CMPUT313-LAB2
By James Devito

1.1 - Introduction

In this experiment we are testing the stop and wait protocol, the Objective of this experiment is to obtain results found in simulation and compare them with expected results by mathematical model as well as implement the stopandwait protocol for use on a 5 node path network. For this experiment we will be using the cnet simulator the protocol is stopandwait and our topology files consist of 4 different topologies as followed

- NETa : The length of each application layer message is 2000 bytes, and the physical layer does not corrupt or lose frames.
- NETb : The length of an application layer message is a uniform random variable with value between 2000 bytes and the default maximum length, and the physical layer does not corrupt or lose frames
- NETc : The length of each application layer message is 2000 bytes, and the physical layer does not lose frames, but corrupts only data frames (that is, ACK frames are delivered reliably) with probability 0.5 each.
- NETd : The length of each application layer message is 2000 bytes, and the physical layer does not lose frames, but corrupts both data and ACK frames with probability 0.5 each.

1.2 - Experimental Obersvations and Data

Perth

	Messages	Bytes	KBytes/sec
Generated	46	94,000	0.31
Received OK	-	-	-
Errors received	-		

Sydney

	Messages	Bytes	KBytes/sec
Generated	-	-	-
Received OK	46	92,000	0.31
Errors received	-		

NETa

```
2 hosts, 0 routers, 0 mobiles, 0 accesspoints
Simulation time                    : 300000000
Events raised                      : 139
Messages generated                 : 46
Messages delivered                 : 46
Message bandwidth                  : 5736
Average delivery time              : 2789142
Frames transmitted                 : 92
Frames received                    : 91
Efficiency (bytes AL) / (bytes PL) : 97.66
EV_REBOOT                          : 2
EV_PHYSICALREADY                   : 91
EV_APPLICATIONREADY                : 46
```

Perth

	Messages	Bytes	KBytes/sec
Generated	35	583,925	1.94
Received OK	-	-	-
Errors received	-		

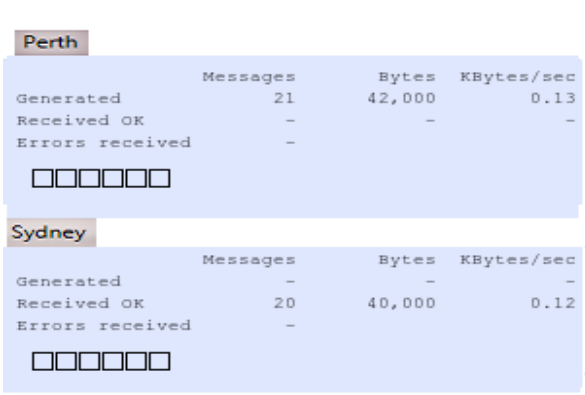
Sydney

	Messages	Bytes	KBytes/sec
Generated	-	-	-
Received OK	34	570,893	1.92
Errors received	-		

NETb

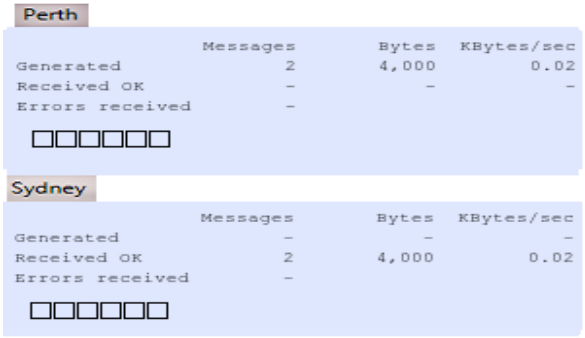
```
2 hosts, 0 routers, 0 mobiles, 0 accesspoints
Simulation time                    : 300000000
Events raised                      : 105
Messages generated                 : 35
Messages delivered                 : 34
Message bandwidth                  : 29475
Average delivery time              : 5285344
Frames transmitted                 : 69
Frames received                    : 68
Efficiency (bytes AL) / (bytes PL) : 98.14
EV_REBOOT                          : 2
EV_PHYSICALREADY                   : 68
EV_APPLICATIONREADY                : 35
```

1.2 - Experimental Obersvations and Data Continued



NETc

```
2 hosts, 0 routers, 0 mobiles, 0 accesspoints
Simulation time                          : 300000000
Events raised                            : 105
Messages generated                       : 21
Messages delivered                       : 20
Message bandwidth                        : 1334
Average delivery time                    : 11987588
Frames transmitted                       : 62
Frames received                          : 61
Frames corrupted                         : 22
Efficiency (bytes AL) / (bytes PL)       : 46.79
EV_REBOOT                                : 2
EV_PHYSICALREADY                         : 61
EV_APPLICATIONREADY                      : 20
EV_TIMER1                                : 22
```



