Ramakrishna Mission
Vivekananda Centenary College, Rahara

Department of Computer Science

Registration No.: A01-1112-117-014-2021 of 2021-2022

Roll No.: 715

Semester: 3rd Semester

Paper: CC-VII

Subject: Computer Networking

Examination Roll No.:

Session: 2022-2023

Index

Problem 1: Simulation of cyclic redundancy check (CRC) error detection algorithm for noisy channel

Aim:

To implement error detection and error correction techniques.

Hardware/Software Requirements:

Turbo C.

Theory:

The upper layers work on some generalised view of network architecture and are not aware of actual hardware data processing. Hence, the upper layers expect error-free transmission between the systems. Most of the applications would not function expectedly if they receive erroneous data. Applications such as voice and video may not be that affected and with some errors they may still function well. Data-link layer uses some error control mechanism to ensure that frames (data bit streams) are transmitted with a certain level of accuracy. But to understand how errors are controlled, it is essential to know what types of errors may occur. CRC is a different approach to detect if the received frame contains valid data. This technique involves binary division of the data bits being sent. The divisor is generated using polynomials. The sender performs a division operation on the bits being sent and calculates the remainder. Before sending the actual bits, the sender adds the remainder at the end of the actual bits. Actual data bits plus the remainder is called a codeword. The sender transmits data bits as codewords.

Algorithm:

1. Open Turbo C++ software and type the program for error detection.
2. Get the input in the form of bits.
3. Append 16 zeros as redundancy bits.
4. Divide the append data using a divisor polynomial.
5. The resulting data should be transmitted to the receiver.
6. At the receiver the received data is entered.
7. The same process is repeated at the receiver.
8. If the remainder is zero there is no error otherwise some error in the received bits.
9. Run the program.

C Program:

```
#include <stdio.h>
char m[50], g[50], r[50], q[50],
temp[50];
void caltrans (int);
void crc (int);
void calram ();
void shift ();
int main (){
int n,i = 0;
char ch, flag=0;
printf("Enter the frame bits");
 while((ch=getc(stdin))!='\n')
m[i++]=ch;
n=i;
for(i=0;i<n-16;i++){
if(r[0]=='1') {
q[i]='1';
calram(); }
else {
q[i]='0';
shiftl(); }
r[16]=m[17+i];
r[17]='\0';
printf("\nremainder
%d:%s",i+1,r);
for(j=0;j<=17;j++)
temp[j]=r[j]; }
q[n-16]='\0'; }
```

```
for(i = 0;i<=16;i++)
 g[i]='0';

g[0]=g[4]=g[11]=g[16]='1';g[17]='\0';
printf("\ngenerator:%s\n",g);
crc(n);
 printf("\n\nquotient:%s",q);
caltrans(n);
 printf("\ntransmitted frame:%s",m);
 printf("\nEnter transmitted
frame:");
 scanf("\n%s",m);
 printf("CRC checking\n");
 crc(n);
 printf("\n\nlast remainder:%s",r);
for(i=0;i<16;i++)
if(r[i]!='0')
flag=1;
else
continue;
if(flag==1)
 printf("Error during transmission");
else
printf("\n\nReceived frame is
correct"); }
void crc(int n) {
int i,j;
for(i=0;i<n;i++)
temp[i]=m[i];
for(i=0;i<16;i++)
r[i]=m[i];
printf("\nintermediate remainder\n");
void calram() {
int i,j;
for(i=1;i<=16;i++)
r[i-1]=((int)temp[i]-
48)^((int)g[i]-48)+48; }
void shiftl() {
int i;
for(i=1;i<=16;i++)
r[i-1]=r[i]; }
void caltrans(int n) {
int i,k=0;
for(i=n-16;i<n;i++)
m[i]=((int)m[i]-
48)^((int)r[k++]-48)+48;
m[i]='\0'; }
```

Output:

```
Enter the Frame Bits:
1011
The msg after appending 16 zeros:
10110000000000000000
The Transmitted frame is:
10110110000101101011
Enter the transmitted Frame
10110110000101101011
Received msg:
10110110000101101011
The Remainder is:
0000000000000000
Received frame is correct.
```

Problem 2: Simulation and implementation of Stop and Wait protocol for noisy channel

Aim:

To simulate and to study Stop and Wait Protocol.

