**Master's Thesis**

Title

# Search Load Distribution Mechanism for Active P2P Networks

# アクティブ P2P ネットワークにおける

# 検索負荷分散機構に関する研究

Supervisor

Prof. Hideo Miyahara

Author

Shi Jiangang

February 13, 2004

Department of Information Networking

Graduate School of Information Science and Technology

Osaka University

Master's Thesis

Search Load Distribution Mechanism for Active P2P Networks

アクティブ P2P ネットワークにおける検索負荷分散機構に関する研究

Shi Jiangang

## Abstract

Peer-to-Peer (P2P) applications have considerably attracted millions of people from Internet researchers to end users, but there is still a long way for such applications to become advanced Internet applications as World Wide Web. The greatest problems regarding P2P applications come from complexity and inconvenience in peer discovery and content location. For example, even if two peers are in the same subnetwork and through the same router to the Internet, they can not be aware of each other without some complex mechanisms, such as a centralized directory server or a bootstrapping node. They are bottlenecks and single points of failure as Web servers in the Internet. Furthermore, since peers connect to and disconnect from a P2P logical network randomly, the location of shared contents and network topologies are continuously changing. Thus, current improvements for web applications, such as caching and CDNs, are not associated with load distribution among those servers in P2P application because they are aimed at developing static contents. If an underlying network is aware of the conditions and demands for overlying applications and provides application-oriented services, it is helpful to improve the QoS (Quality of Service) of networked applications efficiently. For example, for P2P applications, a network can efficiently distribute and balance load among peers by estimating application-level performance from passively gathered packet-level information, with much flexibility and transparently.

In this study, we propose a framework constructed on active network technologies for dynamic and transparent search load distribution for P2P applications. Active routers at a network layer monitor and analyze P2P traffic passing through themselves, then decide whether it is necessary

to conduct some user-defined services to improve the QoS of P2P applications. One application that benefits from active network technologies is “active load balancing for P2P directory servers”. When an active router considers that a directory server is a bottleneck, it redirects client peers to other directory servers. Another application is “active transparent P2P caching”, in which an active router caches directory information and answers query requests on behalf of actual P2P directory servers. We implemented the latter as an active OpenNap cache proxy and showed that our proposal is effective in distributing load on a P2P network depending on predefined conditions. An active router passively monitored and evaluated the load on an OpenNap server. Then, it initiated a virtual cache server as an active service based on the predetermined condition. It could successfully reduce the load on the OpenNap server without deteriorating the QoS of P2P applications. With the proposed framework, one can easily operate desired services in a P2P network with much flexibility.

**Keywords**

load distribution, P2P, active network, active P2P, service deployment

## Contents

## List of Figures

## List of Tables

## 1 Introduction

After Web technology was invented around 1990, the Internet has evolved with breathtaking speed. Now, 14 years later, the Internet population has reached 682,419,512 [1] and still grows daily. In view of network technology, the success of the Internet is based on a client-server model, especially Web servers. An Internet Content Provider (ICP) publishes its HTML contents on a central Web server, and users try to retrieve contents from the server granted they know the server's address. However, for popular ICPs, e.g., Yahoo, they found it impossible to satisfy so many users with one server in spite of deploying the most powerful system and buying more and more bandwidth. To distribute load from a single server, new technologies such as load balancing, Web caching and CDNs (Content Distribution Networks) had been developed. For example, by deploying cache servers at some points near users and making them as proxies for a web server, the load is distributed among servers [1]. Consequently, the original web server becomes liberated from managing a considerable number of requests. In addition, since the load is also distributed over networks, congestion caused by the concentration of traffic is avoided and users experience better transmission performance. Such load distribution technologies have demonstrated that they work very well. One obvious example is that Web servers seems to respond much quicker than before, in spite of the fact that the number of contents and users had grown at a speed much higher than the growth of hardware performance.

However, with the increase in the computing capability of user PCs and the development of access networks, multimedia contents including video and audio streams have become widely used. The data size of each content increases and it requires a large bandwidth near servers than text and simple images. In addition, multimedia contents are so attractive that more and more people have begun using the Internet for entertainment purpose. As a result, the current applications based on the client-server architecture are now collapsing due to the lack of scalability between the volume of contents and the number of users. By deploying proxy servers and distributing load

[1] Last updated on November 24, 2003 at http://www.internetworldstats.com/

among them, such scalability can be attained to some extent. However, they are still deployed and located statically and cannot adopt the changes in demand and user distributions. If an ICP wants to provide sufficient availability guarantees to clients anywhere in the Internet, it has to prepare a considerable number of mirror servers, caches, and proxy servers.

The emergence of Napster, an MP3 file sharing system, followed by hundreds of peer-to-peer (P2P) applications, remind us of the fact that a personal computer connected to the Internet can be not only a client requesting contents from others, but also a server providing local objects to others. When those personal computers are connected together and serve each other, the most powerful service system can be constructed, which can provide almost unlimited computation ability and storage space.

The advantage of P2P is that users, called peers in P2P applications, can exchange information, contents, and objects directly among themselves, without mediation of central servers. A peer does not need to deposit an object that it wants to offer in one or more servers beforehand. A peer who needs the object directly requests the provider peer to provide the consumer peer with the object. As a result of direct communications among peers, we can avoid the concentration of load on some specific points of a network. However, a peer must know from whom it can obtain an object needed. The effectiveness and usefulness of P2P applications largely depend on how each peer discovers other peers and how it locates the desired objects in a dynamically changing P2P network.

Several solutions have been proposed to deal with the problem, such as “Centralized Directory” [2] as in Napster, “Decentralized Directory” as in KaZaA [3] and JXTA search [4], “Query Flooding” as in Gnutella, and “Distributed Hash Table (DHT)”-based mechanisms as in Chord [5]. However, they still rely on some centralized nodes. For example, Napster maintains a directory of contents shared by all peers. It introduces advantages of a client-server architecture such as convenience of management and control and guaranteed QoS for peers, but it also introduces disadvantages of servers including bottlenecks and single points of failure. Mechanisms for

server load distribution have been proposed. For example, in [6], servers are chained together to distribute search load in OpenNap P2P application. However, since most of them are configured statically, they cannot adapt to a dynamically changing network environment. Pure P2P applications such as Gnutella, where there is no directory server, uses a "query flooding" mechanism, in which P2P messages are relayed peer by peer. It succeeds in avoiding the bottlenecks of directory servers with some self-organization protocols. Unfortunately, since those P2P protocols have no knowledge of an underlying network layer, they often bring redundant P2P traffic into the network or guide P2P traffic to congested network links, which consume much network resource and takes a long time to obtain desirable results. In addition, they still need special purpose servers, called bootstrapping nodes, to introduce new peers to a P2P network. In KaZaA, a kind of cluster-based P2P networks [7] does not rely on any statically prepared servers. Instead, at least one leader peer is selected among peers in every cluster to be the directory server. When the leader leaves off, a new leader peer is appropriately selected. To place those centralized nodes dynamically to satisfy a practical situation, some complex algorithms are needed. Such algorithms rely heavily on mutual trust and often involve much communications among peers. Thus, it consumes much network resource and takes a long time to obtain desirable results.

If an underlying network is aware of the conditions and demands of overlying applications and provides application-oriented services, it is helpful to improve the QoS of networked applications efficiently. Active network technologies provide a programmable infrastructure for various of network applications [8]. Active nodes or active routers, which constitute active networks, are programmable and can be dynamically tailored to network administrator's, application's, and even user's demands. Basically they process packets at a network layer, but they can apply application-specific manipulation to packet payload if needed. Sample applications of active network technologies include DDoS defense mechanisms [9] and multimedia broadcast [10].

In this thesis we present a framework based on active network technology to dynamically distribute load among servers or networks transparently. Our framework does not need overlying P2P

applications to be aware of underlying active networks. Furthermore, there is no need for active nodes to communicate with each other to attain the load distribution. Each active node conjectures the current load state of a P2P network from locally available and passively accumulated information. It monitors and analyzes P2P traffic which coincides with the prescribed conditions. Then, it decides whether it should take some actions to contribute to load distribution on a P2P network, without any message exchange.

The rest of the theis is organized as follows. In section 2, an active P2P network is introduced briefly, which consists of a layered network with P2P logical networks overlying active networks operating on a network layer. In section 3, the framework of active load distribution for P2P applications is proposed and described in detail, including a brief introduction of an active node architecture, descriptions of two example of applications, and a scenario of service deployment and management. In section 4, to verify and show the practicality and usefulness of our proposed framework, one application is chosen and implemented on an actual system. We also discuss several experiments conducted. Finally in section 5 a summary is given and some future research studies are described.

## 2 Active P2P Network

Not only P2P applications, but all networked applications benefit from technologies that introduce some kind of intelligence into networks. If a network can offer application-dependent and tailored behaviors towards packets, including prioritized scheduling and intelligent routing, a higher QoS can be provided to users of overlying networked applications. In this study, we consider a layered network with P2P logical networks overlying active networks operating on a network layer. We call it an "active P2P network". In an active P2P network, P2P applications can attain better QoS, in terms of, for example, response time, the number of responses, and the availability of searching results, with the help of active network technologies in a transparent way. P2P applications do not need to be aware of an underlying active network. There is no need to modify P2P applications to benefit from an active network.

In the following subsections, P2P and active network technologies will be explained separately. Then, the concept of an active P2P network, a type of layered network of P2P and active networks will be introduced.

### 2.1 Peer-to-Peer Networks and Applications

Peer-to-peer (P2P) is one of the networked applications that users, called peers, can retrieve objects directly from each other without mediation of servers. Peers organize a logical network with logical links that correspond to the relationship among two peers that are in a neighborhood. In the view of client-server model, each peer is a server, a client, and a node at the same time. For example, when a peer permits other peers to download an MP3 file from its hard disk, the peer plays the role of a server. The peer also obtains files from other peers as a client. When messages exchanged among peers are relayed on logical P2P networks, a peer behaves as a node.

For a peer to obtain an object, it must first find a peer that can provide it with the desired object. Thus, we need mechanisms for a peer to discover other peers and determine which peers have the desired objects. Current P2P applications, most of which are P2P file sharing systems,

have different mechanisms for dealing with such peer discovery and content location problems. Here we employ the way in [2] to classify P2P applications into three basic architectures.

