TCP/IP Buffered Data Transmission

Devin Trejo

devin.trejo@temple.edu

Date: 2/22/2016

I. Summary

In the TCP/IP Buffered Data Transmission lab we look into sending a large message across several smaller datagrams. Blocking data up into smaller sizes is commonly done on the server side when a client requests a large message to be sent across the network. In this experiment we use our PIC32 MCU to provide a 1060 byte message and parse it in smaller blocks of data. We then send each block of data to the client and analyze the transmission in Wireshark to see if the server does send the data successfully and if the client receives the intended full message. We show success in transmission between server and client when our larger message is parsed into two and three separate datagrams with arbitrary delays put in between. The delays are put in to simulate transmission propagation time. Upon trying to send our larger message in 50 byte blocks we receive server re-transmission behavior that is expected when using TCP as a transfer protocol, as discuss in the previous lab.

I. Introduction

Most of web traffic today is to either Google owned YouTube or Netflix [1]. Both these sites provided video entertainment to their end users. Sending video over the backbone of the internet is a bandwidth intensive operation that requires the frames that compose the resulting video to be split across multiple internet frames.



Figure 1: Top Websites by Bandwidth Usage [1]

Today most file transfers exceeded the limit of the 1500 bytes put on by TCP/IP frames [2]. Therefore, a coordination between server and client is necessary for parsing data into smaller data sizes on the server side and assembly of the full larger message on the client side. This lab was a demonstration of such coordination is indeed possible.

We will experiment with sending a 1060 byte message across two buffered frames, three buffered frames, and finally numerous frames 50 bytes in size. We also experiment with putting in delays between each transmission to simulate transmission propagation time. What we uncover in this lab shows how larger data transfer occur when using TCP/IP.

II. Discussion

Two Buffered Message Transmission

The first experiment is to send our message spread across two send buffers. In between each send command we put in an arbitrary delay. The resulting server send code is seen in Code Snippet 1.

We now boot up our PIC32 board and Wireshark to analyze the packets in transit. In the last lab we analyzed the packets sent when the users *'connects'* and *'resets'* the connection via the Visual Basic GUI. The full Wireshark output for our two buffered message send is as follows:

*Ethernet (not port 3389)					- 🗆 ×
<u>File Edit View Go Capture</u>	Analyze Statistics Telephony Wireless	<u>T</u> ools <u>H</u> elp			
🖉 🔳 🖉 🛞 📙 🔚 🗙 🖻	९ ⇔ ⇒ ≅ ⊼ ३ 🚍 🗏 ९ ९	Q, 11			
ip.addr == 192,168,2,105 and tcp	p.port == 6653				Expression +
No Time	Source	Destination	Protocol	Length Info	
- 1 14:24:39.550017	192.168.2.102	192,168,2,105	TCP	66 36430 → 6653 [SV	N] Seg=0 Win=8192 Len=_
2 14:24:39.550353	192.168.2.105	192.168.2.102	TCP	60 6653 → 36430 [SY	N, ACK] Seq=0 Ack=1 Wi
3 14:24:39.550400	192.168.2.102	192.168.2.105	TCP	54 36430 → 6653 [AC	K] Seq=1 Ack=1 Win=642
6 14:24:40.383669	192.168.2.102	192.168.2.105	TCP	57 [TCP segment of	a reassembled PDU]
7 14:24:40.384033	192.168.2.105	192.168.2.102	TCP	60 6653 → 36430 [AC	K] Seq=1 Ack=4 Win=512
8 14:24:41.445670	192.168.2.102	192.168.2.105	TCP	57 [TCP segment of	a reassembled PDU]
9 14:24:41.446025	192.168.2.105	192.168.2.102	TCP	60 6653 → 36430 [AC	K] Seq=1 Ack=7 Win=512
10 14:24:41.446518	192.168.2.105	192.168.2.102	TCP	337 [TCP segment of	a reassembled PDU]
11 14:24:41.497010	192.168.2.105	192.168.2.102	TCP	833 [TCP segment of	a reassembled PDU]
12 14:24:41.497056	192.168.2.102	192.168.2.105	TCP	54 36430 → 6653 [AC	K] Seq=7 Ack=1063 Win=
13 14:24:42.805613	192.168.2.102	192.168.2.105	ТСР	54 36430 → 6653 [F1	N, ACK] Seq=7 Ack=1063
14 14:24:42.806054	192.168.2.105	192.168.2.102	TCP	60 [ICP Dup ACK 9#1] 6653 → 36430 [ALK] S.
15 14:24:42.000401	192.100.2.100	192.100.2.102	TCP	54 36430 × 6653 [AC	N, ACK Seq=1005 ACK-7_
17 14:24:42.800427	192.168.2.102	192.168 2 105	TCP	54 JULY Patransmiss	ion] 36430 > 6653 [ETN
18 14:24:43 106490	192 168 2 105	192 168 2 102	TCP	60 6653 → 36430 [RS	T ACK1 Seg=1064 Ack=7
Ename 1: 66 hytes on wir	e (528 hits) 66 hutes & 0000 0	0 04 33 00 00 02 9	4 de 80	6c 23 88 08 00 45 00	1# F
> Ethernet II Spc: Giga-B	vt 6c-23-88 (94-de-80- 0010 0	0 34 4d ab 40 00 8	0 06 00	00 c0 a8 02 66 c0 a8	.4M.@f
✓ Internet Protocol Version	n 4. Src: 192.168.2.10	2 69 8e 4e 19 fd 2	3 07 76	41 00 00 00 00 80 02	.i.N#. vA
0100 = Version: 4	4 0030 2	0 00 86 46 00 00 0	2 04 05	6 b4 01 03 03 08 01 01	F
0101 = Header Ler	ngth: 20 bytes 0040 0	4 02			
> Differentiated Service	es Field: 0x00 (DSCP: C y				
0 7	[P	ackets: 18 · Displayed: 16 (88.9%)) Profile: Default

Figure 2: Two buffer transfer with 50 msec delay

We filter the Wireshark output to only monitor traffic that contains the IPv4 address of our PIC32 board server "192.168.2.105", and with a destination port of 6653. The first two packets sent originate from our client computer when it attempts to connect to the PIC32 board. The clients attempts a connection and the server responds with an ACK stating that connection is established. The client computer

also responds with an ACK as well to confirm the connection. Following the successful connection we see packets 6 and 7 in the image shown above show the packets sent when attempting a *'reset'* of the connection. Again the client computer sends a reset command to the server and the server responds with an ACK response notifying it received the full message correctly.

