File Distribution Protocol

120010570

CS3102: Data Communication & Networks

1 Analysis: Multicast vs Peer to Peer

Multicast is ideal for sending the same message to multiple nodes: the transmission time will remain constant for any number of nodes. However until IPv6 is support more widely, with its mandatory multicast, Multicast traffic is confined to local subnets. Another issue of Multicast for File Distribution is the slow consumer problem: a Multicast host transmitting at a constant speed is making the assumption that all of its listeners can download at that same speed. If a client cannot consume the data at that rate, it will drop a substantial amount of packets. This could be avoided by transmitting at a variety of speeds, however this is potentially a complex solution which could impact on performance.

A peer-to-peer system in theory should scale well as the number of nodes increases, though not as well as Multicast. P2P systems can be extremely complex to develop, debug and manage, and this is a considerable disadvantage of P2P. However using a small piece of centralised infrastructure they can be simplified and made easier to manage. A central BitTorrent-style tracker could be used to avoid requiring technology akin to a Distributed Hash Table. P2P traffic is accepted on the IPv4 internet. P2P should also handle peers dropping in and out. P2P over TCP will not suffer from the slow consumer problem as TCP's congestion control features will kick in. P2P will also be able to scale better for re-transmitting chunks (e.g. if packets are lost or a host goes down for a short period of time). A multicast system must either loop through data continuously (teletext-style), or receiving peers would ask the host to re-send chunks. Resending chunks will waste the time of every client which already has that chunk.

1.1 Performance Predictions

To analyse the two methods further I decided to calculate the expected performance of Multicast and Peer-to-Peer. The times used in these calculations are based on a 1GB file being transferred over a 100Mbit/s connection. In these calculations it is assumed that each node can upload and download 100Mbit/s at the same time, which is perhaps over-optimistic.

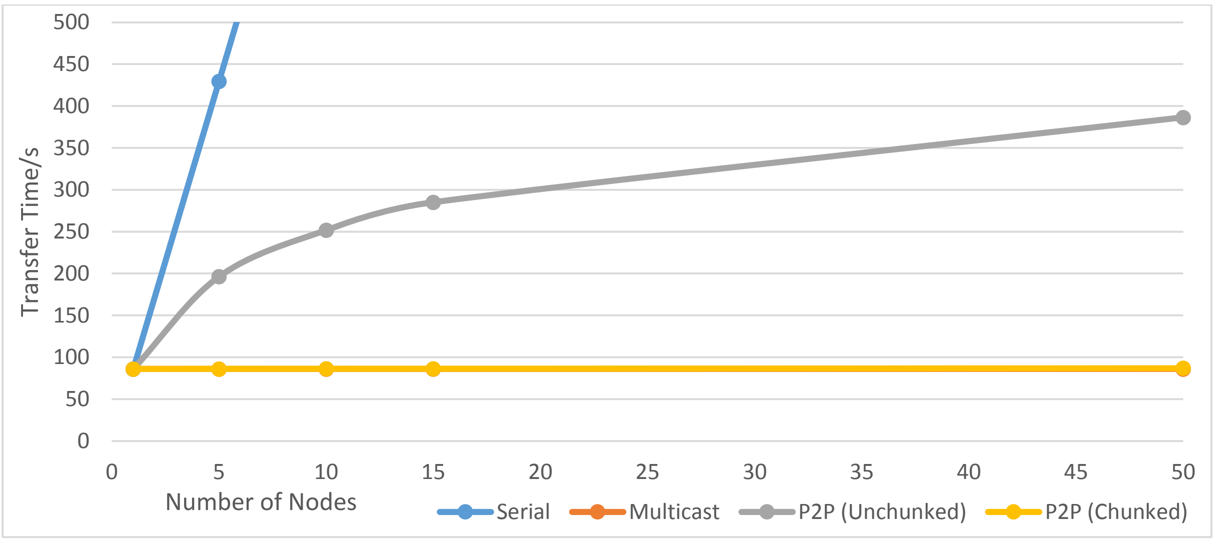


FIGURE 1.0 – GRAPH PREDICTING FILE TRANSFER TIME FOR DIFFERENT ARCHITECTURES ON A LOSSLESS NETWORK. NOTE: LINES FOR MULTICAST AND P2P (CHUNKED) LARGELY OVERLAP.

Figure 1.0 shows that an un-chunked P2P system will be considerably out-performed by Multicast, but will still be significantly more efficient than Serial transfer. In these calculations, Chunked P2P comes across as performing nearly as well as Multicast. I think these calculations assume the switch backplane capacity is enough to cope with every node being maxed out.

1.2 Empirical Research

Before I select which architecture to use in my file distribution system I decided some tests were needed to verify my predictions and assumptions, particularly after the surprising result of Chunked P2P in Figure 1.0.

1.2.1 Test 1: Client-Server Serial transfer

Basic file distribution system used as a benchmark. A single host machine, running iperf will distribute the required amount data. UDP mode used to avoid TCP ramp up artefacts and any packet corruptions.

1.2.2 Test 2: Multicast transfer

Using iperf again, this test will measure the time it takes for all clients listening to the group to receive all of the data.

1.2.3 Test 3: BitTorrent Peer to Peer

Using the most complex software of the 4 tests, this experiment should test whether the performance of a pre-existing P2P file distribution solution matches up with my expectations (Somewhere between Unchunked and chunked P2P in Figure 1.0). I will be using Herd (Garrett & Gadea, 2014) as a P2P system for this test. Herd uses BitTorrent underneath to perform the file transfers.

1.3 Results

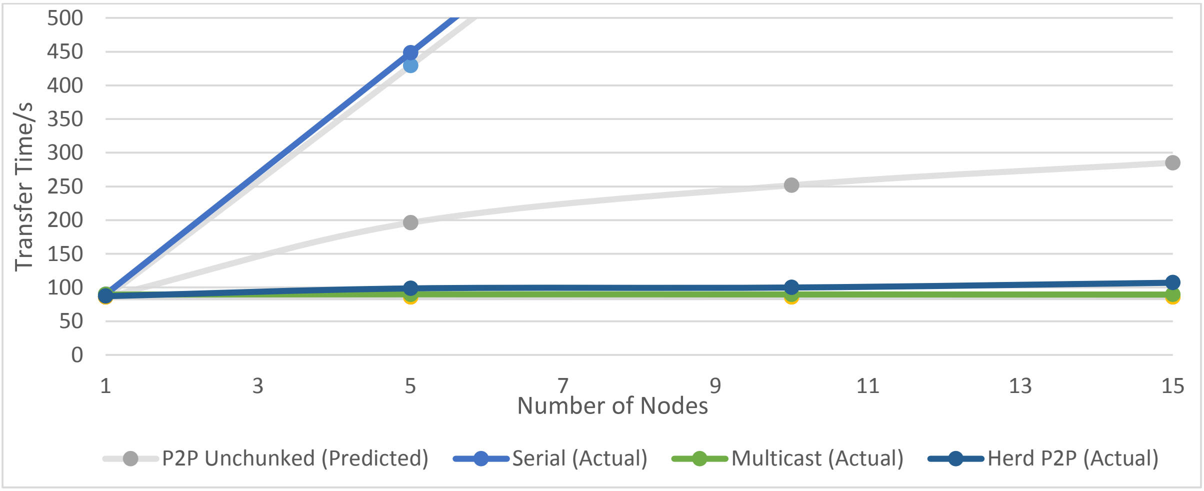


FIGURE 1.1 – EMPIRICAL DATA OF SERIAL, MULTICAST AND A P2P IMPLEMENTATION TRANSFERRING A 1GB FILE ACROSS A SUBNET TO A VARIABLE NUMBER OF NODES. IN GREY ARE THE VALUES FROM FIGURE 1.0

The results stick very closely to the predicted values. The multicast test performs better than P2P, with the transfer taking exactly the same amount of time for any of the numbers of nodes recorded. The P2P software tested performs better than I expected. It does not outperform the predicted "P2P Chunked" however as noted above that value was optimistic.

1.4 Analysis Conclusion

Although it was outperformed in the Empirical Data, I think P2P is the most appropriate architecture to use for this tool. By choosing P2P the application can function on networks outside Multicast-enabled subnets. The speed advantage is not great enough to warrant the restriction of this software for inside a subnet. The P2P architecture may require more careful designing, but the reliability checks will scale far better than for any pure-Multicast solution.

An ideal solution might be a combination of Multicast and P2P to provide the speed boost Multicast provides when used on a low packet-loss LAN subnet, using P2P for re-sending missing chunks.

2 Design: Peer to Peer System

The distribution system must be reliable and efficient. My analysis above shows the way to bring efficiency is to go Peer to peer, distributing larger files in smaller chunks. This design section covers the design of the Protocols, file structures, and system architecture. Security, Reliability, Maintainability and Efficiency were the main goals in mind (roughly in that order) when designing this system.

2.1 Initial Metadata

To be able to join a swarm, and obtain chunks a peer must be in possession of the metadata of the swarm. This metadata will be stored in a JSON-structured file, a file extension of .p2pmeta is suggested. See “Examples/example.p2pmeta” for an example .p2pmeta file.

