COMS 6181 Advanced Internet Services

RELOAD P2PSIP

Final Project Report

Jehanzeb Khan (jk3305)
Dainis Kiusals (dvk2102)
Alejandro Mesa (am3473)

1. Overview

Application layer protocols which are prevalent in the modern Internet either fall in the client/server or peer-to-peer architecture. Most of the application layer protocols that are used directly in the end systems are naturally client/server based protocols where the implementation of the same protocol is different for client [requester of an abstract service] and server [entity processing the request]. HTTP is an example of one such protocol that with the implementation of a few methods on the http clients such as GET and POST methods and their processing on the http servers brings in to realization a genre of services. On the other hand, we see peer-to-peer protocols more dominant in the intermediate systems such as routers where most of the routing protocols such as Routing Information Protocol and Open Shortest Path First Protocol are peer-to-peer in nature, meaning any entity can be a client or a server hence all entities in a peer-to-peer protocol must have symmetric implementations.

Like any other construct in the information and communications technologies both approaches [client/server and peer-to-peer] have pros and cons. Client/server protocols have an advantage of shifting majority of the protocol complexity on the server side, leaving implementations on the end-hosts relatively easy. On the other hand peer-to-peer architecture have advantage or providing scalability, very little dependency on centralized nodes and fault tolerance. Due to the benefits provided by peer-to-peer architecture the ICT community relatively recently started exploring the possibilities of peer-to-peer based protocols in the end-hosts. The benefits provided by peer-to-peer protocols are broad enough to be leveraged in variety of applications in the end hosts, however the contemporary efforts and discussions are mostly related to multimedia applications.

P2PSIP or peer-to-peer SIP is a peer-to-peer protocol that provides a distributed architecture for implementation of such services as multimedia telephony among others without relying on a conventional client/server based infrastructure. For example in SIP numerous central authorities such as a proxy servers and abstract local databases are required. P2PSIP allows implementation of these infrastructure nodes/services among set of cooperating peers that are distributed in nature. Hence P2PSIP is expected to provide scalability, fault tolerance, high availability gains, among others. The P2PSIP protocol is comprised of layered architecture that is implemented solely in the Application Layer [in a typical OSI or TCP/IP layer model]. P2PSIP is comprised of the following layers:

Usage Layer: The usage layer allows different applications such as voicemail and presence to be implemented on top of P2PSIP without a centralized servers' intervention

Storage Layer: The storage layer provides an abstract storage service to the usage

layer for storage, retrieval, update, modification and deletion of usage layer data. The physical location of the actual data change as peers join and/or leave the overlay though this process largely remains transparent to the usage layer

Message Transport Layer: Message transport layer provides reliability mechanisms to the other layers from sending messages across to a different peer(s), in case where the underlying transport channel is unreliable, such as UDP. This layer provides equivalent function of transport layer in the TCP/IP layer.

Topology Plugin: Topology Plugin provides mechanism for routing messages between peers, for locating stored data and for distribution of usage layer data across different peers.

A challenge that any peer-to-peer system may encounter is regulated distribution of keys to the alive peers in an overlay. P2PSIP does not mandate any particular algorithm for dealing with the abovementioned issue. Chord is one of the most popular algorithms that:

1. With high probability ensure equal distribution of keys to the connected nodes in an overlay

2. Besides linear symmetric recursive routing, Chord also provides a simply way of maintaining a routing table that consists of only log*n* unique entries where *n* is the length of nodeID in an Overlay. Since in our respective demo we used 14 bit identifier as a nodeID of a respective peer, only four entries in the routing table are required to be maintained on each node. Each entry is called a Finger, indentified by its index [Finger-1, Finger-2, Finger-3 and Finger-4] in this case. Routing in this way significantly reduces the number of hops required to be traversed when routing from one peer to another without not introducing complexity in the network

3. Provides a method called *stabilize()* for keeping current the predecessor and successor pointers of the respective peers. A predecessor pointer tracks the peer preceding the respective peer in a chord ring, while the successor pointer tracks the peer that follows the respective peer.

Forwarding and Link Management: The forwarding and link management provides the functionality of moving protocols data units across peers, by consulting the routing table from Topology Plugin. This layer provides equivalent function of datalink layer in the TCP/IP layer.

2. Project Purpose

The purpose of this project is to implement sufficient functionalities and interactions of different layers in the P2PSIP architecture. Since given the limited time and expertise we could not implement all of the vital functionalities, however an effort to deliver a reasonable implementation that could demo a boilerplate version is intended. This project also relies on certain assumptions/limitations made throughout its life-cycle. These assumptions are:

1. Security considerations such as signing of messages and data being stored in the overlay are not taken in to account

2. No DNS server for locating the bootstrap nodes is implemented

3. No boostrap server/node [that maintains a database of subset of peers] is implemented

Broadly, the purpose of this project is to learn P2PSIP:

1. To analyze the implementation of peer-to-peer architecture in the end-system

2. To implement very basic functionalities of the P2PSIP, namely joining an overlay by peer(s) and routing messages from different layers

3. Videlicet the above-mentioned implementation, provide a reasonable source code upon which future project can make further progress

3. What was accomplished?

Although the aim of the project was to implement and deliver a set of APIs capable of replicating the modified RELOAD algorithm explained above, time constraints and busy schedules of the team members only permitted the implementation of incomplete, but working sub systems. Following is a description of what was accomplished with respect to each layer.

a. Storage Layer

The Storage Layer was implemented with the following functionality:

- The basic unit of stored data is based on a reduced set of information from the RELOAD StoredData structure, including the stored time, lifetime and stored value. Values stored with an exists flag set to false are considered to be deleted values.
- The only Kind-ID implemented for this project was the single value. The array and dictionary Kind-IDs were omitted due to schedule constraints. Security was not addressed in this implementation, specifically – signatures, access policy and storage replication were not implemented.
- Inter-node communications involves the sending/receipt of Store Requests by nodes, as well as the sending of a Store Response. Functionality for Fetch Requests was also implemented, as well as the sending of a Fetch Response back to the requesting node.
- Intra-node communications included functionality to handle the addition of nodes to the topology. Specifically, an Add Indication message was defined to be sent form the Topology Plugin to the Storage Layer, providing details about the newly added node, triggering the Storage Layer to shift stored values appropriately to their proper owners. The Delete Indication message was also defined but the handling not fully implemented.

b. Topology Plugin

The functionalities implemented in the Topology Plugin layer are as follows:

- Creating a NodeID from the existing IP address of the respective peer(s)
- Sending and processing of Join request
- Maintenance of predecessor and successor pointers by sending Stabilize request after a regular interval
- Notifying the storage layer when a new node leaves or joins the overlay
- Maintenance of routing table
- Providing routing-table lookups to the Forwarding and Link Management Layer

c. Message Transport

The Message Transport is the layer with the least amount of functionality. The layer contains minimal amount of code to handle message exchanges between the layers as described in the RELOAD RFC. Although the specification describes that this layer must provide reliable message exchange between nodes, our implementation only handles message exchanges within a node and across the overlay. To do so, the layer takes advantage of queueing mechanisms incorporated in the Java language to handle incoming and outgoing messages using a FIFO scheme.

The following source files contain the logic implemented for this layer:

- InQueue.java - Implements a Java queue to handle incoming messages. This is mostly used by the Forwarding & Link Management layer as the interface to pass incoming messages to upper layers.
- OutQueue.java - Implements a Java queue to handle outgoing messages. This is used by all upper layers to indicate the Message Transport that it must send a message.
- MessageTransportReceiver.java - Thread used to listed for incoming messages that are pushed into the InQueue.
- MessageTransportSender.java - Thread used to listen for outgoing messages that are pushed to the OutQueue. It passes any outgoing message to the Forwarding and Link Management layer.

d. Forwarding & Link Management

The Forwarding & Link Management layer, as the name suggests, it responsible for sending and receiving messages to and from the overlay, as well as, establishing connections to other nodes. Most of the functionality of this layer was implemented per the RELOAD RFC except for the case of handling messages to wild-card nodes. Yet, the API can be easily extended to include this functionality.

This layer makes use of Java libraries to open TCP connections to other nodes in the overlay. In addition, it maintains a connection table composed of TCP sockets to properly route messages to other nodes. Last, it interfaces with the Topology Plugin to obtain new routes, and with the Message Transport queues to send and receive messages.

The following source files contain the logic implemented for this layer:

- ConnectionTable.java - Implements the connection table by using Java Hashmaps for storing pairs of Nodes and TCP Sockets of concurrent connections maintained with other nodes.
- MessageHandler.java - Thread used to read and parsing incoming messages. It implements the functionality described in section 5.1 of the RELOAD RFC paper.
- MessageListener.java - Thread used to listen for incoming messages. Once a message is received, it spawns a MessageHandler thread to properly parse and route the message.
- MessageSender.java - This class provides the logic to send a message on the wire to another node in the overlay.
- TCPConenction.java - Java interface for encapsulating a TCP connection
- TCPClientConnection.java - Implements the TCPConnection interface and provides functionality for client sockets. This class is used for opening new connections to other nodes.
- TCPServerConnection.java - Implements the TCPConnection interfaces and provides functionality for server sockets. This class is used by the MessageListener class.

4. Test Results

Since the RELOAD RFC only describes a common, but incomplete system that could be used to build P2P systems, it was difficult to develop extensive tests scenarios. Yet, we decided to test the inner workings of the RELOAD specification as described below.

a. Initiate a Single-Node overlay

This is perhaps the simplest test, but necessary to ensure that the overlay is formed. This test ensures that the single-node overlay can be formed which is the starting point of any P2P system.

The results were satisfactory which validates that our multi-threaded implementation can work in a single system without any issues.

b. Join a second node to the overlay

This test ensures that not only the overlay can be grown, but that messages between two nodes can be exchanged without any issues.

Once again, the results were satisfactory and the overlay was grown to two peer nodes.

c. Join a third node to the overlay

This test ensures that the overlay can be grown, but more importantly that he Topology Plugin properly handles route updates.

Unfortunately, due to time constraints, this functionality was not tested, but we believe that the code is capable of performing this test.

d. Perform a Store Request and Answer

This test ensures that storage key-value pairs are properly exchanged between peers as new nodes join the overlay and are made responsible of their own sections.

This functionality was testing when the second node joined the overlay, and it successfully received key-value pairs that it was responsible for from the admitting peer.

e. Perform a Fetch Request and Answer

This test ensures that fetch requests and answers are properly routed to requesting nodes.

This functionality was tested successfully, but with limitations. The storage request was only sent from a client to its responsible peer. So the scenario of querying for storage keys across the overlay was not tested.

f. Route a message through the overlay

This test ensures that a message sent across the overly is routed over multiple nodes. It validates that the Forwarding & Link Management layer, in combination with the Topology Plugin, properly routes a message to other nodes.

This test was performed by attaching two clients, each to a different peer. Client A sent a message a to client B, and the message was properly route through the overlay peers.

g. Sending a Stabilize message to the predecessor

The purpose of this test was to verify that once a node joins an overlay, it sends a stabilize request to its successor indicating its successor pointer. This helps the successor node, update its predecessor pointer in cases a node in within the range of the respective successors and its predecessor joined an overlay. Similarly the successor node reponds back the Stabilize answer with its predecessor pointer, giving an opportunity to its previous predecessor to adjust the its successor if applicable.

5. Next Steps

Future improvements in this RELOAD P2PSIP implementation would expand the current functionality as well as improve security. Enhancements to the system could enable the overlay network to properly function across firewall boundaries, and to implement security features

such as access policies and digital signatures. Replication of stored data would also improve availability, as any unexpected node failures would be gracefully recovered through the transfer of replicated data from neighboring nodes.

Within the Storage Layer, next steps would include addition of the array and dictionary Kind-ID data models, improving the storage capabilities provided to the usage layer, as well as the addition of the Stat and Find request and response primitives. The Forwarding and Link Management Layer should be enhanced to handle wildcard nodes and to handle updates to the connection table based on topology plugin route updates. The Message Transport Layer should be improved to handle reliable message transport, involving retries. The Topology Plugin should validate finger table updates for accuracy, and improve processing/forwarding of route updates to the Forwarding and Link Management Layer.