Packet 9 is the request from the command client to send the message. Recall before we split our full 1061 byte message into two pieces dependent on my birth month (April). The code for splitting the message up into two buffers is shown Code Snippet 2. We expect our two messages to be split into the following lengths:

$$tlen1 = tlen * \left(\frac{BM}{15}\right) = 1061 * \left(\frac{4}{15}\right) = 282$$
(1)
$$tlen2 = tlen - tlen1 = 1061 - 282 = 779$$
(2)

The client sends a message to the server prefixed with the first four bytes as: "02 84". This sequence of numbers signifies to the server to start our full message transfer spanning two datagrams. Looking at our Wireshark capture we can see the indeed that the response from the server is split into two datagrams. The first datagram seen in packet 10 is 337 bytes in length and the second datagram is 833 bytes in length. The lengths are longer than expect since we also have to account for the IPv4 and TCP overhead headers. Analyzing the contents of these messages confirm our message is being sent across two datagrams.

*Ethernet (not port 3389)													-		×
Eile Edit View Go Capture Analyze Statistics Telephony	<u>Wireles</u>	s <u>T</u> oo	ls <u>H</u> e	elp											
	ΘΘ		ī												
			-									(17)			1.4
ip.addr == 192.168.2.105 and tcp.port == 6653						_						X I	<u> </u>	Expression.	. +
No. Time Source	Destinatio	n	Prot	ocol	Lengt	h Ir	nfo								^
9 14 192.168.2.105	192.168	.2.102	TCP		(50 6	653 -	→ 3643	0 [AC	:K] S	eq=1 A	ck=7 Win:	=512	Len=0	_
10 14 192.168.2.105	192.168	.2.102	TCP		33	37 [TCP :	segmer	t of	a ne	assemb	led PDU]			- =
11 14 192.168.2.105	192.168	.2.102	тср		83	33 [TCP	segmer	t of	a rei	assemb	led PDU]			~
Ename 10: 337 bytes on wine (2696 bits) 337 byte	0000	94 de	80.60	23	88.00	04	a3	00 00	92.0	8 88	45 00	1#		F	_
Ethernet II. Scc: Microchi 00:00:02 (00:04:a3:00:	0010	01 43	46 ab	00	00 64	06	88	ea ce	a8 6	2 69	c0 a8	.CF	.d.	i	
Internet Protocol Version 4, Src: 192.168.2.105	0020	02 66	19 fd	8e	4e 00) Øf	42	48 23	07 7	6 48	50 18	.fI	N E	BH#.vHP.	
0100 = Version: 4	0030	02 00	64 ea	00	00 54	43	50	2f 49	50 2	0 28	54 72	d	TC I	P/IP (Tr	
0101 = Header Length: 20 bytes	0040	61 6e	20 50	69	/3 73 6f 74	69 67	61	6f 6c	43 6 2f 4	07 6e	74 72	ansmis ol Pro	ssi (on Contr	
> Differentiated Services Field: 0x00 (DSCP: CS0	0050	72 Ge	65 74	20	50 72	6f	74	6f 63	6f 6	ic 29	20 69	cnet F	Pro 1	tocol) i	
Total Length: 323	0070	73 20	74 68	65	20 62	61	73	69 63	20 2	0 63	6f 6d	s the	ba :	sic com	
Identification: 0x46ab (18091)	0080	6d 75	6e 69	63	61 74	69	6f	6e 20	6c 6	i1 6e	67 75	munica	ati d	on langu	
> Flags: 0x00	0090	61 67	65 20	6f	72 20	70	72	6f 74	6f 6	63 6f	6c 20	age or	r p i	rotocol	
Fragment offset: 0	00a0	6t 66	20 74	68	65 20	49	6e	74 65	72 6	ie 65	74 2e	of the	eIr	nternet.	
Time to live: 100	0000	75 73	65 64	20	61 73	20	61	20 63	6f 6	id 6d	75 6e	used a	as a	a commun	
Protocol: TCP (6)	00d0	69 63	61 74	69	6f 6e	73	20	70 72	6f 7	4 6f	63 6f	icatio	ons	protoco	
> Header checksum: 0x88ea [validation disabled]	00e0	6c 20	69 Ge	20	61 20	70	72	69 76	61 7	4 65	20 Ge	l in a	арі	rivate n	
Source: 192.168.2.105	00f0	65 74	77 6f	72	6b 20	28	65	69 74	68 6	5 72	20 61	etworl	k (e	either a	
Destination: 192.168.2.102	0100	6e 20	69 6e	74	72 61	. 6e	65	74 20	6f 7	2 20	61 6e	n intr	ran e	et or an	
[Source GeoIP: Unknown]	0110	20 65	78 74 6f 75	20	61 72	65	20	29 ZE	74 2	0 75	5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5	extra VOU	ane 1 are	set un	
[Destination GeoIP: Unknown]	0130	77 69	74 68	20	64 69	72	65	63 74	20 6	1 63	63 65	with a	dir e	ect acce	
Source Dorts 6652	0140	73 73	20 74	6f	20 74	68	65	20 49	6e 7	4 65	72 6e	ss to	th e	e Intern	
Destination Port: 36430	0150	65										e			
[Stream index: 0]															
[TCP Segment Len: 283]															
Sequence number: 1 (relative sequence numbe															
[Next sequence number: 284 (relative sequen															
Acknowledgment number: 7 (relative ack numb															
Header Length: 20 bytes															
> Flags: 0x018 (PSH, ACK)															
Window size value: 512															
[Calculated window size: 512]															
[Window size scaling factor: -2 (no window sca															
> Checksum: 0x64ea [validation disabled]															
Urgent pointer: 0															
> [SEQ/ACK analysis]															
ice segment data (203 bytes)															
0 2							Pa	ckets: 1	8 · Disp	layed:	16 (88.9	9%)		Profile: Def	ault 🔡

