NIKHIL BADYAL

BCSE III

69

**Problem Statement -** Implement three data link layer protocols, Stop and Wait, Go Back N Sliding Window and Selective Repeat Sliding Window for flow control.

Sender, Receiver and Channel all are independent processes. There may be multiple Transmitter and Receiver processes, but only one Channel process. The channel process introduces random delay and/or bit error while transferring frames. Define your own frame format or you may use IEEE 802.3 Ethernet frame format.

Hints: Some points you may consider in your design.

**Following functions may be required in Sender**.

**Send:** This function, invoked every time slot at the sender, decides if the sender should

1. do nothing,
2. retransmit the previous data frame due to a timeout, or
3. send a new data frame.

Also, you have to consider current network time measure in time slots.

**Recv\_Ack:** This function is invoked whenever an ACK packet is received. Need to consider

network time when the ACK was received, ack\_num and timestamp are the sender's

sequence number and timestamp that were echoed in the ACK. This function must call the

timeout function.

**Timeout:** This function should be called by ACK method to compute the most recent data

packet's round-trip time and then recompute the value of timeout.

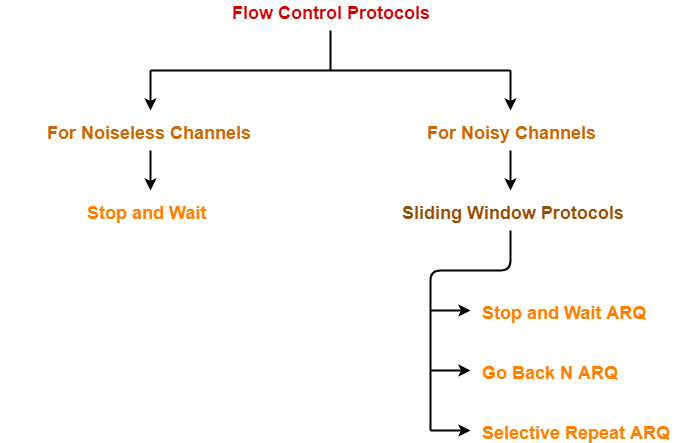
Following functions may be required in Receiver.

**Recv:** This function at the receiver is invoked upon receiving a data frame from the sender.

**Send\_Ack:** This function is required to build the ACK and transmit.

**Sliding window:** The sliding window protocols (Go-Back-N and Selective Repeat) extend the stop-and-waitprotocol by allowing the sender to have multiple frames outstanding (i.e., unacknowledged at any given time. The maximum number of unacknowledged frames at the sender cannotexceed its "window size". Upon receiving a frame, the receiver sends an ACK for the frame'ssequence number. The receiver then buffers the received frames and delivers them insequence number order to the application.

**Performance metrics:** Receiver Throughput (packets per time slot), RTT, bandwidth-delay product, utilization percentage.



In data communications, flow control is the process of managing the rate of data transmission between two nodes to prevent a fast sender from overwhelming a slow receiver. It provides a mechanism for the receiver to control the transmission speed, so that the receiving node is not overwhelmed with data from transmitting node. Flow control should be distinguished from congestion control, which is used for controlling the flow of data when congestion has actually occurred.[1] Flow control mechanisms can be classified by whether or not the receiving node sends feedback to the sending node.

Flow control is important because it is possible for a sending computer to transmit information at a faster rate than the destination computer can receive and process it. This can happen if the receiving computers have a heavy traffic load in comparison to the sending computer, or if the receiving computer has less processing power than the sending computer.

