OMNIX: A topology-independent P2P middleware

Roman Kurmanowytsch, Engin Kirda, Clemens Kerer and Schahram Dustdar

Technical University of Vienna, Distributed Systems Group
Argentinierstr. 8/184-1, A-1040 Vienna, Austria
Phone: +43 1 58801 18418, Fax: +43 1 58801 18491
{R.Kurmanowytsch,E.Kirda,C.Kerer,S.Dustdar}@infosys.tuwien.ac.at

Abstract. *Peer-to-Peer (P2P) computing has been increasingly gaining interest. P2P architectures are no longer just used for sharing music files over the Internet. P2P middleware systems attempt to provide an abstraction between the application and the underlying P2P infrastructure by providing higher-level services such as distributed P2P searches and direct communication among peers. Such systems often provide a pre-defined topology that is suitable for a certain task (e.g., for exchanging files). This topology influences the performance and scalability of the applications that are built on top of it. The problem is that it is difficult to change a topology-dependent architecture once requirements change or problems related to the topology are identified. In this paper, we present Omnix, a topology-independent P2P middleware. Omnix has a pluggable architecture that allows different topology descriptions to be plugged in based on the changing requirements of an application. Applications that use Omnix are provided a consistent API that is not affected whenever the underlying topology has to be changed or adapted.*

Keywords: Peer-to-Peer computing, P2P, Network topology

1 Introduction

Peer-to-Peer (P2P) computing has been increasingly gaining interest. This is because P2P is no longer just used for sharing music files over the Internet. Many P2P systems have already been built for other purposes and are being used (see [1]). An increasing number of P2P systems are used in corporate networks or for public welfare (e.g. providing processing power to fight cancer).

The term topology in P2P computing refers to the structure of the overlaying P2P network, but not the IP network beneath. It comprises peers and the connections between these peers. These connections may be directed, may have different weighs and are comparable to a graph with nodes and vertices connecting these nodes. Defining how these nodes are connected affects many properties of an architecture that is based on a P2P topology. The topology of a P2P architecture significantly influences the performance, search efficiency and functionality, and scalability of a system. Furthermore, it determines its flexibility (i.e., the ability to make changes to the system's configuration).

P2P middleware systems aim to provide an abstraction between the P2P topology and the applications that are built on top of it. These middleware systems offer higher-level services such as distributed P2P searches and support for direct communication among peers. Such systems often provide a pre-defined topology that is suitable for a certain task (e.g., for exchanging files).

All P2P topologies have some advantages and some disadvantages. The difficult question that is often faced by system designers and developers when choosing a P2P architecture is which P2P topology (and thus often which middleware) best fits the requirements. In our MOTION (MObile Teamwork Infrastructure for Organizations Networking)[2] project, for example, one of the main challenges was to choose an appropriate P2P topology for supporting mobile teamwork. The P2P service architecture that we developed supports mobile teamwork by taking into account the different connectivity modes of users, provides access support for various devices such as laptop computers and mobile phones, and uses XML meta-data and the XML Query Language (XQL) for distributed searches and subscriptions. Once we had built a prototype of the platform, we started seeing scalability and performance problems in practice that we had not anticipated before. Because the P2P middleware was an integral part of the system, it was not easy to simply change the current middleware and try out some other topology. Clearly, a more flexible and adaptable middleware was needed for the architecture.

In this paper, we introduce Omnix, a topology-independent P2P middleware. Omnix has a pluggable architecture that allows different topology descriptions to be plugged in based on the changing requirements of an application. The main advantage of Omnix is that it provides a consistent API to the application layer and allows the underlying topology to be changed without making code modifications in the application necessary. Custom-tailored topology descriptions can be constructed and inserted into Omnix, thus, increasing the flexibility of architectures that are built upon it.

This paper is structured as follows. The next section gives an overview of P2P topologies. Section 3 discusses Omnix. Section 4 presents related work. Section 5 discusses some of our future plans and Section 6 concludes the paper.

2 An overview of P2P topologies

This section provides an overview on the most common P2P topologies. It discusses the general advantages and drawbacks that all P2P topologies have in common.