Figure 3: Two buffer transfer Packet 10

	*Ethernet (not port 3389)												-		×
E.	- Edit View, Co. Conture Archael Statistics Talantamu	Minutes	т		Uala										
En	e Edit view do Capture Analyze Statistics Telephony	wreless	0.7	-	Heib										_
		સ્વ	Q. 1	Ē											
	ip.addr == 192.168.2.105 and tcp.port == 6653												$\times \rightarrow$	Expression.	. +
No.	Time Source D	estination		Pr	otocol	Ler	ngth	Info							^
	9 14 192.168.2.105 1	92.168.	2.102	: то	CP		60	6653	→ 364	30 [AC	K] Seq	=1 Ack	<pre>c=7 Win=51</pre>	2 Len=0	
	10 14 192.168.2.105 1	92.168.	2.102	: то	CP		337	[TCP	segme	nt of	a reas	semble	ed PDU]		_
	11 14 192.168.2.105 1	92.168.	2.102	2 T	CP		833	[TCP	segme	nt of	a reas	semble	ed PDU]		
		0000									00.4		14		-
2	Frame 11: 833 bytes on wire (6664 bits), 833 byte	0000	94 de 33 33	46	oc 23 ad 00	00	64 6	14 d. 16 SI	5 60 00 5 f8 c0	0 02 0	5 00 4: 7 69 ct	5 00 9 a8	3F d	·····E·	
Ľ.	Ethernet 11, Src: Microchi_00:00:02 (00:04:a3:00:	0020	82 66	19 1	fd 8e	4e	00 0)f 4	3 63 2	3 07 7	5 48 50	0 18	.fN	Cc#.vHP.	
Ť	Internet Protocol Version 4, Src: 192.168.2.105,	0030	02 00	74 9	9d 00	00	74 2	c 20	3 79 6	F 75 7	2 20 6	3 6f	tt.	your co	
	0100 = Version: 4	0040	5d 70	75	74 65	72	20 6	i 7	3 20 70	72 6	f 76 6	9 64	mputer i	s provid	
	0101 = Header Length: 20 bytes	0050	65 64	20	77 69	74	68 2	20 63	1 20 63	3 6f 7	0 79 20	0 6f	ed with	a copy o	
	> Differentiated Services Field: 0x00 (DSCP: CS0	0060	66 20	74 (68 65	20	54 4	13 50	0 2f 49	9 50 2	3 70 7	2 6f	f the TC	P/IP pro	
	The sife state of the second	0070	57 72	61 (6d 20	6a	75 7	3 74	1 20 6	1732	0 65 7	6 65	gram jus	t as eve	
	Identification: 0x46ad (18093)	0080	72 79	20 1	6T /4	- 68	05 /	2 2	8 63 61 5 75 00	r 60 /	0 /5 /4	4 65	ry other	compute	
	> Flags: 0x00	0090	5 6e	64	20 64	65	73 7	3 6	1 67 69	5 73 2	2 74 6	6 75 F 20	end mess	ares to	
	Fragment offset: 0	00b0	6f 72	20 1	67 65	74	20 6	i9 6	66 61	F 72 6	1 61 74	4 69	or get i	nformati	
	Time to live: 100	00c0	6f 6e	20	66 72	6f	6d 2	0 6	L 6c 7	3 6f 2	0 68 6	1 73	on from	also has	
	Protocol: TCP (6)	00d0	20 61	20 (63 6f	70	79 2	0 61	F 66 20	9 54 4	3 50 2	f 49	a copy	of TCP/I	
	> Header checksum: 0x86†8 [validation disabled]	00e0	50 2e	20	54 43	50	2f 4	19 50	3 20 69	9732	0 61 20	0 74	P. TCP/I	P is a t	
	Source: 192.168.2.105	00f0	77 6f	2d (6c 61	79	65 7	2 20	3 70 73	2 6f 6	7 72 6	1 6d	wo-layer	program	
	Destination: 192.168.2.102	0100	2e 20	54	68 65	20	68 6	9 6	7 68 6	5 72 2	0 6c 6	1 79	. The hi	gher lay	
	[Source GeoIP: Unknown]	0110	55 /2	20 1	20 54	72	61 6	e /:	5 60 65	9 / 3 /	3 69 6	T 66	er, Iran	Smission	
	[Destination GeoIP: Unknown]	0120	20 45 6c 2c	20 1	6d 61	6e	61 6	7 6	5 73 20	2 01 7	8 65 20	9 61	1 manag	es the a	
×	Transmission Control Protocol, Src Port: 6653 (66	0140	73 73	65 (6d 62	60	69 6	ie 6	7 20 6	F 66 2	0 61 20	0 6d	ssemblin	gofam	
	Source Port: 6653	0150	65 73	73 (61 67	65	20 6	if 7	2 20 60	5 69 6	c 65 20	0 69	essage o	r file i	
	Destination Port: 36430	0160	6e 74	6f :	20 73	6d	61 6	ic 6	65 73	2 20 7	0 61 6	3 6b	nto smal	ler pack	
	Stream index: 0	0170	65 74	73 3	20 74	68	61 7	4 20	0 61 73	2 65 2	3 74 7	2 61	ets that	are tra	
	[TCP Segment Len: 779]	0180	6e 73	6d (69 74	74	65 6	54 20	0 6f 70	5 65 7	2 20 7	4 68	nsmitted	over th	
	Sequence number: 284 (relative sequence num	0190	55 20	49 (6e 74	65	72 6	ie 6	5 74 20	0 61 6 0 61 0	e 64 20	072	e Intern	et and r	
	[Next sequence number: 1063 (relative seque	0140 01b0	20 60	61 .	70 65	72	20 7	10 0.	2 /9 20	1 20 7	2 54 4	1 73	laver t	bat reas	
	Acknowledgment number: / (relative ack numb	01c0	73 65	6d (62 6c	65	73 2	0 7	1 68 6	5 20 7	2 65 6. 2 61 6	3 6b	sembles	the nack	
	Header Length: 20 bytes	01d0	65 74	73 :	20 69	6e	74 6	of 20	3 74 68	3 65 2	0 6f 7	2 69	ets into	the ori	
	> Flags: 0x018 (PSH, ACK)	01e0	67 69	6e (61 Go	20	6d 6	5 7	3 73 63	L 67 6	5 2e 20	0 54	ginal me	ssage. T	
	Window size value: 512	01f0	68 65	20 (6c 6f	77	65 7	2 20	3 6c 61	L 79 6	5 72 2	c 20	he lower	layer,	
	[Calculated window size: 512]	0200	49 6e	74 (65 72	6e	65 7	4 20	9 50 73	2 6f 7	4 6f 6	3 6f	Internet	Protoco	
	[window size scaling factor: -2 (no window sca	0210	5c 2c	20 (68 61	6e	64 6	ic 6	5 73 20	74 6	6 65 20	0 61	I, handl	es the a	
	> Checksum: 0x/49d [validation disabled]	0220	04 64 c1 62	22 1	20 70	61	20 /	10 63 16 61	L /2 /4	+ 20 6	F 06 20	4 69	aaress p	art of e	
	Urgent pointer: 0	0230	51 74	20 1	20 70 69 74	20	67 6	5 74	1 73 20	9750 9746	f 20 74	4 68	at it ge	ts to th	
	> [SEQ/ACK analysis]	0250	65 20	72	69 67	68	74 2	0 64	4 65 7	3 74 6	9 6e 6	1 74	e right	destinat	
	TCP segment data (779 bytes)	0260	69 6f	6e 3	2e 20	45	61 6	53 61	8 20 6	7 61 7	4 65 7	7 61	ion. Eac	h gatewa	~
C	2							P	ackets: 1	18 · Displ	ayed: 16	(88.9%	6)	Profile: Def	ault

Figure 4: Two buffer transfer Packet 11

Refering back to Figure 2 we can see the timestamps between packet 10 and 11 equate to a 50.492 msec delay between the two messages. What happens when

we decrease that time length to 0 msecs? Referring back to Code Snippet 1 we simply set the *DelayMsec* function parameter to 0.

