社団法人　電子情報通信学会
THE INSTITUTE OF ELECTRONICS,
INFORMATION AND COMMUNICATION ENGINEERS

信学技報
TECHNICAL REPORT OF IEICE.

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

侍　　建港† 　若宮　直紀† 　村田　正幸††

† 大阪大学 大学院情報科学研究科
†† 大阪大学 サイバーメディアセンター
E-mail: †{shi,wakamiya}@ist.osaka-u.ac.jp, ††murata@cmc.osaka-u.ac.jp

あらまし　P2P 型アーキテクチャは，ネットワーク規模に対する拡張性や対故障性などから，従来のサーバ-クライアント型アーキテクチャの問題を解決するネットワークパラダイムとして注目を集めている．しかしながら，ディレクトリサーバやブートストラッピングノードなど，未だその機構の一部をサーバに依存しているため，それらサーバがボトルネックとなり，十分な性能を得られない．本研究では，アクティブネットワーク技術にもとづく動的な負荷分散のためのフレームワークを提案し，フレームワークの適用例として OpenNap のための検索負荷分散機構を実現した．アクティブノードは，自身を通過する P2P トラヒックを観測することにより，ボトルネックとなるサーバを特定し，必要に応じて P2P アプリケーションの QoS を向上するためのサービスを提供する．アクティブノードによるサービス提供は透過的であり，P2P アプリケーションやユーザがその存在を意識する必要はない．実験により，アクティブノードによる検索負荷分散が達成され，提案したフレームワークが実用的かつ有効であることを示した．
キーワード　負荷分散，Peer-to-Peer，アクティブネットワーク

Active Load Distribution Mechanism for P2P Searching

Jiangang SHI†, Naoki WAKAMIYA†, and Masayuki MURATA††

† Graduate School of Information Science and Technology, Osaka University
†† Cybermedia Center, Osaka University
E-mail: †{shi,wakamiya}@ist.osaka-u.ac.jp, ††murata@cmc.osaka-u.ac.jp

Abstract　P2P network architecture draws much attention for its features of scalable, robust, self-organizing, and distributed. However, most of the current P2P architectures still rely on servers, e.g., directory servers and bootstrapping nodes. They are bottlenecks and single points of failure. In this study, we propose a framework constructed on active network technologies for dynamic and transparent search load distribution. Active nodes at network layer monitor and analyze P2P traffics passing through them, then decide whether it is necessary to conduct some user defined services to improve QoS of P2P applications. We implemented “active OpenNap cache proxy” as an example of an application of our framework. Through experiments, we showed that our proposal was effective to distribute load on a P2P network depending on predefined conditions in a transparent way.
Key words　load distribution, Peer-to-Peer, active network

1. Introduction

The success of P2P file sharing applications recalled us a fact that personal computer at the edge of the Internet can be not only a client but also a server. When they are connected together and serve each other, the most powerful service system can be built, which can provide almost unlimited computation ability and storage space.

However, the effectiveness and usefulness of P2P applications largely depend on how each peer discovers other peers and how it locates the desired resources in a P2P network, which is called P2P searching in this paper. One solution is to deploy directory servers to maintain a directory database for all peers or for only a group of peers. It brings advantages of the client-server model such as convenience of management and control, guaranteed QoS for peers, but it also

introduces disadvantages of servers including bottlenecks and single points of failure. Another solution uses "query flooding" mechanism, in which P2P messages are relayed peer by peer. It succeeds in avoiding bottlenecks of directory servers with some self-organization protocols. However, even flooding-based P2Ps are often called "pure P2Ps", there are still bottlenecks of peers and single points of failure. They are bootstrapping nodes, which introduce peers that a new peer should be connected with to join a P2P network. Thus P2P traffic load is not fairly distributed over a P2P network. Mechanisms for server load distribution have been proposed. For example, in [1], servers are chained together to distribute search load in OpenNap P2P application. However, since most of them are configured in a static way, they cannot adapt to dynamically changing network environment.

