



# Simple Data Link Protocols

## Goals

- 1) Examining Data Link Layer protocols
- 2) Finding the dependence of link utilization to frame size and window size in sliding window protocol

## Introduction

Data Link Layer deals with the algorithms required to achieve reliable, efficient peer to peer communication using the physical layer. These algorithms are designed to provide appropriate services to the network layer such as acknowledged connection oriented services. Several algorithms can be used to provide such a service and the objective of this exercise is to study the concepts of these algorithms.

To provide a reliable service in which the data link at the transmitter node can be perfectly sure about the delivery of each frame to the destination, it should receive a feedback from the receiver acknowledging the correct reception of the transmitted frame. This acknowledge (ACK) can also be used to control the data flow from the source to the destination. A transmitter would not send frames until it receives permission from destination (an ACK for the previous frame) and therefore can not overflow the receiver.

The simplest protocol that can be used for these purposes is called "Stop and Wait". In this protocol, the transmitter sends one frame and then waits for an ACK from the receiver before sending the next frame. To avoid a deadlock which can be caused by a lost frame or a lost ACK, the transmitter can use a timer. When the delay in receiving the ACK exceeds a certain amount (time-out) the transmitter assumes something (the packet or its ACK) has been lost and re-transmit the frame. This protocol works well if the propagation time (the time it takes for the frame to propagate from source to destination) is small compared to the frame time span. There are many situations in which this is a valid assumption. However, when the propagation delay is no longer a negligible factor, this protocol is a wasteful protocol that does not use the system resources efficiently.

The window-based protocols can be used to increase the efficiency of the system by allowing the transmission of a few packets (As much as specified by the transmitter window size) before allowing the reception of an ACK from the receiver. In effect, this would create a pipeline of un-acknowledged frames at the transmitter. The "Go Back-N" and "Selective Repeat" protocols are two protocols that can increase the efficiency of the transmission using this concept. In "Go Back-N" protocol, the transmitter would go back N frames and re-start the transmission again if it does not receive an ACK for a frame or receive a negative acknowledge (NACK) for it. In "Selective Repeat", the transmitter would



only send those frames which have timed out on ACK or have received a NACK for them. Appropriate use of timers in these protocols would prevent deadlocks. In this exercise, we try to become familiar with the network simulator “NS” package and learn to use it properly to investigate the properties of data link layer protocols.

## 1) Stop and Wait Protocol

In this part we want to run a simulation to investigate the dependency of link utilization on the frame length in “Stop and Wait” protocol. We use NS to generate an output file which contains two columns. The first column is frame length and the second column is the transmitter bandwidth for that frame length.

Link utilization is defined as:

$$\text{Transmitter bandwidth/ Link bandwidth}$$

Where for a special time interval T you can find the Transmitter bandwidth as:

$$((\text{number of frames sent during T}) * (\text{number of bytes in each frame})) / T$$

To get a correct result T should be large compared to link delay.

You will set the link bandwidth yourself and therefore, the link bandwidth is also known.

Now you can write a small program in MATLAB to find link utilization and plot it.

In first step you should write a Tcl script, you can write it in any text editor such as Note pad and save it with tcl extension (\*.tcl).

At the first line of your code create a new simulator object by this command: (Be careful of Capital letters. Tcl is a case sensitive language!)

```
set ns [new Simulator]
```

The next step is to commands NS to generate output files. Usually two output file formats are produced by NS (NAM and tr types). The NAM file will be the input to another program called NAM to show a graphical representation of the packet flow.

The tr type is a file which records all the parameters during simulation. The NAM and tr files usually have large sizes specially when the simulation interval is long. So if you command NS to create these files, you should wait a while to allow the simulation to finish its job and produce the files.

