COMP-3670 Assignment 2 Part 2

Group 24: Dariq Ahmed, Curtis DeSlippe, Leanna Lariviere, Yuvi Nanuan

Oct 30, 2020

**Part 1 - DESIGN AND PROGRAMMING ASSIGNMENT**

1. **Design a protocol for this network application and justify your design decision. In your protocol design, you should address: 1) The communication patterns of the network application; 2) The protocol design goals; 3) Its message format, structure and semantics; and 4) Its communication rules.**

Before we get started on designing the protocol, we need to first establish the communication patterns of the network application—how the network app works using TCP connections.

**Q1.1 - Network Application Communication Patterns Analysis**

Let’s call the network app “JobAssigner”. JobAssigner consists of a number of nodes, some of which are job-seeker nodes (“JobSeeker”) while others are job-creator nodes (“JobCreator”). Connections are initiated by JobSeekers, who can each connect to any of the JobCreators, but only one at a time, and never to each other. Each JobCreator can support connections to multiple JobSeekers at once.

In this app, the job-creator nodes function as servers while the job-seeker nodes function as clients. Analyzing the communication patterns that happen at a single job-seeker node:

1. The JobSeeker signals its availability to the JobCreator it connects to. JobCreator notes down this availability, establishing the TCP connection, and returns a job type as a response if any job types are available.
   * 1. The availability value of JobSeeker would likely be a string or boolean value.
     2. The job type would also likely contain a string value (text).
     3. The JobCreator has a list of job types, and a list of job sizes associated with each job. Each job type assigned decrements the corresponding job size.
     4. Some job sizes allow the same job type to be assigned to multiple JobSeekers.
     5. A JobSeeker can accept multiple job types, but only respond to one at a time. Other job types are queued.
     6. Some job types automatically close the connection, while others maintain it. That means the connection must be able to be terminated or maintained depending on the situation, so past client request info is either stored or discarded depending on the job type.
2. After the job is complete, JobSeeker returns both status of the job type and any result back to the JobCreator.
   * 1. As the JobSeeker now provides service (like computation, for example), the JobSeeker and JobCreator now switches roles as client and server. That means both nodes on the connection need to be able to function as both at different times.
     2. The status returned would either be a boolean, an int or float value, or a string value. Any job result returned would likely be the same.
     3. Based on the patterns, the network app has an interactive multicast communication model.
3. Either JobSeeker or JobCreator can close the connection at any time.
   * 1. Again, the connection can be terminated by either node. That means that the connection must be able to hold a state, so that the connection can be maintained if not terminated by the JobSeeker or JobCreator.

**Q1.2 - Protocol Design Goals**

For simplicity, we will refer to our protocol as JCSCP (Job Creator-Seeker Communication Protocol).

The protocol design goals for JCSCP can be determined by analyzing the communication patterns. Based on the analysis in Question 1.1, we can determine that the protocol has the following goals:

1. Allow the reliable transfer of messages that contain number data types such as booleans, ints, floats, as well as text data types such as string values.
2. Allow both persistent and non-persistent connections depending on job type transferred, so that job types can dictate whether the connection is automatically closed after transfer.
3. There is no need for bandwidth restriction or communication security. The protocol messages can afford some delay.
4. Standard TCP error checking is sufficient.
5. Allow the job-seeker and job-creator nodes to exchange roles as clients and servers depending on whether the job-seeker currently has a job type assigned to it. That means the protocol must allow P2P architecture.
6. As a text protocol, JCSCP has a message format that consists of ASCII text characters.
7. Protocol is text protocol, which means it is extendable.

**Q1.3 - Message Format, Structure and Semantics**

Our protocol (JCSCP) is a text protocol. That means that its messages are composed of ASCII characters. An example of the message design for a JCSCP request from a JobSeeker to a JobCreator is given below:

// Message design in text format

Job Type: Addition

Var 1: 3

Var 2: 5

Bad Job: false

Job Done: false

Sum: 8

// Message design implementation in Java

struct JCSCP\_Request {

String jobString; // carries assigned job type

int jobVar1; // variable used to complete job

int jobVar2; // variable used to complete job

boolean badJob; // used for error checking

boolean jobDone; // used for checking job status

double sum; // carries result of completed job

};

The message semantic is command and control. Messages do exchange some data, but they also control the states of the job-seeker and job-creator nodes using the jobType attribute.

Command messages are sent from the JobSeeker to the JobCreator, announcing availability and authenticating the connection. The JobCreator sends command messages back, specifying a jobType that either signals for persistent or non-persistent communication. Control messages allow either node to terminate the connection at any time, while more command messages return both the job status (jobDone) and the job result (sum) back to the JobCreator.

The protocol has the following structure:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| “Job Type: ” | sp | jobString | cr | lf | | |
| “Var 1: ” | sp | jobVar1 | cr | lf | | |
| “Var 2: ” | sp | jobVar2 | cr | lf | | |
| “Bad Job: ” | sp | badJob | cr | lf | | |
| “Job Done: ” | sp | jobDone | cr | lf | | |
| “Sum: ” | sp | sum | cr | lf | | |
| cr | | | lf | | | |
|  | | | | | | |

1. The first six lines are header lines that contain the header field names and their values. Each line has a fixed length of 32 bytes.
2. After a newline, there is a body that contains any additional data such as messages to be printed and displayed when the job is successful or unsuccessful. This line is variable in size, from 20 to 60 bytes.

After analyzing the communication patterns and designing the protocol, we can glean the following communication rules:

1. The job-seeker node (“JobSeeker”) identifies itself to the job-creator node (“JobCreator”) by sending a command message to initiate a connection. The JobSeeker is currently a client, and the connection it establishes is in a non-persistent state.
2. The JobCreator acknowledges by sending back a command message. If a jobType is available from the BufferedReader value bf, the JobCreator assigns the specified job to JobSeeker (JobCreator is the server). Depending on the jobType, a control message may be sent to the JobSeeker that terminates the connection once the jobType is received. This determines whether the connection remains non-persistent or becomes persistent.
3. The JobSeeker completes the job. If the connection is persistent, it sends back another command message to the JobCreator. This message carries updated values for the jobDone, badJob, and sum parameters.
4. The JobCreator receives the message and acknowledges it. JobCreator then either sends another jobType in a command message, or terminates the connection with a control message. If another jobType is sent, the jobType may signal a control message to either terminate or maintain the connection once it is received.
5. **Argue the need for a new application layer protocol for this network application instead of using existing standard protocols (e.g. HTTP, SMTP, WebSocket, etc.)**

Due to the functionality of our network application, using a standard network protocol would not meet the needs of the application. Since our network application design specifies that we would need to be able to send data types like strings, booleans, and integers/doubles, we would need to use a network protocol that supports this.

The protocol would also need to support the type of connections that need to be formed in order for the nodes to communicate. Our application also needs to communicate over TCP/IP networks, so standard protocols that run over the UDP like TFTP would not suffice. Existing standard protocols like SMTP, TFP, NFS, and TELNET simply deal with different types of communications (i.e. emails, files, or server commands over virtual terminals).

Our design requires a two-way connection to be formed; JobSeekers will send data to the JobCreator (job status and results), while Jobcreators will send data to the JobSeekers (job description). A standard protocol like HTTP would not work, as it is a unidirectional service where the client sends a request for web elements from a server.

HTTP would not allow the JobSeekers to send the job status/results to the JobCreators. Our application needs to be able to form connections that can either remain open or be closed upon assignment of a job. Since HTTP is stateless, the connection would be closed after the job type is delivered. The WebSocket protocol is stateful, meaning that the connection would be maintained until closed by either node, as WebSocket saves the information of past client requests.

Both would not work as different job types would need the protocol to be stateless, while others would require it to be stateful, and both HTTP and WebSocket only handle one case. Thus, the creation of a new network application protocol is needed to fit the applications requirements.

1. **Provide the implication source code of your network application protocol with sufficient test cases based on the design goals, message philosophy (format, structure, semantics), and communication rules.**

Here is the link to the implementation of our network application protocol:

<https://github.com/cursit2/3670_Project1>