Software Requirements:

NS-2 Simulator

Theory:

Stop and Wait is a reliable transmission flow control protocol. This protocol works only in Connection Oriented (Point to Point) Transmission. The Source node has a window size of ONE. After transmission of a frame the transmitting (Source) node waits for an acknowledgement from the destination node. If the transmitted frame reaches the destination without error, the destination transmits a positive acknowledgement. If the transmitted frame reaches the Destination with error, the receiver destination does not transmit an acknowledgement. If the transmitter receives a positive acknowledgement, it transmits the next frame if any. Else if its acknowledgement receive timer expires, it retransmits the same frame.

1. Start with the window size of 1 from the transmitting (Source) node.
2. After transmission of a frame the transmitting (Source) node waits for a reply (Acknowledgement) from the receiving (Destination) node.
3. If the transmitted frame reaches the receiver (Destination) without error, the receiver (Destination) transmits a Positive Acknowledgement.
4. If the transmitted frame reaches the receiver (Destination) with error, the receiver (Destination) does not transmit acknowledgement.
5. If the transmitter receives a positive acknowledgement, it transmits the next frame if any. Else if the transmission timer expires, it retransmits the same frame again.
6. If the transmitted acknowledgment reaches the Transmitter (Destination) without error, the Transmitter (Destination) transmits the next frame if any.
7. If the transmitted frame reaches the Transmitter (Destination) with error, the Transmitter (Destination) transmits the same frame.
8. This concept of the Transmitting (Source) node waiting after transmission for a reply from the receiver is known as STOP and WAIT.

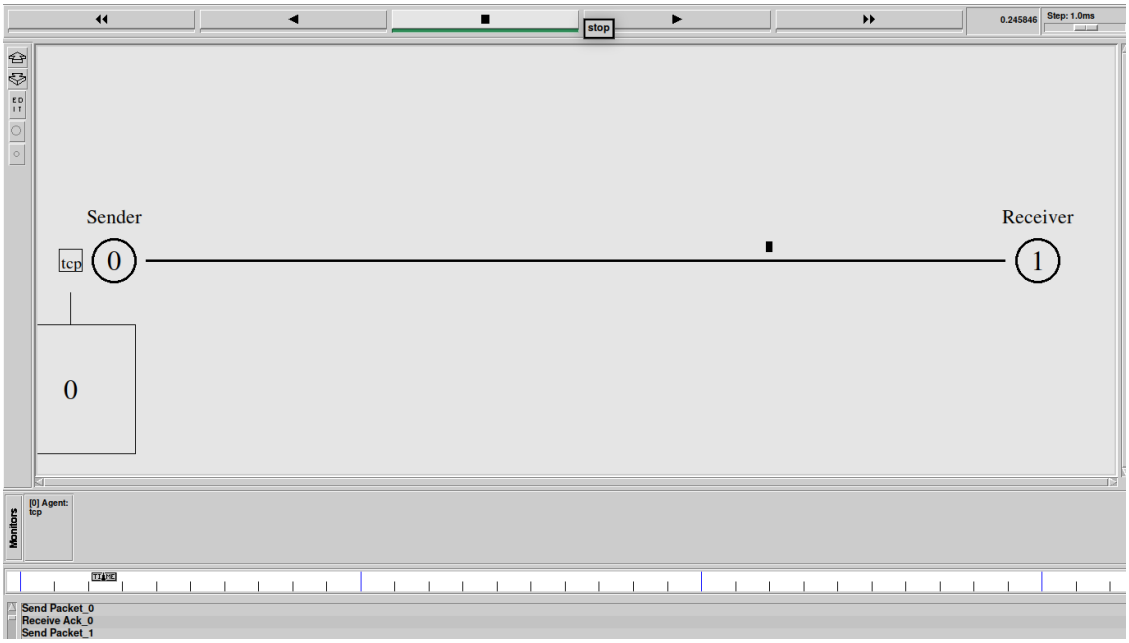
Algorithm:

1. Create a simulator object
2. Define different colours for different data flows
3. Open a nam trace file and define the finish procedure then close the trace file, and execute nam on trace file.
4. Create two nodes that forms a network numbered 0 and 1
5. Create duplex links between the nodes to form a STAR Topology
6. Setup TCP Connection between n(1) and n(3)
7. Apply CBR Traffic over TCP
8. Schedule events and run the program.

