International Journal of Networking and Computing – www.ijnc.org
ISSN 2185-2839 (print) ISSN 2185-2847 (online)
Volume 1, Number 2, pages 157–177, July 2011

OpenWeb: Seamless Proxy Interconnection at the Switching Layer

YOSHIO SAKURAUCHI
Graduate School of Science and Engineering, Ritsumeikan University,
1-1-1 Noji-higashi, Kusatsu, Shiga 525-8577, Japan

RICK MCGEER
Hewlett-Packard Laboratories Palo Alto,
1501 Page Mill Road, Palo Alto, CA 94304, USA

HIDEYUKI TAKADA
College of Information Science and Engineering, Ritsumeikan University,
1-1-1 Noji-higashi, Kusatsu, Shiga 525-8577, Japan

Received: January 27, 2011
Revised: May 19, 2011
Accepted: June 20, 2011
Communicated by Sayaka Kamei

Abstract

In recent years, the amount of Internet traffic has been growing beyond the enhancement of its capacity. Moreover the amount of published information is also growing at an exponential rate. Consequently, there are the demands on performance, robustness, and low latency for a worldwide Internet population. To solve these problems, traditional solutions have led to web proxy cache systems. However, to use such systems, administrators and/or clients are required to do some tedious and error-prone operations because cache systems generally need to be accessed through layer 4-7 scripts and commands, such as the *route* command on Posix systems, and usually, manual configuration or JavaScript code for a web proxy. If cache systems work at the switching layer (layer-2), administrators can introduce the system just by inserting it into the network and clients can use the system transparently. This paper describes OpenWeb, a layer-2 redirection engine implemented as an application of the OpenFlow switch architecture. New open protocols at the switching layer now enable far more robust and seamless packet redirection, without user configuration or unreliable scripts. In addition, performance evaluations compared with traditional systems and simulations run in random networks show that OpenWeb is clearly beneficial.

Keywords: Layer-2 Redirection, Cache System, OpenFlow

1 Introduction

In recent years, the amount of Internet traffic has been growing beyond the enhancement of its capacity. Moreover the amount of published information is also growing at an exponential rate. Consequently, there are the demands on performance, robustness, and low latency for a worldwide Internet

population. These problems are often addressed as exaflood[20] and information explosion[3]. In Japan, the Ministry of Education, Culture, Sports, Science and Technology has supported researches related to these phenomena[1].

To solve these problems, traditional solutions have led to web proxy cache systems (hereinafter just referred to as cache systems). Cache systems process requests from clients on behalf of web servers. Currently, cache systems are generally accessed through layer 4-7 scripts and commands, such as the *route* command on Posix systems, and usually, manual configuration or JavaScript code for proxy settings. As another approach, DNS records which are used to resolve web servers' IP address are often rewritten to make clients use cache systems such as CDNs (Contents Delivery Network). Thus, administrators and/or clients are required some tedious and error-prone operations to use these cache systems. This is far from robustness.

If cache systems work at the switching layer (layer-2), administrators can introduce the system just by inserting it into the network and clients can use the system transparently. It is considerably robust. Using the *OpenFlow*[13] switch architecture, it is possible to implement a cache system which works at the switching layer. The OpenFlow is a new programmable switch abstraction defined by the OpenFlow Switch consortium centered at Stanford University. The overall intent of OpenFlow is to separate the control plane of a switch from its data plane, and make the control plane programmable.

This paper proposes to use new advances at the switching layer to redirect requests of clients to cache systems seamlessly and effortlessly. Technically, OpenWeb, an overlay transfer application of the OpenFlow programmable switching layer, is introduced. OpenWeb can provide good user experience and server load balancing without any troublesome user configuration and difficult-frail administrator management.

The remainder of this paper is organized as follows. In Section 2, basics and related work in the field are described. In Section 3, the OpenFlow framework is introduced and described. Section 4 describes the OpenWeb application on OpenFlow, and show how it seamlessly incorporates new web applications. In Section 5 and 6, a prototype of OpenWeb and some performance results are shown. Then some discussions, conclusions, and directions for further work are offered.

2 Background

This section provides a brief overview of web content and cache systems.

2.1 Web Content Handled by Cache Systems

Generally, there are two types of Web content handled in the field. One is static content and the other is dynamic content. The difference between these two types of content is whether the same content is delivered to all users by the same URL overtime. In the case of static content, the same content is delivered to all the users exactly as it is stored. For example, HTML documents, cascading style sheets, and image files are typical static content. On the other hand, in the case of dynamic content, content is generated and delivered to the users in accordance with the each user's interaction, the time, and so on. For example, personalized pages and frequently-updated pages such as the top pages of news portals are typical dynamic content.

Although cache systems mainly handle static content, some dynamic content can be handled by cache systems considering a certain unit time. For instance, news articles and blog entries can be cached for a certain time. Moreover, even if a web page needs to be generated dynamically, the page will contain some static content such as template HTML documents, image files, and so on. These kinds of content also can be handled by cache systems.

2.2 Contents Delivery Systems

The past decade has seen the gradual introduction of contents delivery systems. From the viewpoint of using data cache, these systems can be seen as a type of cache system and divided into 3 types.

2.2.1 Contents Delivery Network

CDNs seek to provide scalable and high-performance delivery of block data, notably including web pages, saving load on both clients and original hosts, namely a web server which has original content. Examples of CDNs are Akamai[14], CoDeen[15][21][17], and Coral[6].

Some CDNs require several configurations of clients to use the systems. This means that the effect of the systems depends on client behavior. Although other CDNs do not require clients to do something, server administrators need to rewrite DNS records which are used to resolve web servers' IP address. This means that the servers are inseparably connected with the cache systems. These kinds of cache systems are expected to be independent from web servers for robustness.

2.2.2 Peer-to-Peer Service

P2P services seek to provide high-performance and robustness in content distribution/sharing among many collaborating clients. Examples of P2P services are BitTorrent[10], Antfarm[18], and Squirrel[8].

Some P2P services achieve very good performance. To use P2P services, however, clients need to install special softwares. Moreover, cache management in P2P services is difficult to be implemented because clients which have data cache join and leave whenever they like. Although there are some cache management methods such as distributed-hash-table algorithms and simple management by a central server, these methods sacrifice performance and scalability. Note that some methods even have a single point of failure. For these reasons, it can be said that clients should not do anything special to benefit from the systems and the systems should work keeping its performance and scalability.

2.2.3 Storage System

Storage systems seek to provide distributed storage services for high-performance content distribution. Examples of storage systems are Amazon S3 (Simple Storage Service)[16], OceanStore[19], and the Internet Logistics Service[2]. Storage systems can be regarded as a storage-specific CDNs.

Obviously, to benefit from these storage systems, data must be stored in these systems. Moreover the same problems with CDNs can be pointed out.

2.3 Other Related Works

As other existing works which are similar to the proposed method in this paper, there are at least two efforts at the application layer, and one transport-layer protocol.

Ocala[9] is an overlay convergence layer sitting below the transport layer and above the IP layer in the host networking stack. Its fundamental purpose is to intercept networking API calls and redirect them to the appropriate service. It consists of an *overlay-independent* layer, which services all registered overlay services, and an overlay-dependent layer, which formats specific and appropriate requests for the specific service. Ocala was demonstrated on a NAT traversal service and a secure connection service.

Oasis[12] was, similarly, a modified networking layer below the transport layer on the end-host IP stack. Oasis concentrated heavily on adaptive, rule-based routing of end-user packets onto a number of overlays; the rules were sufficiently general to support a wide variety of application protocols.

The WCCP (Web Cache Communication Protocol)[5] is a Cisco-systems inspired protocol offered on a number of Cisco routers. It redirects traffic for the web to a group of web servers, utilizing load balancing. However, this is available only on router products, not on switches definitely.

To the authors' best knowledge, the work described in this paper is the first effort to offer web redirection at the switching layer, without use of modification to end-host code, nor to require IP redirection. Of course it is not necessity to rewrite DNS records.

3 OpenFlow

To implement a cache system which works at the switching layer, the OpenFlow programmable switch abstraction is used. This section introduces and describes the OpenFlow framework based on the OpenFlow white paper[13].

3.1 Feature

An OpenFlow switch adds a *controller* to a network of switches, which is a single commodity server connected to each switch in the network over a secure connection. The controller acts as the control plane for the switches, directing packets on a network-wide basis. Of course the controller cannot make packet decisions at line speed. Thus, the switches present to the controller a data abstraction which permits the controller to program the switches on a coarser grain than the individual packet.

The basic abstraction in OpenFlow is a *flow*, which one can think of as a single TCP session or stream of UDP packets between a single source and a single destination, over a single TCP/UDP port. The OpenFlow switch maintains a *flow table*, which contains the directives to route and manage flows. The controller sends the OpenFlow switch rules for processing flows. This instruction is very powerful, as it permits the controller to make a packet-by-packet decision and also update the switch flow table for subsequent processing of further packets in this flow or packets in similar flows.

3.2 Implementation

OpenFlow has been implemented by a number of vendors on their switch lines, notably including NEC, Cisco Systems, Juniper Networks, and Hewlett-Packard. The level of support for OpenFlow and the efficacy of the implementation varies from vendor to vendor; NEC has announced full support for OpenFlow as a product offering, and HP has made a significant effort achieving a robust OpenFlow implementation as a research project with high bandwidth and extensive support.

In addition, NetFPGA[11] can be used as a OpenFlow switch. NetFPGA is a kind of layer-2 programmable switch based on FPGA (Field-Programmable Gate Array). Note that NetFPGA is made not only for OpenFlow but also all network modules that use hardware rather than software to forward packets.

