L

Transparent Ethernet/IPv4 Loadbalancer

Jens Eliasson <jeneli-8@sm.luth.se>
Erik Holmgren <erihol-8@sm.luth.se>
Tim Johansson <timjoh-8@sm.luth.se>
Peter Magnusson <petmag-8@sm.luth.se>
Chirstian Nilsson <chrnil-8@sm.luth.se>

4th June 2001

Abstract

This paper outlines the design decisions behind the development of an OSI Layer 2 and 3 "Transparent Loadbalancer", and the reasons behind those design decisions. The loadbalancer was implemented in a Xilinx XCV1000 FPGA. The project has been carried out by undergraduate students as a part of the Digital Synthesizes course held in 2001 at Luleå University of Technology [LUTH] by the Department of Computer Science And Electrical Engineering [CSEE] in cooperation with Switchcore [SWCR], an ASIC design company which specializes in ASICs for fast networks.

Contents

List of Figures

Introduction

0.1 About the course

The Digital Synthesizes course held in 2001, with strong emphasis on Ethernet and other networking technologies, has been considered one of the most challenging but also most interesting courses attended so far by the authors. The authors would like to thank the people at the Department of Computer Science and Electrical Engineering and the people at Switchcore for providing an excellent course.

0.2 About the authors

The authors are all undergraduate students at Luleå University of Technology, attending the third year of the Computer Science and Computer Engineering program.

0.3 Who did what?

All of the authors participated in the design and the implementation of VHDL files. Tim Johansson did most of the development of C-sources used to test the design. Christian Nilsson provided various hardware, boot disks etc used in the tests. Peter Magnusson wrote most of the report. Jens Eliasson created the figures used in the report.

0.4 Who is this report written for?

The report is written to be possible to understand by people who have some small understanding of network technologies. The authors have attempted to include references to most standards which the report refers to and dedicated an chapter to explain the technologies used by loadbalancers, and clearly define how the report use terms to avoid confusion.

Chapter 1

Why Loadbalance

1.1 What is loadbalancing, and why loadbalance?

In this paper, the term "loadbalancing" is used to refer to the process of multiple network hosts performing the same kind of operation in parallel. As an example, big web sites like yahoo.com and altavista.com have many networks hosts running at the same time, always ready to serve any client connecting and requesting information from the web server.

1.1.1 Scaling resource usage

When a network host perform limited, simple tasks with few clients connecting, most computers are able to perform that job. But if the tasks are complex, or there are many of them, an advanced and expensive computer is needed. As the tasks grow more and more complex and makes use of more and more resources, it is often simpler and cheaper to use a few less advanced computers and let them share the work over a network, thus loadbalancing the work.

1.1.2 Fault tolerance and ease of maintains

As no system is perfect, it does occasionally occur that a network host stops serving the services it is supposed to. There can be many reasons for this, including human faults, broken hardware or software, or that the system must be maintained in a fashion which requires it to be shut down for a while. If a loadbalancer is in use, it can avoid total shutdown of a service as only one host of many equivalent is malfunctioning. Depending on type of error and the ability of the loadbalancer to deal with it, fault tolerance may vary a lot.

1.2 General problems with loadbalancing

Many network technologies aren't designed to be loadbalance. Therefor there might not be any clear specification of how the loadbalancer should work. Developers must invent their own small algorithms to make it work without breaking standards too much. Loadbalancing usually means a more complex network and a single node (the loadbalancer) which all servers rely on.

Chapter 2

Different types of Loadbalancers

There are many different types of loadbalancers and this chapter will explain general difference. It is however beyond the scope of this paper to determine which type is "best" as choice of technology is likely to depend on many factors such as performance, price and ease of usage, among others.

2.1 Non-transparent loadbalancers

In this paper, the term "Non-transparent loadbalancer" is used to refer to systems where the loadbalancing is visible to clients and third parties in some way. In some cases, it is easy and cheap to implement non-transparent loadbalancing.The downside with non-transparent loadbalancers is that they depend on outside systems understanding the loadbalancing system, which they might not do. Another problem is reconfiguration, how fast will outside systems notice changes? In a common implementation, loadbalancing using round robin in the Domain Name System (DNS), changes may take moderately long time to propagate, which limits the ability to use that system for fault tolerance. On the other hand, loadbalancing using DNS round robin is very simple compared to other loadbalancers, as the feature is a part of the DNS system which is a commonly used Internet standard.