If an underlying network is aware of conditions and demands of overlying applications and provide application-oriented services, it is helpful to improve QoS of networked applications in an efficient way. Active network technologies provide a programmable infrastructure for a variety of network applications [2]. Active nodes or active router, 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 network layer, but they can apply application-specific manipulation to packet payload if needed. Sample applications of active network technologies include DDoS defense mechanisms [3], multimedia broadcast [4], and so on.

This paper proposes a framework based on active network technologies [2, 5] to dynamically distribute load between servers or network links in a P2P network. 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 gathered 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 the load-distribution on a P2P network, without any message exchanging.

The rest of paper is organized as follows. First in section 2., an architecture of our active P2P network is introduced. Then in section 3., the framework is proposed and described in detail. After that, in section 4. an example of application is implemented and evaluated on an actual system. Finally section 5. summarizes the paper and describes some further research works.

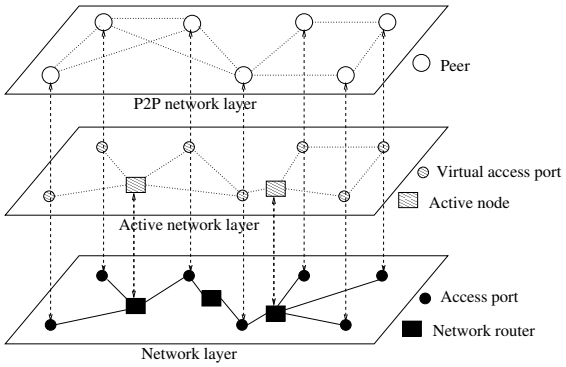

Figure 1 Active P2P Network

2. Active P2P Networks

In recent years, some ideas about deploying services at network layer to improve the performance and behavior of networked applications have been proposed [6, 7]. The active network is a technology that makes networks aware of applications and enables networks behave in accordance with application's demands.

In this study, we consider such a layered network of P2P and active networks architecture as Fig. 1. An IP network consists of nodes, hosts, and links. More specifically, they are routers, ports of network interface of hosts, and physical links. Some routers in an IP network are also active routers. A P2P network consists of peers, i.e., hosts. An active network laying between them consists of active routers and virtual access ports. Since an active network is transparent, both an overlying P2P network and an underlying IP network are not aware of it. Thus, we call ports on an active network virtual access ports. Active routers provide conventional routing functions to those packets that are irrelevant to the active load distribution. Active routers have chances to collect enough information of P2P traffic in a passive way. Then, they can analyze the behavior of P2P applications. Furthermore, by dynamically introducing active services, they process P2P traffic with some application-oriented policy.

3. Active Load Distribution for P2P Applications

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 2 illustrates the architecture of an active router in our framework. It follows the architecture of a typical active node having the NodeOS, which manages the node resources such as link bandwidth, CPU cycles, and storage; the Execution Environments (EEs), which provide API of using

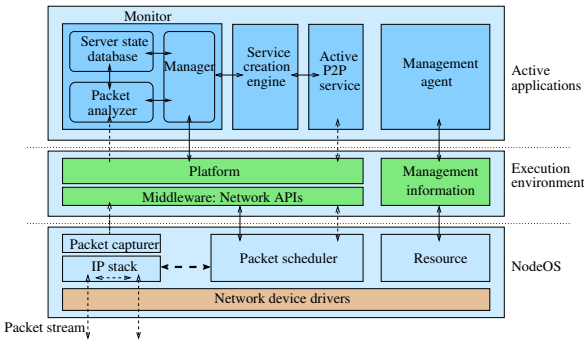

Figure 2 Modules in Active Router

NodeOS 's resources or parameters of other needed resources; and the Active Applications (AAs), which program the virtual machine on an EE to provide an end-to-end service. For active router, packet scheduling and routing function module is the basic element of NodeOS. We propose to deploy "service creation engine" as a proactive P2P service at active router, which is to load other active services. "Monitor" is an active application in charge of collecting information from P2P traffic. It can be loaded by service creation engine as other common user defined "active P2P services" in Fig. 2.

