Burst Round Robin

A Linux Ethernet Bonding Implementation Study

Wilson Ka Wai Shieh

Master of Science in Internetworking

MINT 709 - Capstone Project

Department of Computer Science

Faculty of Science

University of Alberta

Edmonton, Alberta, Canada

Table of Contents

Abstract

In this paper, we document the experience with several link aggregation schemes under Linux. These are implemented in the *bond* kernel loadable module. We introduce a new scheme called burst round robin (BRR). We evaluate two aspects of each scheme: load-distribution and packet reordering.

Keywords

Link Aggregation, Bonding, Bursty Traffic, TCP, Flow, Bonding, Packet Reordering, Load Balancing, Linux Kernel.

1. Introduction

Link aggregation refers to multiple Ethernet network cables or ports used in parallel to increase the link speed beyond the limits of any single cable or port.

It is an inexpensive way to setup a high-speed network that can transfer a lot more data and to allow several devices to communicate simultaneously at their full single-port speed. Another advantage of link aggregation is that it is transparent and poses no impact on protocols and interfaces beyond the end points. With its scalability, reliability and low-cost, it is becoming very popular in today's networks [1].

Having said all the advantages, the most significant challenge in implementing a link aggregation scheme is load balancing - it is necessary to get the most efficiency out of parallel links. However, maintaining load balance may introduce packet reordering. This can cause problems to some popular protocols and applications that rely on in-order packet delivery. Out of order arrivals may be interpreted as packet loss, which may result in packet re-transmission. Out of order arrivals also cause jitter for real-time applications like VoIP. Therefore, the challenge that we face is to preserve the order of the transmitted packet while maintaining the load balance [18] to achieve the maximum efficiency with the bundle of multiple links.

The focus of this paper will be on the link aggregation schemes implemented within Linux.

These schemes include Round-Robin, Active-backup, XOR and much more. We will introduce a new scheme that we will refer to as Burst Round Robin (BRR) as our attempt to tackle the load balancing and packet reordering challenges. We will evaluate the different schemes and look at their performances on the two issues.

The remainder of the paper is organized as follow. In section 2 of this paper, we will look at existing ideas that use multiple low speed lines to form a high bandwidth link, and how they handle the load distribution. Section 3 will introduce the Linux Bonding modules, a description of some of the existing bonding schemes. A new scheme Burst Round Robin will be introduced in section 4, and we will discuss the idea behind BRR. Following that, section 5 will talk about the implementation of BRR in the Linux kernel. Section 6 will show the testing done on the different schemes, their results and comparisons. We will also look at several other related works in Section 7. Finally, this paper will end with a conclusion and discuss any possible future work in Section 8.

2. Link Aggregation Examples

The idea of bonding multiple links has been around for quite some time. Examples would include Multilink Frame Relay, Fractional T1, SuperPipe, and ISDN channel bonding in ISDN, etc. In the following subsections, we will look at these examples. One important aspect of link aggregation is the load balancing issue; we will discuss how load balance is addressed in the following examples.

2.1 Multilink Frame Relay (MFR)

Frame relay is designed to transmit data for intermittent traffic between local area networks and between wide area network end-points. Data are organized in a unit called a frame, which can be of various sizes. Multilink Frame Relay (MFR) is a protocol that combines multiple independent links to provide a single logical link. Using MFR, frames can be fragmented and distributed to the link bundle. Usually a round robin fashion is used, but it does not guarantee sequence of the fragment arrivals due to various link speeds.

A method of distribution is registered as US patent 7184402 [2]. MFR fragments are distributed to one of these links based on a distribution pattern that is based on the total speed of the links and a minimum possible link speed that is supported by the system. A link is selected when it is capable of transmitting the fragments in the fastest transmit time. The fastest transmit time is calculated using the link speed and the transmit time for the link to transmit a fragment allocated previously to the link. Each fragment is distributed to one of the links based on the distribution pattern. This method greatly reduces the chance of out of sequence arrivals.

2.2 T1 Inverse Multiplexing / Multiple T1 Load Sharing

T1 Inverse Multiplexing (Imuxing) combines multiple T1 circuits into one logical data pipe. [3] Data flows are spread across the T1 circuits using a round robin method. It provides scalable bandwidth and fault tolerance with a relative low cost than a high capacity link. A similar but yet different idea to the T1 Inverse Multiplexing is known as Multiple T1 Load Sharing. One

load sharing method “route caching” allots a particular T1 link to each session. With inverse multiplexing, the total available bandwidth for an application will be the sum of all T1 link bandwidth, whereas for load sharing, the bandwidth for an application is limited to one T1 link bandwidth.

2.3 ISDN Channel Bonding

In [**4**], a ramp is described by Burren to allow up to thirty B-channels to be aggregated together and form an Integrated Services Digital Network (ISDN) wide channel (U-channel). Each B-channel is bidirectional channel carrying 64kbps, so the U channel can be up to 30 x 64Kbps. Packets data are transmitted byte-by-byte across all available channels in the wide channel. The ramps, made up of 8 transputers, on the sending and receiving end ensure the data are resembled in the correct order. This setup has allowed formation of larger bandwidth channels in ISDN while maintaining a good load balancing amongst the channels and preserving the data sequence.

2.4 SuperPipe: next generation of high speed trunks

In [5], Deepak introduced a new scheme called SuperPipe to implement a high-speed logical trunk between a pair of core IP/MPLS switches to address the load-balancing issue that other existing schemes face. Its idea is to increase the trunk capacity to 160Gbps by using multiple channels. Incoming IP streams traffics are split by the Packet Distributor at the upstream switch. Each packet is then distributed to one of the sixteen 10Gbps channels using a traffic-meter-based packet dispatching algorithm basically, the channel with the lowest traffic meter reading will be selected to transmit the packet. The packets are recombined at the downstream switch by the Packet Aggregator. This provides a good load balance on the available channels. In order to prevent packet reordering, each packet is sequenced at the Packet Distributor, and there are 16 queues in the Packet Aggregator with a large enough

buffer to absorb any out of sequence packets due to delays. This allows the Packet Aggregator to recombine the packets and maintains their sequential orders.

From all the above examples, we notice that they all have a mechanism to handle the load balancing and packet order issue. In the next section, we will look into the existing bonding facility within Linux and see how multiple interfaces can be aggregated to form one logical interface using different policies.