2.2 Transparent loadbalancers

Analogously to "Non-transparent loadbalancer", this paper uses the term "Transparent loadbalancer" to refer to systems where the loadbalancing is invisible to clients and third parties. In a number of applications, this might improve fault tolerance and offer more advanced options for loadbalancing. Typically a transparent loadbalancer is a dedicated network node or network device which redirects or modifies network sessions. This system has some possible problems which are important to note: if the loadbalancer for some reason fails, the entire system may go down with it. Also, it is possible that the loadbalancer itself

becomes a bottleneck with either not enough bandwidth or being too slow to carry out it's service properly when load is at its peak.

2.3 Loadbalancing at different OSI Layers

Different loadbalancers operate at different OSI Layers [OSI]. Many loadbalancers operate at OSI Layer 7, the Application layer. This allows the loadbalancer to be used in networks with different or changing network technologies. However Layer 7 loadbalancers are unreliable if you are using services and network protocols not supported by the loadbalancer. As an example, an Layer 7 loadbalancer supporting HTTP only will fail to loadbalance FTP. Many loadbalancer operate on OSI Layer 2 ("Data Link", e.g. Ethernet) and 3 ("Network", e.g. IPv4 [R791]) which are easier to interpret than Layer 7. An OSI Layer 2 and 3 loadbalancer support most Layer 4 to 7 protocols without additional code or hardware. Unfortunately the Layer 2 and 3 information is not always enough for loadbalancing, in Internet applications it is good practice to assume that a Layer 7 application may not have the same sessions identifiers at Layer 1 to 6. This is usually solved by designing the servers share information with each others, perhaps using Java Beans [BEAN] or shared databases.

2.4 Software solutions

With "software solution" this paper refers to systems where a general purpose processor is responsible for most of the loadbalancing process. This may be a solution where a server computer is dedicated to perform loadbalancing, or where a small computer is built into a box (which the average user is likely to consider to be a "hardware device", although this paper consider it to be a software solution).

2.4.1 Advantages of software solutions

Processors and other computer components are widely available, and cheap compared to the price of many loadbalancers. Thus it is rather cheap to begin producing software solutions for loadbalancing. It is also easy to "patch" or "upgrade" software if the developers find a flaw in the design. Software is also rather simple to develop compared to hardware.

2.4.2 Disadvantages of software solutions

Software is slow and processors often require cooling systems. Performance, heat and size is likely to be arguments against software solutions. Also, software solution may be more likely to fail than hardware solution: the hardware the software runs on must function properly, the software itself must function properly, and all other softwares such as the operating system must also function properly. As these dependencies are unwanted, it is recommendable to try to limit them by remove hardware not needed (e.g. sound cards) from the computer running the loadbalancer. It is also recommended to use a stable operating system and have a minimal number of additional softwares running on the computer.

2.5 Hardware solutions

With hardware solutions we refer to systems where most of the loadbalancing is carried out by custom made Application Specific Integrated Circuits (ASICs) - or in the layman's term: "chips".

2.5.1 Advantages of hardware solutions

ASICs can be designed to perform a task using quite little power. They are also very fast and can perform some tasks much faster than an ordinary processor. This makes hardware solutions well suited for systems where performance is important. Hardware is also known to be less likely to crash or fail than software.

2.5.2 Disadvantages of hardware solutions

Hardware is considerably harder to develop than software, which leads to a longer "time to market" than software. Hardware is also badly suited for complex algorithms, such as defragmenting fragmented IP-datagrams and then inspection transport and application layers. Hardware solutions are usually more expensive than software solutions, but if production volumes increases the price difference may become small.

2.6 Software / hardware trade-offs

Software is easy to develop and modify, hardware have better performance. It is possible that a trade-off may be a good solution, such as using configurable FPGAs in real products or combining ASICs with processors.

