README

Distributed Systems Assignment 2

Design / Workflow

Aggregation Server - Server starts and takes a port input if desired - Server socket is opened and to accept a connection - Upon reiceved connection content is evaluated to find request type - A new thread is started based on the request type - **If the request is a PUT** - Update lamport timestamp upon reciept - Attempt to parse the XML data using DOM. - If successful take unparsed data and store in a linked list of size 20. If the list is full delete the first element stored. If the XML data came from an already known id then replace the associated feed. - Send a success response along with lamport timestamp. - Update the storage location after a PUT request. - Halt the thread for 12 seconds - Resume and check for a Content Server heartbeat - If no heartbeat is found delete associated XML data and update storage again. - **If the request is a GET** - Update lamport timestamp upon reciept - Create a packet object containing the list of stored feeds and a lamport timestamp - Send packet object to the GET Client - **If the request is bad** - Send a 400 error back through the socket - Do not parse or store any data sent

Content Server - Content Server starts and accepts 3 command line inputs in order: - Chosen ID (can be any string) - Input File location - Network IP address and port number - XML data is parsed. If the data is unable to be passed it will send nothing. If it can be successfully parsed it will be parsed into a string and that string will be sent along with a lamport timestamp in a packet object - Checks the socket input stream for a server response. If the server is not found it will retry to send the request the same address/port number 4 times before halting. - If the response arrives display the response on the command line - If the response is successful maintain connection and send a heartbeat to the server until manual exit. **GETClient** - Get Client starts and accepts 1 command line input: a Network IP address and port number - client makes a GET request to the chosen network - If successful the client takes the recieved packet and parses the XML string using DOM. This output is then printed to the console in a readable format, detailing each entry and feed along with which Content Server it came from.

Compiling

Automated tests rely on the Aggregation Server to be already running unless you wish to test the retry feature of Content Server PUT requests. Compiling process is as follows: - Compile the xml package: javac -d . XMLCreator.java

XMLPrinter.java Packet.java GETPacket.java - Compile the main java files: - javac AggregationServer.java - javac ContentServer.java - javac GETClient.java

Manual Operation and Testing

After compiling each of the 3 main components can be run manually using the java command.

When running the aggregation server: The server will accept a custom port or a default port. The default is 4567. In order for automated testing to function the port will need to remain default or you will need to update the bash scripts in the tests folder.

When running the content server: The server will take 3 inputs. The first is a desired ID. This can be any string. The second is a desired inputfile. This can be any input file so long as its located in the same directory or it defined directory path. The third is the network id and port number written as either https://address:domain or address:domain.

When running the get client: The client will take a single input which is the network address and domain. This operates exactly the same as the 3rd content server input. The client will automatically run the XMLPrinter and will display readable formatted data on the console.

Automated Testing

In the tests folder there are several automated tests. These test several different components including a concurrency test which generates multiple PUT requests at the same time.

Regarding the concurrency test, this uses a linux package called GNU parallel so it will not work without first installing this package on whatever machine is being used to test. All other tests use basic bash.

The test outputs are placed in a folder called **outputs**.