Program:

```
#stop and wait protocol in normal situation

#features : labelling, annotation, nam-graph, and window size monitoring set ns [new Simulator]

set n0 [$ns node] set n1 [$ns node]

$ns at 0.0 "$n0 label Sender" $ns at 0.0 "$n1 label Receiver" set nf [open stop.nam w]

$ns namtrace-all $nf set f [open stop.tr w] $ns trace-all $f

$ns duplex-link $n0 $n1 0.2Mb 200ms DropTail $ns duplex-link-op $n0 $n1 orient right

$ns queue-limit $n0 $n1 10 Agent/TCP set nam_tracevar_ true set tcp [new Agent/TCP]

$tcp set window_ 1 $tcp set maxcwnd_ 1 $ns attach-agent $n0 $tcp

set sink [new Agent/TCPSink] $ns attach-agent $n1 $sink $ns connect $tcp $sink

set ftp [new Application/FTP] $ftp attach-agent $tcp

$ns add-agent-trace $tcp tcp $ns monitor-agenttrace $tcp $tcp tracevar cwnd_

$ns at 0.1 "$ftp start"

$ns at 3.0 "$ns detach-agent $n0 $tcp ; $ns detachagent $n1 $sink" $ns at 3.5 "finish"

$ns at 0.0 "$ns trace-annotate \"Stop and Wait with normal operation\"" $ns at 0.05 "$ns traceannotate \"FTP starts at 0.1\""

$ns at 0.11 "$ns trace-annotate \"Send Packet_0\""
$ns at 0.35 "$ns trace-annotate \"Receive Ack_0\""

$ns at 0.56 "$ns trace-annotate \"Send Packet_1\""
$ns at 0.79 "$ns traceannotate \"Receive Ack_1\""

$ns at 0.99 "$ns trace-annotate \"Send Packet_2\""
$ns at 1.23 "$ns trace-annotate \"Receive Ack_2 \""

$ns at 1.43 "$ns traceannotate \"Send Packet_3\""
$ns at 1.67 "$ns trace-annotate \"Receive Ack_3\""

$ns at 1.88 "$ns trace-annotate \"Send Packet_4\""
$ns at 2.11 "$ns trace-annotate \"Receive Ack_4\""

$ns at 2.32 "$ns trace-annotate \"Send Packet_5\""
$ns at 2.55 "$ns trace-annotate \"Receive Ack_5 \""

$ns at 2.75 "$ns trace-annotate \"Send Packet_6\""
$ns at 2.99 "$ns trace-annotate \"Receive Ack_6\""

$ns at 3.1 "$ns trace-annotate \"FTP stops\""

proc finish {} {
global ns nf
$ns flush-trace
close $nf
puts "running nam..."
exec nam stop.nam & exit 0 }
$ns run
```

Output:



Problem 3: Simulation and implementation of go back n Sliding Window protocol

Aim:

To Simulate and to study of Go Back N protocol

Software Requirements:

NS-2 Simulator

Theory:

Go Back N is a connection-oriented transmission. The sender transmits the frames continuously. Each frame in the buffer has a sequence number starting from 1 and increasing up to the window size. The sender has a window i.e., a buffer to store the frames. This buffer size is the number of frames to be transmitted continuously. The size of the window depends on the protocol designer.

Operation:

1. A station may send multiple frames as allowed by the window size.
2. Receiver sends an ACK *i* if frame *i* has an error. After that, the receiver discards all incoming frames until the frame with error is correctly retransmitted.
3. If sender receives an ACK *i* it will retransmit frame *i* and all packets *i+1, i+2,...* which have been sent, but not been acknowledged.

Algorithm:

1. The source node transmits the frames continuously.
2. Each frame in the buffer has a sequence number starting from 1 and increasing up to the window size.

3. The source node has a window i.e. a buffer to store the frames. This buffer size is the number of frames to be transmitted continuously.
4. The size of the window depends on the protocol designer.
5. For the first frame, the receiving node forms a positive acknowledgement if the frame is received without error.
6. If subsequent frames are received without error (up to window size) cumulative positive acknowledgement is formed.
7. If the subsequent frame is received with error, the cumulative acknowledgment error-free frames are transmitted. If in the same window two frames or more frames are received with error, the second and the subsequent error frames are neglected. Similarly even the frames received without error after the receipt of a frame with error are neglected.
8. The source node retransmits all frames of window from the first error frame.
9. If the frames are errorless in the next transmission and if the acknowledgment is error free, the window slides by the number of error-free frames being transmitted.
10. If the acknowledgment is transmitted with error, all the frames of window at source are retransmitted, and window doesn't slide.
11. This concept of repeating the transmission from the first error frame in the window is called as GOBACK N transmission flow control protocol.