3. Bonding in Linux

3.1 The Linux Bonding Module

Bonding (bonding.o), a Linux Kernel loadable module, first appears in the Donald Becker's beowulf patches for Linux kernel 2.0. The bonding module can be enslaved to multiple Ethernet interfaces under a common bond interface. It is possible to bond network interface cards from different manufacturers and drivers. The bond and all the slaves will appear to have the same hardware address (which is the primary slave's MAC address), thus creating one logical link. As of version 2.6.18.2, there are 7 different modes available in the bonding module. The following section gives a brief description on each mode.

Mode 0 – Round Robin Policy

Transmissions are received and sent out sequentially on each slave interface in the bond starting from the first available one through the last and restart from the beginning. It provides fault tolerance and load balancing. Although this scheme provides the most complete use of each slave interface, the packets may arrive at the destination out of order.

Mode 1 – Active-Backup Policy

Transmissions are received and sent out through the first available slave interface in the bond. This slave is always used until it fails, and in which case, another available slave interface will be activated for transmission. This policy provides fault tolerance, but has no load balance feature.

Mode 2 – Exclusive-Or (XOR) Policy

This policy defaults to use a simple hash function:

(source MAC address XOR'd destination MAC address) modulo slave count

Using the hash function, one of the slave interfaces is selected for transmission for a particular destination MAC address. This provides fault tolerance and load balancing. One drawback of this scheme is that no bonding advantage can be gained when we look at a single client-host environment, resulting in the same performance as a single non-bonded link.

Mode 3 – Broadcast Policy

Transmissions are received and sent out to all bonded slave interfaces. This provides fault tolerance but will put unnecessary load on to the slave interfaces.

Mode 4 – IEEE 802.3ad Dynamic Link Aggregation Policy

This scheme aggregates groups with the same speed and duplex settings, the transmit hash policy dictates which slave interface will transmit an outgoing traffic. A switch that supports IEEE 802.3ad Dynamic link aggregation is a pre-requisite for using this mode.

Mode 5 – Adaptive Transmit Load Balancing Policy

Outgoing traffic distribution assignment is done based on the current load on each slave. Incoming traffic is received by the current active slave.

Mode 6 – Adaptive Load Balancing Policy

This mode is the same as mode 5 with the addition of Receive Load Balancing for IP traffic using ARP negotiation.

The most commonly used policies are modes 0, 1, 2, and 3; we will be looking into these modes in the subsequent sections.

The challenge of a packet distributor is to evenly spread the network load to all available devices without sacrificing in-order packet delivery. Existing modes in Linux Kernel do provide basic fault tolerance when combining separate interfaces, but does not provide solution to the challenge mentioned above. In the subsequent section, we will discuss another policy that can achieve both load balancing and in-order packet delivery.

4. Burst Switching

4.1 Description

Burst Switching is a load balancing technique used to forward internet traffic using multiple links based on flow-level burstiness. In [6], Shi, MacGregor, Gburzynski described a new bonding scheme that aims to tackle the two main potential challenges that parallel links face: load balancing and packet ordering within individual flows. They identified a condition in which two adjacent packets within a same flow will not be reordered. The condition is that the arrival time difference between the two adjacent packets (T_i) must be greater than the product of the total input buffer size (BSZ), overall system utilization (ρ) and the reciprocal of the physical bandwidth (1/B).

The condition is: $T_i > BSZ * \rho/B$

Since the majority of internet traffic is accounted for by TCP flows, and bursts in TCP are very common, they proposed that if the time between two adjacent bursts is big enough, it is possible to switch the packets to another forwarding engine without ending up with reordered packets.

4.2 Pros and Cons

Burst switching technique is simple, as it can distribute packets evenly to achieve a good load balance in most cases. It also reduces re-ordering rate while keeping high throughput.

However, this scheme does not handle one single long live flow well if that is the only flow in transit. In order to maintain the packet orders, Burst Switching has to sacrifice the load balance aspect and will not be able to distribute packets amongst available channels. A better scheme would be to schedule both potent flows and burst within flows [7].

Also note that the algorithm would require higher layer data depends on the definition of a flow. For example, if we are to define a flow with transport layer data (e.g. port number), this burst switching technique will not be suitable for non-TCP or non-UDP traffic. For other traffic, the flow will be restricted to network layer data.

4.3 Project Goal

The primary goal of this project is to implement the Burst Switching technique mentioned above to the Linux packet scheduler via the kernel Bonding module, and to compare the performance of this technique with existing bonding policies. In the upcoming sections, we will describe the implementation details of Burst Round Robin, followed by the experiment observations and analysis.

5. Implementation Details

5.1 The Linux Kernel

To start off the implementation we decided to use the latest stable version of Linux kernel from www.linuxkernel.org, version 2.6.18.2. The kernel source code was downloaded onto the server and the kernel is configured and compiled.

5.2 The Files

/etc/rc.local (see Appendix A) is modified to load the bonding modules and enslaves available Ethernet interfaces under a bond. This file is automatically executed when the machine restarts.

The following files were modified to implement the new Burst Round Robin scheme:

/usr/src/linux-2.6.18.2/include/linux/if_bonding.h

/usr/src/linux-2.6.18.2/drivers/net/bonding/bonding.h

/usr/src/linux-2.6.18.2/drivers/net/bonding/bond_main.c

In if_bonding.h, a new directive is defined for the new mode Burst Round Robin (BRR), the associated mode number is 7.

In bonding.h, a few structs are created to store the flow information; a few new fields are added to existing structs to store additional information that we need to capture in order to implement BRR.

The file bond_main.c is the main file that defines the bonding policies. The BRR is implemented following the algorithm described below within the function bond_xmit_brr(). A new module parameter interburst_threshold is introduced and to be used exclusively by BRR.

5.3 BRR Pseudo Algorithm

The Burst Round Robin algorithm is based on the discussion in [6]. Packets from the same burst in a flow will be queued to the same Ethernet port. A burst is identified by a configurable parameter value “inter-burst threshold” if two adjacent packets from the same flow arrive within this threshold, the new packet is in the same burst as the previous packet. Detail of the algorithm is discussed below.

A flow table structure, implemented as a doubly linked list, is created to store certain information from each incoming packet within a burst: source IP address, destination IP address, source port, destination port, protocol id, last modified timestamp using the jiffies value, and the name of the device in which the packet is queued for transmission. We use the five-tuple: source IP address, destination IP address, source port, destination port, and the protocol id as the identifier for a flow. As mentioned before, this limits our implementation to work primarily with TCP or UDP traffic, for other traffic, the flow will be based on three-tuple: source IP address, destination IP address, and the protocol id; this is very close to the XOR mode implementation.

