Addition of Virtual Interfaces in NetFlow Probe for the NetFPGA

Muhammad Shahbaz
Center for Advanced Studies in Engineering
Islamabad, Pakistan
mshahbaz@case.edu.pk

Zaheer Ahmed
Center for Advanced Studies in Engineering
Islamabad, Pakistan
zaheer@case.edu.pk

Habibullah Jamal
University of Engineering and Technology Taxila
Pakistan
drhjamal@uettaxila.edu.pk

Asrar Ashraf
Center for Advanced Studies in Engineering
Islamabad, Pakistan
asrar@case.edu.pk

Nadeem Yousuf
Center for Advanced Studies in Engineering
Islamabad, Pakistan
nydhami@case.edu.pk

Raania Naeem Khan
Center for Advanced Studies in Engineering
Islamabad, Pakistan
raanianaeem@case.edu.pk

ABSTRACT

In this paper, we present an extended architecture of protocol extraction layer for standalone NetFlow probes. Standalone NetFlow probe over multi-gig data rates is a new paradigm for researchers. The routing of private enterprise traffic over public network requires encapsulating the private IP packet in GRE, L2TP or MPLS. The bandwidth monitoring, network forensics and network traffic analysis is of paramount importance in order to maintain the quality of service (QoS) and security of enterprise networks. The presented architecture targets all the present and future IPv4 and IPv6 network protocols. In this paper, we focus on the protocol extraction mechanisms for technologies providing support for virtual interfaces. We implemented the protocol extraction layer in the NetFlow probe architecture for NetFPGA. The architecture can also be utilized for deep packet inspection (DPI), especially to monitor RTP payloads in the Voice over IP (VoIP) applications. The architecture finds applications reaching high data rates; like Internet Protocol monitoring, VoIP monitoring and QoS monitoring.

General Terms

Design, Management and Performance

Keywords

Computer Networks, Protocol Analysis, NetFPGA, NetFlow

1. INTRODUCTION

In today’s competitive market, advanced data services are migrating from fixed to mobile infrastructure and WAN connectivity is becoming mandatory to extend the enterprise network to mobile infrastructure. The latest trend of businesses is the establishment of many branch offices miles away from the main head office that is vital for providing valuable 24/7 customer support and services across the globe. The integration of WAN and LAN technologies involve multitude of protocols. In order to provide ubiquitous connectivity for bandwidth hungry, hybrid and critical applications, the requirement of network protocol analysis has become all the more important.

NetFlow is a network analysis protocol designed by Cisco Systems [1]. The architectures adopted for the network flow analysis are either router-based or probe-based as shown in Figure 1 and Figure 2. NetFlow on routers provides a Network-wide view of the traffic, for network management and planning as well as traffic analysis and observation. However, NetFlow on probes is well suited for the observation of critical links [2].

In packet switching networks; traffic flow, packet flow or network flow consists of a sequence of packets sent from a source computer to a destination which could be another host, multicast group or broadcast domain. Network flows have been defined in numerous ways.

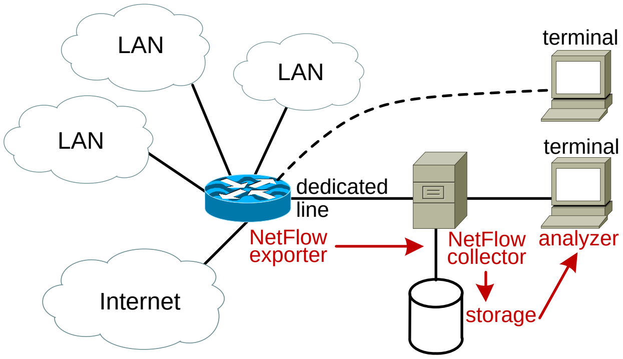


Figure 1. Standard NetFlow Architecture

Cisco defines network flows using a 7-tuple key, as a unidirectional sequence of packets sharing the following 7 values: (1) Source IP address, (2) Destination IP address, (3) Source port for UDP or TCP, 0 for other protocols, (4) Destination port for UDP or TCP, type code for ICMP, or 0 for other protocols, (5) IP protocol, (6) Ingress interface (SNMP if Index), and (7) IP Type of Service (ToS) [1][2].

We present generic protocol analysis architecture to target existing and future internet protocols. In this paper, our point of focus is to provide NetFlow support on NetFPGA for virtual interfaces. Virtual interfaces are usually found in technologies like

Layer 2 Tunneling Protocol (L2TP), Generic Routing Encapsulation (GRE) tunnels and Multiprotocol Label Switching over Virtual Private Network (MPLS-VPN) [3]. The rest of the paper is organized as follows: section 2 describes NetFPGA, section 3 explains the system architecture, section 4 presents resource utilization, and section 5 concludes the paper.

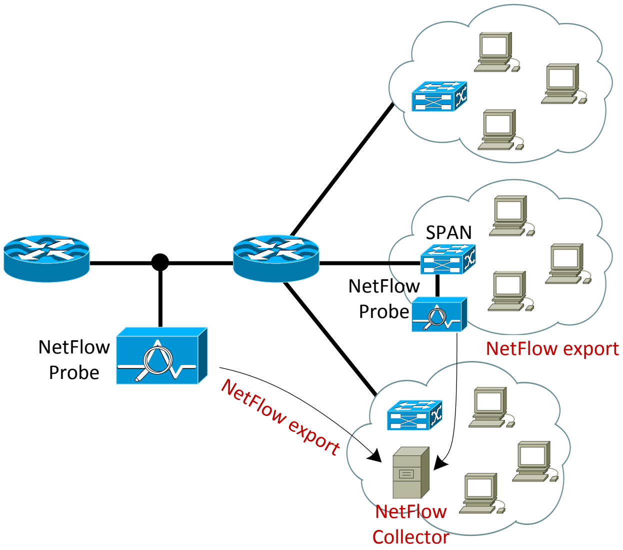


Figure 2. Stand-alone NetFlow Architecture

2. NetFPGA

The NetFPGA is a research platform comprising of three parts, hardware, gateware and software. The hardware is based on a PCI plug-in card equipped with Xilinx Virtex-II Pro 50, four ports of Gigabit Ethernet, two parallel banks of 18Mbit Zero-bus turnaround (ZBT) SRAM and 64 Mbytes DDR DRAM for local processing. The FPGA directly handles all data-path switching, routing, and processing of Ethernet and Internet packets, leaving the software to handle only control-path functions [4]. Gateware is a set of Verilog source files having an integrated pipeline with several components. Gateware is designed to be modular and reconfigurable. Most designs invoke the use of reference pipeline [5]. The reference pipeline shown in Figure 3, is comprised of eight receive queues; eight transmit queues, user data path, and a register system i.e. Register I/O module. Receive and transmit queues are divided into two types: MAC and CPU. Users add and connect their modules to the User Data Path. The Input Arbiter and the Output Queue modules, present in the User Data Path, are the main modules present in almost all NetFPGA designs. Source code for these modules is provided in the NetFPGA Verilog library. The Input Arbiter services the eight input queues in a round robin fashion to feed a wide (64-bit) packet pipeline [6].

Gateware communicates with the software using register system. The register system allows modules to be inserted within the pipeline using minimal effort. The register interface allows software programs running in the host system to send and receive data from hardware modules using PCI device driver utilities provided with the NetFPGA reference designs [7] [8].

3. SYSTEM ARCHITECTURE

As mentioned in [9], NetFlow probe consists of a host computer and a NetFPGA network accelerator card. By exploiting the hardware-software co-design principle, timing critical functions like measurements etc are implemented on the NetFPGA whereas soft constraint functions like control, configuration and collecting processes are implemented within the host computer (Linux OS) as a user software.

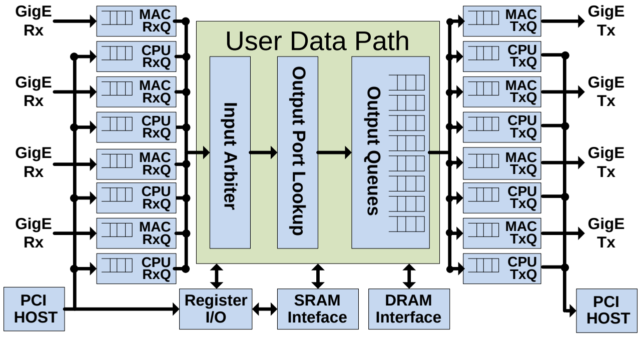


Figure 3. NetFPGA Reference Pipeline

Two new features have been added along with the existing four parameters of the probe mentioned in [9]. These are:

- Packet processing of virtual interfaces consisting of Layer 2 Tunneling Protocol (L2TP), Generic Routing Encapsulation (GRE), and Multiprotocol Label Switching (MPLS).
- Complete software support for virtual interface analysis and processing.

3.1 Hardware Implementation

Architecture of NetFlow v5 Probe [9] utilizes the NetFPGA reference pipeline as a foundation. Main features of the NetFlow v5 Probe are incorporated within the User Data Path; design outside the User Data Path has been used without any modifications.

The highlighted regions in Figure 4 show the newly added and updated blocks. User Data Path has been divided into 4 major blocks: (1) Input Arbiter, (2) Time Stamp, (3) Output Port LookUp, and (4) Output Queues. The communication between these blocks is provided using the NetFPGA interconnection protocol i.e. data bus, control bus, write and ready signals. The Output Port LookUp is further subdivided into Multi-Layer Protocol Extraction, Hash Generation, Flow Lookup, Flow Processing and Output Port LookUp Processor. Each of these blocks has been assigned a specified task. After processing, each block pushes required information in the corresponding Result FIFO (RFIFO), which is then read by the next block in a pipelined fashion. The interconnection of blocks within the Output Port LookUp is provided by the use of simple dual port distributed FIFOs.

3.1.1 Interface Receive/Transmit Queues

The NetFlow design has x4 Ethernet Queues and x4 DMA Queues each for data reception and transmission, shown in Figure 4 [9]. Packet received at an interface is stored in the receive queue (RxQ). During an available time-slot for a particular interface, Input Arbiter reads the packet from the RxQ and transmits the

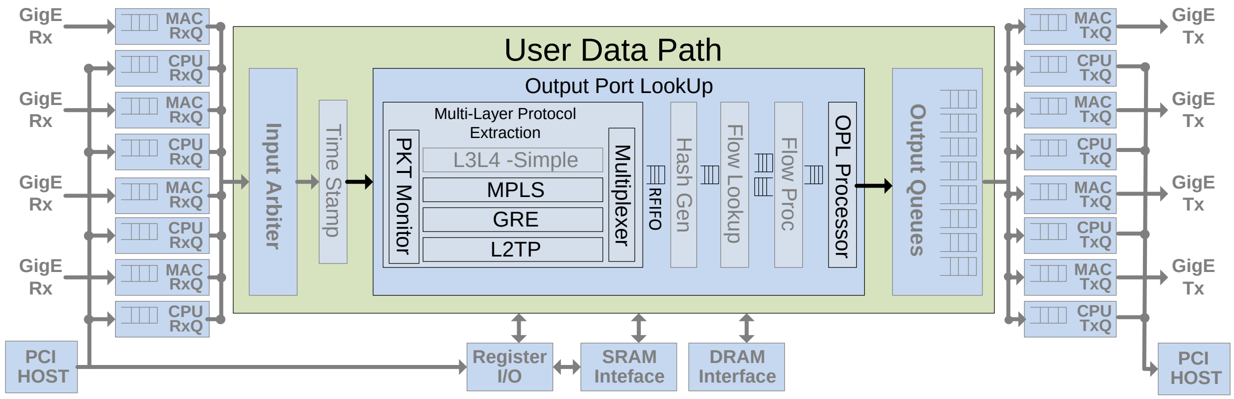
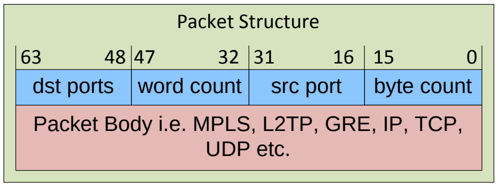


Figure 4. NetFlow Architecture for NetFPGA

complete packet to the Time Stamp block before starting a new transaction for another interface. During the reading process, RxQ encapsulates the packet by attaching a 64-bit word at the start of the packet as shown in the Figure 5. The header contains information regarding the size of the packet (bytes and words) as well as the interface at which the packet arrived. An empty location present in each header for the destination ports address is filled by Output Port Lookup Processor (OPL) block.



Packet Structure

63 48	47 32	31 16	15 0
dst ports	word count	src port	byte count
Packet Body i.e. MPLS, L2TP, GRE, IP, TCP, UDP etc.			

Figure 5. Internal Packet Format

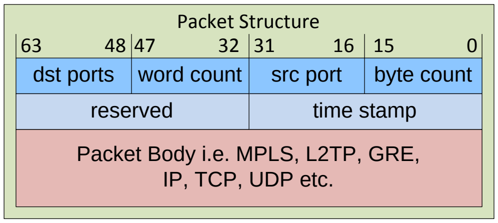
Output Queue (OQ) stores the packet within the transmit queue (TxQ) for a given interface. The TxQ removes the header, attached at the time of the arrival of the packet at the receive interface, and sends it out to the respective transmit interface(s).

3.1.2 Input Arbiter Block

The Input Arbiter block is used from the original NetFPGA library: *reference NIC design*. Within the User Data Path as shown in Figure 4, the first module a packet encounters is the Input Arbiter, who, decides which RxQ to service. It subsequently pulls a packet from that RxQ and hands it over to the Time Stamp block that transmits it to the main processing pipeline.

3.1.3 Time Stamp Block

Functionality of the Time Stamp block remains the same as provided in the NetFlow reference design [9] [11]. The only alteration made to the design is the placement of the Time Stamp module directly after the Input Arbiter (see Figure 4) instead of after the Protocol Extraction block. This has been done to make the timestamp unit independent of the Output Port LookUp block. Timestamp unit inserts the current timestamp value from the timestamp counter with millisecond resolution into the packet header as shown in Figure 6. The remaining functionality of the Timestamp's block remains unchanged from the reference design [11]. After inserting a header for 32-bit timestamp value within the packet structure, it transmits the newly encapsulated packet to the Multi-Layer Protocol Extraction block.



Packet Structure

63 48	47 32	31 16	15 0
dst ports	word count	src port	byte count
reserved		time stamp	
Packet Body i.e. MPLS, L2TP, GRE, IP, TCP, UDP etc.			

Figure 6. Packet Structure with Timestamp field

3.1.4 Multi-Layer Protocol Extraction Block

Multi-Layer Protocol Extraction block is the core of the overall NetFlow probe architecture. The extraction of exact and accurate flow records from different combination of layers is a crucial task. For simplicity we only focus on the virtual interfaces from within logical interfaces for flow analysis [3]. In our case a network flow or packet record or flow record is composed of following 6-tuple keys:

- Source IP address (SrcIP)
- Destination IP address (DstIP)
- Source Port (SPrt) for UDP/TCP and 0 for other protocols
- Destination Port (DPrt) for UDP/TCP, Type/Code for ICMP and 0 for other protocols
- Input – Input Interface (IIF)
- Protocol (Prtcl)

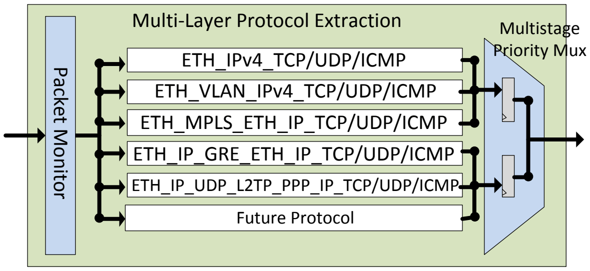

Multi-Layer Protocol Extraction
Multistage Priority Mux
Packet Monitor
ETH_IPv4_TCP/UDP/ICMP
ETH_VLAN_IPv4_TCP/UDP/ICMP
ETH_MPLS_ETH_IP_TCP/UDP/ICMP
ETH_IP_GRE_ETH_IP_TCP/UDP/ICMP
ETH_IP_UDP_L2TP_PPP_IP_TCP/UDP/ICMP
Future Protocol

Figure 7. Multi-Layer Protocol Extraction block diagram

The Protocol Extraction block is composed of following protocols: (1) original L3L4 block provided with the reference NetFlow design [11], (2) MPLS block for the extraction of multi protocol label switched packets with support for only two labels,

(3) GRE block for the parsing of GRE encapsulated protocol packets, (4) L2TP block for mining layer 2 tunneled PPP packets, and (5) Future protocols.

The block level architecture of Multi-Layer Protocol Extraction block is shown in the Figure 7. It consists of a Packet Monitor, a configurable stack of protocol combinations and a Multi-stage Priority Multiplexer. The packet monitor tracks the state of the packet during the extraction process i.e. header or data. Depending on the type of network in use, one can easily add or remove the protocol combinations to handle customized traffic. A very flexible architecture has been provided to ease insertion and deletion of the custom protocol combinations. In cases, where protocol stack increases, the Multi-Stage Priority Multiplexer can be configured to add additional pipeline stages (see Figure 8), to soften the timing constraints.

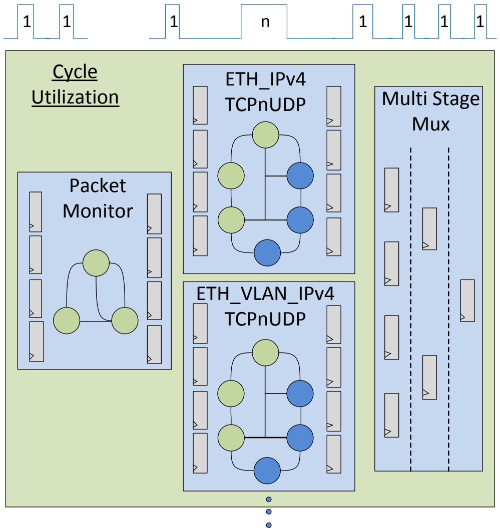
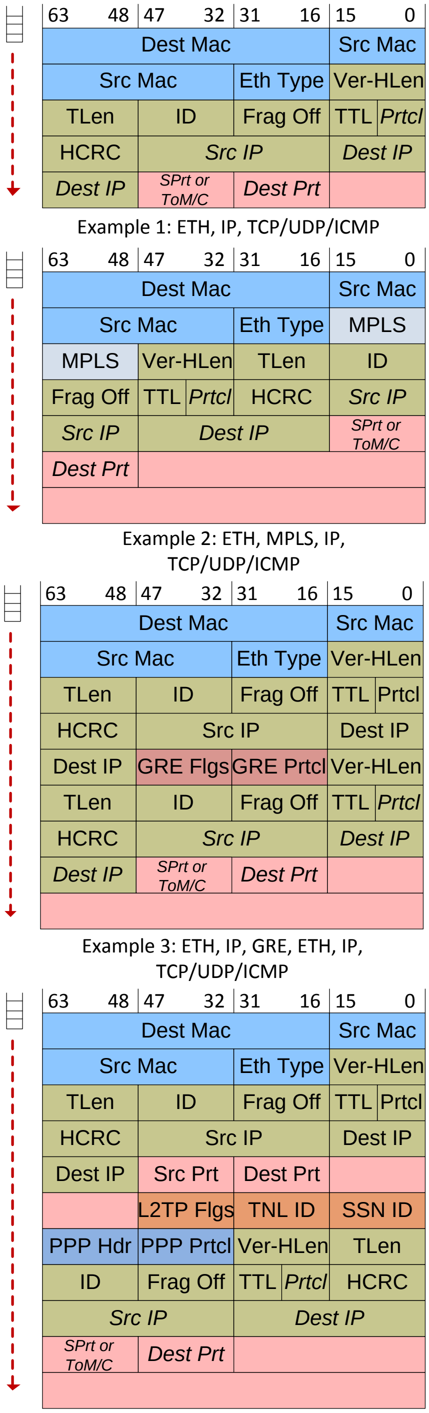


Figure 8. Multi-Stage Protocol Extraction Pipeline

The complete pipeline of the Multi-Stage Protocol Extraction block is shown in Figure 8. When first word of the packet enters the Packet Monitor, it's broadcasted to all the protocol combination blocks in the Multi-Stage Protocol Extraction block with a latency of *2* cycles. After the latency of *n+2* cycles the header information i.e. source IP (SIP), destination IP (DIP), source port (SPORT), destination (DPORT), and Protocol, extracted from one or multiple protocol combination blocks is forwarded to Multi-Stage Priority Multiplexer. Here *n* is equal to the total number of words taken by the largest protocol combination i.e. supported packet with largest header however, if the incoming packet is smaller than the size of the largest protocol combination, *n* is equal to the size of the incoming packet. Each of the protocol combination blocks generates a valid signal in case a match is found for a given packet. The priority of the protocol combinations is maintained by assigning higher priorities to protocols with larger number of combinations and so on. Multi-Stage Multiplexer selects the output of the protocol combination for which there is a match and has the highest priority. The header information is written into the corresponding RFIFO after a pre-defined latency set for the Multi-Stage Multiplexer block, *3* in the case of the design shown in Figure 8. The valid signal acts as a write enable for the RFIFO. The total latency faced by a packet in the design in Figure 8 is *n+7*.


63 48 47 32 31 16 15 0
Dest Mac | Src Mac
Src Mac | Eth Type | Ver-HLen
TLen | ID | Frag Off | TTL | Prtcl
HCRC | Src IP | Dest IP
Dest IP | SPrt or ToM/C | Dest Prt
63 48 47 32 31 16 15 0
Dest Mac | Src Mac
Src Mac | Eth Type | MPLS
MPLS | Ver-HLen | TLen | ID
Frag Off | TTL | Prtcl | HCRC | Src IP
Src IP | Dest IP | SPrt or ToM/C
Dest Prt
63 48 47 32 31 16 15 0
Dest Mac | Src Mac
Src Mac | Eth Type | Ver-HLen
TLen | ID | Frag Off | TTL | Prtcl
HCRC | Src IP | Dest IP
Dest IP | GRE Flgs | GRE Prtcl | Ver-HLen
TLen | ID | Frag Off | TTL | Prtcl
HCRC | Src IP | Dest IP
Dest IP | SPrt or ToM/C | Dest Prt
63 48 47 32 31 16 15 0
Dest Mac | Src Mac
Src Mac | Eth Type | Ver-HLen
TLen | ID | Frag Off | TTL | Prtcl
HCRC | Src IP | Dest IP
Dest IP | Src Prt | Dest Prt
L2TP Flgs | TNL ID | SSN ID
PPP Hdr | PPP Prtcl | Ver-HLen | TLen
ID | Frag Off | TTL | Prtcl | HCRC
Src IP | Dest IP
SPrt or ToM/C | Dest Prt

Example 1: ETH, IP, TCP/UDP/ICMP

Example 2: ETH, MPLS, IP, TCP/UDP/ICMP

Example 3: ETH, IP, GRE, ETH, IP, TCP/UDP/ICMP

Example 4: ETH, IP, UDP, L2TP, PPP, IP, TCP/UDP/ICMP

Figure 9. Examples of Multi-Layer Packets

An example of each of the four supported protocols is shown in the Figure 9. In example 1 and 2, for L3/L4 and MPLS blocks, considering the data word to be 64-bit long; the packet with protocol combination ETH, IP, TCP/UDP/ICMP contains flow information SrcIP, DstIP, SPrt, DPrt and Prtcl etc in word 2, word 3, word 4, and word 5. Similarly in example 3, for GRE block, the flow information is located in words 5, 6, 7, and 8; for example 4, the L2TP block, the flow information is present in words 7, 8, 9, and 10. The Multi-Layer Protocol Extraction block after processing a packet, pushes flow information along with the timestamp in the respective RFIFO. Though the Protocol Extraction block can support IPv6 protocols but for simplicity only IPv4 protocols are considered.

3.1.4.1 MPLS Decoding and Extraction

Multiple Protocol Label Switching (MPLS) tunnels are detected based on lower layer protocol type field as 0x8847. Detection of upper layer protocols in not defined in MPLS standard documents. In order to identify the type of the protocol we propose a detection method based on byte pattern analysis and verification of the upper layer protocol. Currently system supports only IP as MPLS upper layer protocol but any other protocol can easily be integrated. During the analysis of IP detection, the ‘Bottom of Label Stack’ field is checked for value 1 to identify the final MPLS header. The first nibble just after the last MPLS header is checked for IP version (0x4 and 0x6 for IPv4 and IPv6 respectively). If result comes out to be true, we will assume the upper layer to be IP. Next to verify it, we take the sum of the all the header lengths (in bytes) and the total length from the assumed IP header and compare it with the byte count calculated by the RxQ at the time of reception of the packet. If verified then the upper layer protocol is marked as IP.