2.7 More resources

There are many resources available on this subject, and some sites are dedicated to the subject. One such site is loadbalancing.net [LOAD] which is recommended to read, as it contains much information and examples of commercial products from different vendors.

Chapter 3

Design of a transparent loadbalancer

3.1 Choice of technology

As shown in chapter 2, there are many technologies available, making it hard to choose technology. The authors of this report were biased in choice of technology for this project, and being limited in technology was probably a good thing, because designing the loadbalancer was not very easy, especially in the limited time available.

3.1.1 Bias in choice of technology

The project was a part of Digital Synthesizes course, so software solutions were more or less out of the picture even before the design process was initiated. Since all developers had practical knowledge of Ethernet, limited knowledge of other protocols and most of all access to prototype boards with Xilinx XCV1000 FPGAs and Ethernet equipment, Ethernet was more or less an obvious choice of technology. Given the widespread of usage of IPv4 and online documentation of the protocol, IPv4 was also an easy choice.

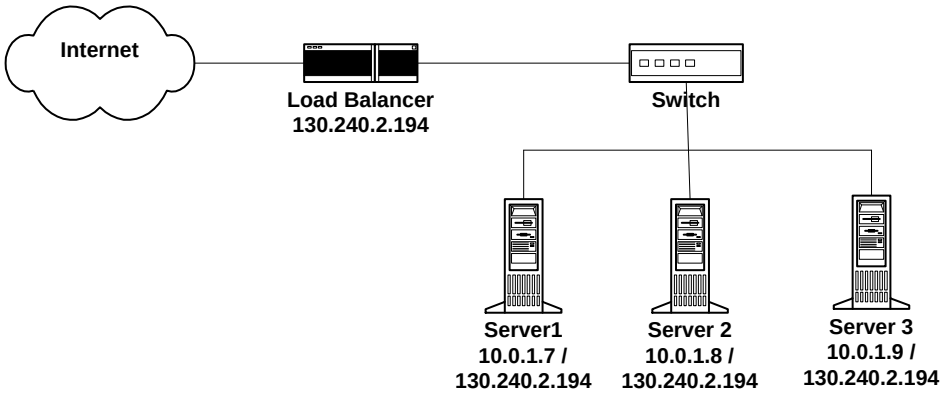
Available documentation

The prime resource used when evaluating an Ethernet/IPv4 solution were "Gigabit Ethernet" [GIGA] and documents from the Internet Engineering Task Force regarding IPv4 and Ethernet [R894, R791, R826].

3.1.2 Technology chosen

- OSI Layer 2 and 3 (Ethernet/IPv4) loadbalancing
- Two Ethernet 10 MBit full duplex interfaces
- FPGA, Xilinx XCV1000
- Cut through technology (saving SRAM for possible future merge with switch)

Figure 3.1: Network Environment



- Data packed into big cells (384 bits of payload) to allow small protocols to operate on single cells.

3.2 Behavior of the loadbalancer

This section describes the loadbalancer is specified to function.

