Department of Computer Science

CS2005 Networks & Operating Systems Task 1

Academic Year 2020-21

Jermain Dunkley

1939797

Table of Contents

1. Introduction

This report is in response to a previous claim by NOSSoft and attempts to provide evidence and explain where necessary the problems arising in the school's new virtual learning application.

For a quick summary of the events to current date, I had discovered numerous bugs—ones that render the application unusable—in the newest update to the application. Previously, the system worked incredibly well, performing all its functions exactly as it ought to, however the newest update makes the application worse than it was before.

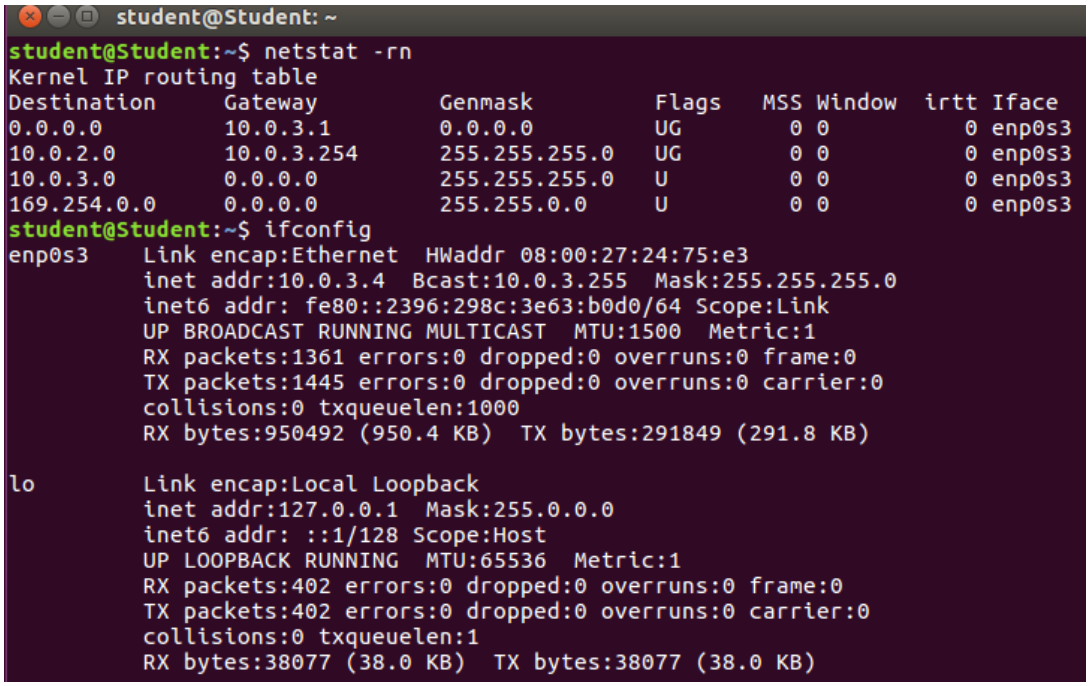
I immediately sought to contact NOSSoft to rollback the update until such a fix for it could be implemented, however I was repeatedly met with reluctance and skepticism, suggesting that there instead might be a problem with our network setup.

As such, this report sets to ease that skepticism and, more importantly, aid in the smooth rollout of a new update that has fixed the various bugs that have arisen in the current update. This report will make clear the various problems that have been encountered so far using Wireshark to systematically capture all transactions between two separate hosts. Both the original and updated client and server will be shown for easy comparison via screenshots. This report will also provide a protocol table detailing the sequence of events between server and client, as well as evidence that our current network settings are correct and running well.

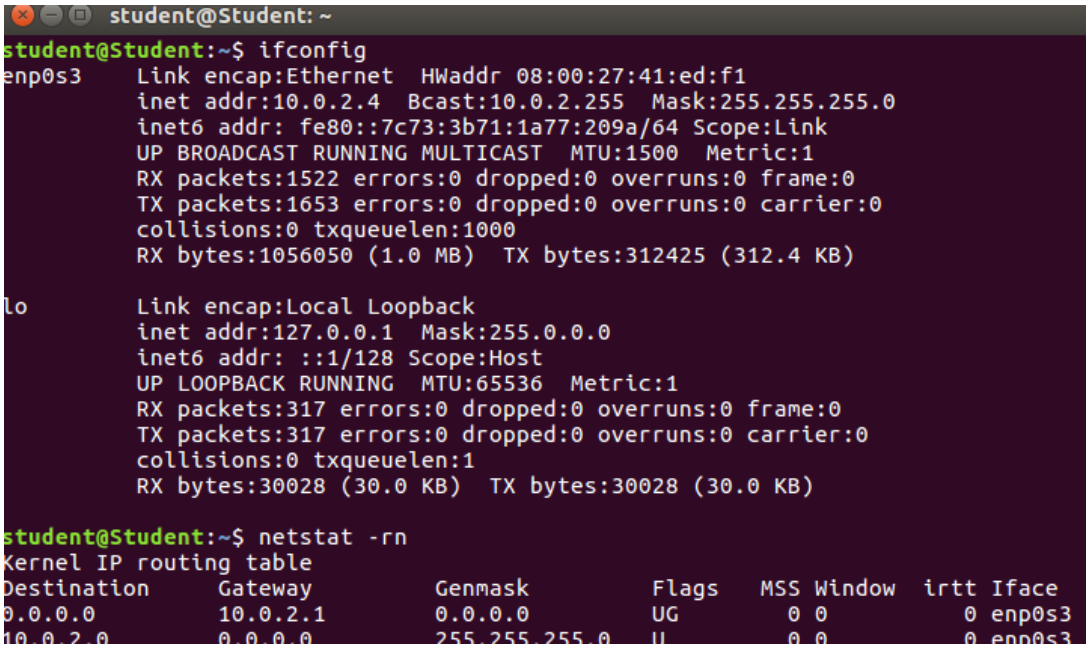
Finally, using the screenshots of both the original and updated calcClient and calcServer, I will layout the problems, how I came across them, the source of those problems being either the client or server, and how those problems should be expected to run.