1. Centralized Directory

   In this architecture, a central server provides a directory service for the all of peers. When a peer launches a P2P file-sharing application, the application first summarizes a list of objects in a local hard disk. Then it registers the list with a peer ID, IP address, and other type of information into a well-known central server, to which it establishes a permanent TCP connection. The server maintains a directory database that maps an object name to a set of IP addresses of provider peers. The directory database is kept up-to-date by receiving transaction messages. When a peer adds or removes an object, it informs the directory server of such changes. When a peer leaves the P2P network, the server detects the disappearance through the termination of the TCP connection.

   To find an object, a peer sends a query message including some keywords and the maximum number of results it expects to the directory server. The server examines its directory database to find records of the object that contains the specified keywords. If there are, the server generates a list of records that match the query. On receiving the list, the peer selects a peer from which it directly retrieves objects.

   Napster, OpenNap, and instant messaging applications such as ICQ, Yahoo messenger, and MSN messages are typical examples of this class.

2. Decentralized Directory

   In this architecture, peers are clustered into groups. The entire logical P2P network consists of groups. A newly launched peer first inquires from a group whether it can join a bootstrapping node. Then, it becomes a member of the specified group. In a group, there is a leader that plays the role of a directory server of the group. It maintains information of the objects deposited by peers in the group. Thus, the mechanisms including registration, query, object

retrieve in a group are similar to the “Centralized Directory” architecture. Additionally, to achieve more results, one leader peer can forward queries from its client peers to another leader peer. Of course, some mechanisms are needed to decide who becomes the leader of a group. A leader peer is chosen statically among peers in a group, or dynamically by an algorithm taking peer’s performance into account.

Morpheus, KaZaA, eDonkey2000, and WinMX are typical applications of this class.

3. Query Flooding

   In this architecture, all peers are equal. Since there is no directory server, this architecture is also called “pure P2P” while the others are called “hybrid P2P”. There is no hierarchical structure in a P2P network. To join a P2P network, a peer first sends a request to a bootstrapping node to provide it with a list of IP addresses of peers that have already participated in the network. Then the new peer advertises its address to those peers. Consequently, a neighborhood is built among them.

   To find an object, a peer conducts “query flooding”. A peer begins flooding all its neighboring peers with query messages. To restrict the range in which the message propagates, a TTL (Time To Live) is appropriately set. Each of the neighboring peers first examines its local disk to determine whether it has the requested object. If it does, the peer sends back a response message to the requesting peer through an inverse way that a corresponding request message traversed. Before further relaying the query message, it decreases TTL. If TTL is larger than zero, the peer sends the message to all of its neighboring peers except for one from which the message originated.

   Gnutella and Freenet are typical applications using the query flooding mechanism.

Table 1 summarizes some advantages and disadvantages of the above three typical P2P architectures. We can see the fact that all of today’s P2P architectures require some always-up nodes. Independent of architectures, there are still bottlenecks and single points of failure.

Table 1: Advantages and Disadvantages of P2P Architectures

| **P2P architectures** | **Advantages** | **Disadvantages** |
|---|---|---|
| Centralized directory | 1. A peer can easily locate an object by sending a query to a directory server.<br>2. It is easy to manage and maintain the directory. | 1. A directory server is a single point of failure.<br>2. A directory server is a performance bottleneck.<br>3. The architecture lacks scalability. |
| Decentralized directory | 1. The load of directory service is distributed.<br>2. Failure of a directory server only affects a group of peers. | 1. A complex algorithm and mechanism are necessary to construct a hierarchical network.<br>2. Distributed directory servers are single points of failure of their groups and bottlenecks.<br>3. A bootstrapping node is necessary and is a single point of failure. |
| Query flooding | 1. All peers are equal.<br>2. No server is needed for searching. | 1. A search involves much communication and traffic.<br>2. A bootstrapping node is necessary and is a single point of failure.<br>3. A complex protocol is needed to maintain a logical P2P network. |

## 2.2 Active Network Technology

An active network (AN) is a network of dynamically and flexibly configurable intelligent nodes, called active nodes. Active nodes are programmable and can be dynamically tailored to network administrator's, application's, and even user's demands. Basically they process packets at network layer, but they can apply application-specific manipulation to packet payload if necessary.

There are three approaches to building an active network, a discrete approach, an integrated approach, and their combination [8, 11]. The first approach is the use of "Programmable switches"[8]. This approach is also called "out-of-band extensions" [12]. It separates the mechanism for injecting programs into a "programmable" node from the actual processing of packets as they flow through a node. A program is sent to a node as they would to a host. When a packet arrives at the node, the corresponding program is selected based on some header information and then executed. If a new version of the program or if a different type of processing is necessary, a new program or service can be deployed at this node. Most importantly, it usually occurs automatically.

The separation of program execution and loading is valuable when it is necessary for program loading to be carefully controlled or when individual programs are relatively large. A program consists of modules. Modules cooperate with each other through interfaces. Such modularization facilitates service deployment. An active node maintains some modules for common and primitive functions and dynamically retrieves a missing module for a special purpose function on demand. However, even though it is convenient to introduce new services at network layer with such programmable switches compared with conventional methods, it is not sufficiently flexible yet. Imagine the following scenario: a customer buys one hour of reservation of high guarantee of network QoS from an ISP for an Internet video conference. In this active network approach, the ISP first composes a program or a policy for prioritized packet scheduling. Then, it distributes and deploys the program into the appropriately chosen active nodes. The program distinguishes those packets belonging to the conference and gives them a favorable service. One hour later, programs in active nodes are stopped.

The second, an integrated approach, is “Capsules”. This approach is also called “in-band active packets” [12], in which programs are integrate into every packet. This means that a packet, called capsule, carries a program by itself. When a capsule arrives at a node, a program contained in the capsule is loaded by a code loading mechanism. Then, the node applies the program to the capsule. This approach can supply packet-specific computations, which is more flexible and dynamic than application or session-related computation by the discrete approach. For example, an active packet contains a small program for dynamic and conditional packet routing depending on network conditions. Every time it arrives on an active node, the program is executed to find a preferable path for the next hop. Despite the limitation in the size of program that can be embedded in an active packet, the integrate approach provides a highly dynamic and flexible deployment of active services. For the video conferencing example, on receiving a reservation, the ISP encapsulates all packets belonging to the conference into active packets at an ingress node. The active packets have a program code for prioritized packet processing. One hour later, the ISP stops the encapsulation. In this way, all conference traffic is given a preferable service without statically configuring active nodes as in the discrete approach.

The third approach is a combination of in-band and out-of-band approaches. Between these two, we also have hybrid approaches. Packets carry only short scripts, and these scripts carry elaborated services, which are loaded via out-of-band signaling.

Active nodes are the basic elements constituting active network. In view of software architecture, an active node is similar to those general software platforms. As Fig. 1 illustrates, an operating system, called “NodeOS” in AN research, manages hardware and software resources including transmission, processing, and storage resources. An execution environment (EE) conceals the details of the operating system from applications and supplies them with APIs, similarly shells in Unix or Java virtual machines. User-defined or application-oriented programs running on an execution environment are called active applications (AAs). Such programs written in C language or Java in some cases, depending on the active network architecture [13].

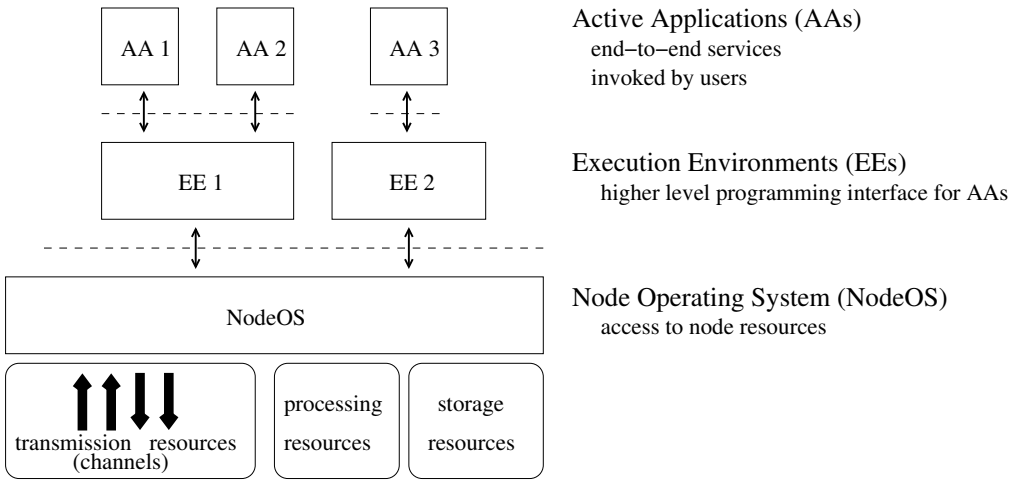



Figure 1: Active Node Architecture: Building Blocks

Active traffic is a stream of active packets, in which signals, configurations, messages, descriptions, executable codes or reference of codes are embedded. In order to distinguish active traffic from conventional IP traffic, the Active Network Encapsulation Protocol [14] was proposed. Active packet has an ANEP header that includes a "Type Identifier" field. Well-known Type IDs are assigned to specific EEs by the Active Network Assigned Number Authority. If a particular EE is present at a node, packets containing a valid ANEP header with the Type ID assigned to that EE will be routed to channels connected to the indicated EE. Obviously, such a mechanism is highly dependent on network layer processing.

Thus, active nodes provide to users, who can be end users or network administrators, the ability of deploying their programs or services to take advantage of resources at active nodes. However, before active network technologies are introduced into the real world of the Internet, we must first answer how to deploy those active applications, as well as resolve problems of authorization and management. A brief introduction about service deployment can be found in the following section 3.

