 bit in 10,000 bits and the message to be delivered is of 1000 bits. The errors may be either clustered or sparse

- **Case 1:** Let us assume errors are sparse and they occur uniformly after every 10,000 bits

  As the message is of 1000 bits, we can say that the message can have at most 1 bit error

  The overhead calculations for correction and detection are as follows -:

  Correction :

  Following the method of hamming(n,k) codes, to keep k(no.of message bits) approximately equal to 1000 we should set r to be 10 i.e we need approximately 10 redundant bits to find at most 1 error bit

  Detection :

  parity bit to detect + no.of bits in message * (1/10th fraction of time in which message received is an error)

  = 1 + (1000 * 1/10)

  = 101 bits

  Thus, in this case, we prefer error correction over detection

- **Case 2:** Assume errors are clustered i.e for e.g. they come in bursts of 100 bits (where at most 2 messages are corrupted)

  Correction :

  As we already know that if the code has a hamming distance of 2*d+1 then upto d errors will always be corrected thus for an error burst of 100 bits the minimum hamming distance required between code words is 201

Detection:

As we have seen earlier that, n-bit burst errors will always be detected using an n-bit checksum

checksum bits + (fraction of messages used to re-transmit * no .of message bits)

= 100 + (2/1000 * 1000)

= 102

Thus, in this case we prefer error detection over correction

**Summary**

We should use error correction when:

1. errors are small and sparse
2. errors are expected
3. re-transmission is not possible for ex-: storage media (like CD,DVD) , real time application like online games

whereas error detection should be used when -:

1. errors are unexpected
2. errors are long

## 9.3 Layer-wise Use of Error Detection and Correction

- Error correction is generally done in the physical layer. The codes and protocols using those codes are as follows -
  - **LDPC:** 802.11, LTE, WiFi, etc
  - **Convolutional Codes:** GSM, GPRS, Satellite Communications, etc
- Error detection with retransmission is used in the link and transport layer. This is because these upper layers have to handle only the residual errors.
- Sometimes application layer itself does error correction. e.g. Some media players may try to correct the errors on a scratched CD/DVD in order to play it.

## 9.4 Retransmission: Automated Repeat reQuest(ARQ)

Till now the following topics have been covered:

- Framing
- Error detection and correction.

The natural extension to error detection would be to send the message again from sender to receiver which is commonly known as **Retransmission**. One of the common techniques of retransmission is **Automated Repeat reQuest(ARQ)**. In ARQ, the sender sends one **data packet message M** at a time to the receiver. The time taken by the message from sender to receiver is termed as **delay D**. The receiver then sends back the **acknowledgement message ACK** back to the sender denoting that it has successfully received the message. Note that the total time taken in the process, i.e. time taken by the message to reach the receiver and acknowledgment message to reach the sender, is **2D**(D+D).

But, various things can go wrong in this process which means that either the original message signal or acknowledgment message can get lost. In both, the cases sender will not be able to know that the data packet has successfully reached the receiver. To handle this problem sender waits for the **Timeout period(TO)** to check if it receives back the acknowledgment signal from the receiver and if not it retransmits the message M back to the receiver.

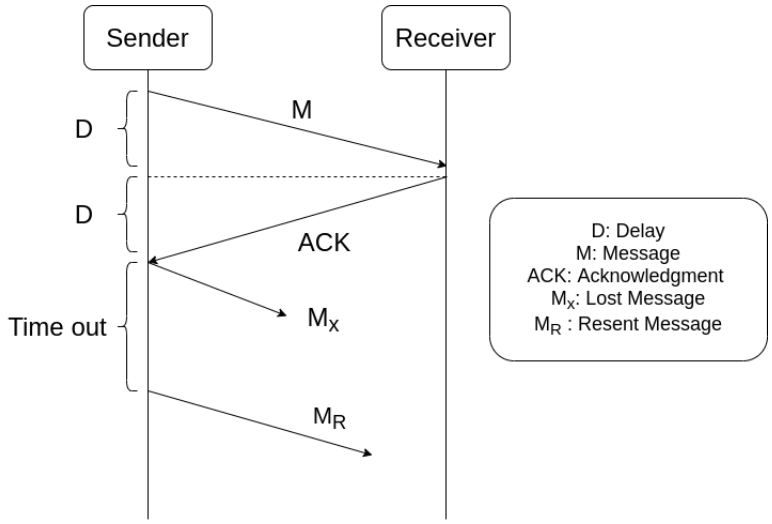


Figure 9.2: Automatic Repeat reQuest(ARQ) Mechanism

## 9.5 Design Challenges of ARQ

There are various challenges that one encounter while designing ARQ, the important among those are:

- **Designing the Timeout:** Choosing the right size of timeout is crucial as too small timeout will be the reason for many duplicate messages received by the receiver. On the other hand, if the timeout is very large, then the network will be idle for a longer period and wouldn't be sending any message. Also, finding approximately an optimal timeout period is easier to find in the Local Area Network(LAN) compared to a more complex and bigger network like the internet.
- **Problem with Duplicates:** The problem of duplicates occurs when the receiver gets more than one message from the sender. This problem arises mainly due to the following reasons:
    1. Lost ACK signal: In this case, the receiver gets the message but the acknowledgment signal is lost in between. Therefore, the sender doesn't know to get the signal that the receiver has successfully obtained the message. Therefore, after the timeout period it retransmits the message.

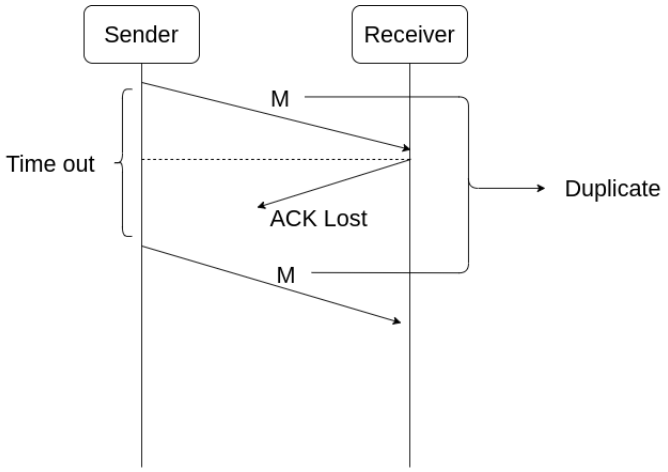


Figure 9.3: Duplicate Signal: Lost Acknowledgement

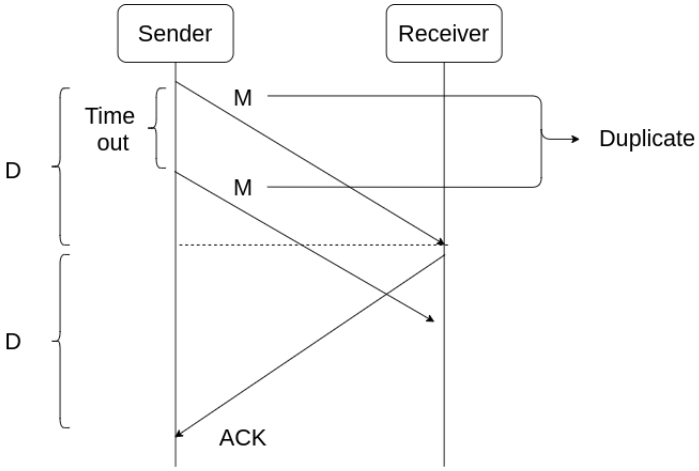


Figure 9.4: Duplicate Signal: Smaller Timeout

2. $TO < 2 \times D$: When the timeout is less than the total propagation time of the signal, the sender sends the duplicate message again even though it has already been received.

## 9.6 Handling Problem of Duplicates

### 9.6.1 STOP AND WAIT ARQ

**Stop and Wait ARQ** solves the problem of duplicates by alternatively assigning 0 and 1 to the frames. It assures that at max only one frame is outstanding at any point in time. Sender sends the first frame (F0) and waits till it gets acknowledgment (ACK0) from the receiver. If it successfully gets ACK0 before time out (TO), then only it sends the next frame (F1). It handles the cases when either signal is lost or ACK is not received before time out as follows:

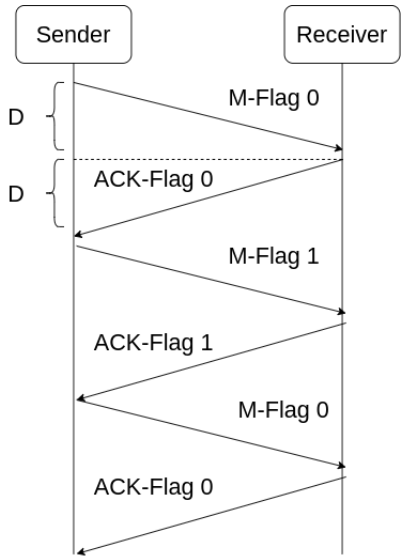


Figure 9.5: Stop and Wait ARQ

- **Message signal lost:** Suppose frame F0 is lost. The Receiver will not get the signal and therefore not send any corresponding acknowledgement (ACK0). After the time out (TO) period is elapsed, the sender will again send the same frame F0.