3.3 Example Application

OpenFlow has been the platform for a wide variety of innovative networking projects. Some examples follow.

- Network Management and Access Control
  OpenFlow's canonical example is Ethane[4]. Indeed, OpenFlow is essentially the network portion of Ethane. Ethane enabled the central definition of network-wide policy on access control and quality of service. The controller checks a new flow against a set of rules, such as:
  - Guests on the network can communicate using HTTP, but only via a specific web proxy;
  - Only IP addresses belonging to human resources can access the personnel database;
  - VOIP phones are not allowed to communicate with laptops.
- Multipathing in Wireless Networks
  It enables clients to achieve good quality of service such as bandwidth and fault tolerance in wireless networks.
- Virtual Machine Migration
  It enables server administrators to migrate virtual machines from an old server to a new server while keeping IP addresses of virtual machines constant.

There are a lot of other examples. OpenFlow has been used for a lot of advanced network researches.

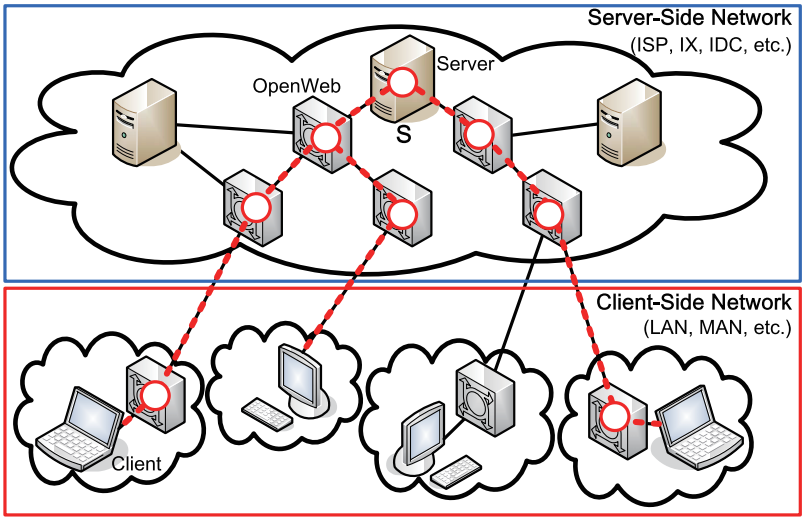

Figure 1: Working Image of OpenWeb

4 OpenWeb

In this section, OpenWeb is proposed as a new open protocol at the switching layer (layer-2). This protocol enables far more robust and seamless packet redirection without requiring user configuration or unreliable scripts. OpenWeb is a layer-2 redirection engine implemented as an application of the OpenFlow switch architecture.

4.1 Concept

The goal of OpenWeb is to enable users to use the cache system without any configurations, and administrators to manage the cache system easily and robustly. In order to achieve this goal, three things described below should be kept in mind.

Independency

The system should be independent from web servers and can handle unexpected requests. This is quite different from traditional systems. For instance, in DNS-based systems, administrators need to rewrite DNS records which are used to resolve web servers' IP address before requests come to the web servers. Thus, if administrators cannot predict the flood of requests and rewrite DNS records, the systems cannot work well. However, if a system can work independently from web servers, unexpected requests can be possibly handled.

Transparency

The system should be able to be used without any troublesome configurations by clients. This means that the effect of the cache system does not depend on client behavior; some clients cannot/do not install special softwares and/or do configurations. This is quite different from traditional systems such as BitTorrent or Coral.

Performance

Processing performance should be the same as traditional systems because it is directly linked with user experience.

Working image of OpenWeb is illustrated in Fig.1. This image just represents the servers, the OpenWeb systems and the clients on the Internet. Requests form the clients go through an array of the OpenWeb systems. OpenWeb is a kind of web proxy systems which works between the clients and the servers at switching layer. This means that if an OpenWeb system has cache of requested

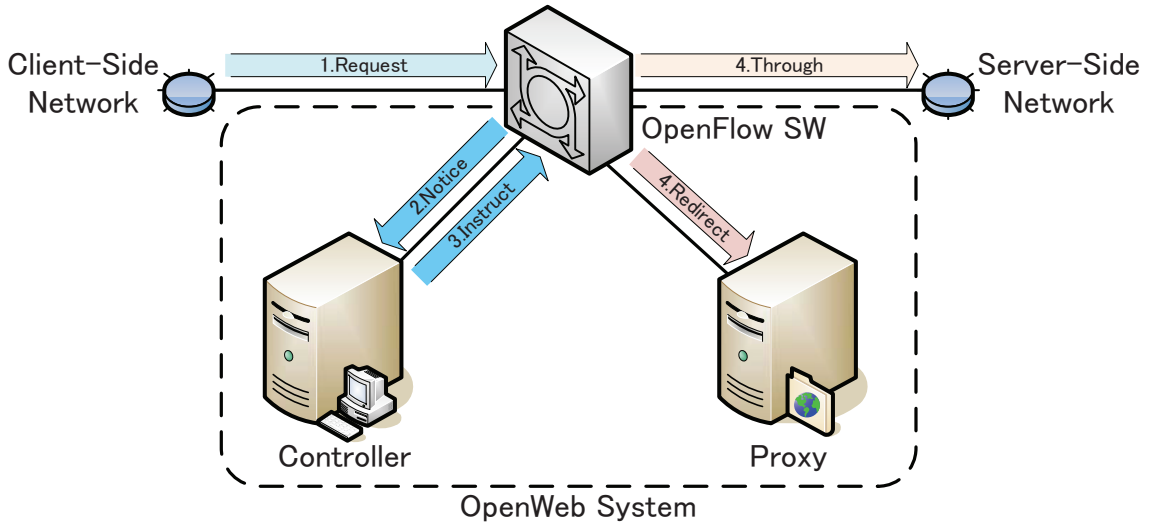

Figure 2: A Typical Architecture

content, content is delivered to the user by the OpenWeb system. Additionally, the clients can use OpenWeb systems transparently.

If sufficient amounts of OpenWeb components are arranged in the Internet uniformly, OpenWeb will autonomously organize a tree structure CDN which represents the paths from clients to a server, which is shown as server S in Fig.1, determined by IP routing. In the CDN, a server which has original contents will be a root, OpenFlow switches will be nodes, and clients will be leafs. Moreover, the CDN tree structure is not application level but network level. This CDN enables scalable and high-performance delivery of content saving load on web servers and providing good user experience.

4.2 Architecture

A general architecture of OpenWeb is illustrated in Fig.2. At least one OpenFlow switch, one controller, and one proxy server are required in one OpenWeb system. When the switch detects a request from the client-side network, the switch notifies it to the controller. Then the controller checks the request if it needs to be redirected or not. According to the controller's decision, the switch puts the request through to the server-side network or redirects the request to the proxy server.

The role of each component is as follows:

- Proxy Server
  General proxy servers are used so far. It processes requests from clients on behalf of web servers. (In the future, other advanced functions such as application containers will be introduced to support real-dynamic content.)

- OpenFlow Switch
  The OpenFlow switch is the heart of the OpenFlow platform and OpenWeb. It is a layer-2 switch managed under the controller as described in Section 3.

- Controller
  The controller determines whether to redirect a request using the redirection methods described in the next section. This decision can be made according to static configurations by administrators or dynamic configurations based on access trends.

Besides the components listed above, clients and servers which have original contents would belong to their own networks. Clients are normal computers used by web users and servers are

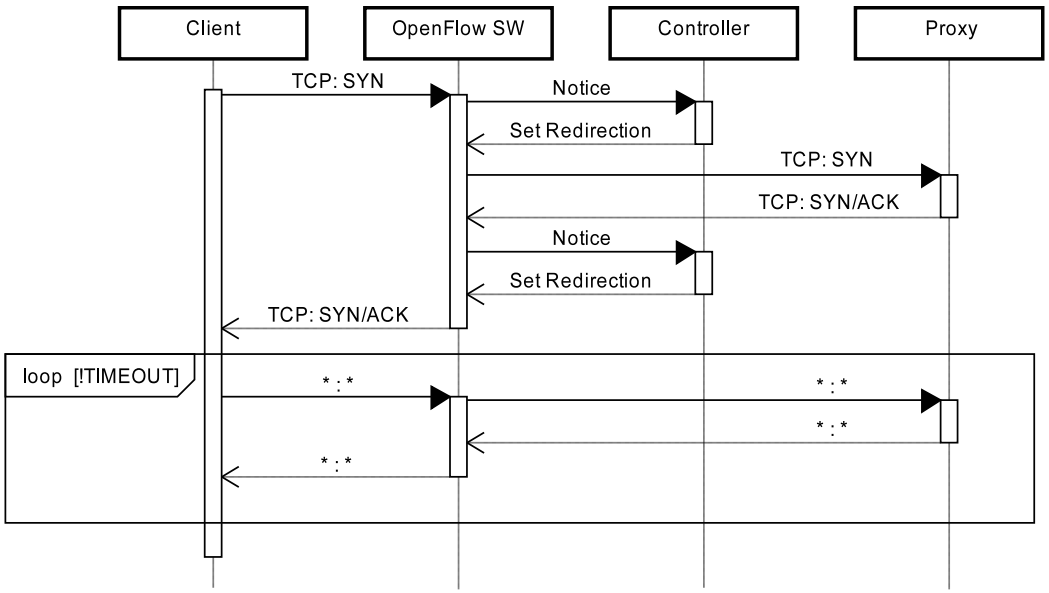

Figure 3: IP-based Redirection

normal computers managed by server administrators. Note that this server administrators are different from administrators of OpenWeb systems; although they can be an administrator of both normal servers and OpenWeb systems at the same time naturally. Of course, no proxy configuration is required on clients and no server configuration is required on servers to use the system because the system works at layer-2. Moreover, clients/servers and OpenFlow switch can belong to either the same network or different networks.

