Whisper Network (File) Transfer System

Department of Computer Science Engineering
University of Nevada, Reno

May 2, 2022

CPE 400.1001 Team
Joshua Dahl, Antonio Massa, Annette McDonough

Instructor
Dr. Engin Arslan

Table of Contents

Abstract

This project aims to create a network of peer-to-peer connected nodes that allow for the synchronization of a file folder across all connected peers. We implemented a system of connecting peers to each other and maintaining a peer connection list for each node. We also developed a system of messages between nodes that allow for locking, unlocking, modifying, and deleting files. This system allowed us to maintain the file contents and structure across the network.

Introduction

Networking infrastructures can be complex and exciting; one particularly unique networking infrastructure is the peer-to-peer network. In a peer-to-peer network, multiple peer nodes connect, acting both as a server providing information to the network and a client that consumes information from the network. We look at how a peer-to-peer whisper network can transfer files. Whisper Networks forward data between nodes so that not every node need be connected to every other node; alternatively fully connected networks connect every peer to every other peer. Whisper Networks should keep the number of peers low to ensure efficient information transfer. This paper will demonstrate why we chose our design decisions, how we handled errors in the network system, what problems we encountered, and an overall analysis of the network design. We will first go over the novelty of the overall project.

Protocol Explanation

We decided to base our implementation on the file transfer project and expanded upon its design by adding peer-to-peer functionality. Instead of sending a file between two nodes, our network ensures that a managed subset of the file system is synchronized between all connected peers. We chose to implement a Whisper Network instead of a fully connected peer-to-peer network. The overall design of the Whisper Network resembles that of a flattened spoke wheel instead of a web-like design featured in a fully connected network. The additional complexity in this design offered more of an exciting challenge, thus increasing our knowledge and understanding of network design.

We use the ZeroTier library for NAT traversal, allowing us to punch through firewalls, dynamic networks, and NAT. ZeroTier is a virtual network connection that can run almost anything on top of it. ZeroTier can also run on most platforms, making it desirable for our purposes. [Stephen Foskett Packrat, website]. The IP addresses used in the program are all virtual IP version 6 addresses used by ZeroTier to uniquely identify peers.

The peer-to-peer network we form between nodes is best thought of as an undirected acyclic graph, as seen in Figure 1 below. We rely on the fact that there is only one path between nodes in the network to simplify our routing algorithm. When a new node joins the network, the node it connects to becomes its “Gateway” to the rest of the network. The Gateway is essential since the new node (and its descendants) will become disconnected from the rest of the network if it goes offline. To prevent separate sub-networks from forming, the Gateway sends information about other backup nodes that it is aware of to connecting peers that they can use should network connectivity be disrupted. The default gateway and backup nodes for the network topology in Figure 1 can be seen in Table 1, in which we see the Node itself, its Gateway, and the backup peers keep track of all predecessors. If a node loses its gateway node, it can use its list of backup peers to try and reconnect to the rest of the network.

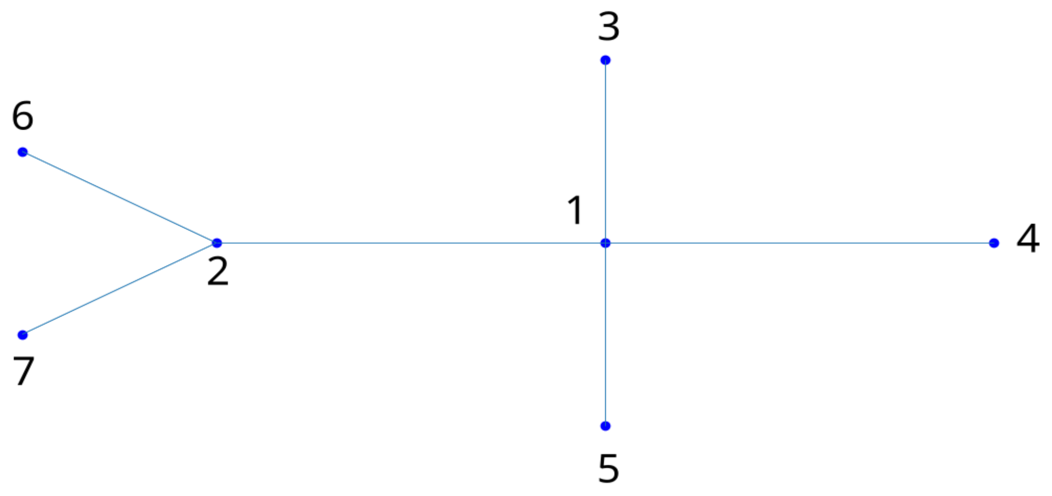
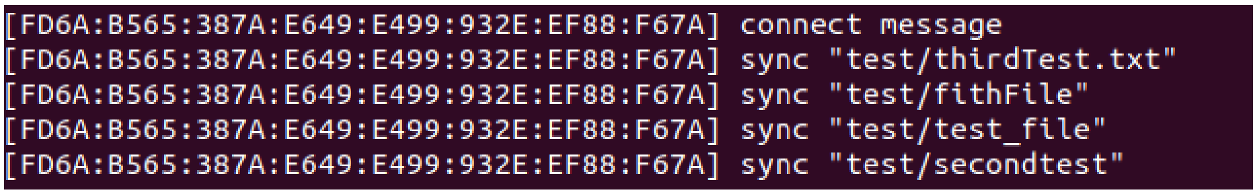


Figure 1: *An example topology our network might create. In this topology peer 1 acts as a gateway for peers 2, 3, 4, and 5. Likewise, peer 2 acts as the gateway for peers 6 and 7.*

Table 1: *Node connections and peers tracked in figure 1.*

Node	Gateway	Peers Tracked
1	-	-
2	1	-
3	1	2
4	1	2,3
5	1	2,3,4
6	2	1
7	2	1,6

The file system is synchronized across the entire network. When a new peer connects, it determines which folders the network is managing and deletes all files in those managed folders. It then waits until it has received all of the files the network is aware of from its Gateway, as shown in Figure 2 below. If a file is modified anywhere in the network, the node that modified the file sends out a lock message to its peers. This message puts the file in a read-only state for all other nodes in the network until the peer that requested the lock has gone ten seconds without modifying the file. Once the node detects that the file is no longer being modified, it releases its lock on the file by sending an unlock message. The unlock message lets all peer nodes know that the file is fully updated and is ready to be modified by other nodes in the network.



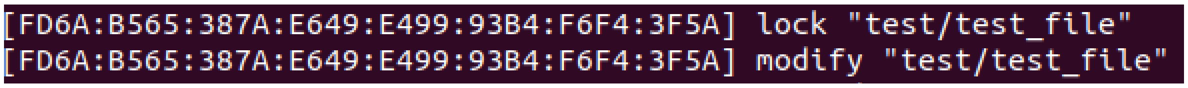
```
[FD6A:B565:387A:E649:E499:932E:EF88:F67A] connect message
[FD6A:B565:387A:E649:E499:932E:EF88:F67A] sync "test/thirdTest.txt"
[FD6A:B565:387A:E649:E499:932E:EF88:F67A] sync "test/fithFile"
[FD6A:B565:387A:E649:E499:932E:EF88:F67A] sync "test/test_file"
[FD6A:B565:387A:E649:E499:932E:EF88:F67A] sync "test/secondtest"
```

Figure 2: *This figure shows the initial synchronization messages. These messages carry the content of all the files the network is managing. Once a new node has received all of these files its file system should be synchronized with the rest of the network.*

Our routing algorithm between peers is similar to the Reverse Link routing algorithm for multicast messages. Packets store a tiny bit of routing information: destination peer, originator peer, and the peer that the packet was last received from. When a peer receives a message, it checks if the packet is destined for itself, and if it is, it can process the message. It then checks to see if the destination peer is directly connected. If so, it forwards the packet to only that peer; otherwise, it forwards the packet to all of its connected peers, except the packet from which it came. Most messages are broadcast to the whole network. Each peer processes the message and then forwards it. The rule preventing forwarding messages to the sending peer implies that the network can be considered as a directed acyclic network with all edges flowing away from the peer the message originates from; this topology ensures that (even while broadcasting) each peer will receive each message exactly once.

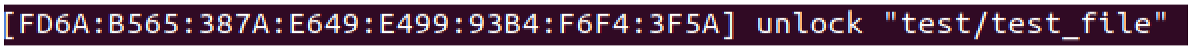
Each node maintains a separate thread for its connected peers, while the file system operations are consecutively executed on the main thread. This allows us to accept data from multiple peers simultaneously while ensuring that the file system is modified atomically. The use of multithreading for the peers improved the throughput of the network system. When a peer receives a message, it forwards the message to other connected peers (except the sender) as appropriate. It then adds the message to a priority queue for processing on the main thread.

The possibility of conflicts became evident when different peers could modify the same file simultaneously. To avoid these conflicts, we created a system to lock the files during modification so that only the peer holding the lock had control over the file. We can see an example of a peer locking a file and modifying it in figure 3 below. While one peer holds a lock on the file, no other peer can write to the file while the lock is held. We change the permissions of the file to ensure this is enforced. If a node receives multiple requests to lock a file, then the timestamps of the requests are compared, and the oldest one is kept as the lock holder. Only unlock messages from the lock holder are accepted, this is facilitated by each lock storing the virtual IP address of the node that holds the lock. Nodes will request the release of their held locks after a few seconds if they detect the locked file is no longer being modified. In figure 4, the message to unlock the file previously being modified is sent.



```
[FD6A:B565:387A:E649:E499:93B4:F6F4:3F5A] lock "test/test_file"
[FD6A:B565:387A:E649:E499:93B4:F6F4:3F5A] modify "test/test_file"
```

Figure 3: *This figure shows the lock and modify messages being sent. Here the file test_file in the folder test is being locked and then modified.*



```
[FD6A:B565:387A:E649:E499:93B4:F6F4:3F5A] unlock "test/test_file"
```

Figure 4: *This figure shows the unlock message being sent. Here the file test_file in the folder test is being unlocked.*

Our network is event-driven, with a priority queue to prioritize the messages that must be processed on the main thread. The order of priority for our messages is as follows: resend, connect, disconnect, lock/unlock, initial file synchronization, any other file operation, and finally, the lowest priority is an arbitrary payload message used for debugging. We attempted to use a SkipList-based lock-free priority queue; however, we encountered an issue with the priority queue losing a few packets. In the end, we wound up using a simple mutex to ensure that only one thread at a time is modifying the queue.

