Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

A Framework for Peer-to-Peer Service Interaction

Jun Li
jli@student.ethz.ch

Dept. of Computer Science
Swiss Federal Institute of Technology (ETH) Zurich
Winter Semester 06/07

Prof. Dr. Roger Wattenhofer
Distributed Computing Group

Advisors: Thomas Locher, Stefan Schmid

A Framework for Peer-to-Peer Service Interaction

Jun Li
jli@student.ethz.ch

Abstract

Although structured/unstructured peer-to-peer (P2P) algorithms proliferate in recent years, building P2P application is still complex and time-consuming due to inadequate infrastructure support. On the other hand, service-oriented architecture significantly reduces developing complexity by further decoupling dependent components. Based on the idea of using services as fundamental elements for developing P2P applications, we propose a P2P service framework to facilitate P2P application development. The framework chooses eQuus as the DHT implementation for the reason of robustness and integrates with OSGi to maximize its usability. Group communication helps the framework to achieve replication with almost free cost. Pull and push based service discovery mechanism enables applications unaware of the underlying remote distribution. Through relaying mechanism, our framework is able to traverse NATs. Based on our framework, we also implemented QChat - a pure P2P chat application.

1 Introduction

Peer-to-peer (P2P) computing abandons the notion of clients or servers but only has equal peers that simultaneously function as both "clients"and "servers"to the other peers on the network. Although structured [34] or unstructured [38] P2P algorithms proliferate in recent years, building P2P application is still complex and time-consuming due to inadequate infrastructure support. On the other hand, service-oriented architecture [26] significantly decouples applications by the service based interaction model in which service descriptions can be published to the registry, service components can be discovered by matching service descriptions with service requirements and then be integrated into the system at runtime. Using services as fundamental elements for developing P2P applications yields a promising solution to this problem.

Unfortunately, designing a framework which seamlessly integrates the two concepts is non-trivial - design decisions are usually made from diverse or even contradictory requirements. First, what kind of P2P algorithm should be used in order to consider application level needs such as replication and locality. Second, how to provide a P2P service interface on top of current service-oriented platforms is a real challenging problem. On the one side, the interface need be compatible with some standard platforms to maximize usability. On the other side, the service clients are able to binding P2P service implementations without noticing their distribution. Furthermore, some practical issues like NATs should be taken into account as early as possible while designing the framework. It is difficult for two peers on different private networks to contact each other directly, however, which is often important for the P2P applications. Thus, the framework is required to have the ability to traverse NATs.

Targeting at reducing the complexity of developing and deploying P2P applications, we propose a peer-to-peer service framework which extends the service-oriented platform with the ability of P2P interaction. The contribution of the framework includes:

- Implementing eQuus [22] - a novel distributed hash table (DHT) suitable for highly dynamic environments and integrating it with OSGi [3] platform - a service-oriented component architecture [7]. eQuus is formulated by standard protocols whose function is abstracted in the form of OSGi services.

- Introducing group communication into P2P system. Since eQuus is particularly designed to handle network churn by clique organization, data replication is obtained for almost free through group communication primitives.
- Supporting both pull based and push based service discovery transparently. Pull/push are two strategies to design the collaboration in the distributed environment. Based on keyword partition and dynamic proxy generation, the framework hides the remote service distribution for P2P applications.
- Using relaying technology to achieve service invocation across NATs. Private peers search for other peers with globally valid IP addresses who would delegate them to publish their services. The connections between members and cliques are reused for relaying in the later service interaction.

The rest of this paper is organized in the following way: We analyze several design challenges and solutions in Section 2. Section 3 describes the architecture of our framework including eQuus design, OSGi service extension and NAT traversal. We demonstrate a P2P chat application - QChat in Section 4. Section 5 describes related work and we conclude in Section 6.

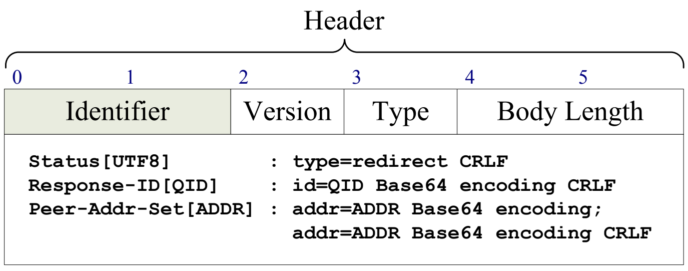
2 Design Principles

The primary challenge of designing a framework for P2P interaction is balancing diverse or even contradictory requirements. The first design decision we should make is whether to choose a binary packet format or an ASCII one. Binary packet format is simple and thus can be parsed efficiently while ASCII packet format is more flexible. Since routing process could be recursive or iterative [5], a trade off need be considered between security and performance. The trend of P2P computing makes every participant computer become a light-weighted server that may handle hundreds or thousands of connections. Two common solutions, thread based or event based design [18], are employed to implement such kind of data intensive applications. Replication is especially important for dynamic network environment under the presence of churn. With the help of group communication [27], replication would be realized for almost free instead of utilizing complicated mechanisms. Last but not least, service discovery is essential for applications built upon our framework. To address this problem, two general retrieval strategies can be divided - *pull* and *push*. How to effectively implement both pull and push strategy in a P2P manner is a real challenge for the framework design.

2.1 Binary vs. ASCII

When data is formatted into a packet, the application can specify packets flexibly as well as the network can transmit longer messages more efficiently and reliably. Typically, a packet consists of two elements: the first element is a header, which marks the beginning of the packet; the second element is the body, which contains the information to be carried in the packet. To meet the flexibility and efficiency requirements, eQuus control packets and application-specific messages are encapsulated differently in binary and ASCII format. Binary format, like some low level communication protocols using octet as transmission unit, focuses on best effort packet processing while ASCII format, like some high-level application protocols regulating contents by BNF [29], focuses on flexible design. The two kinds of formats are used separately in packet header and payload body. An example of a packet is described in Figure 1, and more details are referred to [20].

The format of eQuus packet is transmission protocol agnostic. In case the carrying protocol does not support reliable communication, `Identifier` field indicates the uniqueness of the received packet. The packet body is divided into lines of characters. Each line is composed of a field name and a set of key-value pairs. The interpretation rules such as `UTF8`, `QID` or `ADDR` are specified in



```
Status[UTF8]          : type=redirect CRLF
Response-ID[QID]      : id=QID Base64 encoding CRLF
Peer-Addr-Set[ADDR]   : addr=ADDR Base64 encoding;
                        addr=ADDR Base64 encoding CRLF
```

Figure 1: eQuus Packet Example

the field name part. For the reason of saving space, we choose modified `Base64`[1] to encapsulate binary information. For example, a 128 bit ID needs 128 bytes by naive binary string. However, if represented by `Base64` it only uses 28 bytes. Another advantage of eQuus packet format comparing to RPC based implementations [34] is standard protocol format specification and flexible application message support.

2.2 Recursive vs. Iterative

DHT iterative routing is suggested in [33]: the initial peer begins with looking up its routing table to find the next hop; then it sends a routing request message to the result address. When receiving a request message, a peer would reply with the best route according to its own view. The same procedure is repeated until reaching the destination peer. A concrete example of iterative routing is showed in Figure 2 where the communication pattern is 1 out of N between different cliques.

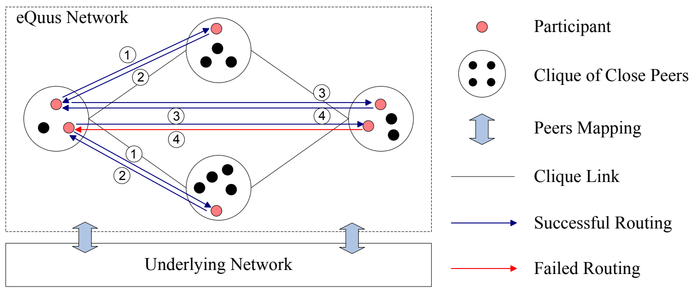


Figure 2: Iterative Routing in eQuus Network

The design decision is made from the trade off that the benefit of security is much more than the sacrifice of performance. Comparing to recursive routing, although iterative routing doubles the cost it is safer for both individual peer and the whole network. Each peer can check the reply result to estimate whether it is reasonable or not. If a malicious peer wants to overload the network by faking numerous meaningless routing packets, it is very hard since each round the misbehaving peer should participate in the communication rather than influences a chain of peers relative to recursive routing. In addition, iterative routing can improve the probability of successful routing due to the possibility of choosing alternative peers. The measurement is illustrated in Figure 3. From the experiment results, we can conclude that iterative routing doesn't incur much more network overhead than recursive routing.

[1]The widely used Base64 encoding is Privacy-enhanced Electronic Mail (PEM) protocol which defines a 64-character alphabet consisting of upper- and lower-case Roman alphabet characters (A-Z, a-z), the numerals (0-9), and the "+" and "/" symbols. The "=" symbol is also used as a special suffix code. Since the "/" and "=" symbol are separators in BNF, they are replaced by the "-" and "#" symbol.

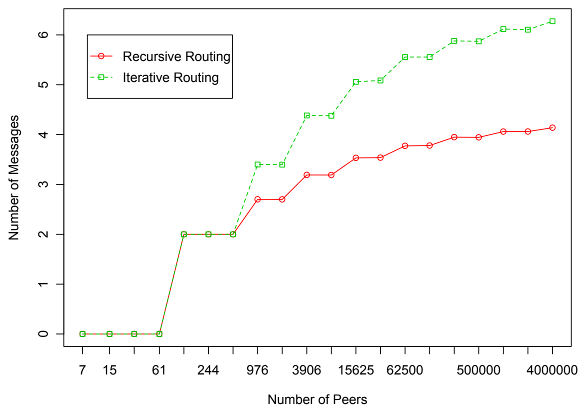


Figure 3: The comparison of recursive routing and iterative routing based on the average number of required messages is displayed. For each number of peers n and for each routing strategy, 10,000 simulations are conducted with the success rate 0.5. We choose 32 as the size of a clique. The base b is 4 and the dimension d of the network is 128 in each run.

2.3 Thread vs. Event

Another issue related to implementing P2P systems is whether to employ a thread based design or an event based design. There is a long debate of both approaches since 1979 when Lauer and Needham pointed out thread-based and event-driven programming are duals of each other [18]. SEDA [39] was an event-driven system and proved that events are a very efficient and scalable model for concurrent programs. As a negative reply, Capriccio [37] made a possible solution for programmers to write high-performance Internet servers using the intuitive one-thread-per-connection style while overcoming the difficulty of tracing cause and effect relationships arising in event-based systems.

Although thread based approach offers a better programming model, it is simple and efficient to implement our framework based on event driven method. A central event scheduler located at the bottom of the framework is used to dispatch different events which can be divided into two categories - time event and network event. Time event is triggered during every fixed period. Network event is related to sending and receiving network messages. To eliminate blocking routings, the framework must utilize asynchronous network I/O through registering callback functions that cope with notifications when some operation is finished.

2.4 Replication vs. Group Communication

Replication is particular useful for P2P applications where the number of total peers is large but the rate of joining and leaving is very high. Many sophisticated methods have been introduced to achieve effective replication and cache consistency in distributed storage systems [9], read/write file systems [24] and service load balancing [8]. Since eQuus is specifically designed to cope with high dynamics and failures and to minimize the probability of data loss by clique organization [22], we can replicate data at a low cost through group communication primitives.

Group communication ensures membership consistency among peers in the same clique. Two approaches, running a consensus protocol among the all previous group members to agree on the future group membership and integrating consensus with the membership protocol and running it only among the correct members, have been considered for practical deployment. The latter one is preferable because of simplicity. Thus, we customize the view-synchronous group communication algorithm introduced in [16, 17] by eliminating retransmission and state recovery mechanisms. It is still correct due to the fact that TCP is used in member communication. The details are described in

Algorithm 1.

Algorithm 1: View-synchronous Reliable Broadcast

```
initialization:;
    /* $P_i$'s sequence number                                    */
    $s \leftarrow 0$;
    /* last v-delivered $P_j$'s sequence number                   */
    $s_j \leftarrow 0 \ \forall j \in [1, n]$;
    $vid \leftarrow 0; view \leftarrow \{P_i\}$ ;
    $new\_vid \leftarrow 0; new\_view \leftarrow \emptyset$ ;
upon v-send(m):;
    send message $(\texttt{send}, vid, s, m)$ to all servers;
    $s \leftarrow s + 1$;
upon receiving $(\texttt{send}, vid, s', m)$ from $P_j$;
    remember $(j, s', m)$ delivered in view $vid$;
    if $(new\_vid = 0)$ or $(new\_vid \neq 0$ and $P_j \in view \cap new\_view)$ then
        v-deliver(m);
        $s_j \leftarrow s'$;
    end
upon v-change(v, V):;
    $new\_vid \leftarrow v; new\_view \leftarrow V$;
    $S \leftarrow \{s_1, \ldots, s_n\}$
    send message $(\texttt{flush}, new\_vid, i, view, S)$ to all;
    block the application;
upon receiving $(\texttt{flush}, v, j, view', S)$ with $v = new\_view$ and $view' = view$ from all
$P_j \in view \cap new\_view$:;
    foreach $P_l \in view$ do
        $t_l \leftarrow$ maximum of the received $s'_l \in S$;
        if $s_l < t_l$ then
            v-deliver all messages from $P_l$ up to $t_l$;
        end
    end
    output v-change(new_vid, new_view);
    $vid \leftarrow new\_vid; view \leftarrow new\_view$;
    $new\_vid \leftarrow 0; new\_view \leftarrow \perp$;
    unblock the application;
```

Another advantage of using view-synchronous group communication in P2P system is that it addresses the problem of communication behind NATs/firewalls. In the same group, some peers may play the role of gateway for others. The epidemic-like [38] property makes it possible to reach hidden peers via indirect links. In addition, combined with reliable (FIFO, causally ordered or atomic) broadcast facilities, group communication can synchronize MERGE and SPLIT operations [22] with little effort.

2.5 Pull vs. Push

A non-trivial aspect of designing a P2P service framework is to efficiently locate services in both pull and push manner instead of naive broadcast approach. Pull means applications actively query required services while push means some agents deliver registration data according to pre-defined conditions. In P2P research field, search engines [30, 36] often use pull approach to retrieve information and event notification systems like FeedTree [32] usually disseminate events based on push approach. Combining the two strategies is necessary because OSGi supports both LDAP [15] query based service discovery and `ServiceTracker` which traces the change of a service.

Sharing the same requirements with OverCite [35], we use keyword partition mechanism to publish service registration data. Each clique stores data for keyword searching on its local disk. The data consist of an inverted index giving a list of service descriptions. Keywords are partitioned among different cliques according to their hash values. This approach derives a double replication mechanism (one for clique replication and another for partition replication) so that discovery latency is reduced by parallel query on multiple cliques and load is balanced among equivalent peers in the same clique.

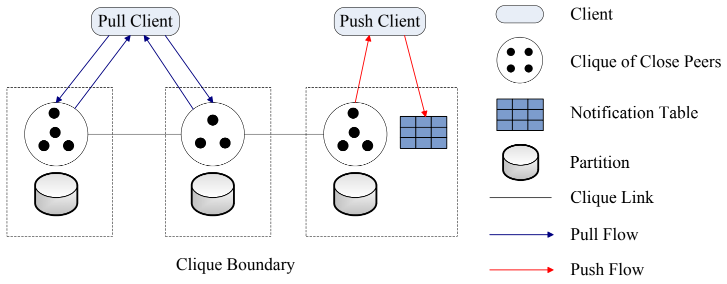


Figure 4: Pull/Push Operation

A typical scenario of pull and push based operation is demonstrated in Figure 4. The pull client first locates potential cliques by parsing LDAP query string into several keywords. Then, it performs keyword queries in parallel and connects results to filter unsatisfied services. The push clique registers trigger conditions first via the same method used in pull scenario. Each responsible clique maintains a table of push clients and delivers the service description when a matched service is stored in its partition.

3 Framework Architecture

Although structured/unstructured P2P algorithms proliferate in recent years, building P2P application is still complex and time-consuming due to inadequate infrastructure support. The trend advocated by service-oriented programming is using services as basic building blocks instead of objects to minimize the coupling that is created by OO. Combining P2P computing and service based platform generates a framework that can efficiently reduce the complexity and time to develop P2P applications and be easily customized to various domains by replacing pluggable components. The architecture of such kind of framework is depicted in Figure 5.

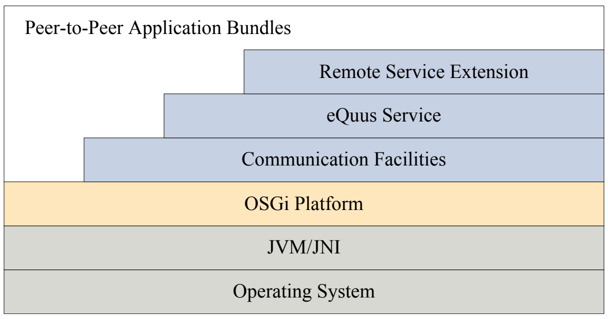


Figure 5: Peer-to-Peer Service Framework

The framework is built on top of OSGi which is a Java platform for developing and deploying extensible and downloadable service applications called bundles. There are two approaches of OSGi collaborative model among bundles. One is package dependency approach and the other is service dependency approach. The bundles are comprised of Java classes and other resources which together can provide functions to end users and provide packages and services to other bundles. Exported and imported packages are described in the bundle manifest file with the package name and the version. When the platform resolves a bundle, it searches exported packages from other bundles to match the bundle's imported packages. A service is defined as a specification (Java interface) and probably has several implementations. Service object conforms to the service-oriented interaction pattern in which a bundle might register a service as well as a collection of key-value pairs describing the service properties. Many different implementations of the same service may be registered by many different bundles and the associated service properties which are static can be used to differentiate among them. When other bundles want to discover a service, the dynamic registry allows the bundles to use a fully qualified service name and an optional LDAP filter over the service properties, and to find and track the appropriate service objects. The platform fully manages the dependencies and security of the collaboration among bundles.

P2P service interaction is enabled by three components - communication facilities, eQuus service and remote service extension. Communication facilities address groupcast as well as application level unicast/multicast. Although the communication facilities are mainly designed for sending and receiving messages among members and cliques in eQuus networks, application bundles can easily reuse them because of the package collaborative model between application and communication bundles. The purpose of eQuus service is to provide a uniform DHT interface to operate eQuus networks. Since the fundamental interaction pattern in OSGi is service-oriented, remote service extension is introduced to support both pull/push based service discovery and service invocation across NAT. Application bundles implementing user logic can utilize all the facilities supplied by the framework to transparently achieve P2P capability. The following paragraphs describe detailed mechanisms in these components.

3.1 eQuus Design

eQuus is the foundation of the framework which is able to support P2P service interaction. Its design determines the framework's properties such as replication ability and locality. eQuus Protocol (QP) [20] formalizes eQuus networks and specifies a reference architecture for implementation. Figure 6 describes the layer organization in eQuus Protocol. Service layer is for eQuus applications which send/receive messages, join eQuus system and lookup specific IDs. Protocol layer defines different entities and their behaviors to implement the provided services and to maintain eQuus network. It also specifies Protocol Data Units (PDUs), routing table and member view. Communication layer defines different types of communication facilities.

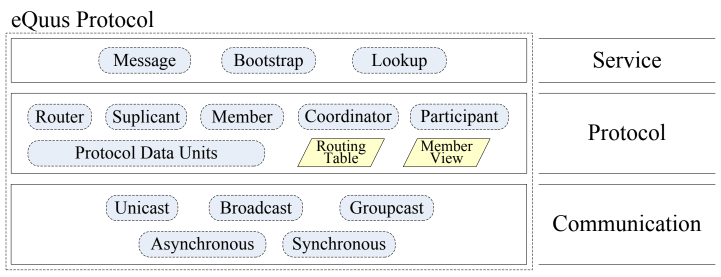


Figure 6: Layers in eQuus Protocol

QP is deployed on computers that need P2P interaction facility. It enables high-level protocols, services or applications communicate with the corresponding parts residing on other computers by logic

IDs. Two primary functions of QP are P2P interaction primitives and autonomous peer management. QP provides interaction primitives in the form of services, including message service which means that given an ID, QP forwards messages hop by hop until reaching the destination peer who is responsible for the ID, bootstrap service which means that controls the host peer to be the member of eQuus overlay network, and lookup service for finding several peers in a clique according to its ID. Autonomous peer management includes clique organization and membership management. Every eQuus clique can handle join request from any individual peer and merge or split clique when the number of members in some clique is too little or too much. Clique organization makes eQuus overlay network fully-decentralized and self-organized. Membership management ensures all members in a clique have a consistent view of their membership.

Both P2P interaction and autonomous peer management function are built upon communication facilities in QP. Instead of specifying data transmission mechanisms such as end-to-end data reliability, flow control and sequencing which the lower TCP/UDP level has already addressed, QP defines high-level communication facilities including unicast, broadcast or groupcast[2] and synchronous/asynchronous communication by wrapping TCP/UDP primitives. Thus, the main purpose of message service which encapsulates functions in communication facilities is for transmitting control messages between two peers based on IDs and not directly for data transmission. Unlike TCP/UDP that differentiates potential receivers with various ports, message service chooses receivers based on message contents.

The primary functionalities of QP depend on different roles defined in the protocol layer. A brief description is listed in the following:

- **Router** A protocol entity performing routing messages and resolving IDs. When receiving a message, router will forward it according to its routing tables. If the destination is the host peer, it will deliver the message to upper layers. It can also resolve ID to a list of members in the corresponding clique.

- **Supplicant** A protocol entity running on a peer that actively controls the peer to join or leave the eQuus network. Since the peer hasn't joined the eQuus network, supplicant can't use communication facilities in eQuus to communicate with other peers. In such case, it must use some mechanisms such as broadcast in IP layer or peer cache to cooperate with some members to initialize its routing tables.

- **Member** A protocol entity running on a peer that passively receives the request of a new peer to join or leave the eQuus network and helps the new peer to be or not to be part of the the eQuus network. When happening peer failure, the members in the same clique should set a threshold to determine the leave of the failure peer by periodically checking its liveness. It is also responsible for distance measurement and keeping membership consistency.

- **Coordinator** A protocol entity playing the role of commander during the course of clique merge and split. A coordinator needs to keep the state of merge and split operations and synchronize among them. To avoid complex distributed election algorithms, the coordinator is chosen with the lowest IP address since all peers in a clique know about each other.

- **Participant** A protocol entity executing the commands from the coordinator during the course of clique merge and split. Most of the time participants passively update their states after receiving a message. In some rare cases, a specific participant is selected to actively carry out operations. For example, the coordinator will shift its responsibility to the peer who is the nearest to the predecessor clique when computing clique partitions in split operation.

[2]The different between broadcast and groupcast is that broadcast does not require consistency while groupcast is a kind of communication among a group of peers in the same clique which must have a consistent view of their membership. Groupcast probably uses view-synchronous broadcast algorithm.

3.2 OSGi Service Extension

As discussed before, both pull and push based service discovery approaches need be supported in order to transparently integrate into OSGi platform. Although keyword partition can help distribute registration data and balance load, a problem arises when a client wants to bind a remote service implementation at runtim. The platform should be able to hide remote service distribution and to handle serialization and communication.

Dynamic Aspect Oriented Programming (dAOP) is capable of runtime adaptation [12] which occurs through the dynamic inclusion of aspects within the code of an application using, e.g., *join points* such as *before*, *after*, or *around* method. These join points cause the dynamically acquired extension to be executed before executing, after executing or instead of, a given method.

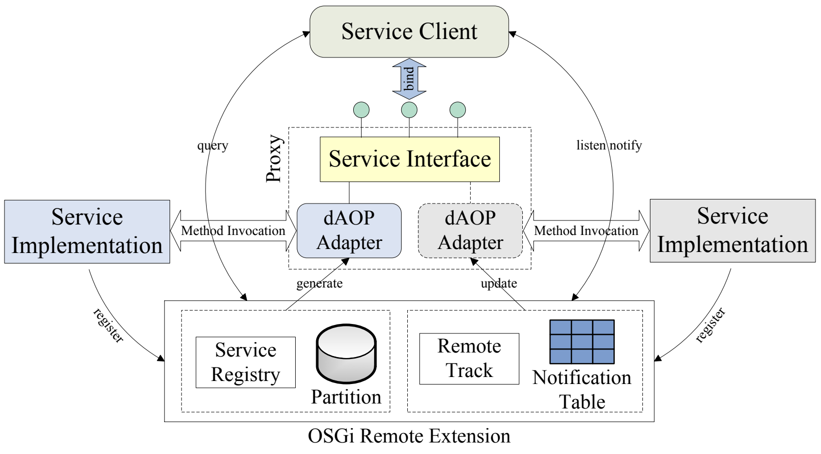


Figure 7: OSGi Peer-to-Peer Service Interaction

The introduction of dAOP makes service clients unaware of the proxy existence and also makes P2P service interaction easy to design and implement on current service systems. Figure 7 shows the approach to enable pull and push based service interaction based on dAOP technology. For a client, the process of using a remote service is the same as that in local environment. A service implementation first publishes itself with some key-value properties on its local registry. The local registry further disseminates the service description to cliques which are responsible for specific keyword partitions. Then, a client queries its local registry for required services from a LDAP string. The local registry also derives the partition information from the LDAP string and communicates with responsible cliques to collect remote service data. If the process is successful, the registry would generate dynamic proxies for discovered services from their descriptions. After that, the client will get the query results and choose some service to bind. A client can also track the life-cycle of a service through `RemoteTrack`. When the `RemoteTrack` detects a change of a interested service, it notifies listened clients via callback functions. If the service has been already registered in the local platform, the `RemoteTrack` updates a new dAOP adapter while keeping the same service reference. This approach simplifies the procedure of binding mutable services.

3.3 Interaction across NAT

Network Address Translation (NAT) is the source of well-known difficulties for P2P communication, since the peers involved may not be reachable at any globally valid IP address [11]. To address this issue, a commonly used traversal technique is known as relaying. Relaying is an inefficient but reliable method which simply makes the communication look like standard client/server interaction except that the message from the client might be relayed several times till it reaches the server. Relaying peers are also called rendezvous servers who are designated for connecting two peers with private IP addresses. As for the two clients, both of them are unaware of the existence of the rendezvous server as if they were communicating with each other directly. In our framework, we use a similar strategy but with a little modification. Because the eQuus implementation establishes a powerful multicast facility which is agnostic to underlying communication protocols, the connections between members and cliques are reused for relaying.

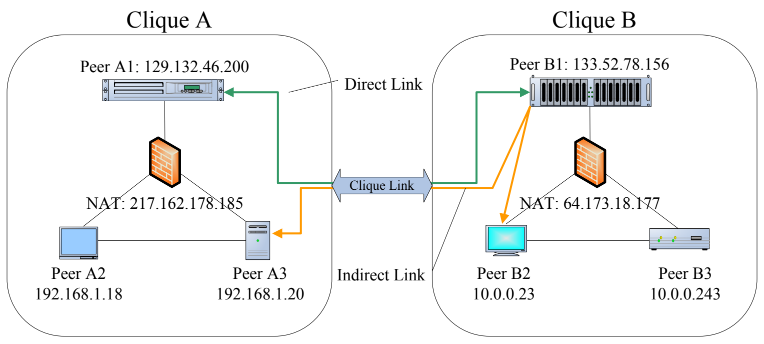


Figure 8: Traverse NAT via Relaying

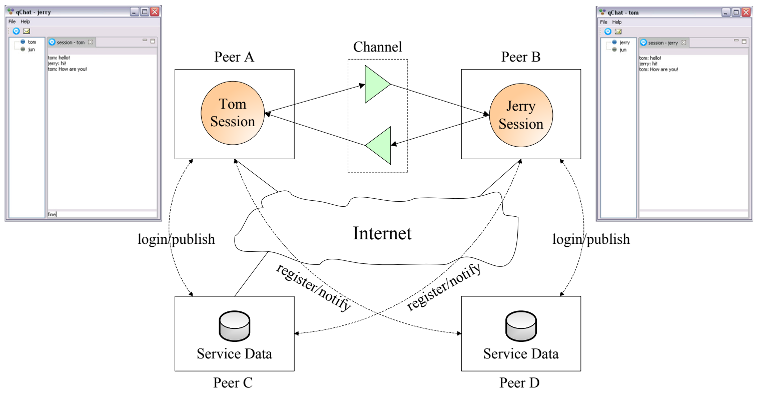
Figure 8 shows a scenario of service interaction across NAT. Suppose two cliques A and B have three members for each. Peer A1 and B1 hold public IP addresses so that they can communicate with each other freely. Other members are hidden behind NATs and only can connect to peers with globally valid IP addresses or in their own private networks. If A1 wants to invoke a service residing on B1, it would form a direct connection. However, it's not the same case when A3 invokes B2's services. Since B2 can't be reached directly, it must find some peer who would delegate it to publish its services. If there are such kind of peers in the same clique, B2 will ask them for service relaying (B1 is the right peer for this scenario). When the first choice doesn't exist, peers will resort to search clique connections to find qualified peers in neighbor cliques.

4 Case Study: QChat

To explore the usability of the framework, we developed a P2P chat application called QChat. Like other chat applications, an end user first logs into the system to get his/her buddy lists as well as their states. To simplify application design, user authentication is eliminated from the login process. If the end user wants to chat with some buddy who is also online, he/she can open a new session and communicate with the buddy through instant message.

The design of QChat is based on service interaction. The communication between chat clients is abstracted as a `Channel` service. The owner of the service receives the messages and the user of the service sends the messages. A service is associated with an end user by a property which contains the user name to identify the service. When two users wants to communicate with each other, two services are used as a dual channel. During a communication session, a `Channel` service has two connected parts residing on corresponding peers. One is for the host GUI application which actually handles incoming messages and another is a dynamic proxy for the buddy to send messages. The proxy is generated by the local service registry who uses `RemoteTrack` to detect the existence of a buddy service and automatically creates a dAOP adapter to communicate with the remote service. If some peers are hidden behind NATs, the interaction process is similar but may incurr several relaying stages.

Figure 9 shows an example of QChat. Suppose user *tom* and *jerry* log into QChat from two separate machines - Peer A and Peer B. Peer C and Peer D happen to be responsible for their service data. After successfully publishing a `Channel` service and retrieving the buddy lists, Peer A uses a LDAP query string "`(buddy=jerry) | (buddy=jun)`"to register buddy service notification. When Peer B publish



tom: hello!
jerry: hi!
tom: How are you!
fine

tom: hello!
jerry: hi!
tom: How are you!

Figure 9: QChat Implementation

the `Channel` service, Peer A is notified and the proxy service is generated. It is the same for Peer B to find Peer A. After *tom* session and *jerry* session are established, the end users can communicate via instant messages. All the logic of QChat is implemented as services so that sending and receiving chat messages are completed by invoking services. All the P2P service publish/discovery is managed by the framework so that remote service interaction is invisible to applications. This separation of concerns strategy hides the complexity of application development. With the help of our framework, it is rather simple to design P2P applications as to design localized ones.

5 Related Work and Comparison

JXTA [2] is an open source P2P platform created by Sun Microsystems in 2001. It is the first try to standardize P2P protocol and framework on top of which various applications [14, 25] could be built. JXTA establishes a virtual network overlay on top of the Internet and non-IP networks, allowing peers to directly interact and self-organize independently of their network connectivity. It also enables application developers, not just network administrators to design network topology that best match their application requirements. Sharing the same idea of JXTA, our framework is also intended to formulate P2P protocols and operations. However, unlike its library based design our framework is embedded with service-oriented architecture which further decouples dependent components in the form of services. In addition, both pull and push based service interaction enables applications transparently integrate with OSGi platform so that service clients are unaware of underlying distribution.

P2P service discovery is able to avoid employing a centralized approach to select available services. SPiDeR [31] supports a variety of Web service discovery operations by organizing service providers into a Chord [34] overlay. Built on this overlay, three different kinds of search operations: keyword-based search, ontology-based search and behavior-based search are provided. CCSD [21] which takes the service context of peers into account makes service-oriented applications capable of context-awareness. It replicates peer's context description in several designated peers and locates a service by forwarding its request to the peer associated with that capacity context. Our framework supports service discovery through LDAP query string in order to match OSGi programming abstraction. Although it's not a powerful discovery mechanism, it can be extended with the ability of ontology-awareness or context-awareness by encapsulating low level LDAP interface with high level ontology/context supported interface.

Another related field is exposing local OSGi services to remote platforms. R-OSGi [28] provides a seamless and noninvasive middleware for accessing remote services in OSGi frameworks. It chooses Service Location Protocol (SLP) [13] to discovery remote services and dynamically generates service proxies to perform remote invocation. Eclipse Communication Framework (ECF) [10] introduces a "remote services"API to expose either a proxy to the client, or allows the client to remotely call methods on a given target either synchronously or asynchronously. It lets the programmer explicitly choose the remote interaction on a service-by-service basis rather than always using a transparent proxy. Comparing to these OSGi remote extensions, our approach has an obvious advantage that the way services are advertised and discovered is completely hidden in the framework implementation. This transparent integration makes applications unconscious of underlying P2P distribution.

6 Conclusion and Future Work

In this paper, we combine both P2P computing and service-oriented architecture and propose a fully decentralized service framework aiming at reducing the complexity of P2P application development. The framework chooses eQuus as the DHT implementation for the reason of robustness and integrates with OSGi to maximize its usability. Group communication helps the framework to achieve replication with almost free cost. Pull and push based service discovery mechanism enables applications unaware of the underlying remote distribution. Through relaying mechanism, our framework is able to traverse NATs. Based on our framework, we also implemented QChat - a pure P2P chat application.

Our future work will include extending communication facilities and developing more P2P applications. During the development, we found that group communication facilities are particular useful for P2P applications such as those ones that need group service invocation. It is valuable to provide an independent service implementation rather than a coupling library customized for eQuus. JGroups [1] is a toolkit for reliable groupcast/multicast communication. The most powerful feature of JGroups is its flexible protocol stack, which allows developers to adapt it to exactly match their application requirements and network characteristics. We will combine our group communication facilities with JGroups to provide an independent OSGi group service bundle. Although P2P applications become more and more popular these years, they are restricted to a small range of applications like file sharing or instant message. P2P Web application [4, 19, 23] is an immature research field due to the difficulty of distributing dynamic Web processing into different peers. We assume with the aid of our service framework a P2P Web system with the ability of dynamic processing can be built.

References

[1] Jgroups – a toolkit for reliable multicast communication. `http://www.jgroups.org`.

[2] The jxta project, `http://www.jxta.org`.

[3] Osgi – open services gateway initiative. `http://www.osgi.org`.

[4] T. BURKARD. Herodotus: A peer-to-peer web archival system, 2002.

[5] Miguel Castro, Peter Druschel, Ayalvadi J. Ganesh, Antony I. T. Rowstron, and Dan S. Wallach. Secure routing for structured peer-to-peer overlay networks. In *OSDI*, 2002.

[6] Miguel Castro and Robbert van Renesse, editors. *Peer-to-Peer Systems IV, 4th International Workshop, IPTPS 2005, Ithaca, NY, USA, February 24-25, 2005, Revised Selected Papers*, volume 3640 of *Lecture Notes in Computer Science*. Springer, 2005.

[7] Humberto Cervantes and Richard S. Hall. Autonomous adaptation to dynamic availability using a service-oriented component model. In *ICSE*, pages 614–623. IEEE Computer Society, 2004.

[8] Tianying Chang and Mustaque Ahamad. Improving service performance through object replication in middleware: A peer-to-peer approach. In Germano Caronni, Nathalie Weiler, Marcel Waldvogel, and Nahid Shahmehri, editors, *Peer-to-Peer Computing*, pages 245–252. IEEE Computer Society, 2005.

[9] Byung-Gon Chun, Frank Dabek, Andreas Haeberlen, Emil Sit, Hakim Weatherspoon, M. Frans Kaashoek, John Kubiatowicz, and Robert Morris. Efficient replica maintenance for distributed storage systems. In *Proceedings of the 3rd Symposium on Networked Systems Design and Implementation (NSDI'06)*, May 2006.

[10] Eclipse.org. Ecf – eclipse communication framework. `http://www.eclipse.org/ecf`.

[11] Bryan Ford, Pyda Srisuresh, and Dan Kegel. Peer-to-peer communication across network address translators. In *USENIX Annual Technical Conference, General Track*, pages 179–192. USENIX, 2005.

[12] Andreas Frei and Gustavo Alonso. A dynamic lightweight platform for ad-hoc infrastructures. In *PerCom*, pages 373–382. IEEE Computer Society, 2005.

[13] E. Guttman, C. Perkins, J. Veizades, and M. Day. Service Location Protocol, Version 2. RFC 2608 (Proposed Standard), June 1999. Updated by RFC 3224.

[14] Emir Halepovic and Ralph Deters. Building a p2p forum system with jxta. In Ross Lee Graham and Nahid Shahmehri, editors, *Peer-to-Peer Computing*, pages 41–48. IEEE Computer Society, 2002.

[15] T. Howes. A String Representation of LDAP Search Filters. RFC 1960 (Proposed Standard), June 1996. Obsoleted by RFC 2254.

[16] Idit Keidar and Roger Khazan. A virtually synchronous group multicast algorithm for wans: Formal approach. *SIAM J. Comput.*, 32(1):78–130, 2002.

[17] Idit Keidar, Jeremy B. Sussman, Keith Marzullo, and Danny Dolev. Moshe: A group membership service for wans. *ACM Trans. Comput. Syst.*, 20(3):191–238, 2002.

[18] Hugh C. Lauer and Roger M. Needham. On the duality of operating system structures. *Operating Systems Review*, 13(2):3–19, 1979.

[19] Jinyang Li, Boon Thau Loo, Joseph M. Hellerstein, M. Frans Kaashoek, David R. Karger, and Robert Morris. On the feasibility of peer-to-peer web indexing and search. In M. Frans Kaashoek and Ion Stoica, editors, *IPTPS*, volume 2735 of *Lecture Notes in Computer Science*, pages 207–215. Springer, 2003.

[20] Jun Li, Thomas Locher, Stefan Schmid, and Roger Wattenhofer. equus protocol specification (version 1.0). Technical report, ETH Zurich, 2007.

[21] Qianhui Althea Liang, Jen-Yao Chung, and Hui Lei. Service discovery in p2p service-oriented environments. In *CEC/EEE*, page 46. IEEE Computer Society, 2006.

[22] Thomas Locher, Stefan Schmid, and Roger Wattenhofer. equus: A provably robust and locality-aware peer-to-peer system. In Alberto Montresor, Adam Wierzbicki, and Nahid Shahmehri, editors, *Peer-to-Peer Computing*, pages 3–11. IEEE Computer Society, 2006.

[23] B. Loo, S. Krishnamurthy, and O. Cooper. Distributed web crawling over dhts, 2004.

[24] Athicha Muthitacharoen, Robert Morris, Thomer M. Gil, and Benjie Chen. Ivy: A read/write peer-to-peer file system. In *OSDI*, 2002.

[25] Mikito Nakamura, Jianhua Ma, Katsuhiro Chiba, Makoto Shizuka, and Yoichiro Miyoshi. Design and implementation of a p2p shared web browser using jxta. In *AINA*, pages 111–116. IEEE Computer Society, 2003.

[26] Mike P. Papazoglou and Dimitrios Georgakopoulos. Service-oriented computing. *Commun. ACM*, 46(10):24–28, 2003.

[27] David Powell. Group communication (introduction to the special section). *Commun. ACM*, 39(4):50–53, 1996.

[28] Jan S. Rellermeyer and Gustavo Alonso. Services everywhere: Osgi in distributed environments. In *EclipseCon*, 2007.

[29] P. Resnick. Internet Message Format. RFC 2822 (Proposed Standard), April 2001.

[30] Patrick Reynolds and Amin Vahdat. Efficient peer-to-peer keyword searching. In Markus Endler and Douglas C. Schmidt, editors, *Middleware*, volume 2672 of *Lecture Notes in Computer Science*, pages 21–40. Springer, 2003.

[31] Ozgur D. Sahin, Cagdas Evren Gerede, Divyakant Agrawal, Amr El Abbadi, Oscar H. Ibarra, and Jianwen Su. Spider: P2p-based web service discovery. In Boualem Benatallah, Fabio Casati, and Paolo Traverso, editors, *ICSOC*, volume 3826 of *Lecture Notes in Computer Science*, pages 157–169. Springer, 2005.

[32] Daniel Sandler, Alan Mislove, Ansley Post, and Peter Druschel. Feedtree: Sharing web micronews with peer-to-peer event notification. In Castro and van Renesse [6], pages 141–151.

[33] Emil Sit and Robert Morris. Security considerations for peer-to-peer distributed hash tables. In Peter Druschel, M. Frans Kaashoek, and Antony I. T. Rowstron, editors, *IPTPS*, volume 2429 of *Lecture Notes in Computer Science*, pages 261–269. Springer, 2002.

[34] Ion Stoica, Robert Morris, David R. Karger, M. Frans Kaashoek, and Hari Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *SIGCOMM*, pages 149–160, 2001.

[35] Jeremy Stribling, Isaac G. Councill, Jinyang Li, M. Frans Kaashoek, David R. Karger, Robert Morris, and Scott Shenker. Overcite: A cooperative digital research library. In Castro and van Renesse [6], pages 69–79.

[36] Chunqiang Tang, Zhichen Xu, and Sandhya Dwarkadas. Peer-to-peer information retrieval using self-organizing semantic overlay networks. In Anja Feldmann, Martina Zitterbart, Jon Crowcroft, and David Wetherall, editors, *SIGCOMM*, pages 175–186. ACM, 2003.

[37] J. Robert von Behren, Jeremy Condit, Feng Zhou, George C. Necula, and Eric A. Brewer. Capriccio: scalable threads for internet services. In *SOSP*, pages 268–281, 2003.

[38] Spyros Voulgaris, Daniela Gavidia, and Maarten van Steen. Cyclon: Inexpensive membership management for unstructured p2p overlays. *J. Network Syst. Manage.*, 13(2), 2005.

[39] Matt Welsh, David E. Culler, and Eric A. Brewer. Seda: An architecture for well-conditioned, scalable internet services. In *SOSP*, pages 230–243, 2001.