TIERPEER: A THREE-TIER FRAMEWORK FOR P2P APPLICATIONS

Abolhassan Shamsaie[†], Jafar Habibi[††], Fatemeh Ghassemi[†††]

Department of Computer Engineering, Sharif University of Technology, Tehran, Iran

Summary
Peer-to-Peer (P2P) gained a lot of interest these days and variety of P2P applications are increasing too. Among these all applications, P2P frameworks try to simplify the development process. In past, they provided some high level functions and had a static architecture dedicated to a network structure or topology. In present, complex applications have become more interesting. For instance, applications require communicating with numbers of different overlay networks, topologies and underlying networks. An open and flexible architecture would be interesting in such cases. Changes and updates of applications make them difficult to maintain. Extendible and pluggable architecture can reduce change and update costs which lead to reduce the maintenance costs. Thus, the need to a new generation of P2P frameworks to support these requirements is inevitable. In this paper, we propose a new extendable framework with a flexible architecture. This framework so-called TierPeer is based on three-tier model of client-server paradigm and OSI model of distributed system paradigm.

Key words:
Peer-to-Peer, P2P Framework, N-Tier Model, OSI Model.

1. Introduction

Since peer-to-peer systems like Napster [12] and Gnutella [7] became so popular, an increasing interest in peer-to-peer systems has been ignited. Numerous business projects and academic searches in this context have been done and an explosion in theories and papers concerning distributed searching algorithms, load balancing algorithms, and garnering statistical data on peer-to-peer systems have been published. The trend in this field is creating a framework that should be able to help programmers to implement and test various components of P2P networks.

Different types of applications, topologies, underlying networks, protocols or platforms have been used in P2P applications. The term topology in P2P computing refers to the structure of the overlaying P2P network. The topology of a P2P application significantly influences the performance, search efficiency and functionality, and scalability of a system. Topologies have some advantages and disadvantages and each application has a fixed topology that disables it to work with other topologies. Underlying networks like ad-hoc network or Internet have different virtual machine libraries and OS primitives, which force an application to be dedicated to its network. Thus, a more flexible and adaptable framework is needed.

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

A difficult decision for system designers is selecting a P2P architecture that fulfils all requirements of an application and best fits its topology. So a framework with an open architecture which is flexible and adaptable can help designers to confide from fulfilling of all topological requirements.

Most of current frameworks have a per-defined topology and programmers should use this fix topology in their applications. Some others that have open architecture are difficult to use and almost always professional programmers use them. Thus openness and simplicity can encourage researchers to use a framework to test their ideas instead of implementing all components of a P2P application which wastes a lot of time. For instance, we may require updating or testing a component with help of an existing application, so it will be interested to plug this component to application and use it without any modification in application source code.

In this paper, we introduce a flexible and adaptable framework architecture for P2P applications. In this context, we call our framework TierPeer. The TierPeer has an open and lightweight architecture which can be used to implement any P2P component and application.

TierPeer has a pluggable architecture that allows different topology descriptions to be plugged, based on change requirements of an application. The main advantage of TierPeer is providing an interface to the application layer and allows the underlying topology to be changed without making any code modification in the application.

Manuscript received February 5, 2007
Manuscript revised February 25, 2007

This paper is structured as follows: Section 2 gives an overview of P2P topologies. Section 3 presents related work. Section 4 discusses TierPeer. Section 5 explains about implementation. Section 6 evaluates the framework. Section 7 explains our future plans and Section 8 concludes the paper.

2 P2P Topologies

This section provides an overview on the most common P2P topologies. The network structure characteristic and the degree of centralization aim at looking at systems from the topological perspective. Two levels of structuring are identified: unstructured and structured. In an unstructured topology, an overlay network is realized with a random connectivity graph. In a structured topology (e.g. DHT based topologies like CAN [5], CHORD [8], PASTRY [6], and TAPESTRY [14], the overlay network has a certain predetermined structure such as a ring or a mesh.

The degree of centralization means to what extent, the set of peers depends on one or more servers to facilitate the interaction among them. Three degrees are identified: fully decentralized, partially decentralized and hybrid decentralized.

In the fully decentralized case (Gnutella), all peers are of equal functionality and none of them is important to the network more than any other peer. The relevant search algorithms are blind and flooding based, that result in reduction of the scalability.

