Design of Web Interface for Advanced Content Switch

Thesis Proposal

By

Jayant Patil

as part of requirements for the degree of

Master of Science in Computer Science

University of Colorado, Colorado Springs

Committee Members :

Approved by Date

_____ _____

Advisor: Dr. Edward Chow

_____ _____

Committee member:

_____ _____

Committee member:

1. Introduction

Background :
With the explosive growth of Internet and its increasingly important role in our lives, the traffic on the Internet is increasing dramatically, which has been growing at over 100% annually [1,2]. The workload on the servers is increasing rapidly so that the servers can easily get overloaded within short time, especially for a popular web site. To overcome the problem of overloading, there are two solutions. One is single server solution, i.e. to upgrade the server to higher resources, but that soon will get consumed and the server will become overloaded.

The other is the multi-server solution, i.e. to build scalable server on a cluster of servers [1,2]. When load increases, we can simply add a new server or more into the cluster to meet increased requests. A very efficient way to accomplish this is to use a load balancer to distribute load among servers in the cluster. Load balancing can be done in two levels, transport level using layer 4 switch or application level using content switch [3].

Application level load balancing (also known as content switching) provides the highest level of control over the incoming traffic. When making a load balancing decisions, the content switch can check the header/content of every packet including HTTP meta header, URL, the payload rather than simply checking TCP/UDP port number or IP address. By examining the content of the request, these switches can make decisions on how to route the request to the real servers.

LCS (Linux-based Content Switch)
The Linux-based Content Switch (LCS) is based on Linux 2.2-16.3 kernel and related LVS package [1] implemented by Weihong Wang. LCS examines the content of the request, e.g. URL in HTTP header and XML payload, besides IP address and port number and forwards the request to real servers based on the predefined content switching rules [4].

LCS has been currently being ported on to Intel's network processor study kit, IXP12EB by Longhua Li [10].

IXP1200
The Intel® IXP1200 Network Processor [5] is the cornerstone of the Intel Internet Exchange Architecture (Intel® IXA). It combines the best attributes of a network ASIC with the flexibility, performance and scalability of programmable embedded processor to accelerate development of next generation Internet products. The IXP1200 Network Processor is specifically designed for network control tasks, such as wire-speed switching and routing of packets or cells in real time.

IXP12EB
The IXP12EB Ethernet Evaluation Kit is powerful tool for developing and verifying hardware and software for IXP1200 Network Processor.
The IXP12EB has been already setup in our lab. The configuration is shown in Figure 1.

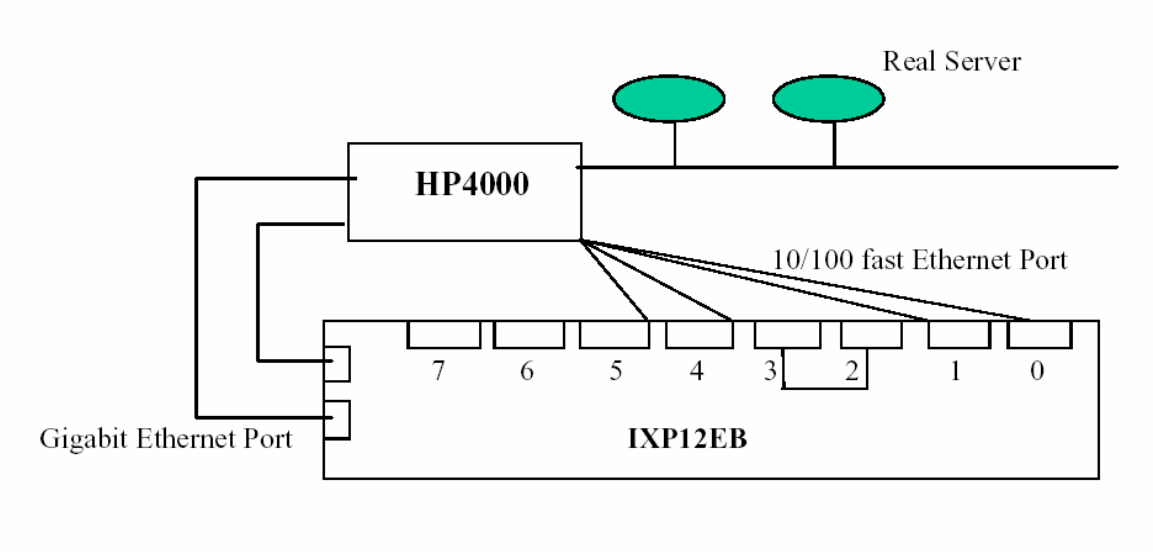

Figure 1.

