PickPacket: A Distributed Parallel Architecture

Dungara Ram Choudhary
Email: dungara@cse.iitk.ac.in
Guide: Dr. Dheeraj Sanghi
Email: dheeraj@cse.iitk.ac.in

Department of Computer Science & Engineering
Indian Institute of Technology
Kanpur, UP, INDIA - 208016

***Abstract*— Use of computers and networks in information exchange has increased in the last few decades and led to establishment of high speed networks (up to 10 Gbps). These network speeds are approaching the memory interface speeds of general purpose processors. Monitoring networks with such high speed is not possible with today's general purpose processors. To solve this problem we propose a distributed parallel architecture for PickPacket[1], a network monitoring tool. We use network processor to split the traffic and then process that using general purpose multicore processor. We try to achieve these goals while preserving the simplicity of current architecture of PickPacket. We extend the "PickPacket Packet Filter" component of PickPacket to support parallelization. Testing of "Gigabit PickPacket" was also a challenging task.**

***Index Terms*— Computer Networks, Network Monitoring.**

I. Introduction

There has been a tremendous growth in the amount of information being transferred between computers with the advent of Internet. Many times this data contains sensitive information in which governments or law enforcement agencies might be interested. It is felt that careful and judicious monitoring of data flowing across the net can help to detect and prevent crime. Such monitoring tools, therefore, can have an important role in helping agencies gather information against terrorism, child pornography/exploitation, espionage, information warfare and fraud. Companies that want to safeguard their recent developments and research from falling into the hand of their competitors also resort to intelligence gathering. Thus there is a pressing need to monitor, detect and analyze undesirable network traffic [1].

Neeraj Kapoor [2] describes design of the network monitoring tool called "PickPacket". PickPacket does context sensitive filtering and can search for specified patterns in network traffic.

Srikanth describes Gigabit PickPacket [3], which provides a distributed architecture for PickPacket but it does not distribute the kernel level overhead of packet processing. The design of splitter provided in this paper is also PC based splitter which will not be able to handle today's high speed gigabit traffic. Iannaccone *et al* [4] provide a prototype of a tool for passive monitoring of gigabit links. In their prototype there is no online processing of data and it is dumped on disks that can be analyzed furthermore.

To fulfill increased data transfer requirement the underlying hardware technology has also evolved rapidly and gigabit networks are become reality. To maintain and monitor these networks is a challenging task. The use of a general-purpose workstation as a traffic monitor [1], [5] may not achieve sufficient performance while purpose-specific ASICs may not be flexible enough. Use of network processors provides flexibility for modification while giving high packet throughput and low packet latency. Network processors meet network performance and flexibility requirements through highly parallel, programmable architecture. Our design of splitter uses network processor to process traffic.

Srikanth [3] describes design of multithreaded version of PickPacket. We have extended this work to support four new application level protocols: Yahoo mail, IMAP, IRC, POP. We also did correctness and performance testing for multithreaded version.

II. Background and Problem Statement

A. Design of PickPacket

The sequential design of PickPacket (see Figure 1) has four components - "PickPacket Configuration File Generator" for assisting the user in setting up the parameters for capturing packets, the "PickPacket Packet Filter" for capturing packets, the "PickPacket Post-Processor" for analyzing packets, and the "PickPacket Data Viewer" for showing the captured data to the user. Most of the on-line processing is done in the component "PickPacket Packet Filter". We provide an architecture that exploits this fact and improves performance of the system.