In the partially decentralized case (KAZAA [11]), a subset of nodes can play more important roles than others, e.g. by maintaining more information about their neighbour peers and thus acting as bigger directories that can improve the performance of a search process. This set of relatively more important peers can drastically vary in size, while the system remains to be functioning.

In the hybrid decentralized case, the whole system depends on one or very few irreplaceable nodes which provide a special functionality in one aspect such as a directory service. However, all other nodes in the system, while depending on one special node, are of equal functionality and they autonomously offer services to one another in a different aspect such as storage. Thus, a system of this class is a hybrid system that is centralized in one aspect and decentralized in another aspect.

3 Related Work

Many P2P systems exist (e.g. Gnutella, Freenet [2], Chord, PASTRY, JXTA [10], XMIDDLE [15], Groove [19] and etc.). However, only a small subset of these P2P systems can be used as framework for other applications. The most closely related project to the TierPeer framework is the JXTA project.

JXTA attempts to provide a set of tools on which peer-to-peer networks and applications can be constructed. However, after working with JXTA, it was realized that a new simpler and more accessible set of tools is needed, which motivate us in creation of the TierPeer framework.

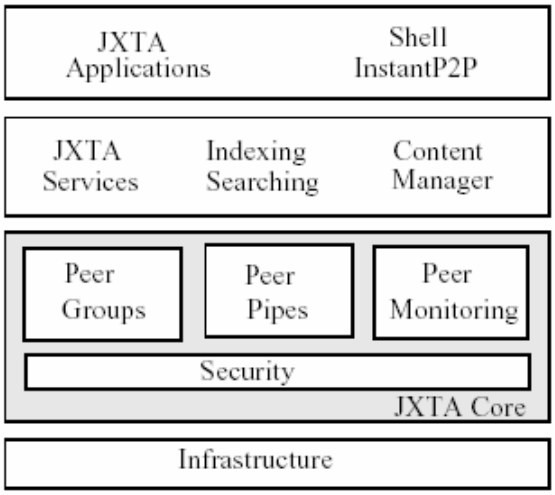

Fig. 1 The JXTA architecture

Figure 1 shows the architecture of JXTA on a conceptual level. It can be broken down in three layers: the core layer (which is mainly responsible for establishing the P2P network structure, the communication, etc.), the services layer (which provides P2P functionality such as indexing, searching, and file sharing) and the applications layer (where the P2P application resides).

One major limitation of JXTA is that the applications based on it, are required to use a predefined topology. Another limitation is the complexity of the JXTA components makes it more difficult to be extended. Furthermore, JXTA requires every peer to parse and generate XML messages. This is a serious problem on computing devices with resource limitations such as mobile phones (e.g. J2ME-enabled phones) and Personal Digital Assistants (PDAs). In [4], a short overview of the main problems of using JXTA [9] with J2ME has been illustrated. We have implemented TierPeer using Java and the size of the total implementation is less than one of the protocols of JXTA. TierPeer has a lightweight architecture and enables the applications, which use it, to run faster.

Another P2P middleware which is for mobile hosts is XMIDDLE. It does not assume the existence of a fixed network or a central authority. It connects peer directly, peers are not used to forward messages to other peers. In XMIDDLE, each peer organizes its content in a tree structure (i.e., XML). It provides primitives for operating

on these trees and to share branches of them. If a peer wants to access the content of another peer, it connects to one of the shared branches. This is comparable to the mounting of a remote file system share. If a remote tree branch has been "mounted", the peer can read and manipulate the data offline. XMIDDLE provides the mechanisms for reconciliation when the mounting peer and the owner of the branch are again connected. If a peer wants to modify an alien branch and has a connection to the owner of that branch, it requests the owner to perform the modification. After that, the owner informs all connected peers about this change. The modification itself is represented by using XMLTreeDiff [16]. For specifying the links between peers, XMIDDLE uses standard XML techniques, namely XLink [17] and XPath [18].

XMIDDLE provides the primitives Connect, Disconnect, Link, and Unlink. It does not provide a functionality for searching content in the overall data tree.

Groove is another framework which is a collaboration software based on the principle of a shared workspace, where all members of a group (i.e., those in the workspace) share the same view. Groove provides servers that are used to detect new peers in the network and to help peers with lower bandwidth to distribute changes. These servers are also used to store content if one or more peers are offline (or not reachable – possibly due to a firewall) and therefore cannot see the changes made at that time. It provides higher level services, such as distributed searches, workflows, offline working, and much more. Groove is targeted at small workgroups, as the communication takes place between all peers. With the increasing number of workgroup members, the communication overhead increases tremendously. Hence, it does not scale very well. At the time of writing, Groove is only available for the Windows platform.