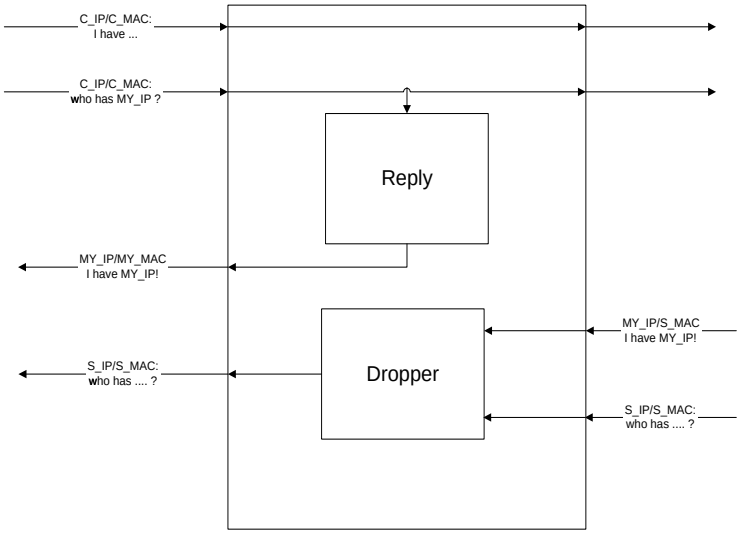
3.2.1 Network environment

The loadbalancer is to be used in Ethernet environments. It will be inserted between two physical Ethernet networks. One of those networks is referred to as "server network" and contains only one kind of Ethernet devices: servers to be loadbalanced. The other network is referred to as "uplink network". The uplink network can consist of a single Internet router or an Ethernet LAN. The loadbalancer has two identifiers, Ethernet address MY_MAC and IPv4 address MY_IP. The identity of the loadbalancer can be configured while operating. The servers in the server network all have configured to have their own IP address as source for general operations, but also a virtual host MY_IP. It is possible to access all servers individually using their normal IP. Figure 3.1 shows how the network could be configured.

3.2.2 Address Resolution Protocol (ARP)

The ARP protocol [R826] is a simple protocol which sends out a request like "who-has 127.0.0.2 ?" when an application attempts to communicate with an Internet host, in this case host 127.0.0.2. When receiving "who-has MY_IP ?" from the uplink network, the loadbalancer will respond "I-do, MY_MAC". It will also let the ARP request pass through into the server network. Since the request will be received by all servers, they will learn the IP and MAC of the sender, and reply. To prevent computers in the uplink network from getting incorrect MACs from the server network, any ARP reply from MY_IP will be dropped when received. Figure 3.2 illustrates how this mechanism works (C_IP is a client IP, S_IP a server IP, MY_IP is the IP of the loadbalancer.)

Figure 3.2: Logical view of ARP behavior



3.2.3 Internet Protocol (IPv4)

If a IPv4 [R791] packet is received from the server network with source address MY_IP the packet will be modified. The Ethernet source address will be set to MY_MAC , in order to make all servers appear to be the same computer. The packet is then forwarded to the uplink network. Packets received from the uplink network with destination address MY_IP will loadbalanced.

How the loadbalancing works

When a packet is to be loadbalanced, a session memory which contains past connections will be read. If the IP source address is not found, the scheduler of the loadbalancer will assign the IP source address an Ethernet destination address which belongs to a server. That source IP / destination MAC pair will be recorded in the memory. If the IP was found in the memory, the previously recorded Ethernet address will be used. The destination Ethernet address of the packet will be substituted by the Ethernet server address assigned to the IP. This allows consistent sessions between one client and server. Old sessions will be removed from the memory by a timeout feature.

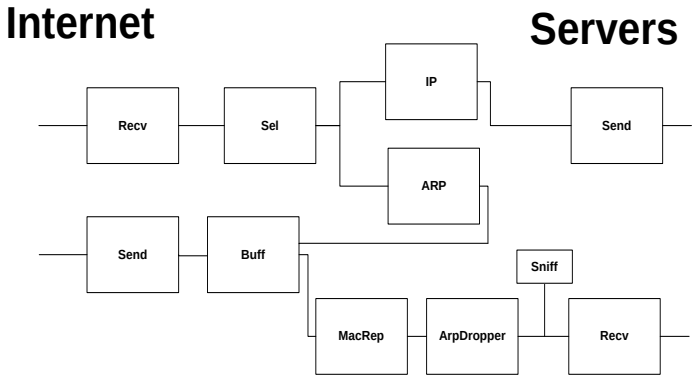
3.2.4 Internet Protocol - Internet Control Message Protocol

The authors considered using the IPv4 sub-protocol ICMP [R792] to verify liveness of servers which have been silent for a while. Lack of time was the major factor behind not implementing this.

3.2.5 Other protocols

Other protocols are not handled by the loadbalancer - it loadbalances IPv4 and responds to certain ARP requests. Other protocols will simply pass through

Figure 3.3: Block design of loadbalancer



the loadbalancer. This is a reasonable default rule: "if I don't know what to do, do the simplest thing and hope I did the right thing". Obviously, this rule will fail to loadbalance most protocols (e.g. IPv6). To implement loadbalancing for other Layer 3 protocols wouldn't be very hard once one protocol has been mastered, but it would take more time than available for the project.