**DIAGRAM (OVER-VIEW)**

**Sender Receiver**

Data Link Layer

Data Link Layer

Physical Layer

Channel

Physical Layer

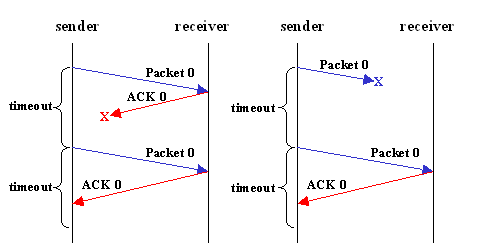
**Sender Receiver**

**GOBACKN Selective GOBACKN Selective**

**Repeat Repeat**

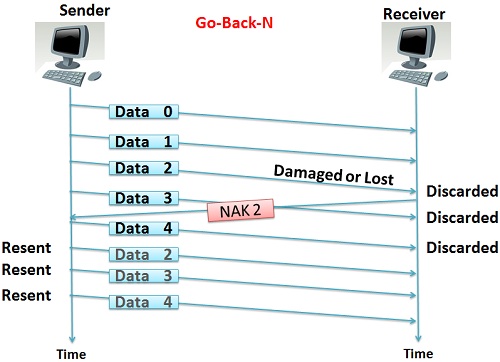


**STOP AND WAIT**

****

**Stop-and-wait ARQ**, also referred to as alternating bit protocol, is a method in telecommunications to send information between two connected devices. It ensures that information is not lost due to dropped packets and that packets are received in the correct order. It is the simplest automatic repeat-request (ARQ) mechanism. A stop-and-wait ARQ sender sends one frame at a time; it is a special case of the general sliding window protocol with transmit and receive window sizes equal to one in both cases. After sending each frame, the sender doesn't send any further frames until it receives an acknowledgement (ACK) signal. After receiving a valid frame, the receiver sends an ACK. If the ACK does not reach the sender before a certain time, known as the timeout, the sender sends the same frame again. The timeout countdown is reset after each frame transmission. The above behaviour is a basic example of Stop-and-Wait. However, real-life implementations vary to address certain issues of design.

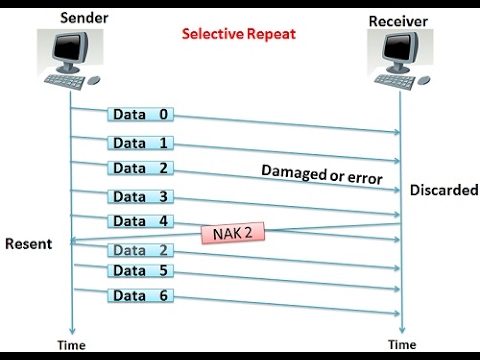
**GO BACK N**



Go-Back-N ARQ is a specific instance of the automatic repeat request (ARQ) protocol, in which the sending process continues to send a number of frames specified by a window size even without receiving an acknowledgement (ACK) packet from the receiver. It is a special case of the general sliding window protocol with the transmit window size of N and receive window size of 1. It can transmit N frames to the peer before requiring an ACK.

The receiver process keeps track of the sequence number of the next frame it expects to receive. It will discard any frame that does not have the exact sequence number it expects (either a duplicate frame it already acknowledged, or an out-of-order frame it expects to receive later) and will send an ACK for the last correct in-order frame.[1] Once the sender has sent all of the frames in its window, it will detect that all of the frames since the first lost frame are outstanding, and will go back to the sequence number of the last ACK it received from the receiver process and fill its window starting with that frame and continue the process over again.

SELECTIVE WINDOW



Selective Repeat is part of the automatic repeat request (ARQ). With selective repeat, the sender sends a number of frames specified by a window size even without the need to wait for individual ACK from the receiver as in Go-Back-N ARQ. The receiver may selectively reject a single frame, which may be retransmitted alone; this contrasts with other forms of ARQ, which must send every frame from that point again. The receiver accepts out-of-order frames and buffers them. The sender individually retransmits frames that have timed out.