4.3 Redirection Method

The redirection methods determine whether to redirect a request to a proxy or transfer it to a server. Two types of packet redirection methods are formulated for OpenWeb. One is the IP-based redirection and the other is the URL-based redirection. In both methods, after a two-way redirection between a client and a proxy is established, client's communication path is redirected to a proxy server until redirection rules become time out. Once the request is decided to be redirected, an OpenFlow switch redirects the request to a proxy server using NAT (Network Address Translation) or NAPT (Network Address Port Translation).

4.3.1 IP-based Redirection

A traditional web server is associated with one IP address per one host name. Therefore, the simplest way of packet redirection is to redirect toward a proxy server a packet which contains an IP address of a host as its destination.

Typical packets flow of the IP-based redirection is illustrated in Fig.3. First, the client sends a SYN (TCP) packet to a host. The OpenFlow switch notifies the receipt of the packet to the controller. The controller looks into the packet and if the destination IP address of the packet is redirection target, the controller sets a rule to the switch to rewrite the destination IP address of the packet with the IP address of the proxy server. In the IP-based redirection, a list of target hosts are prepared as a list of IP-addresses as a condition of redirections. Then, the proxy server replies with a SYN/ACK (TCP) packet to the client. Likewise, the switch notifies the receipt of the reply packet to the controller and the controller sets a rule to the switch to rewrite the source IP address of the packet with the IP address of the host. In this way, a two-way redirection is established.

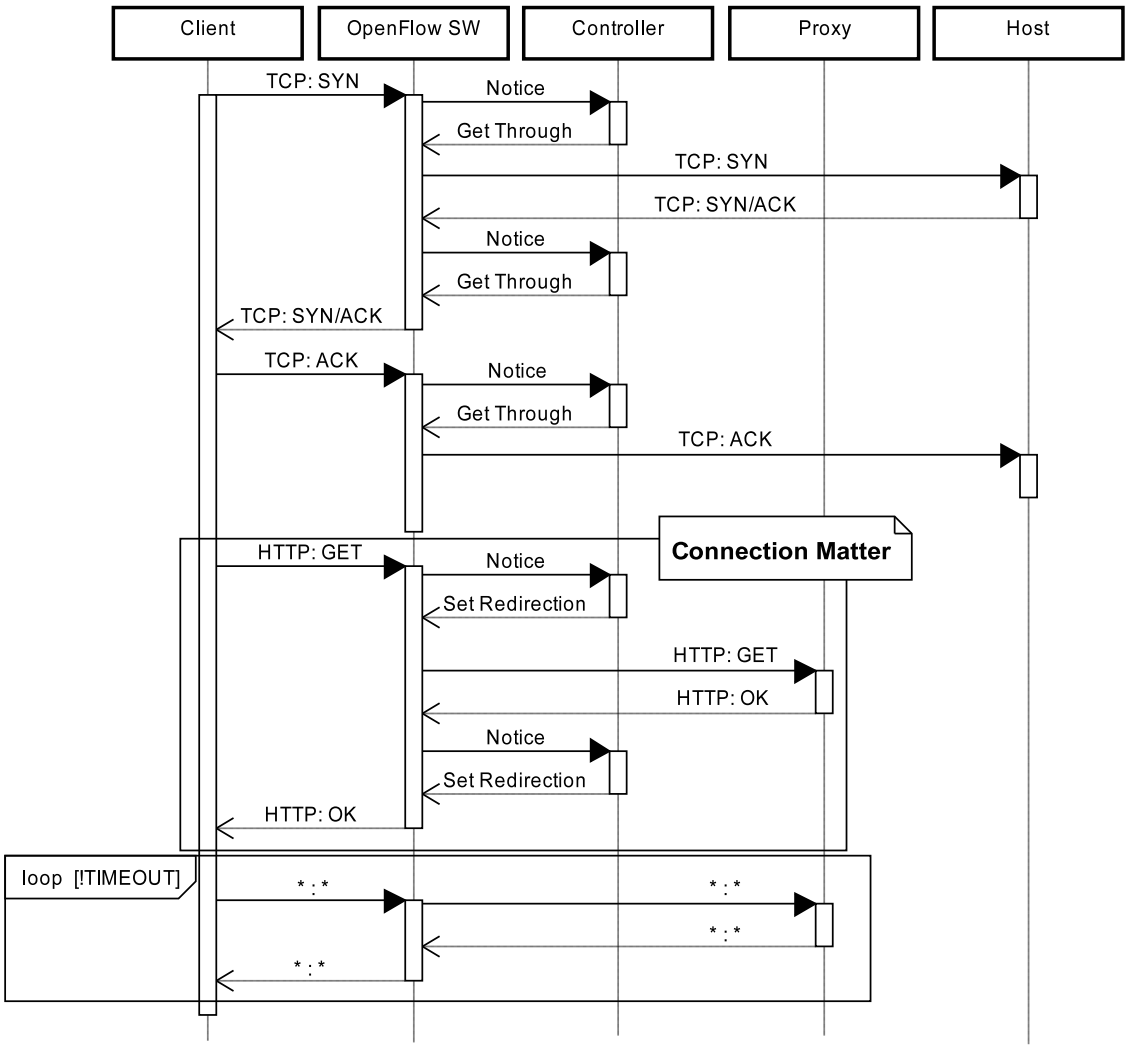

Figure 4: URL-based Redirection

4.3.2 URL-based Redirection

Nowadays, many web sites employ the DNS round robin for load-balancing and the virtual hosts for server resources sharing. It means that there can be several web servers (or IP addresses) for one host name and several host names for one web server (or IP address). The IP-based redirection is not suitable for these situations because making redirection rules one by one is not realistic and far from robustness. For instance, it is almost impossible to know all IP addresses of hosts operated by a DNS round robin system. To solve this problem, the URL-based redirection is proposed. In the URL-based redirection, the controller uses not the destination IP address of the packet, but the URL of the HTTP request.

Typical packets flow of the URL-based redirection is illustrated in Fig.4. First, the client sends a SYN (TCP) packet to a host. The OpenFlow switch notifies the receipt of the packet to the controller. The controller checks if the packet is an HTTP packet or not, by looking at the port number of the packet. If the packet is not an HTTP packet, the controller passes the packet through. If the packet is an HTTP packet, in URL redirection, the controller analyzes the body of the packet. This time, because there is no content in the body of the SYN (TCP) packet, the controller passes the packet through. Then, the packet comes to the host and the host replies with a SYN/ACK

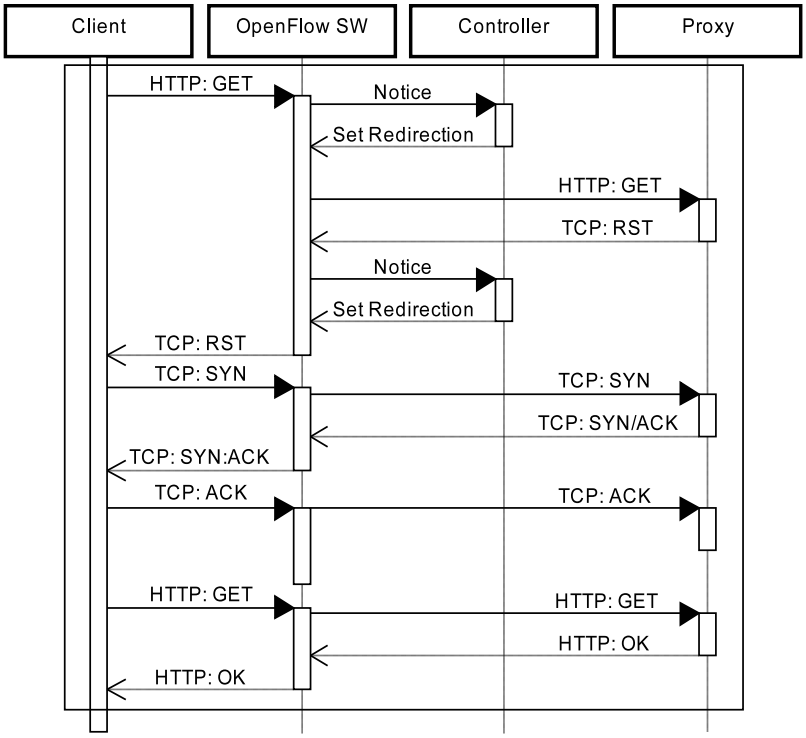

Figure 5: Connection between the Client and the Proxy

(TCP) packet to the client. Likewise, the switch notifies the receipt of the packet to the controller and the controller passes the packet through. An ACK (TCP) packet sent by the client is delivered to the host in the same way.

After that, the client sends a GET (HTTP) packet to the host and the switch notifies it. This time, the controller finds the GET command in the body of the packet. Then the controller looks into it and if the URL of the request is redirection target, the controller sets a rule to the switch to rewrite the destination IP address of the packet with the IP address of the proxy server. In the URL-based redirection, a list of target hosts are prepared as a list of URLs as a condition of redirections. Then, the proxy replies with an OK (HTTP) packet to the client. Likewise, the switch notifies the controller the receipt of the reply packet and the controller sets a rule to the switch to rewrite the source IP address of the packet with the IP address of the host. In this way, a two-way redirection is established.