3.2 Monitor, Measurement and Decision Mechanisms

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

Next, by checking payload of packets and comparing with P2P protocols, a P2P packet analyzer can identify the type of P2P messages with P2P protocol knowledge. For example, each message of OpenNap protocol [1] to or from a server are in the form of $< length >< type >< data >$, where $< length >$ specifies the length in bytes of the $< data >$ portion of the message and $< type >$ specifies the command type. After that, the P2P packet analyzer stores P2P packet information into a server state database, which includes IP addresses, port numbers, P2P message types and so on.

Then, a manager in a monitor application identifies a bottleneck peer, to which load concentrates on. Since there is no any direct way to get information of 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 in a passive way. Metrics of load include the response time, the number of search results, and the query rate.

When the manager identifies a bottleneck based on predefined policies, it requests service creation engine to introduce a corresponding active P2P service to the active router. In the case of centralized mechanism [8], active service deployment is accomplished using SNMP as a management protocol. Being requested by a management station, a service creation engine retrieves an active service from a code repository and starts the service. After service deployment, incoming packets satisfying some pre-determined conditions are forwarded to the active P2P service by some packet redirection policies executed in packet scheduling modules in NodeOS. Management service at local active router should be in charge of inserting or deleting such packet redirection policies.

An active P2P service processes packets in a transparent way. 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 in order 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 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.

4. Active OpenNap Cache Proxy

In this section, we implemented an application based on OpenNap P2P protocol, "active OpenNap cache proxy", as a prototype to investigate the practicality and the effectiveness of our framework.

4.1 OpenNap protocol

OpenNap clones most of Napster's protocols. OpenNap server acts as a directory server for a cluster of peers. A peer connects to a server and notifies 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 are one or more records that match the query, the server returns those records to the requesting peer as a response message. Communications between peer and server are "keep-alive TCP session" based. When the connection is broken out by peer or because of some network problems,

a server releases resource used by the peer by deleting the peer's record in its database.

4. 2 Architecture of Active OpenNap Cache Proxy

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.

As NodeOS, we chose linux platform and Java virtual machine provides an execution environment. NetFilter in Linux 2.4 kernel was chosen to realize the function of the packet scheduler module. Usually, it only passes incoming packets to an output network interface based on routing policy. Those 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. PCAP and its Java's wrapper jpcap were used as a packet capturer.

A packet analyzer thread in monitor in in Fig. 2 analyzes captured P2P packets and inserts history record of target server peers in a server state database. The database consists of two types of tables, i.e., server list and query-response timestamp. A server list is a list of IP address 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. A manager is in charge of controlling all of the other modules in monitor.

A virtual cache proxy is built on an OpenNap server program, but it has caching and forwarding functions additionally as shown in Fig. 3. It is loaded as an active P2P service module by a service creation engine in Fig. 2. 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 server instead of client peer redirected to this cache proxy. A local searcher of a virtual cache proxy receives queries from client peers through a TCP socket and examines its local directory database. The database is updated when it receives registration messages from client peers. The database consists of user name, link type, 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. A cache is used to store those response messages received from the remote server. They are further explained in detail in the next section.

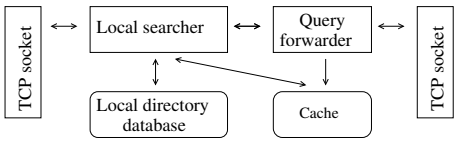

Figure 3 Modules in Virtual Cache Proxy

4. 3 Mechanisms of Active OpenNap Cache Proxy

First, PCAP captures those packets matching the filtering policy "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 address is the address of a server. If $< type >$ is "201" or "202", the message is a response. Therefore, the destination address 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 address 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.

