COMPARISON OF STOP AND WAIT ARQ PROTOCOL AND GO BACK N ARQ PROTOCOL IN NS2 SIMULATOR

Report submitted to the SASTRA Deemed to be University
as the requirement for the course

CSE302: COMPUTER NETWORKS

Submitted by

KANAMARLAPUDI VENKATA NAGA SREEVALLI
(Reg. No.: 123015126, Information Technology)

February 2022

SCHOOL OF COMPUTING
THANJAVUR, TAMIL NADU, INDIA – 613 40

SCHOOL OF COMPUTING

THANJAVUR – 613 401

Bonafide Certificate

This is to certify that the report titled "**COMPARISON OF STOP AND WAIT ARQ PROTOCOL AND GO BACK N ARQ PROTOCOL IN NS2 SIMULATOR**" submitted as a requirement for the course, **CSE302: COMPUTER NETWORKS** for B.Tech. is a bonafide record of the work done by **Shri/Ms. Kanamarlapudi Venkata Naga Sreevalli (Reg. No.123015126), BTech. Information Technology)** during the year 2021-2022, in the School of Computing.

Project Based Work *Viva voc*e held on 12-02-2022

Examiner 1 **Examiner 2**

TABLE OF CONTENTS

ACKNOWLEDGEMENTS

First of all, I would like to thank God Almighty for his endless blessings.

I would like to express my sincere gratitude to **Dr S. Vaidyasubramaniam, Vice-Chancellor** for his encouragement during the span of my academic life at SASTRA Deemed University.

I would forever remain grateful and I would like to thank **Dr A. Umamakeswri, Dean, School of Computing and R. Chandramouli, Registrar** for their overwhelming support provided during my course span in SASTRA Deemed University.

I am extremely grateful to **Dr. Shankar Sriram, Associate Dean, School of Computing** for his constant support, motivation and academic help extended for the past three years of my life in School of Computing.

I would specially thank and express my gratitude to Dr. Dindyal Mahto**, Associate Professor, School of Computing** for providing me an opportunity to do this project and for her guidance and support to successfully complete the project.

I also thank all the Teaching and Non-teaching faculty, and all other people who have directly or indirectly help me through their support, encouragement and all other assistance extended for completion of my project and for successful completion of all courses during my academic life at SASTRA Deemed University.

Finally, I thank my parents and all others who help me acquire this interest in project and aided me in completing it within the deadline without much struggle.

ABSTRACT

In communication (transfer of data) between sender and receiver, sender will always desire to send noise free and loss free data to receiver without error and delay, but in general there will be noise and loss in between medium which sometimes cause loss and noise and cause unreliable data transfer. In these situations, sender don't know if receiver received data correctly and receiver don't know if sender sent data correctly. So, the responsibility of sending data correctly should be taken by someone. We know TCP/IP is a 5layer Internet protocol in which 2nd layer is transport layer (end to end connection oriented) and 3rd layer is Network layer (connection less). When there is a transmission of data from sender to receiver the transport layer of sender is virtually connected to transport layer of receiver. In between there will be lot of routers. Network layer will decide through which router the data packet need to be transmitted. Here as network layer is connectionless, we will face unreliable data transfer because of which noise occur. To eliminate this, we take the help of transport layer which provide **reliable data transfer** as one of its service. This reliable data transfer provides guarantee delivery of data, uncorrupted data and loss less data. As transport layer is providing all these services this layer is heavy weight. So, to provide this reliable data transfer in this noisy and lossy channel we use protocols called arq (automatic repeat request) protocols which is a error control mechanism for data transmission. Arq is in ethernet 802.3 local area network standard. It uses acknowledgements and timers with which we can achieve reliable data transfer. In this two of the arq protocols **stop and wait** and **go back n** arq protocols definitions, methodologies, analysis of these arq protocols in normal, noisy and lossy channel, comparison of these two protocols based on certain parameters like throughput, delay, packet loss rate, link utilization in different environments using ns2 simulator, finite state machines of stop and wait arq and go ack n arq protocols and finally my own idea to improvise arq protocols are discussed.

Keywords:

Reliable data transfer; End to End connection oriented; Transport layer; Network layer; Protocols; Automatic repeat request(arq); Stop and Wait arq; Go Back N arq; Throughput; Packet loss ratio; Delay; Link utilization; Finite state machines; NS2 simulator;

LIST OF FIGURES

LIST OF TABLES

NOTATIONS

Notation	Description
U	Link Utilization
a	ratio of propagation delay to transmission delay
L	length of the data packet
R	Bandwidth
D	distance between sender and receiver
V	velocity
n	number of packets transmitted
ms	millisecond
μs	micro second

ABBREVIATION

RTT	Round Trip Time
ARQ	Automatic Repeat Request
ACK	Acknowledgement
NACK	Negative Acknowledgement
Dp	Propagation delay
Dt	Transmission delay

CHAPTER 1

INTRODUCTION

When there is data transferring between sender and receiver in a unreliable mode, there is a possibility of some means of noise or loss of packets. because of which there is a chance of receiving errors in data. When this is done sender or receiver should take the responsibility to rectify the errors. Here in network layer there is no guarantee of data transfer.so transport layer provided some services which make guarantee delivery of data. transport layer provides connection oriented reliable service over connection less unreliable network service, so it is is heavy weight. **reliable data transfer** is one of the service of transport layer responsible for guarantee delivery of data.

Let us assume **loss free and noise free channel (idle case):**

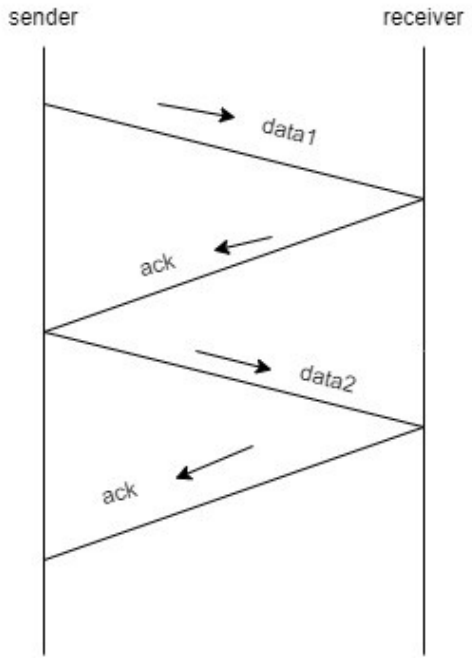


Fig-1.1 Reliable Data Transfer in noisy free and loss free channel

Here application layer and transport layer entity works on same computer so there will be reliable data transfer between application and transport layer. After receiving data from application layer transport layer will attach header to data which contains port number and ip address as a packet to network layer. Now the sender's transport layer send data to receiver through network layer.as the channel is noise free and loss free the data will reach receiver correctly. After receiving data the receiver will send ack (positive acknowledgement) to sender, so the sender will know that the receiver received data packet correctly. The sender will receive ack after RTT (round trip time)

period of time. It means for RTT period of time the sender will be idle. If the application layer haven't send data to sender transport layer then sender will wait for (RTT +time to receive data from application layer.). Here sender transport layer gets data from application layer(above) where as receiver transport later receives data from network layer(below).

But this case won't happen because there will be noise and loss in the channel in general case. But, for guarantee delivery of data there should be no noise and loss. So **automatic repeat request protocol** is used.

ARQ Protocol:

802.3 local area network ethernet is the standard of Arq is in . It is not suitable for satilite communication or wide range communication. Arq is also known as automatic repeat query. ARQ is one such error control mechanism in which receiver uses error detection methods to check if there is any corruption in data. Error control mechanisms uses acknowledgement and time out to validate the data. if it finds any error it sends a negative ack to sender and ask sender to retransmit the same data. Also we use timer at the sender side to get an idea about if the data received receiver. if sender does not receive an ack then sender will think there is an loss occurred and it will send again the same packet after timer expires (after RTT period of time).

STOP AND WAIT ARQ PROTOCOL (Bit Alternative protocol):

It is an arq protocol in which sender sends only one packet at a time and wait to get an acknowledgement from the receiver. The packet contains payload and checksum. the sender will send the next packet If receiver sends a positive ack. sender will retransmit the same packet if receiver receive a corrupted packet . The receiver will find if data is corrupted or not using error detection code (parity bits). If the receiver does not send the packet before RTT(round trip time) the sender will think there is a loss in data packet or acknowledgement packet, it will retransmit the same packet.

How the receiver verify if received packet is noise free data or not?

For this we add a new bit called checksum (parity bit) to data. If number of 1's are even checksum is 0. If number of 1's are odd checksum is 1.

10101010---------- >10101110

Sender side data has 4 one's so checksum is 0, receiver side data has 5 one's so the checksum is 1. As checksum is not same in sender and receiver side we can say there is some error in the data due to noisy channel. Now receiver ask sender to retransmit the same data in the form of negative acknowledgement (NACK). now sender will send the same packet. Here thee is a drawback ,if there is a corruption in ack then sender cannot know if the data received receiver or not. In this situation the sender will again retransmit the data thinking the data haven't reached receiver properly but the receiver has that data, in this situation if sender retransmit then there will be the

possibility of duplication and also there will be extra overhead. So to eliminate this problem we are going to add an extra bit called sequence number. From now the sender will receive 3 types of ack. ACK with sequence number 0, ACK with sequence no 1, corrupted ACK. Whenever the receiver received corrupted packet, the receiver will send ack of previously received packet to speed up the process.

RELIABLE DATA TRANSFER IN NOISY AND LOSSY CHANNEL (Stop and Wait):

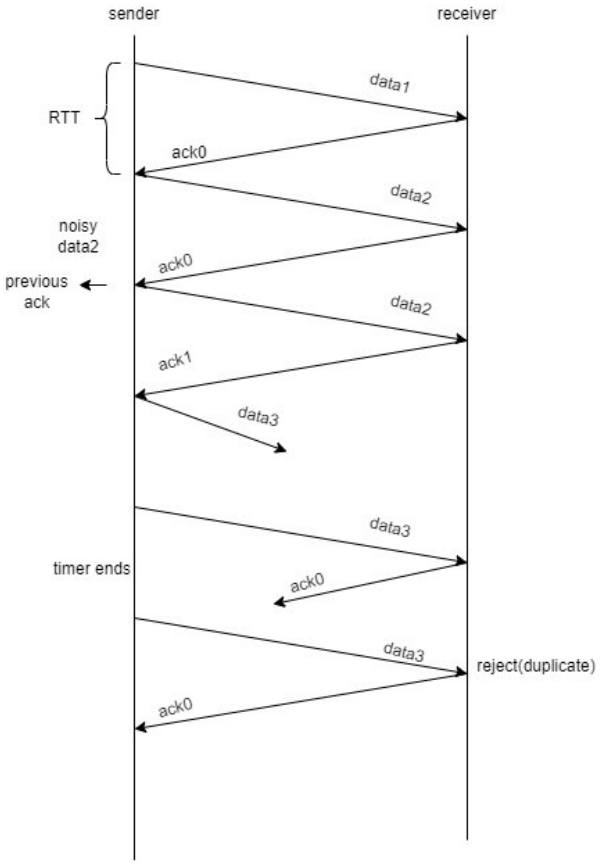


Fig-1.2 Reliable data transfer in noisy and lossy channel (stop and wait protocol)

1) Data1 is transmitted from sender, packet reached receiver without corruption, receiver sent ack0 to sender
2) Data2 sent to receiver, corruption occurred, as corruption occurred the receiver sent the previous ack that is ack0 again to sender.
3) as sender got ack0 sender will retransmit the same data2 to receiver, no corruption, so receiver sent ack1(next sequence number) to sender.
4) Now, sender transmitted data3 to receiver, but in middle data3 is lost, sender side countdown timer will be present which is set to RTT, as data3 is lost, ack will not reach sender, sender will wait till RTT time till timer expires and then it will again retransmit same data3 packet.

5) Sender retransmits same data3 packet to receiver, receiver received data3 with no corruption, but now acknowledgment got lost. As ack not reached sender, after timer expired, sender will get confusion whether data is lost or ack is lost.
6) so sender will again send the same data3 packet to receiver. But receiver already have data3, duplication occurred, so receiver will drop data3 and send ack0 to sender.

Drawback of Stop and Wait arq protocol:

Here after we sent a data packet the sender should be idle for RTT period of time. So the link utilization will be less. So to improve this link utilization we will use another protocol called **Go Back N protocol.**

GO BACK N ARQ PROTOCOL:

Before going to learn go back n arq protocol we learn about pipelining protocol.

Pipelining protocol:

Without waiting for acknowledgement Allow the sender to send multiple packets one after other This protocol is more suitable for wide area network. This is more suitable for wide area networks. This Go back N protocol comes under pipelining protocol. This Go Back N is a **sliding window protocol.**

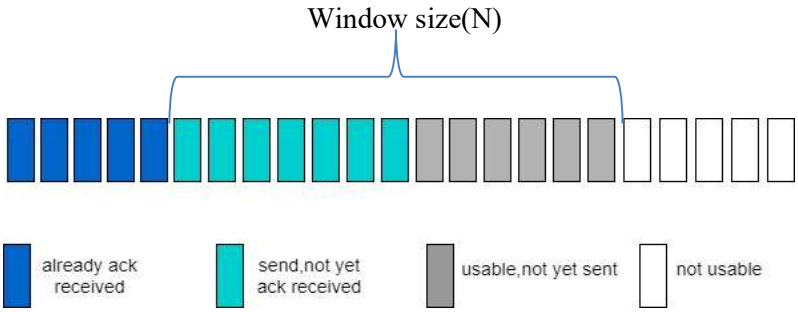


FIG-1.3 sender's view of sequence numbers in Go Back N

In stop and wait protocol as we are sending only one packet at a time we used 1 bit sequence number (0 or 1). But here we are sending N packets at a time , so we have to use sequence number from 0 to N-1. If number of bits in sequence number is i, then window size should be 2^i. if window size id 4 the sequence numbers will be (00,01,10,11).