Another problem that should be considered is the performance of active nodes. Highly intelligent and tailored packet processing introduces additional load to active nodes which may disturb

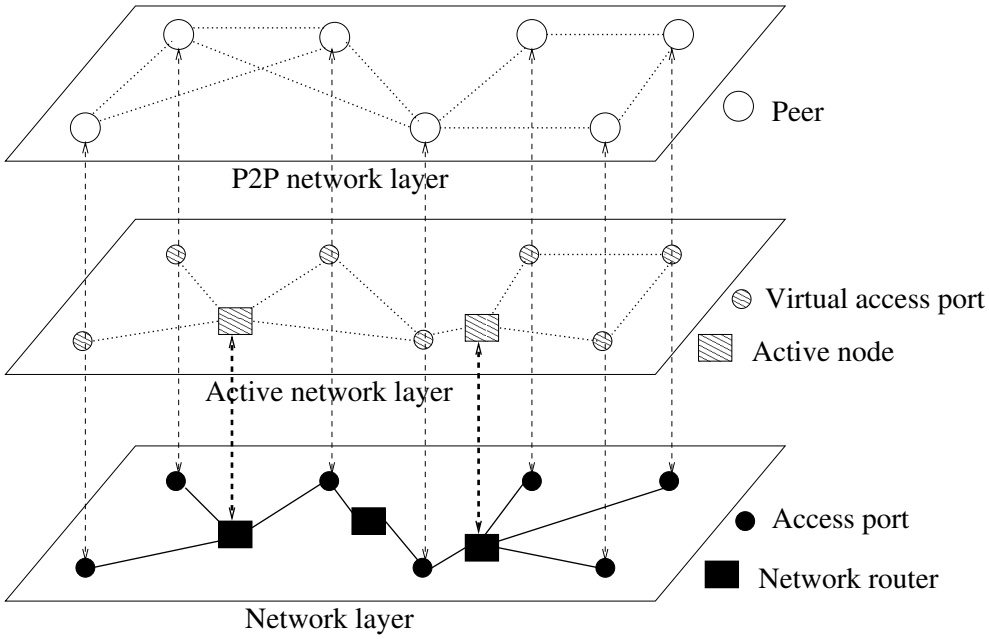



Figure 2: Active P2P Network

the routing and forwarding of general IP packets. Therefore, it is not expected that active network technologies are employed by the Internet core backbone routers, which require that packets be processed at a maximum line speed. Some studies of the performance of PC router-based active nodes can be found in [15].

### 2.3 Layered Network of P2P and Active Networks

In recent years, some ideas about deploying services at network layers to improve the performance and behavior of networked applications have been proposed [16-19]. The Internet which provides only the best-effort service cannot satisfy the QoS demands of advanced networked applications any more. Applications require networks to guarantee or control various QoS in terms of, for example, packet transfer delay, delay jitter, and bandwidth. If networks guarantee the allocation of network resources as in IntServ and DiffServ, applications can provide users with services of high stable quality that can successfully predict the set of entities, i.e., hosts, nodes, and links, and the amount of resources needed. For some classes of applications, even a lower degree of

support from networks is helpful. For example, some multimedia applications have the capability of adapting data emission rate to network conditions. For this purpose, they conjecture network conditions by observing the reception of packets [20], because they do not have any direct means of obtaining information including the topology of the network, the utilization of links on the path that a connection traverses, and the existence and characteristics of other flows that compete for network resources. If underlying networks can provide such applications with detailed information on networks, they can accomplish effective and efficient controls.

In contrast with the above passive mechanisms, the active network is a technology that makes networks aware of applications and enables networks to behave in accordance with application's demands. Since active nodes operate on a network layer and they are deployed in the network as network equipment similar to routers, they are also called active routers. They can easily gather packet-level information such as IP address, port number, protocol type, and timestamp, while observing packets flowing through themselves. Furthermore, they can even analyze and manipulate user data part of a packet. With information on underlying IP networks, active nodes can derive a packet-level QoS such as packet transfer delay, delay jitter, and link utilization. They can further interpret those QoS to application-level QoS with information related to overlying applications. Then, they can configure their behavior to be preferable to such applications to provide them with a better level of service. For example, an active node can adjust the quality of a video stream to the available bandwidth by filtering out some packets [21].

In this study, we consider such a layered network of P2P and active network architecture, as shown in Fig. 2. An IP network consists of nodes, hosts, and links, more specifically, of routers, ports of network interface of hosts, and physical links. Some routers in an IP network are also active nodes. A P2P network consists of peers, i.e., hosts. An active network layer between them consists of active nodes and virtual access ports. Since an active network is transparent, but the overlying P2P network and the underlying IP network are not aware of it. Thus, we call ports on an active network virtual access ports. Routing among network access ports is based on IP routing

protocols, e.g., RIP, OSPF, and BGP. Messages on a P2P application layer are routed using a P2P protocol. Active nodes provide conventional routing functions to packets that are irrelevant to the active load distribution. Active nodes have chances of collecting sufficient information on P2P traffic passively. They analyze the behavior of P2P applications. Furthermore, by dynamically introducing active services, they process P2P traffic with some application-oriented policy. For example, they distribute the load of P2P searching transparently depending on the need.

## 3 Active Load Distribution for P2P Applications

Active nodes in our framework provide conventional routing functions to packets that are irrelevant to the active load distribution. Thus, we call them active routers hereafter. A router constituting an IP network can be either of a conventional router and an active router. Hosts that participate in P2P applications organize a P2P network.

### 3.1 Architecture of Active Routers

Figure 3 illustrates the architecture of an active router in our framework. It follows the architecture of a typical active node having NodeOS, which manages the node resources such as link bandwidth, CPU cycles, and storage; the Execution Environments (EEs), which provide active applications with APIs using NodeOS's resources; and the Active Applications (AAs), which program the virtual machine on an EE to provide an end-to-end service. For an active router, a packet scheduling and routing function module is the basic element of NodeOS. We propose to deploy a "service creation engine" as a proactive P2P service at an active router, which is to load other active services. A "monitor", which consists of "P2P packet analyzer", "server state database", and "manager" modules, is an active application in charge of collecting information from P2P traffic. An "active P2P service" is a dynamic module that can be loaded by service creation engine.

### 3.2 Monitor, Measurement and Decision Mechanisms

First, when incoming packets enter IP stacks and wait for scheduling, a packet capturer module at an active router filters out all packets it has interest in. The packet filtering policy depends on a P2P application. The easiest way is to use a well-known TCP port. For example, "6699" and "8888" are widely used in an OpenNap server and "9700" is in JXTA application.

Next, by checking the payload of packets and comparing keywords of payload with P2P protocols, a P2P packet analyzer can identify the type of P2P messages with P2P protocol knowledge. For example, each message in OpenNap protocol [6] to or from a server are in the form of $< length >< type >< data >$, where $< length >$ specifies the length in bytes of the $< data >$

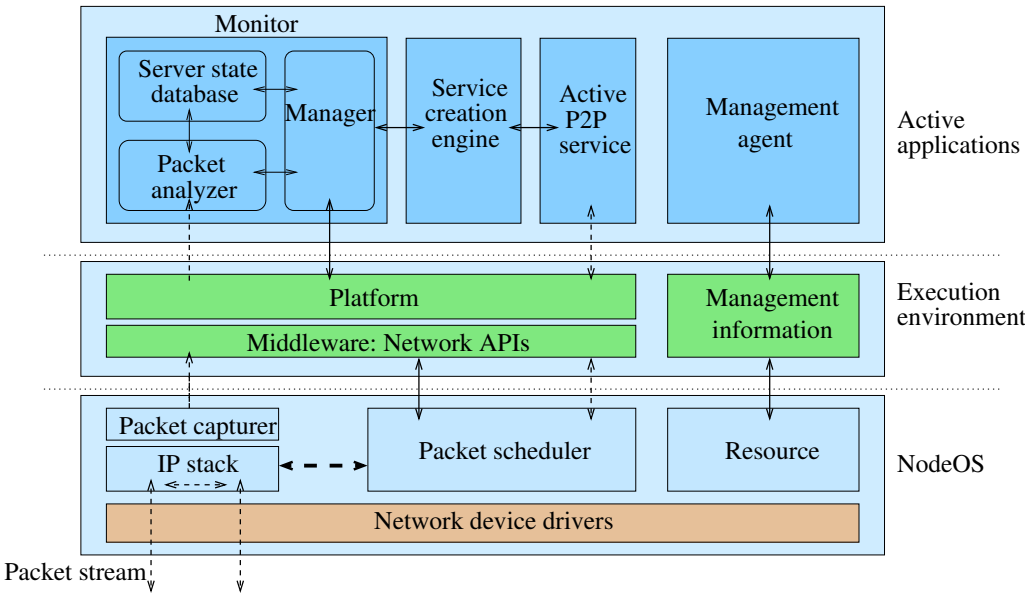



Figure 3: Modules in Active Router

portion of the message and $< type >$ specifies the command type. Then, the P2P packet analyzer stores P2P packet information into a server state database, which includes IP addresses, port numbers, P2P message types, and timestamps.

Then, a manager identifies a bottleneck peer, to which load accumulates on. Since there is no direct way of getting information on load state on CPU, memory, bandwidth on remote peers, an active router estimates the current load state from the server state database which is constructed passively. The metrics of load include response time, the number of search results, and query rate [22]. Although they are not precise, they are practical. For example, an active router can take the response time of a server to measure load. If it finds that the response time is much longer than before or it exceeds the predetermined threshold, it considers that the server is now a bottleneck.

When the manager has identified a bottleneck based on predefined policies, it requests a service creation engine to introduce a corresponding active P2P service to the active router. After service deployment, incoming packets satisfying some predetermined conditions are forwarded to the active P2P service by some packet redirection policies executed in packet scheduling modules in

NodeOS. A management service at a local active router should be in charge of inserting or deleting such packet redirection policies.

An active P2P service processes packets transparently. That is, client peers and servers do not need to know the existence of active P2P services on their communication channels. No static or dynamic configuration is required on client peers and servers to benefit from active P2P services. In some cases, packets are answered by an active P2P service and are not sent to a server, but an overlying P2P application does not notice such interception.

When the monitor considers that the remote server is no longer a bottleneck or is underutilized, it requests the server creation engine to stop the active P2P service. The decision is based on some predetermined policy, such as a threshold-based one. Since the active P2P service communicates with a server instead of client peers, an active router can obtain load state information of the server after introducing the active P2P service.

### 3.3 Examples of Applications of Our Framework

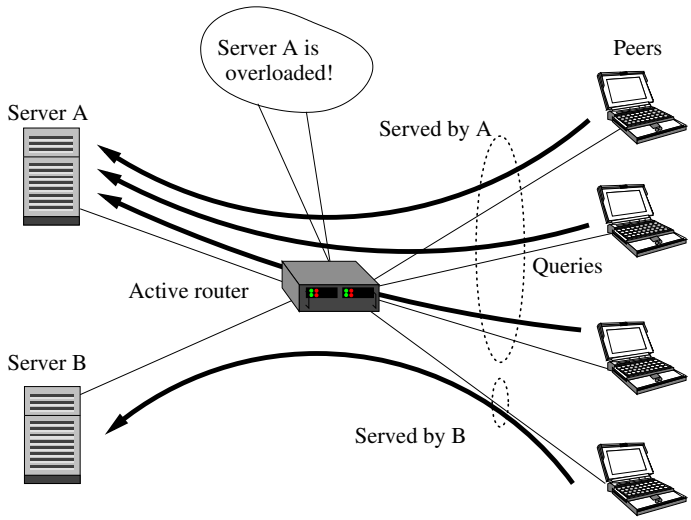
Here, we briefly introduce two examples of applications. The first example is active load balancing for P2P directory servers. A directory server that maintains a list of resources and their location information in P2P applications is a single point of failure and a performance bottleneck. By introducing redundant servers or "Decentralized Directory", the server load can be distributed, but not evenly. OpenNap [6] uses a "Metaserver" as a portal to direct peers to different OpenNap servers, although it is statically arranged and peers must be configured to use a specific portal.

