

Advanced Computer Networking (CS 591)
Term Project - Spring 2003

Implementation of speculative defragmentation
for UDP using programmable Alteon Acenic NIC

Bharat Krishnamurthy – bharat@cs.unm.edu

Sumanth Tamma – sumanth@cs.unm.edu

Quadir S Kareemullah – quadir@cs.unm.edu

Vinod K Doshi – vdoshi@cs.unm.edu

Contents

- 1.Introduction
- 2.Currrent Scenario
- 3.The modification
- 4.The Environment
- 5.The firmware
- 6.Implementation
- 7.Results
- 8.Conclusions
- 9.References
- 10.Acknowledgements

Introduction

The Alteon Acenic is a programmable PCI based Gigabit Ethernet Network Interface Card at the heart of which lie two 32 bit processors running at 88Mhz each. PCI bus is a high performance, synchronous bus with a maximum bandwidth of 528Mbps while operating at 66Mhz on a 64 bit bus.

The local memory map on the NIC , implemented with a 32 bit address space is used to access the external SRAM, external Flash and the internal Tigon registers.

The NIC supports 10, 100 and 1000M bits per seconds transfer speeds and supports both 32 bit and 64 bit hosts.

The current scenario

The gigabit ethernet technology is used mainly in high performance computing where the occurrence of packet losses and garbled messages is very low. Yet, a lot of processing and some network bandwidth is spent in checking for the ordering of the packets , checksumming and packet processing. This can be avoided if we assume that the incoming packets are ordered and correct. This project aims to address this issue and provides an implementation that attempts to improve the performance of the host computer while not degrading the performance of the network.

The Modification

Our modification of the existing implementation uses the UDP data stream and assumes that all the packets arrive in order and are correct. Therefore at the sending side of the transmission, we send data in packet sizes of 4k each which are then split by the driver into 3 ethernet frames of maximum size 1500 bytes . These frames on receipt are buffered into a single 3k sized packet (based on the speculation that they arrived in order) and then sent to the host as a single packet. This reduces the number of interrupts to the kernel and gives the host processor more time to process applications and reduces the time taken to process the packets.

The Environment

The environment we set up for the project involved 2 Intel Xeon based systems running at 2GHz with 256MB of RAM, each with Alteon Acenic Gigabit Network Interface Cards. The operating system was linux kernel 2.4.18. These computers were connected directly without a switch in between .

The firmware

The original source code for the firmware was obtained from jemez.cs.unm.edu and used with the trace enabled driver .To compile the source files and build them in to the required firmware, we used the firmware rebuild tools available on the SSL website.

Implementation

We created a temporary buffer on the NIC of size 4096 bytes. Every time a packet was received, the packet was stripped of the ethernet header and appended to the buffer. A local variable stores the number of packets buffered. When the number of packets buffered reaches 3, the temporary buffer is sent to the host by DMA along with the corresponding buffer descriptor. Once this is done, the temporary buffer is cleared and the local variable reset to zero. This code speculatively defragments the incoming ethernet frames into a single 4k sized IP packet. We also disabled the IP checksumming in the driver and the kernel.

The function `bcopy()` provided in the file `mem.c` of the firmware source was used to buffer the data (as shown in fig. 1)

```
#define SPEC_PACKS 3
U32 spec_address[SPEC_PACKS];
char spec_packet[4096];
U32 total_length=0,spec_count;

void
h_mac_rx_comp(void)
{
.....
.....
spec_address[spec_count]=hd.w0;
spec_length[spec_count]=(md_w1 & TG_MAC_RX_W1_LEN_MASK) - 4 // the -4 is to leave out the checksum
.....

if (spec_count==0 || spec_count==3){
spec_count=0;
spec_length[spec_count]=max_len;
bcopy((U32 *)cur, (U32 *)spec_packet,spec_length[spec_count]);
spec_bdp[spec_count]=bdp;
spec_count++
}
else if (spec_count<SPEC_PACKS){
spec_length[spec_count]=max_len;

bcopy((((U32)cur)+14),(U32*)spec_packet+total_length,(spec_length[spec_count]-14));
spec_bdp[spec_count]=bdp;
total_length+=spec_length[spec_count]-14;
spec_count++;
}
```