Here, if we take window size as 4, the sender will send 4 packets at the same time, without waiting for ack from receiver. Sender also stop sending 5^{th} packet as window size is only 4. When sender got 1^{st} ack, 1^{st} packet will get removed (slide out) from the window, and now 5^{th} packet will enter into window and send to receiver.

RELIABLE DATA TRANSFER IN NOISY AND LOSSY CHANNEL (Go Back N):

The sender is allowed to transmit multiple packets without waiting for ack, but is constrained to have no more than maximum allowed number N, of unacknowledged packets in the pipeline. The sender will maintain a window of size 4. The sequence no will be 00 01 10 11. Let take 6 packets p1 p2 p3 p4 p5 p6. In sender window 1st 4 packets will be there. Size of sender window is N (0 to n-1). Size of receiver window is 1. Sender window will be slide only after getting oldest unacknowledged packet. If we take receiver window it won't slide. It is initialized with seq no of 1st packet.. If it get any other seq no other than required one it will discard and send previously transmitted seq number (Duplicate ack). So that the sender will get to know there is some error and send the same packet. If the receiver received correct data the receiver will send the ack to sender and increment (module increment) the seq number in receiver window by one.

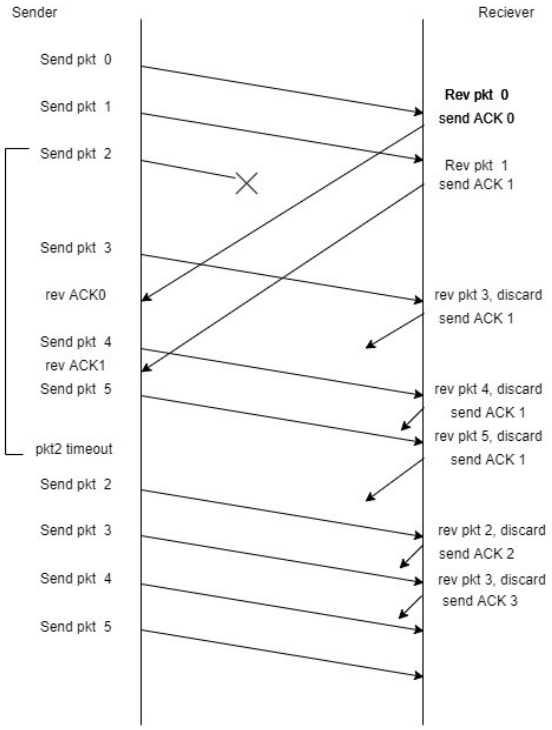


Fig-1.4 Reliable data transfer in noisy and lossy channel (Go Back N protocol)

Explanation for Fig-1.4:

1) Sender send packet 0, packet 1, packet 2, packet 3 to receiver without waiting for ack.
2) Packet 0 is sent to receiver and receiver received correctly and start sending ack0 to sender.
3) Packet 1 is sent to receiver and receiver received correctly and start sending ack1 to sender.
4) Here packet 2 is lost. So, packet 2 is not send to receiver.

5) Now packet 3 is send to receiver but there is a error in data when checked using parity bit, now as data has error, the receiver send previous ack that is ack1 to sender. So that the sender will send the same packet 3 to receiver.
6) Now the sender received ack 0. Now in the window packet 0 is removed and packet 4 will be entered and sender will send packet 4 to receiver.
7) Now the sender received ack 1. Now in the window packet 1 is removed and packet 5 will be entered and sender will send packet 5 to receiver.
8) Here time out occurs for packet 2, so the sender will retransmit all the unacknowledged packets (pkt 2 pkt 3 pkt 4 pkt 5) to receiver again as to send all packets in order according to sequence number.
9) now receiver start receiving all pkt2, pkt3, pkt4, pkt5 and send respective acks to sender.

CHAPTER 2

RELATED WORKS

Busquets González et al. (Busquets González,2021) we propose a new hybrid ARQ error control scheme for the data communication system with multiple parallel channels. It is a combination of Go-Back-N ARQ and Selective-Repeat ARQ protocols. He described the complete operations of the hybrid ARQ scheme and evaluated the throughput efficiency two or three different channels by assumption.

Cipriano et al. (Cipriano,2010) presented mechanisms for automatic repeat request (ARQ) and hybrid ARQ (HARQ) implemented in beyond third generation wireless systems based on OFDMA. He focused on part of the IEEE 802.16 standard family (IEEE 802.16-2005, IEEE 802.16m). He showed some performance curves how HARQ can help in mobility context in reducing performance degradation.

Dosti at el. (Dosti,2017) Analyzed the type-I automatic repeat request (ARQ) protocol performance with ultra-reliability constraints. He showed it is difficult task to achieve a very low packet outage probability.

Karetsi et al (Karetsi,2022) proposed approach Ultra reliable less latency for Communication. He made the study that schemes bear significant limitations stemming from the fact that the ARQ mechanism called the sliding window. For this he proposed the use of two distinct windows; the *sliding window*, and the *coding window* which is widely used in the coding process. He analyzed the performance of the strategy proposed and said it will provide an improved operation in coding at the same time reduces the complexity of coding.

Nada et al. (Nada,2021) proposed a new model mathematically to analyze the service time of Go-Back-N ARQ protocol over noisy channels. The system is a stochastic process. It focused on distributions of service time. Probability Generating Functions (PGF) is derived in terms of error rate parameters and message size.

Valera et al. (Valera,2009) presented a modular multihop automatic repeat request scheme in a system. He evaluated the performance of the opportunistic ARQ using underwater acoustic modems in a shallow underwater.

Vasiliev et al. (Vasiliev 2019) assessed the efficiency of application layer automatic repeat query (AL-ARQ) algorithm in UAV-assisted network. He emulated HD video streaming using B.A.T.M.A.N. routing protocol, Ubuntu Mate 16 virtual machines, and tool called NS-3 simulation.

Zhao et al. (Zhao,2005) proposed an interesting approach to networks comprising multiple relays operating over orthogonal time slots based on a g hybrid (ARQ) generalization.

CHAPTER 3

PROPOSED FRAMEWORK

Analysis of stop and wait arq protocol and go back n arq protocol:

Here we are going to analysis two parameters:

1. Link Utilization
2. End to End delay

Link Utilization for stop and wait arq protocol:

The formula for link utilization is:

$$U = \frac{1}{1+2a}$$

Link Utilization Parameters for Stop and Wait protocol:

Parameters	**values**
Length of packet (L)	1000 Bytes
Bandwidth (R)	2 Gbps
propagation delay (Dp)	20 ms
Transmission delay (Dt)	4 microseconds

propagation delay:

Time taken by the packet to move from one end to other in the channel.

Dp=Distance/Velocity

=20ms

Transmission delay:

Time taken by the node to transmit packet.

Dt=L/R

=1000*8/2*10^9 =4 microseconds.

a=Propagation delay/transmission delay

now, link utilization is 1/1+2*a=0.004/0.004+20=**0.000199**

If utilization is 1 then entire channel bandwidth is used high utilization. Here it is 0.00019 which is closer to zero so it has less link utilization.

Here the actual utilization is R*U=2*10^9*0.000199=398kBps.

Means in 2GBps bandwidth channel only 398kbps is used. so we can say there is less utilization.

Link Utilization for Go Back N arq protocol:

The formula for link utilization is:

$$U = \frac{1 * n}{1+2a}$$

Link Utilization Parameters for Go Back N protocol:

Parameters	**values**
Length of packet (L)	1000 Bytes
Bandwidth (R)	2 Gbps
propagation delay (Dp)	20 ms
Transmission delay (Dt)	4 microseconds
Number of packets (n)	4

Now in RTT period of time we are transmitting 4 packets so utilization will be

U=4*(utilization of 1 packet)

U=4*0.000199

U=0.000796

Here the actual utilization is R*U=2*10^9*0.000796=1592kBps.

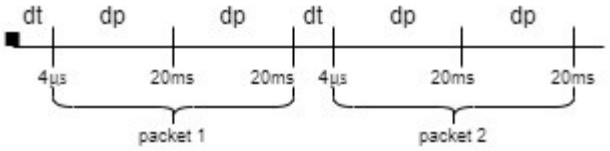
Means in 2GBps bandwidth channel only 1592kbps is used.

Comparison of link utilization and actual utilization:

protocol	Link Utilization	number of bytes are utilized in the given 2GBps bandwidth
Stop And Wait	0.000199	398KBps
Go Back N	0.000796	1592KBps

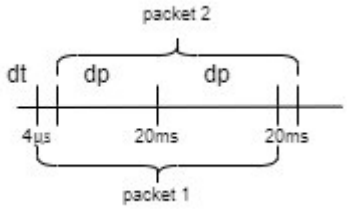
Here, we can see link utilization of go back n is also less, but as compared to stop and wait protocol the link utilization is somewhat more. So, by using go back n we can improve link utilization.

End to End Delay of Stop and Wait protocol:



So, the total time the sender should wait for forwarding next packet to receiver is 40.004ms which is total end to end delay for one packet. That means the sender should wait for 40.00ms to send 2nd packet. To send 4 packets sender will take 4*40.004ms = 160.016ms.

End to End Delay of Go Back N protocol:



So, the total time the sender should wait for forwarding next packet to receiver is 40.004ms which is total end to end delay for one packet. But in go back n there is no need to wait for 40.004ms, the sender can send the 2nd packet after dt (4µs). To send 4 packets sender will take 40.004ms+3(4µs) = 40.016ms.

In stop and wait protocol the sender takes 160.016ms. In go back n protocol the sender takes 40.016ms. So, Go Back N is more efficient than Stop and Wait protocol.

Finite state machine of Stop and Wait protocol in noisy and lossy channel:

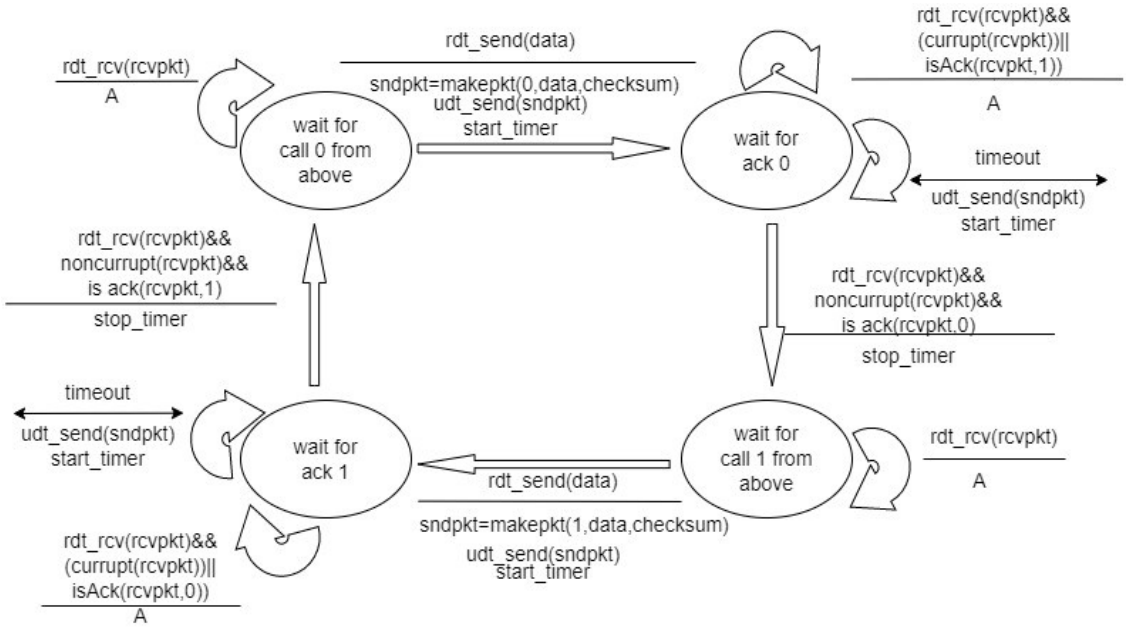


Fig-3.1 stop and wait protocol sender side finite state machine

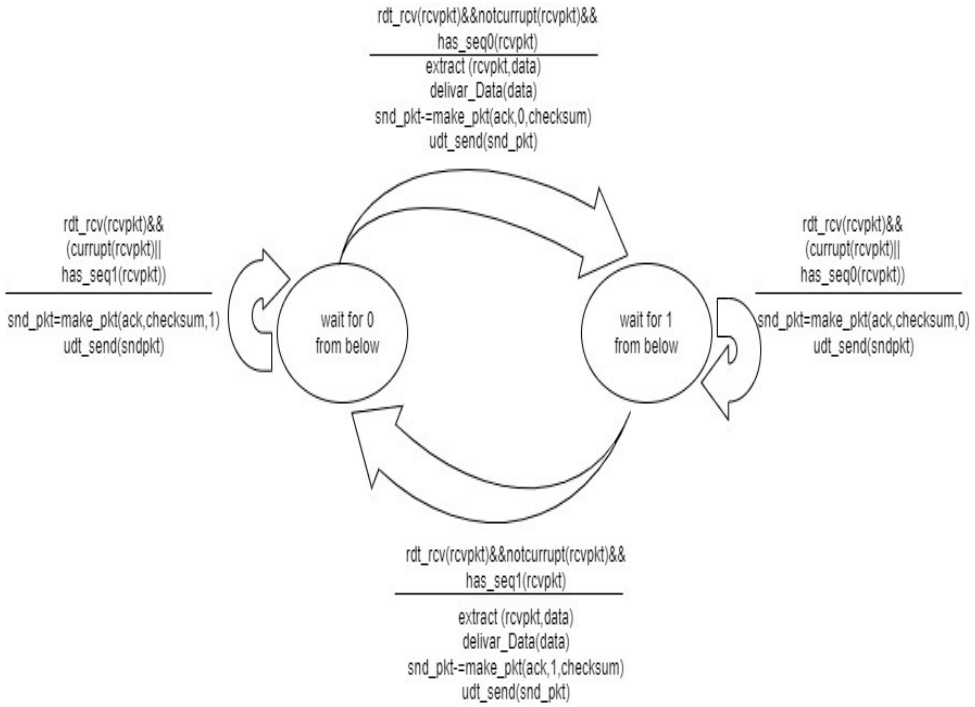


Fig-3.2 stop and wait protocol receiver side finite state machine

THE ALGORITHM FOR THE STOP AND WAIT PROTOCOL AND GO BACK N PROTOCOL IN NS2 SIMULATOR

1) Setting up new simulator
2) Setting nodes in simulator
3) Setting characteristics of nodes
4) Saving trace and nam file
5) Connection Specifics of nodes
6) Tracing the Agent
7) Starting TCP agent
8) Attaching agent to node
9) Starting TCP Sink Agent
10) Attaching sink to node
11) Starting FTP
12) Procedure to finish file
13) Connection characteristics
14) Display characteristics
15) Stop ftp.

CHAPTER 4

SOURCE CODE

NS2 CODE FOR STOP AND WAIT PROTOCOL IN IDLE CASE:

```
#Setting up new simulator
set ns [new Simulator]
#Setting nodes in simulator
set n0 [$ns node]
set n1 [$ns node]
#Setting characteristics of nodes
$ns at 0.0 "$n0 label Sender"
$ns at 0.0 "$n1 label Receiver"
#Saving trace and nam file
set nf [open stop_wait.nam w]
$ns namtrace-all $nf
set f [open stop_wait.tr w]
$ns trace-all $f
#Connection Specifics of nodes
$ns duplex-link $n0 $n1 0.2Mb 200ms DropTail
$ns duplex-link-op $n0 $n1 orient right
$ns duplex-link-op $n0 $n1 color green
$ns queue-limit $n0 $n1 10
#Tracing the Agent
Agent/TCP set nam_tracevar_ true
#Starting TCP agent
set tcp [new Agent/TCP]
$tcp set windowInit_ 1
$tcp set maxcwnd_ 1
#Attaching agent to node
```

```
$ns attach-agent $n0 $tcp
#Starting TCPSink Agent
set sink [new Agent/TCPSink]
$ns attach-agent $n1 $sink
#Attaching sink to node
$ns connect $tcp $sink
#Starting FTP
set ftp [new Application/FTP]
$ftp attach-agent $tcp
#Connection characteristics
$ns at 0.1 "$ftp start"
$ns at 5.8 "$ftp stop"
$ns at 6.2 "$ns detach-agent $n0 $tcp ; $ns detach-agent $n1 $sink"
$ns at 6.3 "finish"
#Display characteristics
$ns at 0.0 "$ns trace-annotate \"stop and wait \""
$ns at 0.1 "$ns trace-annotate \"FTP starts at 0.1\""
$ns at 0.51 "$ns trace-annotate \"Sender start sends packets Packet 0\""
$ns at 0.941 "$ns trace-annotate \"sender receives  Acknowledgement 0 \""
$ns at 0.981 "$ns trace-annotate \"Sender send  Packet 1 to receiver\""
$ns at 1.31 "$ns trace-annotate \"sender Receive Acknowledgement from receiver \""
$ns at 5.8 "$ns trace-annotate \"FTP stops\""
#Procedure to finish file
proc finish {} {
        global ns
        $ns flush-trace
#       puts "filtering..."
#       exec tclsh ../bin/namfilter.tcl Go-Back-N.nam
       puts "running nam..."
       exec nam stop_wait.nam &
```

```
    exit 0}
$ns run
```

NS2 CODE FOR STOP AND WAIT PROTOCOL IN NOISE, LOSSY CONDITION:

```
#Setting up new simulator
set ns [new Simulator]
#Setting nodes in simulator
set n0 [$ns node]
set n1 [$ns node]
#Setting characteristics of nodes
$ns at 0.0 "$n0 label Sender"
$ns at 0.0 "$n1 label Receiver"
$n0 color red
$n1 color blue
#Saving trace and nam file
set nf [open stop_wait_n.nam w]
$ns namtrace-all $nf
set f [open stop_wait_n.tr w]
$ns trace-all $f
#Connection Specifics of nodes
$ns duplex-link $n0 $n1 0.2Mb 200ms DropTail
$ns duplex-link-op $n0 $n1 orient right
$ns duplex-link-op $n0 $n1 color green
$ns queue-limit $n0 $n1 10
#Tracing the Agent
Agent/TCP set nam_tracevar_ true
#Starting TCP agent
set tcp [new Agent/TCP]
set tcp1 [new Agent/TCP]
set tcp2 [new Agent/TCP]
```

```
$tcp set windowInit_ 1
$tcp set maxcwnd_ 1
$tcp1 set windowInit_ 1
$tcp1 set maxcwnd_ 1
$tcp2 set windowInit_ 1
$tcp2 set maxcwnd_ 1
$ns attach-agent $n0 $tcp
$ns attach-agent $n0 $tcp1
$ns attach-agent $n0 $tcp2
#Starting TCPSink Agent
set sink [new Agent/TCPSink]
set sink1 [new Agent/TCPSink]
set sink2 [new Agent/TCPSink]
$ns attach-agent $n1 $sink
$ns attach-agent $n1 $sink1
$ns attach-agent $n1 $sink2
#Attaching sink to node
$ns connect $tcp $sink
$ns connect $tcp1 $sink1
$ns connect $tcp2 $sink2
#Starting FTP
set ftp [new Application/FTP]
$ftp attach-agent $tcp
set ftp1 [new Application/FTP]
$ftp1 attach-agent $tcp1
set ftp2 [new Application/FTP]
$ftp2 attach-agent $tcp2
#Connection characteristics
$ns at 0.1  "$ftp start"
$ns at 0.1  "$ftp1 start"
```

```
$ns at 0.1  "$ftp2 start"
$ns at 0.11 "$ftp1 stop"
$ns at 0.11 "$ftp2 stop"
$ns at 0.11 "$tcp set windowlnit 1"
$ns at 0.11 "$tcp set maxcwnd 1"
$ns at 0.94 "$ns queue-limit $n0 $n1 0"
$ns at 0.95 "$ns queue-limit $n0 $n1 10"
$ns at 1.1  "$ns detach-agent $n1 $tcp ; $ns detach-agent $n1 $sink"
$ns at 1.13 "$ftp1 start"
$ns at 1.13 "$tcp1 set windowlnit 1"
$ns at 1.13 "$tcp1 set maxcwnd 1"
$ns at 1.94 "$ns queue-limit $n0 $n1 0"
$ns at 2.1 "$ns queue-limit $n0 $n1 10"
$ns at 2.1  "$ns detach-agent $n1 $tcp1 ; $ns detach-agent $n1 $sink1"
$ns at 2.2 "$tcp2 set windowlnit 1"
$ns at 2.2 "$tcp2 set maxcwnd 1"
$ns at 2.2 "$ftp2 start"
$ns at 6.8 "$ftp2 stop"
$ns at 7.2 "finish"
#Display characteristics
$ns at 0.0 "$ns trace-annotate \"stop and wait\""
$ns at 0.1 "$ns trace-annotate \"FTP starts \""
$ns at 0.5 "$ns trace-annotate \" sender start Sending Packet_0 to receiver \""
$ns at 0.95 "$ns trace-annotate \" receiver receivers 1st packet\""
$ns at 0.96 "$ns trace-annotate \"Sender starts sending Packet 1 but due to noise packet got lost\""
$ns at 1.14 "$ns trace-annotate \"so sender will again send the same packets to receiver\""
$ns at 1.575 "$ns trace-annotate \"sender gets positive ack from receiver\""
$ns at 6.8 "$ns trace-annotate \"ftp stop\""
#Procedure to finish file
proc finish {} {
```

```
        global ns
        $ns flush-trace
#       puts "filtering..."
#       exec tclsh ../bin/namfilter.tcl Go-Back-N.nam
       puts "running nam..."
       exec nam stop_wait_n.nam &
        exit 0
}
$ns run
```

NS2 CODE FOR GO BACK N PROTOCOL IN IDLE CONDITION:

```
# sliding window mechanism with some features
#Setting up new simulator
set ns [new Simulator]
#Setting nodes in simulator
set n0 [$ns node]
set n1 [$ns node]
#Setting characteristics of nodes
$ns at 0.0 "$n0 label Sender"
$ns at 0.0 "$n1 label Receiver"
#Saving trace and nam file
set nf [open Go-Back-N_n.nam w]
$ns namtrace-all $nf
set f [open Go-Back-N_n.tr w]
$ns trace-all $f
#Connection Specifics of nodes
$ns duplex-link $n0 $n1 0.2Mb 200ms DropTail
$ns duplex-link-op $n0 $n1 orient right
$ns duplex-link-op $n0 $n1 color green
```

```
$ns queue-limit $n0 $n1 10
#Tracing the Agent
Agent/TCP set nam_tracevar_ true
#Starting TCP agent
set tcp [new Agent/TCP]
$tcp set windowInit_ 4
$tcp set maxcwnd_ 4
#Attaching agent to node
$ns attach-agent $n0 $tcp
#Starting TCPSink Agent
set sink [new Agent/TCPSink]
$ns attach-agent $n1 $sink
#Attaching sink to node
$ns connect $tcp $sink
#Starting FTP
set ftp [new Application/FTP]
$ftp attach-agent $tcp
#Connection characteristics
$ns at 0.1 "$ftp start"
$ns at 1.82 "$ftp stop"
$ns at 1.82 "$ns detach-agent $n0 $tcp ; $ns detach-agent $n1 $sink"
$ns at 2 "finish"
#Display characteristics
$ns at 0.0 "$ns trace-annotate \"go back n with window size 4\""
$ns at 0.1 "$ns trace-annotate \"FTP starts at 0.1\""
$ns at 0.51 "$ns trace-annotate \"Sender sends packets Packet 0,1,2,3\""
$ns at 0.94 "$ns trace-annotate \"sender receives packets Ack 0,1,2,3 \""
$ns at 1.068 "$ns trace-annotate \"Sender send  Packet 4,5,6,7 \""
$ns at 1.38 "$ns trace-annotate \"sender Receive Ack 4,5,6,7  \""
$ns at 1.39 "$ns trace-annotate \"Sender start sending  Packet 8,9,10,11\""
```

```
$ns at 1.83 "$ns trace-annotate \"sender start Receiveing  Ack 8,9,10,11 \""
$ns at 2 "$ns trace-annotate \"FTP stops\""
#Procedure to finish file
proc finish {} {
        global ns
        $ns flush-trace
#       puts "filtering..."
#       exec tclsh ../bin/namfilter.tcl Go-Back-N.nam
       puts "running nam..."
       exec nam Go-Back-N_n.nam &
        exit 0
}
$ns run
```