To be exact, although the basic idea of the URL-based redirection method is as explained above, there is a connection matter between the client and the proxy (see the noted rectangle in Fig.4). That is how to enable the client to communicate with the proxy. Because a connection between the client and the proxy is not established at first, it is impossible to make the proxy process the GET (HTTP) packet just by redirecting the packet. The simplest strategy to solve this situation is to make the client take a reconnect action. Typical packets flow of the reconnect strategy is illustrated in Fig.5. First, the client sends a GET (HTTP) packet and the controller detects it and sets a rule to the switch to rewrite the destination IP address of the packet with the IP address of the proxy. The proxy receives the GET (HTTP) packet and returns a RST (TCP) packet because the proxy does not have the connection of it. Then the OpenFlow switch notifies the receipt of the RST (TCP) packet to the controller and the controller sets a rule to rewrite the source IP address of the packet with the IP address of the host. After that, a connection between the client and the proxy is established through the 3-way handshake and now the proxy can process HTTP packets from the client normally. Note that the reconnect strategy depends on TCP implementations and/or client

applications such as web browsers. However, most TCP implementations and client applications take a reconnect action when they receive a RST (TCP) packet for fault resilient. Therefore, OpenWeb employs this strategy so far. More general discussion about connections between clients and proxy servers is held in Section 7.2.

4.4 Caching Algorithm

For the redirection methods, a list of target hosts must be prepared as a condition of redirections. In addition, if there are several proxy servers in a OpenWeb system, redirections should be set to suitable proxy servers. However, in this paper, how to create a list of target hosts and proxy server selection method are not discussed because there a lot of works related these problems. For example, proxy servers can simply cache content which is requested more than a threshold and proxy servers can be selected by *fastest response times*, and so on. Though it is also not discussed in this paper, if an administrator manages several OpenWeb systems, he or she can combine controllers of the systems and apply extensions such as pre-caching of contents to another OpenWeb proxy servers. This kind of caching algorithm could be proposed in the future.

5 Performance Evaluation

A prototype system of OpenWeb has been built based on NetFPGA. Using a prototype, the latency required to redirect packets with NAT and NAPT is evaluated.

5.1 Prototyping

To implement a prototype of OpenWeb, NetFPGA is employed as an OpenFlow switch. In addition, NOX[7] is employed to write a controller program of the OpenFlow platform. Using NOX, control programs can be written in Python. This time, OpenFlow 0.9.0, NOX 0.6.0, and Python 2.6.4 are used.

The NetFPGA-based OpenFlow switch and its controller are working on the same machine. The machine specification is Intel Pentium 4 CPU 3.20 GHz with 1024MB memory, running CentOS 5.4 (Linux 2.6.18).

5.2 Evaluation Setting

The evaluation environment is the same as Fig.2. The controller and the proxy server are connected with the OpenFlow switch and the client is connected with its network. Two network devices, a normal layer-2 switch and a Linux machine running *iptables*, are prepared to be compared with. The Linux machine is the same machine used by OpenWeb system. The normal layer-2 switch just transfers packets as usual and the *iptables* do NAT or NAPT. These two devices replace the OpenFlow switch when each device is evaluated. All components are connected by gigabit Ethernet.

The client tries to get content 100 times per each combination of NAT/NAPT and the network devices. As subject content of the client, 128KB, 256KB, 512KB, 1024KB, and 2048KB files are prepared. The interval time for trial is 1000ms. Every execution time is recorded, and minimum, maximum and mean are evaluated.

5.3 Result

The results comparing OpenWeb with the normal layer-2 switch and the *iptables* are illustrated in Fig.6 and Fig.7. Fig.6 and Fig.7 represent the results for NAT and NAPT, respectively. Note that the scales of the vertical axes of the two graphs are different. The “Direct” represents results by the normal layer-2 switch, the “IPT-*-*” represents results by the *iptables*, the “OPF-*-*” represents results by the OpenFlow switch (OpenWeb), the “*-NAT” and the “*-*-NAT” represent results by NAT, the “*-NAPT” and the “*-*-NAPT” represent results by NAPT, the “OPF-IP-*” represents results by IP-based redirection, and the ”OPF-URL-*” represents results by URL-based redirection.

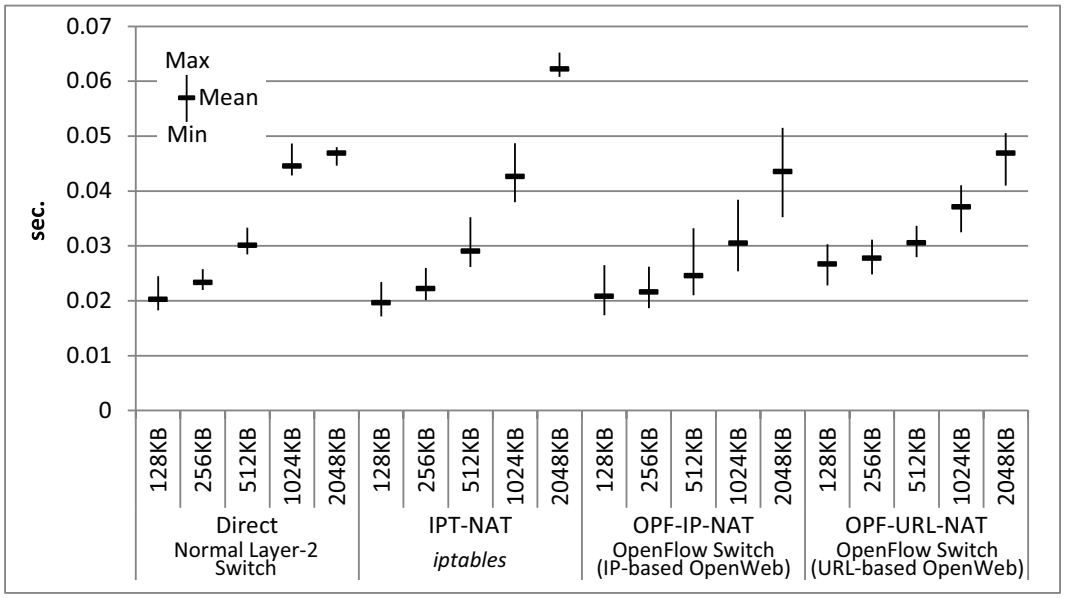

Figure 6: Execution Time of NAT

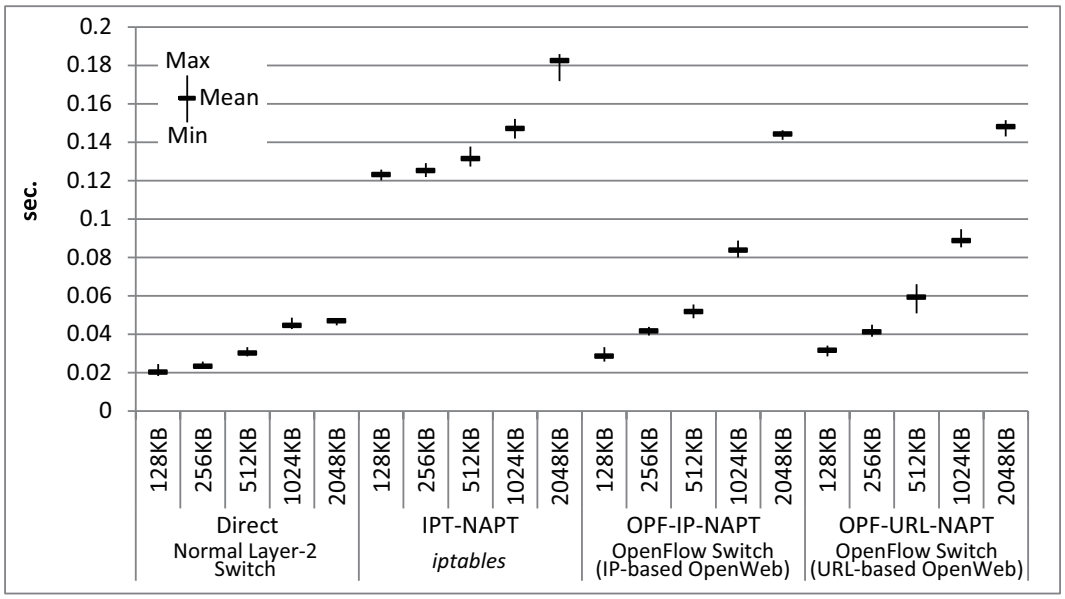

Figure 7: Execution Time of NAPT

Fig.6 shows that OpenWeb with NAT (the "OPF-IP-NAT" and the "OPF-URL-NAT") works faster than *iptables* with NAT (the "IPT-NAT") at large. Furthermore, OpenWeb with NAT works faster than the "Direct" in some cases. The reason for that will be due to the NetFPGA's high performance and/or the layer-2 switch's poor performance. The "OPF-IP-NAT" is a little faster than the "OPF-URL-NAT."

Fig.7 shows that OpenWeb with NAPT (the "OPF-IP-NAPT" and the "OPF-URL-NAPT") works much faster than *iptables* with NAPT (the "IPT-NAPT"). But this time, they work slower than the "Direct" because additional process, rewriting a source and a destination port, is needed. Then there is a little difference between the "OPF-IP-NAPT"'s performance and the "OPF-URL-NAPT"'s performance in this index.

