

Hannah Shaw

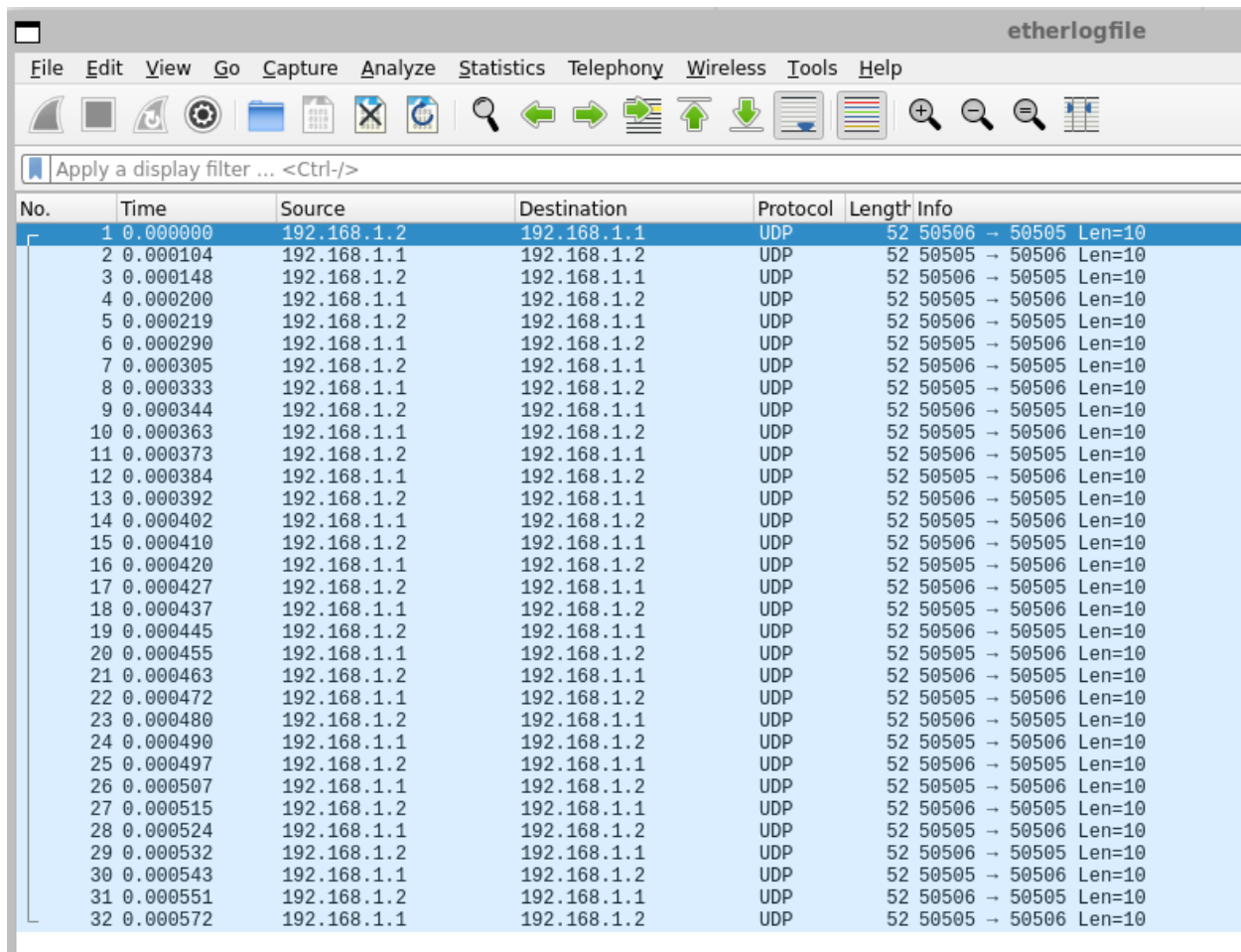
Shaw288

CS536 – Lab3

Write-up PDF

V1

The output of packet sniffing with the lab2 ping app (v1/etherlogfile:



No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	192.168.1.2	192.168.1.1	UDP	52	50506 → 50505 Len=10
2	0.000104	192.168.1.1	192.168.1.2	UDP	52	50505 → 50506 Len=10
3	0.000148	192.168.1.2	192.168.1.1	UDP	52	50506 → 50505 Len=10
4	0.000200	192.168.1.1	192.168.1.2	UDP	52	50505 → 50506 Len=10
5	0.000219	192.168.1.2	192.168.1.1	UDP	52	50506 → 50505 Len=10
6	0.000290	192.168.1.1	192.168.1.2	UDP	52	50505 → 50506 Len=10
7	0.000305	192.168.1.2	192.168.1.1	UDP	52	50506 → 50505 Len=10
8	0.000333	192.168.1.1	192.168.1.2	UDP	52	50505 → 50506 Len=10
9	0.000344	192.168.1.2	192.168.1.1	UDP	52	50506 → 50505 Len=10
10	0.000363	192.168.1.1	192.168.1.2	UDP	52	50505 → 50506 Len=10
11	0.000373	192.168.1.2	192.168.1.1	UDP	52	50506 → 50505 Len=10
12	0.000384	192.168.1.1	192.168.1.2	UDP	52	50505 → 50506 Len=10
13	0.000392	192.168.1.2	192.168.1.1	UDP	52	50506 → 50505 Len=10
14	0.000402	192.168.1.1	192.168.1.2	UDP	52	50505 → 50506 Len=10
15	0.000410	192.168.1.2	192.168.1.1	UDP	52	50506 → 50505 Len=10
16	0.000420	192.168.1.1	192.168.1.2	UDP	52	50505 → 50506 Len=10
17	0.000427	192.168.1.2	192.168.1.1	UDP	52	50506 → 50505 Len=10
18	0.000437	192.168.1.1	192.168.1.2	UDP	52	50505 → 50506 Len=10
19	0.000445	192.168.1.2	192.168.1.1	UDP	52	50506 → 50505 Len=10
20	0.000455	192.168.1.1	192.168.1.2	UDP	52	50505 → 50506 Len=10
21	0.000463	192.168.1.2	192.168.1.1	UDP	52	50506 → 50505 Len=10
22	0.000472	192.168.1.1	192.168.1.2	UDP	52	50505 → 50506 Len=10
23	0.000480	192.168.1.2	192.168.1.1	UDP	52	50506 → 50505 Len=10
24	0.000490	192.168.1.1	192.168.1.2	UDP	52	50505 → 50506 Len=10
25	0.000497	192.168.1.2	192.168.1.1	UDP	52	50506 → 50505 Len=10
26	0.000507	192.168.1.1	192.168.1.2	UDP	52	50505 → 50506 Len=10
27	0.000515	192.168.1.2	192.168.1.1	UDP	52	50506 → 50505 Len=10
28	0.000524	192.168.1.1	192.168.1.2	UDP	52	50505 → 50506 Len=10
29	0.000532	192.168.1.2	192.168.1.1	UDP	52	50506 → 50505 Len=10
30	0.000543	192.168.1.1	192.168.1.2	UDP	52	50505 → 50506 Len=10
31	0.000551	192.168.1.2	192.168.1.1	UDP	52	50506 → 50505 Len=10
32	0.000572	192.168.1.1	192.168.1.2	UDP	52	50505 → 50506 Len=10

Alternating source and destination ip address representing the server and clients ip addresses as the ping packets and their responses are sent back and forth.

The wireshark output of packet sniffing UDP packets from my file transfer (v1/etherlogfile2):

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	fe80::c97:93ff...	ff02::2	ICMPv6	70	Router Solicitation from 0e:97:93:db:1d:45
2	37.358808	192.168.1.1	192.168.1.2	UDP	56	33830 → 50555 Len=14
3	37.358845	192.168.1.2	192.168.1.1	ICMPv6	84	Destination unreachable (Port unreachable)
4	42.496192	62:e6:b7:08:06...	0e:97:93:db:1d...	ARP	42	Who has 192.168.1.2? Tell 192.168.1.1
5	42.496175	0e:97:93:db:1d...	62:e6:b7:08:06...	ARP	42	Who has 192.168.1.1? Tell 192.168.1.2
6	42.496232	62:e6:b7:08:06...	0e:97:93:db:1d...	ARP	42	192.168.1.1 is at 62:e6:b7:08:06:7f
7	42.496239	0e:97:93:db:1d...	62:e6:b7:08:06...	ARP	42	192.168.1.2 is at 0e:97:93:db:1d:45
8	43.124351	192.168.1.1	192.168.1.2	UDP	56	44623 → 50555 Len=14
9	43.124619	192.168.1.2	192.168.1.1	UDP	48	50555 → 44623 Len=6
10	43.130562	192.168.1.1	192.168.1.2	UDP	61	44623 → 50555 Len=19
11	43.130682	192.168.1.2	192.168.1.1	UDP	46	50555 → 44623 Len=4
12	44.130956	192.168.1.1	192.168.1.2	UDP	61	44623 → 50555 Len=19
13	44.131157	192.168.1.2	192.168.1.1	UDP	46	50555 → 44623 Len=4
14	45.131582	192.168.1.1	192.168.1.2	UDP	61	44623 → 50555 Len=19
15	45.131783	192.168.1.2	192.168.1.1	UDP	46	50555 → 44623 Len=4
16	46.132085	192.168.1.1	192.168.1.2	UDP	61	44623 → 50555 Len=19
17	46.132277	192.168.1.2	192.168.1.1	UDP	46	50555 → 44623 Len=4
18	47.132709	192.168.1.1	192.168.1.2	UDP	61	44623 → 50555 Len=19
19	47.132916	192.168.1.2	192.168.1.1	UDP	46	50555 → 44623 Len=4
20	48.133376	192.168.1.1	192.168.1.2	UDP	61	44623 → 50555 Len=19
21	48.133574	192.168.1.2	192.168.1.1	UDP	46	50555 → 44623 Len=4
22	49.133888	192.168.1.1	192.168.1.2	UDP	61	44623 → 50555 Len=19
23	49.134065	192.168.1.2	192.168.1.1	UDP	46	50555 → 44623 Len=4
24	50.134398	192.168.1.1	192.168.1.2	UDP	61	44623 → 50555 Len=19
25	50.134589	192.168.1.2	192.168.1.1	UDP	46	50555 → 44623 Len=4
26	51.135229	192.168.1.1	192.168.1.2	UDP	61	44623 → 50555 Len=19
27	51.135433	192.168.1.2	192.168.1.1	UDP	46	50555 → 44623 Len=4
28	52.136029	192.168.1.1	192.168.1.2	UDP	61	44623 → 50555 Len=19
29	52.136234	192.168.1.2	192.168.1.1	UDP	46	50555 → 44623 Len=4
30	53.136595	192.168.1.1	192.168.1.2	UDP	61	44623 → 50555 Len=19
31	53.136796	192.168.1.2	192.168.1.1	UDP	46	50555 → 44623 Len=4
32	54.137167	192.168.1.1	192.168.1.2	UDP	61	44623 → 50555 Len=19

Looking at the header bytes shown in hexadecimal and translated by wireshark, we see the sender and receiver's IP addresses. The port numbers match the port numbers I specified when I ran it, as shown in screensnip below where the "c5 7b" represent the port number 50555:

▼	User Datagram Protocol, Src Port: 44623, Dst Port: 50555
	Source Port: 44623
	Destination Port: 50555
	Length: 27
	Checksum: 0x8380 [unverified]
	[Checksum Status: Unverified]
	[Stream index: 1]
0000	0e 97 93 db 1d 45 62 e6 b7 08 06 7f 08 00 45 00
0010	00 2f b6 30 40 00 40 11 01 3a c0 a8 01 01 c0 a8
0020	01 02 ae 4f c5 7b 00 1b 83 80 05 00 00 00 6f 6e
0030	20 6f 66 20 76 65 74 68 30 20 61 74 20

Similarly, there are bytes in the ip header bytes that show the sender and recievers IP address, with the corresponding raw hex bytes shown highlighted below:

header checksum: 0x013a [validation disabled]																
[Header checksum status: Unverified]																
Source Address: 192.168.1.1																
Destination Address: 192.168.1.2																
0000	0e	97	93	db	1d	45	62	e6	b7	08	06	7f	08	00	45 00Eb.....E.
0010	00	2f	b6	30	40	00	40	11	01	3a	c0	a8	01	01	c0 a8	./..@..@.:.....
0020	01	02	ae	4f	c5	7b	00	1b	83	80	05	00	00	00	6f 6e	...0{.....on
0030	20	6f	66	20	76	65	74	68	30	20	61	74	20			of veth 0 at

The following bytes were in the application layer payload, they were the bytes I transmitted:

00 00 00 00 00 68 61 6e 6e 61 68 20 66 61 69 74 68 20 73 68

This is significant in my application because it represents the packet I sent, where the first four bytes are all zeros, because this is sequence number1, then the remaining bytes translate to a skewed spelling of my name since the file I sent was mostly junk me typing random words quickly. the exact translation is “Hannah fai th sh” which is close to my full name.

V2

The issue of asynchrony can be handled with shared memory. Making the data structure that kept track of the acks received made it possible for updating and reading of the data structure at any point. Overwriting previous writes is a non issue if implemented correctly, since switching the state of a packet to received multiple times isn't a problem.

Receiver logic for sending ack of metadata

By putting both the ack transmission and data rcvfrom in a while loop, ack for the metadata can be retransmitted five times before continuing to the next iteration of the while loop, which will start the program back to be looking for a new file to be transferred. But if a new data packet is received sufficiently quickly after the metadata ack is sent, then the receiver can continue on accepting data packets from the original sender.

While we artificially lengthen our applications latency with micropace, short payload sizes, artificial packet drops, etc, we can use relativity between runs to gauge performance. Since the lab computers are physically close and built for communication, we can expect the performance of our application to be considerably better than can be expected in a real world data communication. While increasing the size of the files being sent does increase the overall time of transmission, the increase is constant, due to the systems' reliability. As far as I could keep track of, few to no packets were dropped unless forced to do so with the lossmodel file. In general, the larger the files the more chance for errors to occur, which increases the number of retransmissions and extra transmission time, which did not play a role in my simulations.