NS2 CODE FOR GO BACK N PROTOCOL IN NOISE, LOSSY CONDITION:

```
# sliding window mechanism with some features
#Setting up new simulator
set ns [new Simulator]
#Setting nodes in simulator
set n0 [$ns node]
set n1 [$ns node]
#Setting characteristics of nodes
$ns at 0.0 "$n0 label Sender"
$ns at 0.0 "$n1 label Receiver"
$n0 color red
$n1 color blue
#Saving trace and nam file
set nf [open Go-Back-N.nam w]
$ns namtrace-all $nf
```

```
set f [open Go-Back-N.tr w]
$ns trace-all $f
#Connection Specifics of nodes
$ns duplex-link $n0 $n1 0.2Mb 200ms DropTail
$ns duplex-link-op $n0 $n1 orient right
$ns duplex-link-op $n0 $n1 color green
$ns queue-limit $n0 $n1 10
#Tracing the Agent
Agent/TCP set nam_tracevar_ true
#Starting TCP agent
set tcp [new Agent/TCP]
set tcp1 [new Agent/TCP]
set tcp2 [new Agent/TCP]
set tcp3 [new Agent/TCP]
set tcp4 [new Agent/TCP]
set tcp5 [new Agent/TCP]
$tcp set windowInit_ 4
$tcp set maxcwnd_ 4
$tcp1 set windowInit_ 4
$tcp1 set maxcwnd_ 4
$tcp2 set windowInit_ 4
$tcp2 set maxcwnd_ 4
#Attaching agent to node
$ns attach-agent $n0 $tcp
$ns attach-agent $n0 $tcp1
$ns attach-agent $n0 $tcp2
$ns attach-agent $n0 $tcp3
$ns attach-agent $n0 $tcp4
$ns attach-agent $n0 $tcp5
#Starting TCPSink Agent
```

```
set sink [new Agent/TCPSink]
set sink1 [new Agent/TCPSink]
set sink2 [new Agent/TCPSink]
set sink3 [new Agent/TCPSink]
set sink4 [new Agent/TCPSink]
set sink5 [new Agent/TCPSink]
$ns attach-agent $n1 $sink
$ns attach-agent $n1 $sink1
$ns attach-agent $n1 $sink2
$ns attach-agent $n1 $sink3
$ns attach-agent $n1 $sink4
$ns attach-agent $n1 $sink5
#Attaching sink to node
$ns connect $tcp $sink
$ns connect $tcp1 $sink1
$ns connect $tcp2 $sink2
$ns connect $tcp3 $sink3
$ns connect $tcp4 $sink4
$ns connect $tcp5 $sink5
#Starting FTP
set ftp [new Application/FTP]
$ftp attach-agent $tcp
set ftp1 [new Application/FTP]
$ftp1 attach-agent $tcp1
set ftp2 [new Application/FTP]
$ftp2 attach-agent $tcp2
set ftp3 [new Application/FTP]
$ftp3 attach-agent $tcp3
set ftp4 [new Application/FTP]
$ftp4 attach-agent $tcp4
```

```
set ftp5 [new Application/FTP]
$ftp5 attach-agent $tcp5
#Connection characteristics
$ns at 0.1  "$ftp start"
$ns at 0.1  "$ftp3 start"
$ns at 0.1  "$ftp1 start"
$ns at 0.1  "$ftp2 start"
$ns at 0.1 "$ftp4 start"
$ns at 0.1 "$ftp5 start"
$ns at 0.11 "$ftp1 stop"
$ns at 0.11 "$ftp2 stop"
$ns at 0.11 "$ftp3 stop"
$ns at 0.11 "$ftp4 stop"
$ns at 0.11 "$ftp5 stop"
$ns at 0.11 "$tcp set windowlnit 4"
$ns at 0.11 "$tcp set maxcwnd 4"
$ns at 0.94 "$ns queue-limit $n0 $n1 0"
$ns at 0.95 "$ns queue-limit $n0 $n1 10"
$ns at 1.1  "$ns detach-agent $n1 $tcp ; $ns detach-agent $n1 $sink"
$ns at 1.37 "$ftp1 start"
$ns at 1.37 "$tcp1 set windowlnit 4"
$ns at 1.37 "$tcp1 set maxcwnd 4"
$ns at 1.81 "$ns queue-limit $n0 $n1 0"
$ns at 1.89 "$ns queue-limit $n0 $n1 10"
$ns at 1.94  "$ns detach-agent $n1 $tcp1 ; $ns detach-agent $n1 $sink1"
$ns at 2 "$tcp2 set windowlnit 4"
$ns at 2 "$tcp2 set maxcwnd 4"
$ns at 2.3 "$ftp2 start"
$ns at 2.32 "$ftp2 stop"
$ns at 2.33 "$ftp3 start"
```

```
$ns at 2.35 "$ftp3 stop"

$ns at 2.36 "$ftp4 start"

$ns at 2.37 "$ftp4 stop"

$ns at 2.38 "$ftp5 start"

$ns at 2.39 "$ftp5 stop"

$ns at 3 "finish"

#Display characteristics

$ns at 0.0 "$ns trace-annotate \"Sliding Window with window size 4 (noisy and lossy
environment)\""

$ns at 0.1 "$ns trace-annotate \"FTP starts and send message to receiver if the receiver is ready to
accept the messages\""

$ns at 0.5 "$ns trace-annotate \" sender start  Sending  Packet_0,1,2,3 to receiver \""

$ns at 0.94 "$ns trace-annotate \"sender start receiving Acknowledgement 0,1,2,3\""

$ns at 0.955 "$ns trace-annotate \"Sender starts sending Packet_4,5,6,7 but due to noise 1sr packet(4)
got lost\""

$ns at 1.19 "$ns trace-annotate \"Receive receives only 5th 6th 7th packets \""

$ns at 1.575 "$ns trace-annotate \"receiver start receiving packets 4,5,6,7\""

$ns at 1.81 "$ns trace-annotate \"sender starts getting ack 4,5,6,7 from receiver\""

$ns at 1.83 "$ns trace-annotate \"sender sends packets 8,9,10,11 but due to noise packet 8,9 got
lost\""

$ns at 2.1 "$ns trace-annotate \"receiver receives only 2 packets 10,11\""

$ns at 2.5 "$ns trace-annotate \"receiver start receiving packets 8,9,10,11 from sender\""

$ns at 2.9 "$ns trace-annotate \"ftp stop\""

#Procedure to finish file

proc finish {} {

        global ns

        $ns flush-trace

#       puts "filtering..."

#       exec tclsh ../bin/namfilter.tcl Go-Back-N.nam

       puts "running nam..."

       exec nam Go-Back-N.nam &

        exit 0
```

}

$ns run

CODE FOR THROUGHPUT CALCULATION:

```
BEGIN {
recvdSize = 0
startTime = 0.5
stopTime = 7.0}
{
event = $1
time = $2
node_id = $3
pkt_size = $6
level = $4
# Store start time
if (event == "s") {
if (time < startTime) {
startTime = time}}
# Update total received packets' size and store packets arrival time
if (event == "r"){
if (time > stopTime) {
stopTime = time}
recvdSize += pkt_size}}
END {
printf("Average Throughput[kbps] = %.2f\n StartTime=%.2f\nStopTime=%.2f\n",(recvdSize/(stopTime-startTime))*(8/1000),startTime,stopTime);}
```

CODE FOR CALCULATING PACKET LOSS RATE:

```
BEGIN {
receive=0
drop=0
total=0
ratio1=0.0
ratio2=0.0}
```

```
{
if ($1=="r" && $4==3){
receive++;
}
if ($1=="d"){
drop++;}}
END {
total=receive+drop+1
ratio1=(receive/total)
ratio2=(drop/total)
printf("Total packet sent = %d\n",total)
printf("Packets received = %d\n",receive)
printf("Packets dropped = %d\n",drop)
printf("Packet delivery rate = %f\n",ratio1)
printf("Packet loss rate = %f\n",ratio2)}
```

CHAPTER 5

RESULTS

OUTPUT OF STOP AND WAIT IN IDLE CASE:

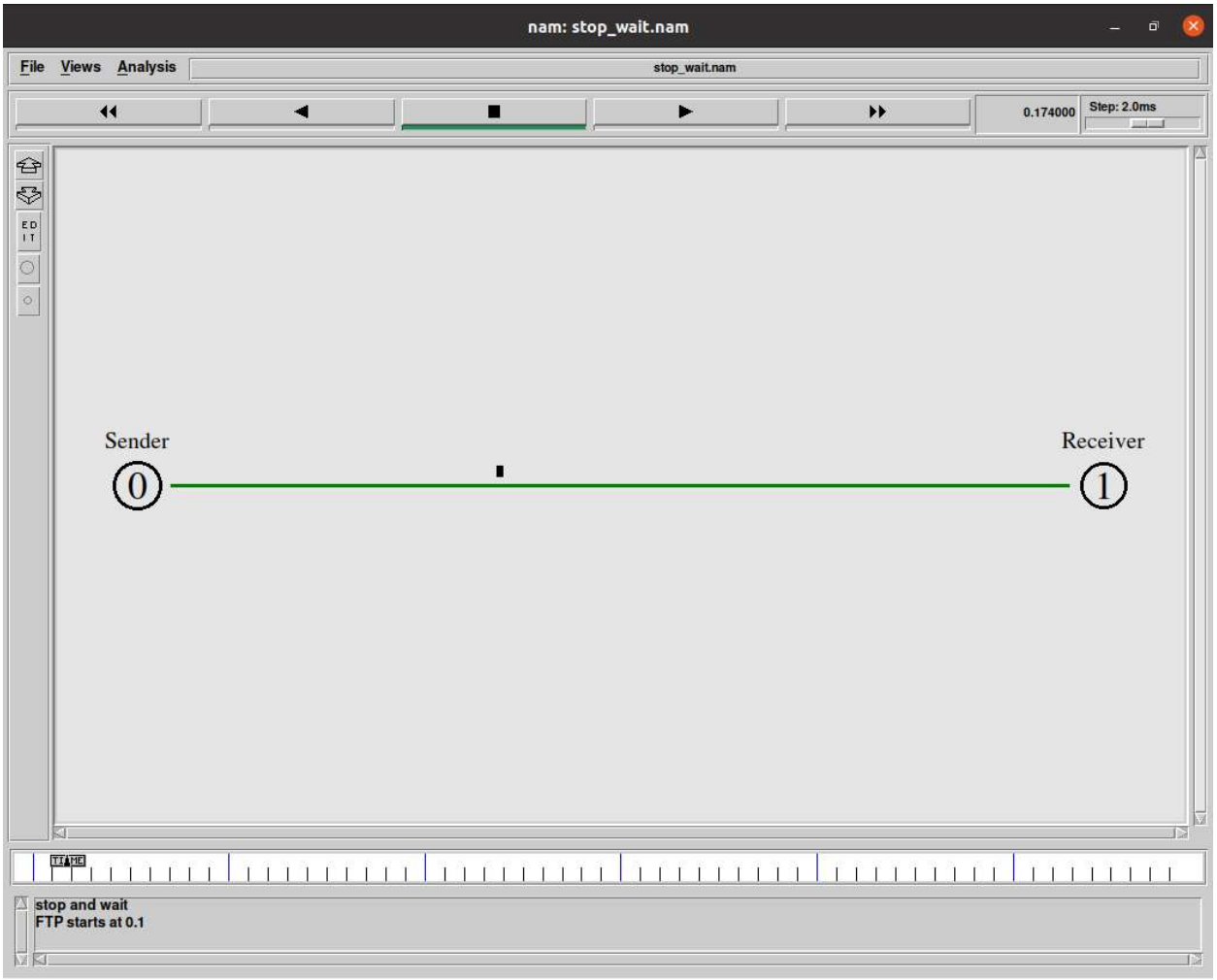


Fig-5.1- Sending a message to receiver to know if receiver is free to take data packets.