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 were supported to make decisions. With response-time metric, response time T_R of every query is calculated as $T_R = T_{fr} - T_q$, in which T_q is the timestamp of a query packet, and T_{fr} is the timestamp of the first response packet. Average of T_R, as T_{Ravg}, is periodically calculated for each server in the server table. Then it is compared with the predefined high threshold T_{RthH}. If T_{Ravg} exceeds T_{RthH}, a virtual cache server for the server will be invoked 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 ini-

tialized to zero and the query rate of the server is derived. The query rate is calculated as, $R = N/T_s$, 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 high threshold R_{thH} to identify the bottleneck server.

Finally, a virtual cache server is loaded for the server by manager as a JAVA thread. For a newly activated virtual cache server to deal with query messages sent to the original server, the manager introduces a new DNAT rule into Netfilter. IPTables 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 the on-going connections. In our experiments, peers redirected to the virtual cache server first disconnect their connections and re-establish 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 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 non-zero, their filenames 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, results are merged together, it is fit to $Rmax$, and finally it is 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 the extra delay. A response message is sent to the peer in the preceding steps as described. 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 the 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 instant when a new record is deposited in the cache is also kept in the cache. For each item of search result from cache, the local searcher will compares its timestamps with the current time. If more than T_{hd}, a predefined threshold, has passed, that item of result will be ignored and the corresponding record is removed from the cache.

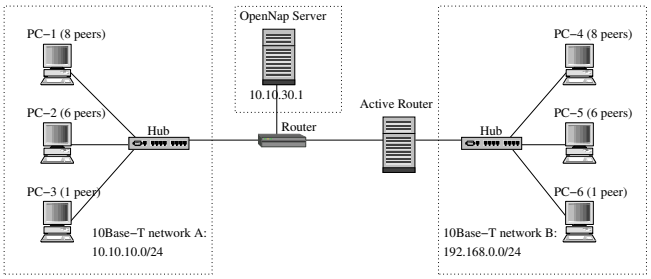

Figure 4 Experimental Environment

4. 4 Experiment and Evaluation

We conducted experiments on the system illustrated in Fig. 4 to verify the practicality and effectiveness of our framework from viewpoints of load-balancing and the 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 sub-networks that are connected with each other by an active router and a general IP router. Either network A or B has 15 peers running at three PCs. Each peer joins the P2P network at random and sends query messages to a server once every 1.5 seconds. Either in network A or B, 455 files are shared among peers. Thus, at the beginning, there is a file list of 910 files at the OpenNap server. We prepare a keyword list for those files. The number of files that a keyword matches follows the Zipf-like distribution. In our experiments, we first defined a list of keywords. Then, 910 file names were generated based on those keywords. As a result, each of ten major keywords matched 91 files, each of twenty matched about 45, each of thirty matched about 30, each of forty matched 23, each of fifty matched 18, each of sixty matched 15, each of seventy matched about 13, each of eight matched about 11, each of ninety matched about 10, and finally each of 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.

The query rate was used as a metric of load state. We empirically employed 9.5 as the threshold in the experiments. We took the maximum search results $Rmax = 100$ in each query request.

Figure 5 depicts the variations of the number of queries that the OpenNap server received and that captured by active router at the interface near to network B. First, only one peer in network B was connected to OpenNap server while there were two peers in network A connecting to OpenNap server. From about 50 seconds, more peers joined the P2P