NETd

```
2 hosts, 0 routers, 0 mobiles, 0 accesspoints
Simulation time                          : 300000000
Events raised                            : 55
Messages generated                       : 2
Messages delivered                       : 2
Message bandwidth                        : 1043
Average delivery time                    : 15332478
Frames transmitted                       : 17
Frames received                          : 17
Frames corrupted                         : 9
Frames lost                              : 26
Efficiency (bytes AL) / (bytes PL)       : 14.08
EV_REBOOT                                : 2
EV_PHYSICALREADY                         : 17
EV_APPLICATIONREADY                      : 2
EV_TIMER1                                : 34
```

Problems

- Mathematical Model

First let's define

- Transmission Rate = R
- Transmission time of a packet = Tpkt
- Message Rate = Rmsg
- Propagation Delay = Tprop
- Size of a packet = L
- Round Trip Time = RTT

Common Calculations

R = 56kbps
L = 2000 bytes
Rmsg = 1000ms
Tprop = 2500ms

Threrrorfree = 1 / (Tpkt + RTT)
Thr = Thr-errorfree / Nr
Where Nr is the average number of times the protocol retransmits

Tpkt = L/R

Tpkt = 2000bytes/56kbps
Tpkt = 0.2857 s

RTT = 2 * (Tprop) + Rmsg
RTT = 6000ms

Please assume these calculations to be common to each test unless stated otherwise

NETa

In the case of *NETa* there is no error generation so our throughput will be error free
Using the above values we find : Thr-errorfree = 1 / (Tpkt + RTT) = 0.1590 pkts/sec
Each packet is 2KB so,
Thr-errorfree = 0.1590 pkts/second * 2KB/pkt = 0.318 KB/sec **= 0.32 KB/sec**
and Thr-messages = 0.1590 msg/sec **= 0.16 msg/sec**

NETb

In the case of *NETb* there is no error generation so our throughput will be error free
L-min = 2000 bytes
L-max = 8000 bytes
L is a uniform random variable and can be expressed as L = (L-min + L-max)/2
L = (2000 + 16000) / 2 = 9000 bytes

Therefore Tpkt = L/R = 5000 bytes / 56 Kbps
Threrrorfree = 1 / (Tpkt + RTT) = 0.1372 pkt/sec
Similarl to NETa (9KB per packet) Thr = 1.235KB/sec **= 1.23 KB/sec**
Thr-messages = 0.1372 msg/sec **= 0.13 msg/sec**

NETc

In the case of *NETc* there is error generation

Using common values we find : Thr-errorfree = 0.1590 pkt/sec
There is only 1 source of error so Nr = 1 / (1 – Pframecorrupt)
Therefore Thr = 0.1590 / (1 / (1 – 0.5))

Giving us a result of Thr = 0.0795 pkt/sec
Each packet is 2Kb
Therefore Thr = 0.09925 pkt/sec * 2KB/pkt = 0.159 KB/sec **= 0.16KB/sec**
and Thr-messages = 0.0795 msg/sec = **0.08 msg/sec**

NETd

In the case of NETd there is error generation in the form of lost and corrupt frames
let Pframecorrupt be P = 0.5
Pframeloss be Q = 0.5

Using common values we find : Thr-errorfree = 0.1590 pkt/sec

In this case there are 2 sources of error so we must first find our Nr
and each source of error applies to 2 different transmistions (ACK and DATA)
Nr = 1 / (((1 – 0.5) * (1 – 0.5)) * ((1 – 0.5) * (1 – 0.5)))
Nr = 16

Therefore Thr = (Thr-errorfree / Nr) = 0.1590 / 16
Thr = 0.00993 pkt/sec
Each packet is 2kb
Therefore Thr = 0.00993 pkt/sec * 2KB/pkt = 0.0198 KB/sec = **0.019 KB/sec**
and Thr-messages = 0.00993 msg/sec = **0.01 msg/sec**

1.3 Results

For **NETa**
Expected: Thr-data = **0.32 KB/sec** Experimental Result : Thr-data = **0.31 KB/sec**
: Thr-msg = **0.16 msg/sec** : Thr-msg = 46 msg/ 300 sec = **0.15 msg/sec**

% Error – Thr-data = | ((Thr-data(found) – Thr-data(expected)) / Thr-data (expected)) | * 100
= ((0.31 – 0.32) / 0.32) * 100 = **3.1%**

similarly % Error Thr-msg = **6.6%**

In the case of NETa our experiment results are within an acceptable error range of our expected results. This confirms that are calculations were acceptably correct.