In this example, active routers act as brokers to distribute load among several servers. Using our framework, client peers do not need to know which server it is actually connected with. Furthermore, load distribution is performed in accordance with server load states. First, an active router captures and analyzes those P2P traffic passing through it to find servers and estimate their loads, as Fig. 4(a) shows. Response time can be chosen as a parameter for measuring the quality of services of a directory server in this case. Response time can be calculated as in function 1. If the

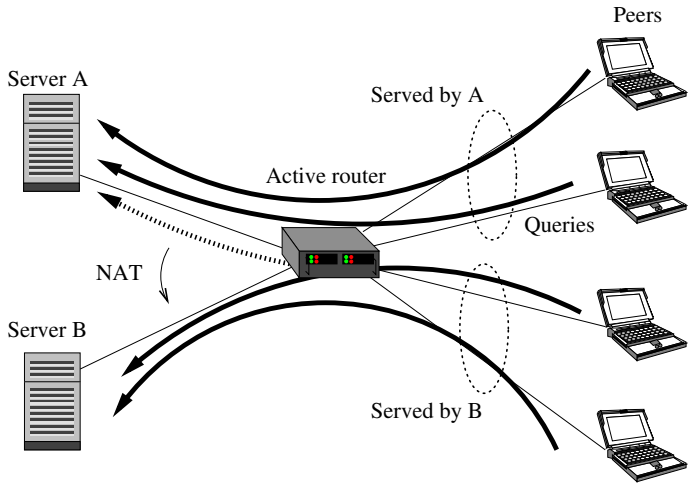
response time at a server becomes longer than before, an active router forwards queries to another server that it considers unloaded. For this purpose, a manager in Fig. 3 activates a load balancing service as an active application at an active router. Then the load balancing service rewrites the destination address of incoming P2P packets to redirect them to an unloaded server as Fig. 4(b) illustrates. Because of address rewriting, the source address of a response message is the address of the server that the active router selected, not that of the server that the client intended. Thus, an active route has to rewrite the source address of response messages to the address of the originally requested server in order to provide transparent interleaving.

Figure 5 illustrates the mechanism of network address translation. In this figure, there are two directory servers, A and B. A client peer establishes a TCP connection to server A. Then it sends a query to server A and receives a response from server A. By observing P2P traffic, an intermediate active router considers that server A is overloaded. It establishes a TCP connection to server B. Then, the next query from the peer to server A is redirected to server B by the active router. If the client peer is connection-aware, for example, if it is employing a TCP session connection, mechanisms to forge original connections are needed. An active router monitors and keeps all parameters associated with the original session, e.g., sequence number of a packet, as shown in Fig. 5 shows. Based on the snapshot, an active router can initiate a "virtual TCP session" to pretend to be the peer. Of course, for connection-unaware peers or UDP sessions, SNAT (source network address translation) and DNAT (destination network address translation) mechanisms in Fig. 5 are sufficient.

The second example of application of our framework is active transparent P2P caching. Contents popularity in P2P networks follows a Zipf-like distribution [23], which implies that caching in the P2P network is still a very efficient means of avoiding redundant traffic as proved by its power on World Wide Web. However, to take advantage of caching, cache servers should be carefully placed in a P2P network taking the distribution of peers, resources, and their popularity into account. Active routers exist over networks. If an active router at an appropriate location, which




(a) Measurement




(b) Load balancing

Figure 4: Active Load Balancing for P2P Directory Servers

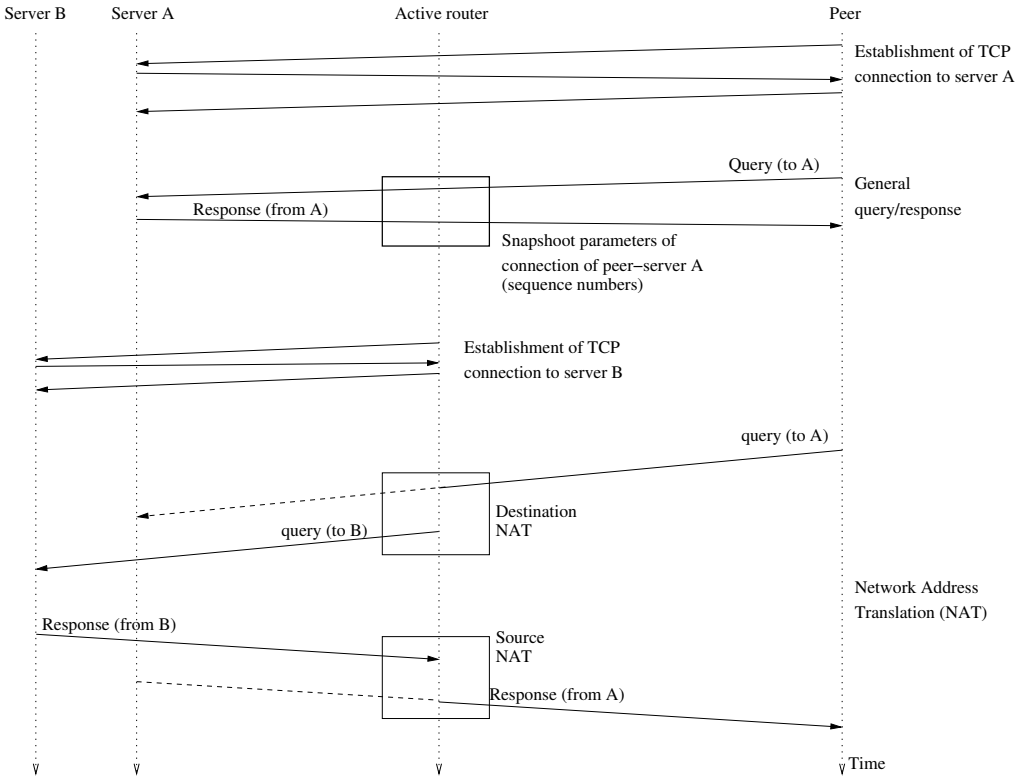
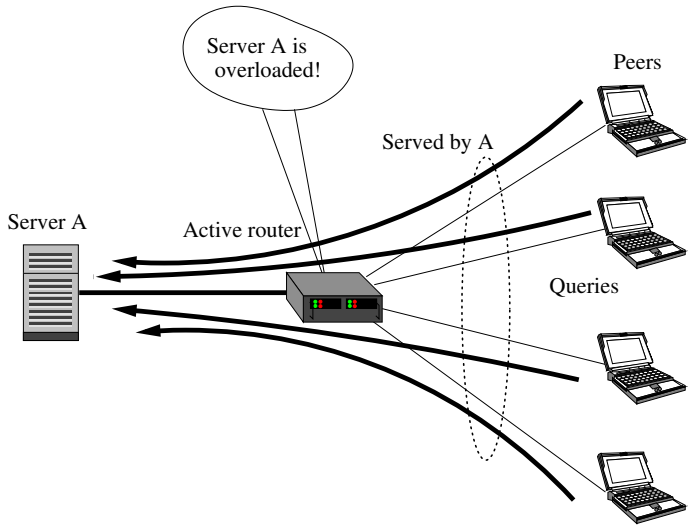
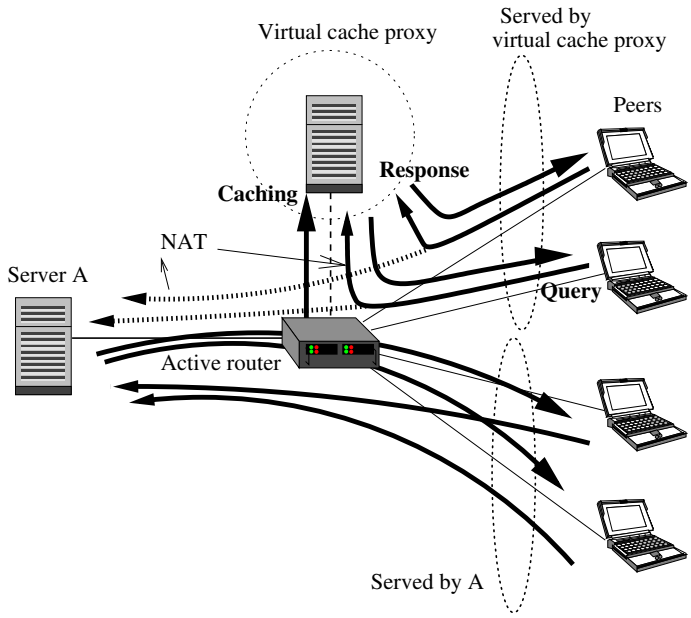



Figure 5: Transparent Query Redirection




(a) Measurement




(b) Virtual Caching

Figure 6: Active Transparent P2P Caching

depends on the conditions of P2P networks and applications, provides a functionality of caching, the QoS of overlying P2P applications is expected to be considerably improved. P2P traffic can be categorized into two: protocol messages such as queries, responses, and other command messages, and object data themselves such as music and video files. Therefore, caching can also divided into two classes: cache of P2P protocol messages and cache of contents.

As Fig. 6(a) shows, in this example, an active router first monitors the P2P traffic directed to a peer that holds resources or a server that answers query messages. When it considers that the load on the peer or the server is too high based on, for example, the number of messages, it initiates a virtual cache proxy as an active P2P service. Messages originating from the peer or the server are investigated and their contents are deposited in the cache, as in Fig. 6(b). Then, the active router redirects all incoming messages for the peer or the server to the local virtual cache, so that the cache can answer requests transparently.

### 3.4 Service Deployment and Management

Obviously, a safe, convenient, and practical service deployment and management system is also needed to load active P2P services in the Internet. The deployment of active services can be divided into two levels [24]. Network level deployment first identifies and selects nodes to run service components. Then, node level deployment selects, loads, and installs adequate implementations of service components on each of the selected active nodes [25, 26].