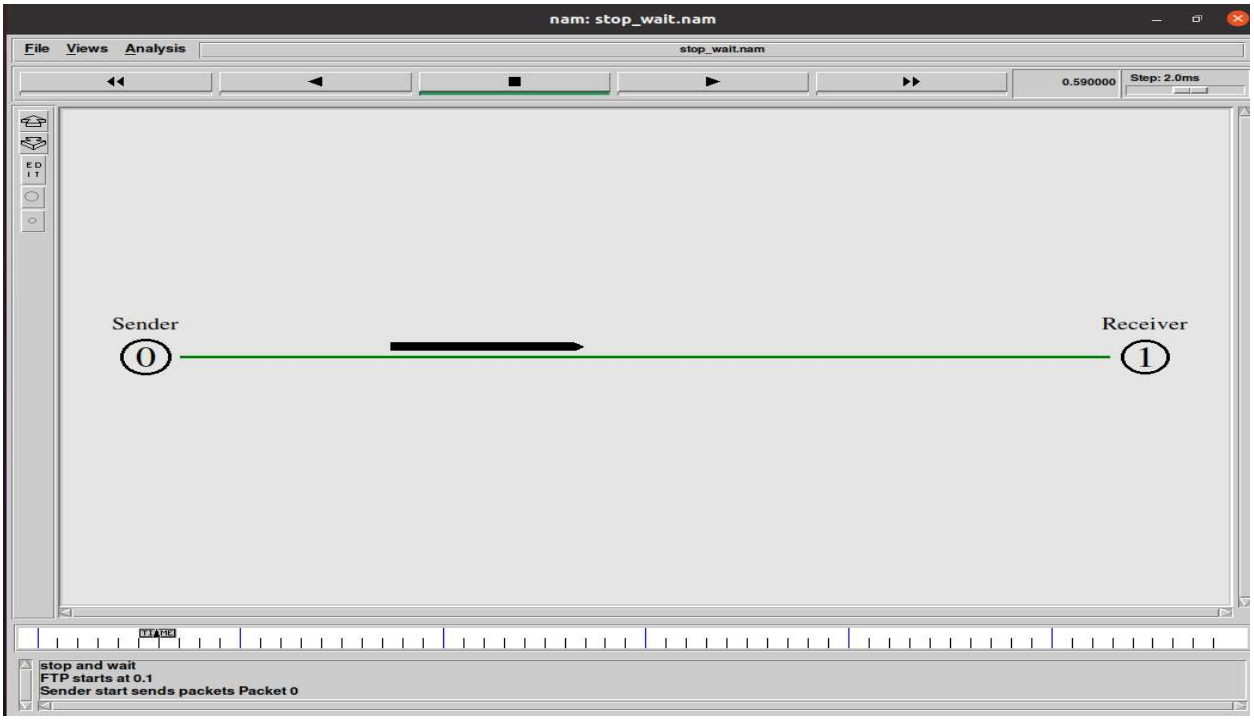


Fig 5.2- sender send 1st packet to receiver

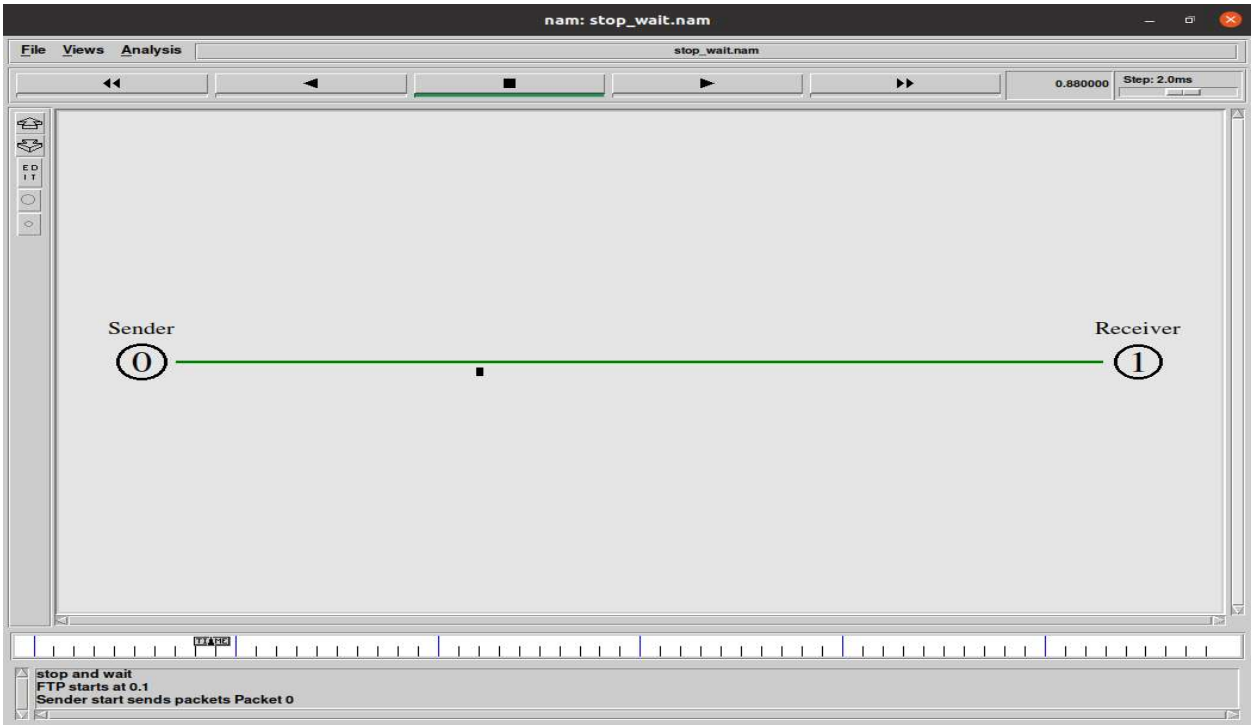


Fig 5.3- receiver send a positive ack to sender.

OUTPUT FOR STOP AND WAIT PROTOCOL IN NOISY, LOSSY CONDITION:

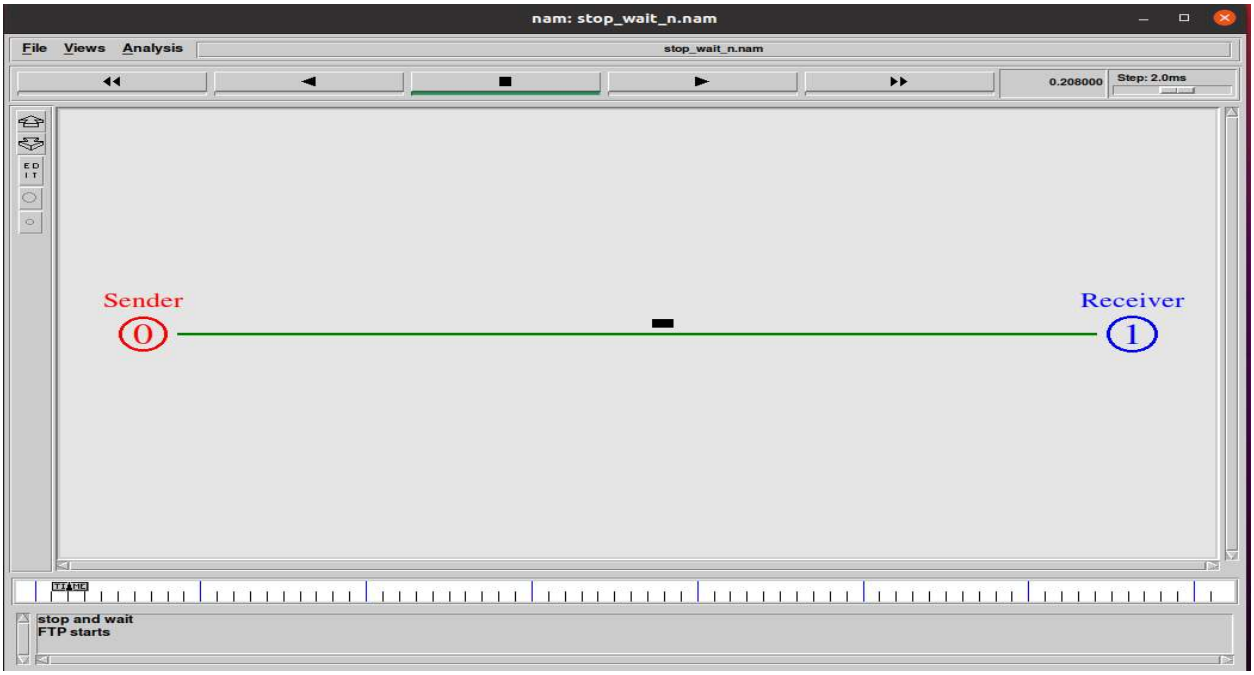


Fig 5.4- sender send a message to receiver to know if the receiver is ready to take data packets

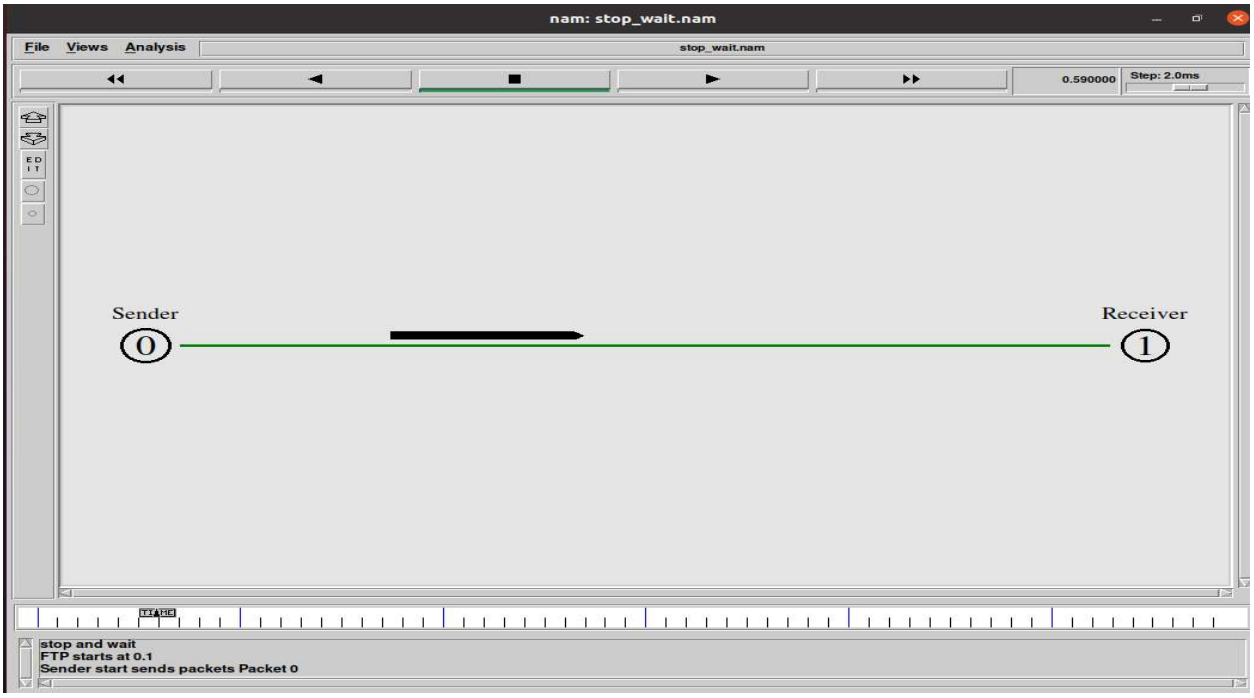


Fig 5.5- sender send a data packet to receiver

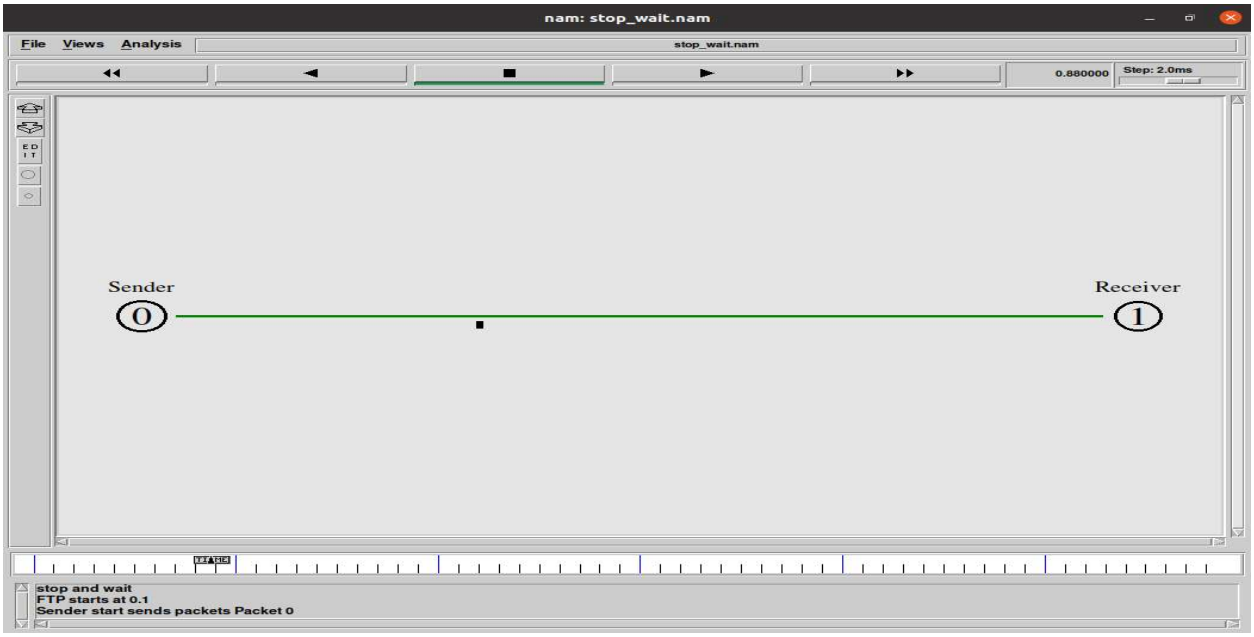


Fig 5.6- receiver send a positive ack to sender

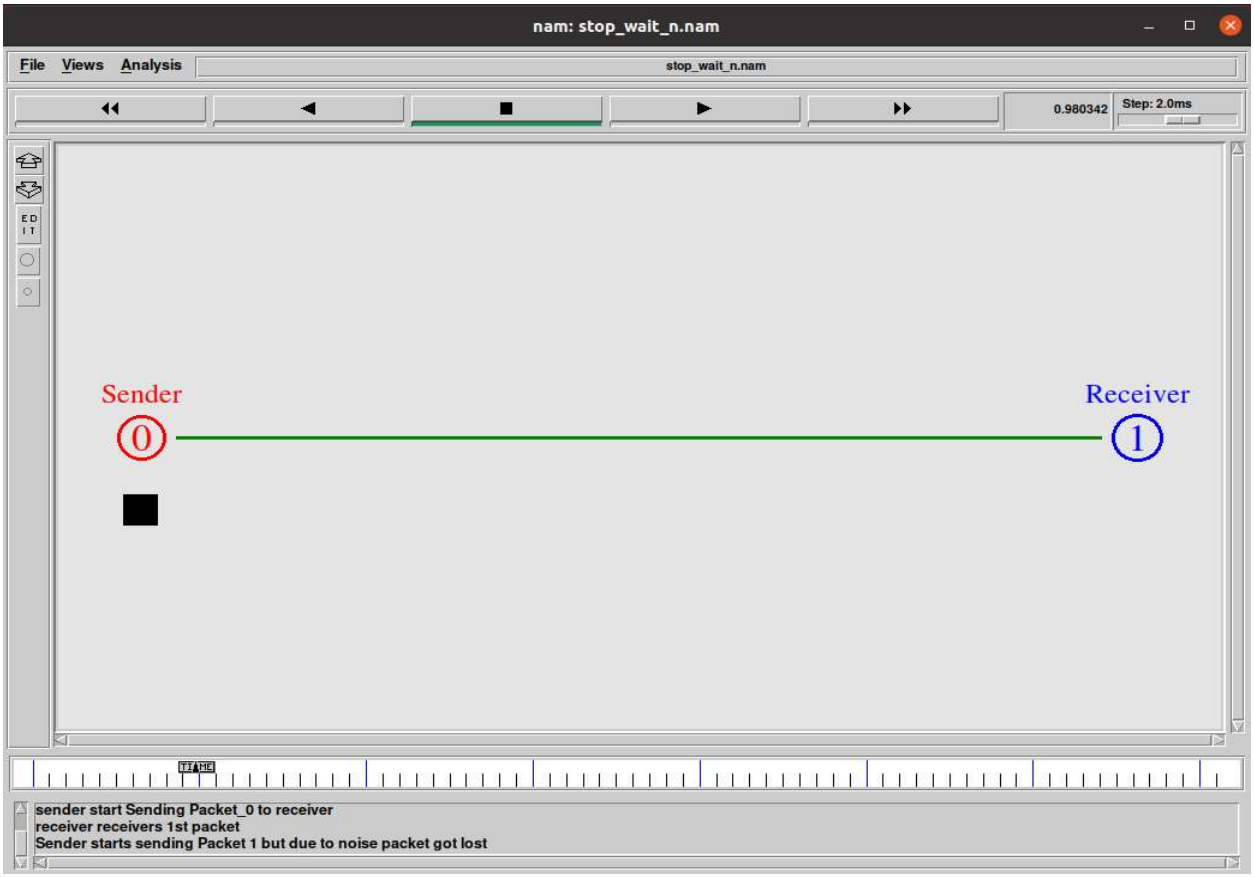


Fig 5.7- a packet is lost due to noise.