2. Test Network Documentation

The following will show that the network I am running the application on is working correctly:



The first thing to note is the IP Address of the client server. After running a test using ifconfig, the figure brought back the inet address, a figure which usually incorporates the host name as well as the IP Address itself. For client machine being run, this appeared as 10.0.3.4.



Likewise, for the server machine, when ifconfig is entered, an IP Address of 10.0.2.4 is seen. Furthermore, I can also provide evidence that packets can be exchanged between source and destination from each host.

```
student@Student:~$ sudo route add -net 10.0.3.0/24 gw 10.0.2.254
[sudo] password for student:
student
Sorry, try again.
[sudo] password for student:
student@Student:~$ netstat -rn
Kernel IP routing table
Destination     Gateway         Genmask         Flags   MSS Window  irtt Iface
0.0.0.0         10.0.2.1        0.0.0.0         UG        0 0          0 enp0s3
10.0.2.0        0.0.0.0         255.255.255.0   U         0 0          0 enp0s3
10.0.3.0        10.0.2.254      255.255.255.0   UG        0 0          0 enp0s3
169.254.0.0     0.0.0.0         255.255.0.0     U         0 0          0 enp0s3
```

```
student@Student: ~

student@Student:~$ sudo route add -net 10.0.2.0/24 gw 10.0.3.254
[sudo] password for student:
student
Sorry, try again.
[sudo] password for student:
student@Student:~$ netstat -rn
Kernel IP routing table
Destination     Gateway         Genmask         Flags   MSS Window  irtt Iface
0.0.0.0         10.0.3.1        0.0.0.0         UG        0 0          0 enp0s3
10.0.2.0        10.0.3.254      255.255.255.0   UG        0 0          0 enp0s3
10.0.3.0        0.0.0.0         255.255.255.0   U         0 0          0 enp0s3
169.254.0.0     0.0.0.0         255.255.0.0     U         0 0          0 enp0s3
```

Executing netstat –rn showcases the connection between the two hosts on both sides. This shows each particular host being able to access the other as a destination, indicated by the IP Addresses underneath the Destination column.

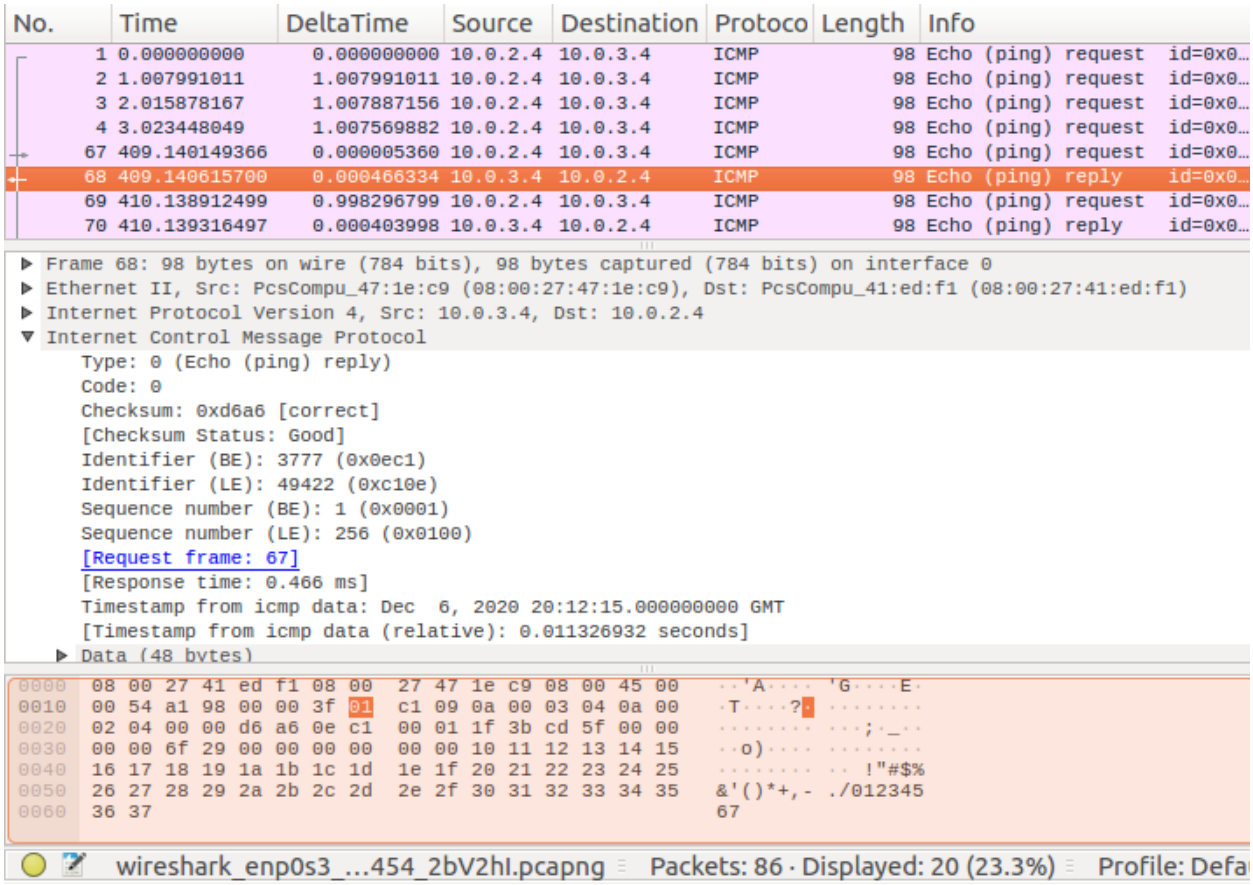
```
student@Student:~$ ping -c 4 10.0.3.4
PING 10.0.3.4 (10.0.3.4) 56(84) bytes of data.
64 bytes from 10.0.3.4: icmp_seq=1 ttl=63 time=0.719 ms
64 bytes from 10.0.3.4: icmp_seq=2 ttl=63 time=0.420 ms
64 bytes from 10.0.3.4: icmp_seq=3 ttl=63 time=0.410 ms
64 bytes from 10.0.3.4: icmp_seq=4 ttl=63 time=0.379 ms

--- 10.0.3.4 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 2997ms
rtt min/avg/max/mdev = 0.379/0.482/0.719/0.137 ms
student@Student:~$
```