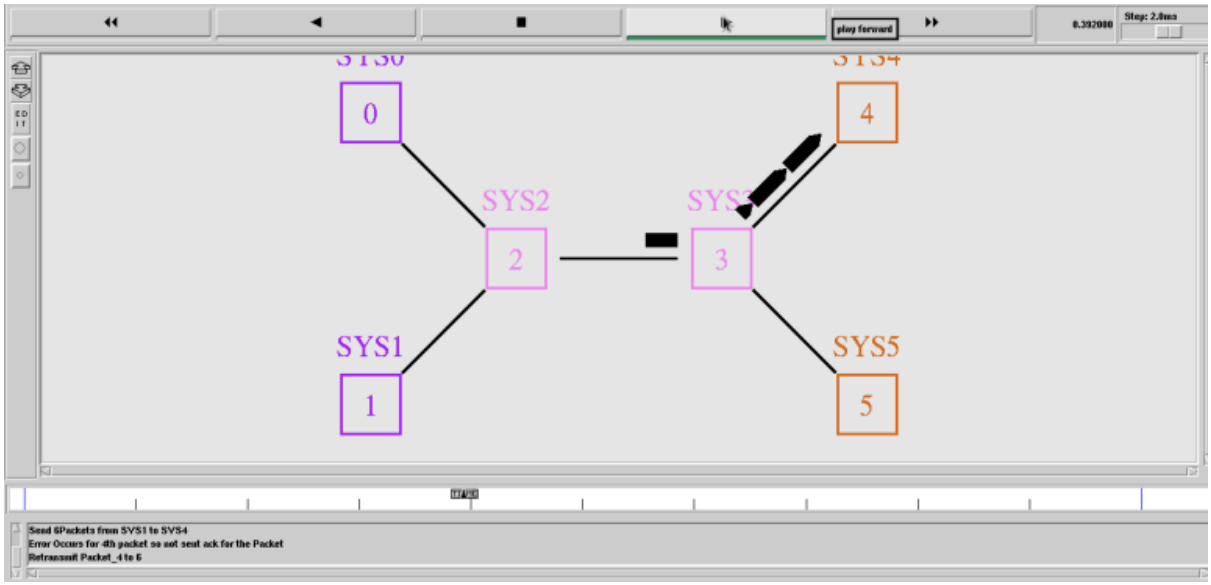
Program:

```
#send packets one by one
set ns [new Simulator]
set n0 [$ns node]
set n1 [$ns node]
set n2 [$ns node]
set n3 [$ns node]
set n4 [$ns node]
set n5 [$ns node]
$n0 color "purple"
$n1 color "purple"
$n2 color "violet"
$n3 color "violet"
$n4 color "chocolate"
$n5 color "chocolate"
 $n0 shape box ;
$n1 shape box ;
$n2 shape box ;
$n3 shape box ;
$n4 shape box ;
$n5 shape box ;
$ns at 0.0 "$n0 label SYS0"
$ns at 0.0 "$n1 label SYS1"
$ns at 0.0 "$n2 label SYS2"
$ns at 0.0 "$n3 label SYS3"
$ns at 0.0 "$n4 label SYS4"
$ns at 0.0 "$n5 label SYS5"
set nf [open goback.nam w]
$ns namtrace-all $nf
set f [open goback.tr w]
$ns trace-all $f
$ns duplex-link $n0 $n2 1Mb 20ms
DropTail $ns duplex-link-op $n0
$n2 orient right-down $ns queue-
limit $n0 $n2 5
$ns duplex-link-op $n1 $n2 orient
right-up
$ns duplex-link $n2 $n3 1Mb 20ms
DropTail $ns duplex-link-op $n2
$n3 orient right
$ns duplex-link $n3 $n4 1Mb 20ms
DropTail $ns duplex-link-op $n3
$n4 orient right-up
$ns duplex-link $n3 $n5 1Mb 20ms
DropTail $ns duplex-link-op $n3
$n5 orient right-down Agent/TCP
set_nam_tracevar_true
set tcp [new Agent/TCP]
$tcp set fid 1
$ns attach-agent $n1 $tcp
set sink [new Agent/TCPSink]
$ns attach-agent $n4 $sink
$ns connect $tcp $sink
set ftp [new Application/FTP]
$ftp attach-agent $tcp
$ns at 0.05 "$ftp start"
$ns at 0.06 "$tcp set windowlnit
6"
$ns at 0.06 "$tcp set maxcwnd 6"
$ns at 0.25 "$ns queue-limit $n3
$n4 0"
$ns at 0.26 "$ns queue-limit $n3
$n4 10"
$ns at 0.305 "$tcp set windowlnit
4"
$ns at 0.305 "$tcp set maxcwnd 4"
$ns at 0.368 "$ns detach-agent $n1
$tcp ;
$ns detach-agent $n4 $sink"
$ns at 1.5 "finish"
```

```
$ns duplex-link $n1 $n2 1Mb 20ms DropTail
```

```
$ns at 0.0 "$ns trace-annotate \"Goback N end\"" $ns at 0.05 "$ns trace-annotate \"FTP starts at 0.01\""
$ns at 0.06 "$ns trace-annotate \"Send 6Packets from SYS1 to SYS4\""
$ns at 0.26 "$ns trace-annotate \"Error Occurs for 4th packet so not sent ack for the Packet\""
```