*Ethernet (not port 3389)						– 🗆 X
<u>File Edit View Go Captur</u>	e <u>A</u> nalyze <u>S</u> tatistics Telephon <u>y</u> <u>W</u> ireless	<u>T</u> ools <u>H</u> elp				
🧵 🔳 🖉 💿 📙 🛅 🗙 🖻	९ ⇔ ⇒ ≅ î ½ 📃 🗏 ९, ९	2 11				
ip.addr == 192.168.2.105 and to	p.port == 6653					Expression +
No. Time	Source	Destination	Protocol	Length I	Info	
74 14:52:59.725052	192.168.2.102	192.168.2.105	тср	66 4	40017 → 6653	[SYN] Seq=0 Win=8192 Len=
75 14:52:59.725437	192.168.2.105	192.168.2.102	TCP	60 6	6653 → 40017	[SYN, ACK] Seq=0 Ack=1 Wi
76 14:52:59.725479	192.168.2.102	192.168.2.105	TCP	54 4	40017 → 6653	[ACK] Seq=1 Ack=1 Win=642
81 14:53:01.137403	192.168.2.102	192.168.2.105	TCP	57 [[ICP segment	of a reassembled PDU
82 14:53:01.137645	102.168.2.105	192.168.2.102	TCP	227 [6653 → 40017	[ACK] Seq=1 ACK=4 Win=512_
84 14:53:01.138365	192.100.2.105	192.100.2.102	TCP	227 [833 [[TCP segment	of a reassembled PDU]
85 14:53:01 138825	192 168 2 102	192.168 2 105	TCP	54.4	40017 + 6653	[ACK] Seg=4 Ack=1063 Win=
	-don (2554 bits) 933 b (2000 64	da 80 67 73 88 6	29.04 a		20 00 00 45 0	a 14 5 14
> Ethernet II, Src: Micro	chi 00:00:02 (00:04:a3: 0010 03	33 00 09 00 00 0	54 06 cc	1 9c c0 a	a8 02 69 c0 a	8 .3di
✓ Internet Protocol Versi	on 4, Src: 192.168.2.10 0020 02	66 19 fd 9c 51 0	00 0f 43	3 5d 94 d	dc b4 f5 50 1	8 .fQ C]P.
0100 = Version:	4 0030 02	00 b6 1d 00 00 7	4 2c 20) 79 6f 7	75 72 20 63 6	ft, your co
0101 = Header Le	ength: 20 bytes 0040 6d	/0 75 74 65 72 2	0 69 73	5 20 70 7	/2 61 76 69 6	4 mputer is provid f ad with a copy o
> Differentiated Servi	ces Field: 0x00 (DSCP: C V 0050 65	20 74 68 65 20 5	64 43 56	20 05 0 21 49 5	50 20 70 72 6	f f the TC P/IP pro
0 2			P	ackets: 96 ·	Displayed: 8 (8.3	%) Profile: Default

Figure 5: Two buffer transfer with 0 msec delay

The figure above shows the Wireshark capture when the delay between the two message is set to zero. In this capture packet 83 and 84 show the transmission of our message. We see no ill effects to decreasing the time between messages to zero. The time delay as seen in the Wireshark capture shows the sever sent the message with a difference of 0.0465 msec delay.

Now we experiment with increasing the delay to outside the millisecond range to see if transmission will still complete. We start by increasing the delay to 5 secs.

📕 *Et	hernet							-		×
<u>File</u>	<u>E</u> dit <u>V</u> iew <u>G</u> o <u>C</u> apture	Analyze Statistics Telephony	<u>Wireless</u>	ols <u>H</u> elp						
4	/ @ 📃 🔚 🔀 🛅	9 @ @ @ \Lambda 🔲 📃	0.0.0	R.R.						
in libra	dor == 192.168.2.105 and tcp	.port == 6653							_ Expression	
No.	Time	Source		Destination	Protoco	Length In	ifo			
	17 15:15:54.128989	192.168.2.102		192.168.2.105	TCP	66 4	3535 → 6653	[SYN] Seq=0	Win=8192	Len
	18 15:15:54.129407	192.168.2.105		192.168.2.102	TCP	60 6	653 → 43535	[SYN, ACK]	Seq=0 Ack	=1 W
	19 15:15:54.129452	192.168.2.102		192.168.2.105	TCP	54 4	3535 → 6653	[ACK] Seq=1	Ack=1 Wi	n=64
	30 15:15:55.718673	192.168.2.102		192.168.2.105	TCP	57 [TCP segment	of a reasse	mbled PDU	1
	31 15:15:55.719026	192.168.2.105		192.168.2.102	TCP	60 6	553 → 43535	[ACK] Seq=1	Ack=4 W1	n=51
	32 15:15:55.719672	192.168.2.105		192.168.2.102	TCP	337 [ICP segment	ot a reasse	mbled PDU	1
	33 15:15:55.769907	192.168.2.102		192.168.2.105	TCP	54 4	3535 → 6653	[ACK] Seq=4	Ack=284 1	Win=
	49 15:16:00.720055	192.168.2.105		192.168.2.102	TCP	833 [ICP segment	ot a reasse	mbled PUU	1
	50 15:16:00.769881	192.168.2.102		192.168.2.105	TCP	54 4	55555 → 66555	[ACK] Seq=4	ACK=1005	w1n
	50 15:10:05.774477	192.168.2.105		192.168.2.102		50 [TCP Keep-AL	1Vej 6653 →	43535 [PSI 25 - 6652	۸۰۰ A
-	57 15:16:05.774505	192.168.2.102		192.100.2.105	TCP	54 [тер кеер-ат.	IVE ACK] 455	55 + 6655	[AC
> Fra	me 17: 66 bytes on wir	re (528 bits), 66 bytes c	0000 00 0	1 a3 00 00 02 9	94 de 80	0 6c 23 88	08 00 45 00	· · · · · · · · · ·	.1#E.	
> Eth	ernet II, Src: Giga-By	/t_6c:23:88 (94:de:80:6c:…	0010 00 3	4 58 12 40 00 8	30 06 0	0 00 c0 a8	02 66 c0 a8	3 .4X.@	f	
> Int	ernet Protocol Version	1 4, Src: 192.168.2.102, …	0020 02 6	aa 0T 19 fd 0 9 gg 4g 99 99 9	10 0/ 1	5 90 00 00 5 54 61 63	00 00 80 02	· .1	•••••	
> Tra	insmission Control Prot	tocol, Src Port: 43535 (4…	0050 20 0	, 00 40 00 00 0)	2 04 0	0 04 01 03	00 00 01 01			
			0040 04 0.	-						
0 7	wirechark prappa DE94404	5-600D-44ED-80AE-00023784E-00A 20	0160220151552	03440		ackete: 70 · Di	colaved: 11/15	7%)	Profile: Doi	fault
	wiresnark_pcaprig_0564406	0-0390/23/D0FC0A_2	0100220151552_6	00000	1 1	ackets, 70 ° DI	spiayeu: 11 (15.	/ /0]	Fronte: Del	auit

Figure 6: Two buffer transfer with 5 sec delay

In this transmission we see the second message being sent in the Wireshark capture with the expected 5 second delay. It appears that after the first message buffer was send (Packet 9) the client eventually timed out and send an ACK message back to the server. The server was still waiting to send the second message however so after 5 seconds the server finally sends the second half the message. A separate ACK message is seen being sent back from the client after this second buffers is received. A tabulated result of the successful two buffer message transfers with arbitrary delay lengths is provided below.