B. Limitations of Sequential Design

The resources of single computer are limited and will not be able to handle gigabit traffic due to following reasons:

1) Processing Power is limited: Pattern matching is a computation intensive process so system will not be able to handle the data at gigabit speed.
2) Memory is limited: PickPacket keeps state information of each connection on the network as well as keeps a queue of packets to keep history of connection. Instantaneous number of active connection on a gigabit link can be of the order of hundreds of thousand. The

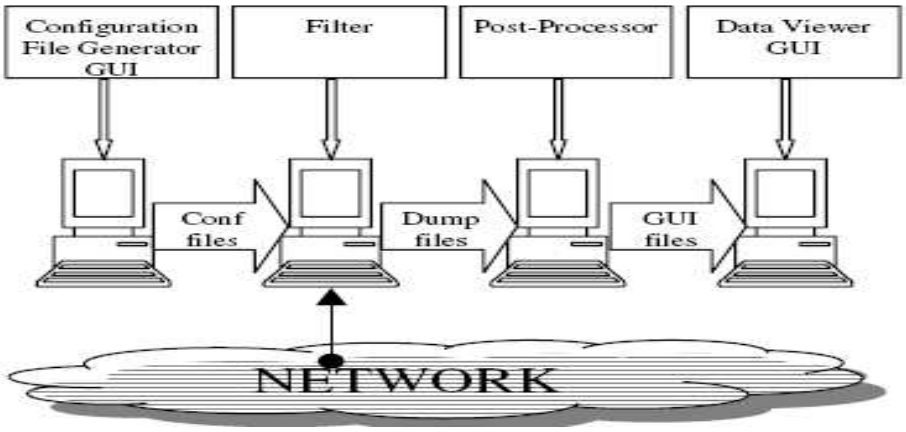

Fig. 1. Sequential Architecture for PickPacket (source [1])

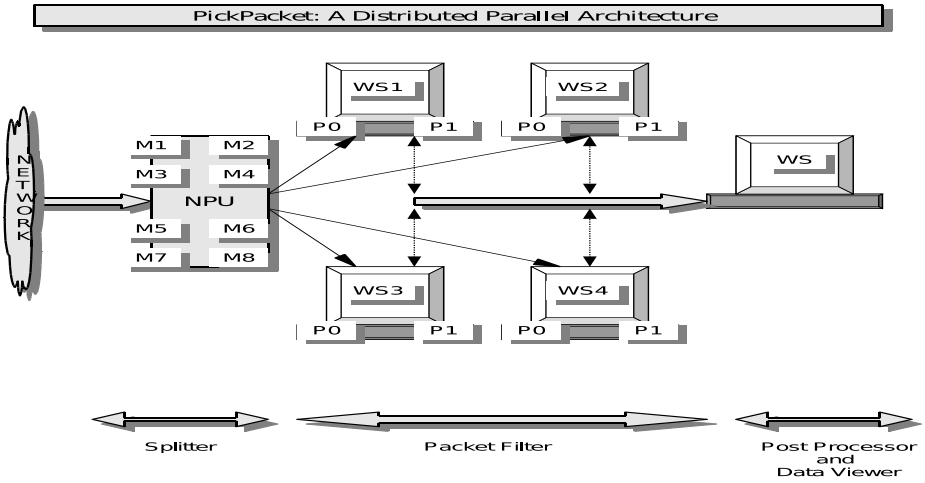

Fig. 2. Distributed Parallel Architecture

system will not be able to allocate memory for these many connections.

3) PCI bus speed is limited: Normally multiple devices share same PCI bus to transfer data to and from memory. This will become a bottleneck when multiple devices are used simultaneously.

Apart from above described constraints there may be other bottlenecks like disk throughput, bus and memory speed etc.

The above limitations put an upper bound on the traffic that sequential version of PickPacket can handle. To overcome above limitations we designed distributed version of PickPacket which can handle up to 700Mbps of traffic while the multithreaded version can handle up to 450Mbps of traffic. The proposed system will be running multithreaded version of PickPacket on each target machine while traffic between target machines is distributed using hardware based splitter. Hence, we call this architecture as "Distributed Parallel Architecture".