2.1 Pure P2P

A pure P2P topology is a network of nodes that do not depend on a central server. Peers are directly interconnected and send messages to their neighbors. In some P2P networks, each peer receiving a request returns the result and in parallel forwards the request to one, some or all (depending on the protocol) of its neighbors. Since there is no central organizational unit, the peers have to maintain the topology and connectivity by themselves.

The advantage of pure P2P systems is the fact that there is no single point of failure (i.e. central server). Hence, the *survivability* of the P2P network increases.

One disadvantage of a fully distributed P2P network without a central authority is that it is difficult to update the system (e.g., whenever a new version of a component is available).

The most important problem of pure P2P systems is the bootstrapping process. When the P2P software is first started on a machine, at least one peer of the overall topology must be known in order to enable the connection to the network. In most cases, this is usually done by providing a fixed, well-known server that serves a list of some of the P2P peers. Another possibility is to employ connections to random IP addresses in certain address ranges where there is a good chance of getting a connection. However, this approach is regarded as being unfriendly and therefore, is only used as a last resort.

The algorithm for forwarding requests and locating data ranges from simple forwarding (e.g. Gnutella) to content-based routing (as, for example, in Tapestry [3]).

Typical examples in use for this type of P2P topology are Gnutella [4], Freenet [5], PAST [6] and Chord [7].

2.2 Server-based P2P

Server-based P2P topologies rely completely on a central server or on a set of well-known servers. There are different ways how a central server might be used. The server can provide search and indexing facilities (e.g. Napster [8]), authenticate users and encryption key distribution (e.g. Grokster [9]) or coordinate the work of the peers (e.g. Seti@HOME [10]).

There are several advantages when using a central server. The protocol can be easily changed and improved without having the problem of convincing all clients to support this protocol (although this is not always regarded as an advantage because a protocol should always be designed in a way that it is flexible enough). Furthermore, the usage of a central server eases the design of the security functionality. Authentication and authorization mechanisms can be employed because all significant traffic (e.g. search requests, login information, etc.) can be sent to a single central server. Security considerations are usually not important in file sharing systems, but they become necessary when the P2P network is to be deployed in a corporate environment. As well as security, billing is also much more easier to manage in a P2P network that can rely on a central server.

There are also drawbacks to central-server-based P2P networks. Obviously, the central server represents a single point of failure. Ideally, P2P networks should take advantage of a central server, but should also be prepared to function without such a service. An example for this functionality is Groove [11]. Groove instances may connect to a central server for permanently storing information. In the absence of a central server, computers running Groove can still interact and provide most of the functionality.

2.3 Hybrid P2P

A hybrid P2P system also uses servers for indexing, searching and message propagation. The difference to the server-based approach is that these servers are chosen from the pool of peers that are currently online.

The idea of using a hybrid P2P topology was first widely used in the FastTrack [12] network. Certain criteria such as bandwidth, latency and CPU power are used to elect the appropriate “server” (in FastTrack they are referred to as “SuperNodes”). A normal peer connects to a SuperNode and sends a list of all shared files. Search requests are sent to the SuperNode only, which, in turn, forwards the message to other SuperNodes. Every SuperNode receiving a query searches the local database of shared files (those made publicly available by the “client” peers).

This type of topology combines the advantages of server-based topologies (efficiency) with those of pure P2P topologies (no single point of failure, survivability).

3 Omnix

The Omnix P2P middleware has a four layer architecture. It is a modular framework where relevant parts of the middleware can be easily replaced by other components (see figure 1).

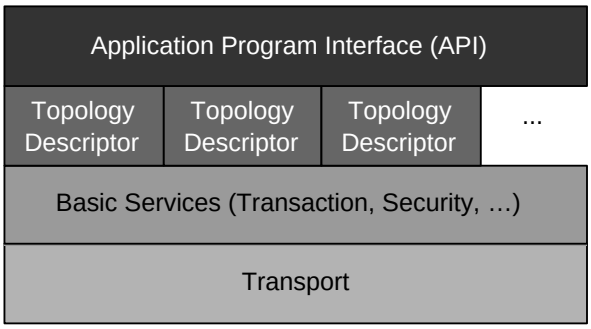

Fig. 1. The the four layer architecture of Omnix.

The lowest layer in the framework is the transport layer. It deals with the sending and receiving of messages over the network.

The interface between the transport layer and the layer above is network-independent. Hence, it is transparent to the upper layers which type of protocol is used beneath (e.g. UDP or TCP). As a result, the same P2P architecture can be used in combination with different network protocols. In the prototype, for example, there exist two transport components for Omnix. One component supports the communication via UDP messages. It uses normal socket operations provided by the Java runtime. Another component has been developed to make use of the datagram socket classes provided by J2ME (Java 2 Micro Edition).

The message exchange in the transport layer is done using a request-reply protocol. When a peer sends a message to another peer, it gets a response indicating how the remote peer has processed the message (indicated by a status code). The messages have

a similar structure to HTTP or SIP (Session Initiation Protocol) messages. A request has a line indicating what kind of message it is, an unlimited number of headers (holding information such as the address of the recipient, peers that have been passed on its way to the recipient, the content-length, etc.). The content is contained at the end of the message structure.

A response contains a status code along with a plain text message, that is used to tell the recipient whether the request has been processed successfully and if not, what the reason for the failure is. The content of a message is only defined by the upper layers of the Omnix architecture.

The message format is not fixed in Omnix. First, arbitrary headers can be added to the message (e.g. for adding the public key or other information that is important for a specific type of P2P topology). Second, the message structure itself can be replaced. By using another class to represent a message, any message format can be employed.

On top of the transport layer is the basic services layer. The basic services layer provides a set of pluggable services such as transactions, encryption and proxy for firewalls. Messages that are delivered by the transport layer are processed in this layer and then passed to the layers above.

The basic services layer uses the concept of pipelines to process messages. A pipeline is a collection of modules where messages are passed through until a module consumes the message. When a message has been received by the transport layer, it is put into the input-pipeline. Sequentially, each module gets a chance to process the message and may pass it to the next module. The message is dropped if the message is invalid or not allowed. A module may also modify the message before it is passed on to other modules. For example, a decryption module decrypts the message at the beginning before other modules process the contents of the message.

When a module receives a message, it may send a response to the originator of the message. Likewise, this message is again sent through a pipeline where modules may modify (e.g. encrypt) the message before it is delivered to the transport layer. A module can also stop the processing of a message and start a new request-reply transaction. An example for this situation would be the encryption of a message. If Omnix is about to send a message to another peer, the encryption module might cache the message and send instead a message to the recipient requesting a public key for encryption. Once the key has been received, the original message can be encrypted and finally sent to the recipient. Since this process could take more time than anticipated, modules are able to send responses back in the pipeline (e.g. sending “Trying” responses that do not indicate a finished request-reply transaction).

One of the major problems of P2P networks are firewalls. There are different ways of coping with firewalls. Many of the P2P systems currently available try to bypass firewalls by using well-known and typically open ports such as the port 80 (HTTP). However, this only forces system administrators to audit all traffic going over port 80.

In Omnix, we decided to give administrators the possibility to use proxies to let peers communicate across a firewall. This way, a peer behind a firewall can instruct the recipient of a message to send a reply to a proxy server instead of the peer itself. The proxy server can then forward the message to the recipient behind the firewall.

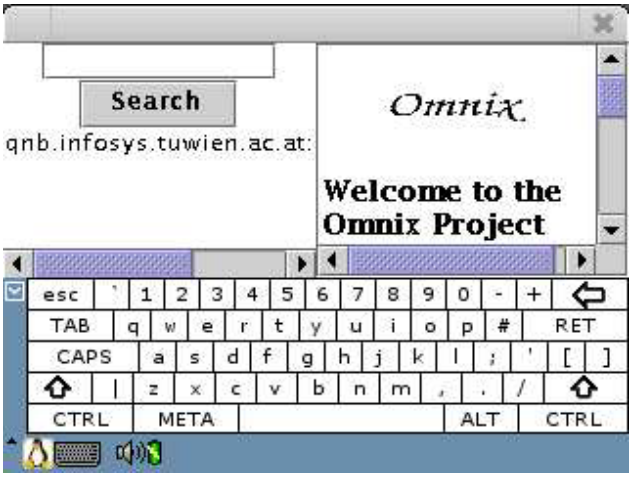

Fig. 2. Screenshot of Omnix running on a Compaq iPAQ with Linux.

On top of the basic services is the topology descriptor layer (i.e., peer discovery and structuring). Here, plugins are used to describe the structure (i.e. topology) of the P2P network. An Omnix instance may have many so-called Topology-Descriptors. These descriptors are responsible for detecting peers and organizing the structure of the network (i.e. performing the appropriate routing algorithms).

The topology descriptions are components that implement the pre-defined topology interface provided by the topology descriptor layer. This interface has simple methods that can be called by the upper layers such as search() (i.e., searching among peers) and connect() (i.e., connecting to a peer).

In the prototype of the framework, we currently provide topology descriptors for the pure P2P and server-based P2P topologies.

On top of the topology descriptor layer is the Application Programming Interface (API). This API is used to build applications on top of the framework.

3.1 Implementation

We have built a prototype of the Omnix framework. It requires no additional libraries (e.g., such as XML parsers). It has been tested on PCs and notebooks running Java 1.4.1 and a Compaq iPAQ H3600 running Familiar Linux 2.4.18-rmk3 with 64 MB of RAM (see figure 2). We used a Java runtime for the ARM processor provided by Blackbox. It provides a full-fledged runtime for Java 1.3.1.

The Omnix prototype also runs on J2ME enables devices (using the Mobile Information Device Profile - MIDP). We used the J2ME Wireless Toolkit version 2.0 Beta 2 under Linux RedHat 7.3. Figure 3 shows a screenshot of the demo application Simplix (see Section 3.2).

Fig. 3. Screenshot of Omnix running on a J2ME mobile phone emulator.

Omnix has been developed to use only the reduced set of standard packages provided by the MID Profile. Some modules, however, might not be used on a mobile phone (e.g. the encryption module that uses security packages provided only by the Java 2 Standard Edition). We have fully integrated a peer running on a mobile phone emulator with peers running on PCs and Linux handhelds.

Although Omnix has been implemented in Java, it is completely language-independent. The communication between peers does not rely on any Java-specific technology (e.g. Remote Method Invocation - RMI).

3.2 Applications and topologies implemented using Omnix

We have implemented two P2P topologies for Omnix. The first one targets mobile devices in a local environment and directly connects these devices with each other. It can be used for various pervasive computing scenarios where devices nearby are connected and all peers may connect and disconnect sporadically.

The other topology implemented is server-based. When a peer wants to provide a service (or a file), it sends a meta-data description of shared services to a central server. This server then processes the search requests.

Omnix is not restricted to the use of Topology Descriptors for a P2P network. To demonstrate this flexibility, we have implemented two proprietary P2P networks. The first one, Simplix, provides a framework for arbitrary services. Currently, there exist services for searching files, people and their .ldif files (LDAP Data Interchange Format, see [13]). Furthermore, one can search for books at Barnes & Nobles and look for references in Bibtex files.

We have also implemented a sophisticated hybrid P2P network that has roughly the same technical structure as the world's most popular P2P network, FastTrack [12]. The system provides SuperNodes to collect meta-data on shared services and files, and to process or forward search requests to other SuperNodes. The system provides additional features such as traffic shaping and load balancing.

4 Related Work

Many academic and commercial P2P systems exist (e.g. Gnutella [4], Freenet [5], Chord [7], PAST [6], etc.). However, only a small subset of these P2P systems can be used as middleware for other applications.

One P2P system that has been primarily designed to function as middleware is JXTA. Similar to Omnix, it is not a P2P application, but a framework that provides an abstraction between the P2P topology and the applications that are built on top of it. One major limitation of JXTA is that the applications that are based on it have to use a pre-defined topology. Furthermore, JXTA requires every peer to parse and generate XML messages. This is may be a serious problem on computing devices with resource limitations such as mobile phones (e.g., such as a J2ME[14]-enabled phone) and Personal Digital Assistants (PDAs). [15] gives a short overview of the main problems of using JXTA with J2ME.

Several other P2P systems that can be used as middleware are Pastry [3], PeerWare [16] and P-Grid [17]. Each of these P2P systems have a fixed P2P topology that cannot be changed. Furthermore, each P2P system has its own set of methods to access the functionality of the P2P system. There exists no standardized way of accessing a P2P network from within an application.

5 Future work

Currently, Omnix does not provide mechanisms to bridge between different topologies and to allow the topology to change dynamically at run-time. Dynamic topology reconfiguration is an interesting

research area and we plan to look at topology descriptions that can change dynamically.

We are currently working on a security module that can be used to encrypt and decrypt messages before they are processed by other modules. This module encrypts the content and most of the header fields that are not relevant for proxies. Hence, important header fields such as the recipient or the message ID are not encrypted. To solve the problem of message tampering, messages could also be signed.

6 Conclusion

Peer-to-Peer (P2P) computing has been increasingly gaining popularity. The P2P paradigm is especially useful in architectures where scalability and configuration flexibility issues are important and distributed search support is needed.

Omnix is a novel P2P middleware that is topology-independent. In contrast to other P2P middleware systems, Omnix can use any topology that is written and plugged into it. The same Application Programming Interface (API) to the Omnix middleware can be used to employ different topologies. Thus, the application developers are always able to use the same API, regardless of the topology of the underlying P2P network.

With the miniaturization and widespread availability of processors and advances in networking capabilities, the demand for P2P architectures in ubiquitous computing environments will grow. More and more devices will be able to communicate with each other soon by connecting to existing network infrastructures such as the Internet, the telephone network, or private company or home intranets. Omnix is an initial step in allowing the creation of highly flexible applications that can run on arbitrary topologies and devices.

References

1. infoAnarchy: infoAnarchy Resources, http://www.infoanarchy.org/?op=special&page=resources (2003)
2. Kirda, E., Fenkam, P., Reif, G., Gall, H.: A Service Architecture for Mobile Teamwork. In: 14th International Software Engineering and Knowledge Engineering Conference (SEKE 2002), Ischia, Italy, ACM Press (Jul. 2002)
3. Rowstron, A., Druschel, P.: Pastry: Scalable, Decentralized Object Location, and Routing for Large-Scale Peer-to-Peer Systems. Lecture Notes in Computer Science **2218** (2001) 329–??
4. Gnutella: Gnutella: The Gnutella homepage, http://gnutella.wego.com/ (2001)
5. Clarke, I., Sandberg, O., Wiley, B., Hong, T.W.: Freenet: A Distributed Anonymous Information Storage and Retrieval System. Lecture Notes in Computer Science **2009** (2001) 46–??
6. Druschel, P., Rowstron, A.: PAST: A large-scale, persistent peer-to-peer storage utility. In: HotOS VIII, Schloss Elmau Germany. (2001)
7. Stoica, I., Morris, R., Karger, D., Kaashoek, F., Balakrishnan, H.: Chord: A Scalable Peer-To-Peer Lookup Service for Internet Applications. In: Proceedings of the ACM SIGCOMM, San Diego, August 27-31 2001. (2001)
8. Napster: Napster homepage, http://www.napster.com/ (2002)
9. Grokster: Grokster homepage, http://www.grokster.com/ (2002)
10. SETI@Home: SETI@Home homepage, http://setiathome.ssl.berkeley.edu/ (2001)
11. Groove: Introduction to Groove, Groove Networks White Paper (2000)
12. FastTrack: The FastTrack Protocol, http://www.fasttrack.nu/ (2002)
13. Good, G.: LDAP Data Interchange Format, Request for Comments 2849, http://www.ietf.org/rfc/rfc2849.txt (2000)
14. : JXTA for J2ME, http://jxme.jxta.org/ (2002)
15. Knudsen, J.: Getting Started with JXTA for J2ME, http://wireless.java.sun.com/midp/articles/jxme/jxta (2002)

16. Cugola, G., Picco, G.: PeerWare: Core Middleware Support for Peer-To-Peer and Mobile Systems. In: PeerWare: Core Middleware Support for Peer-To-Peer and Mobile Systems. Technical report, Politecnico di Milano, May 2001. (2001)
17. Aberer, K.: P-Grid: A self-organizing access structure for P2P information systems. (2001) http://lsirwww.epfl.ch/publications/CoopIS2001.pdf.