Active Load Distribution Mechanism for P2P Application

Jiangang Shi Naoki Wakamiya Masayuki Murata Hideo Miyahara
Graduate School of Information Science and Technology, Osaka University
1-3 Machikaneyama, Toyonaka, Osaka 560-8531, Japan
{shi,wakamiya,murata,miyahara}@ist.osaka-u.ac.jp

Abstract

The peer discovery and contents location usually consume much resource in most of P2P applications. We therefore propose a framework constructed on active network technology to deal with the problem in this study. Active nodes at network layer are able to monitor and analyze P2P traffics which coincide with the prescribed filtering conditions. Then, an active node decides whether it is necessary to conduct some user defined services to improve QoS of P2P network. Two examples of application of this framework are further introduced. One is "Active Load-Balancing for P2P Directory Servers". When an active node considers a directory server is the bottleneck, it tries to redirect client peers to other directory servers. The other is "Active Transparent P2P Caching", in which active nodes answer query requests instead of actual P2P directory servers. We implemented the latter and showed that our proposal was effective to distribute load on P2P network depending on predefined conditions. With the framework proposed here, ones can easily operate desired services in P2P network with much of flexibility.

1 Introduction

The attraction of peer-to-peer (P2P) is that users, called peers in P2P applications, can exchange information directly between themselves, without mediation of servers. As a result of direct communications among peers, we can avoid the concentration of load on some specific point of a network. 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.

Several solutions have been proposed to deal with the problem, such as "Centralized Directory" [1] as in Napster, "Decentralized Directory" as in KaZaA [2] and JXTA search [3], "Query Flooding" as in Gnutella, and "Distributed Hash Table (DHT)" based mechanisms as in Chord [4]. They are helpful for a peer to find neighbors and locate resources, but they still rely on some centralized nodes. For example, Napster or KaZaA need directory servers to maintain directories of contents shared by all of client peers connecting to directory servers. Even in Gnutella, it still needs a boot-strapping node to introduce new peers. In order to place those centralized nodes dynamically to satisfy the practical situation, some complex algorithms are needed. In a cluster-based P2P network [5], at least one leader peer is selected among peers in every cluster. When the leader leaves off, a new leader peer is appropriately selected. 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.

This paper presents a framework based on Active Network [6, 7] to dynamically distribute load on P2P servers over an active network in a transparent way. 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-balancing. 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-balancing on a P2P network, without any message exchanging.

The rest of paper is organized as follows. First in section 2, Active Network is introduced briefly. Then in section 3, the framework we proposed is described in detail. Two sample applications based on the proposed framework are given at section 4 and one implementation is shown in section 5. Finally section 6 summarizes the paper and describes some further research works.

2 Active Network Technology

Active Network technology provides a programmable infrastructure for a variety of network applications [6]. There is not only one way to implement active network, but there are still common features between those different architectures. Active nodes, 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 [8], multimedia broadcast [9], and so on.

Active nodes have three major components: the Node Operating System (NodeOS), which manages the node resources such as link bandwidth, CPU cycles, and storage; the Execution Environments (EEs), each one of which implements a virtual machine that interprets active packets

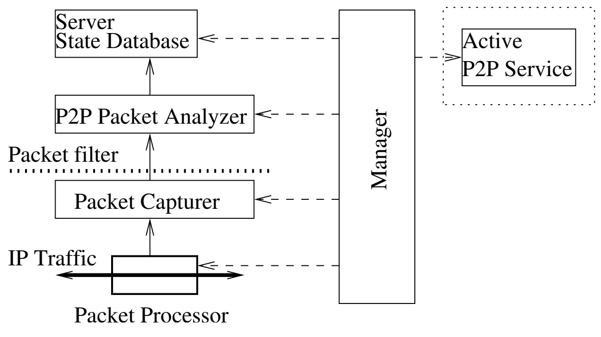

Figure 1: Modules in Active Router

that arrive at the node; and the Active Applications (AAs), which program the virtual machine on an EE to provide an end-to-end service. End systems that host end-user applications are also considered as active nodes having the same structure [9].