Network level service deployment has to solve some problems similar to those in P2P applications, including active node discovery, query and the retrieval of service codes. Thus, it is natural to deploy mechanisms similar to those for P2P applications. The centralized mechanism [27] depends on a management control center or a server which acts as a broker, storage server, or management station. There is a well-known management server. When a node joins an active network, it registers itself into the management server. The management server monitors the status of each active node. When it decides to deploy a service at some active nodes, it first chooses

appropriate active nodes to perform service-oriented functions. Then, it sends requests or commands to those active nodes. The mechanisms of node-level service deployment are used this time to obtain the requested functions. The decentralized mechanism [28] clones the protocol of the Gnutella-type P2P protocol and is based on message flooding, which is similar to "query flooding" in Gnutella. An active node is connected to several neighboring nodes to construct an overlay network similar to a Gnutella-type flat P2P network. When an active node wants to deploy a new service, it first locates the program codes needed. Then, it retrieves and executes them. When an active node wants other active nodes to deploy a new service, it first locates active nodes in addition to the program codes needed. Then, it sends codes to those active nodes. The IP routing protocol-based mechanism discovers active nodes along IP routing paths for the first step in network level deployment. In Ref. [29], the use of OSPF messages to carry information related to active node discovery is proposed. These messages are distributed in a network based on OSPF protocol. An active node that can understand additional information in an OSPF packet updates the information by, for example, inserting its node ID into the packet. Active nodes are successfully discovered and some of them are chosen for service deployment.

In this study, we prefer the centralized mechanism for network-level service deployment, which has been deployed successfully in research [30, 31]. In this model, SNMP (Simple Network Management Protocol) is used as a management protocol, as Fig. 7 illustrates. A central management station is in charge of managing all network equipment including conventional routers and active routers. The SNMP protocol is used as a communication protocol among management station and active routers. All program codes are stored and managed in a code repository server. Node discovery and code location are accomplished by a management station. Obviously, it is very convenient to integrate such a mechanism into current network management systems because most of them employ SNMP-based centralized control.

After network level deployment is accomplished, node level deployment begins at selected active nodes. For node-level deployment, a service creation engine is in charge of loading new

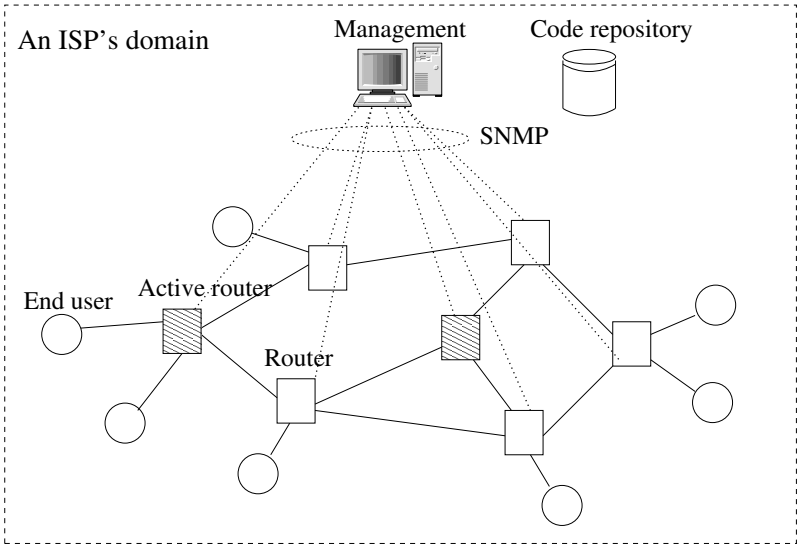



Figure 7: Centralized Deployment, Configuration, and Management

active services. It analyzes commands sent from a management station, and starts a requested active service. If the code of the requested service is not in an active node, a service creation engine retrieves it from the code repository specified by the management station.

The detailed mechanism of service deployment is as follows: A management station obtains state information on active routers by "get" commands in the SNMP protocol before service deployment. Then, based on the collected information, the management station appropriately chooses those active nodes that deploy the service. It uses "set" commands to request active nodes of service deployment. The Management Information Base (MIB) in active routers is similar to those in other nonactive routers except for some items associated with active services, as Fig. 8 illustrates. A branch, named "anarg_sce", is associated with a service creation engine. There are two queues for receiving commands from the management station. "Command_queue" is used to receive service deployment commands, and "config_queue" is to receive service configuration commands. The management station sends those commands to active nodes using "set" commands in the SNMP protocol. A service creation engine dispatches commands in both queues and executes them. For example, on receiving a service deployment command, a service creation

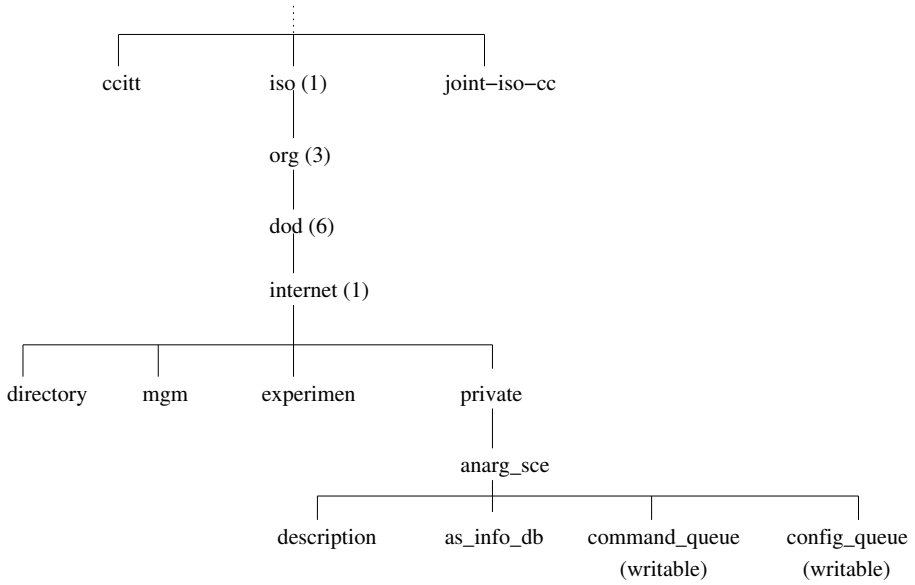



Figure 8: MIB Tree of Active Node

engine examines whether the conditions of the active node can deploy the service. If it can, the service creation engine retrieves related codes from a code repository server. Then, it updates its local MIB to reflect the new conditions, so that the management server can obtain up-to-date state information.

# 4 Implementation of Active OpenNap Cache Proxy

In this section, we implemented an application based on the OpenNap P2P protocol [6], “active OpenNap cache proxy”, as a prototype of active transparent P2P caching for investigating the practicality and effectiveness of our framework.

## 4.1 OpenNap protocol

OpenNap clones most of Napster’s protocols. The OpenNap server acts as a directory server for a cluster of peers. A peer connects to a server and provides it with peer information, including user name, link type, incoming port number, and a list of files that it wants to share with the others. Based on registered information, the server maintains a user information database. To find a file, a peer sends a query message to the server with parameters, such as keyword and the maximum number of results that it desires. Receiving a query, the server examines the user information database. If there is one or more records that match the query, the server returns those records to the requesting peer as a response message. Communications between a peer and a server are “keep-alive TCP session”-based. When the connection is broken by a peer or due to some network problems, a server releases the resource used by the peer by deleting the peer’s record in its database.

## 4.2 Architecture of Active OpenNap Cache Proxy

An active OpenNap cache proxy is a proxy server that answers query messages in behalf of OpenNap servers. In this implementation, an active router monitors P2P traffic belonging to OpenNap sessions. When it considers that a server is overloaded, it activates a virtual cache proxy as an active application to undertake some portion of query traffic. The virtual cache proxy maintains the local cache of directory information and answers queries based on the cache.

Figure 9 illustrates the architecture of an active router in our implementation. The NetFilter in Linux 2.4 kernel was chosen to realize the function of the packet scheduler module. Generally, it only passes incoming packets to an output network interface based on a routing policy, as in

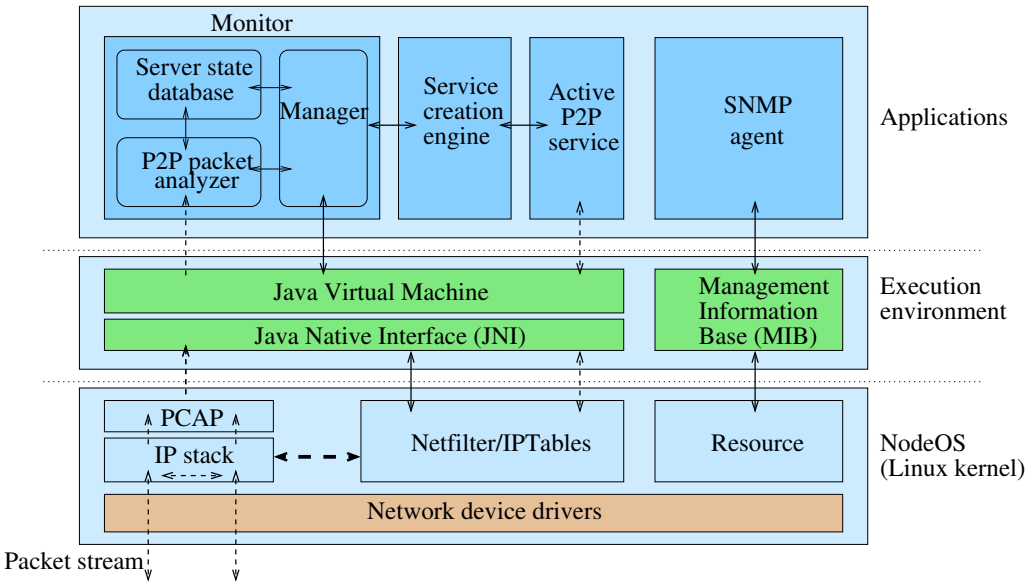
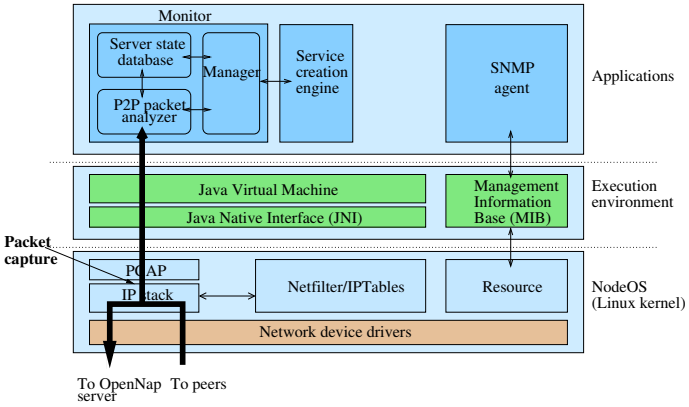