We implemented a file sweeper structure that sweeps the file system checking for created, modified, and deleted files. For every file that we track, if the file's sweep iteration does not match the current iteration, the file has been deleted and must be marked as deleted. If a file has not been modified in the last ten seconds, it gets removed from the fast-track file sweep. We implemented a fast-track sweeper, which can be used for a recently modified subset of files; this functionality is based on iterations with modified files being fast-tracked and then removed from the fast-track list after ten sweeps. The fast track defines locking and unlocking; when a file is fast-tracked, the peer locks it. When it is not fast-tracked, the peer unlocks it. We have a function that performs a total (fast track ignoring) sweep every ten sweeps. We run the sweeping algorithm once every second; thus, a total filesystem sweep occurs once every ten seconds.

When a file is modified we store a hashed version of the file’s contents. We discovered that timestamps on files can change unexpectedly and thus should not be relied on to determine when a file is modified. When both the timestamp and hash have been changed, the file has been modified. Next we will discuss some of the methods our application uses to handle errors.

Error Handling

If a node goes offline, then disconnection is detected by the peers using it as their Gateway. By monitoring the disconnectivity of a node, the next node in the network chain can be prompted to reconnect to the network, avoiding isolated subnetworks. If the disconnected node were a Gateway to several peers, the peers would reconnect to their Gateway’s Gateway.

Our system is designed to ensure data is transmitted with the correct size and with no corruption. Messages carry a hash, which is checked to ensure it was not altered during transit. If our hash does not match, we ask the message to be resent. The naive algorithm chosen to calculate hashes takes the ASCII value of each character in the string and sums them; after the strings are added, they are checked against each other to see if they are equal.

We implemented a monitor class that provides thread-safe wrappers to protect against race conditions between threads. It relies on a single mutex that is locked whenever the data is accessed; it thus allows any non-thread-safe data structure to be made thread-safe by locking access to its members. We support a shared mutex paradigm where multiple threads may take a lock promising only to read the data, but every thread must finish reading before a thread can modify the structure. Our list of peers is protected by this monitor mechanism, ensuring no race conditions occur as peers connect and disconnect.

We decided to lock the files to ensure we did not accidentally incur any network-wide race conditions. We accomplished the locking of files by changing their file permissions with the file system library. There are some possible cons when using file locking. One possible downside to file locking is that we may be waiting incessantly for a file to be unlocked. We avoid this issue by issuing an unlock message if a locked file has not been recently modified. While we designed an extensive system for handling errors, we still encountered bugs within the code. We will go over these in the next section.

Bugs Encountered and Design Discussion

There were several bugs found within the code during the testing process. The first being that when a node was connected to an existing network, it would fully synchronize with the file system. After the initial synchronization, its files would no longer be synchronized with any changes that may have happened on the rest of the network. It also did not send any synchronization messages to the gateway node. This created an issue where files across the network were no longer synchronized until the node reconnected. The gateway node never synchronized with any connecting peers and did not see any changes that may have happened to those peers.

This bug turned out to be quite simple; connection messages were not telling the rest of the network what to manage. Thus, their file sweeper had no files to sweep; providing this folder information to connecting peers fixed the problem.

The second major bug encountered was that the gateway node did not see any messages from connecting peer nodes. When a message was transmitted from the gateway peer, the second peer could see the changes, lock the file, and view the modification messages over the terminal. The bug occurred when the second peer tried to modify or delete a file; no lock appeared, and the file was never modified or deleted. Because the second peer could not modify files, it appeared to be a one-way connection instead of a two-way connection as desired.

This bug turned out to be quite subtle, our unlock code was broken, and thus permissions were never restored. Thus even once the gateway peer unlocked a file, its descendants could not modify it. Fixing the unlocking code fixed this bug.

The third critical bug occurred when a file was modified on the gateway node. When the file was saved, creating a new timestamp, the sweeper detected the change and sent out a "lock" message to lock the file on all peer nodes. It then sent out a "content" message which synchronized the file with all peer nodes. Once the file had not been modified within a determined amount of time, an "unlock" message was sent out to all nodes, and the file was unlocked, preparing it for further modification. The bug happened between the lock and modified messages. Since the file was now locked or placed into a read-only state, the file system discarded any modifications sent to the peers, as the file was in a read-only state.

This bug also had a very straightforward solution: we discard any modification or deletion message not originating from the lock holder and then temporarily add modification permissions to the file. Once we have modified the file, the modification permissions are removed again (nothing needs to be done to the deleted file). This solution relies on the prioritization of messages, since lock messages get processed before file operation messages we can be sure that messages originating from someone other than the current lock holder are invalid. We encountered many issues in the design of this project that we overcame, the next section will discuss some of our results.

Results

One of our primary design decisions was the method of choosing which of the backup peers to use. When a peer leaves the network, we take the backup gateway node from the front of the backup list. By taking from the front of the list and not the back, we maintained a hub and spoke topology; if we were to take a node from the back of the list, the result would be a more linear network topology.

Figure 5 below shows peer 2 connected to its Gateway. The peer node only tracks its gateway node at this point.

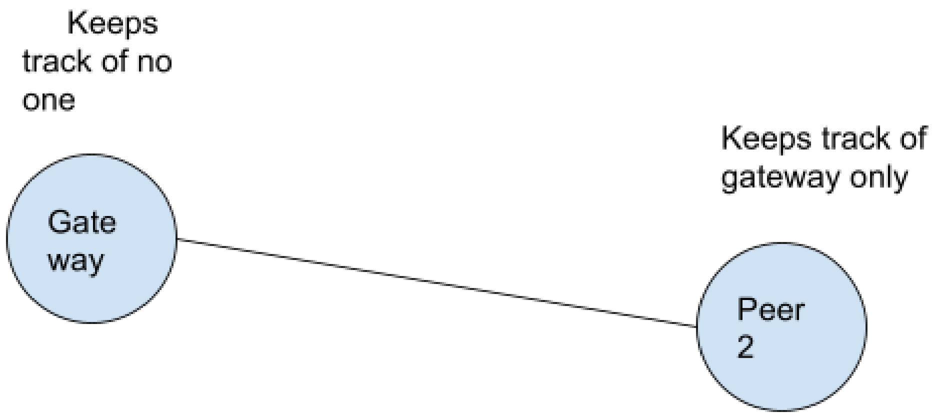


Figure 5: *This figure shows the gateway node connected to peer 2. The gateway node is the first node in the network then peer 2 joins the network. Peer 2 keeps track of its predecessor while the gateway tracks no one.*

Next, we see peer 3 connected to the network. Its Gateway is set as peer 2 and gets a list of backup nodes from it. In figure 6, we see that it gets peer 2's Gateway as its backup.

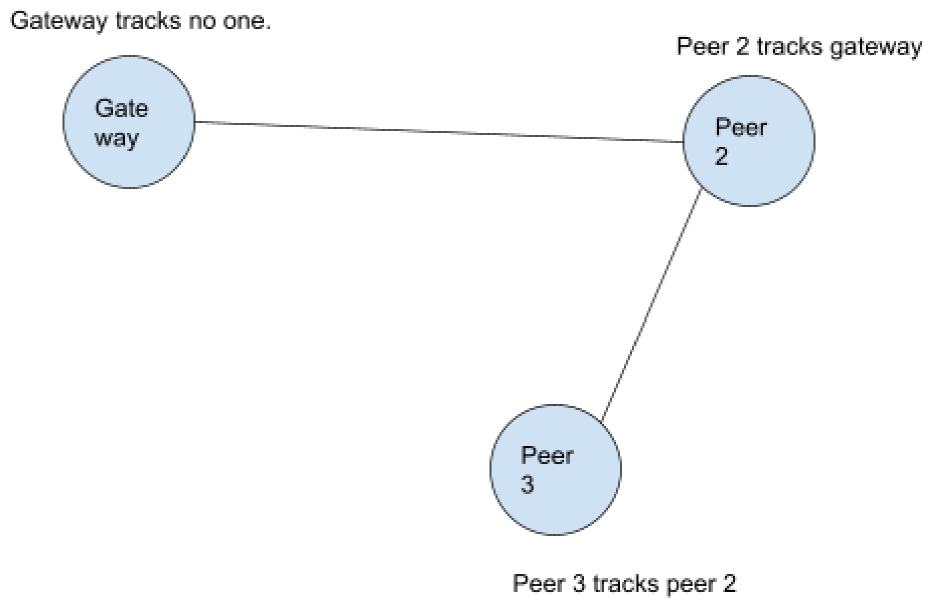


Figure 6: *This figure shows peer 3 making a new connection to peer 2. Peer 3 will now track connections to Peer 2 as well as the Gateway node.*

Now peer 4 connects to the network. Peer 4's Gateway is also set as peer 2 and it is given the list of peer 2's peers as backup peers. The Gateway is set as its first backup peer, and peer 3 is included last. The process is repeated in Figure 7. Peer 5 is connected to peer 2 as its Gateway. It then tracks the Gateway, peer 3, and peer 4 in its backup list.