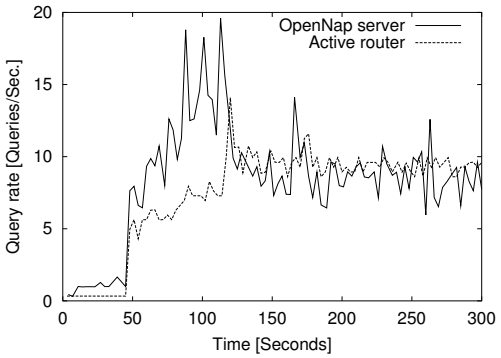

Figure 5 Variation of Query Rates

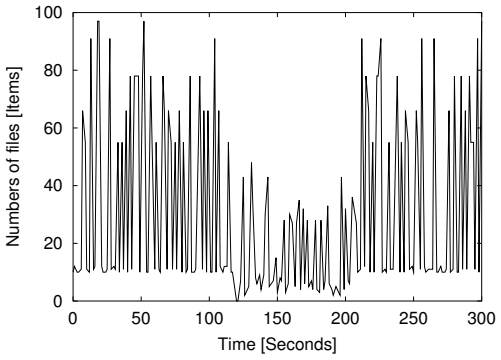

Figure 6 Variation in the Number of Files in Response Messages on Network B

network and were connected to the OpenNap server. At about 120 seconds, the query rate passing through active router exceeded the threshold 9.5. Thus, the active router initiated a virtual cache proxy and introduced a new DNAT policy to Netfilter. As a result, the OpenNap server received queries only from peers in Network A and the cache proxy in the active router.

Figure 6 illustrates the variation in the number of files in a response message. Although there are variations due to keywords chosen, we observe that the quality of search results becomes insufficient after the load-balancing as we mentioned in section 4. 3. However, as time passes, the virtual cache proxy improved its cache by observing query and response messages, and the quality of service was improved.

Response time for peers in network B reduced to the half after the redirection, as Fig. 7 illustrates. It means peers in network B began to receive much better and faster search services than before. Spikes in Fig. 7 correspond instants that 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 the response time increases. Such problem can be avoided by optimizing synchronization mechanism for access to a common data block shared by several threads in an active OpenNap cache proxy.

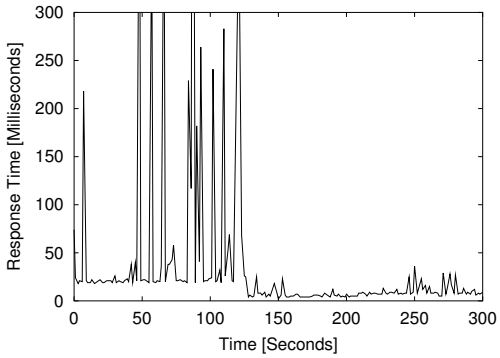

Figure 7 Variation of the Response Time on Network B

5. Conclusions

We proposed a framework to distribute load on dynamically changing P2P network in a transparent way. We implemented an active OpenNap cache proxy as the prototype of application of our framework. Through experiments, the practicality and usefulness of our framework was verified. However, we need to further consider the efficient implementation that attains the load distribution while keeping the level of the quality of service. Furthermore, we will consider other types of applications of our framework, and investigate methods to efficiently and effectively measure the performance of P2P network.

References

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

[2] 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, no. 1, pp. 80–86, January 1997.

[3] J. Ioannidis and S. M. Bellovin, "Implementing pushback: router-based defense against DDoS attacks," in *Proceedings of NDSS 2002*, February 2002.

[4] 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.

[5] D. Wetherall, U. Legedza and J. Guttag, "Intoducing new Internet services: why and how," *IEEE NETWORK Magazine, Special Issue on Active and Programmable Networks*, July 1998.

[6] J. D. Touch and V. K. Pingali, "DataRouter: A Network-Layer Service for Application-Layer Forwarding," in *Proceedings of IWAN 2003*, December 2003, pp. 113–124.

[7] C. Bachmeir, P. Tabery, and J. Kafer, "Towards diverse protection of data streams in programmable application layer overlay networks," in *Proceedings of SoftCOM'03*, October 2003.

[8] C. Kappler J. Quittek, "Remote service deployment on programmable switches with the ietf snmp script mib," in *Proceedings of DSOM '99*, October 1999, pp. 135–147.