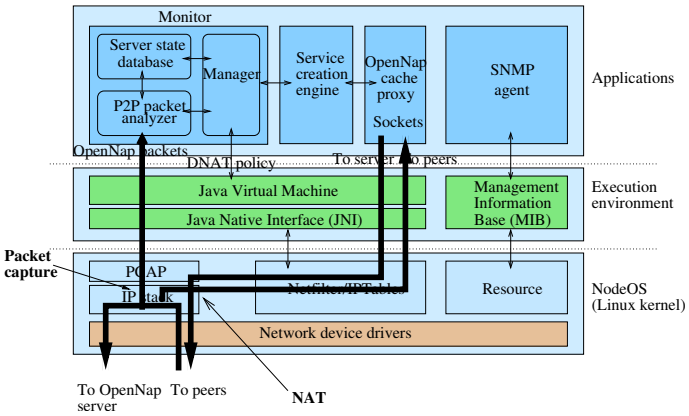

Figure 9: Architecture of Linux-Based Active Router

Fig. 10(a). Packets that need to be answered by the active router are redirected to the locally initiated virtual cache proxy based on a DNAT (Destination Network Address Translation) policy [32], as in Fig. 10(b). "PCAP" [33] and its Java's wrapper "jpcap" [34] were used as a packet capturer. An "analyzer" thread in the "monitor" in Fig. 3 analyzes captured P2P packets and inserts the history record of the target server peers in the "server state database". The database consists of two types of tables as shown in Fig. 11. A server list is a list of IP addresses of servers that an active router finds. Each server in a server list is related to a "query-response timestamp table". Each entry in a query-response timestamp table consists of the sequence number of the query, the timestamp of the query, and the timestamp of the first packet of the corresponding response message. "Manager" is in charge of controlling all of the other modules in the monitor.

A virtual cache proxy is built on an OpenNap server program, but it has additional caching and forwarding functions, as shown in Fig. 12. It is loaded as an active P2P service module by a service creation engine in Fig. 3. It provides directory service to all of its client peers transparently. At the same time, it behaves as a general client peer to a OpenNap server. It queries the remote OpenNap




(a) Monitor and measurement




(b) Packets in/out OpenNap cache proxy

Figure 10: Packet Flows in Active Router

server instead of the client peer redirected to this cache proxy. The local searcher of a virtual cache proxy receives queries from client peers through a TCP socket and examines the local directory database. The database is updated when it receives registration messages from client peers. The database consists of user names, link types, and a list of files to share. A request forwarder is used to forward query messages to a remote OpenNap server through a TCP socket. Cache is used to store those response messages received from the remote server. They are further explained in detail in the next section.

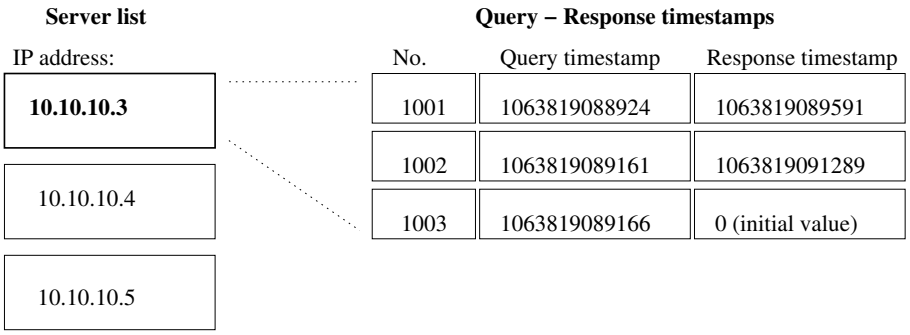



Figure 11: Server State Database

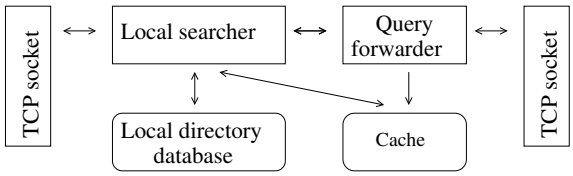



Figure 12: Modules in Virtual Cache Proxy

### 4.3 Mechanisms of Active OpenNap Cache Proxy

Search request messages and corresponding response messages are captured and examined at an active router.

First, PCAP captures those packets matching the filtering policy in which "protocol is TCP and destination port is 8888", in which "8888" is the port number in our OpenNap server.

Next, an analyzer thread in monitor distinguishes message types based on $< type >$ in a packet. If it is "200", then it is a query message sent from a client peer and the destination IP is the address of a server. If $< type >$ is "201" or "202", the message is a response. Therefore, the destination IP of the packet indicates the address of the peer that sent an inquiry to the server, whose address appears as the source IP of the packet. If the server IP is new to the server list, a new entry and a corresponding query-response timestamp table are created. If the message is a query, its timestamp is stored in the table with a new sequence number. The sequence number is initialized to one when a new table is created for a new server.

When the message is a response, the analyzer thread investigates the entry of the corresponding

query message in the query-response timestamp table. If the response timestamp field is zero, i.e., the initial value, it is filled with the timestamp of the packet. Otherwise, the packet is ignored. In OpenNap query-response communications, it is guaranteed that, for a peer, a series of response messages that a server sends follows that of query messages that it received. Thus, we can easily identify the query message that the response message corresponds to. A time limit $T_{rec}$ is set to define how long a record can stay in the query-response timestamp table. Every time the analyzer thread investigates the table, it will delete those records whose query timestamps are over time limit $T_{rec}$.

Then, the manager investigates the server state database. If a server satisfies some predefined conditions, it is regarded as a bottleneck or over loaded. Two schemes are supported to make decisions. With response-time metric, the response time $T_R$ of every query is calculated with the function

$$T_R = T_{fr} - T_q \quad , \tag{1}$$

where $T_q$ is the timestamp of a query packet, and $T_{fr}$ is the timestamp of the first response packet. Average $T_R$, as $T_{Ravg}$, is periodically calculated for each server in the server table. Then it is compared with the predefined threshold $T_{Rth}$. If $T_{Ravg}$ exceeds $T_{Rth}$, a virtual cache server for the server will be used in the next step.

With query-rate metric, the manager counts the number of query packets for each of servers. Every time the timer reaches the predetermined the boundary of slice $T_s$, it is initialized to zero and the query rate of the server is derived. The query rate is calculated using

$$R = N/T_s \quad , \tag{2}$$

where $N$ is the number of query messages in a time slice $T_s$. Average query rate $R_{avg}$ is periodically computed and compared with threshold $R_{th}$ to identify the bottleneck server.

Finally, a virtual cache server is loaded for the server by a manager as a Java thread. For a newly activated virtual cache server to deal with the query messages sent to the original server,

the manager introduces a new DNAT rule into the Netfilter. IPTables [32] are used as an interface to insert a DNAT rule. DNAT can direct client peers to a different server peer transparently. However, for some keep-alive session-based applications, it will destroy on-going connections. In our experiments, the peers redirected to a virtual cache server first disconnected their connections and re-established them.

A query $Q$ consists of several search parameters such as $key$, $Rmax$, $linktype$, and $filesize$. Here we only consider $key$, i.e., keyword, and $Rmax$, i.e., the maximum number of results desired. When a virtual cache server receives a query $Q$, it processes the query as follows: The virtual cache proxy generates a response message and first examines the local directory database using the given keywords. If the number of files that match the keywords is nonzero, their file names and properties are recorded in a response message. If the number of files locally found cannot satisfy the maximum number of results desired $Rmax$, the virtual cache proxy next examines the cache. Then, the results are merged together, fit to $Rmax$, and finally sent to a requesting peer as a response message.

A virtual cache proxy maintains a list of keywords that it has searched. The keywords are then compared with the history list. If they are new, the query $Q$ is forwarded to the server that it is originally directed to. However, the requesting peer does not wait for the response from the original server to avoid delay. A response message is sent to the requesting peer through the steps described above. As a result, peers receive a small result until the virtual cache proxy collects sufficient amount of information in its local database and cache. When the virtual cache proxy receives a response message from the server, it investigates the results. If there are one or more search results, the virtual cache proxy deposits them in its local cache for further use.

A TTL-based mechanism was introduced to keep the contents of the cache up-to-date. Since peers might leave from a P2P network without any notification, the directory database cached at an active router contains meaningless records. The time a new record is deposited in the cache is also kept in the cache. For each item of search result from a cache, the local searcher will compare

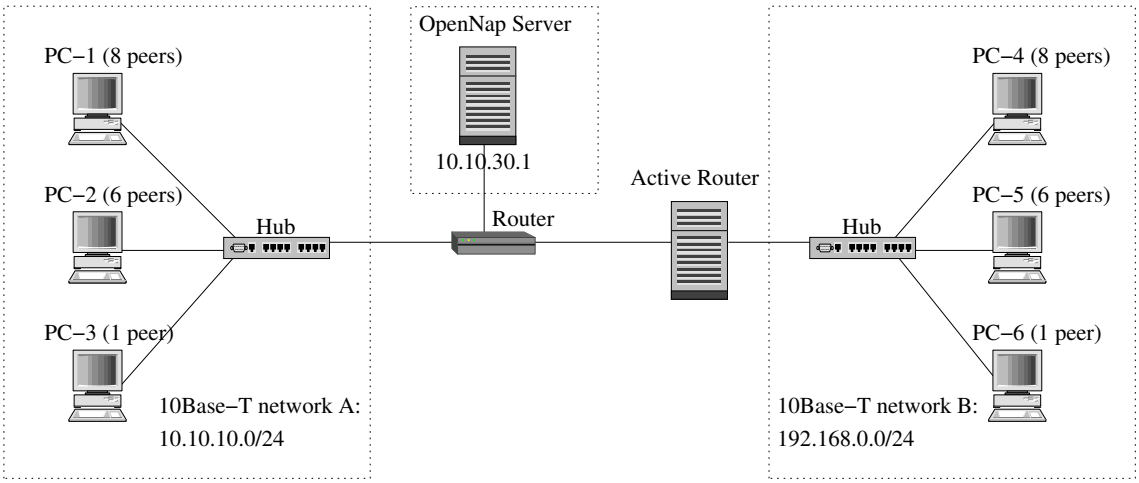



Figure 13: Experimental Environment

its timestamps with the current time. If more than $T_{cache}$, a predefined threshold, has passed, each result item will be ignored and the corresponding record is removed from the cache.