For each incoming packet, the flow table is searched based on the flow identifier. If an entry is found and the difference between the current time and the last updated time on the entry is less than the inter-burst threshold parameter, the packet will be queued to the same interface device. If that difference is larger than the threshold, the timestamp is updated and a new interface device is selected.

The interface device that has the earliest time since the last packet is queued will be selected.

If the flow table search returns no result, a new entry will be added to the table, one interface device will be selected based on the same mechanism described above.

6. Experimental Results

6.1 Experimental Setup

In this section we evaluate the performance of different Linux bonding policies as well as the newly implemented Burst Round Robin policy. During the experiments, we used the following hardware setup. On one end, there is a web server hosting many files with different sizes. The web-server is a Sun blade server with an AMD Opteron 148 Processor running at 1GHz. The operating system running on the server is Ubuntu Linux with kernel version 2.6.18.2 with the additional implementation of the burst round robin policy (as described in previous section) in place for the bonding module. Four Ethernet ports are enslaved under a bond. Each port is linked to a Cisco Catalyst 3600 Series XL switch and configured to be full duplex with low speed 10Mbps.

On the other side of the switch, a client laptop is connected with a high-speed link. It is an Intel Pentium M 1.3GHz machine, running Microsoft Windows XP Professional. There is one Ethernet port linked to the Cisco switch running full duplex with speed set to 100Mbps. All these are setup within one VLAN. A simple depiction of the setup can be seen in Figure 1 below:

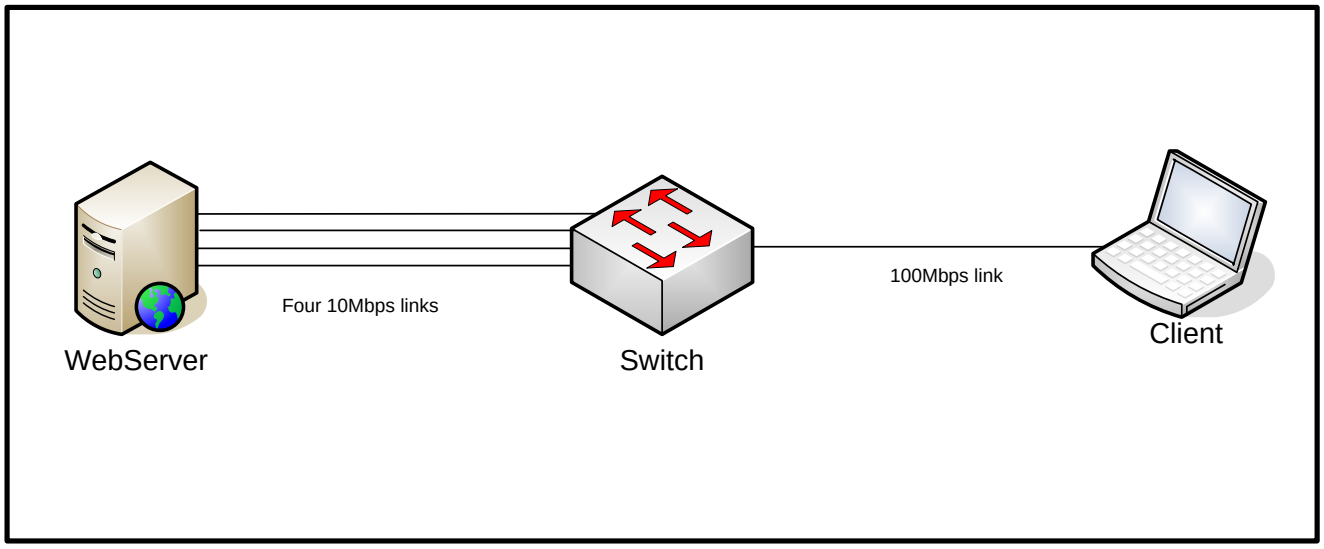

Figure 1

6.2 Experiment Scenarios

A program named http_load [8,9] is used on the client laptop in the experiment. Http_load allows multiple parallel http fetches using one single process. This feature makes the tool favorable to perform testing on the bonding of multiple network interfaces. Overall throughput for the parallel http fetches can be obtained.

The first round of test performed is to compare the result of Burst Round Robin using different interburst thresholds as discussed in section 5.2.

On the server side, several different bonding schemes available in the Linux kernel are tested along with the new burst round robin scheme implemented. The main focus of the comparison is the load distribution and the number of out of order packets. Statistics from each interface are used to obtain the load distribution (number of packets transmitted) for the four links. Ethereal, a network packet sniffer, is used on the client side to obtain the header of the received packets. This allows us to analyze the order of all incoming packets for individual flow and obtain information regarding packet reordering. When we refer to packet order in this paper, we are referring to the packet order for individual flows.

A utility named netem [10] is used to introduce delay on the server Ethernet links. We only paid attention on the packets re-order issue in the native round robin and the burst round robin mode and see how each of these schemes performs under network delay.

6.3 Experimental Measurements

In the experiments, we are obtaining a few different measurements; they include the percentage of packet that is out of order, the throughput and the load balance amongst the Ethernet links.

6.3.1 Out of Order Packets

We adopted the approached used in [11] to measure the rate of out of order packets. We use the output from Ethereal gathered in the receiver end to compute the out of order rate. We kept track of the maximum sequence number found in any received packet so far. When the

next packet arrives, if the sequence number of this new packet is greater than the stored maximum sequence number, it will replace the maximum sequence number; otherwise the out of order counter will be incremented by 1. The scope of the sequence number comparison is within each individual flow. The percentage of out of order packets will be based on the out of order count and the total packets received for all flows.

6.3.2 Throughput

One of the outputs obtained from the http_load utility is the overall throughput for the parallel http-fetches. The value is derived from the total data size received divided by the total elapsed time.

6.3.3 Load Balance

On the server side, before and after each test run, we recorded the total transmitted packets on each available Ethernet interface. The difference of the two values gives us the number of packets sent out by a particular interface. We are also able to compute the load distribution among all the links. We are also interested in the difference between the maximum and minimum percentage load. For example, if the most-utilized interface transmitted 30% of all the packets, and the least-utilized interface transmitted 20%, then the percentage difference will be 10%. The higher the difference, the lesser balanced the load is.

6.4 Experimental Results, Observations and Analysis

6.4.1 Experiment 1

The first set of tests performed is to find out the effect different inter-burst thresholds has on the performance of Burst Round Robin. The test is based on a 12 simultaneous parallel http-fetch.