For **NETb**
Expected: Thr-data = **1.23 KB/sec** Experimental Result : Thr-data = **1.94 KB/sec**
: Thr-msg = **0.13 msg/sec** : Thr-msg = 35 msg/ 300 sec = **0.12 msg/sec**

% Error formula similar to NETa section
% Error – Thr-data = **60.16%**
similarly % Error Thr-msg = **7.69%**

In the case of NETb the error for our estimation of message throughput falls within acceptable error, however in this case we have observed a large error in the throughput for the data, but this is acceptable because in the case of this experiment the design parameter of the size of the message was set to a uniform random variable between 2000 bytes and 16000 bytes. Considering this it is expected that our observed error falls relatively close to 50%.
also we observe that if we were to recalculate the expected value for throughput using the maximum message size (16KB) we find Thr-data = 1.93KB/sec which has a 0.5% difference with the experimental result

For **NETc**
Expected: Thr-data = **0.16 KB/sec** Experimental Result : Thr-data = **0.13 KB/sec**
: Thr-msg = **0.08 msg/sec** : Thr-msg = 21 msg/ 300 sec = **0.07 msg/sec**

% Error formula similar to NETa section
% Error – Thr-data = **18.75%**
similarly % Error Thr-msg = **12.5%**

In the case of NETc we observe a slightly larger error when comparing our expected values and our experimental values. This is likely because of the introduction of random error generation, this will cause different runs to potentially have different end values. I believe if we were to run the simulation for longer or ran the simulation many times we will see a decrease in our error

For **NETd**
Expected: Thr-data = **0.019 KB/sec** Experimental Result : Thr-data = **0.02 KB/sec**
: Thr-msg = **0.01 msg/sec** : Thr-msg = 2 msg/ 300 sec = **0.006 msg/sec**

% Error formula similar to NETa section
% Error – Thr-data = **5.3%**
similarly % Error Thr-msg = **32.9%**

In the case of NETc we observe a very low error in the data throughput, however the throughput in messages is quite significant. This may be attributed to the randomness in lost and corrupt frames and applying it to the simulation, if we were to run the simulation for much longer the error may decrease, therefore the error is within a reasonable threshhold

1.4 Conclusion

In this part I have found that much of the performance in the stop and wait protocol is limited by the round trip time and is hindered even further upon the introduction of errors. This experiment has made it very clear the purpose behind extending the stop and wait protocol to include a sliding window (Go-Back-N) as the performance is not nearly as dependant on the round trip time and is able to send many messages while waiting on propagation.

2. STOP-AND-WAIT Protocol on Path Networks

In this part we are asked to extend the protocol in stopandwait.c to provide reliable data transmission between the two end hosts of a path network where all the internal hosts are routers. Our new protocol “saw.c” is designed for use with its corresponding topology file SAW which has the specifications previously stated. In our new protocol for the case of simplicity we assume that the network has only one traffic flow corresponding to the application layer messages generated by the leftmost host and destined to the rightmost host of the poth. At any instant, each router runs a stopandwait protocol for managing the outgoing traffing to its right neighbour. The is every link runs the stopandwait protocol.

Each router has a buffer for storing frames containing application messages that pass through the router. If a frame containing an application message arrives to a router and finds the slot occupied, the arrived frame is discarded and should be retransmitted.

For this protocol I have decided to segregate the original code and my own editted code by and if statement. Even though this brings a problem of bad coding style I argue the opposite, even though code is being repeated by keeping a copy of the original code it becomes very easy to read which portions of the code implement the original stopandwait.c (hosts) and which portions use our new protocol (router handling). This is a design choice and this choice has been made based on readability this choice also has the added benefit of being able to reference what is happening in the original stopandwait.c while not having to look at other files.

For writing the extended protocol.c I have used stopandwait.c as a base and editted it to represent the new features, implementing the new features has been made much easier thanks to the help of the lab TA's and the example resources made available on the CMPUT313 webpage.

I had some problems implementing saw.c. The protocol works fine for some frames however after a random amount of frames (4-8) the end host gets a ER_BADARG error on write_application, It is a mystery to me what is causing it and the TA was unable to figure out the nature of the error as well. I wound up getting rather frustrated and run out of time trying to fix it and started from scratch and reimplented it on saturday

3. Feedback

First of all I would like to thank the lab TA's and a colleage who embarrasingly enough I forgot to get the name for assisting me in this experiment as well as the eclass recources

This experiment has given me a fresh look into as to why extending the stopandwait protocol to a sliding window (gobackN) is so effective, I have observed that the round trip time has such a significant effect on throughput that being able to send new information while we wait on the RTT can potentially drastically improve efficiency. I have already realized this point through course material but it is refreshing to see it for myself in my own lab results. Some other things that I liked include a clear description as to what is expected in part1 of the lab, Extending the protocol in part 2 was also very informing and I have learned much more about CNET and networking during the process of implementing saw.c

Somethings that I think could use improvement.

- -The wording on the network in part 2 is a little bit weird at first I did not know what was meant until I reread it a few times
- -Lack of examples of a routed network protocol. This issue is hard to deal with as I know a lot of the intent in the lab assignment is getting us to learn CNET for ourselves, but I think an example on a routed protocol would have been immensely useful for completing the assignment as well as a learning resource would have been great though I understand it is difficult to provide such a resource without giving away too much and trivializing the lab