Several other P2P systems that can be used as framework are Pastry, PeerWare [3] and P-Grid [1]. Each of these P2P systems has 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 within an application. Thus, in order to have a simple, lightweight and easy to understand framework, we have designed the TierPeer framework.

4 The TierPeer

Our goal of designing a framework can be broken to the following requirements:

- Simplicity: The framework must be easy to use for programmers. A simple and familiar architecture can be interested.
- Flexibility: To achieve true flexibility, the framework must be platform-independent and topology-independent.
- Maintainability: A well Modulated and layered framework can reduce updating and maintenance cost.
- Extendibility: Pluggable framework enables applications to be extended when needed.
- Testability: Test a new component in distributed environment wouldn't be easy and a testable framework can ease debugging of new components.

We have studied some well known architecture of distributed systems that fulfil some requirements, and use their ideas in our framework to fulfil our requirements. In section 4.1 and 4.2, we explain them and their ideas.

4.1 Three-Tier Architecture

One of the reasons that the client-server paradigm is still widely used on the Internet is the simplicity of the three-tier (figure 2.a) model that made it very popular. Three-tier model is the most well known n-tier model that partitions the work into the three parts:

- The Presentation Tier, which is responsible for displaying information.
- The Business Rules Tier, which is responsible for processing of data.
- The Data Access Tier, which supports back-end services such as databases.

Each of these parts could run on different machines, providing increased flexibility and scalability. Usually, the first tier runs on a client machine and the others run on the server machines. In client-server paradigm, there are multi clients and a single server node, but in P2P paradigm, which is the next trend in distributed system, every node can be both client and server and so called peer. Thus we can also use the three-tier model in the P2P paradigm, and have all tiers in one machine or peer in order to act as both client and server.

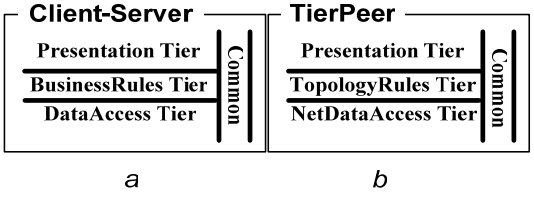

Fig. 2 Three-tier model

We renamed these tiers in order to be more significant in P2P paradigm (figure 2.b) and make this model a basis for our architecture. In this basic model, the Presentation Tier is consisted of application specific modules (e.g. UI modules), the Topology Rules Tier is consisted of modules which are responsible for controlling the topology and routing on overlay networks, and the NetData Access Tier is consisted of modules which support some services such as logging, security services and etc, and connect peers on the edge of overlay networks, independent of underlying platform.

One of the reasons that we have chosen this model as a basic model for our architecture is its simplicity, which is one of the important requirements that we considered in our architecture design. In this basic model, a peer can be a client (request only) and use the Presentation Tier to initiate a request and NetData Access Tier to send requests to and receive replies from its servers (local or remote). It can be a server (serve only) and use Topology Rules Tier to reply or route the requests and NetData Access Tier to send (route) requests to other servers and replies to its clients (local or remote) or receive replies from other servers or requests from its clients. It also can be both client and sever and use all three tiers to act as a client and server independently.

These behaviors of P2P networks are similar to a multi client multi server paradigm that is an extension to multi client single server paradigm. Thus we extended the three-tier model in multi client single server paradigm to adapt to the multi client multi server paradigm. In this way we chose another model to extend our basic model. This model should satisfy the requirements of multi client multi server paradigm. Our chosen model was OSI which is discussed in next section.

4.2 OSI Model

There is a lack of standardization in the P2P area. The International Organization for Standardization (ISO) created the open 7-layer OSI model and made it standard for distributed systems in 1979. This standard, which helps to provide interoperability between different protocols and makes the protocols independent from the underlying devices, is clearly missing in the P2P area.

The main advantages of using a layered model such as the ISO OSI model are:

- Layering helps to identify and understand separate pieces of a complex system.
- The communication between different systems does not require changes to the underlying hardware or software.
- Maintenance and updating is easier due to its modularization.

Thus we chose a layered architecture based on ISO OSI model to extend our basic model and fulfil requirements of a multi client multi server area as mentioned in section 4.1. In this context we viewed a multi client multi server paradigm as a distributed system paradigm and so, we used its ideas. In section 4.3, we design the TierPeer framework according to these ideas.

A three-tier and layered architecture brings lots of features that fulfil our principal requirements such as simplicity, flexibility, maintainability and etc.

4.3 TierPeer Architecture

Our framework is divided to three tiers which form our basic model. This basic model enables users to identify and understand the concept of each tier, according to client-server architecture concepts.

In order to use OSI model in our framework, we divided each tier to some layers, Application Layer, Façade Layer, Network Layer, Data Control Layer, and Transport Layer. Figure 3 shows a high-level abstraction of these tiers and layers.

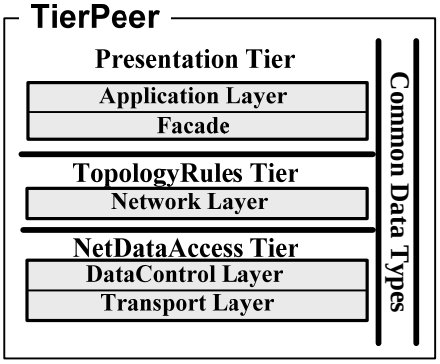

Fig. 3 Layered architecture of TierPeer

4.3.1 Transport Layer

The Transport Layer (figure 4) is one of the layers of NetData Access Tier which is above the underling network.

The Transport Layer (which can be compared to the physical layer of the OSI model) is responsible for the actual sending and receiving of point-to-point messages to and from peers across the available physical network (e.g., the Internet). It consists of Transport Components, a Transport Core and a Transport Interface.

Almost every P2P system currently available has only been built for working in the Internet's IP network (i.e., they are using plain TCP/IP or higher-level protocols such as HTTP or TLS, which are all based on the IP network). TierPeer takes a further step from IP networks and

considers other network types as well (e.g., Bluetooth) to support heterogeneous platforms and protocols.

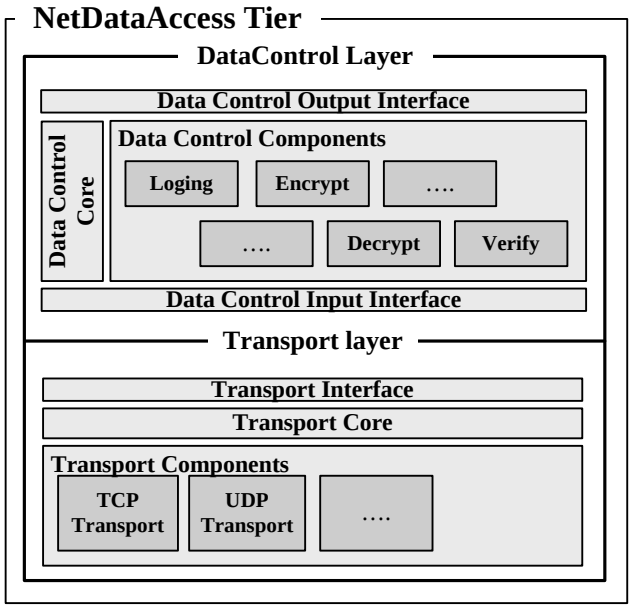

Fig. 4 NetData Access Tier

Each network has its own Transport Component which is consisted of some modules to communicate between peers. Each Transport Component should implements at least one module inherited from BaseTransportModule interface:

```
public interface BaseTransportModule
{
  boolean isResponsibleFor(Contact c);
  Connection newConnection (Contact c);
  void startListener(Contact c);
}
```

Fig. 5 BaseTransport Module

These Transport Components could easily be plugged to the framework and be used in P2P applications. The Transport Components allow multiple protocol structures to be used and make network transparent to higher layers.

The Transport Core is responsible for organizing the Transport Components and dispatching messages between them and Transport Interface. If a message has been received by a Transport Component, it is passed to the Transport Core, which is responsible to deliver it to the Transport Interface. If the Transport Interface gets a message from higher layer, it is passed to the Transport Core which determines the Transport Component which the message should be sent (this information is stored in the message or in the system configuration).

The Transport Interface is an interface between higher layer and the Transport Layer. Thus the higher layer is independent of the Transport Layer and any change in the Transport Layer components would be transparent to the higher layer. The Java interface of TransportInterface is:

```
public interface TransportInterface
{
  void sendNetData(NetData nd);
  void recieveNetData(NetData nd);
  void addListener(Contact c);
}
```

Fig. 6 Transport Interface

4.3.2 Data Control Layer

The Data Control Layer (figure 4) is one of the layers of the NetData Access Tier which is above the Transport Layer.

The Data Control Layer (an extension of the OSI data link layer) deals with the control of incoming and outgoing messages. It consists of two interfaces, one for the outgoing and one for the incoming messages. They make the higher and lower layers transparent of updates or changes in this layer. The Data Control Core is another component of this layer which is consisted of two lists of Data Control Components, each acting upon a received or to-be-sent message. The Data Control Components provide basic services like, logging or encryption/decryption of messages. The main task of this layer is to provide common services for all types of P2P systems. We decided to introduce pluggable Data Control Components that can process incoming and outgoing messages to change the behavior of the system when needed.

Each Data Control Component has some modules that at least one of them should implement following Java interface to be pluggable:

```
public interface BaseControlModule
{
  void setNextModuleIndex(int indx);
  void goToNextModule(NetData nd);
  void stopContinuing();
  boolean isResponsibleFor(NetData nd);
  NetData process(NetData nd);
}
```

Fig. 7 BaseControl Module

The important aspect in this context is that these pluggable components are in a layer between the Transport Layer and the Network Layer. Hence, plugins in this layer operate on point-to-point messages between two peers. This makes it possible to introduce services that are necessary for all P2P topologies (e.g., encryption, error correction, logging, etc.). In JXTA, there exists no such pluggable layer between its communication layer (i.e. Messenger) and the routing layer.

The Data Control Core organizes the Data Control Components that can process a message in five ways. It can ignore the message, change the content of it, do silent processing, send another message or discard it.

4.3.3 Network Layer

The Network Layer (figure 8) is the only layer of Topology Rules Tier which is above the NetData Access Layer.

The Network Layer (which can be compared to the network layer of the OSI model) consists mainly of Network Components, which are coordinated by the Network Core. It is the Network Components that contain the most important part of the P2P applications i.e. routing.

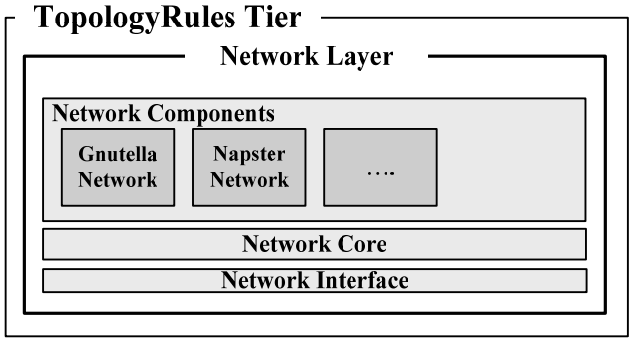

Fig. 8 Topology Rules Tier

A Network Component implements the routing within the P2P network. Depending on the topology of the network, different routing algorithms are required. These Components process all messages that concerning to structure or topology of overlay network. Peers in P2P systems have more autonomy and are self-organizing. They must themselves manage their connectivity to network to shape the topology and support the correctness of P2P application systems. Peer discovery is another problem that should be considered in this layer. Thus all solutions concerning this context for each topology are gathered in one Network Component.

Network Core supports multiple topologies and P2P overlay networks at the same time and sends the received messages from underling layers to appropriate Network Component. It is noticeable that these Network Components are pluggable too. They plug to the framework when one of its modules implements following Java interface:

```
public Interface BaseNetworkModule
{
  boolean isResponsibleFor(Message m);
  void handle(Message m);
}
```

Fig. 9 BaseNetwork Module

The last component of the Network Layer is Network Interface which causes to simplify interaction between this layer and the other layers. It also leads to make future changes and updates transparent to other layers. In general this layer allows more than one application use a topology or an application uses more than one topology (e.g. bridges two topologies).

4.3.4 Facade

The Facade (figure 10) is one of the layers of the Presentation Tier which is above the Network Layer.

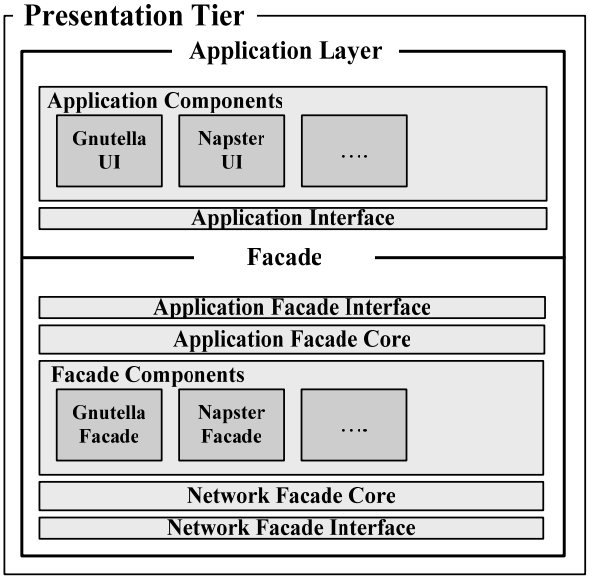

Fig. 10 Presentation Tier

The Facade Layer (which can be compared to the transport layer of the OSI model) consists of Facade

Components, which are coordinated by the Application Facade Core or Network Facade Core. The Facade Layer allows the Application Layer (figure 10) to access the services provided by a topology-independent interface called Application Facade Interface. In fact this layer separates those parts of application which are dependent to their topology and gathers them in this layer. Thus each application can create some topology-independent command and send it to the Facade Layer by Application Facade Interface.

A message from Application Layer to Facade layer is called command. Each application needs some commands to interact with Facade layer. For each command there is a command handler module which must be implemented by programmers. Thus each application has its Facade Component which consists of command handler modules corresponding to its commands. This command handler modules deal with topologies and so any change in topologies will be transparent to the Application Tier. The Java interface of the Application Façade Interface is:

```
public interface AppFacadeInterface
{
  void ExecuteCommand(Command c);
}
```

Fig. 11 Application Facade Interface

This enables the Application Layer to be dependent of implementation of the underlying networks and their changes. Each command handler module is responsible for a command which is sent by one of the above applications and knows how to fulfil it. Thus if some changes happened in the underlying layers, they won't influence the Application Layer and only some of command handler modules of the Facade Layer should be changed to fulfil its relevant task according to new changes. All command handler modules are coordinated by Application Facade Core.

The Network Facade Interface is responsible for receiving and processing messages arrived from underlying layers. Thus the Facade Component of each application consists of some message handler modules too, which are responsible to process their corresponding received messages. All these message handler modules are coordinated by Network Facade Core.

All command and message handler modules are pluggable too and so it enables us to change the topology of an application without any modification in existing components.

4.3.5 Application Layer

The Application Layer is one of the layers of Presentation Tier which is above the Facade Layer.

The Application Layer (a combination of the session layer, presentation layer and application layer) consists of Application Components, which must be developed by programmers. These components are dedicated to an application and they can vary from one application to another.

Each user request is encapsulated in a command and sent to Facade Layer by Application Interface. Thus if a new different request is needed, a new command and command handler module must be added to its Façade Component in Facade Layer and then be used in the Application Layer.

In general Facade Layer enables the Application Layer of P2P applications to be reusable.

4.3.6 Common

All common data types are in Common (figure 12) component which could be used by any layer. This component is similar to Common component of three-tier model which is shown in figure 2.

There is a component called Enterprise in the Common component which is consists of some base classes. The others are some entity types like TCP Contact, UDP Contact, Bluetooth Contact or etc. Each application may use one of these contacts to communicate to other peers. The types of these contacts enable the Transport Layer to determine appropriate Transport Component to send the requests.

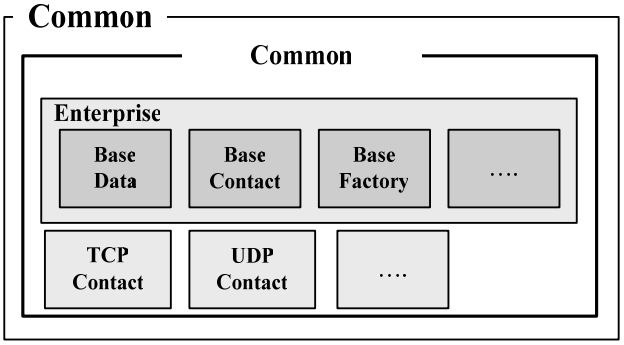

Fig. 12 Common components

4.4 Testability

One of the major benefits of this architecture is that, testing of new components in real world will be simpler.

For example, assume that we want to test a new topology component with an existing application, so it could be done in following manner:

- Implement the Network Components
- Plug it to the Network Layer
- Implement the New Facade Component to allow the application to use the new topology
- Plug it to the Facade Layer

There is no need to modify or change any other layer and so the existing layers are reusable. Thus researchers can concentrate on their research fields and less waste their time. Furthermore the pluggable architecture of this framework could help us to debug a new P2P application.

One of the major problems of P2P framework is testing and debugging of new applications. To test a new application, numerous of computers or devices are needed which almost is impossible to provide. Thus a virtual Transport Component could be plugged which let us to run lots of instances of an application in one or numbers of computers or devices. The problems of using a virtual Transport Components are in applications which use their system storage or system registry to modify, so a virtual machine would be a better choice to test these applications. However they are not often lightweight, so it restricts the number of instances in a computer. Thus a feather weight virtual machine is needed which is one of the research area these days [13].

5 Implementation

We have built a prototype of the TierPeer framework. It has been tested on PCs and notebooks running Java 1.4.1 and does not require additional libraries to run.

We have implemented two applications: a chat application and a file sharing application. The chat application uses a pure P2P topology which targets PCs in a local environment (e.g. LAN). The file sharing application uses a server-base topology like Napster and its target is Internet. Thus a TCP Transport Component and a UDP Transport Component in Transport Layer, a default Data Control Component in Data Control Layer and two Network and Application Components that mentioned above have been implemented and plugged to TierPeer.

We have implemented a virtual TCP Transport Component and a virtual UDP Transport Component too, to enable us to test these two applications.

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

6 Evaluation of TierPeer

As mentioned in section 4 our goal of designing a new framework is following requirements, which we tried to fulfil them:

- Simplicity: The well-known concepts of three-tier and seven-layer models simplify learning and usage of TierPeer model.
- Flexibility: Our architecture is based on OSI model which is an open model and so it's independent of topologies and platforms.
- Maintainability: Three-tier and five-layer architecture of our framework which is well modulated reduces maintenance cost.
- Extendibility: TierPeer is pluggable framework and so it can be extended without any modification in existing components.
- Testability: Pluggable architecture of TierPeer gives us the opportunity to add virtual Transport Component to Transport layer which enable us to test new components.

We although compare TierPeer to some other P2P frameworks in a table 1.

Table 1: Comparison of Frameworks

	XMIDDLE	Groove	JXTA	TierPeer
P2P Framework	No	Yes	Yes	Yes
Support for small device	No	No	No	Yes
Topology independent	No	No	Yes	Yes
Server-less	Yes	Yes	Yes	Yes
Platform Independent	Yes	No	Yes	Yes
Scalable	Yes	No	Yes	Yes
Network protocol independent	Yes	No	Yes	Yes
Support for testing	No	No	No	Yes

7 Future Work

A security module to encrypt and decrypt messages before they are processed by other modules is a candidate future works. It tries to create a so-called web of trust to authenticate peers in a P2P network before any communication. The main challenge of this aim in a P2P

environment is that there is no central authority to check the credentials of a remote peer.

A logging module which logs the messages is one of other future works. It enables us to evaluate different applications or algorithms.

We are planning to create additional topology modules. This is definitely a process over time because it is very likely that new P2P topologies will emerge frequently.

8 Conclusion

P2P is an idea, not a system. Many P2P systems with different topologies evolved in the recent past. The P2P paradigm is especially useful in architectures where scalability and configuration flexibility issues are important and distributed search support is needed.

In this paper, we have shown that the topology of a P2P network has a great deal of impact on its usability, scalability and etc. It is safe to assume that other P2P topologies will emerge in the future. To cope with the challenges that come with the increasing usage of P2P technologies, it is necessary to adapt the P2P topologies to the respective use cases. The TierPeer provides an abstraction from the underlying P2P topology and so it allows programmer to always use the same Interface and the same services provided by the P2P network without depending on a single topology or the device it is running on.

We proposed a three-tier and five-layer architecture that is loosely based on the client-server and ISO OSI models. TierPeer defines the corresponding components of the four lowest layers of the OSI model: physical, data link, network and transport layer. The upper layers are combined in the application layer.