In these figures, outliers in the results are omitted using the Grubbs' test, which is a statical method for outliers. For instance of an outlier, in the OpenWeb environment, the actual maximum time of OPF-IP-NAT for 2048KB is 0.99218321 seconds, which is about 22 times of its mean. We could not identify the reason of this, but there is no factor in the OpenWeb environment itself to cause the delay. This problem would be resolved by the improvements of the OpenFlow environment including NetFPGA and Python.

As seen above, although the stabilization is required, OpenWeb will be favorably comparable to a normal layer-2 switch and *iptables*.

6 Network Simulation

Network simulations are done to investigate how OpenWeb works in a random network. If OpenWeb works effectively even in a random network, OpenWeb will work in the real Internet more effectively because prior knowledge can be used to arrange OpenWeb components.

6.1 Simulated Environment: Random Network

It it presupposed that 1024 nodes exist in one random network. A node represents an autonomous system. In this simulation, two types of nodes are prepared. One is normal node and the other is OpenWeb node.

- Normal Node
  Normal nodes only forward requests to a server according to routing information.
- OpenWeb Node
  OpenWeb nodes redirect requests to proxy servers as far as its maximum number of incoming connections allows. If the capacity is full, it works as a normal node.

The nodes are connected by 5238 (1/100 of complete graph) edges. An edge represents a connection between nodes. Then, a server and clients are arranged in the network randomly. Note that the server and the clients are not nodes, but they belong to nodes. The number of clients is decided at random for each simulation.

6.2 Simulation Scenario

The simulation scenario is as follows.

1. Generate a random network
2. Generate a total of 1024 requests from randomly-arranged clients; all the requests try to reach the server which is the sole server in the network
3. The requests head for the server according to routing control which mimics IP routing
4. Requests coming to OpenWeb nodes will be redirected to proxy servers as far as its capacity allows

Table 1: Possible Ways of Request Process

Processed by	How processed
Server	*Served*
	Waited to be served
	Rejected
OpenWeb	*Redirected*
	Waited to be redirected

Table 2: Parameter Value

Parameter		Value
Number of OpenWeb Nodes		0, 1, 2, 3, ..., 1024
Cache Ratio of OpenWeb Nodes		0%, 25%, 50%, 75%, 100%,
Server	MaxClients	256
	ListenBackLog	511
OpenWeb Node	MaxClients	1, 2, 4, 8, 16, 32, 64, 128, 256, 512

5. Requests coming to the server will be processed as far as its capacity allows

6. Check how each of the requests is processed

Requests can be processed in five ways as shown in Table.1. *Served* indicates that requests are served by the server. *Waited to be served* indicates that requests are waited but served by the server. *Rejected* indicates that requests are rejected and not served by the server. *Redirected* indicates that requests are redirected to proxy servers by OpenWeb. *Waited to be redirected* indicates that requests are waited but redirected to proxy servers by OpenWeb.

6.3 Parameter Setting

All parameters for the simulations are shown in Table.2. First, the number of OpenWeb nodes is a parameter. In one random network, 1024 nodes exist as explained before. Accordingly, the number of OpenWeb nodes ranges from 0 to 1024. Once the number of OpenWeb nodes is determined, a cache ratio of it—how many OpenWeb nodes already have content cache—needs to be determined. The ratio is represented by the percentage of the number of OpenWeb nodes.

Next, there are two parameters for a server and one parameter for OpenWeb nodes. MaxClients represents the maximum number of connections that can be processed concurrently. At the server, requests are served. At the OpenWeb nodes, requests are redirected to proxy servers. ListenBackLog represents the maximum length of the queue of pending connections which means that requests are waited but they will be surely served later. This time, default values of Apache 2.2 are used for these server parameters. A value of MaxClients for OpenWeb node is fixed to 1, 2, 4, 8, 16, 32, 64, 128, 256, and 512. Finally, 1000 trials are done for each parameter combination.

6.4 Result

Because it is not realistic to show all of the simulation results, as an example, Fig.8 shows the results in case when the cache ratio parameter and the MaxClients parameter of OpenWeb node are fixed to 50% and 1, respectively. The horizontal axis represents the number of OpenWeb nodes in the random network. The vertical axis represents the number of requests. Each line represents each result of request processes. As the number of OpenWeb nodes increase, *Served*, *Waited to be Served*, and *Rejected* requests decrease naturally because the total capacity of request redirection increases. In contrast, *Redirected* and *Waited to be redirected* increase of course.

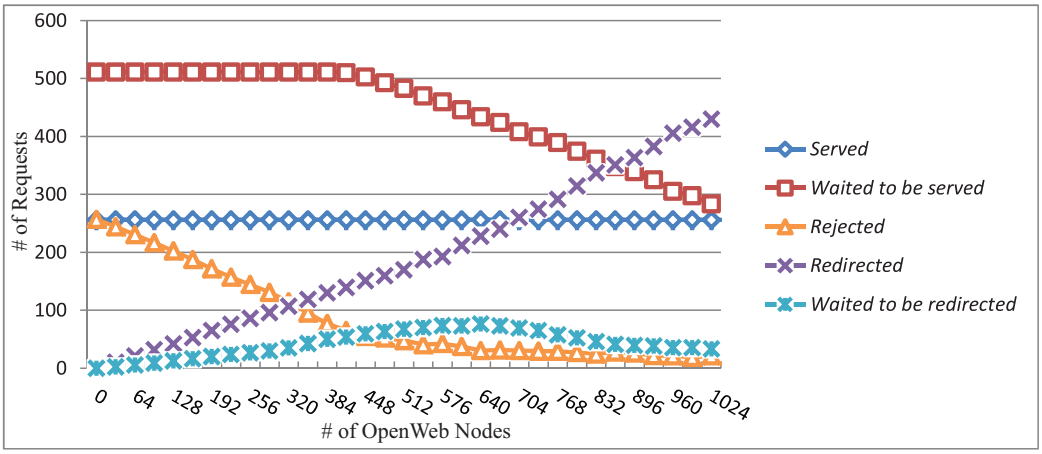

Figure 8: All Requests (OpenWeb::Cache Ratio = 50%, OpenWeb::MaxClients = 1)

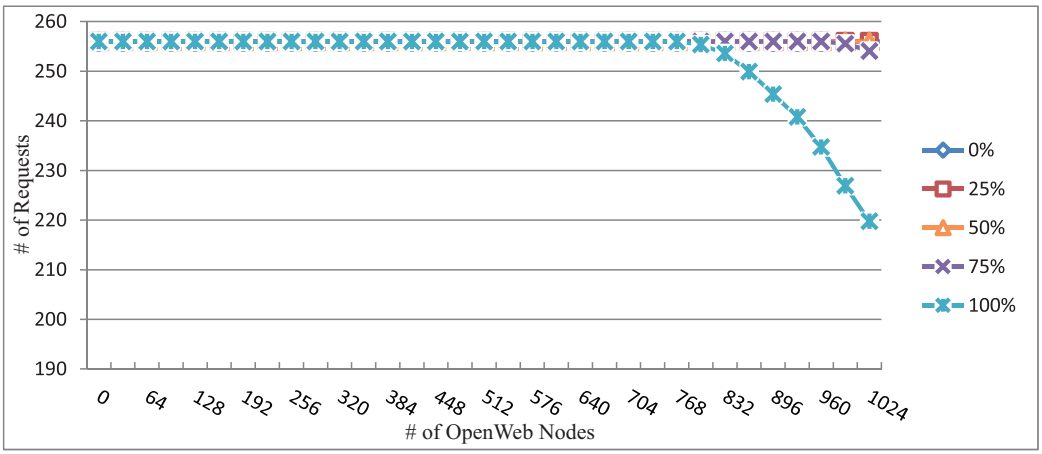

Figure 9: *Served* Requests (OpenWeb::MaxClients = 1)

The detailed results in case when the MaxClients parameter is fixed to 1 are illustrated in Fig.9 through 13. This time each line represents each result of cache ratios. Throughout Fig.9-13, although the difference between cache ratios seems significant, from the viewpoint of content serving to clients, it is not much difference because Fig.11 shows that *Rejected* requests are decreasing in the same way in the latter half. This means that if there are sufficient amount of OpenWeb nodes, it is not a major matter whether OpenWeb nodes initially have content cache or not for content serving to clients.

Clearly, the key of load-balancing is to decrease the number of *Rejected* requests because *Rejected* requests are no more than major source of overload; they demand content from a server which is already in full capacity. It is also directly linked with user experience. Fig.14 and 15 indicate *Rejected* results for a MaxClients value of OpenWeb node at 8 and 64, respectively. As shown in these graphs, the number of *Rejected* requests decrease faster than the increase of the number of OpenWeb nodes in any case. The results show that even if OpenWeb systems are arranged in a network randomly, it plays a role as load balancer well.

According to these results, even if a MaxClients value of OpenWeb node is increased, *Rejected* requests cannot be zero if the number of OpenWeb nodes is less than approximately 128. This convergence means that about 128 OpenWeb nodes are enough to satisfy all the requests in this simulation settings.