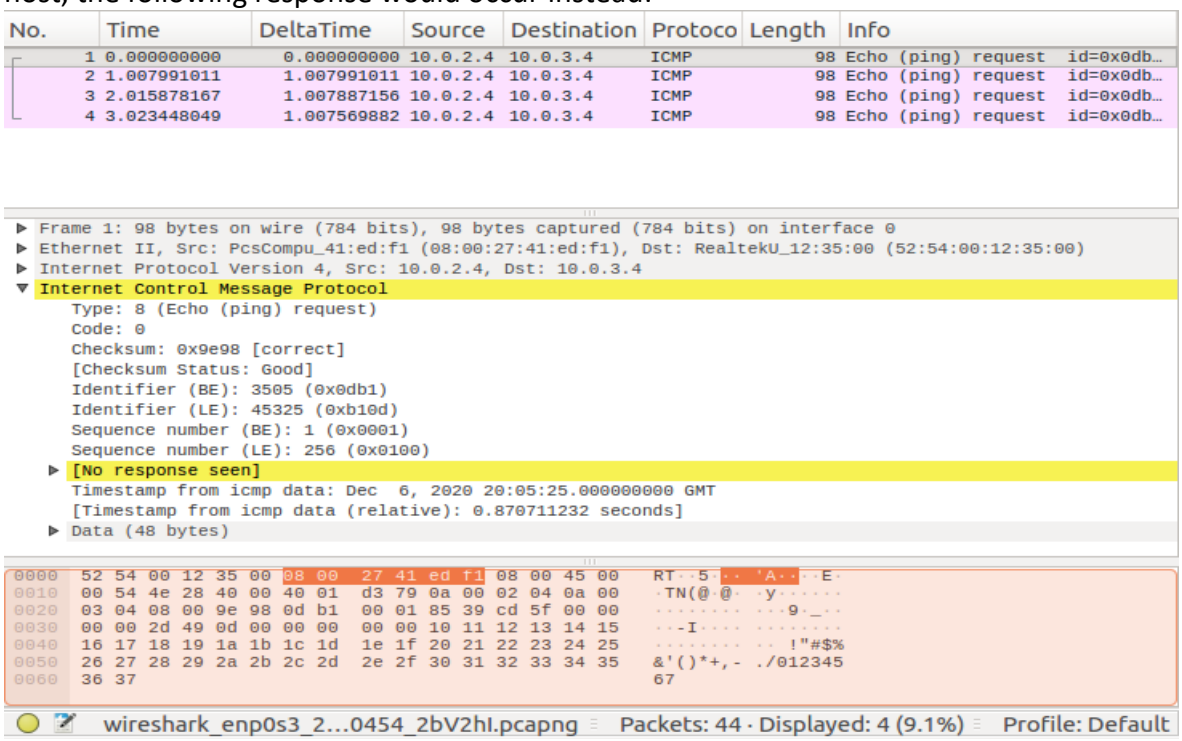
```
student@Student:~$ ping -c 4 10.0.2.4
PING 10.0.2.4 (10.0.2.4) 56(84) bytes of data.
64 bytes from 10.0.2.4: icmp_seq=1 ttl=63 time=0.421 ms
64 bytes from 10.0.2.4: icmp_seq=2 ttl=63 time=0.451 ms
64 bytes from 10.0.2.4: icmp_seq=3 ttl=63 time=0.452 ms
64 bytes from 10.0.2.4: icmp_seq=4 ttl=63 time=0.388 ms

--- 10.0.2.4 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3000ms
rtt min/avg/max/mdev = 0.388/0.428/0.452/0.026 ms
student@Student:~$
```

Executing a ping request of 4 packets to both IP Addresses, shows that both were successfully received on either end. This can also be seen when captured and analysed with Wireshark.