III. Design of the Distributed Parallel Architecture

The main thrust of the project was to improve the performance of the PickPacket by using distributed and parallel computing while preserving simplicity of system. We have implemented splitter on network processor to get performance gain while rest of the processing is done using multiprocessor workstations to avoid complexity in programming. An overview of the distributed parallel architecture is shown in Figure 2. Design of the "PickPacket configuration file generator" remains same. After generation of configuration file it is transferred to other workstations. System connects with network through a network packet processing board. This board uses network processing unit for packet processing. This acts as splitter and traffic is sent to different workstations. These workstations have multiple CPUs and filtering of incoming data will be performed here. The dump files generated after filtering are sent to workstations responsible for post-processing. Processed data is assembled and shown to user using webbased GUI.

We successfully designed, implemented and tested dis-

tributed parallel version of PickPacket. The major challenges faced are designing splitter, understanding the architecture of NPU (IXP2400) and development platform, designing and implementing multithreaded version for "Packet Filter" module.

A. Design of the Splitter

Design of splitter is shown in Figure 3. The setup shown in figure has two layer 2 switches and one network packet processing board. Switch 1 is connected to network and mirrors ingress traffic of each port to a specified port, which forwards it to network packet processing board. Network packet processing board receives it on port 0. The traffic received at port 0 is classified using five tuple namely $\prec$ `source IP address, destination IP address, layer 4 protocol type(TCP or UDP), source port, destination port` $\succ$. For the classification purpose we add all five values and then use a hash function (see Procedure 1) which generates key in the range (0, number of targets - 1). FTP protocol is handled as special case because it uses separate command and data connections. We need to send packets of both connections to same target. To achieve this we use three tuple $\prec$ `srcIp, dstIp, transport level protocol` $\succ$ for calculating hash function for FTP connections.

Procedure 1 Calculation of Hash Index

```
1: tuple_sum ⇐ srcIP + destIP + Protocol
2: if (srcPort = 20 ∨ srcPort = 21 ∨ dstPort = 20 ∨ dstPort
   = 21) ∧ Protocol = TCP then
3:         no op
4: else
5:         tuple_sum ⇐ tuple_sum + srcPort + dstPort
6: end if
7: tuple_sum ⇐ tuple_sum mod16
8: key ⇐ h(tuple_sum)
9: key ⇐ tuple_sum mod number_of_targets
```

Motivation behind choosing this hash function is that we analyzed traffic of IIT Kanpur network for different hash functions and results were almost similar in each case. We preferred this hash function due to its simplicity. Further if due to change in traffic pattern this hash function starts performing inferior then we may change it. As the microengine processor (RISC architecture) does not support division, multiplication and module operations so module is performed using repetitive subtraction.

Based on this key layer 2 header of packet is changed. The destination MAC address is set to MAC address of target machine and source MAC is set to MAC of the transmitting port. All valid traffic is forwarded to network packet processing board's port 1 while erroneous packets are sent to its port 2. The port 1 of network packet processing board is connected to switch 2. Switch 2 is configured for layer 2 static forwarding and splits traffic to target machines.

B. Design of Multithreaded Version

The design of multithread version is described by Srikanth [3]. Tasks need to be performed in PickPacket filter include copying packet from network card to kernel buffer and send it to socket, basic filtering at socket level and application level filtering. The first two tasks are performed by kernel while the last one is accomplished by a user level program (called "pickpacket filter"). Srikanth [3] describes a multi-threaded design of this program. We extended "pickpacket filter" to support four new protocols:Yahoo mail, IMAP, IRC, POP.

In multithreaded design, a fixed number of threads are created and different connections are distributed among them in the desired ratio. All packets belonging to a particular connection will always be handled by same thread. We use a hash function on four tuple $\prec$ `source IP address, destination IP address, source Port, destination Port` $\succ$ to distribute packets among multiple threads in the desired manner. There are two categories of threads on system:*Reading Threads* and *Processing Threads*. *Reading Thread* is used for reading packets from socket buffer and handles it or enqueues it for other threads based on value of hash. On the other hand, *Processing Thread* do not read packets from socket and processes packets from its *pending queue*. Each *Processing Thread* has a buffer called *pending queue* associated with it. Apart from processing done by above threads, kernel reads packets from network card and does basic filtering and enqueues them in socket buffer. If system is congested then kernel will drop the packet instead of sending it to socket buffer. To avoid this, some processors should be kept less congested; hence no threads should be scheduled on them. Procedure 2 describe the packet handling by *Reading Thread* and procedure 3 describe the packet handling by *Processing Thread*.