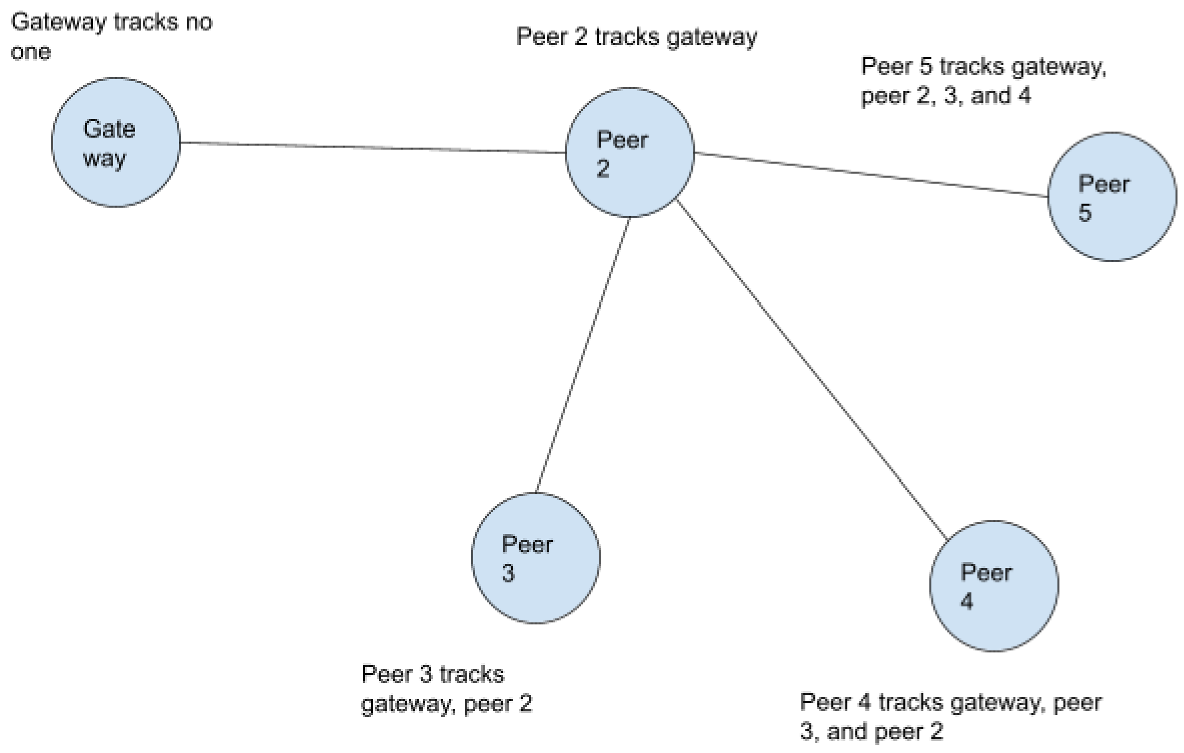

Gateway tracks no one
Gate way
Peer 2 tracks gateway
Peer 2
Peer 5 tracks gateway, peer 2, 3, and 4
Peer 5
Peer 3
Peer 4
Peer 3 tracks gateway, peer 2
Peer 4 tracks gateway, peer 3, and peer 2

Figure 7: *Here we have a fourth and a fifth peer connecting to the network. Each node will keep track of its predecessor in case of its gateway becoming lost or disconnected.*

As a final step in figure 8, peer 2 has disconnected from the network. Peer 3 uses its backup list to connect to peer 2's original gateway node. Peer 4 uses its backup list to connect to the original gateway peer. Furthermore, peer 5 uses its backup list to connect to the original gateway. If it could not connect to the original gateway peer, peer 5 would try to connect to peer 3 and then peer 4.

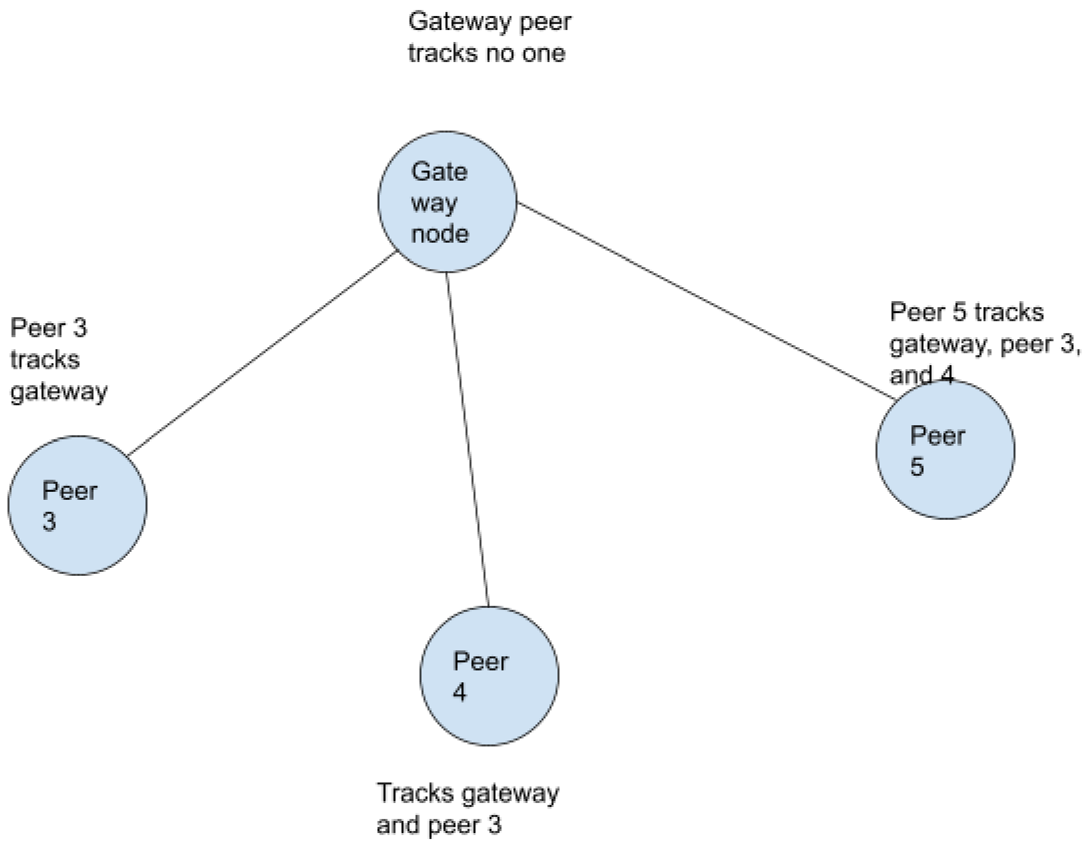

Gateway peer tracks no one
Gate way node
Peer 3 tracks gateway
Peer 5 tracks gateway, peer 3, and 4
Peer 3
Peer 5
Peer 4
Tracks gateway and peer 3

Figure 8: *Peer 2 left the network and peer 3 now tracks the gateway.*

Figure 9 shows the topology that would result if we had instead used the node at the back of the backup list. Peer 3 would have still connected to the original Gateway; however, peer 4 would now be connected to peer 3, and peer 5 would have connected to peer 4. The resulting topology is linear, meaning that a packet traveling from the gateway node to the peer 5 would need to be routed between peers three times. In the hub and spoke topology we chose to use, it would only need to be routed between peers twice.

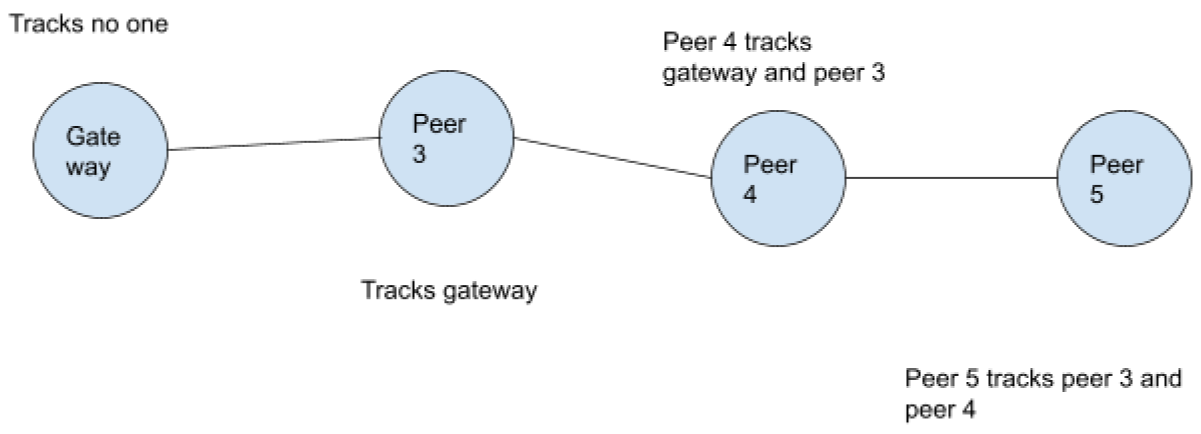

Tracks no one
Gate way
Peer 3
Tracks gateway
Peer 4 tracks gateway and peer 3
Peer 4
Peer 5
Peer 5 tracks peer 3 and peer 4

Figure 9: *This figure shows what a linear topology would look like after the nodes reestablish their connections.*

Our program must route and deserialize data it receives from the network before further processing. We ran two peers: Peer 1, a recent Ryzen 5 laptop, and peer 2, a shared university server. Peer 2 connected to peer 1, and we measured how long it took to route and deserialize the various messages they exchanged. This process was repeated ten times, with various results of time taken for different message types summarized in Figures 10 and 11.