The monitor continuously measures the query rate sent to the OpenNap server for which the virtual cache proxy provides a directory service instead. When the average query rate goes below a low threshold $R_{thL}$, the manager disables a DNAT function in the Netfilter. Then, it stops the local virtual cache proxy and releases related resources.

### 4.4 Experiment and Evaluation

We conducted experiments on the system illustrated in Fig. 13 to verify the practicality and effectiveness of our framework from viewpoints of load balancing and quality of service. When the number of queries that a server receives decreases, we consider that load is moved to the active router. The quality of service in OpenNap application is evaluated in terms of the number of files that a response message contains and the response time that a peer experiences.

The network consists of three subnetworks connected with each other by an active router and a general IP router. Either network A or B has 15 clients running three PCs. Each peer joins the P2P network at random and sends query messages to a server once every 1.5 seconds. Either in

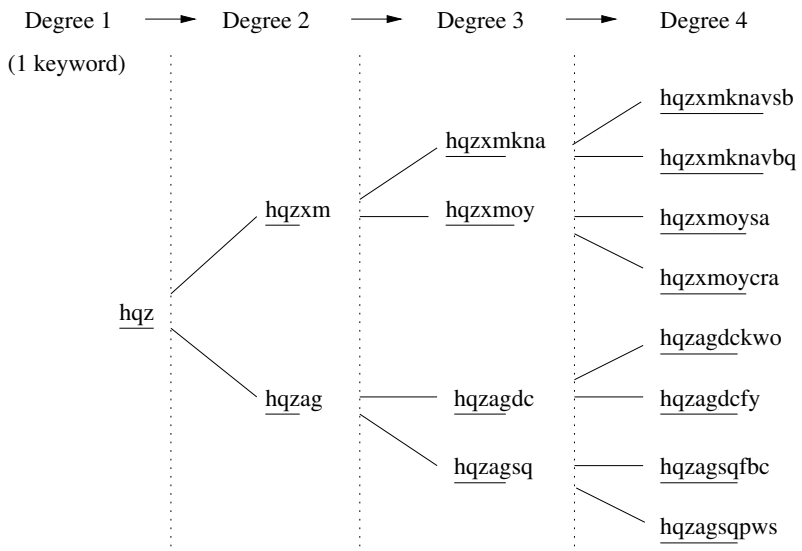



Figure 14: Example of Keyword Generation

network A or B, 455 MP3 files are shared among peers. Thus, at the beginning, there is a file list of 910 files at the OpenNap server. In order to generate a list of keywords which follows a Zipf-like distribution, we consider the following algorithm. First, a keyword is randomly generated as in Fig. 14. From a randomly generated keyword 'hqz' in degree 1, two keywords 'hqzxm' and 'hqzag' in degree 2 are obtained by appending two randomly generated suffixes. Likewise, from the newly generated keywords, four longer keywords in degree 3 are obtained as 'hqzxmkna', 'hqzxmoy', 'hqzagdc', and 'hqzagsq'. Then, we further generated eight additional keywords in degree 4. Thus, we have fifteen keywords. The number of files $N_f$ that a keyword in degree $D$ matches is derived using

$$N_f \approx N_{sum}/D \quad , \tag{3}$$

where $N_{sum}$ is the total number of files and follows a Zipf-like distribution. In our experiments, we first defined a list of ten keywords in degree 1. By appending two or three random characters for each keyword twice, we obtained a list of twenty keywords in degree 2. We generated $10 \times D$ keywords in degree D. Finally, we had 910 keywords in 13 degrees. Then, 910 file names were

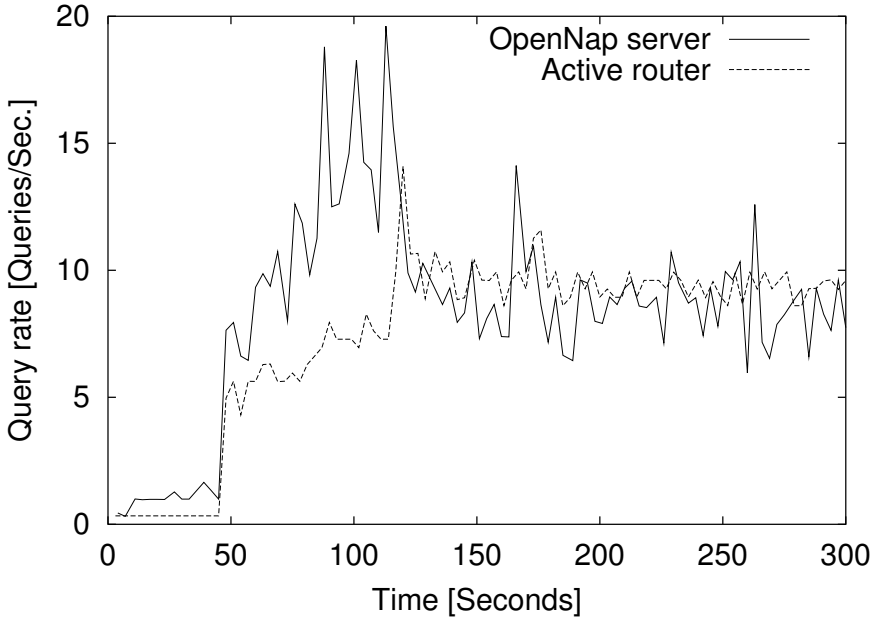



Figure 15: Variation in Query Rate Monitored at Original OpenNap Server and Active Router

generated by adding an extension, such as '.mp3', to those keywords. As a result, each of the ten major keywords matched 91 files, each of the twenty matched about 45, each of the thirty matched about 30, each of the forty matched 23, each of the fifty matched 18, each of the sixty matched 15, each of the seventy matched about 13, each of the eight matched about 11, each of the ninety matched about 10, and finally each of the one hundred matched about 9. Then, files were randomly divided into two groups. Peers in network A shared files in one group and peers in network B did in the other group.

Query rate was used as a metric of load state. We empirically employed 9.5 as the threshold $R_{th}$ in the experiments, and we took the maximum search results $Rmax = 100$ in each query request.

Figure 15 depicts the variation in the number of queries that the OpenNap server received and that captured by the active router at the interface near network B. From the beginning, only one peer in network B was connected to the OpenNap server while there were two peers in network

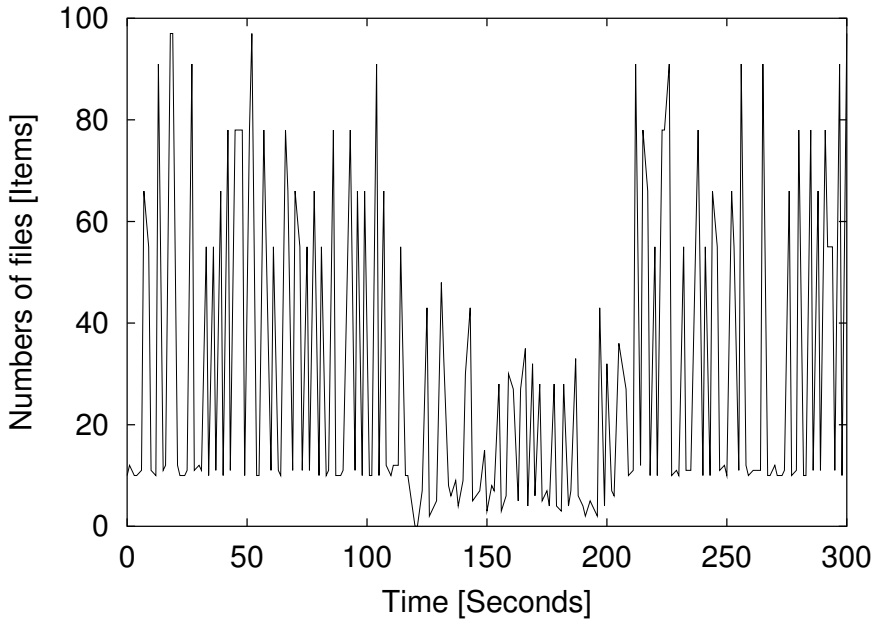



Figure 16: Variation in the Number of Files in Response Messages on Network B

A connected to the OpenNap server. From about 50 seconds, more and more peers from both networks A and B joined the P2P network and connected to the OpenNap server. At about 120 seconds, the query rate passing through the active router exceeded the threshold 9.5. Thus, the active router initiated a virtual cache proxy and introduced a new DNAT policy to the Netfilter, so that query messages from each peer in network B are redirected to the virtual cache proxy. As a result, the OpenNap server received queries only from peers in Network A and the cache proxy in the active router.

Figure 16 illustrates the variation in the number of files in a response message. Although there are variations due to the keywords chosen, we observe that the quality of search results deteriorates after load balancing as mentioned in section 4.3. However, with time, the virtual cache proxy improved its cache by observing query and response messages, and the quality of service improved.