Coded Server Delay Between Buffers	Wireshark Observed Delay	Two Buffers Successfully Sent?		
0 msec	0.0465 msec	Yes		
25 msec	25.636 msec	Yes		
50 msec	50.492 msec	Yes		
500 msec	500.333 msec	Yes		
1000 msec	1000.174 msec	Yes		
2500 msec	2500.508 msec	Yes		
5000 msec	5000.383 msec	Yes		

Table	1:	Two	buffer	transfer	with	Arbitrar	v Dela	vs
rubio		1 110	Danoi	ti anoioi	vvici i	/ u Midiai j	Doid	<i>y</i> 0

From the results observe red it does not appear there is a limit to the delay between message buffers. Transmission propagation time needs to be account for however to ensure the client does not request a re-transfer of a message that is still in transmission. For example, on a Gigabit Ethernet physical link between a server and client the propagation time can be expected to be in the millisecond range. However in satellite communications propagation times should be expected to be in the seconds range. In the former scenario a client should be programed so it waits for a longer time between transmissions.

Three Buffered Message Transmission

Next we repeat the experiment complete before but with our message split into three separate buffers. As before our message lengths are parsed based on my birth month/day of April 6th.

$$tlen1 = tlen * \left(\frac{BM}{25}\right) = 1061 * \left(\frac{4}{25}\right) = 169$$
(3)
$$tlen2 = tlen * \left(\frac{BD}{100}\right) = 1061 * \left(\frac{6}{100}\right) = 63$$
(4)
$$tlen3 = tlen - tlen1 - tlen2 = 1061 - 169 - 63 = 829$$
(5)

Parsing code is seen in Code Snippet 3. Sending the message is the same as two message buffer send but with an extra delay and send command added (see Code Snippet 2).

Again we start with a delay of a default of 50 msecs.

*Ethernet	- 🗆 X
<u>File Edit View Go Capture Analyze Statistics Telephon</u>	<u>/ W</u> ireless <u>T</u> ools <u>H</u> elp
◢ ■ ∅ ◉ 📙 🗅 🗙 🖻 🔍 ⇔ ⇔ 🕾 🛪 🌡 🧮	0, 0, 0, II
ip.addr == 192.168.2.102 and tcp.port == 6653	Expression +
No. Time Source	Destination Protocol Length Info
5 22:15:46.339258 192.168.2.102	192.168.2.105 TCP 66 56661 → 6653 [SYN] Seq=0 Wi…
6 22:15:46.339616 192.168.2.105	192.168.2.102 TCP 60 6653 → 56661 [SYN, ACK] Seq
7 22:15:46.339673 192.168.2.102	192.168.2.105 TCP 54 56661 → 6653 [ACK] Seq=1 Ac
8 22:15:47.256001 192.168.2.102	192.168.2.105 TCP 57 [TCP segment of a reassembl
9 22:15:47.256360 192.168.2.105	192.168.2.102 TCP 60 6653 → 56661 [ACK] Seq=1 Ac
10 22:15:47.256940 192.168.2.105	192.168.2.102 TCP 224 [TCP segment of a reassembl
11 22:15:47.307029 192.168.2.105	192.168.2.102 TCP 118 [TCP segment of a reassembl
12 22:15:47.307076 192.168.2.102	192.168.2.105 TCP 54 56661 + 6653 [ACK] Seq=4 Ac
13 22:15:47.357433 192.168.2.105	192.168.2.102 TCP 884 [TCP segment of a reassembl
- 14 22:15:47.407619 192.166.2.102	192.100.2.105 ICP 54 50001 + 0055 [ACK] Seq=4 AC
> Frame 10: 224 bytes on wire (1792 bits), 224 byte	0000 94 de 80 6c 23 88 00 04 a3 00 00 02 08 00 45 001#E.
> Ethernet II, Src: Microchi_00:00:02 (00:04:a3:00:	0010 00 d2 00 17 00 00 64 06 cf ef c0 a8 02 69 c0 a8di.
> Internet Protocol Version 4, Src: 192.168.2.105,	0020 02 66 19 Td dd 55 00 0T 42 43 65 94 1T 13 50 18 .TU. BLP.
> Transmission Control Protocol, Src Port: 6653 (66	0040 61 6e 73 6d 69 73 73 69 6f 6e 20 43 6f 6e 74 72 ansmissi on Contr
	0050 6f 6c 20 50 72 6f 74 6f 63 6f 6c 2f 49 6e 74 65 ol Proto col/Inte
	0060 72 6e 65 74 20 50 72 6f 74 6f 63 6f 6c 29 20 69 rnet Pro tocol) i
	0070 73 20 74 68 65 20 62 61 73 69 63 20 20 63 6f 6d s the ba sic com
	0080 6d 75 6e 69 63 61 74 69 6f 6e 20 6c 61 6e 67 75 municati on langu
	0030 01 07 05 20 01 72 20 70 72 01 74 67 65 65 70 20 age or p rotocol
	00b0 20 49 74 20 63 61 6e 20 61 6c 73 6f 20 62 65 20 It can also be
	00c0 75 73 65 64 20 61 73 20 61 20 63 6f 6d 6d 75 6e used as a commun
	00d0 69 63 61 74 69 6f 6e 73 20 70 72 6f 74 6f 63 00 ications protoc.
0 2	Packets: 15 · Displayed: 10 (66.7%) Profile: Default

Figure 7: Three buffer transfer with 50 msec delay

Our first message is 224 bytes in length which again includes the overhead of the TCP and IPv4 headers. We can see that our actual message is contained in the packet if we view the ASCII print out at the bottom of the Wireshark output. The second byte occurs approx. 50 msecs after a length of 118 bytes. Last we see the reminder of our message sent in Packet 13. The Visual Basic application (as seen in Figure 8) confirms our last message was sent with a length of 830 bytes (829+1 with '\0' termination).

ECE4532 Client1 v14	_		×
Server IP 192.168.2.105 Port 6653 Client IP 192.168.2.102	Connect Close		
Reset Transfer User1	User2		
Console Transfer Completed Bytes Transfered	110 121 117 97	^	
830	101	~	

Figure 8: VB out: Three buffer transfer with 50 msec delay

Interestingly enough the client sends an ACK (Packet 12) back to the server after the transmission of the second message. An ACK message was not sent back after the transmission of Packets 10 and 11. We now investigate the effect of delaying the time between messages to see if we can determine why this behavior occurs.