**Packet format**

All packets exchanged between the sender and the receiver adheres to the following format:

Format 1 (Assignment 1)

**Explanation**

|  |  |  |
| --- | --- | --- |
| Fields | Characters | Description |
| Destination Mac | 17 | Mac Address |
| Source Mac | 17 | Same as above |
| Frame Type | 1 | Tells whether frame is end frame or not.  N = Normal Frame which is not a final segment of the message.  E = End frame which is the final segment of a message. |
| Sequence Number | 1 | Frame No. |
| Payload Length | 2 | Length of message |
| Payload (Actual Message) | 0-99 | Actual message, must not exceed the 99-char limit. |
| Checksum | 1 | Any error detection module implemented. |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Fields | Destination  MAC | Source MAC. | Frame type  (N/E) | Payload length  (decimal) | Payload  (Actual Message) | Error Detection  (Checksum) |
| Example | < | 01 | N | 06 | nikhil |  |

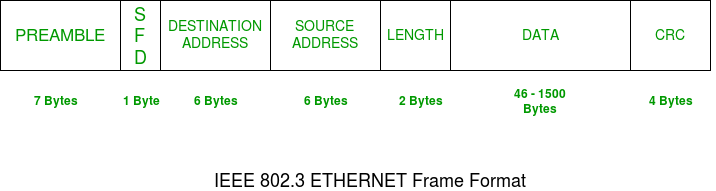
**Destination Address** – This is 6-Byte field which contains the MAC address of machine for which data is destined.

**Source Address** – This is a 6-Byte field which contains the MAC address of source machine. As Source Address is always an individual address (Unicast), the least significant bit of first byte is always 0.

**Length** – Length is a 2-Byte field, which indicates the length of entire Ethernet frame. This 16-bit field can hold the length value between 0 to 65534, but length cannot be larger than 1500 because of some own limitations of Ethernet.

**Data** – This is the place where actual data is inserted, also known as Payload. Both IP header and data will be inserted here if Internet Protocol is used over Ethernet. The maximum data present may be as long as 1500 Bytes. In case data length is less than minimum length i.e. 46 bytes, then padding 0’s is added to meet the minimum possible length.

**Cyclic Redundancy Check (CRC)** – CRC is 4 Byte field. This field contains a 32-bits hash code of data, which is generated over the Destination Address, Source Address, Length, and Data field. If the checksum computed by destination is not the same as sent checksum value, data received is corrupted.



Format 2

|  |  |
| --- | --- |
| Field | Explanation |
| Packet Type | 32 bit unsigned integer |
| Packet Length | 32 bit unsigned integer |
| Sequence Number | 32 bit unsigned integer |
| Payload | | byte sequence, maximum 500 bytes |

The **Packet Type** field indicates the type of the packet. It is set as follows:

|  |  |
| --- | --- |
| Type | Explanation |
| 0 | Data Packet |
| 1 | Acknowledgement (ACK) Packet |
| 2 | | End-Of-Transfer (EOT) Packet |

The **Packet Length** field specifies the total length of the packet in bytes, including the packet header. For ACK and EOT packets, the size of the packet is just the size of the header.

For data packets, the **Sequence Number** is the modulo 256 sequence number of the packet, i.e., the sequence number range is [0...255].

For ACK packets, **Sequence Number** is the sequence number of the packet being acknowledged.

**Receiver Physical Layer**

This java class behave like the Physical Layer in the OSI model.

It has followed function to behave like the layer.

* Main – the class starts running from this method.
* receiveStopWait () – this is the main function which creates and process and

the frames and forward these frames to the Data link layer.

* getframeNo () – This function helps in getting the no. of frames from as sender

sends them.

* datalinkLayer – This is the instance of the datalink layer which represent a

datalink layer in actual.

* It contains a methods receiveStopWait() – which accepts the frames

and check whether its ok to accept it or should it be discarded.

**Receiver DataLink Layer**

This java class behave like the Physical Layer in the OSI model.

It has followed function to behave like the layer.

* receiveStopWait () – It’s the main function which accepts the frame, extract and then process the frame.
* It then returns with the status of the status. Whether to resend or ask for next package.

**Sender DataLink Layer**

This java class behave like the Physical Layer in the OSI model.

It has followed function to behave like the layer.

* sendMessageStopWait- It is the main function which creates frames and then send them to the physical layer which in turn send them to receiver.
* createAllFrames- This method created all frames from the data and buffer them in a array.
* sendFrameCount- It acknowledge the receiver about the frame count which sender will be sending.
* SenderPhysicalLayer.sendFrameStopWait- It has a instance of physical layer. After creating frames data link layer forward them to physical layer which in turn sends to receiver.