IV. Implementation of Gigabit PickPacket

ENP-2611, a PCI board is used as network packet processing board for the implementation of splitter. This board uses IXP2400 as network processing unit. IXP2400 have eight microengines (processors) for packet processing and each microengine can support up to eight threads. These threads have zero context switch overhead. We configured port 0 in receive only mode while port 1 and port 2 in transmit only mode. We use one microengine for reading packets from port 0, one microengine for processing the packets received and one for transmission of packets.

The port 1 of ENP-2611 is connected to a DLink DGS-3312SR switch configured for layer-2 static forwarding. This switch is connected to target machines which can be up to 16.

To provide a interface to user for configuring target machines for splitter, a program called "spltconf" is implemented The program can also be used for load balancing at macro level by using same target more than once. The program dynamically generates "microcode" based on target machines configuration. It then compiles the program and uploads to ENP-2611. All these steps are transparent to user. A XML file "splitter.xml" is used to save configuration of target machines.

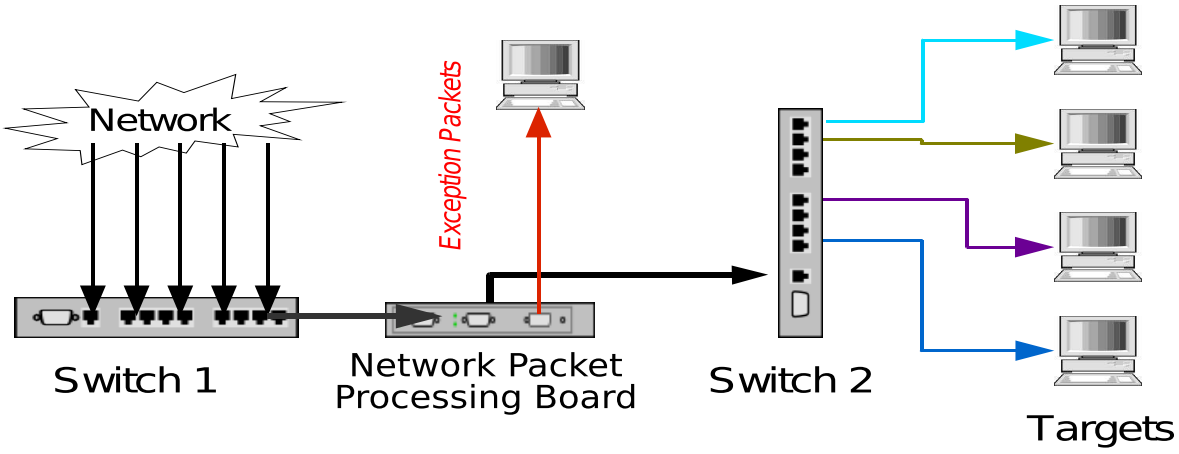

Fig. 3. Architecture of Hardware Splitter

Procedure 2 Packet Handling by Reading Thread

```
1: while true do
2:   wasEmpty = false;
3:   pkt ⇐ read a packet from socket
4:   dmplx ⇐ search dynamic demultiplexer table for entry
5:   if dmplx = empty then
6:     key ⇐ hash(packet)
7:     dmplx ⇐ search static demultiplexer table for key
8:   end if
9:   if dmplx ≠ empty then
10:    if pending queue of dmplx.thread is empty then
11:      wasEmpty = true
12:    end if
13:    enqueue packet to pending queue of dmplx.thread
14:    if wasEmpty then
15:      send signal to dmplx.thread
16:    end if
17:  end if
18:  while pending queue is not empty do
19:    pkt ⇐ read packet from pending queue
20:    process(pkt)
21:  end while
22: end while
```

Procedure 3 Packet Handling by Processing Thread

```
1: while true do
2:   if pending queue is empty then
3:     wait(signal)
4:   else
5:     pkt ⇐ read packet from pending queue
6:     process(pkt)
7:   end if
8: end while
```

To merge all dump files generated at targets machines we implemented a utility called "dumpcat" which concatenates all dumps in one dump. It also provides support for checking integrity of dump files.