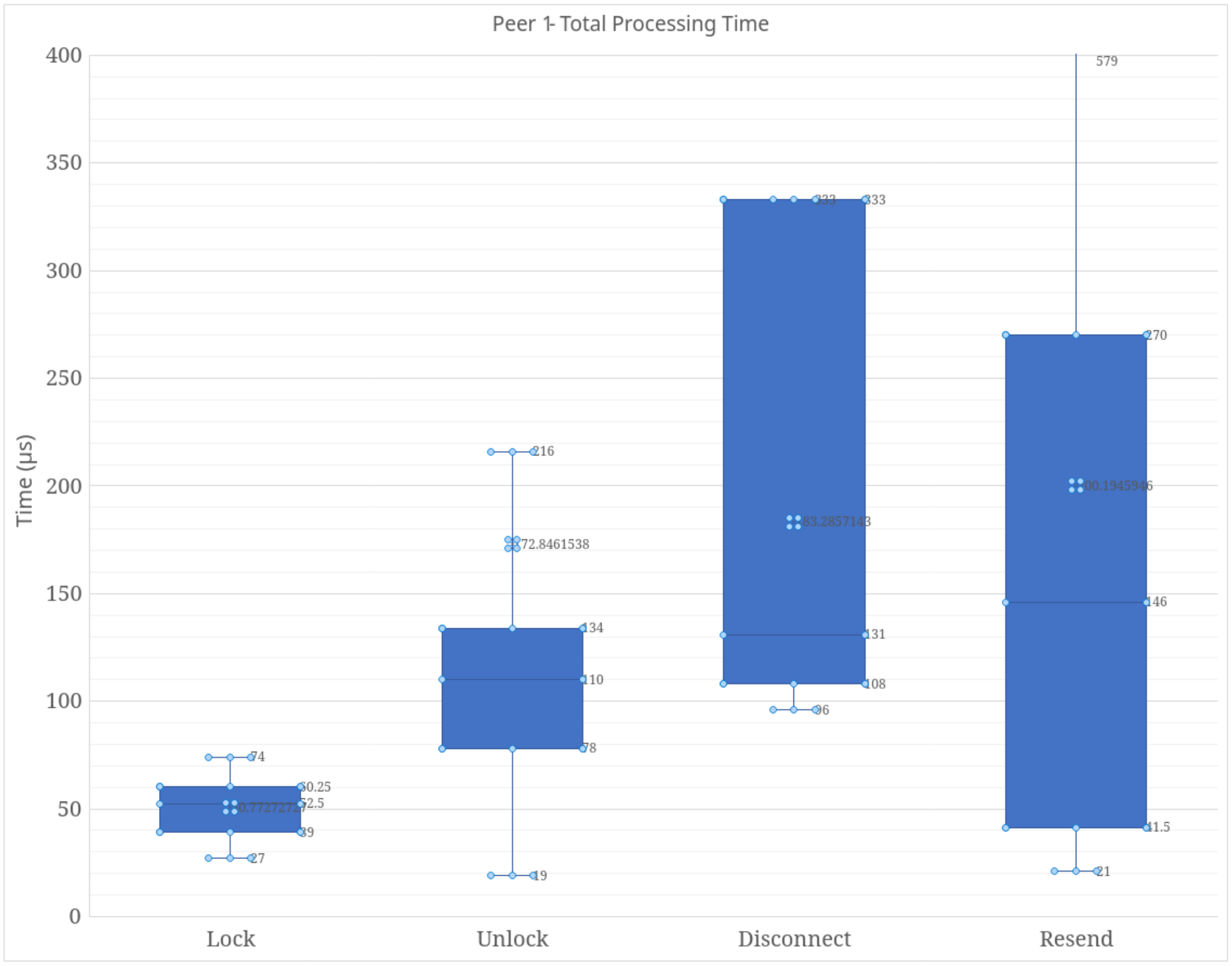


Figure 10: *Box and Whisker plots summarizing the total processing time of peer 1. The gateway node does not connect, thus we have no information about its connection time. Disconnect and resend messages take the longest time and have the largest variance in running time. No operation here (with one outlier) takes more than half a millisecond.*

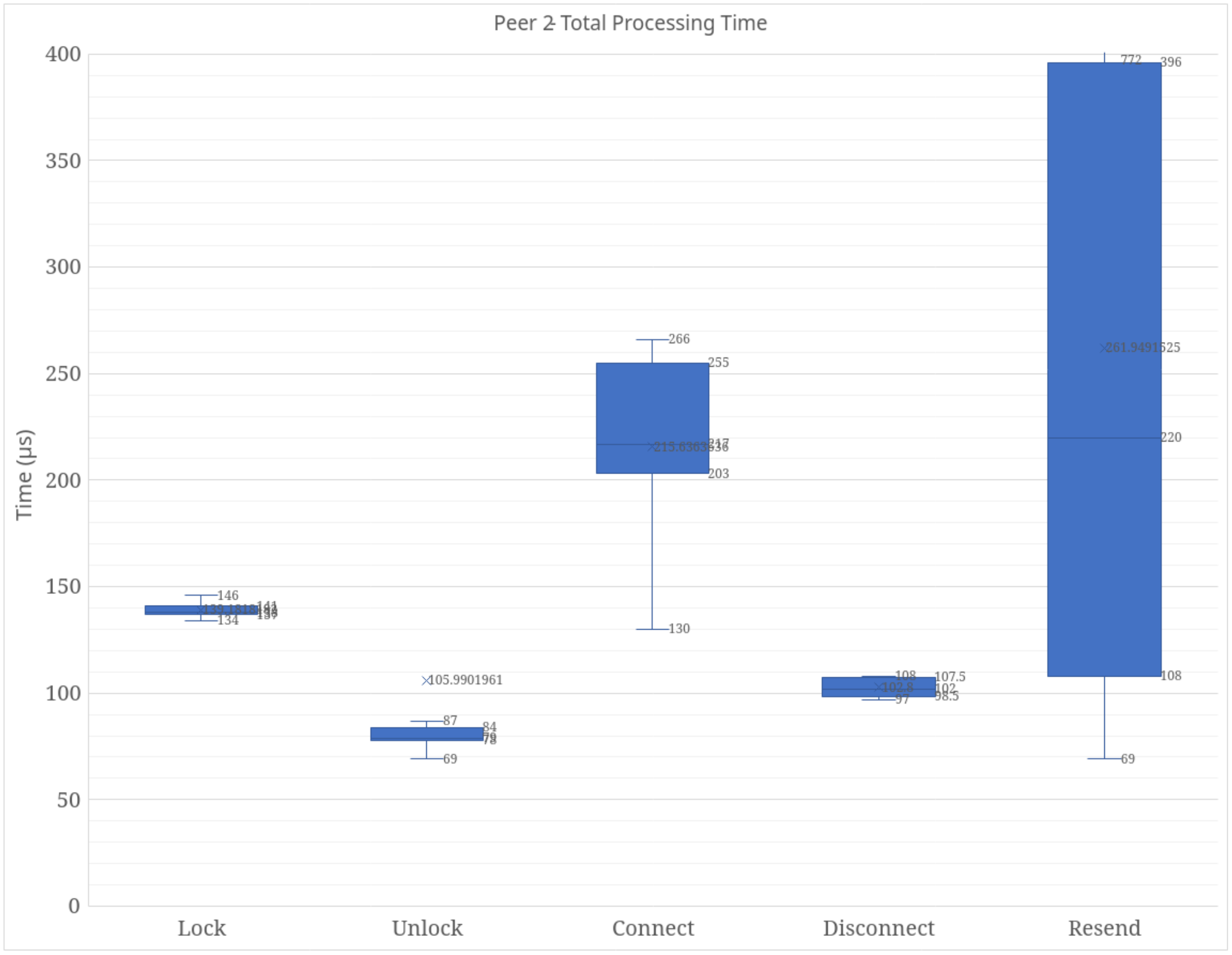


Figure 11: *Box and Whisker plots summarizing the total processing time of peer 2. Peer 2 being a shared server is expected to take longer to process results, our collected data reflect that. However the property of messages taking less than half a millisecond to process still holds.*

As peer 1 needs to establish the network, it takes longer to process messages, as we can see in the figures above. On the other hand, peer two can process messages rather quickly without the overhead of getting the things set up. We can see this in comparison between the two figures.

Additionally, we measured how long it took for the system to process messages carrying file content. We used a similar methodology to the one described above, except we sent files of three different sizes: 15 bytes, 1,000 bytes, and 10,000 bytes. As expected, increasing the size of the file increases the processing time. The results for both of our peers are summarized in figures 12 and 13.