These files should be opened at the beginning of the script and they are closed at the end of the script. Here we only need the NAM file so we open it at the start of the simulation:

```
set namf [open out.nam w]
namf is the handles for out.nam file.
```

You also want another output file to save transmitter bandwidth in it. Open another file for this purpose:



```
set tb [open b.txt w]
```

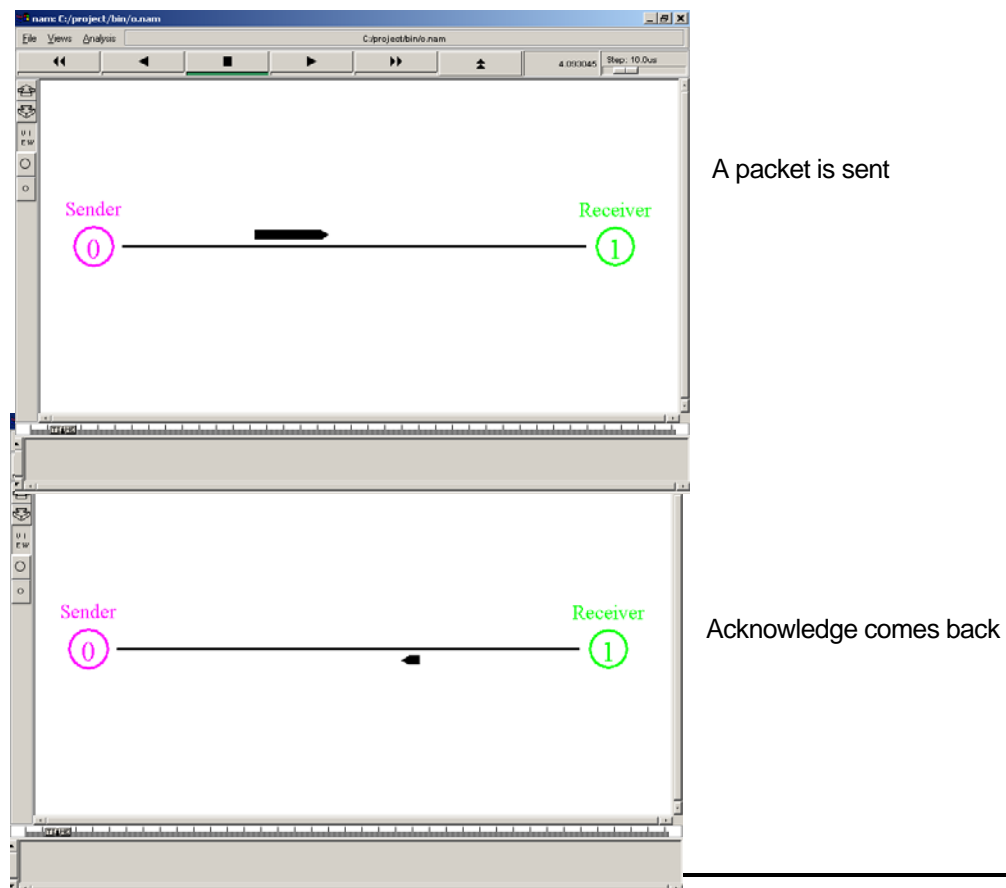
Now, write a procedure and call it at the end of the simulation to close the output files and run NAM to see the output:

```
proc finish {} {  
    global ns tb namf  
    $ns flush-trace  
    close $tb  
    close $namf  
    exec nam out.nam &  
    exit 0  
}
```

Add this command to your code to write all packets properties and conditions in NAM file.

```
$ns namtrace-all $namf
```

The next step is to define a network with a transmitter and a receiver node that uses the stop and wait algorithm.





The following two lines define the two nodes for your network:

```
set n0 [$sns node]
set n1 [$sns node]
```

Two nodes n0 and n1 are connected by a link with a 1Mb bandwidth, 2ms delay and DropTail queue format. (What Drop Tail queue means does not have any importance here.)

```
set linebw 1Mb
$sns duplex-link $n0 $n1 $linebw 2ms DropTail
```

To send and receive data you need to put agents over nodes.

NS does not support agents in Data link layer but it has TCP and UDP agents in transport layer. Transport layer and data link Layer have some common tasks such as flow control, so TCP agents can also implement the sliding window protocol and act the same as sliding window agents. However, they do not have a constant window size and change their window size dynamically.