fig. 1 : the defragmentation code

The buffered packet was sent to the host when the buffer was filled .

```
if (spec_count==3)
{
    if (!if (!(*nicfp->q_dma_to_host_stub)
        (bdp->BD_host_addr,
        spec_packet,
        total_length,
        TIGON_TYPE_RECV_DATA,
        ring_index,
        nicfp->dma_to_host_data_state))
}
```

fig. 2 : DMA of the buffered packet

Results

The table below shows the CPU utilization and network throughput given by netperf using the default acenic driver module for the command:

netperf -H<hostaddress> -l60 -C -tUDP_STREAM -- -m4906

Trial no.	Throughput 10 ⁶ bits/sec	CPU Utilization %US
1	802	41.9
2	796.6	38.57
3	802	39.93
4	797.5	42.52
5	798	42.64
6	798.1	42.43
7	792.4	43.84
8	809.8	41.58
9	787.6	47.08
10	811	45.09

Fig. 3 : Readings obtained at MTU 1500 bytes with the original Acenic module.

Trial no.	Throughput 10 ⁶ bits/sec	CPU Utilization %US
1	942.9	50.43
2	914.3	48.99
3	935.7	51.53
4	930	47.53
5	885.5	59.38
6	923.8	63.93
7	917.6	56.3
8	936.7	58.34
9	922.7	51.3
10	923.8	50.48

Fig. 4 : Readings obtained at MTU 4096 bytes with the original acenic module.

Trial no.	Throughput 10 ⁶ bits/sec	CPU Utilization %US
1	790.2	35.6
2	799	34.88
3	795.4	37.2
4	795.9	39.6
5	780	36.34
6	798.1	40.66
7	801.2	38.85
8	805.6	42.32
9	799.2	41.76
10	799.6	43.22

Fig. 5 : Readings obtained at MTU 1500 bytes with the modified acenic module.

Trial no.	Throughput 10 ⁶ bits/sec	CPU Utilization %US
1	942.9	51.33
2	914.3	49.8
3	935.7	46.23
4	930	44.36
5	885.5	45.42
6	923.8	48.22
7	917.6	52.68
8	936.7	52.6
9	922.7	53.45
10	923.8	49.8

Fig. 5 : Readings obtained at MTU 4096 bytes with the modified acenic module.

The graphs below show a comparison of the readings obtained with the original firmware and our modified firmware.