Coded Server Delay	Wireshark Ob	served Delay	Two Buffers
Between Buffers	Packet 1-2	Packet 2-3	Successfully Sent?
0 msec	0.482 msec	0.335 msec	Yes
25 msec	25.149 msec	25.573 msec	Yes
50 msec	50.106 msec	50.508 msec	Yes
500 msec	449.746 msec	500.502 msec	Yes
1000 msec	1000.372 msec	1000.411 msec	Yes
2500 msec	2500.126 msec	2500.497 msec	Yes
5000 msec	5000.118 msec	5000.605 msec	Yes

Table 2: Three buffer transfer with Arbitrary Del

Again at any delay interval the message will still eventually reach the client. The Visual Basic Application updates itself to reflect the latest message that has been

received. Also of note is that at values greater than 50 msecs we always see an ACK message between packet 1, 2 and 3. The Wireshark capture below shows the a experiment with a delay of 55 msecs. Note the ACKs between messages.

*Ethernet		– 🗆 ×
Eile Edit View Go Capture Analyze Statistics Telephony Wi	reless <u>T</u> ools <u>H</u> elp	
∡ ■ ∅ ⊛ 📙 📇 🕱 😋 ۹. ⇔ ⇔ 🕾 🗿 💆 🚍 🗐 ۹.	Q. Q. 11	
ip.addr == 192.168.2.102 and tcp.port == 6653		Expression
No. Time Time d/dt Source	Destination	Protocol Length Info
5 23:1 0.000000 192.168.2.102	192.168.2.105	TCP 54 64798 → 6653 [FIN, ACK] Seq=1 A
6 23:1 0.000339 192.168.2.105	192.168.2.102	TCP 60 6653 → 64798 [RST, ACK] Seq=1 A
141 23:1 7.563950 192.168.2.102	192.168.2.105	TCP 66 64981 → 6653 [SYN] Seq=0 Win=81
142 23:1 0.000359 192.168.2.105	192.168.2.102	TCP 60 6653 → 64981 [SYN, ACK] Seq=0 A
143 23:1 0.000051 192.168.2.102	192.168.2.105	TCP 54 64981 → 6653 [ACK] Seq=1 Ack=1
165 23:1 2.153337 192.168.2.102	192.168.2.105	TCP 57 [TCP segment of a reassembled P
166 23:1 0.000355 192.168.2.105	192.168.2.102	TCP 60 6653 → 64981 [ACK] Seq=1 Ack=4
167 23:1 0.000482 192.168.2.105	192.168.2.102	TCP 224 [TCP segment of a reassembled P
168 23:1 0.050996 192.168.2.102	192.168.2.105	TCP 54 64981 → 6653 [ACK] Seq=4 Ack=17
169 23:1 0.004133 192.168.2.105	192.168.2.102	TCP 118 [TCP segment of a reassembled P
170 23:1 0.050229 192.168.2.102	192.168.2.105	TCP 54 64981 → 6653 [ACK] Seq=4 Ack=23
1/1 23:1 0.005237 192.168.2.105	192.168.2.102	ICP 884 [ICP segment of a reassembled P
> Frame 169: 118 bytes on wire (944 bits), 118 bytes ca.	0000 94 de 80 6c 23 88 00 04	a3 00 00 02 08 00 45 001#E.
> Ethernet II. Src: Microchi 00:00:02 (00:04:a3:00:00:0	0010 00 68 00 0a 00 00 64 06	d0 66 c0 a8 02 69 c0 a8 .hdfi
> Internet Protocol Version 4, Src: 192.168.2.105, Dst:	0020 02 66 19 fd fd d5 00 0f	42 ec ab 21 c2 50 50 18 .f B!.PP.
> Transmission Control Protocol, Src Port: 6653 (6653),	0030 02 00 a0 4e 00 00 6f 6c	20 69 6e 20 61 20 70 72Nol in a pr
	0040 69 76 61 74 65 20 66 65	74 77 6T 72 6D 20 28 65 1Vate ne twork (e
	0060 74 00 67 22 0 61 6e 20 0060 74 20 6f 72 20 61 6e 20 0070 29 2e 20 57 68 00	20 09 06 74 72 01 06 09 Iller an Intranet 65 78 74 72 61 66 65 74 tor an extranet), Wh.
0 7		Packets: 178 · Displayed: 13 (7.3%) Profile: Default

Figure 9: Three buffer transfer with 55 msec delay

Multiple 50 Byte Buffered Message Transmission

For our final experiment we run our server so that it sends our message parsed up into 50 byte chunks. The code for parsing the data into 50 byte sizes is seen in Code Snippet 4. Note we congregate the data upon running sending it instead of declaring 20 char arrays to store our message into the 50 bytes chunks. We use one 50 byte chunk *'tbfr1'* and transfer data into it using *'memcpy'*. We keep track of how many bytes we have sent thus far in a new int variable declared as *'bytesSent'*. The copying of data into different buffers induces a overhead between each transmission.

For our baseline experiment we run our server with a 50 msec delay between each message sent. The resulting Wireshark capture is shown below.