Looking at figure 2, it is observed that the throughput performance is very close for different inter-burst threshold and different number of parallel http fetches. We can see that with inter-

burst threshold of 4 milliseconds, it yields the lowest throughput compared other higher threshold values. With 4 parallel fetches, it has the best throughput performance compared to 8 and 12 parallel fetches.

As for load distribution, it is the most balanced between the four links is for the 4 parallel fetches. As we increase the number of parallel fetches, the balance is no longer 100% even, but the load distribution is still considered relatively even. Figure 3 shows the summary of the result. The y-axis is the percentage difference of the maximum and minimum link load: the lower the value and the more balance the load is distributed.

There are a very small percentage 1% of packets that are out of order as long as the inter-burst threshold is over 8 milliseconds, please refer to figure 4.

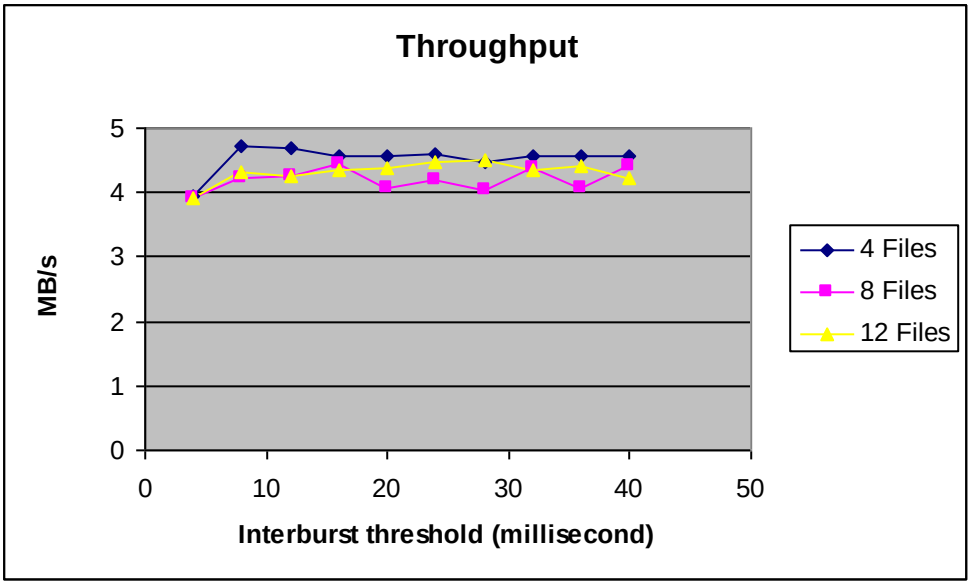

Figure 2

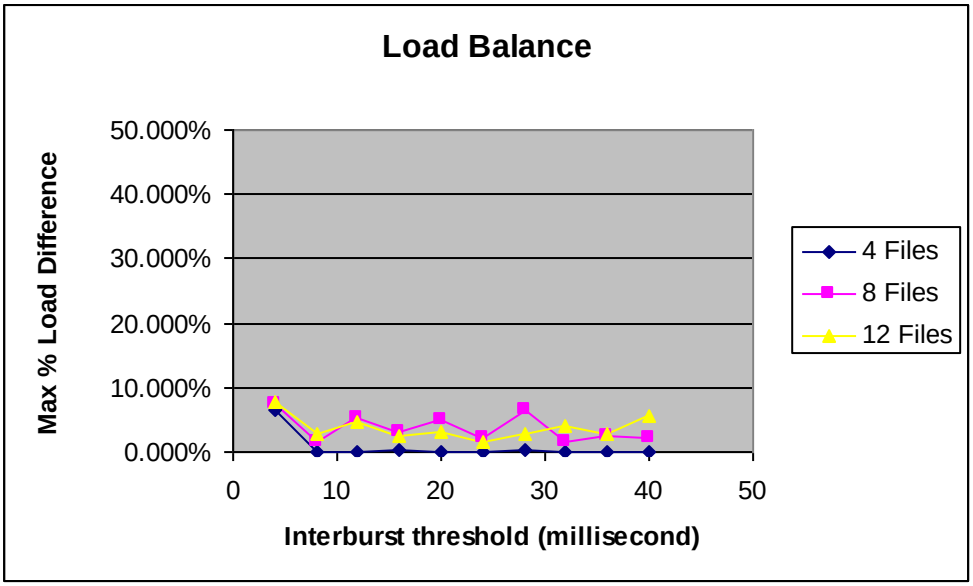

Figure 3

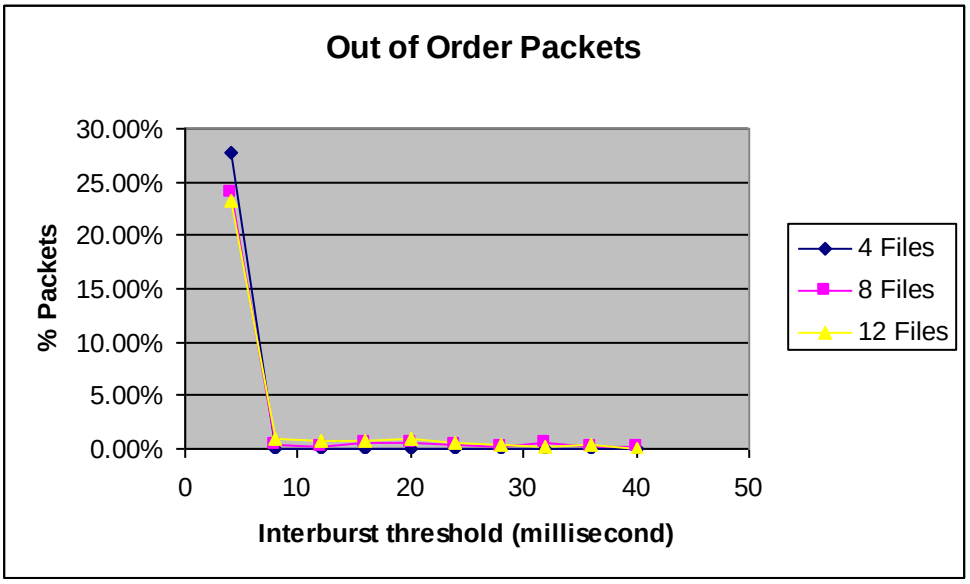

Figure 4

The results are as expected; a threshold of 4 milliseconds is too small. With such threshold, almost every other few packets will be considered as a new burst and thus required an assignment of a new device, resulting in low throughput, uneven load distribution and a high percentage of out of order packets.