Swarm Metadata	Description
Hash Type	Identifies the checksum algorithm. Important for future security, SHA-256 may eventually not be secure enough.
Metadata Hash	Acts as an ID for the Swarm, ensures that an innocent peer won’t join a swarm with a corrupt metadata file. This digest protects the HashType, SM Hostname, filenames, chunk hashes and file digests for each file/chunk.
Swarm Manager Hostname	A peer joining the swarm will register with this Swarm Manager.
List of file metadata:	
Filename	Used to correctly name the downloaded file
File Hash	Ensure the entire file is transferred and pieced together correctly
List of chunk metadata:	
Chunk size	Allows file size to be calculated
Chunk hash	Allows integrity of chunk data received from peer to be verified.

2.2 Distribution Architecture

The distribution system will contain two main applications: The Peer and the Swarm Manager. A smaller third application will create the initial metadata file based on input files.

2.2.1 Swarm Manager

The Swarm Manager will maintain a list of active peers which have registered for the swarm. Peers will send a register command regularly to stay on the list of peers for a particular swarm. Peers will then be able to request the list of active peers for a swarm. A swarm is uniquely identified by the Metadata hash in the initial metadata.

2.2.2 Peer

When started, each peer will use the information in the .p2pmeta file to check for the existence of the files to be downloaded. If the files do not exist, the peer will create them and allocate enough hard drive space for the download. If the files exist the peer will verify the integrity of the files and chunks to determine which file(s) or chunk(s) are missing. This architecture should support pausing/resuming of downloads without any issues and without requiring any special shutdown code to run (supporting sudden power off).

2.2.2.1 Peer Behaviour

Peers looking to acquire chunks in the swarm will connect to the Swarm Manager using the swarm metadata. The peer will then connect to other peers using information obtained from the Swarm Manager.

When a peer connects to another peer, they will exchange a list of chunks they each possess. The pair can then request appropriate chunks from each other simultaneously. When both of these peers are complete, they will disconnect from each other.

The algorithms for peer selection and chunk selection will be important to ensure efficient use of resources in the swarm. A very basic algorithm will be used first (choose the lowest chunk ID/first peer in list), but if time allows this should be improved later.

2.3 Network Protocol Design

2.3.1 Swarm Manager Protocol

JSON is used to encode the messages sent to/from the Swarm Manager. JSON is a good candidate for this as it is a widely implemented data exchange format. All data exchanged at this step is Human readable, allowing for easy encoding and handling. The human readability should ensure it is easy to implement and debug the Swarm Manager. JSON if handled correctly is also very compatible with future changes, as old clients should just ignore new fields in the JSON object.

See "Examples/Swarm Manager Messages.json" for example messages the Swarm Manager handles/sends.

2.3.2 Peer to peer protocol

Unlike the Swarm Manager protocol, peers will exchange binary data. I considered two ideas for this protocol: firstly an FTP-like system where JSON would be exchanged on a "control" socket, and a separate socket would be set up for the sending of binary data. The second idea was to have a small header which split up control vs data messages on the same socket.

Operating multiple sockets between peers would make the protocol simpler, but it would make the program harder to allow through firewalls (Multiple ports would have to be opened). Using a single socket would also reduce the latency between agreeing to exchange chunk data and actually sending/receiving chunk data as TCP sockets have a 3-way handshake which could slow things down.

2.3.2.1 Peer to Peer Message format

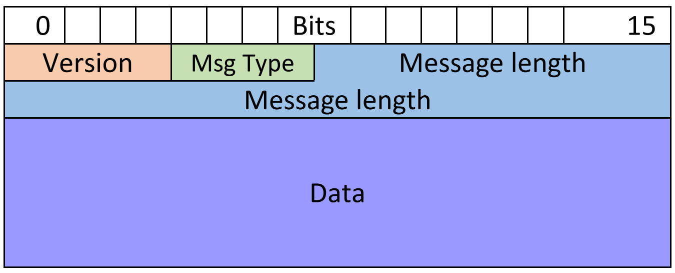


FIGURE 2.0 – P2P MESSAGE HEADER FORMAT.

In this implementation, the version is always 0000, and the message type is either 0001(Data) or 0000(Control). For control messages, Data should be interpreted as a JSON object. See "Examples/Peer Messages.json" for examples.

For Data messages, the message data should be interpreted as follows.

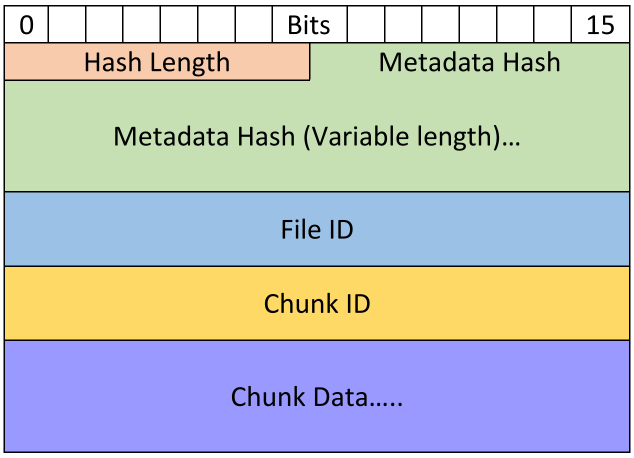


FIGURE 2.1 – P2P DATA MESSAGE SUB-HEADER.

3 Implementation

3.1 Testing

Two main testing strategies were used: Unit testing and some more general ‘white-box’ testing.

Unit testing in this project is mainly focused on areas in which subtle bugs would cause potentially hard to track down problems: Message Parsing, File Parsing and Message serialisation. Some of the Swarm Manager general logic has good test coverage as well as I tested out Mockito, a mocking-based unit testing framework. I found it to be slightly less than convenient though, so the Peer code has less good test coverage.

More general testing was performed by testing specific features (Such as requesting chunks from multiple peers, or resuming a partial download) at a time. By attempting to minimise other variants it helped make debugging just about possible. In general I found debugging P2P networking code very challenging due to the high numbers of connections, threads and possible points of failure. One strategy for preventing hard-to-debug problems, which I worked out quite late into development, was to treat one peer-to-peer connection as very disposable, shutting down the socket as soon as anything even maybe went wrong (Short timeouts, etc). This behaviour reduced problems such as chunks being stuck “INPROGRESS” and minimised issues relating to the java program not shutting down because a thread wouldn’t exit correctly and then would block on join().

3.2 Issues Encountered

3.2.1 Chunk & Peer selection algorithms

During an early pass of the peer code the Chunk selection and peer selection algorithms were drafted as roughly: “Choose an available Chunk/Peer with the lowest ID number”. This “algorithm” made it all the way through the main development process and it wasn’t until fairly extensive testing (10-15 nodes transferring 250MB+ files) that the problems of this method really began to show.

The most obvious side effect of these basic algorithms was in peer selection. If the peer at the top of the peer list (as retrieved from the swarm manager) disconnected, then a peer still left in the swarm is stuck attempting to connect to this peer until the Swarm Manager finally prunes the disconnected peer from the list. This issue could also have been avoided by using an effective choking system. My choking implementation was ineffective because the results of choking a peer were overwritten by the frequent peer list downloads from the swarm manager.

4 Extras

4.1 Compression

Compressing certain files would lead to massive reductions in transfer sizes, but the science of applying appropriate compression is complex and would be best suited to an entirely separate application. By integrating compression into the metagen step, or even before that, large speed gains could be made without over complicating this application. Another method for providing compression might be to use TLS for secure communication between peers/swarm manager. TLS could be used with compression enabled in this scenario (The CRIME exploit which forces HTTPS to disable compression is not applicable here as attacker could not inject partial data into the stream). This compression would provide significant benefit to text files, the overhead involved will likely make transferring highly compressed (video) files slower. The privacy benefits may outweigh these negatives though.

4.2 Multi-Threaded

The application is multi-threaded to make best use of any available network bandwidth. Each socket will block a read thread to ensure a response is processed as quickly as possible and each Peer-to-Peer connection has a separate write thread to allow writes to be issued from multiple threads. Designing & developing the program to be thread-safe was challenging, and certainly not without problems. I found java's **`synchronized`** keyword, when applied correctly, extremely useful for this.

4.3 Security / Data Integrity

Through careful use of message digests, it should not be possible for a malicious peer to inject bad chunks into another peer's download. Any chunk failing the pre-supplied checksum will be rejected and re-downloaded. The only point of trust is the .p2pmeta file, which must be securely transferred to the node expecting to join the swarm.

4.4 Fault Tolerance and Download Resume support

Peers can connect and disconnect at any point, providing excellent fault tolerance and download resume support. Every other peer's download will complete so long as there is at least one copy of each chunk in the remaining swarm. This aspect of the system could be made more tolerant by using a rarest-first chunk selection algorithm. Using this algorithm would speed up the time it takes between the swarm starting, and the time where each chunk exists on multiple nodes.