The message (no.68) is sending a reply back to the IP Address which requested the packet. In this case, the source for the reply is the 10.0.3.4 host, while the destination is the 10.0.2.4 host. The request and reply would be the opposite, however if initiating ping on the 10.0.3.4 host. Without the host machines being connected, if a ping request was to be sent to either host, the following response would occur instead:



3. calcClient and calcServer Documentation

	calcClient		**calcServer**
			[run calcServer]
	[run calcClient]		
			[accept calcClient connection]
	WHILE NOT TERMINATED		WHILE NOT TERMINATED
		#S1	SEND “calculator server ready and waiting” TO calcClient
	RECEIVE “calculator server ready and waiting” FROM calcServer		
	READ “menuOption” FROM user		
	IF USER INPUT “1”		
	SEND “add operands” TO calcServer		
			RECEIVE “add operands” FROM calcClient
			SEND “send operands to add” TO calcClient
	RECEIVE “send operands to add” FROM calcServer		
	USER INPUT “3 1”		
			RECEIVE “3 1” FROM calcClient
			SEND “4” To calcClient
	RECEIVE “4” FROM calcServer		
	PRINT “4”		
#C1	SEND “next operation please” TO calcServer		
			RECEIVE “next operation please” FROM calcClient
			Go back to #S1
	IF USER INPUT “2”		
	SEND “sub operands” TO calcServer		
			RECEIVE “sub operands” FROM calcClient
			SEND “send operands to sub” TO calcClient
	RECEIVE “send operands to sub” FROM calcServer		
	USER INPUT “3 1”		
			RECEIVE “3 1” FROM calcClient
			SEND “2” TO calcClient
	RECEIVE “2” FROM calcServer		
	PRINT “2”		
	Go back to #C1		
	IF USER INPUT “3”		
	SEND “multi operands” TO calcServer		

			RECEIVE “multi operands” FROM calcClient
			SEND “send operands to multiply” TO calcClient
	RECEIVE “send operands to multiply” FROM calcServer		
	USER INPUT “4 2”		RECEIVE “4 2” FROM calcClient
			SEND “8” TO calcClient
	RECEIVE “8” FROM calcServer		
	PRINT “8”		
	Go back to #C1		
	IF USER INPUT “4”		
	SEND “div operands” TO calcServer		
			RECEIVE “div operands” FROM calcClient
			SEND “send operations to divide” TO calcClient
	RECEIVE “send operands to divide” FROM calcServer		
	USER INPUT “16 2”		
			RECEIVE “ 16 2” FROM calcClient
			SEND “8” TO calcClient
	RECEIVE “2” FROM calcServer		
	PRINT “2”		
	Go back to #C1		
	IF USER INPUT “0”		
	SEND “endcomms” TO calcServer		
	[Terminate]		
			RECEIVE “endcomms” FROM calcClient
			[Terminate]
	ENDWHILE		ENDWHILE

4. calcClientUpdate and calcServerUpdate Documentation

	updatedCalcClient		**updatedCalcServer**
			[run calcServer]
	[run calcClient]		
			[accept calcClient connection]
	WHILE NOT TERMINATED		WHILE NOT TERMINATED
		#S1	SEND “calculator server ready and waiting” TO calcClient
	RECEIVE “calculator server ready and waiting” FROM calcServer		
	READ “menuOption” FROM user		
	IF USER INPUT “1”		
	SEND “add operands” TO calcServer		
			RECEIVE “add operands” FROM calcClient
			SEND “send operands to sub” TO calcClient
	RECEIVE “send operands to sub” FROM calcServer		
	USER INPUT “3 1”		
			RECEIVE “3 1” FROM calcClient
			SEND “2” To calcClient
	RECEIVE “2” FROM calcServer		
	PRINT “2”		
#C1	SEND “next operation please” TO calcServer		
			RECEIVE “next operation please” FROM calcClient
			Go back to #S1
	IF USER INPUT “2”		
	SEND “sub operands” TO calcServer		
			RECEIVE “sub operands” FROM calcClient
			SEND “send operands to add” TO calcClient
	RECEIVE “send operands to add” FROM calcServer		
	USER INPUT “3 1”		
			RECEIVE “3 1” FROM calcClient
			SEND “4” TO calcClient
	RECEIVE “4” FROM calcServer		
	PRINT “4”		
	Go back to #C1		
	IF USER INPUT “3”		

	SEND “multi operands” TO calcServer		
			RECEIVE “multi operands” FROM calcClient
			SEND “send operands to multiply” TO calcClient
	RECEIVE “send operands to multiply” FROM calcServer		
	USER INPUT “4 2”		RECEIVE “4 2” FROM calcClient
			SEND “16” TO calcClient
	RECEIVE “16” FROM calcServer		
	PRINT “16”		
	Go back to #C1		
	IF USER INPUT “0”		
	SEND “div operands” TO calcServer		
			RECEIVE “div operands” FROM calcClient
			SEND “send operations to divide” TO calcClient
	RECEIVE “send operands to divide” FROM calcServer		
	USER INPUT “16 2”		
			RECEIVE “ 16 2” FROM calcClient
			SEND “8” TO calcClient
	RECEIVE “8” FROM calcServer		
	PRINT “8”		
	Go back to #C1		
	IF USER INPUT “4”		
	SEND “endcomms” TO calcServer		
	[Terminate]		
			RECEIVE “endcomms” FROM calcClient
			[Terminate]
	ENDWHILE		ENDWHILE

