

# OogP2P Framework Overview

Sanjay Ginde\*  
Duke University  
sjg@duke.edu

David Goldberg†  
Duke University  
dtg@duke.edu

Chris Zeiders‡  
Duke University  
cmz@duke.edu

December 4, 2003

## Abstract

The OogP2P framework is a framework for developing and running applications on a peer-to-peer system. The OogP2P framework is designed in such a way that users can develop and run new applications while having to add a constant,  $O(1)$ , amount of code to the existing framework, without any modification to pre-existing applications running on the system, and without any modification to the network model. *Applications* are not limited to programs such as file sharing, chat, and networked games, but also include peer-to-peer components such as algorithms for searching, load balancing, and network-data gathering, to name a few. Developers can create applications by following the Communication and Application Architectures as specified in this paper and the framework documentation[2].

The OogP2P framework provides the base networking protocols for a decentralized peer-to-peer network and any applications developed from our framework are geared to run on this network .

We developed a few applications that span different areas of p2p systems in order to show the flexibility and extendibility of the OogP2P framework: a simple chat application, an in-house implementation of the Chord DHT searching algorithm, a statistical application to garner statistics about the Chord

search's performance, and adapted FreePastry-a previously existing implementation of the DHT searching algorithm Pastry developed at Rice University-to work within the OogP2P framework.

## 1 Introduction

Since the emergence of popular peer-to-peer systems like Napster and Gnutella, there has been an overwhelming interest in peer-to-peer systems. This can be seen today in both the sheer volume of p2p programs and the wide scope of applications p2p systems support. Further, peer-to-peer systems have ignited a strong interest in both the academic and commercial worlds; there has been an explosion in theories and papers concerning distributed searching algorithms, load balancing algorithms, and garnering statistical data on peer-to-peer systems by institutions ranging from Microsoft to MIT. These trends indicate a clear need to do more than simply create new p2p applications: one should be able to readily implement and test various components of p2p networks along a static baseline.

The OogP2P<sup>1</sup> Framework seeks to fulfill these needs. It was designed to specifically address three important concerns of p2p application design: to implement p2p features without changing the network model, to implement those same features with a constant,  $O(1)$ , amount of added code to the existing system, to use a modular design which would allow

---

\*sjg@duke.edu

†dtg@duke.edu

‡cmz@duke.edu

---

<sup>1</sup>OOG stands for *The Object-Oriented Gangsta's*

for avoiding modifications of existing features when adding new features and thus be able to test different p2p metrics along a set baseline. As a result of these implementation decisions, such a framework would allow for a variety of interesting applications in settings ranging from Educational to Commercial.

## 2 Related Work

The most closely related project to the OogP2P Framework is the JXTA project <sup>2</sup>. JXTA attempts to provide a set of tools on which peer-to-peer networks and applications can be constructed. The project existed before OogP2P and was looked at as an alternative to creating a whole new framework. However, after working with JXTA, it was realized that a new simpler, more accessible set of tools was needed, thus resulting in the creation of the OogP2P framework.

Another important field of related work is the development of scalable, self-organizing topologies for peer-to-peer systems. These systems (CAN [3], Chord [6], Pastry [4], Tapestry [7]) provide efficient routing of information across the network. They are high-level systems that when implemented on the OogP2P framework, demonstrate the framework's extendibility and flexibility. In particular, Chord and Pastry are implemented as case studies in section 8.

## 3 Network Model

The OogP2P networking model performs two fundamental tasks-it facilitates the sending and receiving of information between clients and can determine which current feature running on the system can process the received data.

The OogP2P framework was designed with the idea of a simple, yet robust networking model. This led eventually to a set of classes that would operate self-sufficiently, ensuring that the networking core would need no modification no matter the features

---

<sup>2</sup>More on JXTA can be found at <http://www.jxta.org/>.

one cared to implement. This principle of a constant network core does not necessarily exclude changing the actual network topology. We successfully implemented both Chord and Pastry, two organization algorithms that utilize a ring topology.

While it may not be the most efficient network implementation, its simplicity lends itself to be easily understood and attainable for even the novice network developer, as well as open to improvement from more expert networking developers.

## 4 Communication Architecture

Built on top of the network layer is the Communication Architecture. The main goal of this layer is to provide a simple, yet powerful means for structuring communication between nodes. It consists of providing interface objects to hide the network model from coders, so those with little or no network programming experience can program within the system. (It also allows for a new network model to be slipped in if desired).

Also, the Communication Architecture is built on a protocol that consists of two objects, the Nettable and InfoReceived. Nettables essentially provide wrappers around data and information passed through traffic. All network traffic must extend from the Nettable object as to provide a common way all data can be handled and distributed to the different applications running on the framework. The Nettables have parallel InfoReceived objects that are used to process those Nettables on local nodes. More information on the Communication Architecture can be found in the framework documentation at [2].