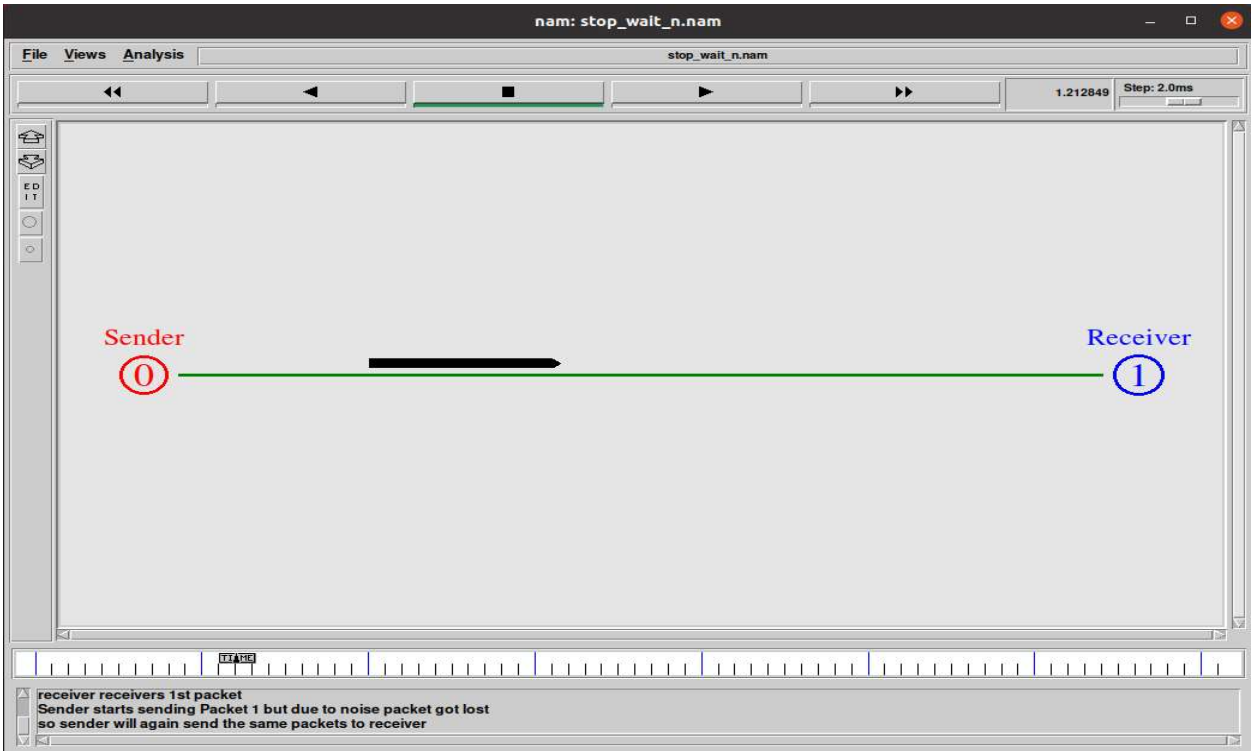


Fig 5.8- sender resend the same packet to receiver

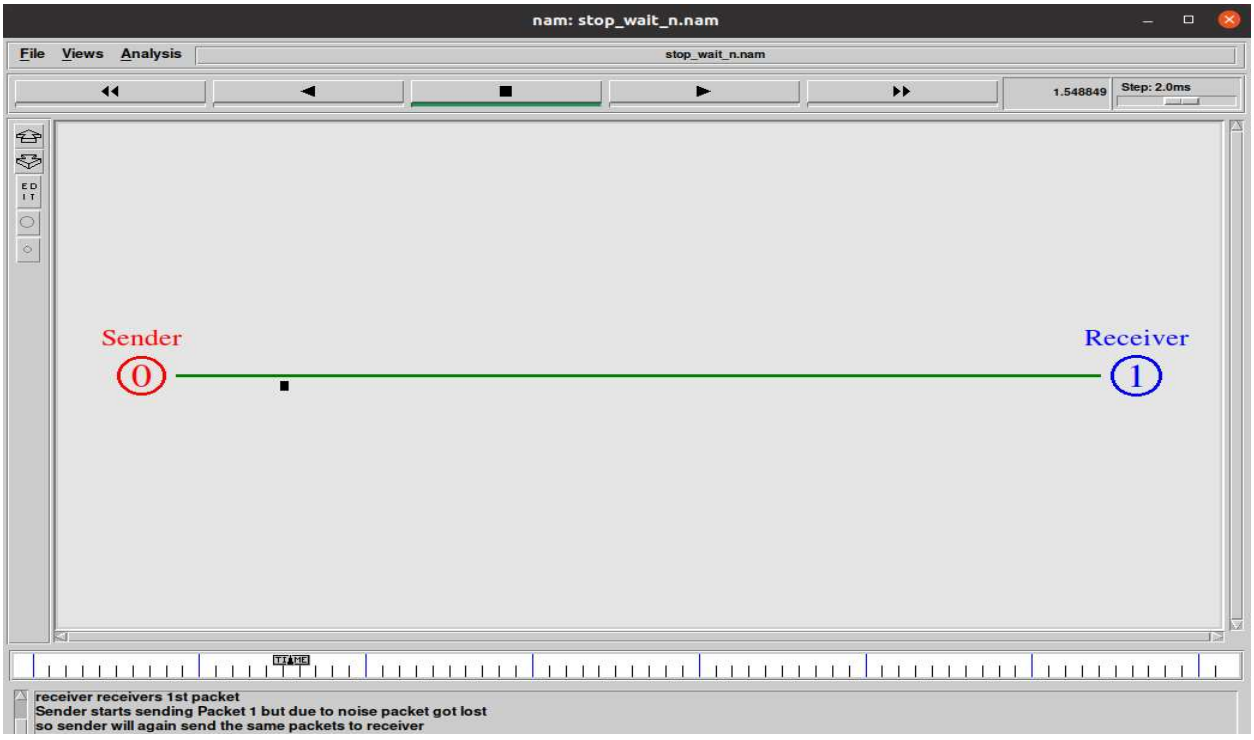


Fig 5.9- receiver send positive ack to sender

OUTPUT FOR GO BACK N PROTOCOL IN IDLE CASE:

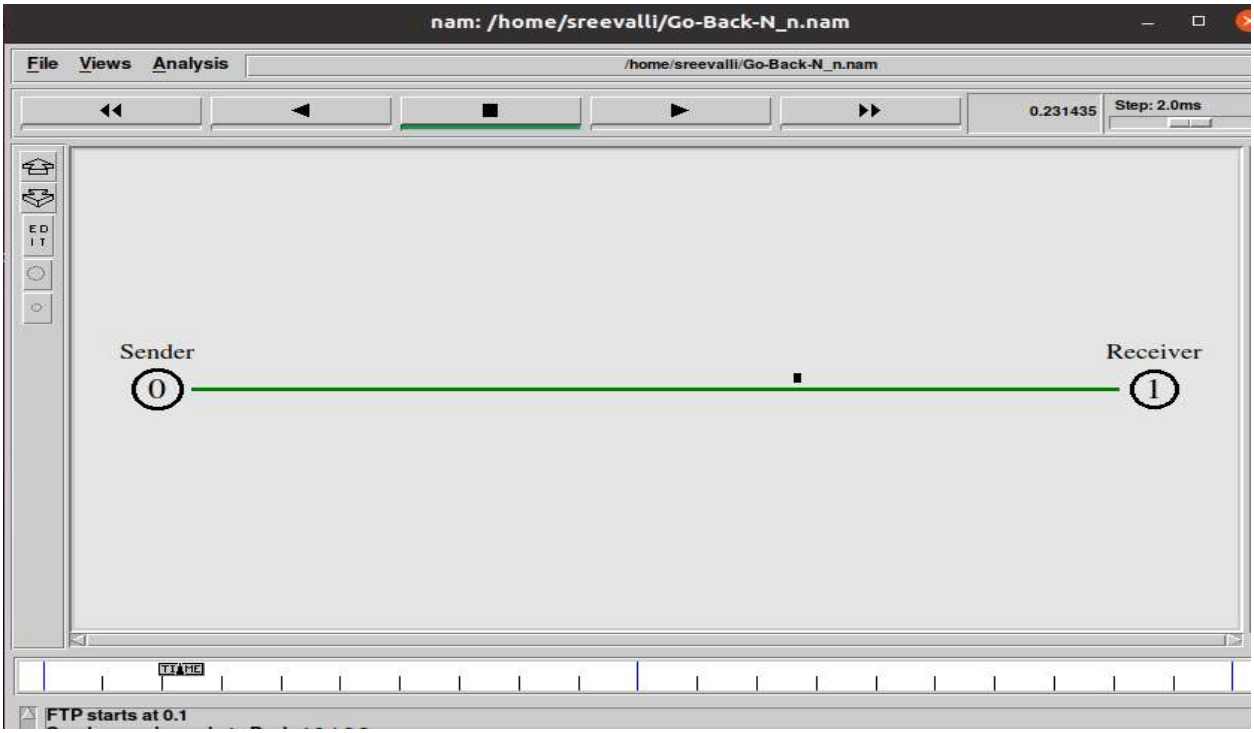


Fig 5.10- Sending a message to receiver to know if receiver is free to take data packets.

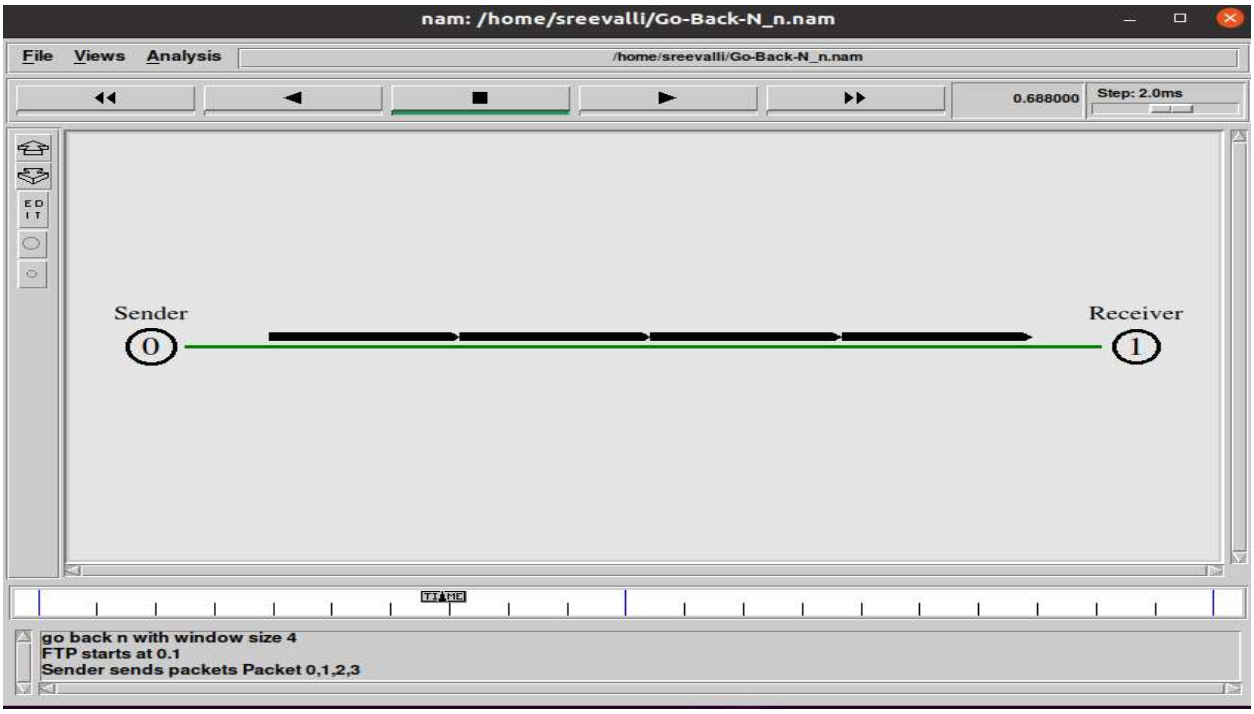


Fig 5.11- sender send 4 packets to receiver

Fig 5.12- receiver send 4 positive ack to sender

OUTPUT FOR GO BACK N NOSSY, LOSSY CHANNEL:

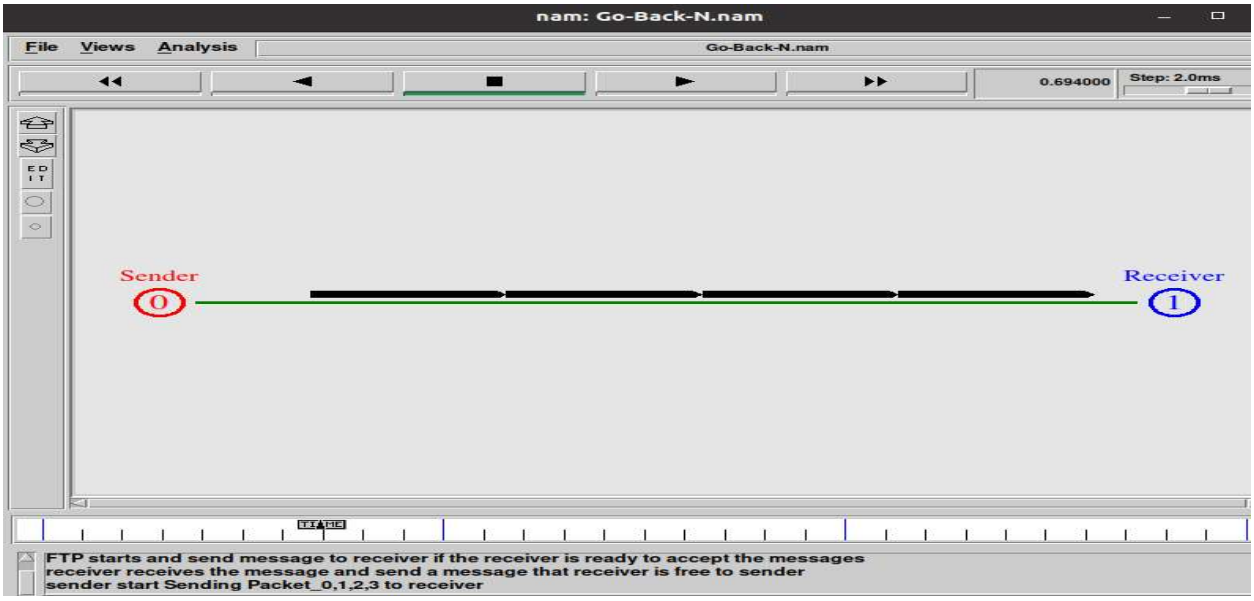


Fig 5.13- send 4 packets to receiver

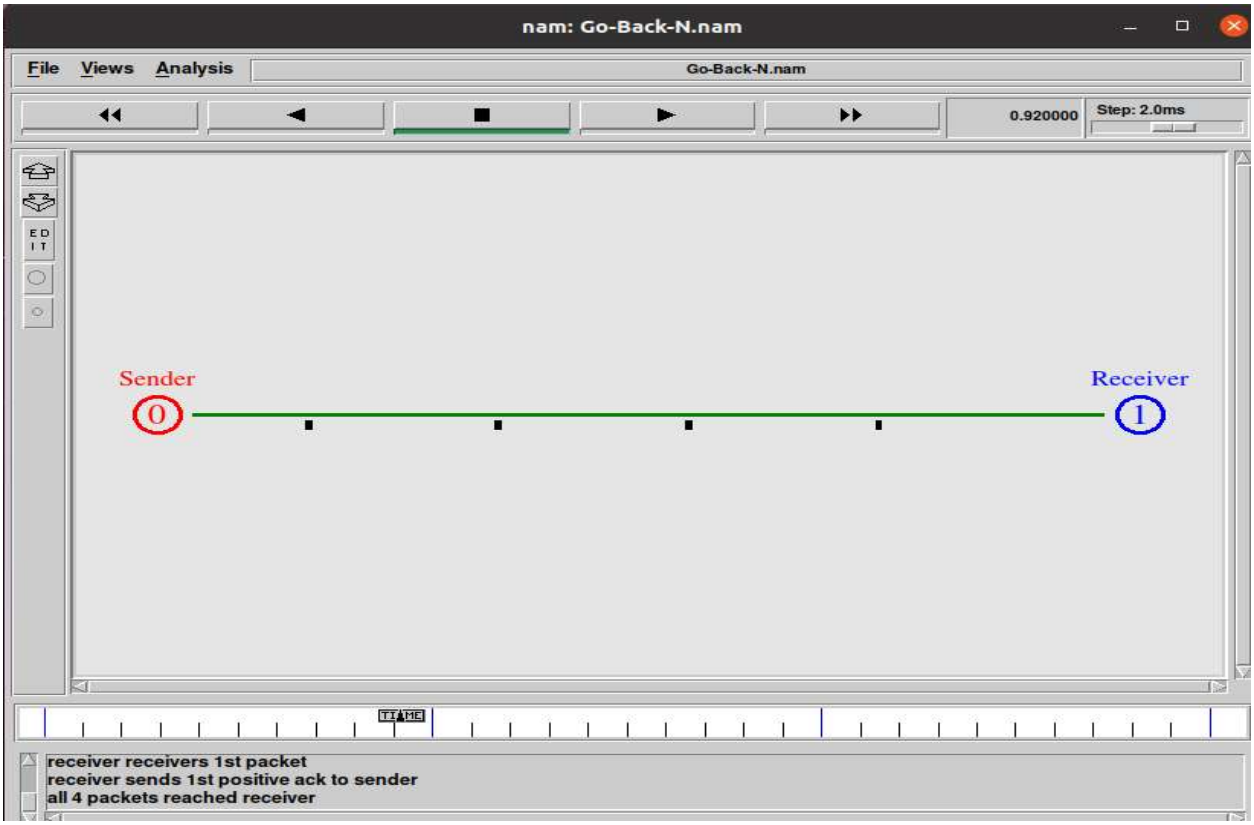


Fig 5.14- receiver send 4 positive ack to sender

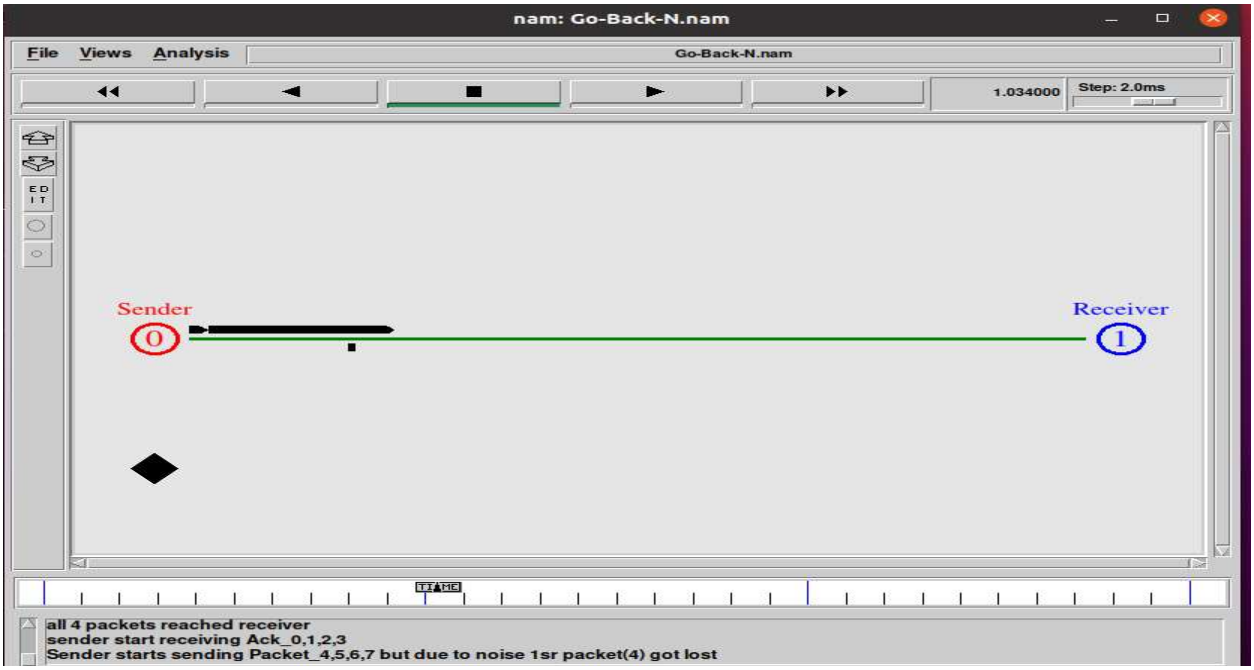


Fig 5.15- one packet is lost due to noise. So, all 4 ack's will not be send

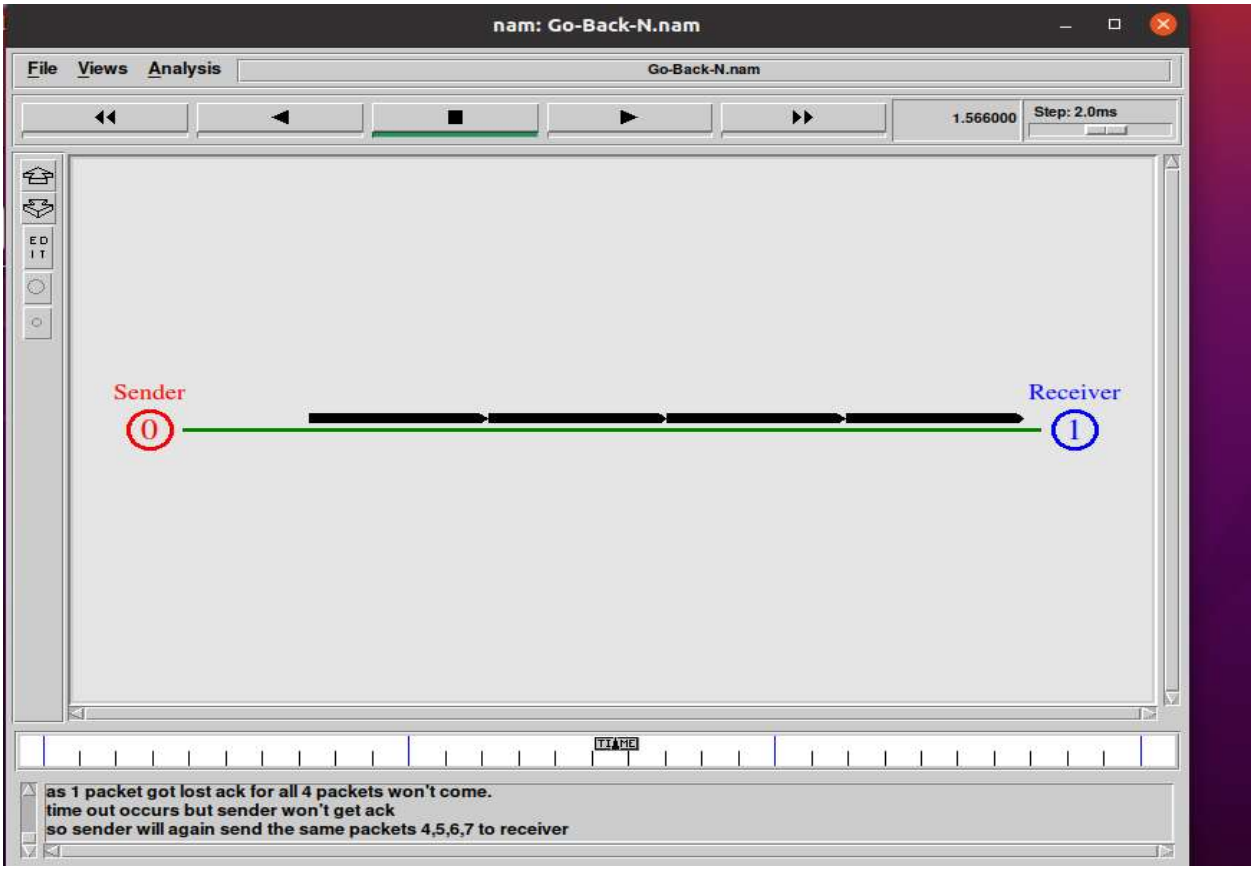


Fig 5.16- sender again send all 4 same packets to receiver

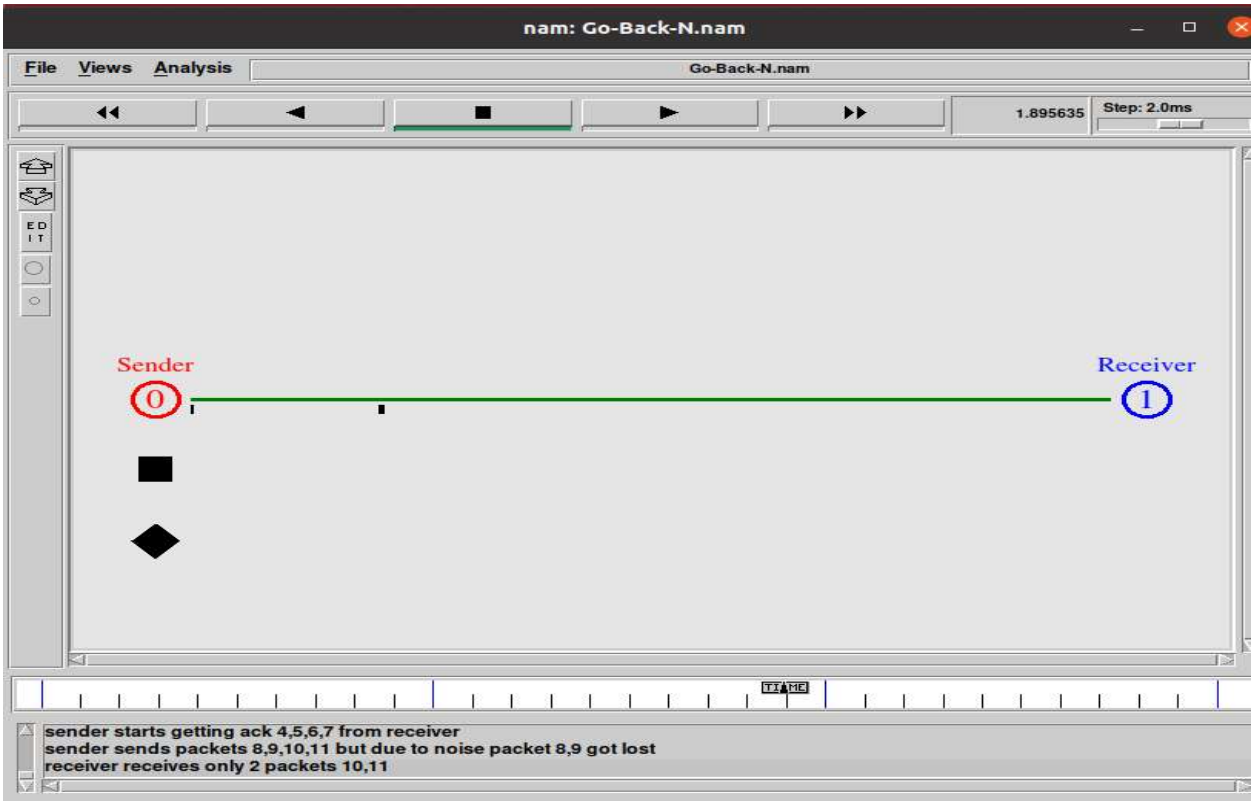


Fig 5.17- 2 packets got lost, so 4 ack's won't be send

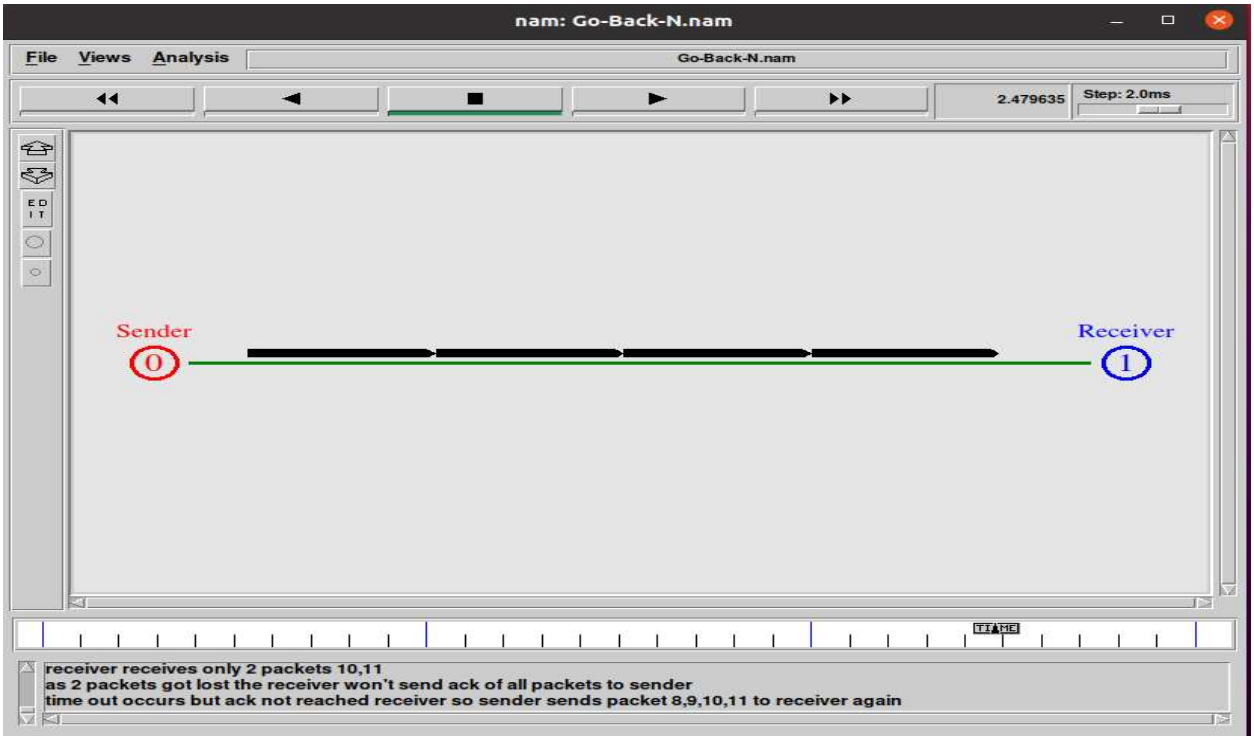
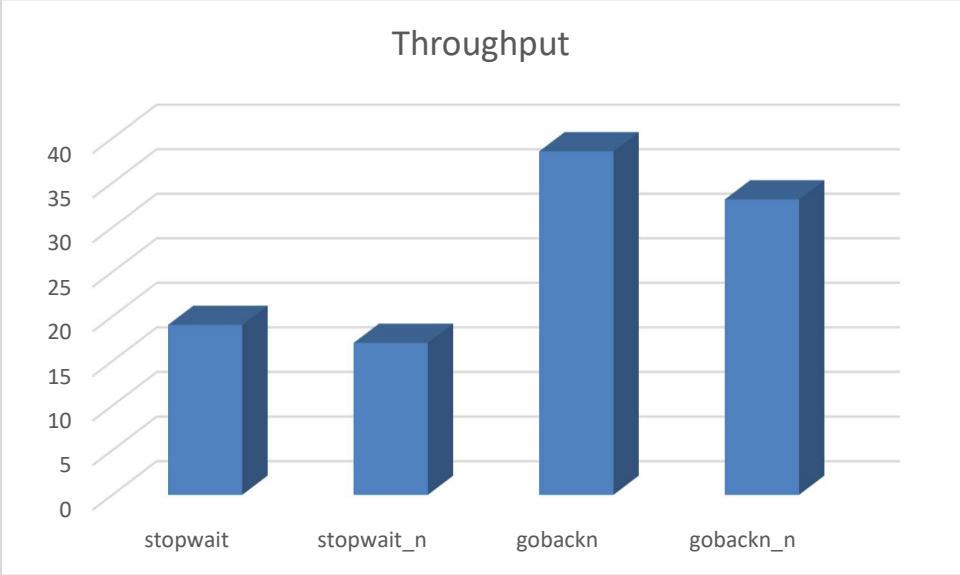


Fig 5.18- sender again send all 4 packets to receiver

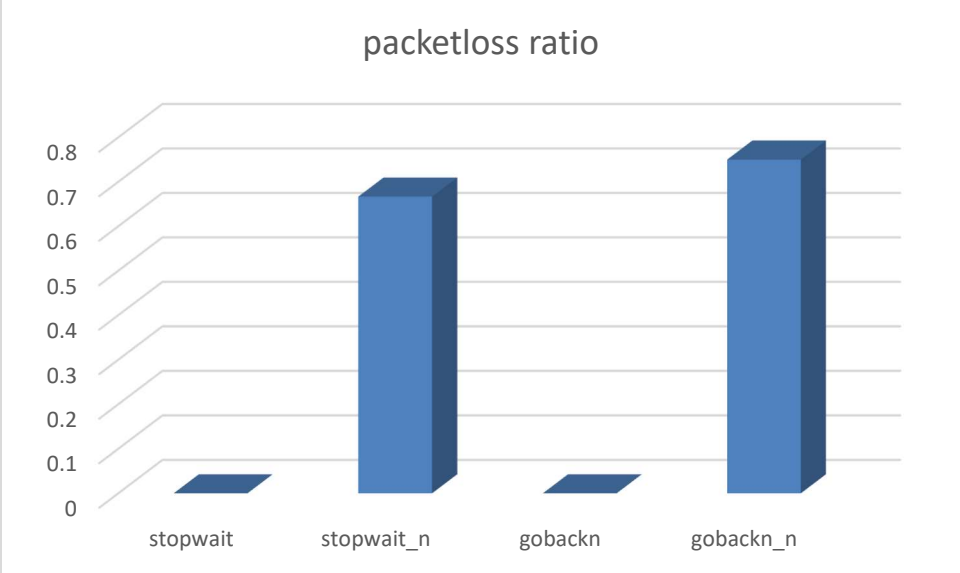
CHAPTER 6

PERFORMANCE EVALUATION

COMPARISON OF THROUGHPUT FOR ALL 4 CASES:



COMPARSION OF PACKETLOSS RATE IN ALL 4 CASES:



CHAPTER 7

CONCLUSION

In data transfer between sender and receiver, to get reliable data transfer we use stop and wait arq protocol and go back n protocol. We also have selective repeat protocol which is extension to go back n protocol which has higher efficiency than Go Back N arq protocol. In this we compared stop and wait protocol and go back n protocol, in which we get to know go back n has higher efficiency and throughput. Nowadays, the tcp in transport layers using hybrid version of ARQ, to be more precise a mixture of both Go Back N and selective repeat. Here, stop and wait protocol is useful for short range distance and go back n is used for long range distance. To improvise Go Back N throughput I have a small idea, we will see in future work.

CHAPTER 8

FUTURE WORK

We know go back n is more suitable for wide range distances. So, if any error occurred the sender should send same data again to receiver which will be an added delay. And, also as it is wide range to send the same data, it should travel the same distance once more. To be precise if there is no error, it should travel RTT period of time. If there is error the same packet take 2*RTT period of time. As it is wide range the distance will be in kilometers. So, to eliminate this extra delay to some extent we can use a proxy server (Temporary storage) in between sender and receiver. When sender send data to receiver, the same data will also be sent to proxy server, if the receiver send positive ack, both the sender and proxy will remove that packet from window, if there is an error, without sending negative ack to sender, the receiver will send ack to proxy server, as the proxy server will be in between or nearer to receiver, it won't take long time to retransmit the same data, means it will take less than 2*RTT period of time. In this way we can reduce delay.