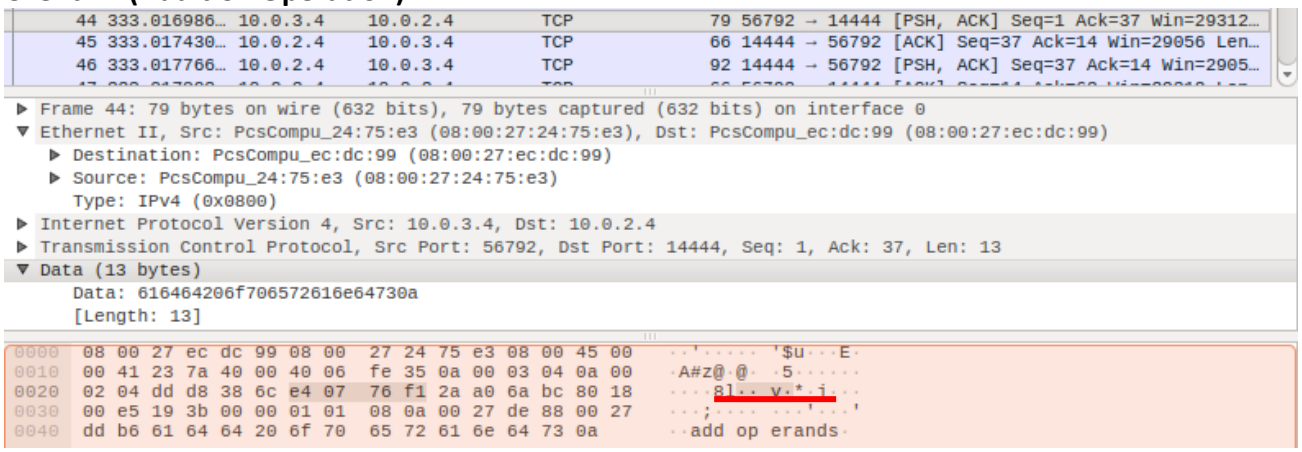
The updated calcClient and calcServer protocols were captured using Wireshark and analysing the transactions of packets between two machines. This information was displayed in Wireshark’s packet-contents window from which I was able to glean the problems that have arisen from the update. These problems have been identified in red in the above protocol table.
What I have documented would lead to me believe that certain parts of the code for the application are linking to methods they shouldn’t be, and that these may be due to extra data added to the code that is causing bugs to appear in the application.

5. Report to the NOSSoft Management

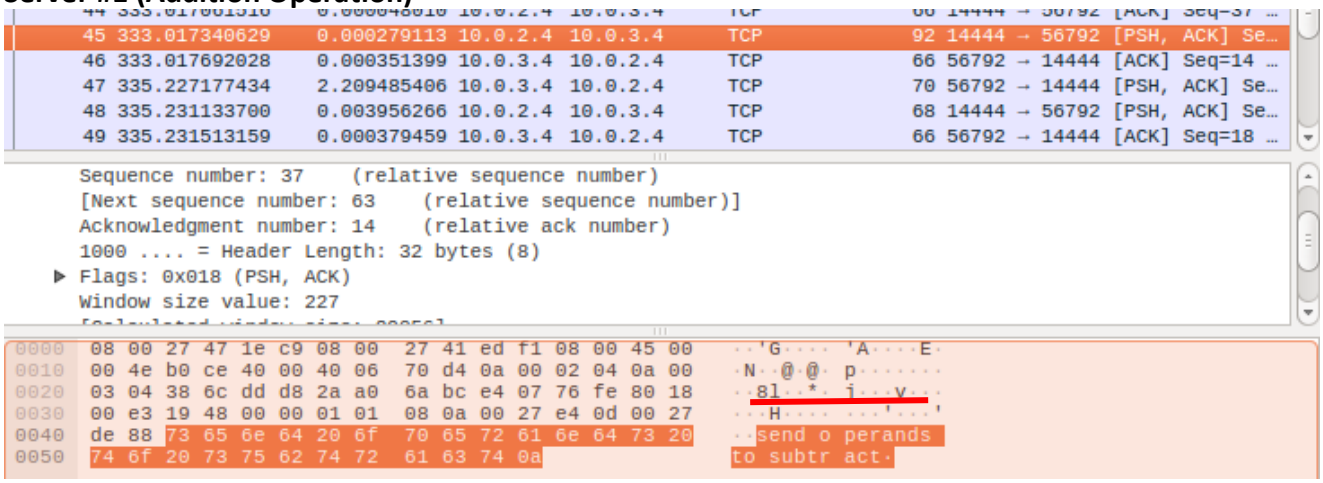
This section aims to clarify and expand upon the problems that I have been facing with the newer update. To quickly identify, the first problem lies in the swapping of certain functions, although there are two different groups of swapped functions and the origin of those problems differ with each group. The final problem lies in the multiplication function.

I'll start by expanding on the first point, namely, that the addition and subtraction functions have been swapped with one another. The test performed, as seen with the protocol table above, involved inputting the numbers "3 1" as operands into both the addition and subtraction operations. There is strong evidence to suggest that this occurred on the server side, as beginning with a message exchanged from the server (10.0.2.4) to the client (10.0.3.4), information within the third row of the packet-content window began to differ from previous messages. This is reflected in the protocol table in which the same packet sends back the 'operands to sub' message to the client protocol. The proper function of both addition and subtraction operations should be reversed, with addition adding and subtraction subtracting numbers.