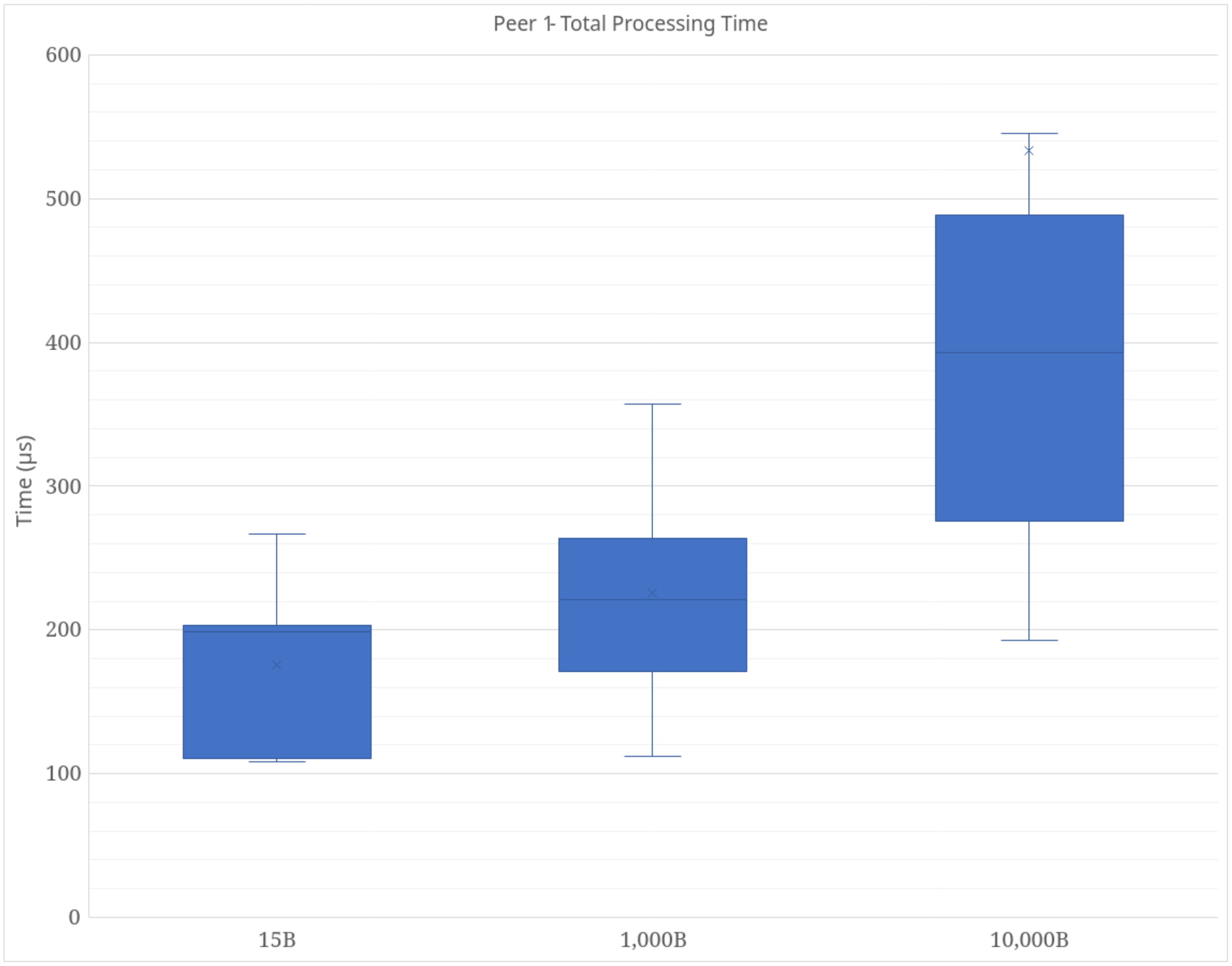


Figure 12: *Box and Whisker plots summarizing the total processing time of three different sized files on peer 1. Time taken generally increases with file size.*

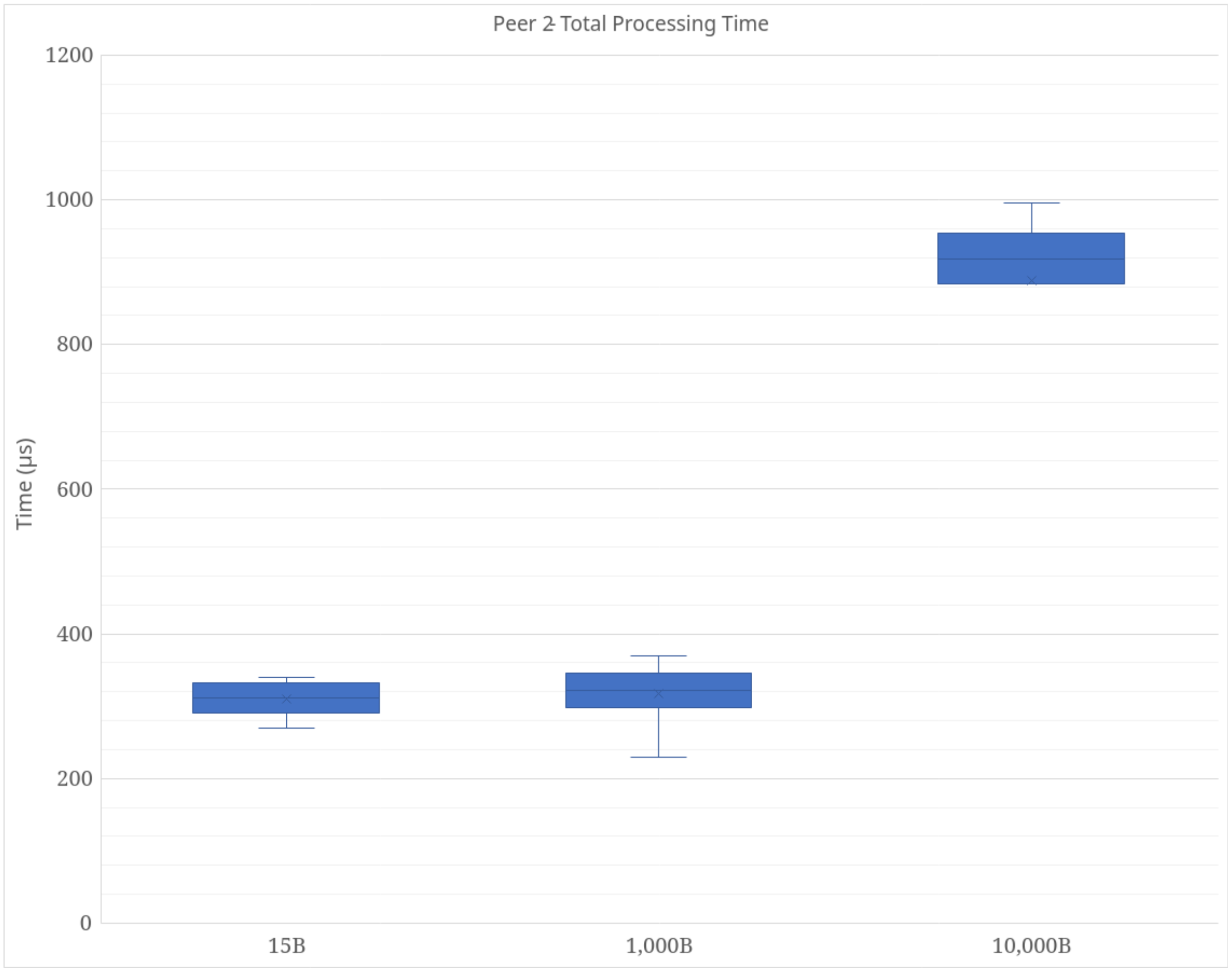


Figure 13: *Box and Whisker plots summarizing the total processing time of three different sized files on peer 2. Time taken generally increases with file size, and it takes substantially longer than on peer 1.*

We additionally measured the system resources utilized by the program. However, the results proved to be quite uninteresting, but are summarized in table 2 nonetheless. Physical memory utilized remained constant, the amount of virtual memory in use varied between executions of the program but remained constant between program executions, and CPU utilization remained very close to zero percent, with occasional spikes up to sixteen percent. All of these results remained consistent on all machines we measured.

Table 2: *Hardware utilization of the program.*

CPU:	0 - 16%
Physical Memory:	522724 bytes (0.499 MB)
Virtual Memory:	9000 - 9200 bytes (8.789 - 8.984 kB)

These results indicate that our program is well suited to its role as a background service. It uses very little CPU time and only half a megabyte worth of RAM while still being able to quickly process even large files in an imperceivable amount of time.

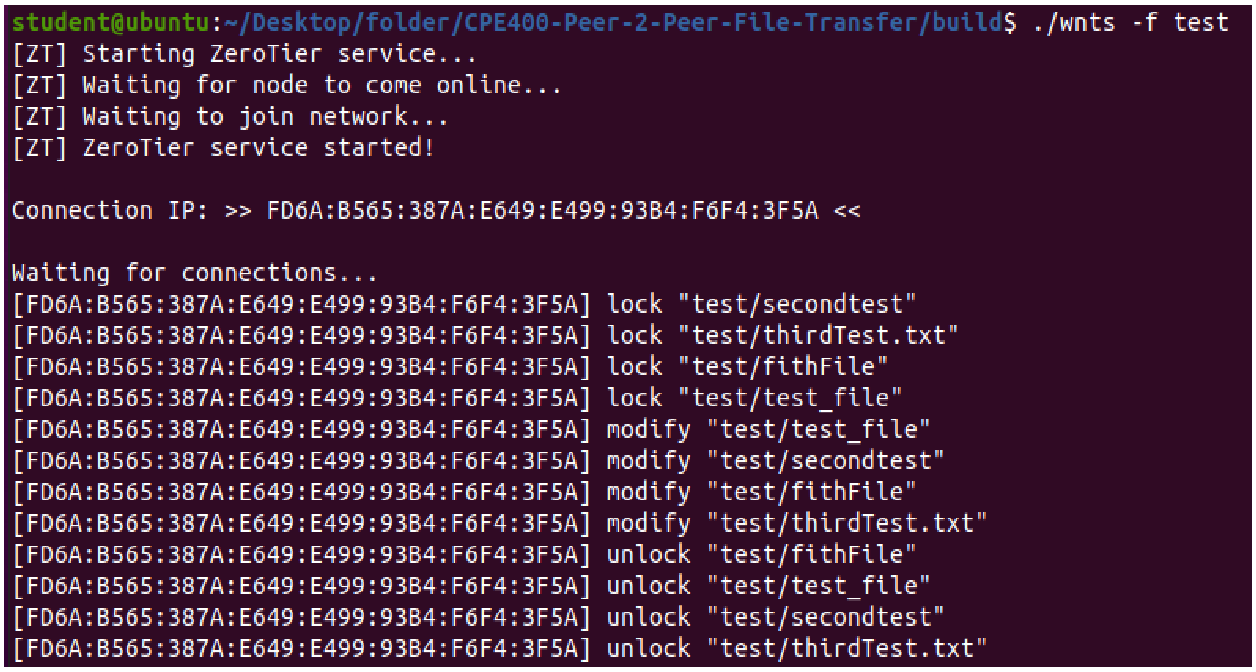
Setup

The first step is to pull the code from our github repository to set up the program. Next, a set of commands are needed to set up and create the program, as seen in the figure below. Open a terminal, and enter the commands:

```
git clone
https://github.com/joshuadahlunr/CPE400-Peer-2-Peer-File-Transfer
git submodule init
git submodule update --init --recursive #This installs the necessary
libraries our code depends on and updates them accordingly.
```