5 Evaluation

5.1 Functionality

I'm extremely happy with the functionality of this practical – it is a fully functioning peer to peer file distribution system with some useful additions. For a while during development, not long after my first transfers went through, I was disappointed by the reliability of the system as there was a ~50% error rate for a 15node transfer. However after deciding to close sockets on a short timeout for peers, and pinning down a thread synchronisation bug this error rate went down considerably. One feature which is missing is verifying the contents of the .p2pmeta file on load. The hash is read in, but the file is not verified using it. If I had significant time to add more features to this project, I'd like to explore the possibility of a cross-over between Multicast and P2P. Being able to take advantage of Multicast on a LAN, whilst still being able to participate in the swarm over the internet could be an extremely efficient method of replicating files.

5.2 Performance

Overall, this system scales extremely well compared to a straight serial TCP transfer to multiple nodes. Before implementing the Random Peer/Chunk selection, I was slightly disappointed with the speed of file transfers, consistently less than 30% link utilisation. Figure 5.0 below shows how much of a speed up adding even Random peer/chunk selection made Figure 6.0 in the appendix shows that it continues to scale even up to 70ish nodes. This P2P system is notably slower than Herd at distributing to one node. I think it would be possible to produce some considerable speed gains by making some changes to the protocol (reducing control message sizes, queueing chunk requests), implementing proper peer choking and implementing a better chunk selection algorithm (E.g. Rarest first. Perhaps also requesting last few chunks from all peers to speed up the end).

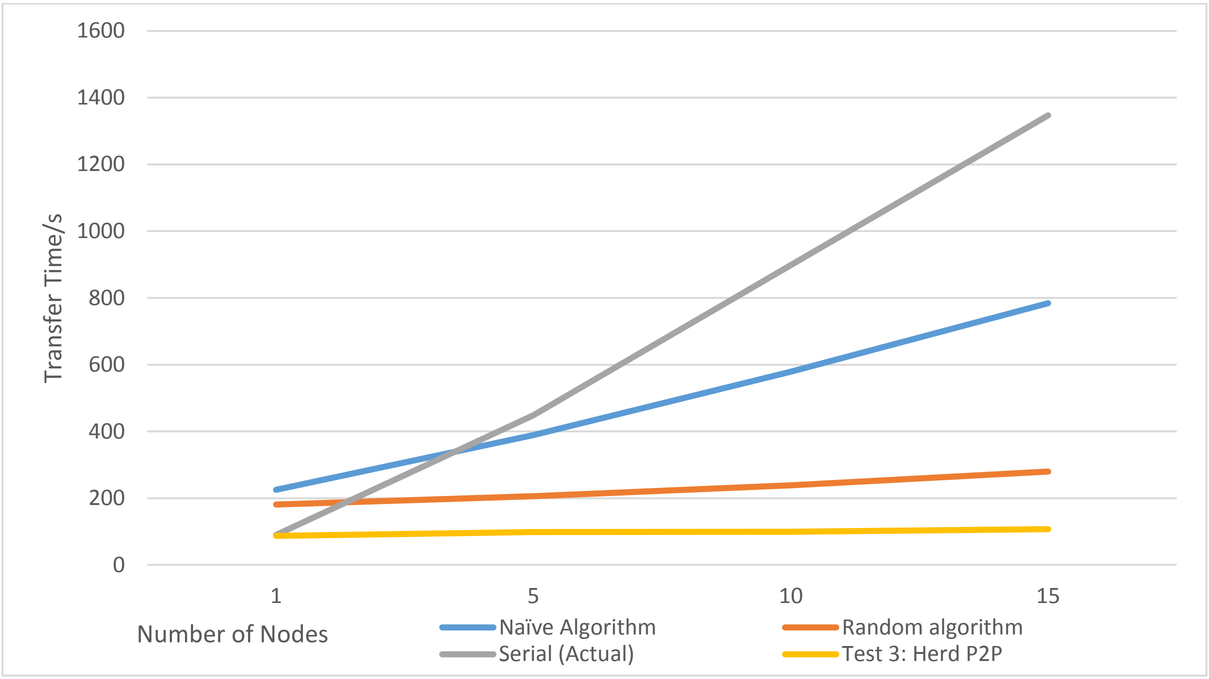


FIGURE 5.0 – GRAPH PLOTTING PERFORMANCE OF THIS P2P FILE DISTRIBUTION APPLICATION PERFORMANCE (NAÏVE AND RANDOM SELECTION ALGORITHMS) AGAINST RESULTS FROM FIGURE 1.1.

5.3 Code Quality

I'm not quite as happy with the quality of the code as I am with the functionality, the iterative development approach I took has meant that although at a high level the system is well designed, as features were layered onto the Peer, technical debt accumulated and code quality declined. The UI did not have as much design time put into it as it should have, and as a result it isn't as nice as it could be. The console output and the "business-logic" are quite tightly-coupled. Code Quality is the number 1 area this project could be improved on. If I had more time to spend on it, this is where I would start:

- Develop a UI Event Handler for events such as onPeerConnection/Disconnection, onChunkAcquisition, onDownloadComplete etc. This would make it easy to tidy up the user interface, and un-couple the code and console output.
- Move the Swarm Manager communication to a different thread. Updating the peer list every half second is un-necessary and eliminates any benefit of our system for removing peers from the list if we can't connect to them.
- Analyse PeerConnection for a way to break it up into smaller tasks. Complex & error-prone lock/chunk reserving logic in there.

6 Appendix

6.1 Graph Figures – Data and Methods

6.1.1 Figure 1.0

Data found in Analysis sheet of Results.xlsx

6.1.1.1 Estimation Calculations

Chunk Size(**C**)	Total File Size (**D**)	Bandwidth (**B**)
262144	1073741824	12500000

Architecture	Total transfer time formula	Explanation
Serial	$t(n) = \frac{nD}{B} = O(n)$	Bit-containing packets served serially. Total bits to serve $= n \times B$
Multicast	$t(n) = \frac{D}{B} = O(1)$	Same packets served to all clients simultaneously.
P2P (un-chunked)	$t(n) = \sum_{i=1}^{n} \frac{D}{iB}$	Upload begins only after download is complete. Each subsequent client downloads with 1 more seeder than the last.
P2P (chunked)	$let\ x = \frac{D}{C},\ \ t(n) = \frac{C(x + N - 1)}{B}$	The optimal architecture for speed is a chain of peers, $(n-1)$ links long. The total time is the time until the last peer has downloaded the last chunk. This will be the time taken to transfer the last chunk to the first link + the length of the chain. $x =$ number of chunks

Figure 1.1

Data found in Analysis Empirical Results sheet of Results.xlsx. The data transferred was 1GB in size.

Serial tests were performed by running `iperf -su` on one node, and running `iperf –uc “hostname” –b 200Mb –n 1GB` on 1, 5, 10 or 15 other nodes.

Multicast tests were performed the other way around to serial. The following command was run on each receiving node:

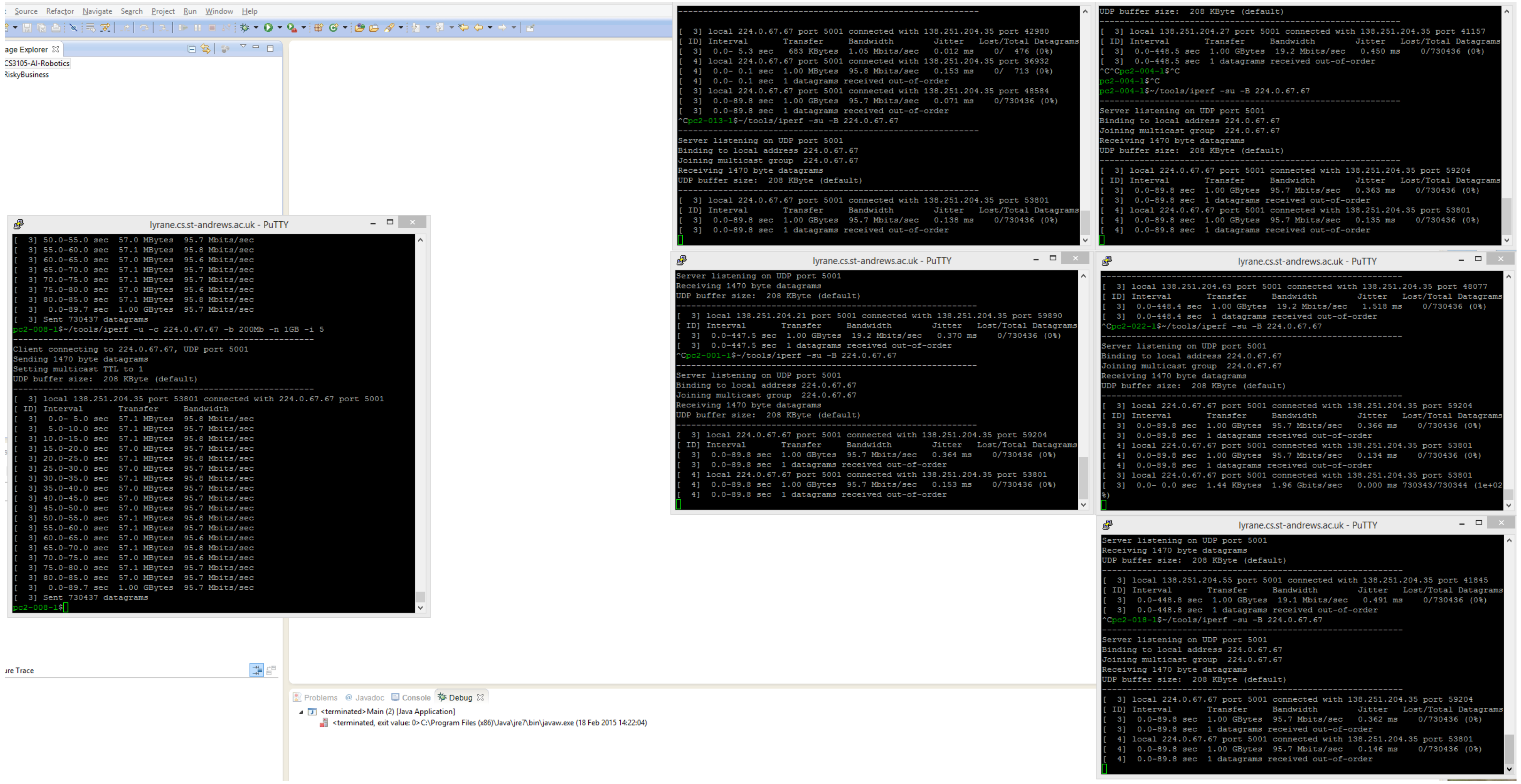
```
iperf -su 224.0.67.67
```

And this was run on each node:

```
iperf -u -c 224.0.67.67 -b 200Mb -n 1GB -i 5
```

Herd was run using: `herd /tmp/sample2.txt /tmp/sample.txt nodes.txt`

6.1.1.2 *Multicast Testing using iperf*


Source Refactor Navigate Search Project Run Window Help
age Explorer
CS3105-AI-Robotics
RiskyBusiness
ıre Trace
Problems Javadoc Console Debug
lyrane.cs.st-andrews.ac.uk - PuTTY

```
<terminated>Main (2) [Java Application]
<terminated, exit value: 0>C:\Program Files (x86)\Java\jre7\bin\javaw.exe (18 Feb 2015 14:22:04)
```

```
[  3] 50.0-55.0 sec  57.0 MBytes  95.7 Mbits/sec
[  3] 55.0-60.0 sec  57.1 MBytes  95.8 Mbits/sec
[  3] 60.0-65.0 sec  57.0 MBytes  95.6 Mbits/sec
[  3] 65.0-70.0 sec  57.1 MBytes  95.7 Mbits/sec
[  3] 70.0-75.0 sec  57.1 MBytes  95.7 Mbits/sec
[  3] 75.0-80.0 sec  57.0 MBytes  95.6 Mbits/sec
[  3] 80.0-85.0 sec  57.1 MBytes  95.8 Mbits/sec
[  3]  0.0-89.7 sec  1.00 GBytes  95.7 Mbits/sec
[  3] Sent 730437 datagrams
pc2-008-l$~/tools/iperf -u -c 224.0.67.67 -b 200Mb -n 1GB -i 5
------------------------------------------------------------
Client connecting to 224.0.67.67, UDP port 5001
Sending 1470 byte datagrams
Setting multicast TTL to 1
UDP buffer size:  208 KByte (default)
------------------------------------------------------------
[  3] local 138.251.204.35 port 53801 connected with 224.0.67.67 port 5001
[ ID] Interval       Transfer     Bandwidth
[  3]  0.0- 5.0 sec  57.1 MBytes  95.8 Mbits/sec
[  3]  5.0-10.0 sec  57.1 MBytes  95.7 Mbits/sec
[  3] 10.0-15.0 sec  57.1 MBytes  95.8 Mbits/sec
[  3] 15.0-20.0 sec  57.0 MBytes  95.7 Mbits/sec
[  3] 20.0-25.0 sec  57.1 MBytes  95.8 Mbits/sec
[  3] 25.0-30.0 sec  57.0 MBytes  95.7 Mbits/sec
[  3] 30.0-35.0 sec  57.1 MBytes  95.8 Mbits/sec
[  3] 35.0-40.0 sec  57.0 MBytes  95.7 Mbits/sec
[  3] 40.0-45.0 sec  57.0 MBytes  95.7 Mbits/sec
[  3] 45.0-50.0 sec  57.0 MBytes  95.7 Mbits/sec
[  3] 50.0-55.0 sec  57.1 MBytes  95.8 Mbits/sec
[  3] 55.0-60.0 sec  57.1 MBytes  95.7 Mbits/sec
[  3] 60.0-65.0 sec  57.0 MBytes  95.6 Mbits/sec
[  3] 65.0-70.0 sec  57.1 MBytes  95.8 Mbits/sec
[  3] 70.0-75.0 sec  57.0 MBytes  95.6 Mbits/sec
[  3] 75.0-80.0 sec  57.1 MBytes  95.7 Mbits/sec
[  3] 80.0-85.0 sec  57.0 MBytes  95.7 Mbits/sec
[  3]  0.0-89.7 sec  1.00 GBytes  95.7 Mbits/sec
[  3] Sent 730437 datagrams
pc2-008-l$
```

```
------------------------------------------------------------

[  3] local 224.0.67.67 port 5001 connected with 138.251.204.35 port 42980
[ ID] Interval       Transfer     Bandwidth        Jitter   Lost/Total Datagrams
[  3]  0.0- 5.3 sec   683 KBytes  1.05 Mbits/sec   0.012 ms    0/  476 (0%)
[  4] local 224.0.67.67 port 5001 connected with 138.251.204.35 port 36932
[  4]  0.0- 0.1 sec  1.00 MBytes  95.8 Mbits/sec   0.153 ms    0/  713 (0%)
[  4]  0.0- 0.1 sec  1 datagrams received out-of-order
[  3] local 224.0.67.67 port 5001 connected with 138.251.204.35 port 48584
[  3]  0.0-89.8 sec  1.00 GBytes  95.7 Mbits/sec   0.071 ms    0/730436 (0%)
[  3]  0.0-89.8 sec  1 datagrams received out-of-order
^Cpc2-013-l$~/tools/iperf -su -B 224.0.67.67
------------------------------------------------------------
Server listening on UDP port 5001
Binding to local address 224.0.67.67
Joining multicast group  224.0.67.67
Receiving 1470 byte datagrams
UDP buffer size:  208 KByte (default)
------------------------------------------------------------
[  3] local 224.0.67.67 port 5001 connected with 138.251.204.35 port 53801
[ ID] Interval       Transfer     Bandwidth        Jitter   Lost/Total Datagrams
[  3]  0.0-89.8 sec  1.00 GBytes  95.7 Mbits/sec   0.138 ms    0/730436 (0%)
[  3]  0.0-89.8 sec  1 datagrams received out-of-order
```