Output:



Problem 4: Simulate and implement selective repeat sliding window protocol.

Aim:

To study an simulate selective repeat window protocol.

Theory:

Selective Repeat ARQ is a specific instance of the Automatic Repeat-reQuest (ARQ) Protocol. It may be used as a protocol for the delivery and acknowledgement of message units, or it may be used as a protocol for the delivery of subdivided message sub-units. When used as the protocol for the delivery of messages, the sending process continues to send a number of frames specified by a window size even after a frame loss. Unlike GoBack-N ARQ, the receiving process will continue to accept and acknowledge frames sent after an initial error. The receiver process keeps track of the sequence number of the earliest frame it has not received, and sends that number with every ACK it sends. If a frame from the sender does not reach the receiver, the sender continues to send subsequent frames until it has emptied its window. The receiver

continues to fill its receiving window with the subsequent frames, replying each time with an ACK containing the sequence number of the earliest missing frame. Once the sender has sent all the frames in its window, it re-sends the frame number given by the ACKs, and then continues where it left off. The size of the sending and receiving windows must be equal, and half the maximum sequence number (assuming that sequence numbers are numbered from 0 to n-1) to avoid miscommunication in all cases of packets being dropped. To understand this, consider the case when all ACKs are destroyed. If the receiving window is larger than half the maximum sequence number, some, possibly even all, of the packages that are resent after timeouts are duplicates that are not recognized as such. The sender moves its window for every packet that is acknowledged.

Advantage:

1. Fewer retransmissions.

Disadvantages:

1. More complexity at sender and receiver.
2. Receiver may receive frames out of sequence.

Algorithm:

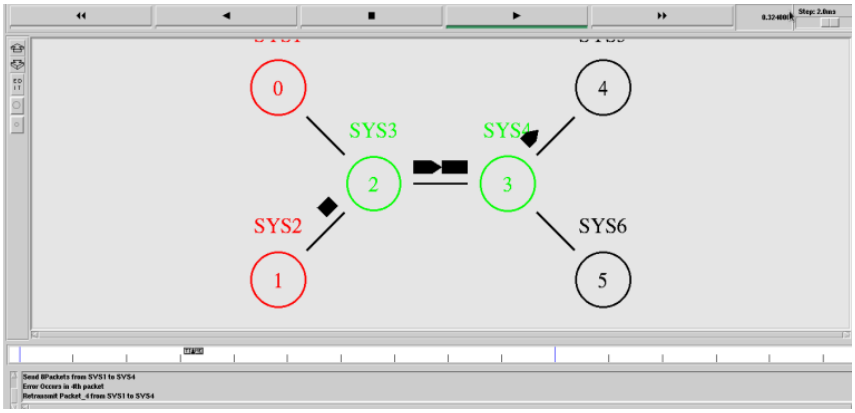
1. The source node transmits the frames continuously.
2. Each frame in the buffer has a sequence number starting from 1 and increasing up to the window size.
3. The source node has a window i.e. a buffer to store the frames. This buffer size is the number of frames to be transmitted continuously.
4. The receiver has a buffer to store the received frames. The size of the buffer depends upon the window size defined by the protocol designer.
5. The size of the window depends according to the protocol designer.
6. The source node transmits frames continuously till the window size is exhausted. If any of the frames are received with error only those frames are requested for retransmission (with a negative acknowledgement).
7. If all the frames are received without error, a cumulative positive acknowledgement is sent.
8. If there is an error in frame 3, an acknowledgement for the frame 2 is sent and then only Frame 3 is retransmitted. Now the window slides to get the next frames to the window.
9. If acknowledgment is transmitted with error, all the frames of window are retransmitted. Else ordinary window sliding takes place. (* In implementation part, Acknowledgment error is not considered).
10. If all the frames transmitted are errorless the next transmission is carried out for the new window.
11. This concept of repeating the transmission for the error frames only is called Selective Repeat transmission flow control protocol.