*Ethernet			– 🗆 X
<u>File Edit View Go Capture Analyze Statistics Telephony Wi</u>	reless <u>T</u> ools <u>H</u> elp		
🛋 🔳 🔬 📵 📙 🛅 🗙 🖆 🔍 👄 🗢 🕾 🗿 🛃 🚍 🗨	Q Q 1		
ip.addr == 192.168.2.102 and tcp.port == 6653			Expression +
No. Time Time d/dt Source	Destination	Protocol Lengt	h Info
147 23:3 0.000000 192.168.2.102	192.168.2.105	TCP	66 5759 → 6653 [SYN] Seq=0 Win=819
148 23:3 0.000354 192.168.2.105	192.168.2.102	TCP	60 6653 → 5759 [SYN, ACK] Seq=0 Ac.
149 23:3 0.000049 192.168.2.102	192.168.2.105	TCP	54 5759 → 6653 [ACK] Seq=1 Ack=1 W.
150 23:3 1.140249 192.168.2.102	192.168.2.105	TCP	57 [TCP segment of a reassembled P.
151 23:3 0.000377 192.168.2.105	192.168.2.102	TCP	60 6653 → 5759 [ACK] Seq=1 Ack=4 W.
152 23:3 0.000437 192.168.2.105	192.168.2.102	TCP	105 [ICP segment of a reassembled P.
153 23:3 0.050113 192.168.2.102	192.168.2.105	TCP	54 5/59 → 6055 [ACK] Seq=4 ACK=52
155 23:3	192.168.2.105	TCP	54 5759 → 6653 [ACK] Seg=4 Ack=103
156 23:3 0.000364 192.168.2.105	192.168.2.102	TCP	105 [TCP segment of a reassembled P.
157 23:3 0.049682 192.168.2.102	192.168.2.105	TCP	54 5759 → 6653 [ACK] Seg=4 Ack=154
158 23:3 0.000355 192.168.2.105	192.168.2.102	TCP	105 [TCP segment of a reassembled P.
159 23:3 0.050237 192.168.2.102	192.168.2.105	TCP	54 5759 → 6653 [ACK] Seq=4 Ack=205
160 23:3 0.000141 192.168.2.105	192.168.2.102	TCP	105 [TCP segment of a reassembled P.
161 23:3 0.050421 192.168.2.105	192.168.2.102	TCP	105 [TCP segment of a reassembled P.
162 23:3 0.000043 192.168.2.102	192.168.2.105	TCP	54 5759 → 6653 [ACK] Seq=4 Ack=307.
163 23:3 0.049745 192.168.2.105	192.168.2.102	TCP	105 [TCP segment of a reassembled P.
164 23:3 0.050471 192.168.2.105	192.168.2.102	TCP	105 [TCP segment of a reassembled P.
165 23:3 0.000043 192.168.2.102	192.168.2.105	TCP	54 5759 → 6653 [ACK] Seq=4 Ack=409.
183 23:3 5.752254 192.168.2.105	192.168.2.102	TCP	60 [TCP Keep-Alive] 6653 → 5759 [P.
■ 184 23:3 0.000026 192.168.2.102	192.168.2.105	TCP	54 [ICP Keep-Alive ACK] 5759 → 665.
> Frame 152: 105 bytes on wire (840 bits), 105 bytes ca. > Ethernet II, Src: Microchi, 00:00:02 (00:04:a3:00:00:a. > Internet Protocol Version 4. Src: 192, 168,2.168, Det	0000 94 de 80 6c 23 88 00 04 0010 00 5b 00 08 00 00 64 06 0020 02 66 19 fd 16 7f 00 0f	a3 00 00 02 08 0 d0 75 c0 a8 02 6 42 42 eb 63 83 c	0 45 001#E. 9 c0 a8 .[d. ui 5 50 18 .fB.c.P.
> Transmission Control Protocol, Src Port: 6653 (6653),	0030 02 00 4f 29 00 00 54 43	50 2f 49 50 20 2	8 54 720)TC P/IP (Tr
	0040 b1 be /3 bd b9 /3 /3 b9 0050 6f 6c 20 50 72 6f 74 6f	63 6f 6c 2f 49 6	e 74 72 ansmissi on contr e 74 65 ol Proto col/Inte
	0050 6t 6c 20 50 72 6t 74 6t 0060 72 6e 65 74 20 50 72 6f	63 61 6C 21 49 6 00	74 65 ol Proto col/Inte rnet Pro .
		Packets: 184 · Disr	Profile: Default

Figure 10: Multiple buffer transfer with 50 msec delay

The completion time between messages still stays around the expected 50 msecs. The overhead induced by copying the message during the send operation is negligible.

By parsing up the message into 50 byte blocks we expect to see around 22 messages being sent to the client. However we seem to have reached a limitation in the amount of packets a client will accept. After 8 consecutive message sends from the server we see no more messages in transmission. The limitation is not due to the delay in between message sends either. We experimented with sending messages with varying delays from 0 msecs to 5000 msecs and the same behavior is observed.

*Ethernet			– 🗆 X
<u>File Edit View Go Capture Analyze Statistics Telephony Wi</u>	reless <u>T</u> ools <u>H</u> elp		
🔟 🗏 🖉 📙 🔚 🗙 🗳 🍳 🗢 🕾 🕾 🖉 🗶 📃 🗨	Q, Q, II		
ip.addr == 192.168.2.102 and tcp.port == 6653			Expression +
No. Time Time d/dt Source	Destination	Protocol Len	gth Info ^
12 00:1 0.050467 192.168.2.102	192.168.2.105	TCP	54 11995 → 6653 [ACK] Seq=4 Ack
25 00:1 4.949618 192.168.2.105	192.168.2.102	TCP	105 [TCP segment of a reassemble
26 00:1 0.050474 192.168.2.102	192.168.2.105	TCP	54 11995 → 6653 [ACK] Seq=4 Ack
39 00:1 4.949713 192.168.2.105	192.168.2.102	TCP	105 [TCP segment of a reassemble…
42 00:1 0.050060 192.168.2.102	192.168.2.105	TCP	54 11995 → 6653 [ACK] Seq=4 Ack
47 00:1 4.950198 192.168.2.105	192.168.2.102	TCP	105 [TCP segment of a reassemble
48 00:1 0.049937 192.168.2.102	192.168.2.105	TCP	54 11995 → 6653 [ACK] Seq=4 Ack
58 00:1 4.950162 192.168.2.105	192.168.2.102	TCP	105 [ICP segment of a reassemble
59 00:1 0.000559 192.168.2.102	192.108.2.105	TCP	54 11995 → 0055 [ACK] Seq=4 ACK
67 00:1 0 050521 102 168 2 102	102.100.2.102	TCP	E4 1100E + 66E2 [ACK] Soged Ack
70 00.1 / 040720 102 168 2 105	192.168.2.103	TCP	105 [TCD segment of a reassemble
71 00.1 0 050095 192 168 2 102	192.168.2.102	TCP	54 11995 + 6653 [ACK] Segue Ack
81 00:1 4.950142 192.168.2.105	192.168.2.102	TCP	105 [TCP segment of a reassemble.
82 00:1 0.049750 192.168.2.102	192.168.2.105	TCP	54 11995 → 6653 [ACK] Seg=4 Ack
151 00:1 78,459468 192,168,2,105	192,168,2,102	TCP	105 [TCP Spurious Retransmission
152 00:1 0.000001 192.168.2.105			105 [TCP Spurious Retransmission
153 00:1 0.000000 192.168.2.105			105 [TCP Spurious Retransmission
154 00:1 0.000000 192.168.2.105			105 [TCP Spurious Retransmission
155 00:1 0.000000 192.168.2.105			
156 00:1 0.000001 192.168.2.105			
157 00:1 0.000000 192.168.2.105			
158 00:1 0.000030 192.168.2.102			
159 00:1 0.000019 192.168.2.102	192.168.2.105		54 [TCP Dup ACK 82#2] 11995 → 6
160 00:1 0.000011 192.168.2.102	192.168.2.105	тср	54 [TCP Dup ACK 82#3] 11995 → 6
Ename 160: 54 bytes on wire (432 bits), 54 bytes cant	0000 00 04 a3 00 00 02 94 de	80 6c 23 88 08	00 45 00
Ethernet II. Src: Giga-Byt 6c:23:88 (94:de:80:6c:23:8	0010 00 28 22 84 00 00 80 06	00 00 c0 a8 02	66 c0 a8 .("f
> Internet Protocol Version 4, Src: 192.168.2.102, Dst:	0020 02 69 2e db 19 fd f4 3a	93 5f 00 0f 43	da 50 10 .i:C.P.
> Transmission Control Protocol, Src Port: 11995 (11995	0030 f9 58 86 3a 00 00		.x.:
0 7		Packets: 185 · D	isplayed: 43 (23,2%) Profile: Default
V 2		, octets, 105 D	Profile: Default