## 5 Implementing features with constant O(1) code

The OogP2P framework's Application Architecture [2] was specifically designed to enable the incorporation of new features with a constant, O(1), amount of added code to the framework-system, while

simultaneously avoiding alterations to pre-existing features running on top of the system. This is primarily made possible by controlling the way in which incoming data is processed. By virtue of the design of the framework, arriving data is processed using a factory pattern built around all the features running on this system. This means that to add new functionality, only two things need to be done: write the new code (separate from the framework code) in order to implement the new feature, and then add one line of code to the existing code that moderates the factory.

This simplicity in the creation of new features without modification of old code provides a couple of important advantages. First, and perhaps foremost, is the fact that one can easily apply metrics to test different features without worrying about how code differences will affect timing. For instance, if one were to be interested in the amount of time it took to find a file on a million nodes in a Chord implementation vs. in a Pastry implementation, one could simply design the two sets of packets needed, and get the metrics without worrying about how changes in existing code might have affected those measurements. Second, the developer can implement and debug a feature completely separate from the OogP2P framework and then adapt this feature to work within the OogP2P framework, allowing for applications designed on other systems to be readily modified to run on the OogP2P system.

The term, *features* can apply to any type of assessment necessary in p2p research from load balancing to node-failure/join statistics, demonstrating the flexibility and extendibility of the framework.

## 6 Modular Code

Another strength of the OogP2P framework lies in its modular design. The structure of OogP2P allows for the truly independent implementation of features: each new feature can be added to the system without worrying about affecting the greater structure of the p2p program or other features that

previously exist. This necessarily implies that the basic framework code as well that of other features can also remain unchanged. This truly simplifies the job of the programmer. Specifically, if one were to be interested in creating a new feature, like a chat application, all that would be necessary would be a new package containing the code to implement a chat application and the addition of one line of code to the application factory. This advantage can be helpful in a variety of ways. In an educational atmosphere, for instance, a teacher could assign students a project in which they were to implement a tic-tac-toe game. To do so, the students would need only create the tic-tac-toe package, write the one line of code that would add the application to the factory, and not have to worry about how the code would fit into the rest of the framework.

Also, in a commercial atmosphere, if a company were to decide to implement a suite of conferencing applications like chatting, teleconferencing, and whiteboard, the projects could be developed simultaneously without worry of one person's code interfering with another due to the fact that no code from within the application would need alteration. Although this can be done using CVS, this eliminates any chance of error that seems to be common when a lot of people are working on the same code. Further, as there is no necessity to modify previously existing features or code, the programmers do not have to concern themselves with the possibility of changing some class that will affect how others function. This reduction in the amount of human error and time spent cleaning up human error can provide huge advantages for any commercial endeavor.

## 7 Settings For Use

As mentioned previously, this framework can be useful in a range of settings. A select list will be examined, with a short description of how one might implement the OogP2P framework in each situation.

- **Educational Setting:**

- Instructors can assign programming projects to novice developers in introductory courses and with a few additions to work within the OogP2P framework, the project can be a p2p-networked application.
- The OogP2P framework takes advantage of some key Object-Oriented development designs and can be used as an introduction or a complimentary teaching tool to understanding OO-programming.

- **Academic Uses**

- Researchers can implement components of p2p systems, such as searching algorithms or load balancing algorithms, and have to only be concerned primarily with writing the code to implement their algorithm.
- Researchers can develop combine similar applications running on the system in order to develop some optimization schemes. For example, a researcher can develop searching-optimization logic to decide when to use parts of the Chord and Pastry algorithms.
- Can be used as a statistical tool to compare the performance of p2p components.

- **Commercial Uses**

- Source Control- Could have multiple people working on different packages
- Could be used in a variety of testing situations

## 8 Case Studies

The following section aims to demonstrate the extendability and flexibility of the OogP2P framework.

1. Chord [6], Chord Statistics
2. Pastry [4]

### 8.1 Chord: In-House Implementation

Chord is a distributed searching algorithm that uses consistent hashing in order to look up a key and find the node in the network that tells the searching node where its desired resource is located. To do this, it keeps track of only  $\log(N)$  nodes in the network (called fingers), and uses a form of distributed binary search to find the exact node it is looking for. This is in contrast to most decentralized P2P searching algorithms, which flood the network with requests.

We successfully implemented a version of Chord to run on the OogP2P network. The *hard part* of chord-implementing the logic for creating a ring topology and hashing nodes/keys-is contained in objects independent of the architectures of the OogP2P framework, and in fact was created in an environment outside of the OogP2P framework. All the debugging needed to ensure that Chord was running correctly was done in this outside environment.

In order for our Chord to run on the OogP2P system, certain classes were needed to function within the Communication Architecture:

ChordRingQuery	FindFingerSuccessor
ChordRingExists	FoundFingerSuccessor
AddKeyToTable	GetMapping
FindKey	GetMappingAnswer
FoundKey	Stabilize
InsertKey	StabilizeResponse
Notify	

For every piece of communication the chord nodes need to pass between themselves, two objects were created; one object that contains the data and is sent over the network, and one object to execute some logic when the first object is received. Examples of such communication include when a node joins, adding files to the system, and updating node structure, to name a few.