Client #1 (Addition Operation)



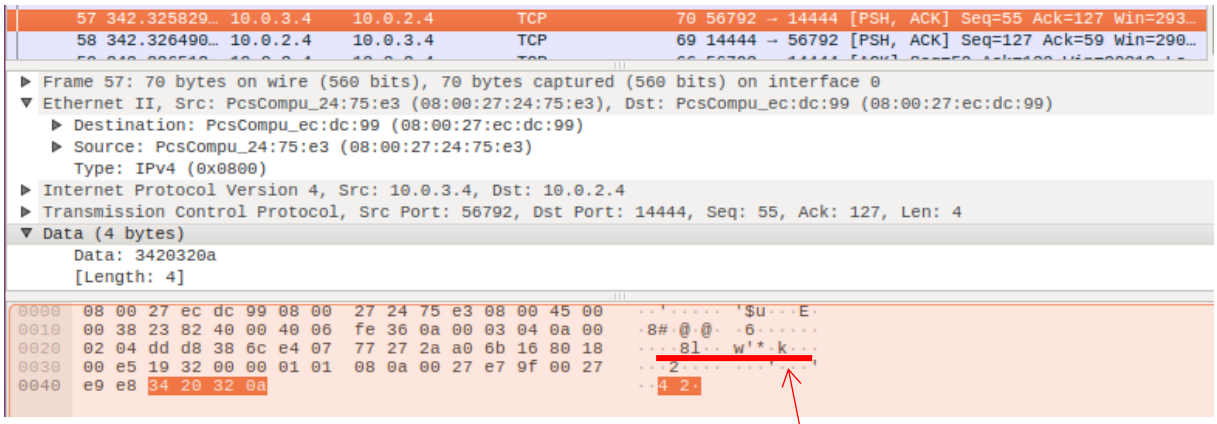
Server #1 (Addition Operation)



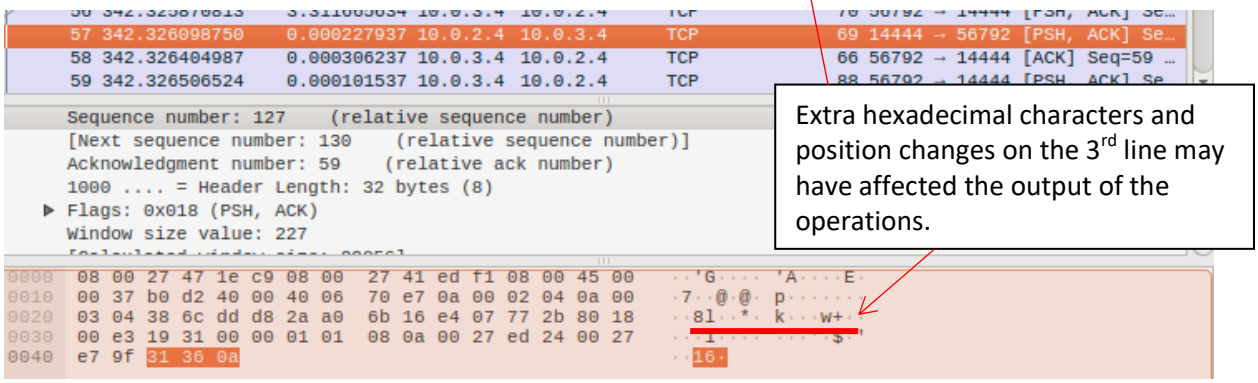
The second problem lies in two other operations being switched. These would be the quit and division operations, or options “0” and “4”. A test was performed in which the contents of each packet were analysed, much like the previous problem. This however, appears to be on the client side. As seen in the protocol table, when “4” is entered, a request for ‘endcomms’ is sent to the server and when “0” is entered, a request for ‘div operands’ is sent. This can pose a significant problem if not quickly fixed as the quit operation under a differently named function suggests an unusable application. As with the previous problem, the behaviour of each function should be switched but from the client code rather than the server.

The third problem is the multiplication function. When used, it does not swap functions like the others, but produces a completely different function entirely. It acts as an exponentiation where the first number is the base and the second is a power raised by. For this test, I entered “4 2”, the result being “16”. A second test of “10 2” with the result being “100” confirmed my suspicions. This problem occurs on the server’s side and appears to be interpreting the operands as an exponent rather than actual multiplication. Instead of this, when entering operands such as “4 2”, this should multiply and display “8”. Likewise “10 2” should display “20”.

Client #1 (Multiplication Operation)