For multithreaded version of "pickpacket filter", we extended work done by Srikanth [3] to support four new protocols:Yahoo mail, IMAP, IRC, POP.

V. Testing of Gigabit PickPacket

Testing of "Gigabit PickPacket" is performed in two phases. In first phase we tested correctness of system while second phase consist performance testing. Correctness testing is performed in controlled environment by varying different system parameters while during performance testing we changed traffic speeds.

A. Generation of Gigabit Traffic

Due to unavailability of Gigabit Network we designed a setup which generates traffic at Gigabit speed such that the traffic pattern has resemblance with real world traffic pattern. We assume that the link being monitored is serving many users which are connected through low speed links (i.e. less then 100 Mbps) which is generally true in real world. This assumption is necessary because we are splitting traffic based on transport layer connection. Network traffic on a typical Gigabit link (at ISP's) have the property that there are hundreds of thousand of connections at a particular instant of time while traffic due to single connection is of order of Mbps. To generate the traffic at high speed we captured packets (in "dump file") from a live link of speed 34 Mbps. We created 5 dump files by capturing packets at different times. These files are merged to generate a larger dump by reading packets in round robin fashion from each dump file. The resultant file is again split in parts equal to replay agents. The replay agents varies in there replay capabilities so the split is performed in ratio of replay speed of agent. These files are transferred to respective replay agent. Each replay agent is connected to a Gigabit switch which is configured to mirror all incoming traffic to port 1. We used traffic from this port as input to our system.

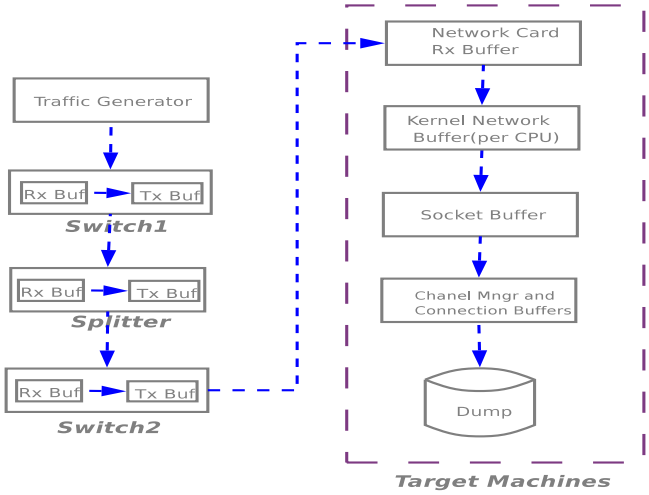

Fig. 4. Path of Packet in Distributed System showing Buffers

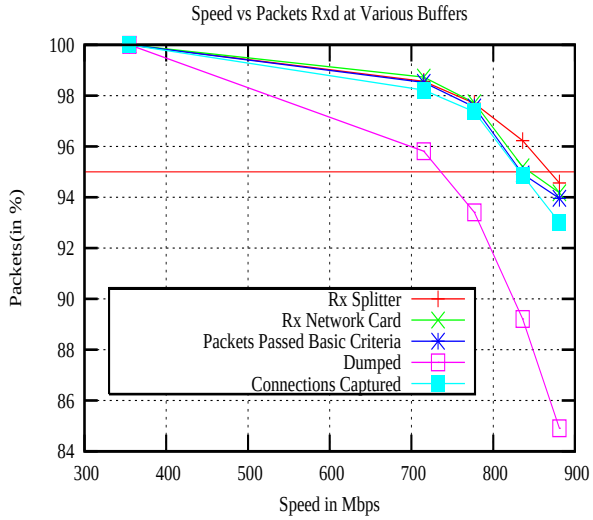

Fig. 5. Results of Testing of Splitter

B. *Testing of Splitter*

Correctness testing of splitter is performed in a controlled environment using a single machine as replay agent. We used various combinations of target machines and results were confirmed with expected results.