6.4.2 Experiment 2

The second set of experiment is to compare the different available bonding modes: mode 0, 1, 2, 3, and 7, which are Balance Round Robin, Active Backup, Balance XOR, Broadcast and Burst Round Robin respectively. For each mode, we run a parallel http-fetch of files (ranging from 1 file to 20 files) from the server to the client. Note that for Burst Round Robin, we are using an inter-burst threshold of 12 milliseconds. The results are summarized in the following graphs.

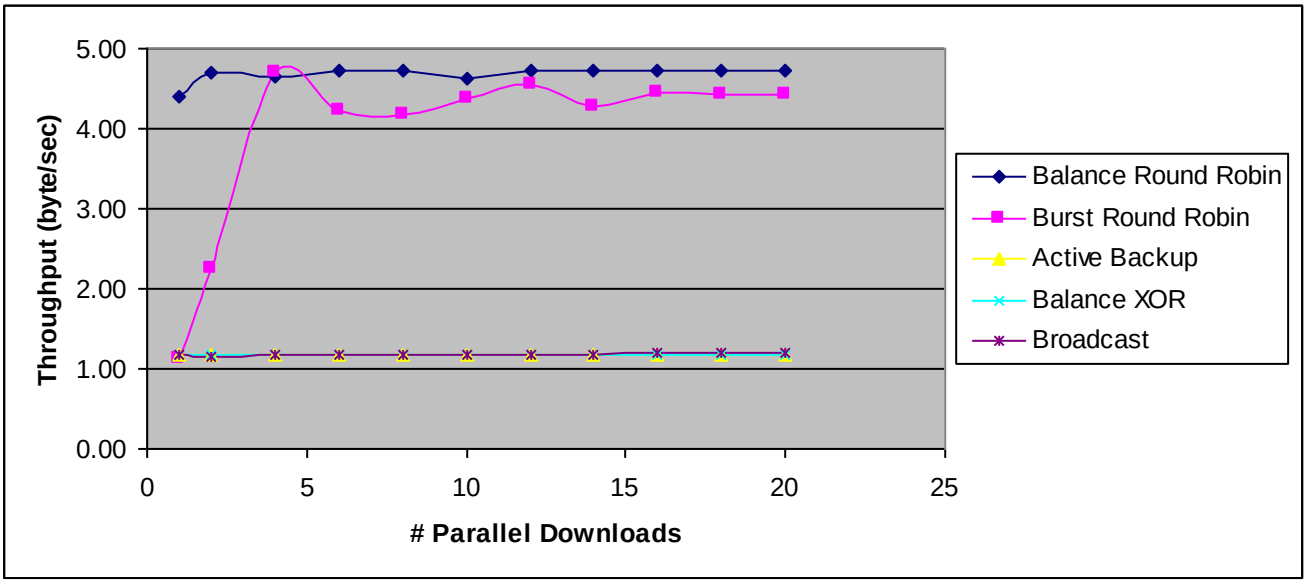

Figure 5

When there is 1 single file http fetch in our setup, the Balance Round Robin outperformed the rest of the bonding modes by a factor of 4. This is as expected as Balance Round Robin utilizes all available links even if there is only 1 file (1 flow). For the remaining modes, bonding has no effect when there is only 1 flow.

For Active Backup, it always uses the same link as long as it does not fail, so we do not see any changes in throughput for different number of flows. For Balance XOR, we see a constant throughput as well because we only have one laptop (one address), which resulted in the same Ethernet link being selected every time. For Broadcast, packets are sent to all available links, so the throughput is the same as 1 file on 1 link.

The more interesting modes to compare are Balance Round Robin and Burst Round Robin.

For Burst Round Robin, throughput increases as the number of parallel downloads increases from 1 to 4. At 4 parallel downloads, throughput reaches its peak value, which is as expected since all 4 links are utilized to distribute the packets for the 4 downloads. The Balance Round Robin has a pretty even throughput in different number of downloads, as this scheme has no dependency on the number of flows. The throughput of Balance Round Robin is always slightly higher than that of the Burst Round Robin. Possible reasons account for this could be the overhead involved in the Burst Round Robin with all the flow table searching, re-assigning packets flow based on burst to different links.