The next step is to create a build directory. Once the directory is created, change to the build directory and run the command "cmake .." followed by the "make" command. This builds the final executable and prepares the program to be run.

To run the program, we need first to create our initial node. This is done by running the command "./wnts", followed by the argument "-f" and a file path to the folder we want the network to synchronize. This will establish the initial gateway node that subsequent nodes will connect to form the network. Once established, the node will begin waiting for new nodes to connect and sweep all files in its connection path, as shown in figure 14 below.



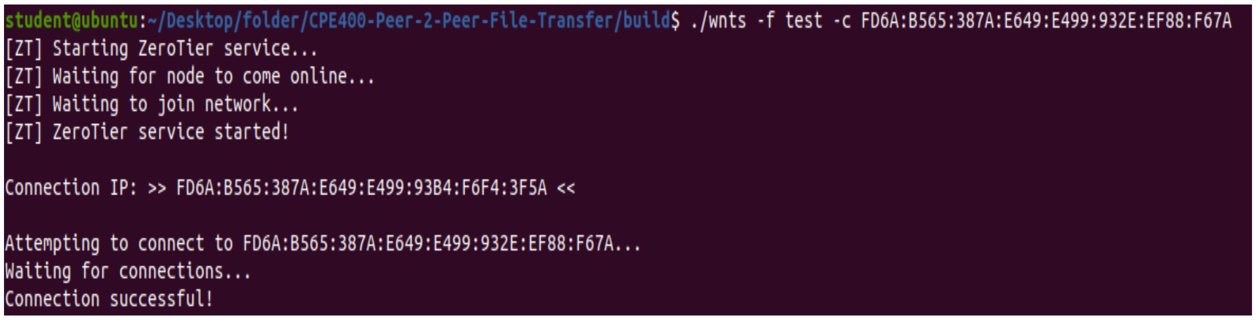
```
student@ubuntu:~/Desktop/folder/CPE400-Peer-2-Peer-File-Transfer/build$ ./wnts -f test
[ZT] Starting ZeroTier service...
[ZT] Waiting for node to come online...
[ZT] Waiting to join network...
[ZT] ZeroTier service started!

Connection IP: >> FD6A:B565:387A:E649:E499:93B4:F6F4:3F5A <<

Waiting for connections...
[FD6A:B565:387A:E649:E499:93B4:F6F4:3F5A] lock "test/secondtest"
[FD6A:B565:387A:E649:E499:93B4:F6F4:3F5A] lock "test/thirdTest.txt"
[FD6A:B565:387A:E649:E499:93B4:F6F4:3F5A] lock "test/fithFile"
[FD6A:B565:387A:E649:E499:93B4:F6F4:3F5A] lock "test/test_file"
[FD6A:B565:387A:E649:E499:93B4:F6F4:3F5A] modify "test/test_file"
[FD6A:B565:387A:E649:E499:93B4:F6F4:3F5A] modify "test/secondtest"
[FD6A:B565:387A:E649:E499:93B4:F6F4:3F5A] modify "test/fithFile"
[FD6A:B565:387A:E649:E499:93B4:F6F4:3F5A] modify "test/thirdTest.txt"
[FD6A:B565:387A:E649:E499:93B4:F6F4:3F5A] unlock "test/fithFile"
[FD6A:B565:387A:E649:E499:93B4:F6F4:3F5A] unlock "test/test_file"
[FD6A:B565:387A:E649:E499:93B4:F6F4:3F5A] unlock "test/secondtest"
[FD6A:B565:387A:E649:E499:93B4:F6F4:3F5A] unlock "test/thirdTest.txt"
```

Figure 14: *Establishing the first node. When the first node first sweeps the network it locks them all, notes that they have been modified, and releases them. This process is not repeated with future connections since the network is already aware of those files.*

Now that the first node is established, other nodes can begin to connect to the network. Each subsequent node will now need to add the “-c” argument and the IP address of the node they want to connect to.



```
student@ubuntu:~/Desktop/folder/CPE400-Peer-2-Peer-File-Transfer/build$ ./wnts -f test -c FD6A:B565:387A:E649:E499:932E:EF88:F67A
[ZT] Starting ZeroTier service...
[ZT] Waiting for node to come online...
[ZT] Waiting to join network...
[ZT] ZeroTier service started!

Connection IP: >> FD6A:B565:387A:E649:E499:93B4:F6F4:3F5A <<

Attempting to connect to FD6A:B565:387A:E649:E499:932E:EF88:F67A...
Waiting for connections...
Connection successful!
```

Figure 15: *This figure shows the messages for a newly connected node to the gateway node. Here the Accepted connection message along with the sync message is displayed.*

Once the peer has connected, it can be stated that the network is fully established. Any new node can now connect to any connected node in the network and begin synchronizing files.

Conclusion

Designing and implementing a file transfer protocol using Whisper Networking had many advantages and disadvantages. One of the positive features is the peer node design; pulling from the front of the backup list and not the back when a peer leaves the network allows for efficient recovery and replacement nodes to be selected to result in a higher quality topology. A disadvantage would be that the reliability of a Whisper Network when the demographic is kept small is exceptional, but this reliability does not scale, and it takes longer and longer to propagate messages as the network grows, hubs get overloaded, and the number of peer hops needed increase.

While collecting data in our testing phase, we came across a few differences in measurements. The differences allowed us to analyze why they were present and how to apply the knowledge from the collected data. Because we were able to see the differences in the transmission of files and other transactions, we noticed the workload that the gateway peer had compared to a peer entering the network. The ability to transfer files through a Whisper Network was a challenging and educational experience for the team. The overall experience with Whisper Networks and file transferring will be invaluable in the future, especially considering the meteoric rise in popularity of peer-to-peer-based blockchain technology.

References

Stephen Foskett Packrat (website). Retrieved from https://blog.fosketts.net/2022/01/14/how-to-connect-everything-from-everywhere-with-zerotier/

Appendix A

Timing Measurements

May 2, 2022

CPE 400.1001 Team
Joshua Dahl, Antonio Massa, Annette McDonough

Instructor
Dr. Engin Arslan

Peer 1						Peer 2				
ID	Message	Total (μs)	Deserialization (μs)	Routing (μs)		ID	Message	Total (μs)	Deserialization (μs)	Routing (μs)
1	Lock	55	31	24		1	Lock	140	71	69
1	Lock	49	34	15		1	Lock	141	72	69
1	Lock	61	35	26		1	Lock	144	73	71
1	Lock	37	22	15		1	Lock	139	71	68
1	Lock	39	18	21		1	Lock	138	70	68
1	Lock	27	16	11		1	Lock	136	69	67
1	Lock	47	27	20		1	Lock	134	68	66
1	Lock	38	22	16		1	Lock	146	76	70
1	Lock	42	26	16		1	Lock	138	71	67
1	Lock	54	29	25		1	Lock	138	71	67
1	Lock	64	34	30		1	Lock	137	71	66
1	Lock	51	32	19		2	Unlock	367	84	283
1	Lock	37	23	14		2	Unlock	358	86	272
1	Lock	59	34	25		2	Unlock	70	34	36
1	Lock	47	27	20		2	Unlock	331	85	246
1	Lock	60	34	26		2	Unlock	301	81	220
1	Lock	39	21	18		2	Unlock	71	34	37
1	Lock	63	37	26		2	Unlock	327	99	228
1	Lock	74	41	33		2	Unlock	288	80	208
1	Lock	56	34	22		2	Unlock	71	33	38
1	Lock	54	28	26		2	Unlock	251	64	187
1	Lock	64	27	37		2	Unlock	306	84	222
2	Unlock	46	23	23		2	Unlock	71	33	38
2	Unlock	84	38	46		2	Unlock	317	81	236
2	Unlock	51	18	33		2	Unlock	69	33	36
2	Unlock	55	32	23		2	Unlock	332	122	210
2	Unlock	159	74	85		2	Unlock	84	34	50
2	Unlock	46	25	21		2	Unlock	78	32	46
2	Unlock	141	61	80		2	Unlock	77	32	45
2	Unlock	30	9	21		2	Unlock	79	33	46
2	Unlock	44	25	19		2	Unlock	78	32	46