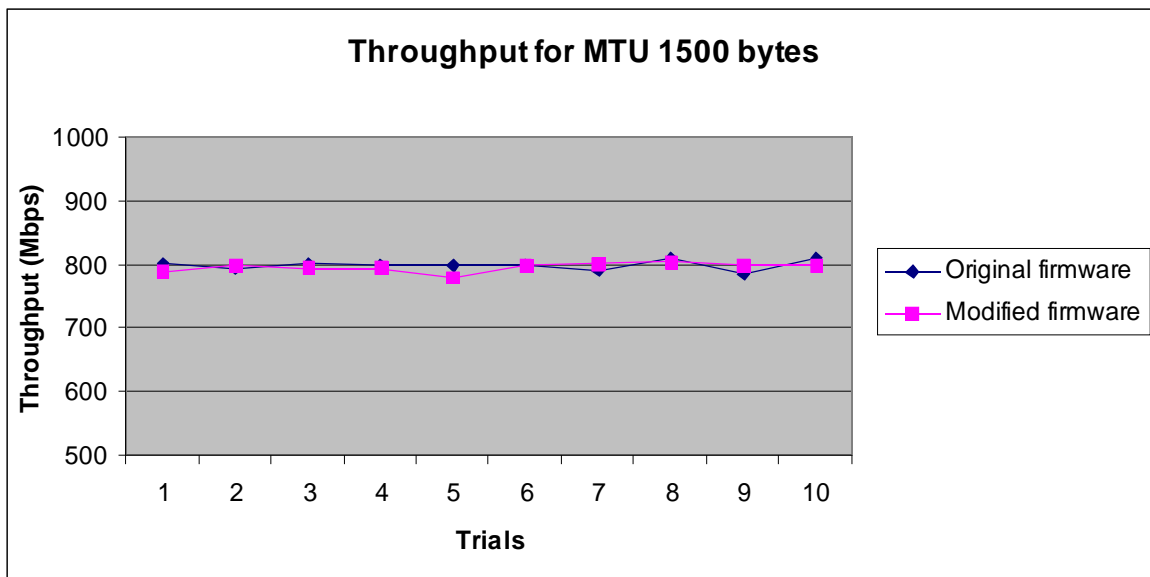


Fig 6: A comparison of throughputs with the two firmwares for MTU 1500 bytes

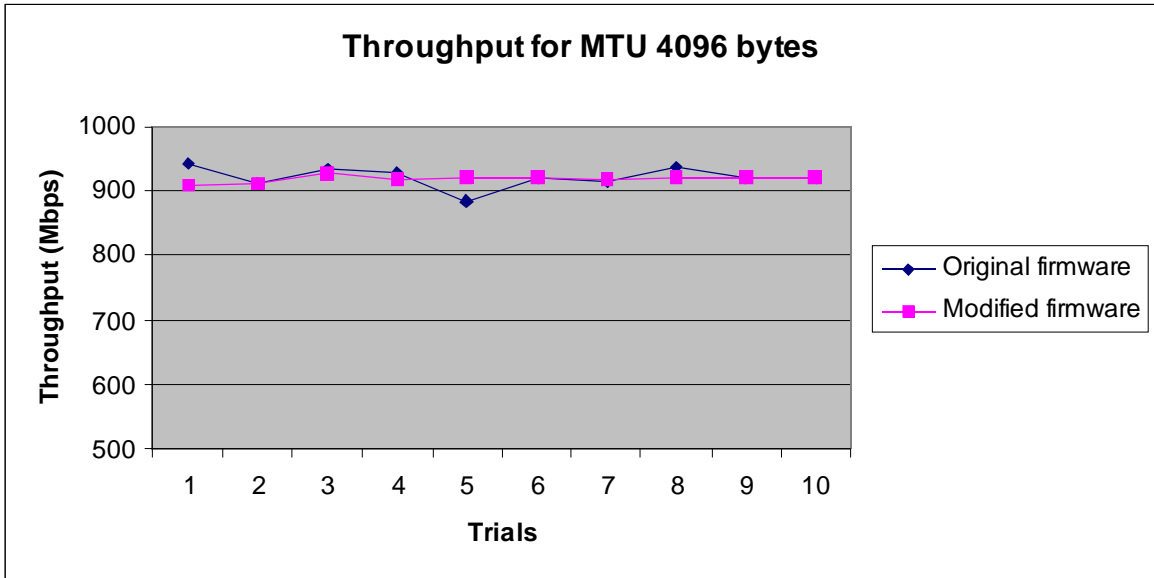


Fig 7: A comparison of throughputs with the two firmwares for MTU 4096 bytes

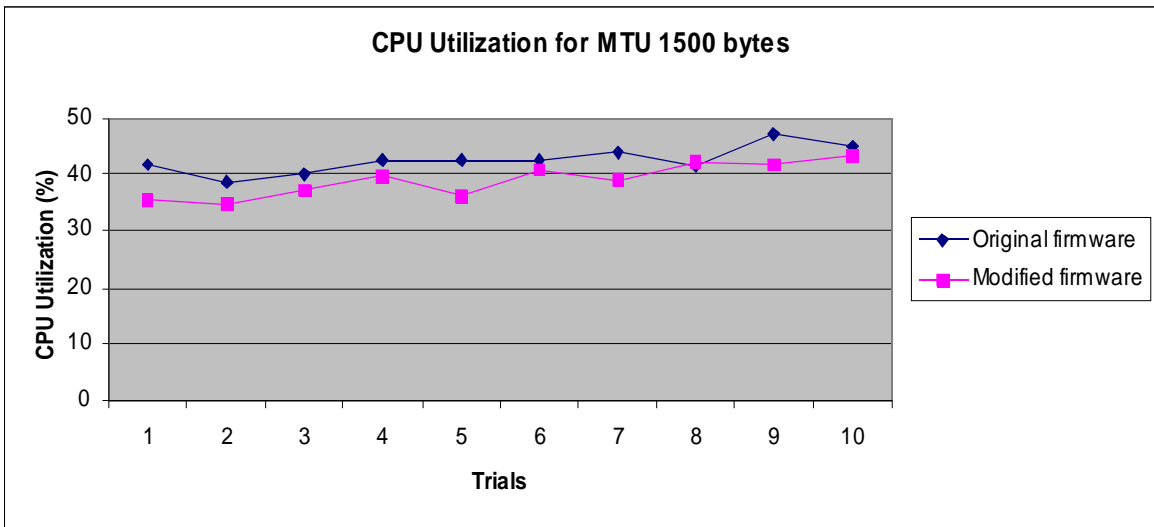


Fig 8: A comparison of CPU Utilization with the two firmwares for MTU 1500 bytes

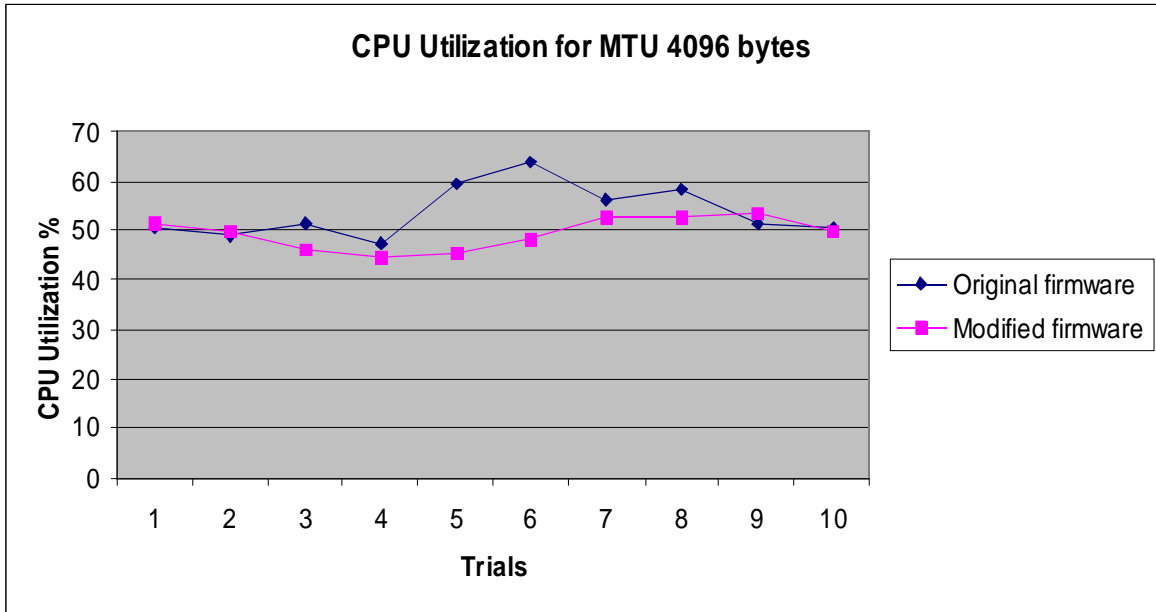


Fig 9: A comparison of CPU Utilization with the two firmwares for MTU 4096 bytes

Conclusions

The speculative defragmentation of the UDP packets did not provide us with any kind of an improvement in the network throughput but it was noticed that the CPU utilization decreased. This is due to the reduced number of interrupts to the kernel. Hence we have been able to allocate more processor cycles to the applications without affecting the network throughput. One very important enhancement to this project would be the implementation of a zero copy mechanism where we could DMA the data from the NIC directly to the application that is expecting the data.

References

- 1.The Scalable Systems Lab website – <http://www.cs.unm.edu/~ssl/>
- 2.Tigon/PCI Ethernet Controller Documentation.
- 3.TCP/IP Illustrated . Vol 1 W. Richard Stevens . Addison – Wesley
- 4.*Speculative Defragmentation – A technique to improve the Communication Software Efficiency for Gigabit Ethernet* – Christian Kurmann , Michael Muller , Felix Rauch and Thomas M Stricker at the Laboratory for Computer Systems, Swiss Federal Institute of Technology(ETH).
- 5.Homepage of Pete Wyckoff – <http://www.osc.edu/~pw/>
- 6.Linux Device Drivers, 2nd Edition By Alessandro Rubini ,Jonathan Corbet. O'Reilly & Associates, Inc.

Acknowledgements

We would like to thank Prof. Barney Maccabe for guiding us through the project and Wenbin Zhu for providing the firmware source files and tips and guidelines everytime we needed it.