**Sender Physical Layer**

This java class behave like the Physical Layer in the OSI model.

It has followed function to behave like the layer.

* sendFrameStopWait- It is the main function which accepts frame from DataLink layer and sends them to Receiver.

**Channel**

This channel class represent a physical channel, which can add random Delays and add errors to the frame(single bit or burst).

It has following classes to do that.

* injectBurstError- This method injects random single byte error in the frame.
* injectSingleBitError- This method injects multiple errors in the frame.
* addDelay () – It add random delays, sometime enough that sender wait time is up. So, sender resends everything.

**Sender program**

The sender program takes three arguments:

* protocol selector – 0 for Go-Back-N or 1 for Selective Repeat;
* the value of a timeout in milliseconds;
* the filename to be transferred.

The sender transfers the file reliably to the receiver program. The timeout is used as the timeout period for the reliable data transfer protocol. During the transfer, the sender program creates packets as big as possible, i.e., containing 500 bytes payload, if enough data is available. After all contents of the file have been transmitted successfully to the receiver and the corresponding ACKs have been received, the sender sends an EOT packet to the receiver. The sender exits after receiving the response EOT from the receiver. It is assumed that EOT packets are never lost.

**Receiver program**

The receiver program takes two arguments:

* protocol selector – 0 for Go-Back-N or 1 for Selective Repeat;
* the filename to which the transferred file is written.
* When the receiver program receives the EOT packet, it sends an EOT packet back and exits.

Both sender and receive will run using the same protocol. Go-Back-N on both ends, or selective repeat on both ends.

**Addressing**

The following addressing scheme is used with OS-assigned port numbers.

1. The receiver program is started first and must write its 'R' socket address information (hostname and port number) into a file \*recvInfo\* that is read by the channel emulator.
2. The channel emulator is started next and uses this information to send packets towards the receiver.
3. The same mechanism is used between the sender and the emulator, i.e., the emulator writes its 'B' addressing information into a file \*channelInfo\* which is then read by the sender.

All files are read and written in the current directory. The contents of each file are the IP address (or hostname) and port number, separated by space.

**Channel**

The channel emulator is started with the following syntax:

*java -jar channel.jar <max delay> <discard probability> <random seed> <verbose>*

1. All Data and ACK packets are subject to a random delay, uniformly distributed between 0 and `<max delay>` milliseconds.
2. All Data and ACK packets are subject to random discard with a probability of `<discard probability>`.
3. If `<random seed>` is set to a non-zero value, this seed is being used to initialize the random number generator. Multiple runs with the same seed produce the same channel behaviour. If `<random seed>` is set to zero, the random number generator is seeded with the current system time.
4. If `<verbose>` is set to a non-zero value, the channel emulator outputs information about its internal processing.

**How to run**

1. Run the Receiver

*java Receiver <protocol selector> <filename>*

The Receiver will output the **recvInfo** file.

1. Run the Channel

*java -jar channel.jar <max delay> <discard probability> <random seed> <verbose>*

1. Run the Sender

java Sender <protocol selector> <timeout> <filename>

**Design ideas**

Both protocols use the common class, called **Packet**, for the transmission packet. This class follows the packet format described in the assignment, which can easily convert to the \*Bytes\* format and vice versa. Here are some more design ideas for the two protocols.

**Go-Back-N protocol**

The design follows the description of Go-Back-N as discussed in class. In the **GBNSender**, a queue and a semaphore with initial value of 10 are used to implement sender's sliding window. The sender uses a single timer to schedule the timeout task, which sends out all the packets in the current window. For the **GBNReceiver**, cumulative acknowledgement is used to inform the sender of the receiving status. Some multi-threading techniques such object lock are used to synchronize the operations for the sender queue and timer.