Program:

```
#send packets one by one
set ns [new Simulator]
set n0 [$ns node]
set n1 [$ns node]
set n2 [$ns node]
set n3 [$ns node]
$n4 shape circle ;
$n5 shape circle ;
$ns at 0.0 "$n0 label SYS1"
$ns at 0.0 "$n1 label SYS2"
$ns at 0.0 "$n2 label SYS3"
$ns at 0.0 "$n3 label SYS4"
```

```
set n4 [$ns node]
set n5 [$ns node]
$n0 color "red"
$n1 color "red"
$n2 color "green"
$n3 color "green"
$n4 color "black"
$n5 color "black"
$n0 shape circle ;
$n1 shape circle ;
$n2 shape circle ;
$n3 shape circle ;
DropTail$ns duplex-link-op $n2
$n3 orient right $ns duplex-link
$n3 $n4 1Mb 10ms DropTail $ns
duplex-link-op $n3 $n4 orient
right-up
$ns duplex-link $n3 $n5 1Mb 10ms
DropTail $ns duplex-link-op $n3
$n5 orient right-down Agent/TCP
set_nam_tracevar_true
set tcp [new Agent/TCP]
$tcp set fid 1
$ns attach-agent $n1 $tcp
set sink [new Agent/TCPSink]
$ns attach-agent $n4 $sink
$ns connect $tcp $sink
set ftp [new Application/FTP]
$ftp attach-agent $tcp
$ns at 0.05 "$ftp start"
$ns at 0.06 "$tcp set windowlnit
8"
$ns at 0.06 "$tcp set maxcwnd 8"
$ns at 0.25 "$ns queue-limit $n3
$n4 0"
$ns at 0.26 "$ns queue-limit $n3
$n4 10"
$ns at 0.30 "$tcp set windowlnit
1"
$ns at 0.30 "$tcp set maxcwnd 1"
$ns at 0.30 "$ns queue-limit $n3
$n4 10"
$ns at 0.47 "$ns detach-agent
$n1 $tcp;
$ns detach-agent $n4 $sink"
$ns at 1.75 "finish"
$ns at 0.0 "$ns trace-annotate
\"Select and repeat\""
$ns at 0.05 "$ns trace-annotate
\"FTP starts at 0.01\""
$ns at 0.06 "$ns trace-annotate
\"Send 8Packets from SYS1 to
SYS4\""
$ns at 0.0 "$n4 label SYS5"
$ns at 0.0 "$n5 label SYS6"
set nf [open Srepeat.nam w]
$ns namtrace-all $nf
set f [open Srepeat.tr w]
$ns trace-all $f
$ns duplex-link $n0 $n2 1Mb 10ms
DropTail $ns duplex-link-op $n0
$n2 orient right-down $ns queue-
limit $n0 $n2 5
$ns duplex-link $n1 $n2 1Mb 10ms
DropTail $ns duplex-link-op $n1
$n2 orient right-up
$ns duplex-link $n2 $n3 1Mb 10ms
DropTail $ns duplex-link-op $n2
$n3 orient right
$ns duplex-link $n3 $n4 1Mb 10ms
DropTail $ns duplex-link-op $n3
$n4 orient right-up
$ns duplex-link $n3 $n5 1Mb 10ms
DropTail $ns duplex-link-op $n3
$n5 orient right-down Agent/TCP
set_nam_tracevar_true
set tcp [new Agent/TCP] $tcp
set fid 1
$ns attach-agent $n1 $tcp
set sink [new Agent/TCPSink]
$ns attach-agent $n4 $sink
$ns connect $tcp $sink
set ftp [new Application/FTP]
$ftp attach-agent $tcp
$ns at 0.05 "$ftp start"
$ns at 0.06 "$tcp set windowlnit
8"
$ns at 0.06 "$tcp set maxcwnd 8"
$ns at 0.25 "$ns queue-limit $n3
$n4 0"
$ns at 0.26 "$ns queue-limit $n3
$n4 10"
$ns at 0.30 "$tcp set windowlnit
1"
$ns at 0.30 "$tcp set maxcwnd 1"
$ns at 0.30 "$ns queue-limit $n3
$n4 10"
$ns at 0.47 "$ns detach-agent $n1
$tcp;
$ns detach-agent $n4 $sink"
$ns at 1.75 "finish"
$ns at 0.0 "$ns trace-annotate
\"Select and repeat\""
$ns at 0.05 "$ns trace-annotate
\"FTP starts at 0.01\""
$ns at 0.06 "$ns trace-annotate
\"Send 8Packets from SYS1 to
SYS4\"" }
$ns run
```