Mainly, there are two mechanisms to load a program onto an active node. Capsule model uses ANEP (Active Network Encapsulation Protocol) protocol [10] to encapsulate a program into IP packets and marks those IP packets with a special IP Protocol ID. When such packet arrives at an active node, the node interprets and applies a program to the packet. Another is Programmable switchlet model. It uses an additional communication channel to retrieve a program from a deposit or a neighboring active node and packets carry data and parameters for computations.

The proposal in this research needs to be implemented on Programmable switchlet-based active network, because usually an application in the P2P world is too large to fit in a single packet.

3 Active Load Distribution Mechanism

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

As Fig. 1 illustrates, an active router consists of six modules. “Packet Processor” and “Packet Capturer” belong to NodeOS functions. “P2P Packet Analyzer”, “Server State Database”, and “Manager” are active applications. EE (Execution Environment) provides those active applications with the interfaces to access “Packet Processor” and “Packet Capturer” in NodeOS. “Active P2P Service” is also an active application. It is a dynamic module loaded from this active node or downloaded from remote active nodes. An active router works in the following way.

First, a packet processor forwards incoming packets to desired output ports. A packet capturer monitors all packets passing through the packet processor and copies some out of them to a P2P packet analyzer. 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. We also need a self-study algorithm to deal with other cases. Next, a P2P packet analyzer investigates P2P packets and obtains information such as IP address, port number, packet type, and P2P message type. That information is stored in a server state database.

Then, a manager 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 [11]. Although they are not precise, they are practical. For example, an active router can take the response time of a server to measure the load. If it finds that response time becomes much longer than before or it exceeds the pre-determined threshold, it considers the server is now a bottleneck.

When the manager identifies a bottleneck based on predefined policies, it introduces a corresponding active P2P service to the active router. 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 manager considers that a server is no longer a bottleneck or underutilized, it stops 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 Two Sample Applications

In this section, we briefly introduce two sample applications that benefit from our framework.

4.1 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 [12] uses a “MetaServer” as a portal to direct peers to different OpenNap servers, but it is statically arranged and peers must be configured to use a specific portal.

In this proposal, 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 though it to find servers and estimate their loads. The response time can be chosen as a parameter to measure the quality of services of a directory server in this case. If the response time at a server becomes longer than before, an active router forwards queries to another unloaded server. A manager activates a load-balancing service and redirects P2P packets to the service by, for example, rewriting the destination address of incoming packets. The load-balancing service decides a server to which packets are forwarded. For response messages to reach client peers through the active router, the source address is also changed to IP address of the load-balancing service.

4.2 Active Transparent P2P Caching

Contents popularity in P2P networks follows a Zipf-like distribution [13], which implies that caching in P2P network is still a very efficient way to avoid redundant traffic as being proved its power at the web-based Internet. However, to take advantage of caching, cache servers should be carefully placed in a P2P network taking into account the distribution of peers, resources, and their popularity. P2P traffic can be categorized into two: protocol messages such as queries, responses, and other commands 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.

In this example, an active router first monitors 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. 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 in a transparent way.

In the following section, we implemented this type of application as an example.

5 Implementation of Active OpenNap Cache Proxy

In this section, we implemented an application to investigate the practicality and the effectiveness of our framework.

5.1 Model

OpenNap [12] [14] follows Napster's protocol. There is 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 the peer shares with the others. Based on registered information, the server maintains a user information database. When a peer searches a file, it 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.

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 cache proxy as an active application to undertake some portion of query traffic. The cache proxy maintains the local cache of directory information and answers queries based on the cache.

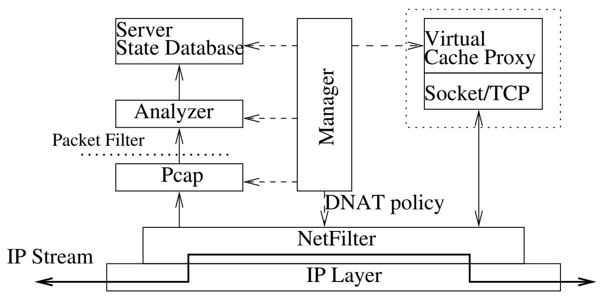

Figure 2: Active Router on Linux

Figure 2 illustrates the architecture of an active router in our implementation. NetFilter in Linux 2.4 kernel was chosen to realize the function of the packet processor module in Fig.1. 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 Translators) policy [15]. "PCAP" [16] and its Java's wrapper "jpcap" [17] were used as a packet capturer. A thread of "Analyzer" in Fig. 2 acts as P2P packet analyzer and it manipulates "Server State Database". The database consists of two types of tables. 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. "Manager" is in charge of controlling all of the other modules.