But the disadvantage is as we are keeping a proxy server it will be of extra cost. The proxy also should have size of window.

CHAPTER 9

REFERENCES

(Busquets,2021) Busquets González, M. (2021). *Software-defined implementation and practical evaluation of ARQ schemes over Visible Light Communication* (Master's thesis, Universitat Politècnica de Catalunya).

(Cipriano, 2010) Cipriano, A. M., Gagneur, P., Vivier, G., & Sezginer, S. (2010, September). Overview of ARQ and HARQ in beyond 3G systems. In *2010 IEEE 21st International Symposium on Personal, Indoor and Mobile Radio Communications Workshops* (pp. 424-429). IEEE.

(Karetsi, 2022) Karetsi, F., & Papapetrou, E. (2022). Lightweight network-coded ARQ: An approach for Ultra-Reliable Low Latency Communication. *Computer Communications*

(Li, 2018) Li, Q., & Chen, C. X. (2018, October). A hybrid ARQ protocol for the communication system with multiple channels. In *2018 International Conference on Information and Communication Technology Convergence (ICTC)* (pp. 222-227). IEEE.

(Makki,2014) Makki, B., Svensson, T., & Zorzi, M. (2014, December). Green communication via Type-I ARQ: Finite block-length analysis. In *2014 IEEE Global Communications Conference* (pp. 2673-2677). IEEE.

(Mo,2012) Mo, H., Mingir, A. C., Alhumyani, H., Albayram, Y., & Cui, J. H. (2012, October). Uw-harq: An underwater hybrid arq scheme: Design, implementation and initial test. In *2012 Oceans* (pp. 1-5). IEEE.

(Nada, 2020) Nada, F. (2020). Performance analysis of go-back-N ARQ protocol used in data transmission over noisy channels. *Adv. Sci. Technol. Eng. Syst. J*, *5*(4), 612-617.

(Nada, 2021) Nada, F. A. (2021, March). Service Time Analysis of Go-Back-N ARQ Protocol. In *International Conference on Advanced Machine Learning Technologies and Applications* (pp. 559-569). Springer, Cham.

(Vasiliev, 2019) Vasiliev, D., Chunaev, A., Abilov, A., Kaysina, I., & Meitis, D. (2019, November). Application layer arq and network coding for qos improving in uav-assisted networks. In *2019 25th Conference of Open Innovations Association (FRUCT)* (pp. 353-360). IEEE.

http://www2.ic.uff.br/~michael/kr1999/3-transport/3_040-principles_rdt.htm

https://www.studytonight.com/computer-networks/gobackn-automatic-repeat-request

https://www.studytonight.com/computer-networks/gobackn-automatic-repeat-request

https://www.astesj.com/publications/ASTESJ_050472.pdf