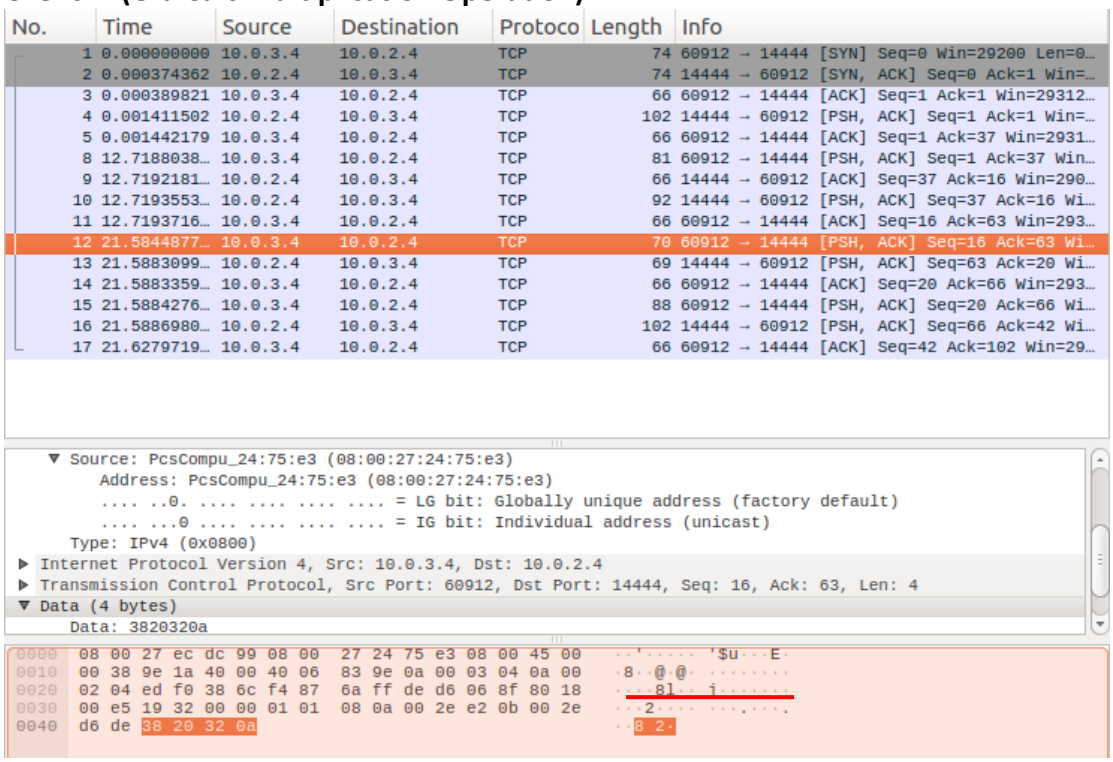


Server #1 (Multiplication Operation)



For comparison, this is how the same multiplication operation would have worked in the previous version of the application:

Client#1 (Old Calc Multiplication Operation)

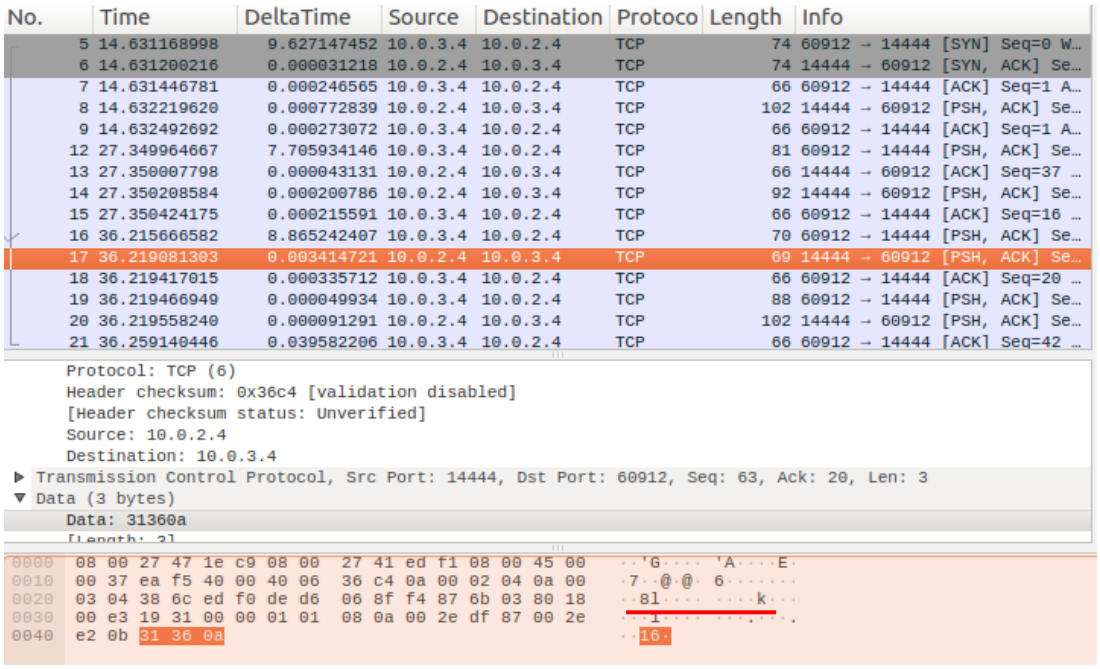


No.	Time	Source	Destination	Protoco	Length	Info
1	0.000000000	10.0.3.4	10.0.2.4	TCP	74	60912 → 14444 [SYN] Seq=0 Win=29200 Len=0…
2	0.000374362	10.0.2.4	10.0.3.4	TCP	74	14444 → 60912 [SYN, ACK] Seq=0 Ack=1 Win=…
3	0.000389821	10.0.3.4	10.0.2.4	TCP	66	60912 → 14444 [ACK] Seq=1 Ack=1 Win=29312…
4	0.001411502	10.0.2.4	10.0.3.4	TCP	102	14444 → 60912 [PSH, ACK] Seq=1 Ack=1 Win=…
5	0.001442179	10.0.3.4	10.0.2.4	TCP	66	60912 → 14444 [ACK] Seq=1 Ack=37 Win=2931…
8	12.7188038…	10.0.3.4	10.0.2.4	TCP	81	60912 → 14444 [PSH, ACK] Seq=1 Ack=37 Win…
9	12.7192181…	10.0.2.4	10.0.3.4	TCP	66	14444 → 60912 [ACK] Seq=37 Ack=16 Win=290…
10	12.7193553…	10.0.2.4	10.0.3.4	TCP	92	14444 → 60912 [PSH, ACK] Seq=37 Ack=16 Wi…
11	12.7193716…	10.0.3.4	10.0.2.4	TCP	66	60912 → 14444 [ACK] Seq=16 Ack=63 Win=293…
12	21.5844877…	10.0.3.4	10.0.2.4	TCP	70	60912 → 14444 [PSH, ACK] Seq=16 Ack=63 Wi…
13	21.5883099…	10.0.2.4	10.0.3.4	TCP	69	14444 → 60912 [PSH, ACK] Seq=63 Ack=20 Wi…
14	21.5883359…	10.0.3.4	10.0.2.4	TCP	66	60912 → 14444 [ACK] Seq=20 Ack=66 Win=293…
15	21.5884276…	10.0.3.4	10.0.2.4	TCP	88	60912 → 14444 [PSH, ACK] Seq=20 Ack=66 Wi…
16	21.5886980…	10.0.2.4	10.0.3.4	TCP	102	14444 → 60912 [PSH, ACK] Seq=66 Ack=42 Wi…
17	21.6279719…	10.0.3.4	10.0.2.4	TCP	66	60912 → 14444 [ACK] Seq=42 Ack=102 Win=29…

```
▼ Source: PcsCompu_24:75:e3 (08:00:27:24:75:e3)
      Address: PcsCompu_24:75:e3 (08:00:27:24:75:e3)
      .... ..0. .... .... .... .... = LG bit: Globally unique address (factory default)
      .... ...0 .... .... .... .... = IG bit: Individual address (unicast)
   Type: IPv4 (0x0800)
▶ Internet Protocol Version 4, Src: 10.0.3.4, Dst: 10.0.2.4
▶ Transmission Control Protocol, Src Port: 60912, Dst Port: 14444, Seq: 16, Ack: 63, Len: 4
▼ Data (4 bytes)
   Data: 3820320a
```

```
0000  08 00 27 ec dc 99 08 00  27 24 75 e3 08 00 45 00   ··'····· '$u···E·
0010  00 38 9e 1a 40 00 40 06  83 9e 0a 00 03 04 0a 00   ·8··@·@· ········
0020  02 04 ed f0 38 6c f4 87  6a ff de d6 06 8f 80 18   ····8l·· j·······
0030  00 e5 19 32 00 00 01 01  08 0a 00 2e e2 0b 00 2e   ···2···· ········
0040  d6 de 38 20 32 0a                                 ··8 2·
```