WindRiver VxWorks and Tornado IDE

Tornado [7] is an integrated environment for software cross-development. It provides an efficient way to develop real-time and embedded applications with minimal intrusion on the target system. Tornado comprises the following elements [7,8]:

- VxWorks [9], a high-performance real-time operating systems
- Application-building tools (compilers and associated programs)
- An integrated development environment (IDE) that facilitates managing and building projects, establishing and managing host-target communication, and running, debugging, and monitoring VxWorks applications

The Tornado environment is designed to provide this full range of features regardless of whether the target is resource-rich or resource-constrained. Tornado facilities execute primarily on a host system, with shared access to a host-based dynamic linker and symbol table for a remote target system. The target server and target agent mediates communication between the host tools and VxWorks.

The development environment is already set up in our lab as shown in Figure 2.

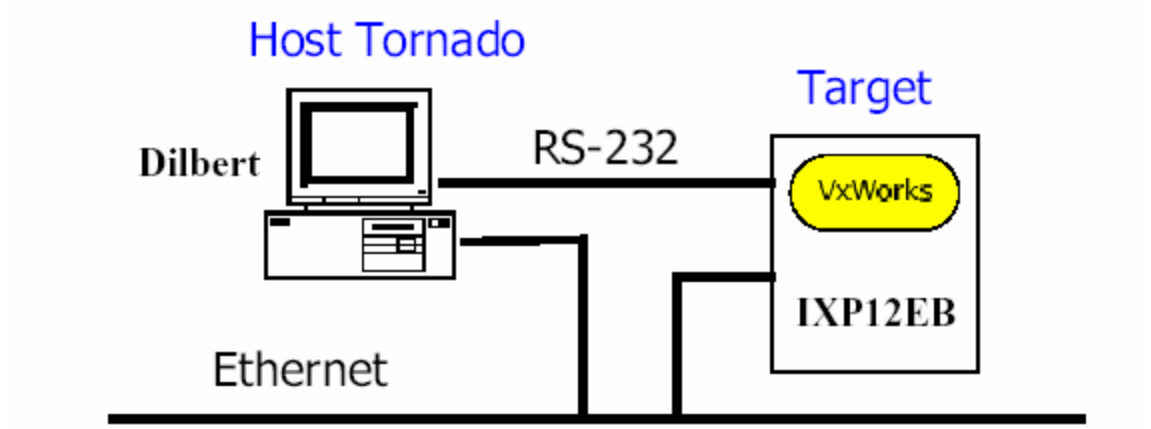

Figure 2. Development Environment set up

Requirements :
As the work on Advanced Content Switch Design (ACSD) has entered in advanced stages, a requirement was felt to have an efficient, reliable and user-friendly interface to the ACSD. This interface can be used to perform lots of configuration tasks as well as any other tasks to interact with the switch. Some examples of such configuration required are the ip address of the switch, configuration of the ports, changing/adding/deleting one or more rules of the rule set.

2. Goal

The goal of this thesis work is to determine the requirements to design an efficient, reliable and user-friendly interface to the ACSD. The interface is needed for configuring the content switch, for updating the content switching rules and network parameter for improving the performance, and for retrieving the network session/statistical data. We will start by evaluating alternate technologies and comparing multiple solutions for the interface.

Let us look at what resources are available to design the interface. First and foremost limitation is that the switch has no hard disk storage. VxWorks does have a simulated file system resident in a part of ram memory. But essentially, there is no elaborate directory structure and file system available for the software. Also, The memory available is also very limited, to the tune of couple of hundred kilobytes.