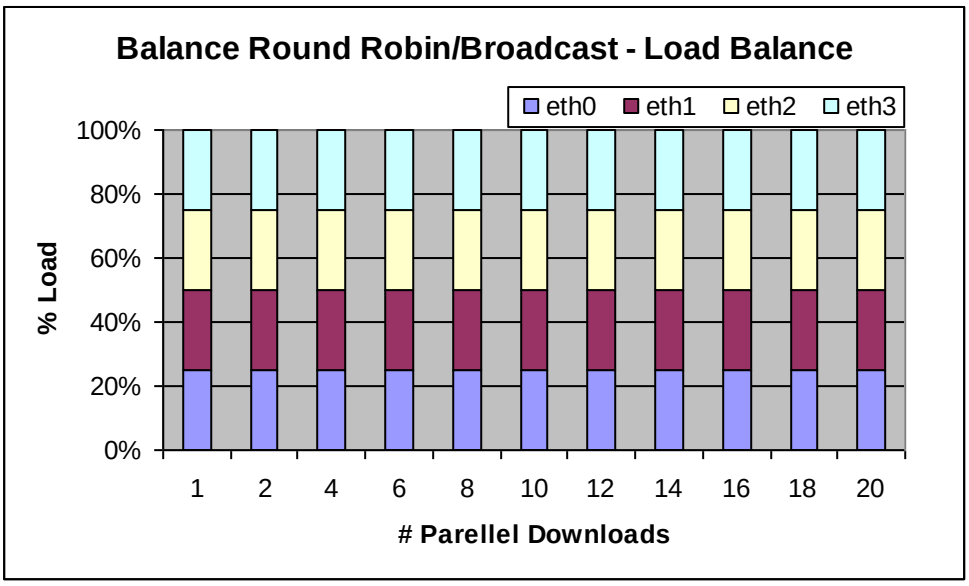

Figure 6

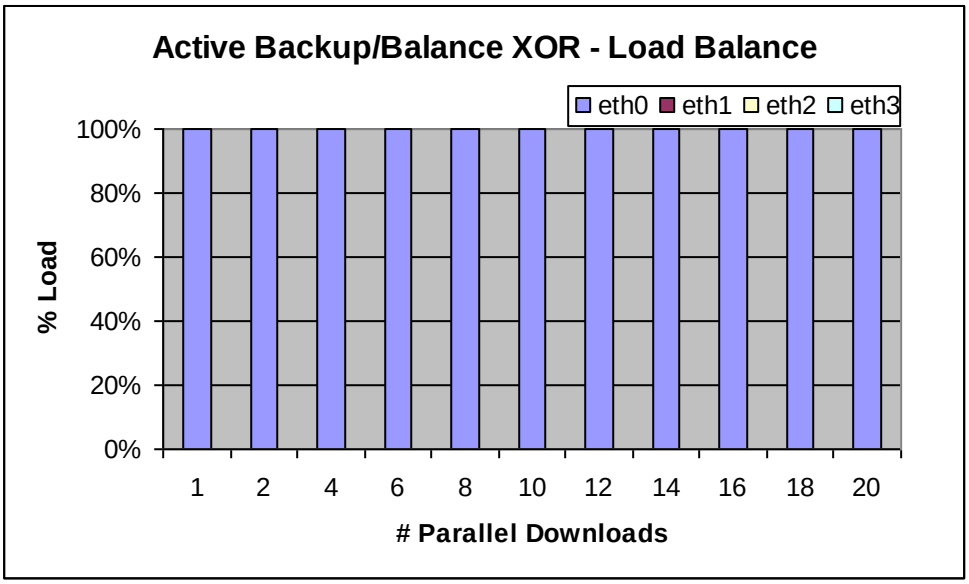

Figure 7

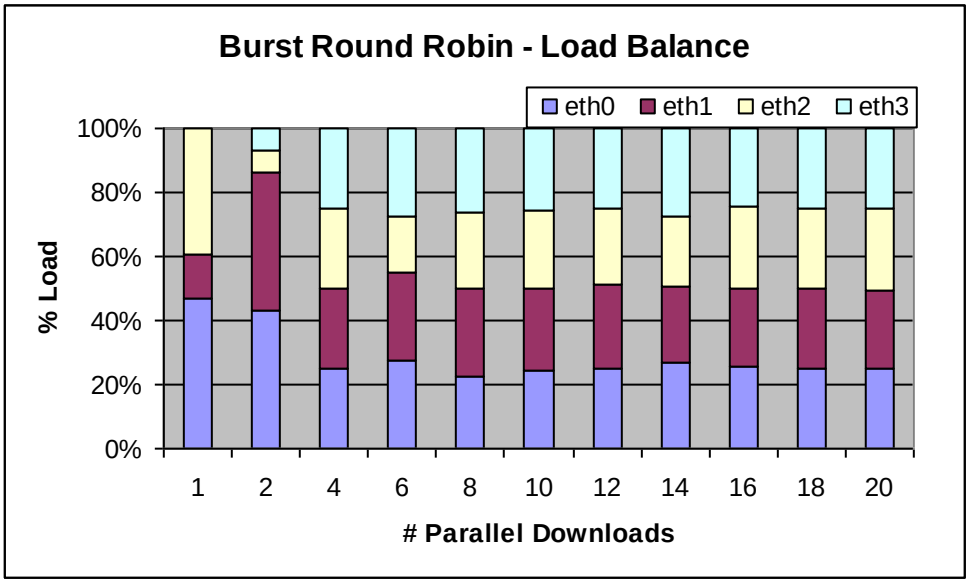

Figure 8

For Balance Round Robin and Broadcast modes, the same number of packets is distributed to all 4 available links; therefore you see an even load distribution in Figure 6.

As mentioned earlier on, Active Backup and XOR utilize only 1 link, so there is no load balance between the 4 available links at all; all loads are piled onto a single link (Figure 7)

The Burst Round Robin provides a more distinctive load balance graph (Figure 8) compared

with the rest. With 1 file, 3 links are used. When a new burst of the flow arrives, the link that has been idle the longest will be selected. From the graph, we can see that there are actually only 3 bursts in this particular flow.

As the number of parallel downloads increases to 2, all 4 links are used sometime in the download. The two flows will be using two different links to start, and with the changes in burst, other links will be utilized.

For the rest of this test, we incremented the number of parallel downloads by two each time. We can see that if the number of parallel downloads is a multiple of 4 (i.e. 4, 8, 12, 16, and 20); the load distribution is very even, approximately 25% for each of the link. When the number of downloads is 6, 10, 14 and 18, we see a small fluctuation on the load distribution, it is still relatively even but not as close as 25% each.

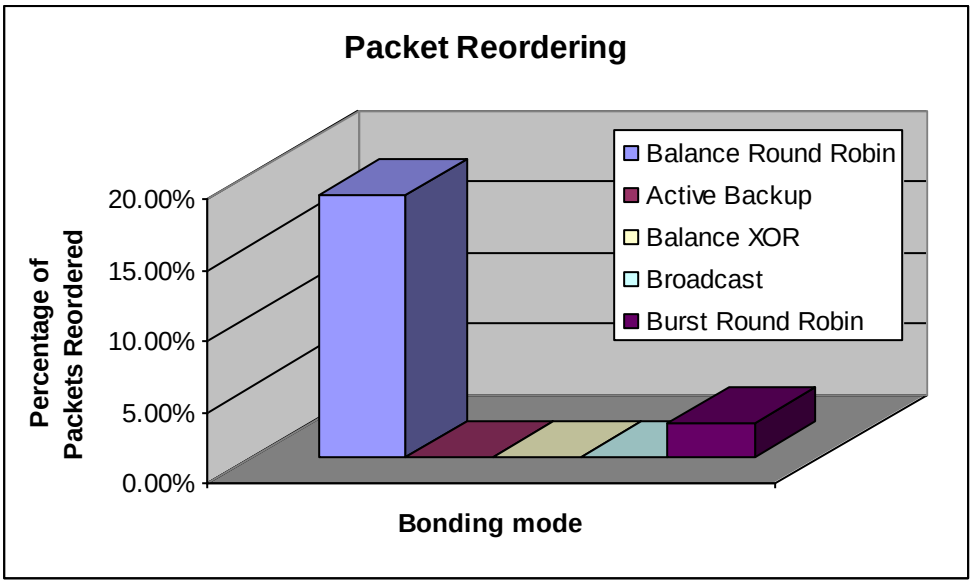

Figure 9

We look at the packet reordering by running a 12-file download:

Active Backup, Balance XOR and broadcast are all packet ordering tolerant.

Packet reordering only happens in Balance Round Robin and Burst Round Robin. Balance Round Robin has close to 18.5% of packets that are out of order.