Figure 11: Multiple buffer transfer with 5000 msec delay

Behavior during the 5000 msec experiment showed the server responded to the client sending a "TCP Spurious Retransmission" or re-transmission of a previously sent frame [2]. From the Wireshark capture shown above, transmission packet 25 was sent and but then retransmitted in packet 151. The client responds to the server in packet 158 that what it received in packet 151 was a duplicate frame of packet 25.



Figure 12: Demonstration of Spurious Retransmission [2].

The server believes that packet 25 was lost in transmission so it re-transmits the original message to the client. The client later tells the server the TCP message is a duplicate. After this conversation between client and server the server terminates transmission of the rest of the message.

III. Conclusion

What we learned in this lab experiment was the process of sending data across multiple message buffers and the limitations that come with it. In our first experiment we looked at sending our ~1060 byte message across two buffers which worked without hiccup. The second experiment expanded our message

across three buffers and again the message was received by the client without error. Even when introducing long delays between each buffer we saw that the client was able to decipher the message. With the results from these experiments we can ensure that even if there is a long transmission propagation time we can expect each frame to be acknowledged by the client correctly.

Behavior during the multiple 50 bytes transfer was a bit more concerning. First it was important to have your socket options set so that the PIC32 board would not wait for a full message buffer to send your message. Specifying *'TCP_NODELAY'* in the socket options function ensured our message was not buffered. However still we were receiving transmission errors since our entire never was successful in reaching the client. The effect was not dependent on delays between message transmissions either. The server appears to be confused about what message the client received and attempted re-transmission of the packets 1-8 of our 50 byte blocked data. Further work needs to be done to understand why transmission did not complete successfully between client and server. After modifying various parameters on the server side socket operations behavior of this problem could not be fixed.

IV. Appendix

```
// If the received message starts with a second byte is
// '84' it signifies a initiate transfer
if(rbfr[1]==84){
    mPORTDSetBits(BIT_2); // LED3=1
    // Send full message
    //
    send(clientSock, tbfr1, tlen1+1, 0);
    DelayMsec(50);
    send(clientSock, tbfr2, tlen2+1, 0);
    mPORTDClearBits(BIT_2); // LED3=0
}
```

Code Snippet 1

```
// We store our desired transfer paragraph
11
char myStr[] = "TCP/IP (Transmission Control Protocol/Internet Protocol) is "
    "the basic communication language or protocol of the Internet.
    "It can also be used as a communications protocol in a private "
    "network (either an intranet or an extranet). When you are set up "
    "with direct access to the Internet, your computer is provided
    "with a copy of the TCP/IP program just as every other computer '
    "that you may send messages to or get information from also has "
    "a copy of TCP/IP. TCP/IP is a two-layer program. The higher "
    "layer, Transmission Control Protocol, manages the assembling "
    "of a message or file into smaller packets that are transmitted "
    "over the Internet and received by a TCP layer that reassembles "
    "the packets into the original message. The lower layer, "
    "Internet Protocol, handles the address part of each packet so "
    "that it gets to the right destination. Each gateway computer on "
    "the network checks this address to see where to forward the "
    "message. Even though some packets from the same message are "
    "routed differently than others, they'll be reassembled at the "
    "destination.\0";
// Store my birthmonth
11
int BM = 4;
// Chunk up our data
11
tlen = strlen(myStr) + 1;
tlen1 = tlen*BM/15;
tlen2 = tlen - tlen1;
char tbfr1[tlen1 + 1];
char tbfr2[tlen2 + 1];
// Chunk up data into desired lengths
11
memcpy(tbfr1, myStr, tlen1);
memcpy(tbfr2, myStr+tlen1, tlen2);
// Null terminate the string array
11
tbfr1[tlen1] = '\0';
tbfr2[tlen2] = '\0';
```

Code Snippet 2

```
// Store my birthmonth
11
int BM = 4;
int BD = 6;
// Chunk up our data
11
tlen = strlen(myStr) + 1;
tlen1 = tlen*BM/25;
tlen2 = tlen*BD/100;
tlen3 = tlen - tlen2 - tlen1;
char tbfr1[tlen1 + 1];
char tbfr2[tlen2 + 1];
char tbfr3[tlen3 + 1];
// Chunk up data into desired lengths
11
memcpy(tbfr1, myStr, tlen1);
memcpy(tbfr2, myStr+tlen1, tlen2);
memcpy(tbfr3, myStr+tlen1+tlen2, tlen3);
// Null terminate the string array
11
tbfr1[tlen1] = '\0';
tbfr2[tlen2] = '\0';
tbfr3[tlen3] = '\0';
```

Code Snippet 3

```
if(rbfr[1]==84){
    mPORTDSetBits(BIT_2); // LED3=1
    bytesSent = 0;
    //sent = 0;
    while (bytesSent < tlen){</pre>
        memcpy(tbfr1, myStr+bytesSent, tlen1);
        if (bytesSent > 1049){
            tbfr1[tlen-bytesSent+1] = '\0';
            send(clientSock, tbfr1, tlen-bytesSent+1, 0);
        }
        else{
            tbfr1[tlen1] = '\0';
            // Loop until we send the full message
            11
            send(clientSock, tbfr1, tlen1+1, 0);
       }
        bytesSent += tlen1;
        DelayMsec(100);
    }
    mPORTDClearBits(BIT_2); // LED3=0
}
mPORTDClearBits(BIT_0); // LED1=0
```

Code Snippet 4

Full raw source code available upon request. Email devin.trejo@temple.edu.

V. References

- [1 D. FITZGERALD and D. WAKABAYASHI, "Apple Quietly Builds New
- Networks," Wall Street Journal, 3 February 2014. [Online]. Available: http://www.wsj.com/news/articles/SB10001424052702304851104579361201 655365302. [Accessed 21 February 2016].
- [2 J. BONGERTZ, "Spurious Retransmissions," 6 June 2013. [Online].
-] Available: https://blog.packet-foo.com/2013/06/spuriousretransmissions/comment-page-1/. [Accessed 21 February 2016].
- $\ensuremath{\left[3\ensuremath{\,W.}\ensuremath{\,Stallings}\xspace, Data and Computer Communications, Person Education Inc. ,$
-] 2014.