2	Unlock	112	56	56		2	Unlock	77	32	45
2	Unlock	36	23	13		2	Unlock	78	32	46
2	Unlock	182	84	98		2	Unlock	77	32	45
2	Unlock	34	12	22		2	Unlock	98	52	46
2	Unlock	51	26	25		2	Unlock	79	33	46
2	Unlock	143	56	87		2	Unlock	78	32	46
2	Unlock	48	29	19		2	Unlock	82	33	49
2	Unlock	115	56	59		2	Unlock	83	33	50
2	Unlock	19	8	11		2	Unlock	83	33	50
2	Unlock	37	21	16		2	Unlock	78	32	46
2	Unlock	203	81	122		2	Unlock	82	34	48
2	Unlock	57	29	28		2	Unlock	82	33	49
2	Unlock	95	43	52		2	Unlock	82	33	49
2	Unlock	66	37	29		2	Unlock	85	34	51
2	Unlock	128	61	67		2	Unlock	82	34	48
2	Unlock	27	9	18		2	Unlock	84	33	51
2	Unlock	44	25	19		2	Unlock	84	34	50
2	Unlock	126	17	109		2	Unlock	111	59	52
2	Unlock	124	29	95		2	Unlock	84	35	49
2	Unlock	144	28	116		2	Unlock	78	33	45
2	Unlock	160	16	144		2	Unlock	78	33	45
2	Unlock	114	14	100		2	Unlock	78	33	45
2	Unlock	98	22	76		2	Unlock	78	32	46
2	Unlock	74	14	60		2	Unlock	139	34	105
2	Unlock	94	17	77		2	Unlock	81	33	48
2	Unlock	103	21	82		2	Unlock	77	32	45
2	Unlock	139	27	112		2	Unlock	77	33	44
2	Unlock	157	26	131		2	Unlock	83	33	50
2	Unlock	87	15	72		2	Unlock	115	66	49
2	Unlock	1781	13	1768		2	Unlock	77	33	44
2	Unlock	108	21	87		2	Unlock	77	32	45
2	Unlock	135	26	109		2	Unlock	78	32	46
2	Unlock	143	29	114		2	Unlock	78	32	46
2	Unlock	115	17	98		2	Unlock	78	33	45
2	Unlock	85	11	74		2	Unlock	78	33	45

2	Unlock	87	13	74		2	Unlock	78	33	45
2	Unlock	167	29	138		2	Unlock	77	32	45
2	Unlock	149	27	122		2	Unlock	77	33	44
2	Unlock	116	23	93		2	Unlock	78	33	45
2	Unlock	111	20	91		2	Unlock	134	86	48
2	Unlock	106	21	85		2	Unlock	82	33	49
2	Unlock	72	15	57		2	Unlock	82	34	48
2	Unlock	115	23	92		2	Unlock	81	34	47
2	Unlock	160	64	96		2	Unlock	78	33	45
2	Unlock	106	21	85		2	Unlock	81	33	48
2	Unlock	119	14	105		2	Unlock	87	34	53
2	Unlock	110	23	87		2	Unlock	78	32	46
2	Unlock	122	24	98		2	Unlock	79	33	46
2	Unlock	1610	16	1594		2	Unlock	77	32	45
2	Unlock	120	16	104		2	Unlock	81	33	48
2	Unlock	140	27	113		2	Unlock	79	33	46
2	Unlock	78	15	63		2	Unlock	78	32	46
2	Unlock	63	12	51		2	Unlock	78	32	46
2	Unlock	216	35	181		2	Unlock	78	33	45
2	Unlock	78	17	61		2	Unlock	79	33	46
2	Unlock	125	25	100		2	Unlock	151	37	114
2	Unlock	94	15	79		2	Unlock	78	33	45
2	Unlock	133	25	108		2	Unlock	77	32	45
2	Unlock	1615	25	1590		2	Unlock	78	33	45
2	Unlock	122	25	97		2	Unlock	79	33	46
2	Unlock	110	23	87		2	Unlock	78	33	45
2	Unlock	78	15	63		2	Unlock	77	32	45
2	Unlock	138	28	110		2	Unlock	78	33	45
2	Unlock	130	27	103		2	Unlock	78	32	46
2	Unlock	105	21	84		2	Unlock	78	32	46
2	Unlock	176	29	147		2	Unlock	79	33	46
2	Unlock	130	26	104		2	Unlock	77	32	45
2	Unlock	136	22	114		2	Unlock	117	44	73
2	Unlock	265	22	243		2	Unlock	79	33	46
2	Unlock	133	27	106		2	Unlock	87	35	52

2	Unlock	125	25	100		2	Unlock	83	34	49
2	Unlock	103	22	81		2	Unlock	113	46	67
2	Unlock	147	35	112		2	Unlock	81	34	47
2	Unlock	86	17	69		2	Unlock	82	34	48
2	Unlock	139	29	110		2	Unlock	83	34	49
2	Unlock	132	24	108		2	Unlock	77	40	37
2	Unlock	101	22	79		2	Unlock	84	34	50
2	Unlock	73	13	60		2	Unlock	83	34	49
2	Unlock	1746	14	1732		2	Unlock	83	34	49
2	Unlock	104	23	81		2	Unlock	77	40	37
2	Unlock	85	11	74		2	Unlock	84	34	50
2	Unlock	97	15	82		2	Unlock	83	34	49
2	Unlock	56	12	44		7	Connect	208	91	117
2	Unlock	91	21	70		7	Connect	266	141	125
2	Unlock	115	41	74		7	Connect	217	90	127
2	Unlock	78	14	64		7	Connect	203	87	116
2	Unlock	98	14	84		7	Connect	208	87	121
2	Unlock	149	14	135		7	Connect	171	83	88
2	Unlock	1651	27	1624		7	Connect	262	108	154
2	Unlock	75	13	62		7	Connect	255	92	163
2	Unlock	136	27	109		7	Connect	222	90	132
2	Unlock	90	17	73		7	Connect	230	113	117
2	Unlock	85	14	71		7	Connect	130	61	69
2	Unlock	86	14	72		8	Disconnect	108	37	71
2	Unlock	77	15	62		8	Disconnect	102	33	69
2	Unlock	92	14	78		8	Disconnect	100	34	66
2	Unlock	128	27	101		8	Disconnect	107	35	72
2	Unlock	124	25	99		8	Disconnect	97	33	64
2	Unlock	129	25	104		10	Resend	220	177	43
2	Unlock	97	22	75		10	Resend	218	176	42
2	Unlock	53	19	34		10	Resend	219	175	44
2	Unlock	77	15	62		10	Resend	220	177	43
2	Unlock	92	14	78		10	Resend	218	173	45
2	Unlock	128	27	101		10	Resend	406	203	203
2	Unlock	124	25	99		10	Resend	107	45	62

2	Unlock	129	25	104		10	Resend	373	193	180
2	Unlock	97	22	75		10	Resend	106	44	62
2	Unlock	53	19	34		10	Resend	379	194	185
8	Disconne ct	108	22	86		10	Resend	107	45	62
8	Disconne ct	115	24	91		10	Resend	401	197	204
8	Disconne ct	167	28	139		10	Resend	105	45	60
8	Disconne ct	131	25	106		10	Resend	373	193	180
8	Disconne ct	96	15	81		10	Resend	106	45	61
8	Disconne ct	333	27	306		10	Resend	373	193	180
8	Disconne ct	333	27	306		10	Resend	103	44	59
10	Resend	149	113	36		10	Resend	371	192	179
10	Resend	92	75	17		10	Resend	114	47	67
10	Resend	117	85	32		10	Resend	375	194	181
10	Resend	61	47	14		10	Resend	113	45	68
10	Resend	86	67	19		10	Resend	396	214	182
10	Resend	276	205	71		10	Resend	102	44	58
10	Resend	27	10	17		10	Resend	416	196	220
10	Resend	330	261	69		10	Resend	102	44	58
10	Resend	50	19	31		10	Resend	369	192	177
10	Resend	323	252	71		10	Resend	103	44	59
10	Resend	51	18	33		10	Resend	407	221	186
10	Resend	398	287	111		10	Resend	103	44	59
10	Resend	27	8	19		10	Resend	408	221	187
10	Resend	353	260	93		10	Resend	106	45	61
10	Resend	29	9	20		10	Resend	410	222	188
10	Resend	239	180	59		10	Resend	127	66	61
10	Resend	43	15	28		10	Resend	772	203	569
10	Resend	220	156	64		10	Resend	148	85	63
10	Resend	27	10	17		10	Resend	451	241	210
10	Resend	269	173	96		10	Resend	147	84	63

10	Resend	27	10	17		10	Resend	409	222	187
10	Resend	265	194	71		10	Resend	103	44	59
10	Resend	31	12	19		10	Resend	407	220	187
10	Resend	320	234	86		10	Resend	103	44	59
10	Resend	48	18	30		10	Resend	433	245	188
10	Resend	328	252	76		10	Resend	104	44	60
10	Resend	27	10	17		10	Resend	406	221	185
10	Resend	268	180	88		10	Resend	69	26	43
10	Resend	32	10	22		10	Resend	419	224	195
10	Resend	2016	1899	117		10	Resend	106	44	62
10	Resend	31	12	19		10	Resend	413	223	190
10	Resend	247	187	60		10	Resend	159	52	107
10	Resend	39	15	24		10	Resend	446	253	193
10	Resend	327	255	72		10	Resend	123	51	72
10	Resend	53	20	33		10	Resend	430	252	178
10	Resend	367	247	120		10	Resend	126	50	76
10	Resend	50	18	32		10	Resend	396	196	200
10	Resend	266	196	70		10	Resend	167	92	75
10	Resend	31	10	21		10	Resend	376	197	179
10	Resend	180	138	42		10	Resend	121	49	72
10	Resend	31	10	21		10	Resend	375	194	181
10	Resend	197	151	46		10	Resend	107	45	62
10	Resend	24	10	14		10	Resend	372	194	178
10	Resend	349	266	83		10	Resend	106	45	61
10	Resend	46	20	26		10	Resend	473	286	187
10	Resend	340	253	87		10	Resend	163	94	69
10	Resend	76	18	58		10	Resend	430	216	214
10	Resend	252	195	57		10	Resend	112	46	66
10	Resend	38	15	23		10	Resend	367	192	175
10	Resend	247	186	61		10	Resend	103	44	59
10	Resend	40	16	24		10	Resend	370	192	178
10	Resend	245	189	56		10	Resend	112	46	66
10	Resend	39	16	23		10	Resend	411	222	189
10	Resend	174	129	45		10	Resend	111	46	65
10	Resend	27	10	17		10	Resend	405	230	175