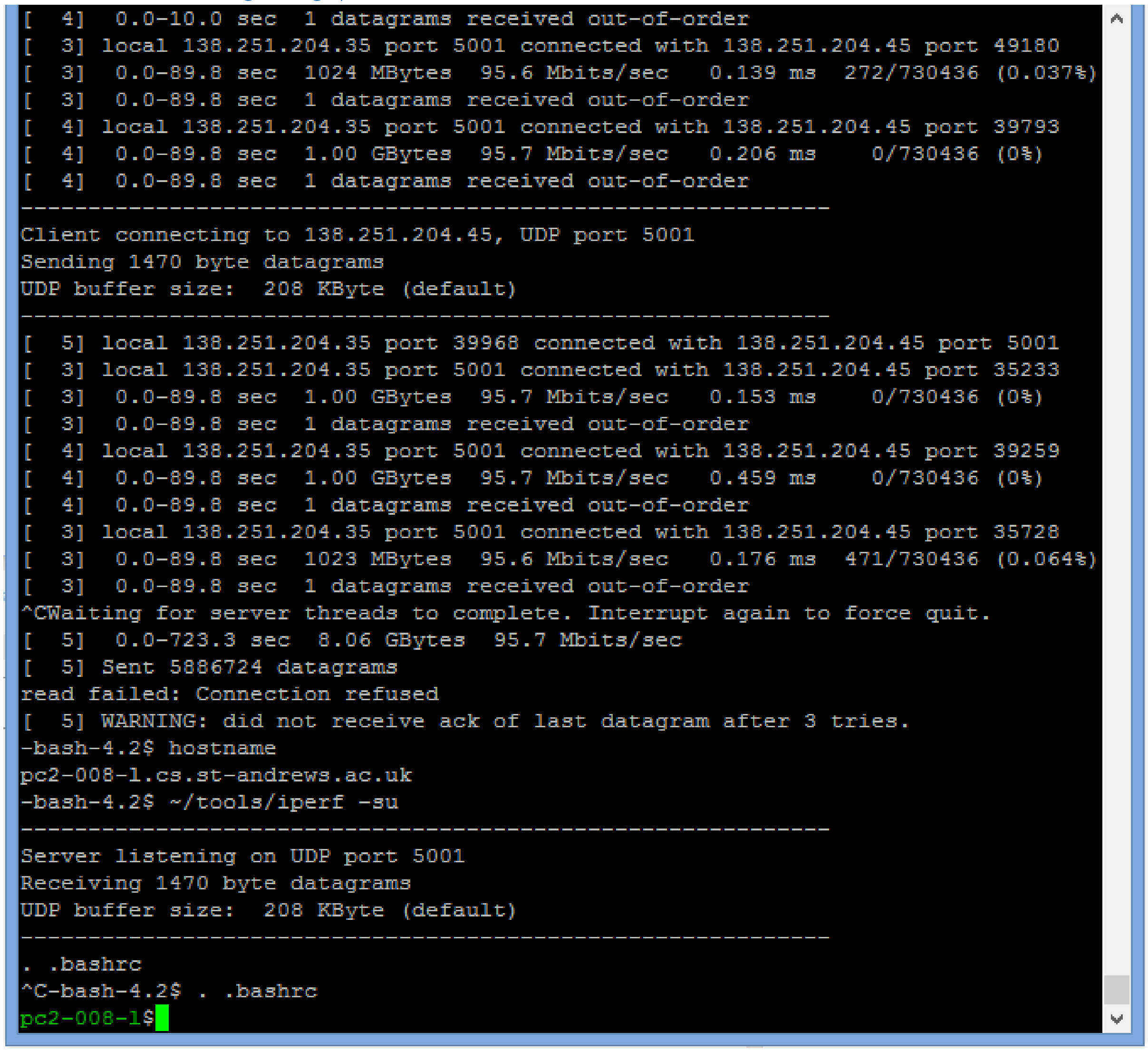
```
UDP buffer size:  208 KByte (default)
------------------------------------------------------------
[  3] local 138.251.204.27 port 5001 connected with 138.251.204.35 port 41157
[ ID] Interval       Transfer     Bandwidth        Jitter   Lost/Total Datagrams
[  3]  0.0-448.5 sec  1.00 GBytes  19.2 Mbits/sec   0.450 ms    0/730436 (0%)
[  3]  0.0-448.5 sec  1 datagrams received out-of-order
^C^Cpc2-004-l$^C
pc2-004-l$^C
pc2-004-l$~/tools/iperf -su -B 224.0.67.67
------------------------------------------------------------
Server listening on UDP port 5001
Binding to local address 224.0.67.67
Joining multicast group  224.0.67.67
Receiving 1470 byte datagrams
UDP buffer size:  208 KByte (default)
------------------------------------------------------------
[  3] local 224.0.67.67 port 5001 connected with 138.251.204.35 port 59204
[ ID] Interval       Transfer     Bandwidth        Jitter   Lost/Total Datagrams
[  3]  0.0-89.8 sec  1.00 GBytes  95.7 Mbits/sec   0.363 ms    0/730436 (0%)
[  3]  0.0-89.8 sec  1 datagrams received out-of-order
[  4] local 224.0.67.67 port 5001 connected with 138.251.204.35 port 53801
[  4]  0.0-89.8 sec  1.00 GBytes  95.7 Mbits/sec   0.135 ms    0/730436 (0%)
[  4]  0.0-89.8 sec  1 datagrams received out-of-order
```

```
Server listening on UDP port 5001
Receiving 1470 byte datagrams
UDP buffer size:  208 KByte (default)
------------------------------------------------------------
[  3] local 138.251.204.21 port 5001 connected with 138.251.204.35 port 59890
[ ID] Interval       Transfer     Bandwidth        Jitter   Lost/Total Datagrams
[  3]  0.0-447.5 sec  1.00 GBytes  19.2 Mbits/sec   0.370 ms    0/730436 (0%)
[  3]  0.0-447.5 sec  1 datagrams received out-of-order
^Cpc2-001-l$~/tools/iperf -su -B 224.0.67.67
------------------------------------------------------------
Server listening on UDP port 5001
Binding to local address 224.0.67.67
Joining multicast group  224.0.67.67
Receiving 1470 byte datagrams
UDP buffer size:  208 KByte (default)
------------------------------------------------------------
[  3] local 224.0.67.67 port 5001 connected with 138.251.204.35 port 59204
[ ID] Interval       Transfer     Bandwidth        Jitter   Lost/Total Datagrams
[  3]  0.0-89.8 sec  1.00 GBytes  95.7 Mbits/sec   0.364 ms    0/730436 (0%)
[  3]  0.0-89.8 sec  1 datagrams received out-of-order
[  4] local 224.0.67.67 port 5001 connected with 138.251.204.35 port 53801
[  4]  0.0-89.8 sec  1.00 GBytes  95.7 Mbits/sec   0.153 ms    0/730436 (0%)
[  4]  0.0-89.8 sec  1 datagrams received out-of-order
```

```
------------------------------------------------------------
[  3] local 138.251.204.63 port 5001 connected with 138.251.204.35 port 48077
[ ID] Interval       Transfer     Bandwidth        Jitter   Lost/Total Datagrams
[  3]  0.0-448.4 sec  1.00 GBytes  19.2 Mbits/sec   1.518 ms    0/730436 (0%)
[  3]  0.0-448.4 sec  1 datagrams received out-of-order
^Cpc2-022-l$~/tools/iperf -su -B 224.0.67.67
------------------------------------------------------------
Server listening on UDP port 5001
Binding to local address 224.0.67.67
Joining multicast group  224.0.67.67
Receiving 1470 byte datagrams
UDP buffer size:  208 KByte (default)
------------------------------------------------------------
[  3] local 224.0.67.67 port 5001 connected with 138.251.204.35 port 59204
[ ID] Interval       Transfer     Bandwidth        Jitter   Lost/Total Datagrams
[  3]  0.0-89.8 sec  1.00 GBytes  95.7 Mbits/sec   0.366 ms    0/730436 (0%)
[  3]  0.0-89.8 sec  1 datagrams received out-of-order
[  4] local 224.0.67.67 port 5001 connected with 138.251.204.35 port 53801
[  4]  0.0-89.8 sec  1.00 GBytes  95.7 Mbits/sec   0.134 ms    0/730436 (0%)
[  4]  0.0-89.8 sec  1 datagrams received out-of-order
[  3] local 224.0.67.67 port 5001 connected with 138.251.204.35 port 53801
[  3]  0.0- 0.0 sec  1.44 KBytes  1.96 Gbits/sec   0.000 ms 730343/730344 (1e+02
%)
```

```
Server listening on UDP port 5001
Receiving 1470 byte datagrams
UDP buffer size:  208 KByte (default)
------------------------------------------------------------
[  3] local 138.251.204.55 port 5001 connected with 138.251.204.35 port 41845
[ ID] Interval       Transfer     Bandwidth        Jitter   Lost/Total Datagrams
[  3]  0.0-448.8 sec  1.00 GBytes  19.1 Mbits/sec   0.491 ms    0/730436 (0%)
[  3]  0.0-448.8 sec  1 datagrams received out-of-order
^Cpc2-018-l$~/tools/iperf -su -B 224.0.67.67
------------------------------------------------------------
Server listening on UDP port 5001
Binding to local address 224.0.67.67
Joining multicast group  224.0.67.67
Receiving 1470 byte datagrams
UDP buffer size:  208 KByte (default)
------------------------------------------------------------
[  3] local 224.0.67.67 port 5001 connected with 138.251.204.35 port 59204
[ ID] Interval       Transfer     Bandwidth        Jitter   Lost/Total Datagrams
[  3]  0.0-89.8 sec  1.00 GBytes  95.7 Mbits/sec   0.362 ms    0/730436 (0%)
[  3]  0.0-89.8 sec  1 datagrams received out-of-order
[  4] local 224.0.67.67 port 5001 connected with 138.251.204.35 port 53801
[  4]  0.0-89.8 sec  1.00 GBytes  95.7 Mbits/sec   0.146 ms    0/730436 (0%)
[  4]  0.0-89.8 sec  1 datagrams received out-of-order
```

6.1.1.3 Serial testing using iperf



```
[  4]  0.0-10.0 sec  1 datagrams received out-of-order
[  3] local 138.251.204.35 port 5001 connected with 138.251.204.45 port 49180
[  3]  0.0-89.8 sec  1024 MBytes  95.6 Mbits/sec   0.139 ms  272/730436 (0.037%)
[  3]  0.0-89.8 sec  1 datagrams received out-of-order
[  4] local 138.251.204.35 port 5001 connected with 138.251.204.45 port 39793
[  4]  0.0-89.8 sec  1.00 GBytes  95.7 Mbits/sec   0.206 ms    0/730436 (0%)
[  4]  0.0-89.8 sec  1 datagrams received out-of-order
------------------------------------------------------------
Client connecting to 138.251.204.45, UDP port 5001
Sending 1470 byte datagrams
UDP buffer size:  208 KByte (default)
------------------------------------------------------------
[  5] local 138.251.204.35 port 39968 connected with 138.251.204.45 port 5001
[  3] local 138.251.204.35 port 5001 connected with 138.251.204.45 port 35233
[  3]  0.0-89.8 sec  1.00 GBytes  95.7 Mbits/sec   0.153 ms    0/730436 (0%)
[  3]  0.0-89.8 sec  1 datagrams received out-of-order
[  4] local 138.251.204.35 port 5001 connected with 138.251.204.45 port 39259
[  4]  0.0-89.8 sec  1.00 GBytes  95.7 Mbits/sec   0.459 ms    0/730436 (0%)
[  4]  0.0-89.8 sec  1 datagrams received out-of-order
[  3] local 138.251.204.35 port 5001 connected with 138.251.204.45 port 35728
[  3]  0.0-89.8 sec  1023 MBytes  95.6 Mbits/sec   0.176 ms  471/730436 (0.064%)
[  3]  0.0-89.8 sec  1 datagrams received out-of-order
^CWaiting for server threads to complete. Interrupt again to force quit.
[  5]  0.0-723.3 sec  8.06 GBytes  95.7 Mbits/sec
[  5] Sent 5886724 datagrams
read failed: Connection refused
[  5] WARNING: did not receive ack of last datagram after 3 tries.
-bash-4.2$ hostname
pc2-008-l.cs.st-andrews.ac.uk
-bash-4.2$ ~/tools/iperf -su
------------------------------------------------------------
Server listening on UDP port 5001
Receiving 1470 byte datagrams
UDP buffer size:  208 KByte (default)
------------------------------------------------------------
. .bashrc
^C-bash-4.2$ . .bashrc
pc2-008-l$
```

6.1.2 Figure 5.0

Data found in My Implementation sheet of Results.xlsx. A 256MB file was transferred for these tests, and the average was multiplied by 4 for comparison with the previously recorded values in Figure 1.1.

See Example Output for screenshots of this testing.

To perform the tests needed for Figure 5.0 and 6.0 I ran the following command from my host server:

```
cat nodes.txt | xargs -P15 -I"HOST" -n1 ssh HOST "~/cs3000/DCN-CS3106/Practical\ 1/run.sh"
```

Run.sh contained:

```
java -jar ~/cs3000/DCN-CS3102/Practical\ 1/P2PDistribution/peer.jar ~/cs3000/DCN-CS3102/Practical\ 1/sample2.p2pmeta /var/tmp/output/
```

6.1.3 Figure 6.0

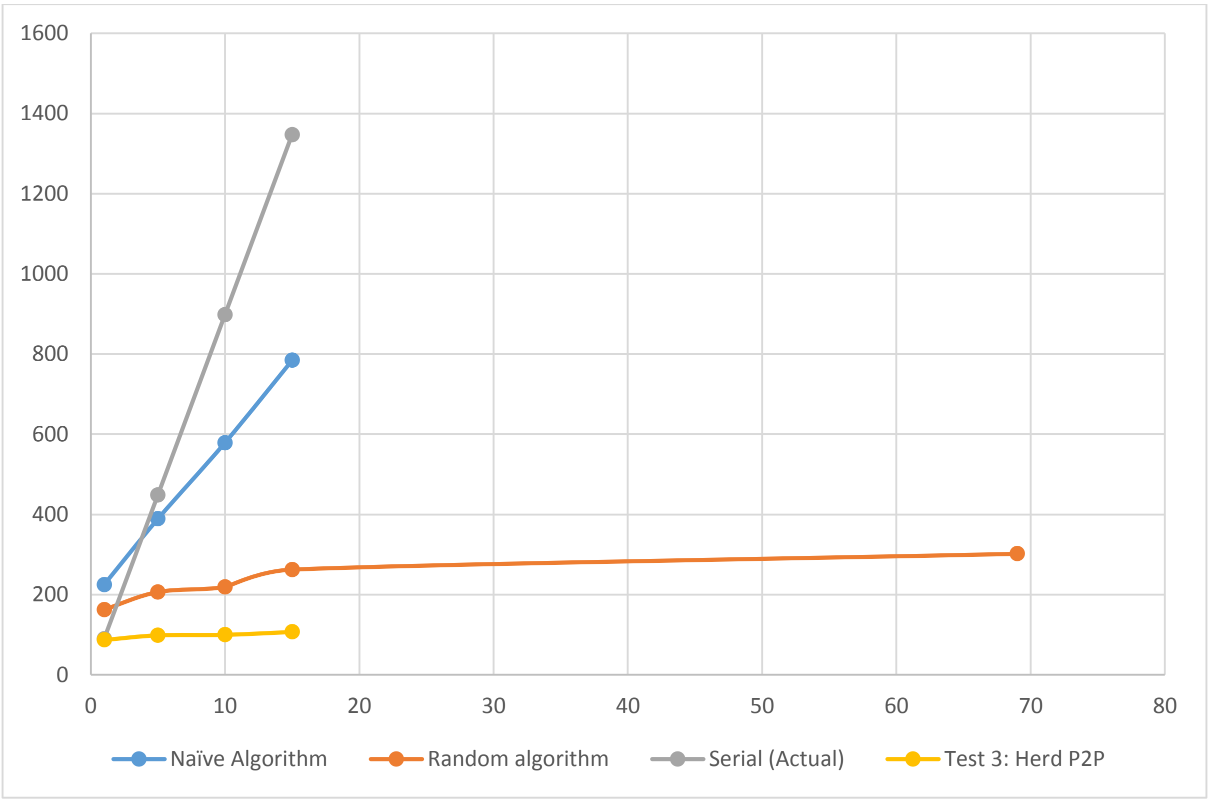
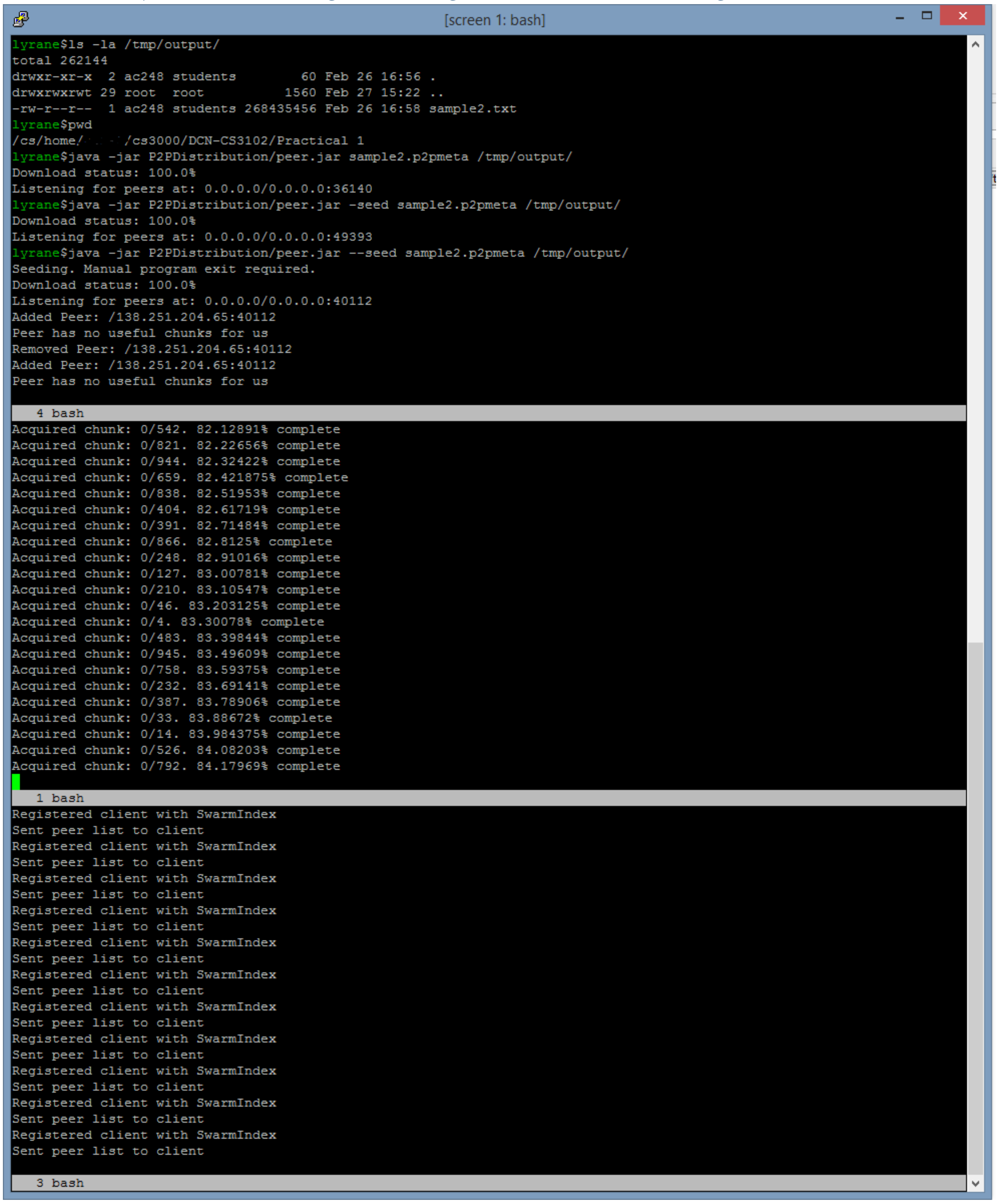


FIGURE 6.0 – GRAPH SHOWING EXTREME SCALING PERFORMANCE OF THIS P2P SYSTEM.

6.2 Example Output

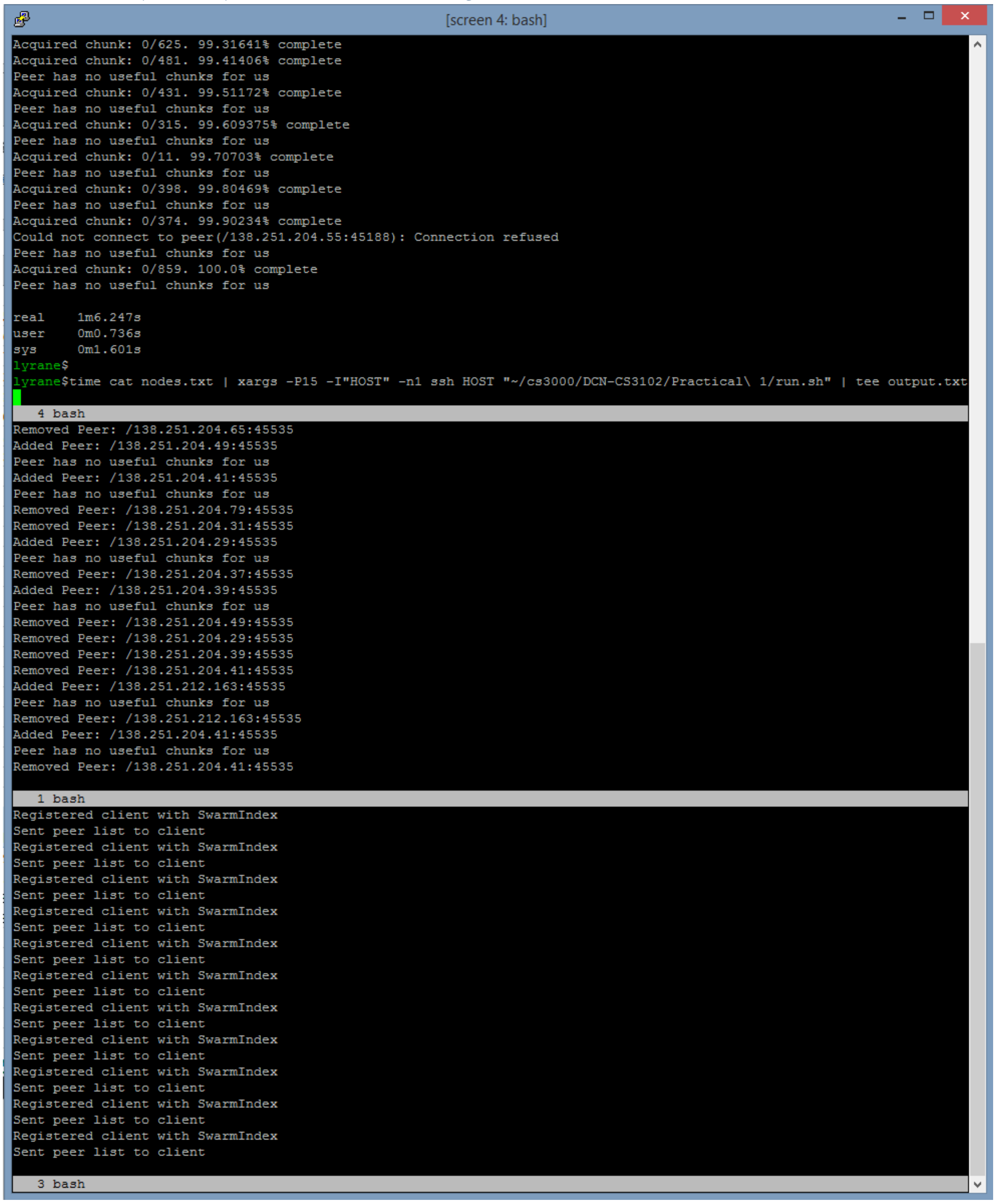
6.2.1 Example Swarm Manager, Seeding node and Downloading node



[screen 1: bash]

```
lyrane$ls -la /tmp/output/
total 262144
drwxr-xr-x  2 ac248 students        60 Feb 26 16:56 .
drwxrwxrwt 29 root  root          1560 Feb 27 15:22 ..
-rw-r--r--  1 ac248 students 268435456 Feb 26 16:58 sample2.txt
lyrane$pwd
/cs/home/[redacted]/cs3000/DCN-CS3102/Practical 1
lyrane$java -jar P2PDistribution/peer.jar sample2.p2pmeta /tmp/output/
Download status: 100.0%
Listening for peers at: 0.0.0.0/0.0.0.0:36140
lyrane$java -jar P2PDistribution/peer.jar -seed sample2.p2pmeta /tmp/output/
Download status: 100.0%
Listening for peers at: 0.0.0.0/0.0.0.0:49393
lyrane$java -jar P2PDistribution/peer.jar --seed sample2.p2pmeta /tmp/output/
Seeding. Manual program exit required.
Download status: 100.0%
Listening for peers at: 0.0.0.0/0.0.0.0:40112
Added Peer: /138.251.204.65:40112
Peer has no useful chunks for us
Removed Peer: /138.251.204.65:40112
Added Peer: /138.251.204.65:40112
Peer has no useful chunks for us

   4 bash
Acquired chunk: 0/542. 82.12891% complete
Acquired chunk: 0/821. 82.22656% complete
Acquired chunk: 0/944. 82.32422% complete
Acquired chunk: 0/659. 82.421875% complete
Acquired chunk: 0/838. 82.51953% complete
Acquired chunk: 0/404. 82.61719% complete
Acquired chunk: 0/391. 82.71484% complete
Acquired chunk: 0/866. 82.8125% complete
Acquired chunk: 0/248. 82.91016% complete
Acquired chunk: 0/127. 83.00781% complete
Acquired chunk: 0/210. 83.10547% complete
Acquired chunk: 0/46. 83.203125% complete
Acquired chunk: 0/4. 83.30078% complete
Acquired chunk: 0/483. 83.39844% complete
Acquired chunk: 0/945. 83.49609% complete
Acquired chunk: 0/758. 83.59375% complete
Acquired chunk: 0/232. 83.69141% complete
Acquired chunk: 0/387. 83.78906% complete
Acquired chunk: 0/33. 83.88672% complete
Acquired chunk: 0/14. 83.984375% complete
Acquired chunk: 0/526. 84.08203% complete
Acquired chunk: 0/792. 84.17969% complete

   1 bash
Registered client with SwarmIndex
Sent peer list to client
Registered client with SwarmIndex
Sent peer list to client
Registered client with SwarmIndex
Sent peer list to client
Registered client with SwarmIndex
Sent peer list to client
Registered client with SwarmIndex
Sent peer list to client
Registered client with SwarmIndex
Sent peer list to client
Registered client with SwarmIndex
Sent peer list to client
Registered client with SwarmIndex
Sent peer list to client
Registered client with SwarmIndex
Sent peer list to client
Registered client with SwarmIndex
Sent peer list to client
Registered client with SwarmIndex
Sent peer list to client

   3 bash
```

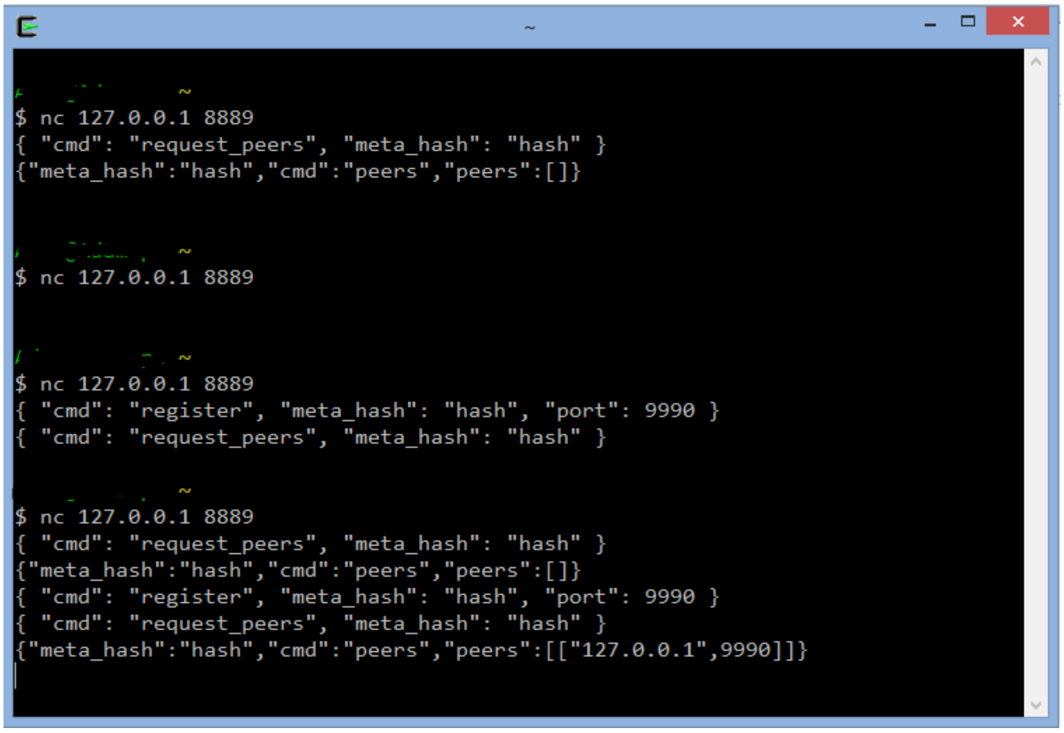
6.2.2 Example Output when downloading to 15 nodes



[screen 4: bash]

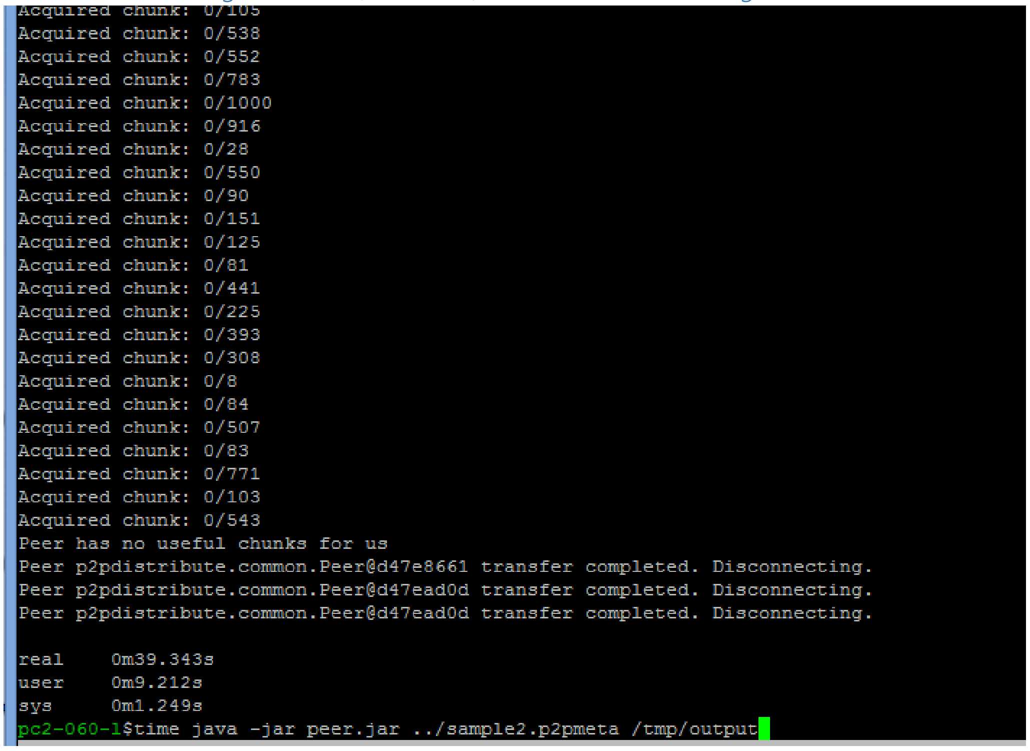
```
Acquired chunk: 0/625. 99.31641% complete
Acquired chunk: 0/481. 99.41406% complete
Peer has no useful chunks for us
Acquired chunk: 0/431. 99.51172% complete
Peer has no useful chunks for us
Acquired chunk: 0/315. 99.609375% complete
Peer has no useful chunks for us
Acquired chunk: 0/11. 99.70703% complete
Peer has no useful chunks for us
Acquired chunk: 0/398. 99.80469% complete
Peer has no useful chunks for us
Acquired chunk: 0/374. 99.90234% complete
Could not connect to peer(/138.251.204.55:45188): Connection refused
Peer has no useful chunks for us
Acquired chunk: 0/859. 100.0% complete
Peer has no useful chunks for us

real    1m6.247s
user    0m0.736s
sys     0m1.601s
lyrane$
lyrane$time cat nodes.txt | xargs -P15 -I"HOST" -n1 ssh HOST "~/cs3000/DCN-CS3102/Practical\ 1/run.sh" | tee output.txt

   4 bash
Removed Peer: /138.251.204.65:45535
Added Peer: /138.251.204.49:45535
Peer has no useful chunks for us
Added Peer: /138.251.204.41:45535
Peer has no useful chunks for us
Removed Peer: /138.251.204.79:45535
Removed Peer: /138.251.204.31:45535
Added Peer: /138.251.204.29:45535
Peer has no useful chunks for us
Removed Peer: /138.251.204.37:45535
Added Peer: /138.251.204.39:45535
Peer has no useful chunks for us
Removed Peer: /138.251.204.49:45535
Removed Peer: /138.251.204.29:45535
Removed Peer: /138.251.204.39:45535
Removed Peer: /138.251.204.41:45535
Added Peer: /138.251.212.163:45535
Peer has no useful chunks for us
Removed Peer: /138.251.212.163:45535
Added Peer: /138.251.204.41:45535
Peer has no useful chunks for us
Removed Peer: /138.251.204.41:45535

   1 bash
Registered client with SwarmIndex
Sent peer list to client
Registered client with SwarmIndex
Sent peer list to client
Registered client with SwarmIndex
Sent peer list to client
Registered client with SwarmIndex
Sent peer list to client
Registered client with SwarmIndex
Sent peer list to client
Registered client with SwarmIndex
Sent peer list to client
Registered client with SwarmIndex
Sent peer list to client
Registered client with SwarmIndex
Sent peer list to client
Registered client with SwarmIndex
Sent peer list to client
Registered client with SwarmIndex
Sent peer list to client
Registered client with SwarmIndex
Sent peer list to client

   3 bash
```

6.2.2.1 Testing Swarm Manager using netcat



```
[illegible] ~
$ nc 127.0.0.1 8889
{ "cmd": "request_peers", "meta_hash": "hash" }
{"meta_hash":"hash","cmd":"peers","peers":[]}


[illegible] ~
$ nc 127.0.0.1 8889


[illegible] ~
$ nc 127.0.0.1 8889
{ "cmd": "register", "meta_hash": "hash", "port": 9990 }
{ "cmd": "request_peers", "meta_hash": "hash" }

[illegible] ~
$ nc 127.0.0.1 8889
{ "cmd": "request_peers", "meta_hash": "hash" }
{"meta_hash":"hash","cmd":"peers","peers":[]}
{ "cmd": "register", "meta_hash": "hash", "port": 9990 }
{ "cmd": "request_peers", "meta_hash": "hash" }
{"meta_hash":"hash","cmd":"peers","peers":[["127.0.0.1",9990]]}
```

6.2.2.2 Recording P2P transfer times for 1 node (Random algorithm)



```
Acquired chunk: 0/105
Acquired chunk: 0/538
Acquired chunk: 0/552
Acquired chunk: 0/783
Acquired chunk: 0/1000
Acquired chunk: 0/916
Acquired chunk: 0/28
Acquired chunk: 0/550
Acquired chunk: 0/90
Acquired chunk: 0/151
Acquired chunk: 0/125
Acquired chunk: 0/81
Acquired chunk: 0/441
Acquired chunk: 0/225
Acquired chunk: 0/393
Acquired chunk: 0/308
Acquired chunk: 0/8
Acquired chunk: 0/84
Acquired chunk: 0/507
Acquired chunk: 0/83
Acquired chunk: 0/771
Acquired chunk: 0/103
Acquired chunk: 0/543
Peer has no useful chunks for us
Peer p2pdistribute.common.Peer@d47e8661 transfer completed. Disconnecting.
Peer p2pdistribute.common.Peer@d47ead0d transfer completed. Disconnecting.
Peer p2pdistribute.common.Peer@d47ead0d transfer completed. Disconnecting.

real    0m39.343s
user    0m9.212s
sys     0m1.249s
pc2-060-l$time java -jar peer.jar ../sample2.p2pmeta /tmp/output
```

7 References

Garrett, R., & Gadea, L. (2014, September 25). Herd: A single-command bittorrent distribution system, based on Twitter's Murder. Github. Retrieved from https://github.com/russss/Herd