Server#1(Old Calc Multiplication Operation)



No.	Time	DeltaTime	Source	Destination	Protoco	Length	Info
5	14.631168998	9.627147452	10.0.3.4	10.0.2.4	TCP	74	60912 → 14444 [SYN] Seq=0 W…
6	14.631200216	0.000031218	10.0.2.4	10.0.3.4	TCP	74	14444 → 60912 [SYN, ACK] Se…
7	14.631446781	0.000246565	10.0.3.4	10.0.2.4	TCP	66	60912 → 14444 [ACK] Seq=1 A…
8	14.632219620	0.000772839	10.0.2.4	10.0.3.4	TCP	102	14444 → 60912 [PSH, ACK] Se…
9	14.632492692	0.000273072	10.0.3.4	10.0.2.4	TCP	66	60912 → 14444 [ACK] Seq=1 A…
12	27.349964667	7.705934146	10.0.3.4	10.0.2.4	TCP	81	60912 → 14444 [PSH, ACK] Se…
13	27.350007798	0.000043131	10.0.2.4	10.0.3.4	TCP	66	14444 → 60912 [ACK] Seq=37 …
14	27.350208584	0.000200786	10.0.2.4	10.0.3.4	TCP	92	14444 → 60912 [PSH, ACK] Se…
15	27.350424175	0.000215591	10.0.3.4	10.0.2.4	TCP	66	60912 → 14444 [ACK] Seq=16 …
16	36.215666582	8.865242407	10.0.3.4	10.0.2.4	TCP	70	60912 → 14444 [PSH, ACK] Se…
17	36.219081303	0.003414721	10.0.2.4	10.0.3.4	TCP	69	14444 → 60912 [PSH, ACK] Se…
18	36.219417015	0.000335712	10.0.3.4	10.0.2.4	TCP	66	60912 → 14444 [ACK] Seq=20 …
19	36.219466949	0.000049934	10.0.3.4	10.0.2.4	TCP	88	60912 → 14444 [PSH, ACK] Se…
20	36.219558240	0.000091291	10.0.2.4	10.0.3.4	TCP	102	14444 → 60912 [PSH, ACK] Se…
21	36.259140446	0.039582206	10.0.3.4	10.0.2.4	TCP	66	60912 → 14444 [ACK] Seq=42 …

```
   Protocol: TCP (6)
   Header checksum: 0x36c4 [validation disabled]
   [Header checksum status: Unverified]
   Source: 10.0.2.4
   Destination: 10.0.3.4
▶ Transmission Control Protocol, Src Port: 14444, Dst Port: 60912, Seq: 63, Ack: 20, Len: 3
▼ Data (3 bytes)
   Data: 31360a
   [Length: 3]
```

```
0000  08 00 27 47 1e c9 08 00  27 41 ed f1 08 00 45 00   ··'G···· 'A····E·
0010  00 37 ea f5 40 00 40 06  36 c4 0a 00 02 04 0a 00   ·7··@·@· 6·······
0020  03 04 38 6c ed f0 de d6  06 8f f4 87 6b 03 80 18   ··8l···· ····k···
0030  00 e3 19 31 00 00 01 01  08 0a 00 2e df 87 00 2e   ···1···· ········
0040  e2 0b 31 36 0a                                    ··16·
```

6. Conclusions

Throughout this report, I have documented proof that my network was functioning correctly when using the app. This was done through the use of several tests, each which have helped to explain my knowledge and awareness of how data is exchanged through the network. I have also identified a list of problems with how I came across them as well as providing how they should work. These problems seem to stem from broken code and appear on both the server and client side depending on the problem. I hope that this may be enough to allay any doubts you may have had and persuade you to fix the current update, as it is imperative that a fix is implemented soon. As the system is now, I could not recommend it to the teaching faculty and students, but should the application return to a similar state as it was in its previous iteration, this opinion would change to reflect that. I hope you take this report into careful consideration as you move forward with the application.