Considering user experience, it's an important factor if a user's request is waited or not waited (or even rejected). According to the results, as a MaxClients value of OpenWeb node increases,

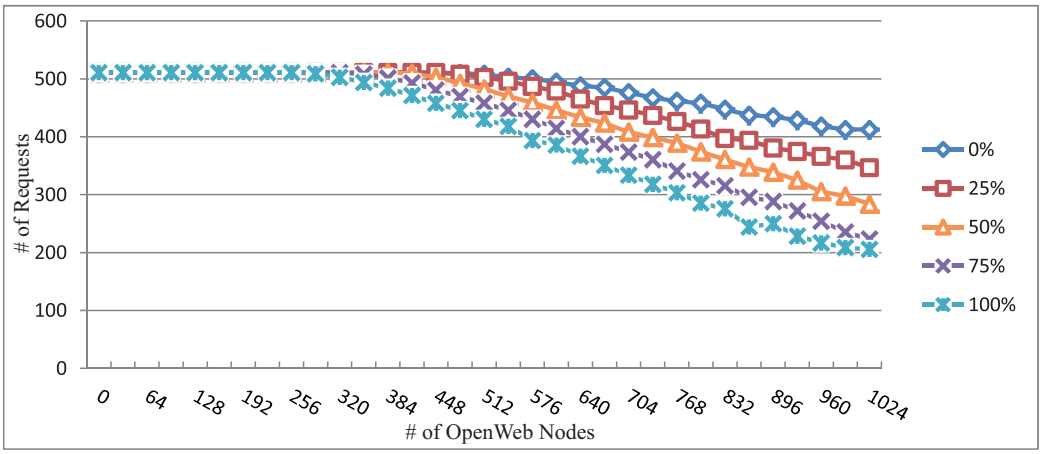

Figure 10: *Waited to be served* Requests (OpenWeb::MaxClients = 1)

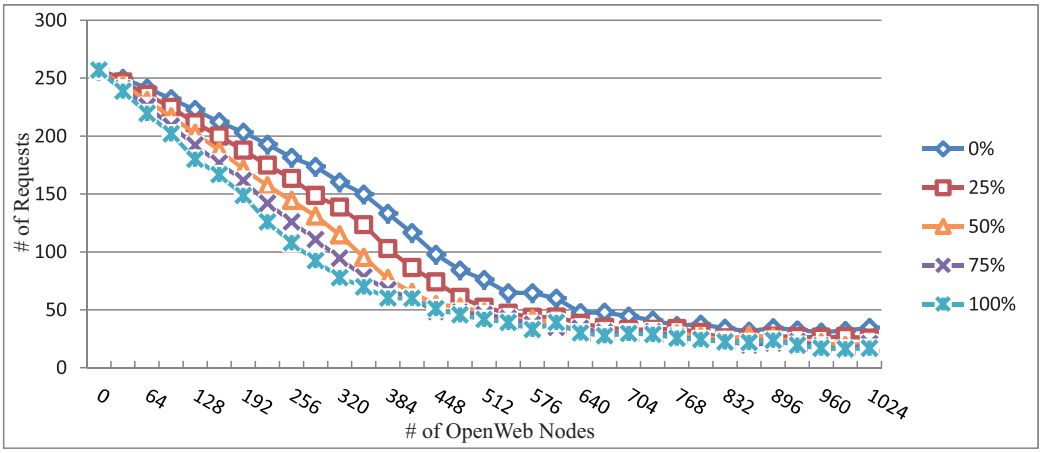

Figure 11: *Rejected* Requests (OpenWeb::MaxClients = 1)

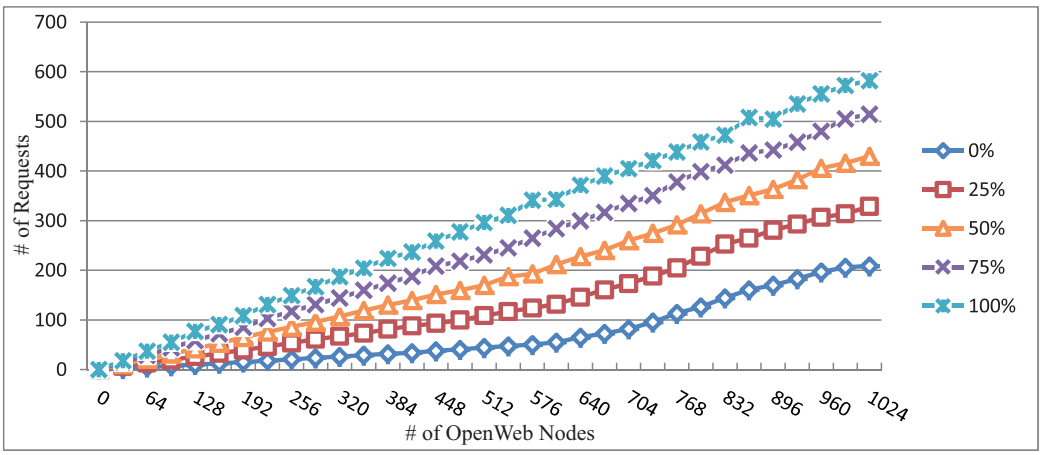

Figure 12: *Redirected* Requests (OpenWeb::MaxClients = 1)

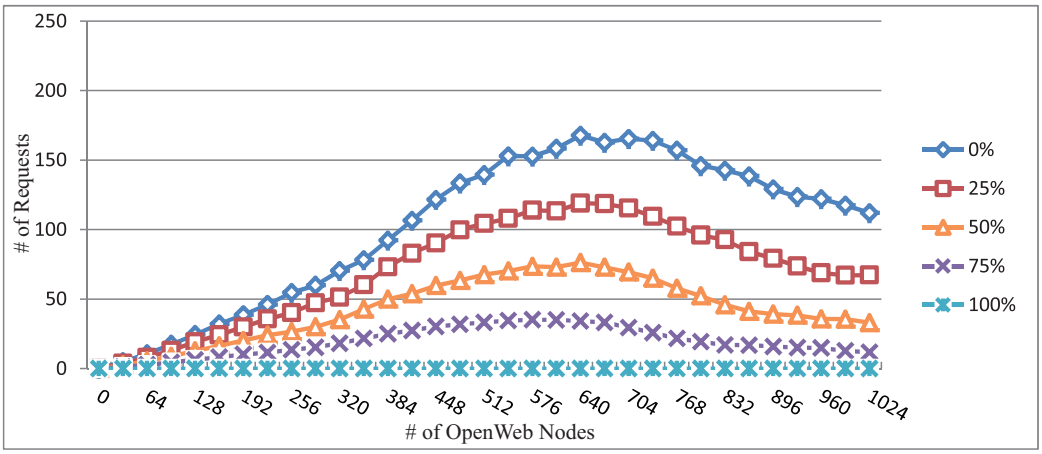

Figure 13: *Waited to be redirected* Requests (OpenWeb::MaxClients = 1)

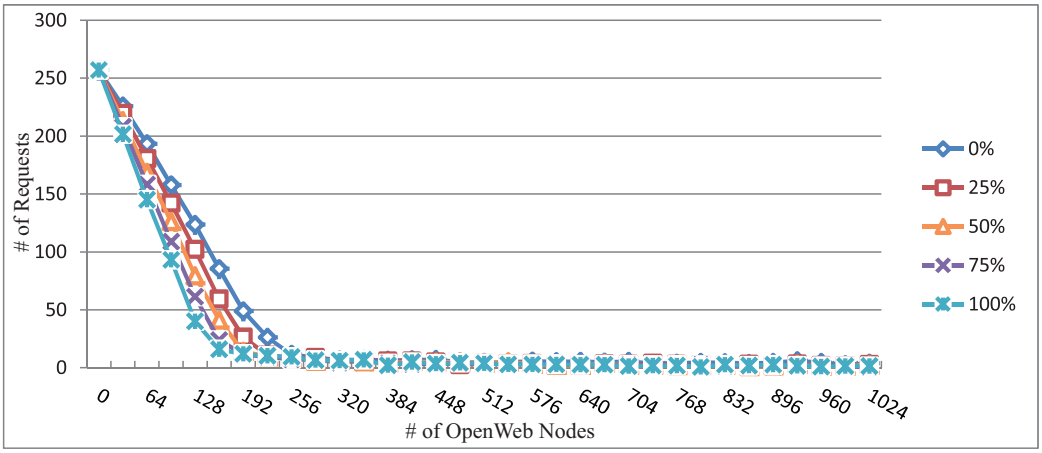

Figure 14: *Rejected* Requests (OpenWeb::MaxClients = 8)

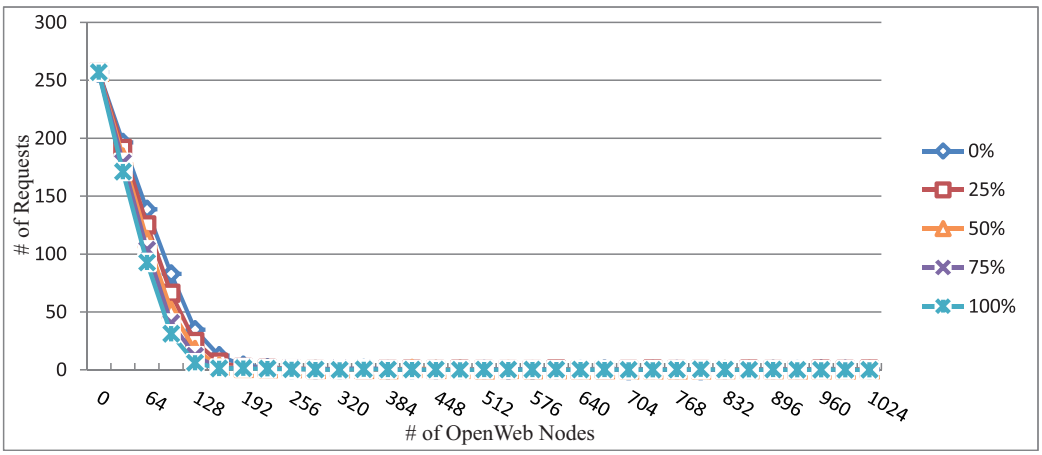

Figure 15: *Rejected* Requests (OpenWeb::MaxClients = 64)