- **ACK signal lost:**. Suppose the receiver gets F0, acknowledges it by sending ACK0 but the acknowledgement signal is lost. After timeout (TO), the sender will again send FO as it didn't get the acknowledgement. Now the receiver has two copies of the same message. But since the frame number of both the signals are same and at max, only one signal can be outstanding, The receiver concludes that both frames are the same and discards the current frame(F0). Though it sends the acknowledgement (ACK0) again.

#### 9.6.1.1 Limitations

Although **Stop and Wait ARQ** solves the issue of duplicate messeges, it harms the performance. The sender has now to wait for **TO**(timeoout) time to send the next message. The network remains idle till that time. Lets see following example to understand how the performance gets affected -.
**Qn:** We are transmitting messages over a channel where link rate is 1 Mbps, delay is 50 ms. The frames are each of 1000 bits. What would be the speed of data transfer?
**Ans:** Since the sender has to wait for twice the delay to send next frame, it can send frames at a rate of 100ms (0.1s) i.e data rate = 1000/0.1 = 10 Kbps. Even though our link speed is so good (1Mbps), the speed is limited by the wait the sender has to do to get acknowledgement back.
The performance of this technique degrades with increasing delay. That is why it is not a good fit for large bandwidth-delay channels. Although if the delay is smaller, which is the case with smaller networks like LAN, this technique performs significantly better.

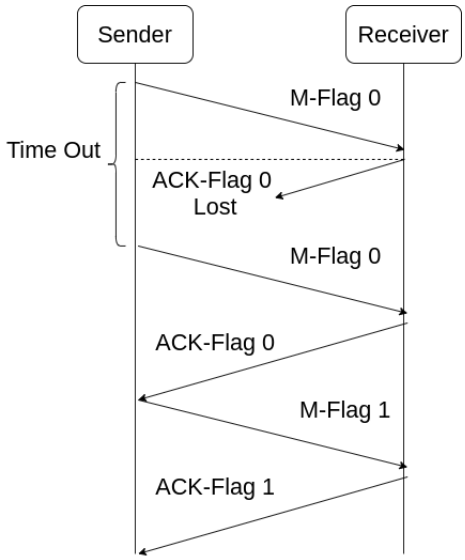


Figure 9.6: Stop and Wait ARQ: Lost Acknowledgement

### 9.6.2 GO-BACK-N-ARQ

As we can observe in the last example that the rate of data transmission is much lower as compared to link rate. So one of the ways through which we can improve the usage of a connection is Go-Back-N ARQ
**Go-Back-N ARQ** allows the sender to transmit N number of frames even without receiving an acknowledgement packet from the receiver(where N is the window size of the sender). If the receiver receives any duplicate frame it had already acknowledged or an out-of-order frame, it sends a negative acknowledgement to the sender. Once the sender receives a negative acknowledgement it will stop transmitting the frames, go back to the rejected frame and re-transmit all the subsequent frames This method efficiently uses a connection(than Stop-and-wait ARQ), since unlike waiting for an acknowledgement for each packet, the connection is still being utilized as packets are being sent i.e, during the time that would otherwise be spent waiting for packets, more packets are being sent.

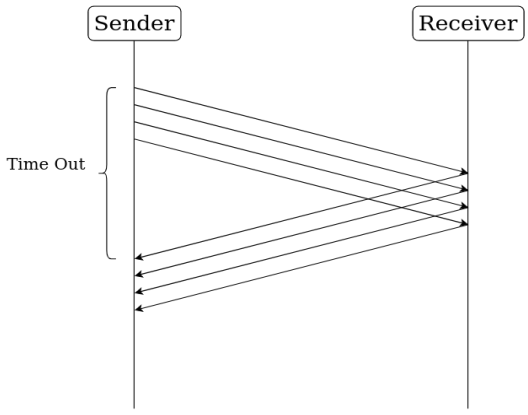


Figure 9.7: GO-Back-N ARQ

## References

[Alan Bensky] "Short-range Wireless Communication (Third Edition)", *https://www.sciencedirect.com/topics/engineering/convolutional-coding*

[Wikipedia] *https://en.wikipedia.org/wiki/Low-density_parity-check_code*

[Tutorial's Point] *https://www.tutorialspoint.com/low-density-parity-check-ldpc*