The Burst Round Robin packet re-order rate is about 2.4%, significant less than Balance Round Robin. The small re-ordering rate could be further reduced if we increase the inter-burst threshold parameter value.

6.4.3 Experiment 3

The last set of tests is to compare the out of order rate on Balance Round Robin and Burst Round Robin, particularly in different amounts of delay. Delays are introduced by a utility netem on the 4 available links. The test is carried out with a 4-file parallel download. The inter-burst threshold for Burst Round Robin is set to 12 milliseconds.

The following graphs summarize the results of the tests:

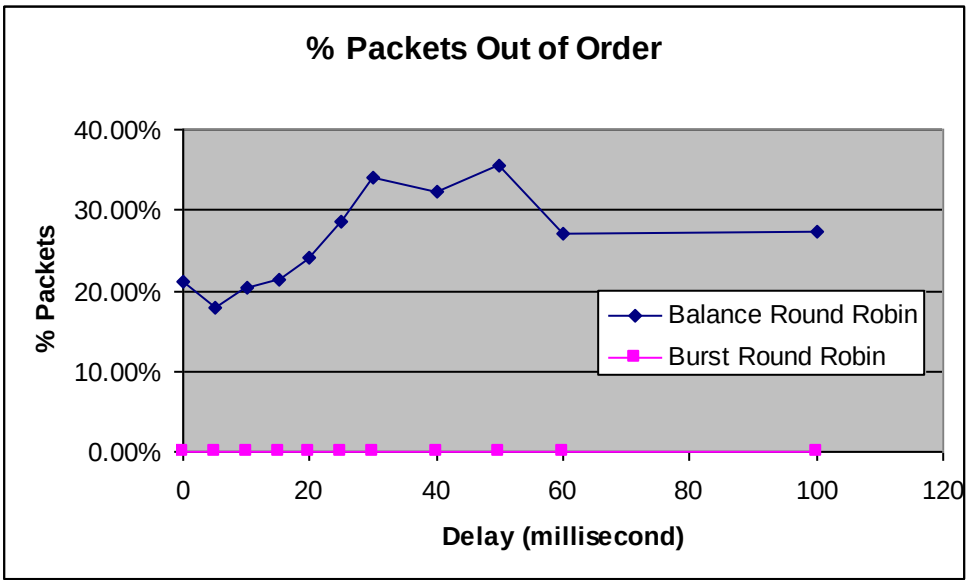

Figure 10

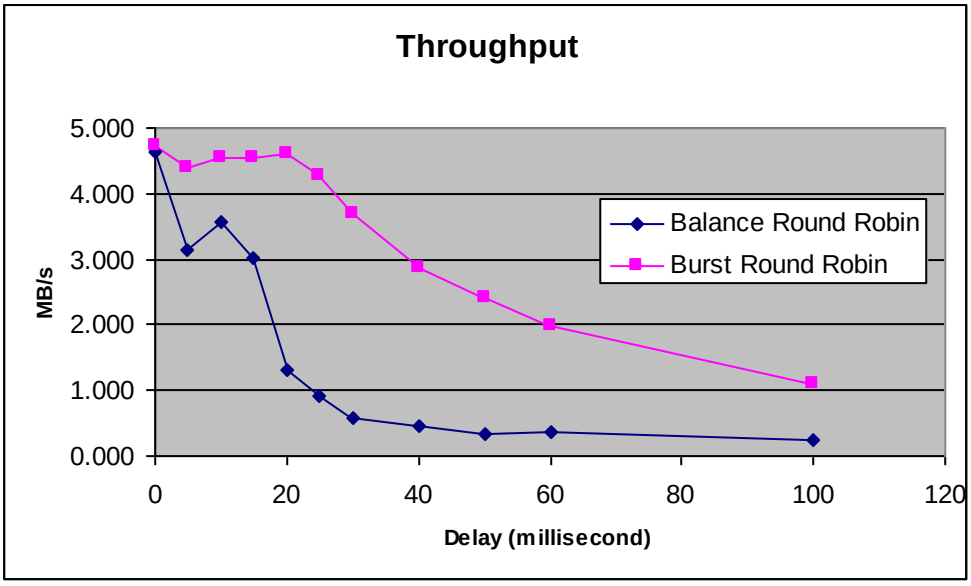

Figure 11

As we introduce delay to the parallel connections between the switch and the server, we notice that for Balance Round Robin, the packet out of order rate increases as delay increases from 0 ms to 50 ms. On the other hand, delay has no effect on Burst Round Robin in terms of the packet reordering.

We also look at the effect of delays on the overall throughput for each of the bonding schemes. Balance Round Robin significantly slowed down in throughput. It drops from 4.7 MB/s down to 0.2 MB/s when a delay of 100 ms is introduced. Burst Round Robin sustains a relatively high throughput as delay goes up to 25 ms, then the throughput drops as delay increases. Comparing the throughputs of these two schemes at a delay of 100 ms, Burst Round Robin is about 5 times faster than Balance Round Robin.

7. Related Work

There have been different efforts in the research community to improve the performance on Linux channel bonding. In this section, we will look at other approaches researcher took to increase the performance on link bonding.

One interesting direction is demonstrated in [12], where Andersen and Hanenkamp used a number of different scheduling algorithms for heterogeneous interfaces (i.e. gigabit Ethernet interface and fast Ethernet interface). They introduced the following bonding modes: N:1 mode, Least-Queue mode N:1/Primary mode, and the Cut-off-Least-Queue mode. Their experiments showed great performance gain.

- N:1 mode – Since they are using interfaces with two different speeds, this mode is implemented just like Balance Round Robin, but using a N:1 ratio. For example, if an interface is 10 times faster than the other, N will be 10, 10 packets will be queued to the faster interfaces for every packet queued to the slower interface.
- Least-Queue – Packets are directed to the device with the least queue length. If the queue length is the same, the faster interface will be chosen.
- N:1/Primary – A threshold of packet size is set. If the packet size is below this threshold, the N:1 strategy will be used. Otherwise, the larger packets will be queued to the primary interface that has a higher bandwidth.
- Cut-off-Least-Queue – Large packets that exceed the size threshold will be sent to the primary interface, and smaller packets will be queued using the Least-Queue scheme. The mode is best suited for high volume of small packet traffic.

The different modes suggested above placed main focus on load balancing to achieve high throughput, but it does not have any mechanism in place to prevent possible packet reordering.