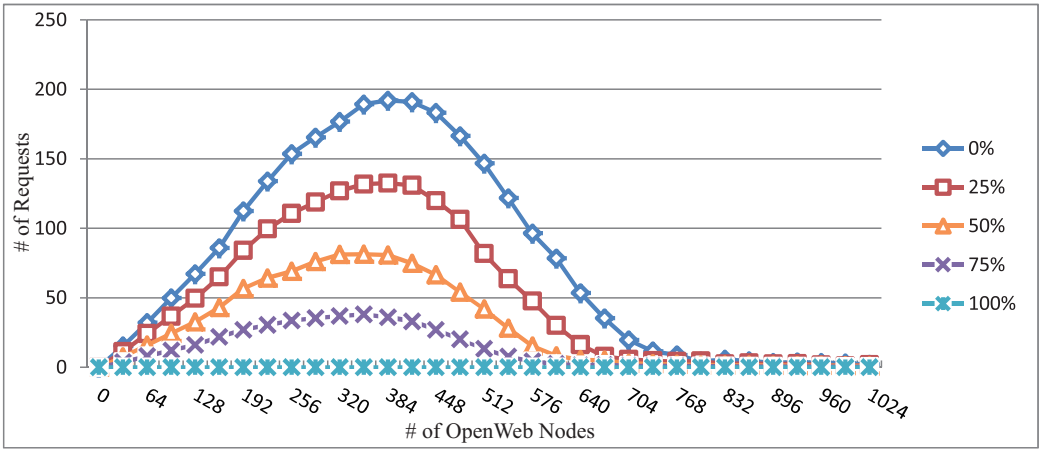

Figure 16: *Waited to be redirected* Requests (OpenWeb::MaxClients = 8)

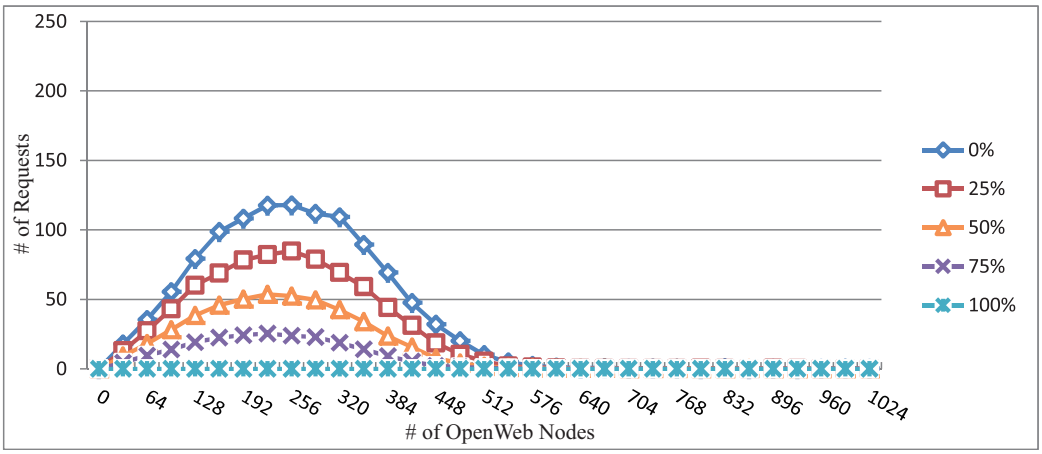

Figure 17: *Waited to be redirected* Requests (OpenWeb::MaxClients = 64)

a bell curve of *Waited to be redirected* requests gets smaller but it also seems to converge. Fig.16 and 17 indicate *Waited to be redirected* results for a MaxClients value of OpenWeb node at 8 and 64, respectively. This convergence means that at least about 110 clients need to be waited if all OpenWeb nodes do not have content cache.

7 Discussion

7.1 User Experience Metric

To know how much OpenWeb nodes are required for real situation, user experience metrics should be formulated.

Table.3 shows three possible metrics. A satisfaction level of clients is indicated by the number, from 0 to 1. 0 indicates that clients are completely dissatisfied and 1 indicates that clients are completely satisfied. S1 is the most loose metric; no matter how long clients are waited and who served clients with content, the clients are satisfied if they can get content. S2 is focused on waiting time; no matter who served clients with content, the clients are satisfied if they do not be waited. S3 is one of strict metrics; clients mind waiting time and who served them with content.

For instance, from the results of Section 6, Fig.18 shows the results in case when the cache ratio parameter and the MaxClients parameter of OpenWeb node are fixed to 50% and 1, respectively.

Table 3: User Experience Metric

	Served	*Waited to be served*	*Rejected*	*Redirected*	*Waited to be redirected*
S1	1	1	0	1	1
S2	1	0.5	0	1	0.5
S3	1	0.5	0	0.5	0.25

Fig.19 shows the results in case when the cache ratio parameter and the MaxClients parameter of OpenWeb node are fixed to 50% and 8, respectively. Fig.20 shows the results in case when the cache ratio parameter and the MaxClients parameter of OpenWeb node are fixed to 50% and 64, respectively. The horizontal axis represents the number of OpenWeb nodes in the random network. The vertical axis represents the total score of user experience.

Throughout Fig.18-20, it would be fair to say that S1 and S2 can be used as a metric to determine the number of OpenWeb node in a network because they seem to converge in any case. On the other hand, S3 would be hard to use as a metric because its curve seems to transform depending on the MaxClients parameter of OpenWeb node.

For example, in case when the cache ratio parameter and the MaxClients parameter of OpenWeb node are fixed to 50% and 64, using S1 metric, it can be said that about 128 OpenWeb nodes are enough to satisfy all clients demands.

7.2 Connections Between Clients and Proxy Servers

Of course, clients get content from proxy servers through TCP connections. Therefore, it's a key perspective how connections are established between clients and proxy servers in redirection-based cache systems like OpenWeb.

If a cache system works at layer-2 to layer-4, TCP connections are established between clients and proxy servers directly. In this case, the number of TCP connections used in a session is one as clients connect to original hosts.

In contrast, if a cache system works at layer-5 to layer-7, TCP connections between clients and the cache system are established first because information of layer-5 to layer-7 is not available before TCP connections are established. After the information is obtained, TCP connections between the cache system and proxy servers are established. In this case, the number of TCP connections used in a session is at least two.

Obviously the former can achieve better performance than the latter because the number of TCP connections is small. TCP connections supply sequence control, congestion control, and so on which take some processing time. Alternatively the latter can control the system's behavior more flexibly than the former because information of layer-5 to layer-7 can be used.

Though OpenWeb can use information of all layers (layer-2 to layer-7), OpenWeb works at layer-2. This means that the number of TCP connections is basically one. However, to get information of layer-5 to layer-7, TCP connections should be established. OpenWeb employ the reconnect strategy to get the information as explained in Section 4.3.2.

Although this strategy will work for the time being, there is a little problem of transparency. Technically, clients can find some traces of redirections but they cannot believe that their requests are redirected because the traces are the same as normal error recovery protocols of TCP connections. In order to hide the redirection form clients completely, at least two TCP connections are required like the latter case. Of course it is easy to establish two TCP connections. However, performance would be more important than transparency in this case. Alternatively, though proxy servers get to be difficult to be maintained, it is also possible to modify a TCP implementation and a proxy application to enable the proxy servers to process HTTP packets even there are no connections before the HTTP packets arrive at the proxy servers.

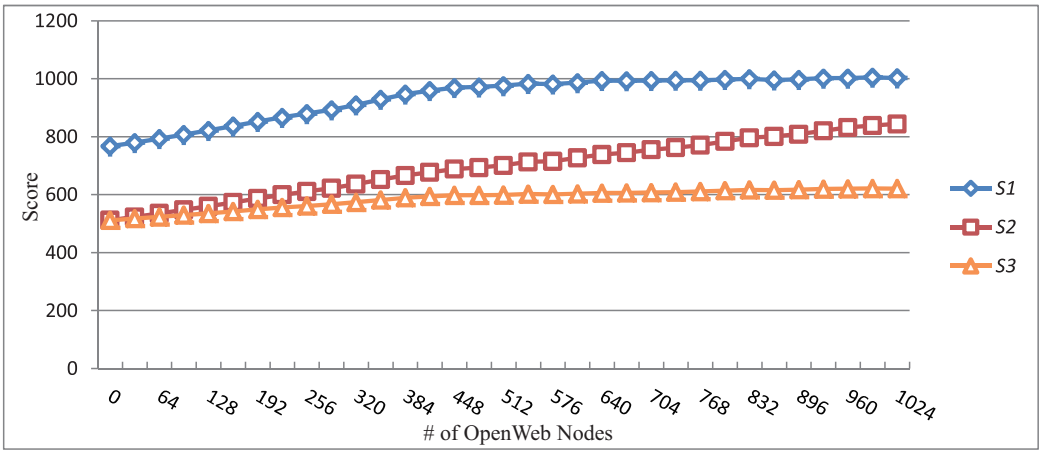

Figure 18: User Experience (OpenWeb::Cache Ratio = 50%, OpenWeb::MaxClients = 1)