Classes also had to be created to work within the Application Architecture:

ChordApplication	ChordFactory
------------------	--------------

## ChordAdministrator

All the classes that were created in order to implement the Chord feature were encapsulated under a ChordApplication object. [2]

This ChordApplication object was then added to the collection of features the OogP2P system supports and was then successfully run.

## 8.2 Chord Statistics

Another feature, a package of statistics, was created in order to collect some information on the network and the effectiveness of our Chord implementation. The metrics included the number of hops needed to find a file, what data is being sent between clients, and the time it takes for a search query to produce a result.

Again, the classes to implement these statistics is added code and does not involve any modification to the existing network and framework.

After the statistics feature was designed and implemented, the appropriate objects were created for data to be sent through the network. Furthermore, all objects correlating to the statistics feature were encapsulated inside a RecordApplication object as specified by the Application Architecture.

## 8.3 Pastry: Adapting an Existing Implementation

Like Chord, Pastry is a distributed algorithm that uses consistent hashing for look-up resources on the peer-to-peer network. However instead of creating our own implementation of the algorithm, like we did with Chord, we adapted an implementation in existence well before OogP2P was envisioned: Rice University's FreePastry<sup>3</sup>. By adapting FreePastry we had the goal of showing another facet of flexibility of the OogP2P framework.

---

<sup>3</sup>FreePastry can be found at Rice University at <http://freepastry.rice.edu/>

The process started by extracting the pertinent parts of FreePastry and modifying them slightly to make them follow the style of the framework. Very little of the algorithmic code within the classes was changed.

Also, like Chord described before, all these new additions did not require modifying to the existing framework and network. It was all additional code created in a package separate from the framework. All these new classes were incorporated within a PastryApplication object, allowing it to be easily plugged into OogP2P system as a new independent feature.

## 9 Future Work

Currently the GUI is a bare-bones Console Window. Future work will involve making GUI much more functional and user-friendly, through the use of good layout and good use of pull-down menus, buttons, etc.

There is a need for the generalization of both the input and output mechanisms. Currently the input and output are directly tied to the Console Window. This should be generalized so that future means of input and output and be developed for the system.

The processing of commands is a little bit odd in the OogP2P Framework. Under the current implementation, the output of the console window is directly tied to the client to which you are connected thus the processing of the commands is done on that client. This is counterintuitive since commands should be processed locally.

An original goal of the project was to run the framework on the large scale Internet emulator ModelNet<sup>4</sup>. However, due to complications with Java code and IP generation and time constraints this goal was not able to be reached. In the future, the problems are hoped to be worked out to allow for large scale test.

---

<sup>4</sup>ModelNet can be found at Duke University's ISSG at <http://issg.cs.duke.edu/modelnet.html>

## 10 Conclusion

This paper presents the OogP2P Framework as an extensible, easy to use, and flexible system on which peer-to-peer components and applications can be developed and tested. It is built on top of a simple, yet powerful Communication Architecture that allows varying forms of interaction between peer nodes. Also, the Application Architecture is designed in such a way to allow only the need for a constant,  $O(1)$ , amount of additional code to incorporate a new component or feature to the system.

The case studies for Chord, Chord Statistics, and Pastry all show the design features of the OogP2P Framework. Chord and Chord Statistics show the extensibility by adding new, diverse features, while Pastry showed flexibility by the adaptation of existing code.

With the rapidly increasing interest in peer-to-peer technologies the OogP2P Framework hopes to provide a useful tool in the development and testing of existing and new P2P algorithms and applications.

## Acknowledgements

The OogP2P Framework would not be what it is today without the help and guidance of Robert Duvall, who was the advisor to the project. We must also thank Patrick Reynolds and Amin Vahdat for helping us find resources and providing support throughout. And although we did not get a chance to run and test the framework on the Internet emulator ModelNet, we would still like to thank Ken Yocum and David Becker for helping us get setup on the system for possible work in the future.

And lastly, we would like to thank Owen Astrachan and Jun Yang for taking time out of their busy schedules and serving on presentation committee for the project.

## References

- [1] Gamma, E., Helm, R., Johnson, R., Vlissides, J. *Design Patterns: Elements of Reusable Object-Oriented Software*, Reading, Mass.: Addison-Wesley, 1995
- [2] Ginde, Sanjay, David Goldberg, and Chris Zeiders, *OogP2P Framework Documentation*, 2003.
- [3] Ratnasamy, S., P. Francis, M. Handley, R. Karp, and S. Shenker. *A Scalable Content-Addressable Network*. In *Proc. of ACM SIGCOMM*, Aug. 2001.
- [4] Rowstron, Antony and Peter Druschel. *Pastry: Scalable, decentralized object location and routing for large-scale peer-to-peer systems*.
- [5] Shalloway, Allan and James R. Trott. *Design Patterns Explained: A New Perspective on Object-Oriented Design*, Addison-Wesley, 2002.
- [6] Stoica, Ion, et al. *Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications*. 2002
- [7] Zhao, B. Y., J. D. Kubiatowicz, and A.D. Joseph. *Tapestry: An infrastructure for fault-resilient wide-area location and routing*. Technical Report UCB//CSD-01-1141, U.C. Berkeley, April 2001.