Performance testing is done using the gigabit network setup described in previous section. We used 9 replay agents to generate the traffic at controlled speeds. A total of 15782100 packets are sent, having average packet size of 357 bytes. The packets are sent at different speeds varying from 300Mbps to 900Mbps.

Figure 4 shows the path of the packet in the system. This shows various buffers used in the path. Packet drop in the system will occurs only at these buffers. We put packet counters at these buffers which report packets received by the buffer or transmitted by the buffer.

The graph shown in Figure 5 shows results obtained during performance testing. The results of the packets received/transmitted at particular buffer, are reported as percentage of expected values. Expected values are obtained by replaying the traffic at lower speed. The graph also reports the statistics for dumped packets and number of connections captured.

The packet drop at Rx buffer of splitter is mainly due to drop at switch 1(see Figure 3). The switch is unable to handle traffic beyond 750 Mbps and packet drop increases rapidly. There is no packet loss observed at Rx buffer of splitter as well as between Rx and Tx buffers. The network processor handled all the packets successfully without any drop.

The another important observation is that number of packets dropped at buffer as well as number of connections captured follows same trend as drop at switch 1. In opposite the number of packets dumped decreases rapidly. This is because sometimes loss of even one packet may result in loss of entire connection.

The pickpacket successfully handled 700Mbps traffic while dumping around 96% packets of the expected value. Although this is not the upper bound for the performance of the splitter because there is a major contribution in this drop is from traffic dropped at switch 1. In real world there will be no drop at switch 1(due to retransmission).

We performed another benchmark in which we tested performance of hardware splitter only. We connected port 0 of ENP-2611 directly to a computer having Gigabit network interface. The machine was running GNU/Linux with kernel version 2.6.9. We used pktgen module of kernel to construct the packet in memory and sent it at high speed in controlled manner. We used two target machines. This time we were not running PickPacket on target machines but counted number of packets received on network interface. The traffic generated such that after hashing packets was forwarded to target machines in round robin fashion. The result of this benchmark is shown in table I. We are unable to replay dump files at these speeds using single replay agent due to disk throughput limits. Hence these results could not be verified on real world traffic.

TABLE I

PERFORMANCE OF SPLITTER

Packet Size(bytes)	Speed(Mbps)	Pkts Rxd(%)	
		Splitter	Targets
357	942	100	100
1500	985	100	100

It is important to note that processing cost at NPU is independent of the layer-4 protocol as well as of number of targets used. Hence we can expect same amount of performance from splitter given adequate quality switches and sufficient target machines.

We compared the packets received by each target machine with the ideal case results(i.e. if packets are divided equally