3.3 Block design & implementation

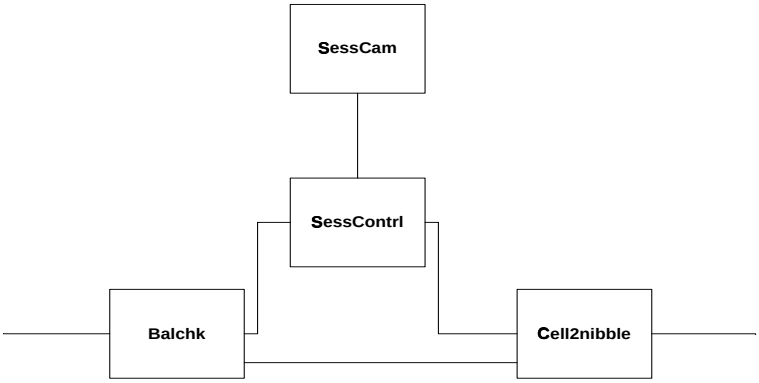
3.3.1 Overview

The figure 3.3 shows the internal structure of the loadbalancer. The uplink network traffic is received by "Recv" which will sample network traffic into cells and forward to a selector unit "Sel". The selector will either only forward to "IP", or to "IP" and "ARP". The "IP" module is the loadbalancing unit which will modify cells before forwarding them to the "Send" module. Server network traffic will be received by another "Recv" unit which forwards to the "Sniff" and the "ArpDropper" unit. The "Sniff" unit allows the system to dynamicly recognize servers being added to the server network, or remove them by timeouts if they are not responding. ArpDropper will dropp ARP Replies from MY_IP (see section 3.2.2 and figure 3.2). Traffic not dropped will be forwarded to "MacRep". "MacRep" will modify the traffic so that the Ethernet source always is MY_MAC if sending IP is MY_MAC. The traffic is then forwarded to "Buff" which also receives output from the "ARP" module. "Buff" will forward ARP replies and server traffic to the "Send" module.

3.3.2 Overview the IPv4 loadbalancing unit

The figure 3.4 shows how the "IP" module is implemented. The cells will be received by "Balcheck", which will forward to "SessControl" or "Cell2nibble". "SessControl" will receive the first cell of any IPv4 packet, and with the aid of the memory module "SessCam" it will be loadbalanced.The scheme loadbalancing is described in section 3.2.3.

Figure 3.4: Block design of IPv4 loadbalancing unit



3.4 Result

3.4.1 Extendibility

- Big cells of data offers rather simple handling of small protocols.
- Can (given some work) be merged with a switch or router.
- One unit, "SessCam", schedules all new sessions to a specific server. Current implementation uses round robin, but it is trivial to implement other schedulers - just write a new compatible unit and wire all information it need to it. Possible future schedulers could attempt to harmonize incoming or outgoing bandwidth. It would not be very hard to interface with agents on the server side either, "Sniff" already contains examples of extracting information from packets sent on the server side.

3.4.2 Drawbacks and limitations

- The design makes use of many resources (gates, slices, memory etc depending on target technology) and consumes most of the slices in the Xilinx XCV1000 FPGA. The design includes e.g. Content Addressable Memories with time out of entries, which consumes many slices. The size of the loadbalancer makes it impossible to merge with the available Ethernet switch code from the assignments during the SMD106 course, given current target technology.
- Many modules uses numeric constants instead of typed constants available in an library file. Preferably all code should use typed constants, so that e.g. one change of cell size should affect all units. With current source, modifying such constants will break most of the design.
- Layer 2 and 3 loadbalancer. General drawbacks with layer 2 and 3 loadbalancer affects this design as well.