Virtual cache proxy is loaded as an active P2P service module illustrated in Fig. 1. Basically, a virtual cache proxy is built on a basic OpenNap server. It provides directory service to all of its client peers transparently. At the same time, it is a general client peer to a remoter OpenNap server. It queries the remote OpenNap server instead of client peer redirected to this cache proxy. A virtual cache proxy consists of four modules as illustrated in Fig. 3. Local searcher receives queries from client peers through a TCP socket and examines the local directory database. The

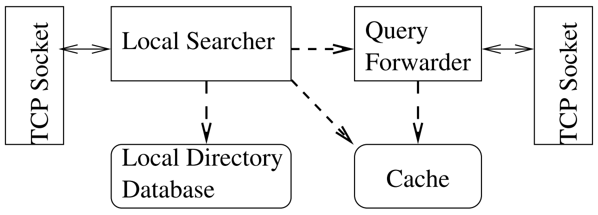

Figure 3: Modules in Virtual Cache Proxy

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. 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.

5.2 Mechanisms

To evaluate the load state of an OpenNap server, we employed query rate and response time as metrics. Query messages and corresponding response messages are captured and examined at an active router. Each message of OpenNap protocol [12] to or from a server are in the form of $< length >< type >< data >$, where $< length >$ and $< type >$ are 2 bytes-length each. $< length >$ specifies the length in bytes of the $< data >$ portion of the message. $< type >$ specifies the command type. For example, “200” means a query message, “201” is a response search result, and “202” is a notification of the end of a series of “201” messages from server.

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 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 module 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 generates and 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.

1. Response-Time Metric

   Response time RT of every query is calculated with the following function:

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

   in which T_q is the timestamp of a query packet, and T_{fr} is the timestamp of the first response packet. Average of RT, as RT_{av}, is periodically calculated for each server in the server table. Then it is compared with the predefined high threshold RT_{thH}. If RT_{av} exceeds RT_{thH}, a virtual cache server for the server will be invoked in the next step.

2. Query-Rate Metric

   The manager counts the number of query packets for each of servers. Every time the count reaches the predetermined threshold N, it is initialized to zero and the query rate of the server is derived. The query rate is calculated as,

   $$QR = N/(T_{fq} - T_{lq}) \quad , \tag{2}$$

   where T_{fq} is the timestamp of the first query packet among N and T_{lq} is the timestamp of the N-th query packet. Average query rate QR_{avg} is computed periodically and compared with high threshold QR_{thH} as in the response-time metric.

Finally, a virtual cache server is loaded for the server by manager as a JAVA thread. It is built on an OpenNap server program, but it has caching and forwarding functions additionally. 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 [15] 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 as shown later.

When a virtual cache server receives a query Q, it processes the query in the following steps. 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.

Step 1: The virtual cache proxy 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.

Step 2: 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.

Step 3: 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. As a result, peers receive a small result until the virtual cache proxy collects sufficient amount of information in its local database and cache. It will be shown in section 5.3. 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, a predefined threshold, has passed, that item of result will be ignored and the corresponding record is removed from the cache.

5.3 Experiments

We conducted experiments on the system illustrated in Fig. 4 and verified 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 two sub-networks that are connected with each other by an active router. Network A has one OpenNap server and seven clients, Peer 1 - 7. Network B has six clients, Peer 8 - 13. Each peer sent query messages to server once every two seconds. A server has information of 120 files that the P2P network has. 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, 120 file names were generated based on those keywords. As a result, each of six major keywords matched twenty files, each of ten matched twelve, each of twenty matched six, each of thirty matched four, and finally each of sixty matched two. 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.