Output:



Problem 5: Simulation and implementation of distance vector routing algorithm.

Aim:

To simulate and study the Distance Vector routing algorithm using simulation.

Software Requirement:

NS-2 Simulator

Theory:

Distance Vector Routing is one of the routing algorithms in a Wide Area Network for computing shortest path between source and destination. The Router is one main devices used in a wide area network. The main task of the router is Routing. It forms the routing table and delivers the packets depending upon the routes in the table either directly or via an intermediate device. Each router initially has information about its all neighbours. Then this information will be shared among nodes.

Algorithm:

1. Create a simulator object.
2. Define different colours for different data flows.
3. Open a nam trace file and define finish procedure then close the trace file, and execute nam on trace file.
4. Create n number of nodes using for loop.
5. Create duplex links between the nodes.
6. Setup UDP Connection between n(0) and n(5).
7. Setup another UDP connection between n(1) and n(5).
8. Apply CBR Traffic over both UDP connections.
9. Choose distance vector routing protocol to transmit data from sender to receiver.
10. Schedule events and run the program.

Program:

```
set ns [new Simulator]
set nr [open thro.tr w]
$ns trace-all $nr
set nf [open thro.nam w]
$ns namtrace-all $nf
proc finish { } {
global ns nr nf
$ns flush-trace
close $nf
close $nr
exec nam thro.nam &
exit 0 }
for { set i 0 } { $i < 12} { incr
i 1 } {
set n($i) [$ns node]}
for {set i 0} {$i < 8} {incr i} {
$ns duplex-link $n($i) $n([expr
$i+1]) 1Mb 10ms DropTail }
$ns duplex-link $n(0) $n(8) 1Mb
10ms DropTail
$ns duplex-link $n(1) $n(10) 1Mb
10ms DropTail $ns duplex-link
$n(0) $n(9) 1Mb 10ms DropTail $ns
duplex-link $n(9) $n(11) 1Mb 10ms
DropTail $ns duplex-link $n(10)
$n(11) 1Mb 10ms DropTail $ns
duplex-link $n(11) $n(5) 1Mb 10ms
DropTail set udp0 [new Agent/UDP]
$ns attach-agent $n(0) $udp0
set cbr0 [new
Application/Traffic/CBR]
$cbr0 set packetSize_ 500
$cbr0 set interval_ 0.005
$cbr0 attach-agent $udp0
set null0 [new Agent/Null]
$ns attach-agent $n(5) $null0
$ns connect $udp0 $null0
set udp1 [new Agent/UDP]
$ns attach-agent $n(1) $udp1
set cbr1 [new
Application/Traffic/CBR]$cbr1
set packetSize_ 500
$cbr1 set interval_ 0.005
$cbr1 attach-agent $udp1
set null0 [new Agent/Null]
$ns attach-agent $n(5) $null0
$ns connect $udp1 $null0
$ns rtproto DV
$ns rtmodel-at 10.0 down
$n(11) $n(5)
$ns rtmodel-at 15.0 down $n(7)
$n(6)
$ns rtmodel-at 30.0 up $n(11)
$n(5)
$ns rtmodel-at 20.0 up $n(7)
$n(6)
$udp0 set fid_ 1
$udp1 set fid_ 2
$ns color 1 Red
$ns color 2 Green
$ns at 1.0 "$cbr0 start"
$ns at 2.0 "$cbr1 start"
$ns at 45 "finish"
$ns run
```

Output: