A LIGHTWEIGHT FRAMEWORK FOR PEER-TO-PEER PROGRAMMING*

Nadeem Abdul Hamid
Berry College
2277 Martha Berry Hwy.
Mount Berry, Georgia 30165
(706) 368-5632
nadeem@acm.org

ABSTRACT

Peer-to-peer systems (P2P) have become one of the most popular Internet applications in use today. Implementing a P2P protocol requires the developer to manage a number of issues related to socket handling (both from a server and client perspective), threads, and message passing (or alternate means of communication between nodes). These infrastructure-related issues are often independent of the algorithmic details of the protocol itself. While there are frameworks available that encapsulate these low-level details for developers, they may not be suitable for students in an introductory networking course because of the overhead of having to learn the interface of a full-featured library. This paper presents the development of a lightweight, pedagogical framework for implementing and experimenting with P2P protocols and applications, aimed especially at students in introductory networking courses.

INTRODUCTION

Peer-to-peer systems (P2P) have become one of the most popular Internet applications in use today. According to recent analyses, P2P traffic takes up more than half of the bandwidth on the Internet [6]. P2P applications organize and manage resources at the edges of the Internet (e.g., user PCs) in a decentralized manner, with little or no interaction with centralized servers [9]. Thus, in order to operate as a P2P node, a computer needs to perform tasks typical to both traditional servers as well as clients, and to constantly monitor nodes joining and leaving the network.

Implementing a P2P protocol requires the developer to manage a number of issues related to socket handling (both from a server and client perspective), threads, and message passing (or alternate means of communication between nodes). These infrastructure-related issues are often independent of the algorithmic details of the protocol itself. While there are frameworks available that encapsulate these low-level details for developers, they may not be suitable for students in an introductory networking course because of the overhead of having to learn the interface of a full-featured library. Perhaps even less attractive, from an educational perspective, is that such libraries may provide too much infrastructure, so that the user is forced to use a particular P2P discovery algorithm, for example. Changing or experimenting with the underlying algorithm for searching the P2P network in such libraries requires digging into the internals, a daunting prospect for novices.

This paper presents the development of a lightweight, pedagogical framework for implementing P2P protocols and applications. It is "lightweight" by virtue of a simple interface and implementation, suitable for student experimentation and extension. The framework provides the code needed for basic infrastructure tasks such as those mentioned above: setting up a server socket, threads to handle incoming connections, and dispatching requests to appropriate handlers. This allows users of the framework (e.g., students) to focus more immediately on implementing the particular details of a P2P protocol and related algorithms.

Along with source code, this project provides an online tutorial introduction to developing P2P applications that walks through the actual implementation of the framework infrastructure. The library itself is currently implemented in both Python and Java; P2P applications that use either version of the library are interoperable. This framework has been developed for a newly introduced course, "Netcentric Computing," at the author's college and is in a state of active prototyping and revision.

In the remainder of this paper, some background and related work is discussed. Then, an overview of the library's design and a couple of sample applications are given. A number of directions for future development are discussed before concluding.

BACKGROUND

Peer-to-peer networking is an active research topic in the CS community. A closely related concept is that of an overlay network: a computer network built on top of another one. Many P2P networks are overlays because they run on top of the IP-based Internet, routing messages using identifiers other than IP addresses. While students with an understanding of writing servers and clients should be able, in principle, to write a P2P program, it may be difficult to understand how to combine both server and client functionality in a single entity while managing other issues unique to P2P. At the least, it seems that students would spend much time on getting the infrastructure working instead of being able to concentrate on the essence of the P2P design.

Standard networking textbooks, such as [6, 7], include high-level discussions of peer-to-peer and overlay networks. Also, there are several good survey articles and volumes [2, 10, 12] written on P2P technologies. Our desire is to allow students to gain more experience with protocols such as those described in these works or in other current

research. As mentioned in the introduction, there are a number of full-featured P2P infrastructures available for use, such as JXTA, Pastry [8], and Groove [4]. Besides being too complex to fit into a few weeks of an introductory networks course, however, such infrastructures often encapsulate a particular P2P protocol and are intended more for developing applications based on the provided protocol.

A LIGHTWEIGHT FRAMEWORK FOR P2P DEVELOPMENT

As discussed above, the main objective of developing this library was to get users who want to implement a P2P protocol up and running in a short time. Once the developer has decided what types of messages will be sent by the protocol, it is, for the most part, simply a matter of programming handlers for each message and registering them with the system. The library encapsulates operations common to P2P applications. While a variety of instantiations for P2P protocols may eventually be developed and provided with the library, the idea is to allow students to gain experience developing and debugging implementations themselves. The source code for the library is currently less than 600 lines, in both the Java and Python versions.

The Peer module and peer connections

The Peer module (implemented as a class in Python and Java) manages the operations of a single node in the P2P network. It contains a main loop which listens for incoming connections and creates separate threads to handle them. The programmer registers handlers with the module for various message types, and the main loop dispatches incoming requests to the appropriate handler. The peer is initialized by providing a port to listen for incoming connections, and optionally a host address (i.e. IP address, which may be automatically determined) and node identifier.

A list of known peers is also maintained by the node, which may be accessed and modified through the Peer module. The size of the list may be limited, and peers may be accessed using their identifiers or their sequential position in the list. Besides a list of handlers for various method types, the node also stores a user-supplied function for deciding how to route messages, and can be set up to run stabilization operations at specific intervals.

The PeerConnection module encapsulates a socket connected to a peer node. The library uses TCP/IP sockets for communication between nodes. A peer connection object provides high-level methods for sending and receiving messages and acknowledgments. It ensures messages are encoded in the correct format and tries to detect various errors.

Message format and handlers

Messages exchanged between nodes in the P2P system are prefixed by a header composed of a 4-byte identifier for the type of the message and a 4-byte integer holding the size of the data in the message. (We are also considering including a unique identifier with every message sent, but currently if a protocol requires such, it may be included in the body of the message.) The 4-byte message code can be viewed as a string, so the user of the library may come up with appropriate strings of length 4 to identify the various

types of messages exchanged in the system. When the main loop of a peer receives a message, it dispatches it to the appropriate handler based on the message type.

A message handler is simply a function object in Python (or an object supporting the handler interface in Java) that receives a reference to an open peer connection and the message data. Handlers can be registered for any message type identified by a 4-byte string. Currently only one handler per type may be used. When a node receives a message from a peer, the receiving node sets up a peer connection object, reads in the message type and remainder of the message, and launches a separate thread to handle the data. The peer connection is automatically closed when the message handler completes its task.

Routing and stabilization

The nodes in a P2P network may route messages based on the peer names (identifiers), instead of simply IP addresses. This would be an instance of an overlay network. Our framework allows nodes to be assigned names as required by the protocol being implemented. The programmer can then register a routing function with the Peer module that decides how to route messages based on the peer names.

Because nodes are continuously joining and leaving a P2P network, protocols may require some sort of stabilization routine to be run at regular intervals. At the least, a node may need to "ping" its known peers every now and then to make sure they are still there. The library provides users a simple way to register such stabilization routines, which are run in a separate thread at specified intervals.

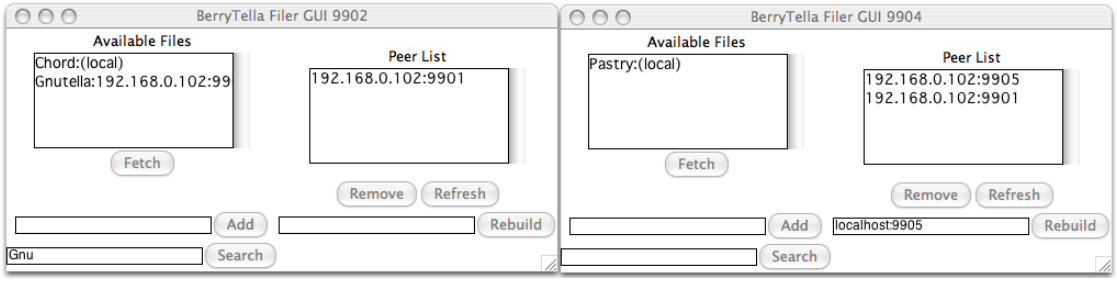

Figure 1: A simple P2P file-sharing application.

SAMPLE PROTOCOL IMPLEMENTATIONS

A Simple File Sharing System

Provided with the P2P framework library is a simple instantiation of a P2P protocol for file-sharing with a GUI interface (see Figure 1). The protocol implemented is extremely simple, leaving its improvement as a possible assignment. (As with the library itself, the additional code for this application is under 600 lines.) There are 7 types of messages exchanged by the nodes (actually 9, including positive and negative acknowledgments). Peer identifiers are strings of the form "host:port", using IP addresses directly for routing purposes. The maximum size of the peer list (known peers) is specified upon startup, as is the TCP port on which a node listens for connections.

This program implements a pure P2P network– there is absolutely no centralized directory of peers. A node joins the network by contacting another peer that is already in it. The new node builds its own list of known peers by performing a depth-first search of peers known to the initial contact. The depth of the search is limited by a number-of-hops parameter, and continues until the size of the peer list has reached its limit (or there are no more peers to contact). The algorithm also ensures, of course, that the node itself is not added to its own peer list, nor are duplicate peer names.

Querying the network involves searching for a file name. This is achieved by a brute-force algorithm that floods the known peers with QUERY messages. The peers either respond with a QRESPONSE message or propagate the query onto their own respective peers. The process is limited by a number-of-hops counter that is decremented in each message as it is propagated. Once the location of a file has been discovered, it can be fetched to the local node by directly contacting the peer that owns the file. The peer does not necessarily have to appear in the immediate peer list in this case, because peer names provide the direct IP address and port to contact. Upon receiving a FGET message, a peer replies with the contents of the requested file, if it is valid.

Other message types used in this protocol are NAME, to request a peer's canonical identifier; LIST, to which a peer responds with a list of all other peers in the network that it knows about; and JOIN, which requests that a node add a specified host/port combination to its list of known peers. Also, the QUIT message indicates a graceful exit of a node from the network.

The GUI code sets up the window frame components and event handlers for the various buttons, shown in Figure 1. It then launches the peer main loop, and starts a simple stabilization routine to ping peers every few seconds to make sure they are still alive. Local files can be registered with the peer and files stored elsewhere in the network can be searched by file name (or substring thereof). If the number of names in the peer list decreases due to nodes leaving the network, the "Rebuild" button allows the user to launch the depth-first search routine for new peers, given a node name to start from.

A Chord Protocol Implementation

The Chord project "aims to build scalable, robust distributed systems using peer-to-peer ideas" [1]. Chord is a search protocol that addresses the problem of efficiently locating a node that stores a particular data item. An instance of a class of distributed systems known as distributed hash tables, Chord maps keys onto nodes and adapts efficiently as nodes continuously join and leave the system [11]. Theoretical and experimental results show that communication costs scale logarithmically with the number of Chord nodes (unlike the exponential query algorithm described in the previous section).

Briefly, Chord assigns every node a numeric identifier by hashing the node's IP address. Similarly, every key (e.g., file name) is hashed to a numeric identifier. The identifiers are ordered in a "circle", modulo 2^m, where m is an input parameter to the protocol. The minimum amount of routing information required is for each node to be aware of its successor on the circle. However, to make searching more efficient, nodes

store an extra "finger table" of size *m*, allowing quick access to nodes up to halfway around the identifier circle.

The basic Chord algorithms described in [11] have been implemented by the author on top of the P2P infrastructure library of this work. (Details omitted due to space.) This demonstrates an intended use of the framework for class assignments or projects: students may research, or be given, a "real-world" or experimental protocol and are asked to implement it based on the text description or pseudocode given in a research article. It often takes a bit of testing and debugging to figure out critical details omitted in the pseudocode description. Understanding such details will hopefully help students appreciate such protocols and networked computing in more depth.

FUTURE WORK

There are many exciting features that can be incorporated into the framework to better support students' debugging, testing, and analysis of implemented protocols. Currently, the peer connection module directly interacts with the socket module in the standard language library. We plan to add an extra layer of indirection which would allow messages to either be sent over the actual network, or through a simulated layer on the local machine. The extra layer could also log all messages to the local machine as they are sent over the network, or forward copies to a server for monitoring and analysis. We would also like to integrate better logging support for local debugging messages.

To support testing and analysis, it would be interesting as a long term goal to interface with an existing network simulator framework. Additionally, for testing purposes, we are considering a simple scripting language to launch nodes and generate test messages between them. Again, this may be tied into an interface with large-scale experimental platforms such as PlanetSim [5] or PlanetLab [3].

CONCLUSION

This paper has presented initial development of a resource to help educators introduce P2P protocol development and application programming into the coursework of a networks class. The P2P framework is designed to be "lightweight" enough to allow students to browse through the source code, aided by a tutorial introduction to P2P programming, to help them develop their own protocol implementations. The library itself, implemented in both Python and Java, encapsulates basic network and threading tasks to facilitate implementation of P2P protocols.

ACKNOWLEDGMENTS

This work was supported by a Berry College Summer Grant for Course Development. Patrick Valencia, undergraduate student, assisted with the Java development.

REFERENCES

[1] The Chord project, pdos.csail.mit.edu/chord, retrieved September 2006.

[2] Androutsellis-Theotokis, S., Spinellis, D., A survey of peer-to-peer content distribution technologies. *ACM Computing Surveys*, 36(4):335–371, 2004.

[3] Chun, B., Culler, D., Roscoe, T., Bavier, A., Peterson, L., Wawrzoniak, M., Bowman, M., Planetlab: an overlay testbed for broad-coverage services. *SIGCOMM Computer Communications Review*, 33(3):3–12, 2003.

[4] Edwards, J., *Peer-to-Peer Programming on Groove®*, Boston, MA: Addison Wesley Professional, 2002.

[5] Garcia, P., Pairot, C., Mondjar, R., Pujol, J., Tejedor, H., Rallo, R., Planetsim: A new overlay network simulation framework, *Software Engineering and Middleware, SEM 2004 (Lecture Notes in Computer Science)*, 3437:123–137, March 2005.

[6] Kurose, J. F., Ross, K. W., *Computer Networking: A Top-Down Approach Featuring the Internet, 3/E*. Boston, MA: Addison-Wesley, 2005.

[7] Peterson, L. L., Davie, B. S., *Computer Networks: A Systems Approach, 3/E*. San Francisco: CA, Morgan Kaufmann, 2003.

[8] Rowstron, A. I. T., Druschel, P., Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems, *Middleware '01: Proceedings of the IFIP/ACM International Conference on Distributed Systems Platforms Heidelberg*, 329–350, 2001.

[9] Shirky, C., What is p2p. . . and what isn't, www.openp2p.com/pub/a/p2p/2000/11/24/shirky1whatisp2p.html, retrieved September 2006.

[10] Steinmetz, R., Wehrle, K., *Peer-to-Peer Systems and Applications (Lecture Notes in Computer Science)*, Secaucus, NJ: Springer-Verlag, 2005.

[11] Stoica, I., Morris, R., Liben-Nowell, D., Karger, D. R., Kaashoek, M. F., Dabek F., Balakrishnan, H., Chord: a scalable peer-to-peer lookup protocol for internet applications, *IEEE/ACM Transactions on Networking*, 11(1):17–32, February 2003.

[12] Subramanian, R., Goodman, B. D., *Peer to Peer Computing: The Evolution of a Disruptive Technology*, Hershey, PA: Idea Group Publishing, 2005.