**Selective Repeat protocol**

The Selective Repeat version of pipelined reliable data transfer also follows the description as discussed in class. Each packet comes with its own logical timer, called **TimerPacket**. The **SRSender** uses a **hashmap**, a queue and a semaphore to implement the sliding window, which allows O(1) runtime access to the packet and fast queue operations (**poll**, **peek** and **offer**). The selective acknowledgement is used on the receiver side, which allows to reduce unnecessary retransmissions of packets.



SENDER

Main

Create a local Copy and start timer.

Remove Packet from buffer.

Remove Packet from buffer.

Channel

SendPacket

SendPacket

Create a local Copy and start timer.

SendData

SendData

Packet S

ACK Rec

Packet S

ACK Rec

Start ()

Start ()

Selective

GBN

Channel

Pass Packet to receiver

Pass Packet to sender

Discard Packet

Add Random delay

Stream to receive DATA

Process Packet

Stream to receive ACK

main

SENDER

Main

Packet Rec

ACK Send

ACK send

Packet RecvK Rec

Channel

SendACK

SendACK

SendData

SendData

Start ()

Start ()

Selective

GBN

main()

**Channel**

injectError ()

AddDelay

Send ()

SenderPhysicalLayer.sendFrameStopWait

sendFrameCount ()

createAllFrames ()

sendmessageStopWait ()

Main()

checkChecksum ()

sendAckStopWait

dataLinkLayer.receiveStopWait

extractCrc ()

getFrameType

extractFrameNo

getNoFrame

receiveStopWait()

**Analysis**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **Stop and Wait ARQ** | **Go back N** | **Selective Repeat** | **Remarks** |
| **Efficiency** | 1 / (1+2a) | N / (1+2a) | N / (1+2a) | Go back N and Selective Repeat gives better efficiency than Stop and Wait ARQ. |
| **Window Size** | Sender Window Size = 1  Receiver Window Size = 1 | Sender Window Size = N  Receiver Window Size = 1 | Sender Window Size = N  Receiver Window Size = N | Buffer requirement in Selective Repeat is very large.  If the system does not have lots of memory, then it is better to choose Go back N. |
| **Minimum number of sequence numbers required** | 2 | N+1 | 2 x N | Selective Repeat requires large number of bits in sequence number field. |
| **Retransmissions required if a packet is lost** | Only the lost packet is retransmitted | The entire window is retransmitted | Only the lost packet is retransmitted | Selective Repeat is far better than Go back N in terms of retransmissions required. |
| **Bandwidth Requirement** | Bandwidth requirement is Low | Bandwidth requirement is high because even if a single packet is lost, entire window has to be retransmitted.  Thus, if error rate is high, it wastes a lot of bandwidth. | Bandwidth requirement is moderate | Selective Repeat is better than Go back N in terms of bandwidth requirement. |
| **CPU usage** | Low | Moderate | High due to searching and sorting required at sender and receiver side | Go back N is better than Selective Repeat in terms of CPU usage. |
| **Level of difficulty in Implementation** | Low | Moderate | Complex as it requires extra logic and sorting and searching | Go back N is better than Selective Repeat in terms of implementation difficulty. |
| **Acknowledgements** | Uses independent acknowledgement for each packet | Uses cumulative acknowledgements (but may use independent acknowledgements as well) | Uses independent acknowledgement for each packet | Sending cumulative acknowledgements reduces the traffic in the network but if it is lost, then the ACKs for all the corresponding packets are lost. |
| **Type of Transmission** | Half duplex | Full duplex | Full duplex | Go back N and Selective Repeat are better in terms of channel usage. |

**Conclusions-**

* Go back N is more often used than other protocols.
* SR protocol is less used because of its complexity.
* Stop and Wait ARQ is less used because of its low efficiency.
* Depending on the context and resources availability, Go back N or Selective Repeat is employed.
* Selective Repeat and Stop and Wait ARQ are similar in terms of retransmissions.
* Go back N and Selective Repeat are similar in terms of efficiency if sender window sizes are same.
* SR protocol may be considered as a combination of advantages of Stop and Wait ARQ and Go back N.
* SR protocol is superior to other protocols but because of its complexity, it is less used.