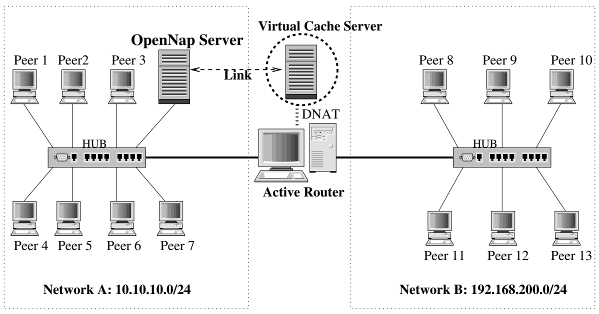

Figure 4: Experimental Environment

The query rate was used as a metric of load state. We empirically employed $QR_{thH} = 10.0$ in the experiments, and we took the maximum search results $Rmax = 100$ in each query request as default. Before 160 seconds, the active router behaves as a conventional router. It becomes active at 160 seconds.

Figure 5 depicts the variations of the number of queries that the OpenNap server and the virtual cache server received. At 170 seconds, the active router decided to initiate 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. All peers in network B first lost connections to the OpenNap server. At 220 seconds, they re-established connections to the same OpenNap server, but actually their connections were directed to the virtual cache proxy.

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 5.2. However, as time passes, the virtual cache proxy retrieved information from the original server through forwarding query messages, and the quality of service was improved.

Peers in network B suffered from the longer response time after the redirection, as Fig. 7 illustrates. One of reasons is that only one thread managed a cache table for all queries that the active router receives. In addition, to avoid the competition and inconsistency, the cache table was locked when a new record was deposited in. It disturbed local searcher in investigating the cache. We can expect the response time can be reduced when programs are fully optimized, and more powerful equipment is employed. We also need to consider further efficient architecture in the next stage of the proof of concept.

6 Conclusion and Future Work

We proposed a framework to distribute load on dynamically changing P2P network in an active way. A sample application has been implemented with OpenNap protocol at Linux platform. The results of experiments showed that

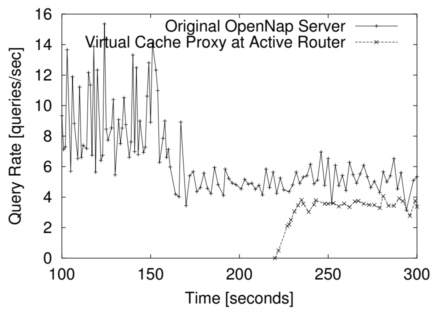

Figure 5: Variation of query rates at original OpenNap server and virtual cache proxy

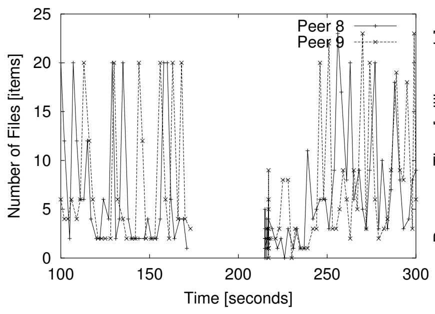

Figure 6: Variation in the number of files in response messages on network B

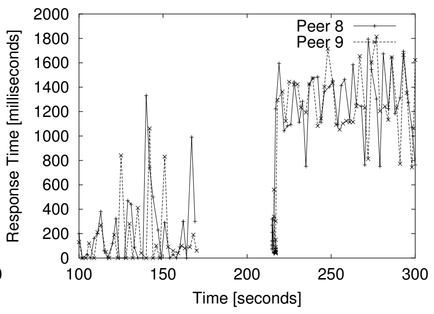

Figure 7: Variation of the response time on network B

our framework can successfully distribute the load on the server in a transparent way. 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.

Acknowledgment

This work was partly supported by the Special Coordination Funds for Promoting Science and Technology of the Ministry of Education, Culture, Sports, Science and Technology of Japan.

References

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

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

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

[4] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek and H. Balakrishnan, "A scalable peer-to-peer lookup service for Internet applications," in *Proceedings ACM SIGCOMM*, 2001.

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

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

[7] 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.

[8] J. Ioannidis and S. M. Bellovin, "Implementing pushback: router-based defense against DDoS attacks," in *Proceedings of Network and Distributed System Security Symposium*, 1997.

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

[10] S. Alexander, B. Braden, C. Gunter, A. Jackson, A. Keromytis, G. Minden and D. Wetherall, "Active network encapsulation protocol (ANEP)," 1997. available at `http://www.cis.upenn.edu/?switchware/ANEP/`.

[11] 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)*, 2002.

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

[13] 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.

[14] "OpenNap-NG." available at `http://opennap-ng.sourceforge.net/`.

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

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

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