One more demanding requirement is ease of operation – what this mean is the interface should be easy to use and it should be accessible from most places i.e. any PC on the network.

Looking at the current commercial world, there are two major technologies used in the industry for interface – telnet based and web based. As it became clear in initial discussions, using web technology to design the interface has lots of advantages over other technologies.

3. Plan of work

Alternatives for interface

Currently there are mainly two alternatives used in the industry to provide configuration interface – telnet based and web-server based.
As described above, one of requirements set forth were ability to access and configure the ACSD from any point on the network. Also simplicity of interface was important. Looking at this, I decided to follow a web-server approach. This approach was also obvious considering the fact that the ACSD is a networking device.

Web-server

We will investigate the memory requirements and processing capability of the existing available web server packages including Apache, Java-based web servers, and Goahead. Apache has been the most popular web servers used in the market with the source code available. It is also configurable through the configuration directives and loadable modules

Another possibility was running JVM over VxWorks and using java-based webserver. I looked at couple of java-based web servers – Jigsaw [11], Avenida [12]. There is a Jwork packages available on VxWork. We will investigate the availability of these packages.

Then there is the Goahead webserver [13]. One of first appeal of Goahead was it was already ported on to VxWorks. Also, it had some more benefits for an embedded application like ours like – very small memory footprint and monolithic structure.

Technology to process requests

Another milestone in the research was deciding technology to process requests for configurations and content switching rule update. Once the content switching rule update is downloaded, task creation and switching need to be performed without interrupting the existing sessions. There are various competing technologies available to process the http requests – CGI, active server pages (ASP), java server pages (JSP), PHP, servlets etc.

File-upload

In order to change rules in rule module, best approach is to change and compile the rule base and then upload the binary file up to the switch. We will investigatge the Form-based File Upload in HTML describes detailed protocol handling file-uploading process [14].

Security

Security is a major concern for the networked devices. We will examine the security issue related to the web access. The default choice is the use of SSL. We will examine different levels of secure accesses.

Benchmark and Testing

After the installation of the chosen web server and implementation of the related cgi programs, we will develop benchmark program, test the system and measure the performance of the web-based system management interface on IXP12EB.

4. Deliverables

The deliverables will include:

- Design documentation
- Source code for implementing the webserver on IXP 12EB and interface.
- Testing documentation

5. References

[1]. "Linux Virtual Server", http://www.linuxvirtualserver.org

[2]. High Performance Cluster Computing:Architechures and Systems, Vol 1&2, by Rajkumar Buyya(Editor), May 21, 1999, Prentice Hall

[3]. Gregory Yerxa and James Hutchinson, "Web Content Switching", http://www.networkcomputing.com

[4]. C. Edward Chow and Weihong Wang, "Design and Implementation of a Linux-based Content Switch", to be published in Proceedings of Second International Conference on Parallel and Distributed Computing, Applications and Techniques. http://cs.uccs.edu/~chow/pub/contentsw/status/chow1.doc

[5]. Intel® IXP1200 Network Processor http://developer.intel.com/design/network/products/npfamily/ixp1200.htm

[6]. Intel® IXA (Internet Exchange Architecture) http://developer.intel.com/design/network/ixa.htm

[7]. WindRiver Tornado Development Tools http://www.windriver.com/products/html/tornado2.html

[8]. Tornado User's Guide (Wondows Version) 2.0

[9]. WindRiver VxWorks, http://www.windriver.com/products/vxworks5/index.html

[10]. C. Edward Chow and Longhua Li, "The Design and Implementation of Content Switch on IXP12EB"

[11]. Jigsaw – W3C's Server http://www.w3.org/Jigsaw

[12]. Avenida – 100% pure Java-based web server http://www.serverwatch.com/webserver-avenida.html

[13]. Goahead webserver from GoAhead Software - http://www.goahead.com/

[14]. Form-based File Upload in HTML - http://www.cis.ohio-state.edu/cgi-bin/rfc/rfc1867.html