TierPeer has a pluggable architecture and so there are lots of opportunities to extend the framework. For example some new Transport Components of other networks could be implemented and plugged. There are different opportunities for extension of the Data Control Components of Data Control Layer.

Thus with the rapidly increasing interest in peer-to-peer technologies the TierPeer Framework provides a useful tool in the development and testing of existing and new P2P algorithms and applications.

REFERENCES

[1] K. Aberer, “P-Grid: A self-organizing access structure for P2P information systems”.Sixth International Conference on Cooperative Information Systems (CoopIS 2001), Lecture Notes in Computer Science, 2172:179–194.

[2] I. Clarke, O. Sandberg, B. Wiley and T. W. Hong, “Freenet: A distributed anonymous information storage and retrieval system”. Lecture Notes in Computer Science, 2009:46–49, 2001.

[3] G. Cugola and G. Picco, “Peerware: Core middleware support for peer-to-peer and mobile systems”, Technical report, Politecnico di Milano, May 2001.

[4] J. Knudsen, “Getting started with jxta for j2me”, http://developers.sun.com/techtopics/mobility/midp/articles/jxme/.

[5] S. Ratnasamy, P. Francis, M. Handley, R. Karp and S. Schenker, “A scalable content-addressable network”. In SIGCOMM '01: Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications, pages 161–172, New York, NY, USA. ACM Press.

[6] A. Rowstron and P. Druschel, “Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer”, systems. Lecture Notes in Computer Science, 2218:329–351, 2001.

[7] A. Sotira, “What is gnutella?”, http://www.gnutella.com/news/4210.

[8] I. Stoica, R. Morris, D. Karger, F. Kaashoek and H. Balakrishnan, “Chord: A scalable Peer-To-Peer lookup service for internet applications”. InProceedings of the 2001 ACM SIGCOMM Conference, pages 149–160.

[9] jxme Project home, http://jxme.jxta.org.

[10] JXTA, http://www.jxta.org.

[11] Kazaa, http://www.kazaa.com.

[12] Napster, http://www.napster.com.

[13] Y. Yu, F. Guo, S. Nanda, , L. C. Lam and T. C. Chiueh, “A feather-weight virtual machine for windows Applications”,InVEE '06: Proceedings of the 2nd international conference on Virtual execution environments, pages 24–34, New York, NY, USA. ACM Press.

[14] B. Y. Zhao, J. D. Kubiatowicz and A. D. Joseph, “Tapestry: An infrastructure for fault-tolerant wide-area location and routing”. Technical Report UCB/CSD-01-1141, UC Berkeley, 2001.

[15] C. Mascolo, L. Capra, and W. Emmerich. "An XML-based Middleware for Peer-to-Peer Computing", In IEEE International Conference on Peer-to-Peer Computing, Use of Computers at the Edge of Networks (P2P, Grid, Clusters), 2001.

[16] IBM alphaWorks. XML TreeDiff http://www.alphaworks.ibm.com/tech/xmltreediff, 1999.

[17] S. DeRose, E. Maler, and D. Orchard. "XML Linking Language (XLink), 1.0". Technical report, World Wide Web Consortium, Jun 2001.

[18] J. Clark and S. DeRose. "XML Path Language (XPath) ", Version 1.0, W3C Recommendation, http://www.w3.org/TR/xpath, 1999.

[19] Groove Networks, http://www.nfcr.org/, 2006

Abolhassan Shamsaie received his B.Sc. degree in Software Engineering from University of Tehran, Iran in 2004. He is currently an M.Sc. student in Sharif University of Technology, Tehran, Iran. His research interests are Performance Evaluation, Simulation, System Analysis & Design and Distributed Systems.

Jafar Habibi received his PhD in Computer Science from The University of Manchester, UK in 1998. He joined the Department of Computer Engineering, Sharif University of Technology, Tehran, Iran, as an assistant professor. His research interests are Software Engineering, Information Systems, Simulation, System Analysis & Design and Distributed Systems.

Fatemeh Ghassemi received her B.Sc. degree in Software Engineering from University of Tehran, Iran in 2004 and M.Sc from University of Isfahan. She is currently an PHD student in Sharif University of Technology, Tehran, Iran. Her research interests are Verification, Semantic of Programming Languages And the Refinement Calculus.