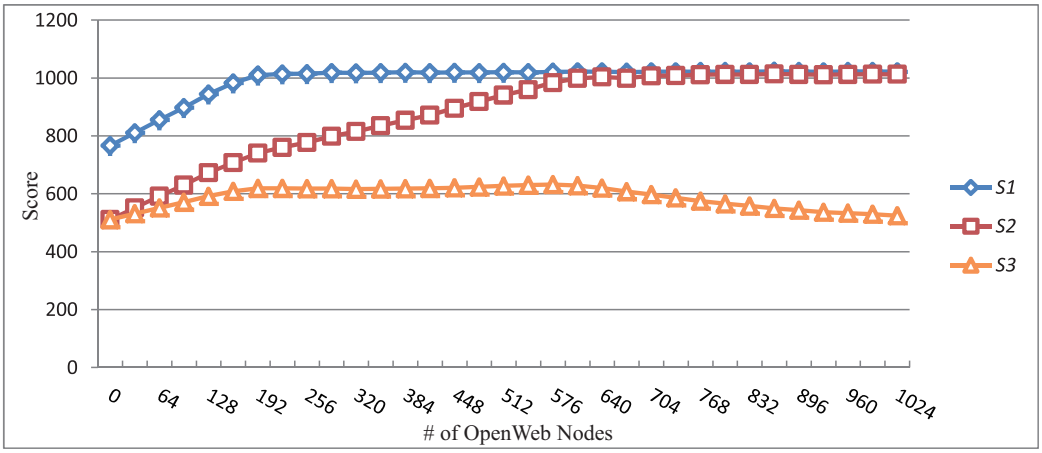

Figure 19: User Experience (OpenWeb::Cache Ratio = 50%, OpenWeb::MaxClients = 8)

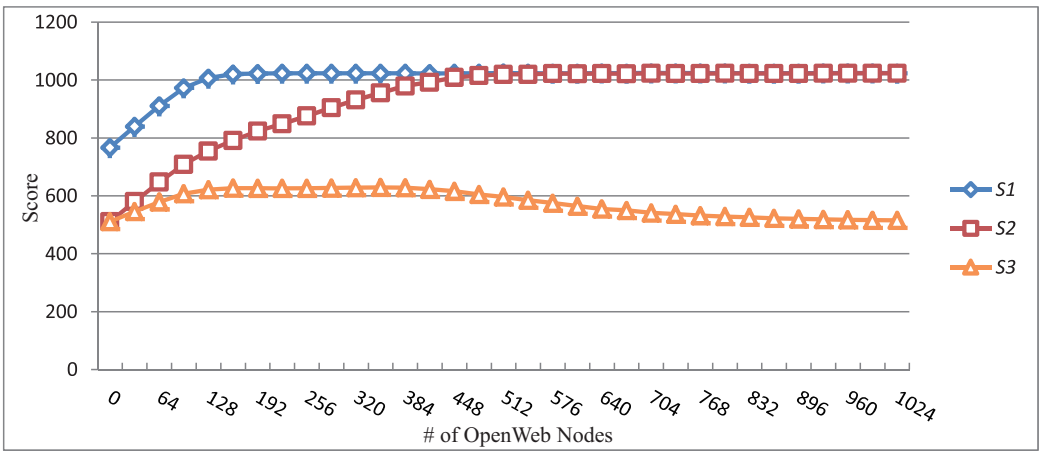

Figure 20: User Experience (OpenWeb::Cache Ratio = 50%, OpenWeb::MaxClients = 64)

7.3 Verification of Research Goal

Before the conclusion, the aim of this research is revisited. The aim of the research is to enable users to use the cache system without any configurations, and administrators to manage the cache system easily and robustly. Achieving this aim, three things have been kept in mind.

The first was independency. The system should be independent from web servers and handle unexpected requests. The network simulations which have been done in random network have confirmed these requirements.

The second was transparency. Clients can use the system without any troublesome configurations and the effect does not depend on client behavior. Of course, these are satisfied because OpenWeb redirects requests at layer-2 using the OpenFlow platform.

The last was performance. The basic performance evaluation shows that performance of OpenWeb achieves some positive results.

As seen above, it can be said that the goal has been almost accomplished. In addition, using prior knowledge to arrange OpenWeb components, and applying advanced caching algorithm, OpenWeb can evolve toward the future.

8 Conclusion

This paper presented the design of OpenWeb, a transparent overlay connection proxy resident on an OpenFlow layer-2 switch. OpenWeb requires no configurations of clients to use the system. While only the context of an HTTP connection proxy is described in this paper, the functionality is fully generalizable to any IP-based overlay proxy service, including the services accessed by predecessor services such as Oasis and Ocala. Further, the programmability and fine-grained nature of the OpenFlow programming and switching environment offers possibilities which are not presented for earlier writes in this field; in particular, different proxy and overlay services can be accessed on a per-application, per-content, or per-user basis, among others. As future work, it is planed to explore those possibilities, and other rich application areas.

References

[1] New it infrastructure for the information-explosion era, mext grant-in-aid for scientific research on priority areas. http://www.infoplosion.nii.ac.jp/info-plosion/ctr.php/m/IndexEng/a/Index/.

[2] M. Beck, Y. Ding, T. Moore, and J. Plank. Transnet Architecture and Logistical Networking for Distributed Storage. In *Workshop on Scalable File System and Storage Technologies*, 2004.

[3] T. Berners-Lee, R. Cailliau, J.F. Groff, and B. Pollermann. World-wide web: The information universe. *Internet Research*, 20(4):461–471, 2010.

[4] Martin Casado, Michael J. Freedman, Justin Pettit, Jianying Luo, Nick McKeown, and Scott Shenker. Ethane: Taking Control of the Enterprise. In *Proceedings of SIGCOMM*, 2007.

[5] M. Cieslak, D. Forster, G. Tiwana, and R. Wilson. Web Cache Coordination Protocol (WCCP) V2. Technical report, Internet-Draft, http://www.wrec.org/Drafts/draft-wilson-wrec-wccp-v2-00.txt, 2000.

[6] Michael J. Freedman, Eric Freudenthal, and David Mazieres. Democratizing Content Publication with Coral. In *Proc. 1st USENIX/ACM Symposium on Networked Systems Design and Implementation (NSDI '04)*, 2004.

[7] N. Gude, T. Koponen, J. Pettit, B. Pfaff, M. Casado, N. McKeown, and S. Shenker. NOX: towards an operating system for networks. *ACM SIGCOMM Computer Communication Review*, 38(3):105–110, 2008.

[8] S. Iyer, A. Rowstron, and P. Druschel. Squirrel: A decentralized peer-to-peer web cache. In *Proceedings of the twenty-first annual symposium on Principles of distributed computing*, pages 213–222. ACM, 2002.

[9] Dilip Joseph, Jayanthkumar Kannan, Ayumu Kubota, Karthik Lakshminarayanan, Ion Stoica, and Klaus Wehrle. OCALA: An Architecture for Supporting Legacy Applications over Overlays. In *3rd USENIX/ACM Symposium on Networked Systems Design and Implementation*, May 2006.

[10] Arnaud Legout, Nikitas Liogkas, Eddie Kohler, and Lixia Zhang. Clustering and sharing incentives in bittorrent systems. In *Proceedings of the 2007 ACM SIGMETRICS international conference on Measurement and modeling of computer systems*, SIGMETRICS '07, pages 301–312, New York, NY, USA, 2007. ACM.

[11] J.W. Lockwood, N. McKeown, G. Watson, G. Gibb, P. Hartke, J. Naous, R. Raghuraman, and J. Luo. NetFPGA–An Open Platform for Gigabit-Rate Network Switching and Routing. In *Microelectronic Systems Education, 2007. MSE'07. IEEE International Conference on*, pages 160–161. IEEE, 2007.

[12] Harsha V. Madhyastha, Arun Venkataramani, Arvind Krishnamurthy, and Thomas Anderson. Oasis: An Overlay Aware Network Stack. *SIGOPS Operating Systems Review*, 40:41–48, 2006.

[13] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J.Rexford, S. Shenker, and J. Turner. OpenFlow: enabling innovation in campus networks. *ACM Computer Communication Review*, 38(2):69–74, 2008.

[14] E. Nygren, R.K. Sitaraman, and J. Sun. The Akamai Network: A Platform for High-Performance Internet Applications. *ACM SIGOPS Operating Systems Review*, 44(3):2–19, 2010.

[15] Vivek S. Pai, Limin Wang, KyoungSoo Park, Ruoming Pang, and Larry Peterson. The Dark Side of the Web: An Open Proxy's View. In *HotNets-II*, 2002.

[16] Mayur R. Palankar, Adriana Iamnitchi, Matei Ripeanu, and Simson Garfinkel. Amazon s3 for science grids: a viable solution? In *Proceedings of the 2008 international workshop on Data-aware distributed computing*, DADC '08, pages 55–64, New York, NY, USA, 2008. ACM.

[17] KyoungSoo Park and Vivek S. Pai. Scale and Performance in the CoBlitz Large-File Distribution Service. In *Proceedings of the 3rd Symposium on Networked Systems Design and Implementation (NSDI '06)*, 2006.

[18] Ryan S. Peterson and Emin Gun Sirer. Antfarm: efficient content distribution with managed swarms. In *Proceedings of the 6th USENIX symposium on Networked systems design and implementation*, 2009.

[19] Sean Rhea, Patrick Eaton, Dennis Geels, Hakim Weatherspoon, Ben Zhao, and John Kubiatowicz. Pond: the OceanStore Prototype. In *Proceedings of the 2nd USENIX Conference on File and Storage Technologies (FAST '03)*, 2003.

[20] B. Swanson. The Coming Exaflood. *Wall Street Journal*, 20:A11, 2007.

[21] Limin Wang, KyoungSoo Park, Ruoming Pang, Vivek S. Pai, and Larry Peterson. Reliability and Security in the CoDeeN Content Distribution Network. In *Proceedings of the USENIX 2004 Annual Technical Conference*, 2004.