In this part we want a “Stop and Wait” agent. This is like a sliding window agent with a window size of 1. By a little trick you can make a TCP agent act like this. Just set its maximum window size to 1 so that it cannot increase its window size or decrease it and it will have the constant window size which is equal to 1.

There are some other parameters related to a TCP source here is some of them:

- maxcwnd\_: maximum congestion window size
- windowlnit\_: initial window size in slow start
- seqno\_ : highest sequence number for data from data source to TCP
- ack\_ : highest ack seen from receiver

```
#Create a TCP agent
set tcp [new Agent/TCP]
$tcp set window_ 1
$tcp set maxcwnd_ 1
$tcp set packetSize_ 10
$tcp set maxseq_ 500000
```

Now attach the agent to n0

```
$sns attach-agent $n0 $tcp
```

An agent requires a source to generate traffic for it. FTP application can be used for this purpose:

```
# Create a FTP traffic source and attach it to tcp
set ftp [new Application/FTP]
```



```
$ftp attach-agent $tcp
```

You also need an agent in the receiver which acts as traffic sink. It receives packets and sends acknowledges. This must be attached to node n1.

```
#define a sink agent for TCP source  
set tcpsnk [new Agent/TCPSink]  
#attach the sink agent to node1  
$ns attach-agent $n1 $tcpsnk  
$ns connect $tcp $sink
```

Define the start time for the traffic sources:

```
$ns at 1 "$ftp start"
```

Write a procedure which is called every 2 seconds to record the transmitter bandwidth. To find the source bandwidth you can use the following approach:

Each ack that a TCP source receives has a sequence number. In reality the sequence number is not a large number so it will wrap around after a while. But you can set the maximum sequence number for the TCP source to a large number so that it does not wrap around. The change in sequence number of ack packets in any time interval equals to the number of packets which have been received during that time interval.

The record procedure is called once in the main program and it will call itself again and again. In this procedure the number of received bytes in a 2 sec interval and the frame length is written to a file. The frame length is then incremented for the next interval.

```
proc record {} {  
    global tb framelength tcp bw0 temp1 ns  
    #Set the time after which the procedure should be called again  
    set time 3  
    #save the highest sequence number of received frames in a variable called temp  
    set temp [$tcp set ack_ ]  
    # the number of recieved frames: (current sequence number) - (previous sequence number)  
    set bw0 [ expr ($temp - $temp1) * $framelength ]  
    # save current sequence number for later use  
    set temp1 $temp  
    #Get the current time  
    set now [$ns now]  
    #save frame length and transmitter bandwidth  
    puts $tb " $framelength [expr ($bw0 / $time)] "  
    #increase frame length for next time interval  
    set framelength [expr $framelength + 10 ]  
    $tcp set packetSize_ $framelength  
    #call the procedure again  
    $ns at [expr $now+$time] "record"  
}
```



The finish procedure should be executed at the end of simulation time. By this command you can execute it in right time.

```
$ns at 100 "finish"
```

**Note: don't forget to call your procedure.**

Note: to save more points in the output file we have adapted such a big number for simulation time. So it may take a long time for the simulation to be finished because NS wants to create a large file for NAM. If you want the simulation to run much more faster, don't command NS to generate any NAM output file.

**You can see the simulation timer in Command DOS environment or in UNIX terminal using below command**

```
puts "$now"
```

**or save it in your output file using**

```
puts $tb "$now"
```

The last line finally starts the simulation.

```
$ns run
```

Now you can save your file and start the script by running the command

```
'DOS> ns yourfile.tcl'
```

In MS-DOS environment or you can simply right click on it and open it with ns.exe in windows .

At the end of the simulation you can see the file lu.txt that has two columns.

Use MATLAB to normalize the second column by the link bandwidth ( 8 Mb) and plot it versus the first column.