The response time for peers in network B won halved after the redirection, as Fig. 17 illus-

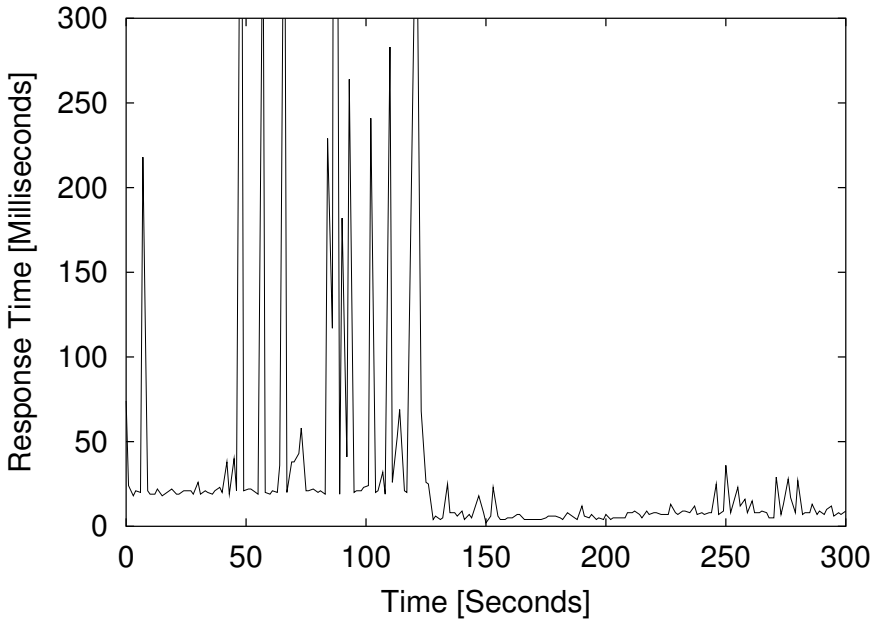



Figure 17: Variation in Response Time on Network B

trates. This means that peers in network B began to receive much better and faster search services than before. Spikes in Fig. 17 corresponds to instants at which new peers joined the P2P network. When new peers join and register themselves, a virtual cache proxy updates its local directory database. It disturbs local searcher in processing queries and response time increases. Such problem can be avoided by optimizing the synchronization mechanism for access to a common data block shared by several threads in an active OpenNap cache proxy.

In this experiment, the load introduced by active packet processing including the virtual cache proxy was very small compared with the processing capability of the equipment; actually, it was imperceptible. However, we need further investigation for the case that there are considerable numbers of flows and that highly complicated packet processing is required at an active router.

# 5 Conclusions

As it has been introduced in the beginning of this thesis, the P2P architecture still has a lot of problems even though it has attracted millions of people. The greatest technical problems in P2P are search-related, including peer discovery and content location. The current solutions to P2P search can be classified into three approaches, i.e., centralized directory, decentralized directory, and query flooding-based pure P2P. By analyzing these three architectures, we showed that they have performance bottlenecks that decrease the quality of search service in P2P applications.

Thus, an active network was introduced. We combined the two logical networks, P2P and active networks, in an active P2P network. The reason we considered to deploy an active network in this study is its programmability in the network layer. In other words, we can tailor the behavior of networks to provide some application-specific specialized services to P2P applications transparently. Active routers can passively collect packet-oriented and network-level information. Then, they can dynamically deploy some helpful service based on monitoring-based presumption.

Based on the concept of an active P2P network, we proposed a framework to distribute load on dynamically changing P2P network actively. We used a Linux-based PC-router as the platform of an active router. A monitor service is deployed at an active router and then configured to monitor P2P packets and measure the performance of P2P applications. When some predetermined conditions are met, the monitor requests a service creation engine to load and activate a certain service to improve the performance of P2P applications by distributing P2P load for example. We described two examples of such services in this thesis. One was “active load balancing for P2P directory servers”, and the other was “active transparent P2P caching”.

We then implemented the latter for an OpenNap application. The results of our experiments showed that our framework can successfully distribute load on the server transparently.

This study focused on the design of the framework as the first step in our research. We will next consider better implementation in which we can attain load distribution while maintaining a high QoS. We need to evaluate the performance of active routers in providing both active and

conventional packet processing services. Furthermore, we consider other P2P applications that benefit from our framework. Our framework is sufficiently general and is expected to contribute to the performance improvement of P2P applications, in a transparent way.

## Acknowledgements

I would like to express my gratitude to my supervisor, Prof. Hideo Miyahara of Osaka University, for his encouragement and valuable comments.

I would like to express my sincere appreciation to Prof. Masayuki Murata for his extensive help and continuous support throughout my studies and in the preparation of this thesis.

I am most grateful to Associate Prof. Naoki Wakamiya who has always given me appropriate guidance and invaluable direct advice.

Finally, I heartily thank my friends and colleagues in the Department of Information Networking, Osaka University for their support.

## References


[1] "BlueCoat (Cacheflow)." available at `http://www.bluecoat.com/`.

[2] J. F. Kurose and K. W. Ross, *Computer Networking (Second Edition)*. Addison-Wesley, 2002.

[3] "KaZaA." available at `http://www.kazaa.com/`.

[4] "JXTA Search Project." available at `http://search.jxta.org/`.

[5] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek and H. Balakrishnan, "Chord: A scalable peer-to-peer lookup service for internet applications," in *Proceedings of the ACM SIGCOMM '01 Conference*, pp. 149–160, Augest 2001.

[6] "OpenNap." available at `http://opennap.sourceforge.net/`.

[7] B. Krishnamurthy, J. Wang and Y. Xie, "Early measurements of a cluster-based architecture for P2P systems," in *Proceedings of ACM SIGCOMM Internet Measurement Workshop*, November 2001.

[8] D. L. Tennenhouse, J. M. Smith, W. D. Sincoskie, D. J. Wetherall, and G. J. Minden, "A survey of active network research," *IEEE Communications Magazine*, vol. 35, pp. 80–86, January 1997.

[9] J. Ioannidis and S. M. Bellovin, "Implementing pushback: router-based defense against DDoS attacks," in *Proceedings of Network and Distributed System Security Symposium*, February 2002.

[10] H. Akamine, N. Wakamiya and H. Miyahara, "Heterogeneous video multicast in an active network," *IEICE Transactions on Communications*, vol. E85-B, pp. 284–292, January 2002.

[11] D. L. Tennenhouse and D. J. Wetherall, "Towards an active network architecture," *Computer Communication Review*, vol. 26, April 1996.

[12] G. Carle, “Tutorial: Active networks - architecture, technology, applications, standardisation,” in *Proceedings of the Second International Working Conference on Active Networks (IWAN 2000)*, October 2000.

[13] J. D. Touch and V. K. Pingali, “Multiple language family support for programmable network systems,” in *Proceedings of the Fifth Annual International Working Conference on Active Networks (IWAN 2003)*, pp. 175–186, December 2003.

[14] S, Alexander, B. Braden, C. Gunter, A. Jackson, A. Keromytis, G. Minden and D. Wetherall, “Active network encapsulation protocol (ANEP).” Active Network Working Group Draft, available at `www.cis.upenn.edu/~switchware/ANEP/`, July 1997.

[15] D. Wetherall, “Active network vision and reality: Lessons from a capsule-based system,” in *Proceedings of 17th ACM Symposium on Operating System Principles (SOSP ’99)*, pp. 64–79, December 1999.

[16] P. Tabery, C. Bachmeir, and X. Li, , “Optimizing mobile network datagram routing through programmable proxying,” in *Proceedings of the 5th IEEE International Conference on Mobile and Wireless Communication Networks*, October 2003.

[17] J. D. Touch and V. K. Pingali, “DataRouter: a network-layer service for application-layer forwarding,” in *Proceedings of the Fifth Annual International Working Conference on Active Networks (IWAN 2003)*, December 2003.

[18] C. Bachmeir, P. Tabery, and J. Kafer, “Towards diverse protection of data streams in programmable application layer overlay networks,” in *Proceedings of IEEE 11th International Conference on Software, Telecommunications and Computer Networks, SoftCOM’03*, October 2003.

[19] C. Bachmeir and P. Tabery, “Provision of transparent application layer diverse protection services based on lightweight software agents in p2p networks,” in *Proceedings of 15th IASTED*

*International Conference on Parallel and Distributed Computing and Systems, PDCS'03*, November 2003.

[20] S. C. Chen, M.. L. Shyu, I. Gray, and H. Luo, "An adaptive multimedia transmission protocol for distributed multimedia applications," in *Proceedings of the 5th International Workshop on Multimedia Network Systems and Applications (MNSA'2003), in conjunction with The 23rd International Conference on Distributed Computing Systems (ICDCS'03)*, May 2003.

[21] T. Yamada, N. Wakamiya, M. Murata and H. Miyahara, "Implementation and evaluation of video-quality adjustment for heterogeneous video multicast," in *Proceedings of the 8th Asia-Pacific Conference on Communications (APCC'02)*, September 2002.

[22] B. Yang and H. Garcia-Molina, "Efficient search in peer-to-peer networks," in *Proceedings of the 22nd IEEE International Conference on Distributed Computing Systems (ICDCS'02)*, July 2002.

[23] R. Gunther, L. Levitin, B. Shapiro and P. Wagner, "Zipf's law and the effect of ranking on probability distributions," *International Journal of Theoretical Physics*, vol. 35, pp. 395–417, 1996.

[24] M. Bossardt, T. Egawa, H. Otsuki, and B. Plattner, "Integrated service deployment for active networks," in *Proceedings of Fourth Annual International Working Conference in Active Networks (IWAN 2002)*, December 2002.

[25] M. Bossardt, L. Ruf, R. Stadler, and B. Plattner, "A service deployment architecture for heterogeneous active network nodes," in *Proceedings of 7th Conference on Intelligence in Networks (IFIP SmartNet 2002)*, April 2002.

[26] M. Bossardt, A. Muhlemann, R. Zurcher, and B. Plattner, "Pattern based service deployment for active networks," in *Proceedings of the Second International Workshop Active Network Technologies and Applications (ANTA 2003)*, May 2003.

[27] M. Fry and A. Ghosh, "Application level active networking," *Computer Networks (Amsterdam, Netherlands: 1999)*, vol. 31, no. 7, pp. 655–667, 1999.

[28] I. Liabotis, O. Prnjat, and L. Sacks, "SORD: self-organizing resource discovery protocol for ALAN," in *Proceedings of the Fifth Annual International Working Conference on Active Networks (IWAN 2003)*, December 2003.

[29] R. Keller, B. Plattner, "Self-configuring active services for programmable networks," in *Proceedings of the Fifth Annual International Working Conference on Active Networks (IWAN 2003)*, December 2003.

[30] Y. S. Wang, J. Touch , "Application deployment in virtual networks using the x-bone," in *Proceedings of 2002 DARPA Active Networks Conference and Exposition (DANCE'02)*, May 2002.

[31] C. K. J. Quittek, "Remote service deployment on programmable switches with the ietf snmp script mib," in *Proceedings of 10th IFIP/IEEE International Workshop on Distributed Systems: Operations and Management, DSOM '99*, pp. 135–147, October 1999.

[32] "NetFilter." available at `http://www.netfilter.org/`.

[33] "TCPDUMP/LIBPCAP." available at `http://www.tcpdump.org/`.

[34] "JPCAP." available at `http://sourceforge.net/projects/jpcap/`.