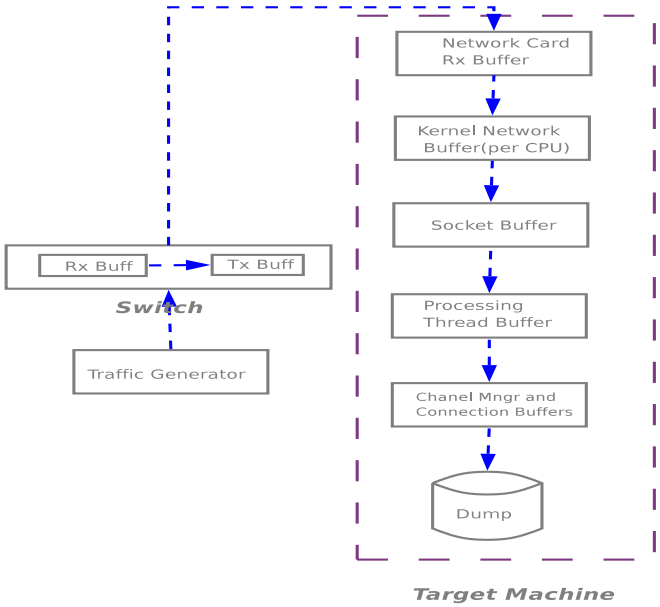

Fig. 6. Path of Packet in Multithreaded System showing Buffers

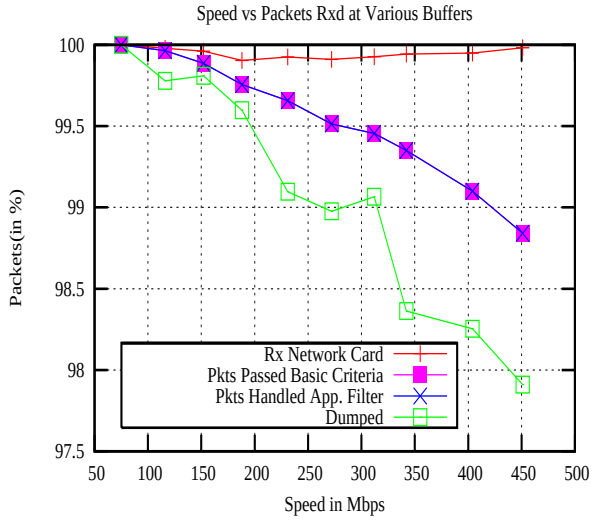

Fig. 7. Results of Testing of Multithreaded PickPacket

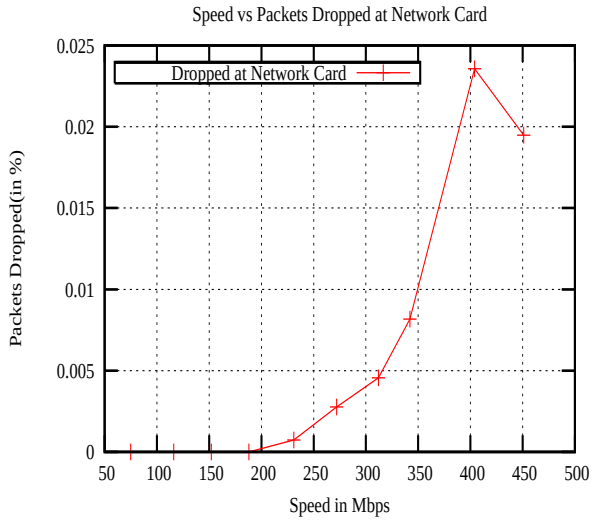

Fig. 8. Speed vs Drop of Packets at Network Interface

in target machines). The hash function used by us performed well in every test and a maximum of 20% deviation is found from best possible outcome.

C. Testing of Multithreaded Design

The correctness testing of the multithreaded version is performed on dual core Xeon machine with 2.0GHz of CPU and 1.0 GB RAM and running Linux kernel 2.6. We used various combinations of threads with different load and compared the results with expected results.

To test performance of multiprocessor version we generated traffic up to 500 Mbps using approach described in previous section. We used two replay agents, which are connected to a DGS-3308TG switch. DGS-3208TG is mirroring all incoming packets to port 0. Port 0 of the switch is connected to network interface of computer running pickpacket. This computer has four "AMD Opteron Processor 850", 16 GB of memory and SCSI disks for secondary storage.

The path taken by the packet in the system and different buffer are shown in Figure 6. The result of the testing is shown in Figure 7. Figure 8 shows relation between packet drop at network interface of target with speed of the traffic.

PickPacket successfully dumped 98% packet of the expected value at 450 Mbps. Increase in speed apart 450Mbps will cause huge drop at network card of target machine.

VI. Contribution of the Work

We successfully designed and implemented distributed version of Gigabit PickPacket which is able to handle 700Mbps of traffic. We also extended multithreaded version of PickPacket to support four new protocols. The multiprocessor version is able to handle 450Mbps of speed.

In the first phase of the project a hardware splitter is developed. We created a list of problems faced during the project and documented their possible solutions. A interface for configuring number of target machines and their MAC address was also developed. In the second phase of project we implemented multithreaded version of pickpacket and performed performance testing.

VII. Conclusion and Future Work

We have developed Gigabit PickPacket which can handle traffic up to 700Mbps. We used hardware splitter which can handle up to 942Mbps of traffic and can split the traffic in target machines. The splitter uses five tuple namely $\prec$ `source IP address, destination IP address, layer 4 protocol type(TCP or UDP), source port, destination port` $\succ$ for classification purpose. The system provides option for specifying up to

16 target machines and it can be changed easily. We have already highlighted the hash function used for classification. We have documented all problems faced during development. The document will serve as reference for future work on NPU boards.

We also extended multiprocessor version of pickpacket to support four new protocols:Yahoo mail, IMAP, IRC, POP. The system is able to handle traffic up to 450Mbps. In the parallel execution too, one particular stream of packets is being processed by same thread after demultiplexing.

In future basic packet filtering can be implemented on network packet processing board. This will minimize the traffic on the target machines at least to half.

Other major addition can be done by porting current multithreaded version to 64-bit architecture. The current multithreaded version was developed for 32-bit architecture. The system can not allocate more than 4GB memory for the process and hence can handle a limited number of connections at an instance of time. Given the nature of Gigabit traffic, the multithreaded version will never be able to handle traffics of order of Gigabit. Porting 64-bit architecture will help in supporting more number of connections.

VIII. ACKNOWLEDGMENT

I would like to express my heartfelt gratitude to Prof. Dheeraj Sanghi without whose guidance, inspiration and constant motivation, this project would not have been possible. I would also like to thank the other team members involved in development of PickPacket for their cooperation and support.

REFERENCES

[1] B. Pande, D. Gupta, D. Sanghi, and S. K. Jain, "The Network Monitoring Tool - PickPacket," in *ICITA '05: Proceedings of the Third International Conference on Information Technology and Applications (ICITA'05) Volume 2*. Washington, DC, USA: IEEE Computer Society, 2005, pp. 191–196.

[2] N. Kapoor, "Design and Implementation of a Network Monitoring Tool," Master's thesis, Department of Computer Science & Engineering, IIT Kanpur, April 2002.

[3] P. S. Srikanth, "Gigabit PickPacket: A Network Monitoring Tool for Gigabit Networks," Master's thesis, Department of Computer Science & Engineering, IIT Kanpur, May 2004.

[4] G. Iannaccone, C. Diot, I. Graham, and N. McKeown, "Monitoring very high speed links," in *IMW '01: Proceedings of the 1st ACM SIGCOMM Workshop on Internet Measurement*. New York, NY, USA: ACM Press, 2001, pp. 267–271.

[5] J. Apisdorf, K. Claffy, K. Thompson, and R. Wilder, "OC3MON: Flexible, Affordable, High Performance Staistics Collection," in *LISA '96: Proceedings of the 10th USENIX conference on System administration*. Berkeley, CA, USA: USENIX Association, 1996, pp. 97–112. [Online]. Available: http://www.usenix.org/publications/library/proceedings/lisa96/full_papers/kc/kc.ps

[6] K. Anagnostakis and H. Bos, "IXPMON: An Efficient and Extensible Packet Monitoring System," ongoing, University of Pennsylvania, USA. [Online]. Available: http://www.cis.upenn.edu/~anagnost/ixpmon

[7] E. J. Johnson and A. R. Kunze, *IXP2400/2800 Programming: The Complete Microengine Coding Guide*. USA: Intel Press, April 2003.

[8] I. Corp., "Intel IXP2400 Network Processor." [Online]. Available: http://www.intel.com/design/network/products/npfamily/ixp2400.htm

[9] K. K. Niraj Shah, William Plishker, "Comparing network processor programming environments: A case study," in *2004 Workshop on Productivity and Performance in High-End Computing (P-PHEC)*, February 2004.

[10] Intel Corp., *Programmers Reference Manual*, November 2003.