10	Resend	289	211	78		10	Resend	130	45	85
10	Resend	43	15	28		10	Resend	367	193	174
10	Resend	324	257	67		10	Resend	114	47	67
10	Resend	67	27	40		10	Resend	374	195	179
10	Resend	233	180	53		10	Resend	130	68	62
10	Resend	44	15	29		10	Resend	371	193	178
10	Resend	271	195	76		10	Resend	104	44	60
10	Resend	43	13	30		10	Resend	366	193	173
10	Resend	285	223	62		10	Resend	123	51	72
10	Resend	49	15	34		10	Resend	373	195	178
10	Resend	287	217	70		10	Resend	121	49	72
10	Resend	51	19	32		10	Resend	413	196	217
10	Resend	1758	1691	67		10	Resend	108	44	64
10	Resend	25	9	16		10	Resend	369	193	176
10	Resend	358	241	117		10	Resend	126	51	75
10	Resend	21	7	14		10	Resend	363	192	171
10	Resend	311	242	69		10	Resend	143	57	86
10	Resend	52	19	33		10	Resend	379	196	183
10	Resend	173	131	42		10	Resend	104	44	60
10	Resend	29	10	19		10	Resend	368	193	175
10	Resend	158	116	42		10	Resend	140	54	86
10	Resend	26	10	16		10	Resend	466	256	210
10	Resend	468	353	115		10	Resend	148	58	90
10	Resend	32	10	22		10	Resend	439	207	232
10	Resend	171	128	43		10	Resend	149	59	90
10	Resend	25	9	16		10	Resend	429	206	223
10	Resend	284	214	70		10	Resend	149	58	91
10	Resend	53	20	33		10	Resend	430	202	228
10	Resend	222	167	55		10	Resend	129	52	77
10	Resend	26	10	16		10	Resend	380	195	185
10	Resend	320	240	80		10	Resend	111	45	66
10	Resend	51	19	32		10	Resend	407	230	177
10	Resend	1789	1713	76		10	Resend	103	44	59
10	Resend	46	17	29		10	Resend	489	221	268
10	Resend	286	216	70		10	Resend	108	46	62

10	Resend	101	21	80		10	Resend	367	192	175
10	Resend	282	184	98		10	Resend	104	44	60
10	Resend	41	17	24		10	Resend	394	194	200
10	Resend	178	139	39		10	Resend	105	44	61
10	Resend	29	10	19		10	Resend	387	193	194
10	Resend	324	249	75		10	Resend	106	45	61
10	Resend	51	19	32		10	Resend	411	236	175
10	Resend	434	228	206		10	Resend	104	44	60
10	Resend	147	72	75		10	Resend	374	194	180
10	Resend	244	184	60		10	Resend	104	44	60
10	Resend	40	18	22		10	Resend	366	193	173
10	Resend	463	309	154		10	Resend	104	44	60
10	Resend	56	19	37		10	Resend	363	193	170
10	Resend	309	239	70		10	Resend	103	45	58
10	Resend	48	16	32		10	Resend	369	193	176
10	Resend	269	216	53		10	Resend	106	44	62
10	Resend	44	17	27		10	Resend	387	211	176
10	Resend	429	350	79		10	Resend	107	45	62
10	Resend	38	15	23		10	Resend	443	266	177
10	Resend	301	234	67		10	Resend	106	44	62
10	Resend	94	54	40		10	Resend	369	193	176
10	Resend	298	225	73		10	Resend	105	44	61
10	Resend	47	18	29		10	Resend	366	192	174
10	Resend	235	182	53		10	Resend	103	44	59
10	Resend	46	16	30		10	Resend	366	193	173
10	Resend	579	247	332		10	Resend	133	72	61
10	Resend	91	22	69		10	Resend	369	193	176
10	Resend	208	151	57		10	Resend	158	88	70
10	Resend	26	10	16		10	Resend	373	195	178
10	Resend	344	268	76		10	Resend	104	44	60
10	Resend	53	21	32		10	Resend	369	193	176
10	Resend	339	227	112		10	Resend	141	44	97
10	Resend	50	18	32		10	Resend	414	196	218
10	Resend	235	177	58		10	Resend	103	44	59
10	Resend	38	13	25		10	Resend	366	192	174

10	Resend	185	137	48		10	Resend	104	44	60
10	Resend	26	9	17		10	Resend	369	192	177
10	Resend	1895	1831	64		10	Resend	117	47	70
10	Resend	30	10	20		10	Resend	374	196	178
10	Resend	238	183	55		10	Resend	104	44	60
10	Resend	37	15	22		10	Resend	371	193	178
10	Resend	175	136	39		10	Resend	120	46	74
10	Resend	21	7	14		10	Resend	413	235	178
10	Resend	225	173	52		10	Resend	104	45	59
10	Resend	42	13	29		10	Resend	421	196	225
10	Resend	161	115	46		10	Resend	133	60	73
10	Resend	50	18	32		10	Resend	505	255	250
10	Resend	228	172	56		10	Resend	150	57	93
10	Resend	40	15	25		10	Resend	471	222	249
10	Resend	248	194	54		10	Resend	141	57	84
10	Resend	42	14	28		10	Resend	528	298	230
10	Resend	184	141	43		10	Resend	108	46	62
10	Resend	23	8	15		10	Resend	414	217	197
10	Resend	213	163	50		10	Resend	121	46	75
10	Resend	35	9	26		10	Resend	466	199	267
10	Resend	317	254	63		10	Resend	112	46	66
10	Resend	28	10	18		10	Resend	444	222	222
10	Resend	1751	1703	48		10	Resend	110	46	64
10	Resend	52	28	24		10	Resend	351	164	187
10	Resend	193	142	51		10	Resend	106	44	62
10	Resend	24	8	16		10	Resend	482	226	256
10	Resend	320	241	79		10	Resend	107	45	62
10	Resend	52	19	33		10	Resend	420	239	181
10	Resend	194	146	48		10	Resend	444	222	222
10	Resend	24	8	16		10	Resend	110	46	64
10	Resend	205	151	54		10	Resend	351	164	187
10	Resend	30	10	20		10	Resend	106	44	62
10	Resend	217	159	58		10	Resend	482	226	256
10	Resend	33	11	22		10	Resend	107	45	62
10	Resend	176	130	46		10	Resend	420	239	181

10	Resend	25	9	16						
10	Resend	256	198	58						
10	Resend	25	10	15						
10	Resend	302	230	72						
10	Resend	106	18	88						
10	Resend	301	226	75						
10	Resend	51	19	32						
10	Resend	302	226	76						
10	Resend	50	20	30						
10	Resend	252	173	79						
10	Resend	44	15	29						
10	Resend	146	116	30						
10	Resend	176	130	46						
10	Resend	25	9	16						
10	Resend	256	198	58						
10	Resend	25	10	15						
10	Resend	302	230	72						
10	Resend	106	18	88						
10	Resend	301	226	75						
10	Resend	51	19	32						
10	Resend	302	226	76						
10	Resend	50	20	30						
10	Resend	252	173	79						
10	Resend	44	15	29						
10	Resend	146	116	30						
					15 Bytes					
4	Content	206	84	122		4	Content	278	71	207
4	Content	181	93	88		4	Content	333	89	244
4	Content	201	95	106		4	Content	324	96	228
4	Content	111	55	56		4	Content	270	82	188
4	Content	108	52	56		4	Content	311	84	227
4	Content	200	104	96		4	Content	294	84	210
4	Content	199	81	118		4	Content	302	95	207
4	Content	267	120	147		4	Content	333	115	218
4	Content	110	52	58		4	Content	340	84	256

						4	Content	311	87	224
1000 Bytes										
4	Content	185	107	78		4	Content	229	149	80
4	Content	357	58	299		4	Content	332	93	239
4	Content	307	42	265		4	Content	343	112	231
4	Content	264	78	186		4	Content	324	90	234
4	Content	112	61	51		4	Content	292	88	204
4	Content	221	136	85		4	Content	320	90	230
4	Content	171	113	58		4	Content	370	93	277
4	Content	201	117	84		4	Content	356	94	262
4	Content	262	130	132		4	Content	301	90	211
4	Content	229	123	106		4	Content	301	84	217
4	Content	170	80	90						
10000 Bytes										
4	Content	545	243	302		4	Content	519	149	370
4	Content	325	223	102		4	Content	884	93	791
4	Content	287	220	67		4	Content	938	112	826
4	Content	193	134	59		4	Content	920	90	830
4	Content	398	302	96		4	Content	884	88	796
4	Content	265	164	101		4	Content	917	90	827
4	Content	393	289	104		4	Content	943	93	850
4	Content	432	325	107		4	Content	985	94	891
4	Content	1963	181	1782		4	Content	995	90	905
						4	Content	903	84	819