

CS4105 Practical 2 : Discover and Download

Matriculation ID : 190015412

November 22, 2024

Contents

1	Introduction	2
2	General Infrastructure	2
3	Requirement 1 : Discover	2
3.1	Operation	2
3.2	Testing	2
3.2.1	Test R1.1	2
3.2.2	Test R1.2	3
3.2.3	Test R1.3	3
4	Requirement 2 : Search	4
4.1	Design	4
4.2	Operation	4
4.3	Testing	4
4.3.1	Test R2.1	4
4.3.2	Test R2.2	5
4.3.3	Test R2.3	5
4.3.4	Test R2.4	6
4.3.5	Test R2.5	6
4.3.6	Test R2.6	6
5	Requirement 3 : Download	7
5.1	Protocol Specification	7
5.1.1	Protocol Description	7
5.2	Design	8
5.3	Operation	8
5.4	Testing	9
5.4.1	Test R3.1	9
5.4.2	Test R3.2	9
5.4.3	Test R3.3	9
5.4.4	Test R3.4	10
5.4.5	Test R3.5	10
5.4.6	Test R3.6	11
5.4.7	Test R3.7	11
5.4.8	Test R3.8	11

6 Requirement 4 : Critical Analysis	12
6.1 Scalability	12
6.2 Performance	14
6.3 Privacy & Security	14

1 Introduction

The purpose of this practical is to develop a Local Area Network (LAN) based, distributed, file download application utilizing multicast communication over IPv6. In doing so, we design and implement a control-plane protocol to support the required functionalities of this application. Overall, I have implemented all four requirements of this practical for my application with consideration of scalability, performance, security and privacy aspects.

2 General Infrastructure

Generally, I have decided to use a dispatcher-based[2] approach to handle the various operations that the application must support. In this approach, when a message is received by the application, a central dispatcher process first inspects the message to determine its type (e.g search-request, advertisement, download-request, etc), and then delegates it to a message-specific queue to await processing. With this, I have separate queues for each type of message an application can receive. Using a thread pool, separate threads are used to process messages out of the queues continuously (at an interval). For example, all incoming advertisements are queued in a data structure, where a worker thread continuously iterates through the stored advertisements, inspects them, and removes the advertisements which have expired.

3 Requirement 1 : Discover

3.1 Operation

When a node executes the application, it joins an IPv6 multicast group and begins to send out advertisements containing information about its search and download capabilities. At the same time, the node listens of incoming advertisements from other nodes in the multicast group. At any time, the user can enter the command, `:nodes`, which will display all advertisements that current node has received which have not expired, corresponding to each node in the multicast group (other than itself).

3.2 Testing

3.2.1 Test R1.1

Test Purpose : to demonstrate the basic functionality of advertisements and multicast node discovery.

```
[filename | '.' | '..' | ':services' | ':nodes' | ':search' | ':download' | ':quit' | ':help']
:nodes
Node [1]
  Identifier : jz75@pc7-007-1.cs.st-andrews.ac.uk
  Port      : 4105
  Services  : search=true,download=true
Node [2]
  Identifier : jz75@pc7-033-1.cs.st-andrews.ac.uk
  Port      : 4105
  Services  : search=true,download=true
```

Figure 1: Sample output from executing the “:nodes” command in the command line interface. In this test, three different lab machines are running the application within the same LAN : pc7-033-1, pc7-005-1, and pc7-007-1. The output is from pc7-005-1.

```
jz75@pc7-005-1.cs.st-andrews.ac.uk Joined Multicast Group
tx-> :jz75@pc7-005-1.cs.st-andrews.ac.uk:1732144301841:20241120-231141.841:advertisement:4105:search=true,download=true:
rx-> :jz75@pc7-033-1.cs.st-andrews.ac.uk:1732144302642:20241120-231142.642:advertisement:4105:search=true,download=true:
tx-> :jz75@pc7-005-1.cs.st-andrews.ac.uk:1732144302841:20241120-231142.841:advertisement:4105:search=true,download=true:
rx-> :jz75@pc7-007-1.cs.st-andrews.ac.uk:1732144303453:20241120-231143.453:advertisement:4105:search=true,download=true:
```

Figure 2: Snippet of log file from the pc7-005-1 showing the transmission of tx and rx advertisements

3.2.2 Test R1.2

Test Purpose : to demonstrate the application’s function of updating old advertisements with newly received advertisements corresponding to the same node/identifier.

In the same session as the previous test (Test R1.1), the pc7-033-1 node updates its capabilities indicating that it does not support file download, thus altering its advertisement message from before in Test R1.1.

```
:nodes
Node [1]
  Identifier : jz75@pc7-007-1.cs.st-andrews.ac.uk
  Port      : 4105
  Services  : search=true,download=true
Node [2]
  Identifier : jz75@pc7-033-1.cs.st-andrews.ac.uk
  Port      : 4105
  Services  : search=true,download=false
```

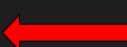


Figure 3: In the same pc7-005-1 node as before, we can see that after executing the :nodes command, the listed advertisement information for the pc7-033-1 node now indicates that downloading is not possible. This shows that the advertisement that the current pc7-005-1 node had stored for the pc7-033-1 from before (in Test R1.1) has been updated.

3.2.3 Test R1.3

Test Purpose : to test that expired messages are removed from the node’s advertisement queue.

```
Node [1]
  Identifier : jz75@pc7-007-1.cs.st-andrews.ac.uk
  Port      : 4105
  Services  : search=true,download=false
```

Figure 4: Output from running the `:nodes` command on node `pc7-005-1` some time after node `pc7-033-1` leaves the multicast group session. We can see that the advertisement information from `pc7-033-1` no longer appears in the command line output.

4 Requirement 2 : Search

4.1 Design

For processing incoming search request messages, I have implemented a queue-based approach. Incoming search requests are added to a specific queue, where a worker thread runs to continuously pop messages out of the queue and process them. The processing of each message involves performing a file search for the specified search-string, then transmitting an appropriate search-response message back to the sender of the search-request. A queue-based approach is also used to handle incoming search-response messages. These messages are again added to a specific queue where worker thread pops out search-response messages one at a time and prints out its contents to the user. I am using queue-based approaches in these two instances because search-requests and search-responses can arrive in bursts and can take time to process (such as performing a directory walk to match a search-string).

The process of creating and sending out a search-request message is a rather lightweight process that can only occur once at a time (cannot send out multiple search requests concurrently). Consequently, I have decided not to use an outbound transmission queue with separate worker threads.

4.2 Operation

In the interface, the user can enter the `:search` command, at which point they will be prompted to enter a search-string which can be queried against each node's file system within the multicast group. The search-string will always be substring-matched against each node's file system and can be matched against both file and directories.

4.3 Testing

NOTE : for these tests, I have omitted the log outputs since they are too long to be easily readable within the report document. Also all tests were performed with 3 different nodes (lab clients) connected to a multicast group : `pc7-033-1`, `pc7-007-1`, and `pc7-005-1`. Outputs are from `pc7-033-1`.

4.3.1 Test R2.1

Test Purpose : To demonstrate basic search functionality with a search-string that is meant to substring match against the name of a file or directory (not path).

```
[filename | '.' | '..' | ':services' | ':nodes' | ':search' | ':download' | ':quit' | ':help'] :search
Please enter your search string : text1
Searching multicast group for : text1...

[Search Result] : root_dir/dir1/text1-1.txt @ jz75@pc7-007-l.cs.st-andrews.ac.uk
[Search Result] : root_dir/dir1/text1-2.txt @ jz75@pc7-007-l.cs.st-andrews.ac.uk
[Search Result] : root_dir/dir1/text1-3.txt @ jz75@pc7-007-l.cs.st-andrews.ac.uk
[Search Result] : root_dir/dir1/text1-1.txt @ jz75@pc7-005-l.cs.st-andrews.ac.uk
[Search Result] : root_dir/dir1/text1-2.txt @ jz75@pc7-005-l.cs.st-andrews.ac.uk
[Search Result] : root_dir/dir1/text1-3.txt @ jz75@pc7-005-l.cs.st-andrews.ac.uk
```

Figure 5: Output from using the `:search` command, specifying a “text1” as the search-string to query against all nodes in the multicast group. We can see that we get search-result messages from the 2 other nodes in the multicast group, where for each node, the search-string correctly substring-matches to three different resources (text files in this case).

4.3.2 Test R2.2

Test Purpose : To demonstrate that search functionality works for matching against file paths as well (in addition to names).

```
Please enter your search string : dir1/text1-1.txt
Searching multicast group for : dir1/text1-1.txt...

[Search Result] : root_dir/dir1/text1-1.txt @ jz75@pc7-007-l.cs.st-andrews.ac.uk
[Search Result] : root_dir/dir1/text1-1.txt @ jz75@pc7-005-l.cs.st-andrews.ac.uk
```

Figure 6: Output from using the `:search` command, specifying a “dir/text1-1.txt” as the search-string to query against all nodes in the multicast group. We can see that we get search-result messages from the 2 other nodes in the multicast group, where for each node, the search-string correctly substring-matches to one resource.

4.3.3 Test R2.3

Test Purpose : To demonstrate that if a search-string doesn’t match any resources, a node will respond to a search-request with a search-error message.

```
Please enter your search string : SomeFileThatDoesNotExist
Searching multicast group for : SomeFileThatDoesNotExist...

[Search Error] : No Result @ jz75@pc7-007-l.cs.st-andrews.ac.uk
[Search Error] : No Result @ jz75@pc7-005-l.cs.st-andrews.ac.uk
```

Figure 7: Output from using the `:search` command, specifying “SomeFileThatDoesNotExist” as the search-string to query against all nodes in the multicast group. This search-string is already known to not exist at any of the multicast nodes. We can see that we get search-error responses from the two other nodes in the multicast group specifying that no results were found.

4.3.4 Test R2.4

Test Purpose : To demonstrate performing a search when there are no other nodes connected to multicast group.

```
[filename | '.' | '..' | ':services' | ':nodes' | ':search' | ':download' | ':quit' | ':help'] :search  
[SEARCH ERROR] : No other nodes in multicast group right now.  
[filename | '.' | '..' | ':services' | ':nodes' | ':search' | ':download' | ':quit' | ':help'] █
```

Figure 8: In this test, only pc7-033-1 node is in the multicast group. We can see when the user tries to use the `:search` command, they are given an error message indicating that no other nodes are in the multicast group.

4.3.5 Test R2.5

Test Purpose : To demonstrate that if a node sends out a search request at the same time as another node, it will only receive and display search results corresponding to its own search-request.

Output from pc7-033-1 :

```
Please enter your search string : text2  
Searching multicast group for : text2...  
[Search Result] : root_dir/dir2/text2-1.txt @ jz75@pc7-005-l.cs.st-andrews.ac.uk  
[Search Result] : root_dir/dir2/text2-2.txt @ jz75@pc7-005-l.cs.st-andrews.ac.uk  
[Search Result] : root_dir/dir2/text2-1.txt @ jz75@pc7-007-l.cs.st-andrews.ac.uk  
[Search Result] : root_dir/dir2/text2-2.txt @ jz75@pc7-007-l.cs.st-andrews.ac.uk
```

Output from pc7-007-1 :

```
Please enter your search string : text1  
Searching multicast group for : text1...  
[Search Result] : root_dir/dir1/text1-1.txt @ jz75@pc7-005-l.cs.st-andrews.ac.uk  
[Search Result] : root_dir/dir1/text1-2.txt @ jz75@pc7-005-l.cs.st-andrews.ac.uk  
[Search Result] : root_dir/dir1/text1-3.txt @ jz75@pc7-005-l.cs.st-andrews.ac.uk  
[Search Result] : root_dir/dir1/text1-1.txt @ jz75@pc7-033-l.cs.st-andrews.ac.uk  
[Search Result] : root_dir/dir1/text1-2.txt @ jz75@pc7-033-l.cs.st-andrews.ac.uk  
[Search Result] : root_dir/dir1/text1-3.txt @ jz75@pc7-033-l.cs.st-andrews.ac.uk
```

Figure 9: Two nodes, pc7-033-1 and pc7-007-1, send out a search request at the same time for search strings, “text2” and “text1”, respectively. We can see in the top image, that pc7-033-1 only receives search results for its own search-request with the search-string, “text2”. In the bottom image we can see that pc7-007-1 only receives search results for its own search-request for the search-string, “text1”.

4.3.6 Test R2.6

Test Purpose : to test that a node that does not have search capability will not provide a search response for a search request.

```

[filename | '.' | '..' | ':services' | ':nodes' | ':search' | ':download' | ':quit' | ':help'] :nodes
Node [1]
  Identifier : jz75@pc7-005-1.cs.st-andrews.ac.uk
  Port      : 4105
  Services  : search=true,download=true
Node [2]
  Identifier : jz75@pc7-007-1.cs.st-andrews.ac.uk
  Port      : 4105
  Services  : search=false,download=true

[filename | '.' | '..' | ':services' | ':nodes' | ':search' | ':download' | ':quit' | ':help'] :search

Please enter your search string : text1-1.txt
Searching multicast group for : text1-1.txt...

[Search Result] : root_dir/dir1/text1-1.txt @ jz75@pc7-005-1.cs.st-andrews.ac.uk

```

Figure 10: The user first uses the `:nodes` command which indicates that the node with the identifier “jz75@pc7-007-1.cs.st-andrews.ac.uk” does not have search capability. The user then uses the `:search` command to search for the search-string “text1-1.txt” and only gets a response from “jz75@pc7-005-1.cs.st-andrews.ac.uk” and not from the “jz75@pc7-007-1.cs.st-andrews.ac.uk” node which does not have search capability.

5 Requirement 3 : Download

5.1 Protocol Specification

The exact specification can be found in the file : `code/protocol_specification.txt`

5.1.1 Protocol Description

1. A requester node can request to download a file from a target node by sending out a `<download-request>` message specifying the identifier of the target node, along with a file-string which should be a logical filename that uniquely matches a file in the target node’s file system.
2. When a target node receives a `<download-request>` message from the multicast channel, it checks that the target identifier in the message matches its own identifier.
3. If the target node’s identifier matches the target identifier in the message, it will determine all files that match the specified file-string.
4. If the target node determines that no files match the file-string, it will respond to the requesting node with a `<download-error>` message indicating that no matches were found.
5. If the target node determines that more than one file matches the file-string, it will also respond with a `<download-error>` error message indicating that multiple matches were found.
6. If the target node determines that the specified file-string matches exactly one file in its file system, it will create an ephemeral TCP socket for unicast file transfer. After doing so, it will respond with a `<download-result>` message indicating that unique match was found, along with addressing information for the ephemeral file transfer socket.

7. Once the requesting node receives a `<download-result>`, it will use the specified addressing information to connect to the target node's ephemeral file-transfer socket to initiate the file transfer.
8. File is downloaded to a directory specified in the configuration file
(`code/downloads/<target-identifier>/`)

5.2 Design

Similar to search-requests, I have decided to use a queue-based approach for handling incoming download-requests. Incoming requests are added to a message-specific queue where a dedicated worker thread pops messages from the queue for processing. For each message, the worker thread searches for the specified file-string in its file system and determines what kind of download-response message to respond to the requester with. If the file-string correctly identifies the full logical path-name of exactly one file, the target node will create an ephemeral TCP socket (for file-transfer) and a new thread to keep the socket listening for incoming connections, and then send a download-result message to the requester containing the addressing information for connecting to the ephemeral TCP socket. This new thread continues to run until it either times out, or receives a client connection, at which point it will transfer the file.

On the requester (client) side, incoming download responses (to previously sent download requests) are also added to a message-specific queue and processed by a dedicated worker thread. If the message is a download-error, it will indicate so to the user through the interface. In my protocol, download-error messages indicate the number of files in the target node's file system that were matched to the file-string specified in the corresponding download-request. This is to allow download errors on the requester side to be more informative, indicating whether the error was due to there being no matches or more than once match with the file-string. If the worker thread determines that the message is a download-result, it creates an ephemeral TCP client to connect to the socket whose addressing information was specified in the download-result message. A new thread is created to keep this client running until it either successfully completes the file transfer, or until a timeout period is reached.

The act of sending out download-request messages is rather lightweight and sparse (compared to receiving them), and therefore are not queued and processed by threads but rather sent out immediately in response to the user using the `:download` service.

5.3 Operation

The user can initiate a download request by using the `“:download”` command. The user will then be prompted for a file-string that identifies a remote file to download, and then prompted to specify the target node's identifier from whom the user wants to remotely download the file from. The user will then get a notification indicating whether they received a download-error or download-result from the target node. In the case of a download-error, a message will be outputted indicating the nature of the error. In the case of a download-result, a message will be outputted indicating so and that file transfer has been initiated. Once the file transfer is complete, a message will be displayed to the user indicating the file that was downloaded.

If the target node in the download request does not exist in the advertisements received so far, or has advertised that they do not have the download capability, a message will be outputted to the user indicating so, and the download request will not be transmitted.

5.4 Testing

5.4.1 Test R3.1

Test Purpose : To test the basic functionality of downloading a valid file from another node in the multicast group.

```
[filename | '.' | '..' | ':services' | ':nodes' | ':search' | ':download' | ':quit' | ':help'] :download
Please enter your file string : root_dir/dir1/text1-1.txt
Please enter target identifier (username@hostname) : jz75@pc7-007-l.cs.st-andrews.ac.uk
Sending out download request for root_dir/dir1/text1-1.txt to jz75@pc7-007-l.cs.st-andrews.ac.uk...

[DOWNLOAD RESULT] found file @ jz75@pc7-007-l.cs.st-andrews.ac.uk
Initiating file transfer...
Downloaded file [text1-1.txt] from pc7-007-l.cs.st-andrews.ac.uk
```

Figure 11: Here two lab clients, pc7-033-1 and pc7-007-1, are connected to a multicast group. The user enters the :download command and specifies and file-string of “root_dir/dir1/text1-1.txt” and a target node identifier of “jz75@pc7-007-1.cs.st-andrews.ac.uk”. The interface first outputs what type of download-response it got from the target node (in this case a download-result), and then indicates that it has initiated the file transfer. When the file has been downloaded, a message is displayed to the user indicating so.

5.4.2 Test R3.2

Test Purpose : to test the error handling functionality when the file-string specified in a download request has no matches at the target node.

```
Please enter your file string : SomeFileThatDoesNotExist
Please enter target identifier (username@hostname) : jz75@pc7-007-l.cs.st-andrews.ac.uk
Sending out download request for SomeFileThatDoesNotExist to jz75@pc7-007-l.cs.st-andrews.ac.uk...

[DOWNLOAD ERROR] : No Results @ jz75@pc7-007-l.cs.st-andrews.ac.uk
4 HINT : Use the :search command to first determine the target node's files.
```

Figure 12: Here two lab clients, pc7-033-1 and pc7-007-1, are connected to a multicast group. On pc7-033-1, the user enters a file-string that does not exist in the target node, “SomeFileThatDoesNotExist”, and after sending out the download-request, the interface indicates to the user that a download-error was received, specifically mentioning that no results were found at the target node.

5.4.3 Test R3.3

Test Purpose : to test the error handling functionality when the file-string specified in a download request is ambiguous and matches multiple files at the target node.

```

Please enter your file string : text
Please enter target identifier (username@hostname) : jz75@pc7-007-l.cs.st-andrews.ac.uk
Sending out download request for text to jz75@pc7-007-l.cs.st-andrews.ac.uk...

[DOWNLOAD ERROR] : Non-unique file-string -> 9 matching files found @ jz75@pc7-007-l.cs.st-andrews.ac.uk
↳ HINT : Use the :search command to first determine the target node's files.

```

Figure 13: Here two lab clients, `pc7-033-1` and `pc7-007-1`, are connected to a multicast group. On `pc7-033-1`, the user enters an ambiguous file-string, “`text`”, that matches multiple files at the target node where after sending out the download-request, the interface indicates to the user that a download-error was received, specifically mentioning that 9 results were found at the target node.

5.4.4 Test R3.4

Test Purpose : Test that sending a download request (using `:download`) to a target node that is not currently connected to multicast channel is not transmitted, where a corresponding error message is then presented to the user through the interface.

```

Please enter your file string : root_dir/dirl/text1-1.txt
Please enter target identifier (username@hostname) : jz75@someHostname.ac.uk
Sending out download request for root_dir/dirl/text1-1.txt to jz75@someHostname.ac.uk...

[DOWNLOAD ERROR] : target jz75@someHostname.ac.uk does not exist in multicast group.
↳ HINT : use the :nodes command before to check what nodes are part of multicast group.

```

Figure 14: Here the user uses the `:download` command and specifies a target identifier that does not exist in the multicast group, and gets a download error message indicating so.

An additional check was done to ensure that the download request was not transmitted in the logs.

5.4.5 Test R3.5

Test Purpose : Test that sending a download request to a node that does not have the download capability is caught by the application and prevented from transmitting.

```

Please enter your file string : root_dir/dirl/text1-1.txt
Please enter target identifier (username@hostname) : jz75@pc7-007-l.cs.st-andrews.ac.uk
Sending out download request for root_dir/dirl/text1-1.txt to jz75@pc7-007-l.cs.st-andrews.ac.uk...

[DOWNLOAD ERROR] : jz75@pc7-007-l.cs.st-andrews.ac.uk does not have download capability.
↳ HINT : use the :nodes command before to check capabilities.

```

Figure 15: Here two lab clients, `pc7-033-1` and `pc7-007-1`, are connected to a multicast group. On `pc7-033-1`, the user specifies a target identifier of `jz75@pc7-007-l.cs.st-andrews.ac.uk` which does not have the download capability. The user gets back a download error message indicating this is the case.

An additional check was done to ensure that the download request was not sent out in the logs.

5.4.6 Test R3.6

Test Purpose : test that sending a download request specifying a remote directory (rather than a file) leads to a download error.

```
Please enter your file string : root_dir/dir1/text1-1.txt
Please enter target identifier (username@hostname) : jz75@pc7-007-1.cs.st-andrews.ac.uk
Sending out download request for root_dir/dir1/text1-1.txt to jz75@pc7-007-1.cs.st-andrews.ac.uk...

[DOWNLOAD ERROR] : jz75@pc7-007-1.cs.st-andrews.ac.uk does not have download capability.
↳ HINT : use the :nodes command before to check capabilities.
```

Figure 16: Here two lab clients, pc7-033-1 and pc7-007-1, are connected to a multicast group. On pc7-033-1, the user specifies a file-string that is a directory, “root_dir/dir1”, and “jz75@pc7-007-1.cs.st-andrews.ac.uk” as the target node. The user gets a download error which correctly indicates that no results were found at the target node.

5.4.7 Test R3.7

Test Purpose : test that sending a download request of a file-string that does not specify the full logical path-name of the file leads to a download error.

```
Please enter your file string : text1-1.txt
Please enter target identifier (username@hostname) : jz75@pc7-007-1.cs.st-andrews.ac.uk
Sending out download request for text1-1.txt to jz75@pc7-007-1.cs.st-andrews.ac.uk...

[DOWNLOAD ERROR] : No Results @ jz75@pc7-007-1.cs.st-andrews.ac.uk
↳ HINT : Use the :search command to first determine the target node's files.
```

Figure 17: Here two lab clients, pc7-033-1 and pc7-007-1, are connected to a multicast group. On pc7-033-1, the user specifies a file-string that is a filename rather than the full logical path-name of the file in the target node’s filesystem, “text1-1.txt”. The user gets an download error which correctly indicates that no results were found at the target node.

5.4.8 Test R3.8

Test Purpose : Test that target node will only transfer a file to the client that sent the corresponding download-request, and not any client that connects to its ephemeral file-transfer TCP socket.

There are no figures to demonstrate this test, but the functionality is located in the main methods of the `FileServer.java` and `FileClient.java` files. The test involves using the `FileServer.java` class to start an ephemeral file-transfer socket on the pc7-033-1 lab client with an expected client hostname of pc7-007-1.cs.st-andrews.ac.uk, and then using the `FileClient.java` class on pc7-005-1 to connect to the ephemeral file-transfer socket and attempt to initiate the download. If the requested file does not get downloaded onto the pc7-005-1 node, then the test passes. Ultimately this test passed as the target node was correctly able to identify that the client attempting to download the file (pc7-005-1) was not the client that it expected to download the file (pc7-007-1).

6 Requirement 4 : Critical Analysis

6.1 Scalability

By using a combination of message queues for incoming messages and worker threads to process messages out of those queues, my application effectively decouples its ingestion (receipt) of messages from its processing of messages (such as searching through its file system and generating responses). This helps address the possibility of receiving bursts of incoming messages such as search-requests and download-requests. By using multiple queues with separate worker threads for processing each queue, a degree of load distribution is implemented. A separate thread is also used to send out periodic advertisements at a constant interval. As mentioned before, a central thread coordinator/dispatcher is used to manage all threads and to delegate incoming messages to their designated queue to be processed by its corresponding worker thread. All worker threads are managed as a thread pool by the dispatcher in order to prevent thread explosion.

For the download functionality, when a node receives a valid download-request, it creates an ephemeral TCP socket (`FileServer.java`) for unicast file transfer, along with a separate thread to keep it listening for a client connection. This potentially could lead to a thread explosion as the thread created to keep the socket listening is not managed by a fixed thread pool, thus a new thread and socket are created for each valid download-request. A timeout is implemented for the socket to prevent threads from accumulating if there is an issue that prevents a client from connecting, but a queuing system with worker threads could be a more scalable solution. For example, a thread pool of n threads could be created so that at any time, n file transfers can take place concurrently, where any additional file-transfer tasks would wait in a queue for a thread to become available. A UDP based file-transfer approach could also improve scalability without needing to create new threads for each connection.

I have chosen to use ephemeral sockets to handle file transferring since these unicast connections are relatively short-lived. Only one file can be requested per download-request, therefore the unicast connections for file transfer only have to persist for the duration of a single file transfer within a local area network. An alternative approach could be to create a single persistent socket that all clients could connect to. Bursts of incoming client connections could similarly be queued or processed by separate threads. This socket would be separate from the multicast socket and therefore run on a separate thread to listen continuously. If the application were to receive download requests at a constant high frequency, this approach could be viable. However, if download-requests are sparse, this approach would lead to wastage of resources in keep this TCP socket running even when it's not being used.

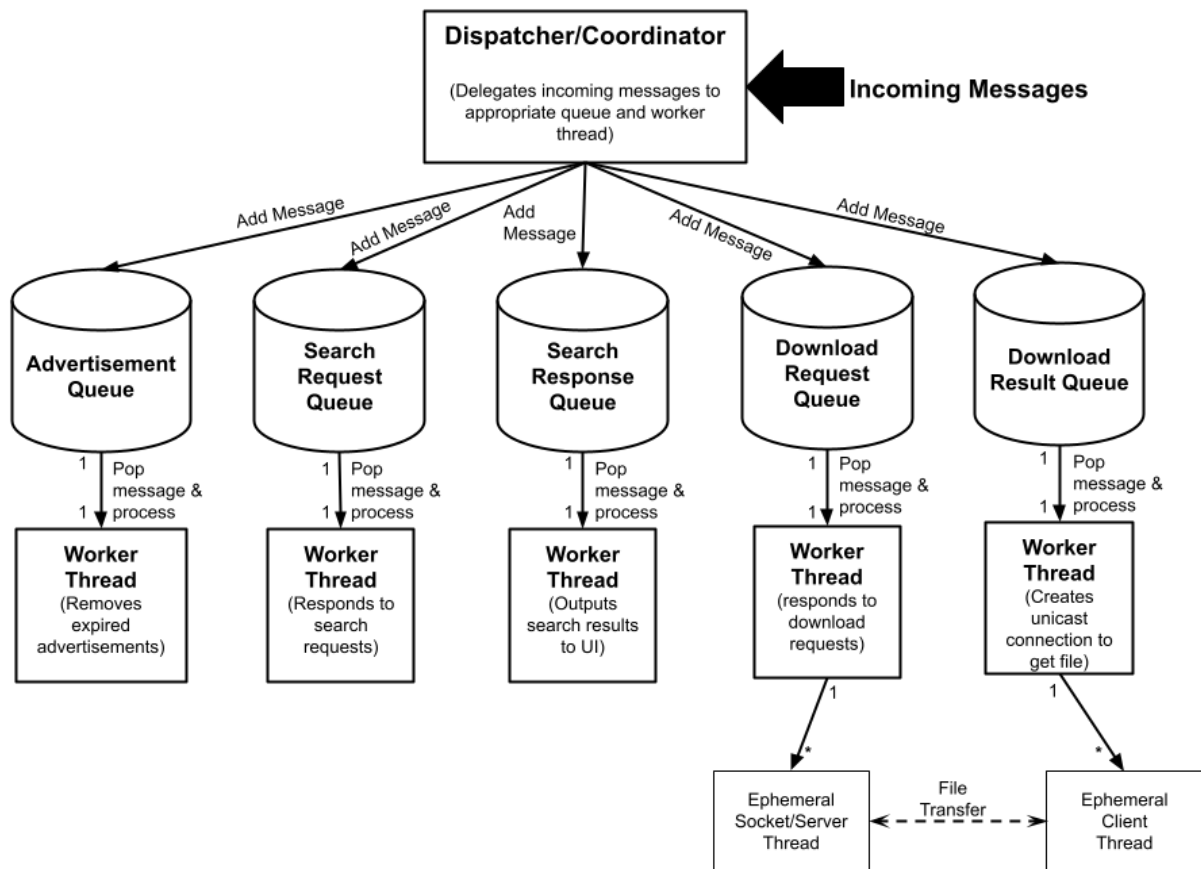


Figure 18: A simplified diagram of how multi-threading is used to handle incoming messages. Note that all boxes labeled “Worker Thread” are managed by a thread pool. This diagram does not include the functionality for sending out messages.

Messages in each queue are processed by a single thread each. Depending on the system’s resources, each queue could be processed by more than one worker thread, such as using a thread pool for greater multiprocessing. The central coordinator/dispatcher could potentially also benefit from a queued multi-threaded approach. There is some overhead in inspecting each received message, determining what type of message it is, and delegating the message to the appropriate message-specific queue. If the influx of messages is high enough, the messages could be added directly to a central queue without inspection, where worker threads could process messages out of that central queue, inspect them, and then delegate them their appropriate message-specific queue (e.g Search Request queue, Advertisement queue, etc).

Similar to multicast DNS[3, 1], I am also using a form of pseudo-random delays for configuring transmission timing. For example, once a node joins a multicast group, it begins sending out advertisements at a fixed interval. If, theoretically, every node were to join at the same time, their sending of advertisements could synchronize, leading to bursts of advertisement traffic. To circumvent this, the worker thread delegated to the task of sending out advertisements is configured to use a pseudo-random delay before it begins executing.

6.2 Performance

As mentioned in the previous section, multi-threading is used to keep the application's functionality scalable. The use of multi-threading in turn should offer some performance benefits for the user. The interface of the application is asynchronous with respect to many of the backend operations. For example, when a user sends a search request using the `:search` command, there is no built-in functionality that forces the interface to wait until all search responses come back from all nodes in the multicast group before displaying the search responses to the user. Instead, a timeout is implemented in the interface to provide some time for all incoming search responses to be processed and displayed to the user before prompting the user for their next command. As a result, the multi-threaded nature of the application is very important to enable timely results in response to user actions in the interface. Even if there were a mechanism to force the interface to wait for backend operations, minimizing delay is still optimal. It would be detrimental in a single-threaded scenario, where the operation of processing incoming search responses must wait for another operation, such as removing all expired advertisements.

There are multiple areas where further performance improvements can be implemented to reduce the amount of traffic over the multicast channel. There are known mechanisms in multicast DNS that help reduce traffic which could be applicable for this scenario such as Known-Answers Suppression[1, 3] and Duplicate Questions/Answers Suppression[1, 3]. More generally, caching can be used to reduce certain types of messages. Currently, each node is configured to ignore messages, such as search-result messages, if they are not addressed to the node. Instead, the node could cache the search results addressed to other nodes, where these cached results could be used for future search requests - essentially implementing a form of Duplicate Question Suppression. Extra consideration would be needed such as determining aspects such as cache record TTL values which could depend on factors such as how often we expect file systems to change. Caching schemes can not only enhance scalability by reducing the number of messages sent, but also offer performance benefits for the user by reducing delays in retrieving relevant results for requests.

The application does make other efforts to reduce transmissions. For example, when sending out a download-request for a particular target node, the application first checks its stored advertisements to see if the node exists in the multicast group and has advertised that it has the download capability before transmitting the request.

6.3 Privacy & Security

When discussing privacy and security, it is important to consider the context of the application. As a local area network application, the task of identifying the user base and defining trust can be more straightforward due to the more limited and controlled environment. If we can already trust our user base, it may be unnecessary to implement privacy and security measures within the protocol but rather rely on pre-existing infrastructure such as the university's firewalls to block out-of-network bad actors. My implementation assumes a trusted LAN and includes minimal security measures. Multicast messages and their contents can be viewed by all nodes in the network and nodes are assumed to not modify the application to behave in an unexpected way. The only area where some consideration of user identification is made is for the download functionality. As mentioned before, before transferring a requested file to a client, the file transfer server (ephemeral socket) first determines that the client that has connected to it is the same client that

sent the original download-request for the given file to be transferred.

If a more defensive programming approach were adopted, a lot of policy needs to be defined. If it is the case that not everyone in the local area network can be trusted, then a method for identifying and authenticating nodes in the multicast group needs to be implemented. Then protected assets (e.g. messages) would require an encryption scheme to prevent the wrong LAN nodes from viewing the content of those assets. There are many different encryption schemes that can be used, but in general, any approach utilizing encryption can add overhead as encryption is a computationally expensive task.

There are a number of ways a malicious actor within the LAN could disrupt the user experience of other nodes. For example, any node could be modified to send unprompted search/download response messages to nodes that never sent a request to begin with. A malicious actor could theoretically spam a node in the multicast group with search-result messages as a DOS attack, causing search-result messages to flood the user's interface. This could also be done to implement a form of buffer-overflow attack to fill the target node's search-response queue. Prevention techniques could involve enforcing that incoming search-response must correspond to a previously transmitted search-request, where search-response message that do not have a match are ignored. This would involve creating an outbound data structure to store all previously transmitted requests, possibly with TTL values.

Buffer overrun attacks are also possible for request messages. A node could be modified to send a high frequency of search/download requests to a target node to fill its buffers and waste resources. A possible solution could be to set a maximum size for the receive buffers, where additional messages are ignored. A more fine-grained solution could be to implement a detection method to determine the frequency at which a node is sending requests, and hence block the node if its frequency is too high.

As mentioned in the scalability discussion, the file transfer implementation is susceptible to thread explosion since a new thread is created for each ephemeral TCP socket on the server's side. This could be exploited by a malicious actor by spamming large quantities of valid download requests, causing the target node to create a large amount of threads in a short burst to handle those requests, thus leading to resource exhaustion. As mentioned, a queue-based thread pool approach could alleviate scalability concerns, but also to address this security risk.

References

- [1] Saleem Bhatti. *CS4105 Discovery Protocols*. Oct. 2024. URL: https://studres.cs.st-andrews.ac.uk/CS4105/Lectures/wk05/cs4105-discovery_protocols.pdf.
- [2] *Database Net Services Administrator's Guide*. URL: https://docs.oracle.com/cd/E11882_01/network.112/e41945/dispatcher.htm.
- [3] M. Korchmal S. Cheshire. *Multicast DNS*. rfc. URL: <https://www.rfc-editor.org/rfc/rfc6762.html>.