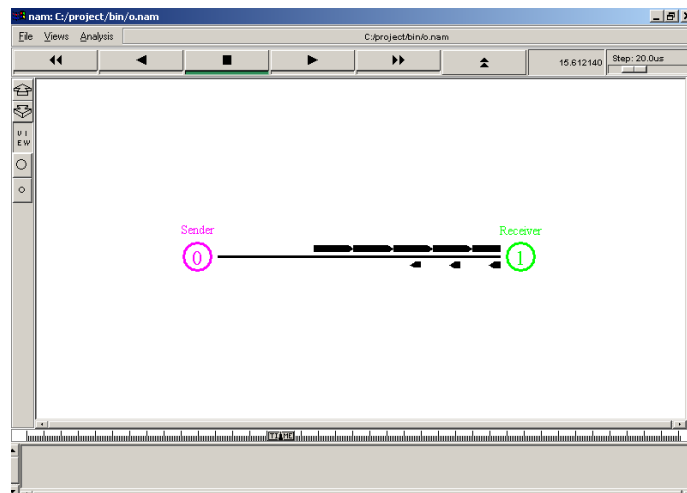
## II) Sliding Window Protocol

In this part we want to find the relation between link utilization and window size for a sliding window protocol. The approach is same as the first part except that we record the window size instead of frame length and the window size is incremented each time the record procedure is called.

```
proc record {} {  
    global sink tb tcp linebw bw0 temp1 ns  
    #Set the time after which the procedure should be called again
```



```
set time 6
#save the current sequence number of TCP source in a variable called temp
set temp [$tcp set t_seqno_]
# the number of recieved frames: (current sequence number) - (previous sequence number)
set bw0 [expr ($temp - $temp1) * 10]
# save current sequence number for later use
set temp1 $temp
#Get the current time
set now [$ns now]
set wind [$tcp set window_]
set maxwind [$tcp set maxcwnd_]
#save bandwidth
puts $tb " $wind [expr ($bw0 / $time)] "
#increase frame length for next time interval
$tcp set window_ [expr $wind + 2]
$tcp set maxcwnd_ [expr $maxwind + 2]
#call the procedure again
$ns at [expr $now+$time] "record"
}
```



Note: real size of a TCP packet is not the one you have set with `packetsize_` parameter. There is an overhead. Set the size of packets to a number and run the simulation. In the NAM window right click on a packet and monitor it. Nam will show you the real size of the packet and you can find out how many overhead bytes are present. If you don't consider this overhead you will never have the link utilization equal to one.

Use "color" and "duplex-link-op" commands to put your nodes in proper place and direction and with different color in NAM for better animation. For more information, refer to NS document files.

Run NS and use the output file to plot the link utilization versus window size.

Note: You can change the link bandwidth and delay by trial and error and find appropriate range that is useful for this experiment.

If the curve has an unexpected shape for large window sizes, this may have been caused by the small queue size.



### III) Sliding Window with varying window sizes

Repeat part I for window size=2, 4, 6, 8 and save the results (frame length and transmitter bandwidth) for each window size in a separate file.

Now use these results to plot the link utilization versus  $T_{prop}/T_{frame}$  for various window sizes. Plot all of them in the same graph. (You may use logarithmic x-axis for better representation of the results)

### IV) Sliding window protocols in channels with loss

In this part we want to find the dependence of link utilization to frame size in the presence of error.

Insert a loss on the link between  $n0$  and  $n1$ . To do this we can insert a loss module that drops packets with a programmable statistical characteristic using the following commands:

```
set loss_module [new ErrorModel]
$loss_module set rate_ .1
$loss_module set min_ 0
$loss_module set max_ 1
$loss_module ranvar [new RandomVariable/Uniform]
$loss_module drop-target [new Agent/Null]
$ns at .01 "$ns lossmodel $loss_module $n0 $n1"
```

The other point is that TCP changes its window size whenever an error occurs. If you want a constant window size you can write a routine that sets the window size for the TCP source periodically and do not let it change when an error occurs:

```
proc wind {} {
global tcp ns
$tcp set maxcwnd_ 5
$tcp set windowinit_ 5
$tcp set window_ 5
$tcp set cwnd_ 5
set now [$ns now]
$ns at [expr $now+.001] "wind"
}
```

You can call this procedure in the main program once and it will call it self periodically. Configure NS to generate an output file and you will process it later to get your desired parameters

```
set file1 [open err.tr w]
$ns trace-all $file1
```