Gabler suggested another bonding mode in [19], a new algorithm based on layers 3 and 4 data is created and is now widely used by network switch manufacturers and is also implemented into the IBM AIX operating system. The channel assignment is done using the following formula:

Channel = ((src port XOR dst port) XOR (src IP XOR dst IP)) modulo (number of channels)

This algorithm increases the hashing variability and reduces the chance of disproportional load balance. This scheme resembles our Burst Round Robin in a few ways – mainly that it is also utilizing flow information. But since it depends on the transport layer data as well, it has the same limitation that it only works with TCP and UDP traffic. The main difference between Burst Round Robin and Gabler's new hashing algorithm is that our Burst Round Robin takes advantages of the bursty characteristics and is able to further spread the load more evenly amongst the links.

8. Conclusion and Future Work

In this paper, we discussed a new link aggregation scheme - Burst Round Robin. This scheme takes advantage of the characteristics of bursty Internet traffic. An implementation was made in Linux kernel bonding module and a series of experiments were performed to compare existing bonding modes with the Burst Round Robin. All the experiments have shown good performance on the new scheme. Burst Round Robin appears to provide good load balancing on the parallel connections, while maintaining an even load balance. It also maintains high throughput and is resistant to out of order packet deliveries.

Some possible future work on this area would be to test the Burst Round Robin with a more complicated environment and configuration setup. For example:

1) Implement bonded links in both the server and receiving machine with Burst Round Robin, and connect them through a switch;

2) Server with Burst Round Robin implemented bonded links connects to a receiver with a single 10 Gbps link via a switch;

3) Server with a single 10 Gbps link connects to a receiver with Burst Round Robin implemented bonded link via a switch;

4) Replace the switch in previous example by a large network;

5) Use more realistic traffic patterns between sender and receiver, e.g. different flow sizes and flow timings;

6) Increase delay on the receiver end;

7) Improve the efficiency of the implementation of the Burst Round Robin scheme:

a) A dynamic inter-burst threshold that auto-adjusts based on the network traffic;

b) Removal of flow table entry if an entry expires to reduce the size of the flow table.

9. Appendices

9.1 Appendix A – File/Configuration

/etc/rc.local file:

```
#!/bin/sh -e
#
# rc.local
#
# This script is executed at the end of each multiuser runlevel.
# Make sure that the script will "exit 0" on success or any other
# value on error.
#
# In order to enable or disable this script just change the execution
# bits.
#
# By default this script does nothing.
#
#added by WS
modprobe bonding mode=7 miimon=100
modprobe e100
ifconfig bond0 10.3.31.194 netmask 255.255.255.0 up
ifenslave bond0 eth0
ifenslave bond0 eth1
ifenslave bond0 eth2
ifenslave bond0 eth3
#end - added by WS
exit 0
```

10. Acknowledgement

The project idea is from Dr. Weiguang Shi and his preliminary manuscript [13] on this topic.

Thank you to Dr. Mike H. MacGregor and Dr. Weiguang Shi for their help in shaping the scope of this project. Thank you to the MINT program in University of Alberta for providing the lab facility and test equipment.

11. References

[1] Jon C. R. Bennett, Craig Partridge, and Nicholas Shectman, "Packet Reordering is Not Pathological Network Behavior," IEEE/ACM Transactions on Networking, vol. 7, No. 6, pp. 789-798, December 1999.

[2] US patent 7184402 [online]. Available from: http://linkgrinder.com/Patents/Method_for_mult_7184402.html [Accessed Nov 2006]

[3] Inverse Multiplexing, ComTest Technologies Inc. [online]. Available from: http://www.comtest.com/tutorials/imux.html [Accessed Oct 2006]

[4] J. W. Burren. Flexible Aggregation of Channel Bandwidth in Primary Rate ISDN. Symposium proceedings on Communications architectures & protocols, pp. 191-196, 1989.

[5] D. Mathur. SuperPipe: Next generation of high speed trunks. ACM International Conference Proceeding Series; Vol. 90, Proceedings of the 2004 international symposium on Information and communication technologies, pp. 38-43, 2004.

[6] W.Shi, M. H. MacGregor, and P. Gburzynski. A Scalable Load Balancer for Forwarding Internet Traffic: Exploiting Flow-level Burstiness. Proceedings of the 2005 symposium on Architecture for networking and communications systems, pp. 145-152, 2005.

[7] W. Shi, M. H. MacGregor, and P. Gburzynski. Load balancing for parallel forwarding, *IEEE/ACM Transactions on Networking*, vol. 13, no. 4, 2005, pp. 790-801, 2005.

[8] Http_Load website [online]. Available from: http://www.acme.com/software/http_load/ [Accessed Dec 2006]

[9] Http_Load website [online]. Available from: http://www.orenosv.com/misc/ [Accessed Dec

2006]

[10] Netem website [onlilne]. Available from: http://linux-net.osdl.org/index.php/Netem [Accessed Jan 2007]

[11] W. Shi, L. Kencl. Sequence-Preserving Adaptive Load Balancer. Proceedings of the 2006 ACM/IEEE symposium on Architecture for networking and communications systems, pp. 143-152, 2006.

[12] D. Andresen, S. Hanenkamp. Heterogeneous Channel Bonding Revisited. Proceedings of the IASTED International Conference on Parallel and Distributed Computing and Systems (PDCS 2003), pp. 387-392, 2003

[13] Weiguang Shi. Burst Round Robin: a Load Balancing Scheme for Link Aggregation.

[14] Linux Kernel Bonding Documentation. Available from: http://lxr.linux.no/source/Documentation/networking/bonding.txt?v=2.6.18 [Accessed Sep 2006]

[15] A. Hendel. Link Aggregation Trunking. IEEE 802 – Training Session, November 11, 1997. Available from: http://www.ieee802.org/3/trunk_study/tutorial/ahtrunk.pdf [Accessed Sep 2006]

[16] Linux Kernel website [online]. Available from: http://www.kernel.org/ [Accessed Sep 2006]

[17] Linux Kernel Newbies website [online]. Available from: http://kernelnewbies.org/ [Accessed Sep 2006]

[18] J. Bennett, C. Partridge, and N. Shectman, "Packet Reordering is Not Pathological Network Behavior," IEEE/ACM Transactions on Networking, vol. 7, No. 6, pp. 789-798, 1999.

[19] J. Gabler. Better Bonding Ethernet Load Balancing. Lawrence Berkeley National Laboratory. Paper LBNL-58935. Sep. 2006

[20] Linux Channel Bonding SourceForge Project [online]. http://sourceforge.net/projects/bonding [Accessed Sep 2006]