3.5 Tests and statistics

3.5.1 Source code statistics

The loadbalancer source code consists of about 25 VHDL-files. Most modules are small, less than 200 lines of code. This can be interpreted as the design is successful in maintaining simplicity while solving a moderately complex task. Version control[1] was only implemented later in the project, when most source code already was written and the authors began putting implemented blocks together. Most logged entries in version control logs are moderately instructive, allowing some limited understanding of committed changes, though logs entries preferably should have been much more detailed. The log shows many committed changes in ARP logic the last few days before the project deadline. The late changes were fast fixes when a previous, somewhat simpler scheme for ARP, was found to be flawed. The current scheme is pretty much the same, but in difference to prior scheme it seems to work.

3.5.2 Implementation statistics, Xilinx XCV1000 realization

Many modules, such as the CAM memory, are written to be generic and can be made smaller or larger depending on single constants. The loadbalancer synthesized with 64 CAM entries and 8 server entries makes use of more than 70% of the Xilinx slices. With 128 CAM entries the synthesizer runs out of slices. The 20 MHz clock frequency does not appear to be a problem.

3.5.3 Testing

Implementation delays due to ARP problems have made it impossible to run advanced tests, so the authors does not guarantee that the design is free from flaws. However, the loadbalancer has successfully been tested on a large network (LAN/Internet) with 2 servers and 12 clients. The testing indicates that the design can cope with network noise[2] and small scale loadbalancing.

Loadbalancer configuration used in tests

The IP 30.240.2.194 (0x82f002c2) was used as MY_IP. Using a small ficl[3] script, the loadbalancer was configured as follows:

```
: wr ! ;
: WriteIP 0x8010000 wr ;
0x82f002c2 WriteIP
```

[1] The authors used the CVS package for Version Control. CVS logs source code changes with a short message written by the person modifying a file. CVS allows many useful features, such as listing all logged messages, comparison between different revisions etc.

[2] broadcasts etc generated by other Internet and LAN hosts

[3] ficl is a small scripting language which the prototype board uses to interface with a bus, allowing simple run time communication with the design.

Server configuration used in tests

Linux servers with IP aliasing support was used in the tests. Therefor the Linux servers where configured as follows:

```
ifconfig lo 127.0.0.1 255.0.0.0
ifconfig eth0 130.240.xxx.xxx 255.255.0.0
ifconfig eth0:1 130.240.2.194
```

Using a small test-server and a couple of clients, the design was put through a couple of tests. The design was explicitly tested to:

- verify ability to perform small scale loadbalancing
- verify ability to allow direct connection to server IPs
- verify that ARP tables etc of clients did not change unexpectedly

All three tests were successful.

Bibliography

[LUTH] Luleå University Of Technology, LTU, 2001-06-04 http://www.luth.se/

[CSEE] Department of Computer Science and Electrical Engineering, CSEE, 2001-06-04 http://www.sm.luth.se/

[SWCR] Switchcore - Silicon for the fastest networks, Switchcore AB, 2001-06-04 http://www.switchcore.com/

[OSI] Protocol Stacks in Relationship to the OSI Model, Lexicon Consulting, 2001-06-04 http://www.lex-con.com/osimodel.htm

[BEAN] Tutorial and Training: Beans, Sun Microsystems, 2001-06-04 http://developer.java.sun.com/developer/onlineTraining/Beans/

[LOAD] LoadBalancing.net, Zerowait Corp, 2001-06-04 http://www.loadbalancing.net/

[R826] IETF RFC 826: An Ethernet Address Resolution Protocol, David C. Plummer, November 1982. http://www.ietf.org/rfc/rfc0826.txt

[R791] IETF RFC 791: Internet Protocol, ISU University of Southern California, September 1981. http://www.ietf.org/rfc/rfc0791.txt

[R894] IETF RFC 894: A Standard for the Transmission of IP Datagrams over Ethernet Networks, Charles Hornig, April 1984 http://www.ietf.org/rfc/rfc0894.txt

[R792] IETF RFC 792: Internet Control Message Protocol, J. Postel, September 1981 http://www.ietf.org/rfc/rfc0792.txt

[GIGA] Gigabit Ethernet: Migrating to High-Bandwidth LANs, J. Kadambi, I. Crayford, M. Kalkunte, ISBN 0-13-913286-4