And close this file in finish procedure:

```
close $file1
```

During the simulation a file named out.tr will be created for you. The output file contains 12 columns that each of them records a parameter. Here is a sample selection of this file:

```
r 1.130687 2 0 tcp 552 ----- 1 2.0 5.0 112 365
+ 1.130687 0 1 tcp 552 ----- 1 2.0 5.0 112 365
d 1.130687 0 1 tcp 552 ----- 1 2.0 5.0 112 365
r 1.131294 1 6 tcp 552 ----- 2 3.0 6.0 36 203
+ 1.131294 6 1 ack 40 ----- 2 6.0 3.0 36 375
- 1.131294 6 1 ack 40 ----- 2 6.0 3.0 36 375
r 1.1324 6 1 ack 40 ----- 2 6.0 3.0 33 366
+ 1.1324 1 0 ack 40 ----- 2 6.0 3.0 33 366
- 1.1324 1 0 ack 40 ----- 2 6.0 3.0 33 366
```

As you can see, each row represents an event and the details of the event characteristics are listed in each row using the following format:

Event	Time	From node	To node	Packet type	Packet size	Flags	Fid	Src address	Dst address	Seq number	Packet id
-------	------	-----------	---------	-------------	-------------	-------	-----	-------------	-------------	------------	-----------

- Event: this field can take four values
- r: receive
- +:en queue
- -:de queue
- d :drop
- time: the time at which the event occurred
- from node: number of the input node of the link at which the event occurred
- to node: number of the output node of the link at which the event occurred
- packet type: type of the packet .it can be tcp, ack or any other acceptable type.
- Packet size: size of packet in bytes
- Flags: ----- means no flag has been set .
- ---A--- means congestion window reduced
- Flow id: users can specify an id for each flow. this id can be used to specify a color for each flow.
- Src address: source address of the packet in the form: node.port
- dst address: destination address of the packet in the form: node.port
- Seq number: the sequence number of packet in network layer
- Packet id: a unique number given to each packet

Now you can use the output file to compute the link utilization:



First you should find out the total transmission time. If you take a look at the output file you will see that this time is much smaller than the total simulation time and this is due to the behavior of TCP source. TCP agent implements an algorithm which is used to control the network congestion. To do so, TCP source stops transmission for a while if it does not receive ACKs properly. Therefore, when there is a lossy link, TCP source would become idle at some time intervals and this can be clearly seen by monitoring the time of the events recorded in the output file. You can detect these idle intervals in the output file by irregular jumps in time column.

Here is an example:

```
r 4.38336 1 0 ack 40 ----- 0 1.0 0.0 29 76  
+ 5.35736 0 1 tcp 13000 ---A--- 0 0.0 1.0 30 77
```

Here you can see that there is no packet transmission for approximately one second.

Write a simple matlab program and find the sum of all jumps in time column. Subtract the simulation time from this value to find out the effective transmission time (Teff) (Total time minus the times in which the source was idle).

The next important parameter is the number of packets which were actually transmitted (and received properly) during this effective transmission time. This equals to highest receive sequence number in the output file.

And the last step is to find the link utilization:

Link utilization= $(\text{highest sequence number}) * T_{\text{frame}} / T_{\text{eff}}$

Run the simulation several times for different values of frame length and find link utilization for each frame lengths. Use the results to Plot the link utilization versus frame length. Compare the result of your simulation to the theoretical results obtained in the class and comment on your results.

## V) Experiment Report:

Write a proper report using MS Word and include the results and discussions of your results in the report.

At the end of your report, write a one page summary that describes an introduction to the following concepts:

- Network Simulator software, its main features and its architecture.
- A brief introduction to Tcl and OTcl.
- How NS, C++, Tcl and OTcl work together to provide an effective simulation environment

You should then pack this report with the programs you have written and the summary that you have written together, zip them and send one file to the E-mail address of the course. The received file should contain:

- The scripts written for each part
- The report that contains the results of simulations in the form of plots. For each part, explain the results you have found and how they are related to the theories discussed in the course.
- The summary page.