3.1.5 Hash Generator Block

The Hash Generator block provided with the reference NetFlow design has been used as a baseline [9] [11]. The Packet Bus interface i.e. data bus, control bus, write and ready signals have been replaced with a standard FIFO interface.

The Hash Generator retrieves flow information from the previous RFIFO in the pipeline and computes a 64-bit hash value using CRC-64; and, pushes the hash value in the next RFIFO. The hash value is computed using the following fields only: SrcIP, DstIP, SPrt, DPrt, IIF, and Prtcl.

3.1.6 Flow LookUp & Processing Blocks

Flow LookUp block receives a 64-bit hash value from the Hash Generator block and splits it into two parts. The addressing scheme employed is taken from the reference NetFlow design as shown in Figure 10.

Each memory location, indexed using the first part of the hash value, contains 8 hash values from 8 different network flows (equal in width of the second half of the hash value) referred as fingerprints. These fingerprints are compared to the second half of the split hash. The following assumptions are made depending upon the comparison of the hash value and the finger prints:

- A flow record already exists in the memory, if a match with one of the fingerprints is generated. Its address is acquired as the join of first part of the hash and the rank of the matched fingerprint.
- If there is no match and the number of flow records in the set is lower than 8, free space is used to enter a new fingerprint.
- If there is no match and no space then an arbitrary flow record in the set is expired and replaced with a new one. When a flow record is about to be expired, the Flow LookUp receives a command from the Flow Processing block to delete the corresponding fingerprint from the hash table. It is sent back to the Flow Processing block immediately after the command is executed.

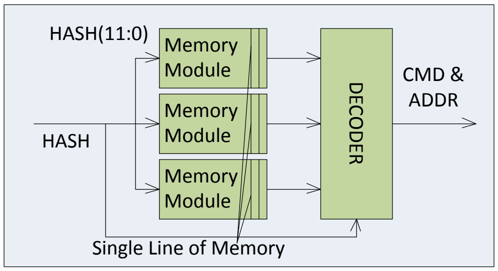


Figure 10. Flow LookUp Block

Flow Processing block controls performs the following tasks:

- Initialization of new flow records
- Updating of existing flow records
- Expiration of inactive flow records

3.1.7 Output Port Lookup Processor

The Record Wrapper block, as mentioned in [11], has been replaced with the OPL Processor block. The OPL processor carries out the same functionalities as that of the Record Wrapper block i.e. it temporarily stores released flow records.

As soon as the following conditions are met, stored flow records are packetized into NetFlow v5 format and written into the OQ block of the NetFPGA platform.

- 15 records are present in the buffer
- The first record in the buffer is 20 ms old

The NetFlow v5 datagram is sent to the output interface(s) specified by the software register [9]. The collector can be installed directly to NetFPGA host machine or output ports of NetFPGA card can emit NetFlow packets to distant collector.

3.1.8 Intermediate Result FIFOs

Result FIFOs have been used to hold intermediate results between the corresponding modules in the NetFlow v5 pipeline as shown in the Figure 4.

3.2 Software Implementation

The host software has been upgraded to provide complete support for virtual interfaces (GRE, L2TP, and MPLS).

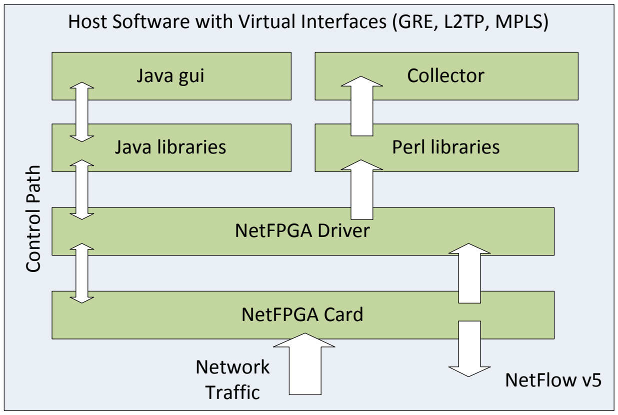


Figure 11. Software Hierarchy

The other basic functionalities like configuration, statistical interfaces and information collection remain the same as mentioned in [9] [11]. Hierarchical view of the software architecture is shown in Figure 11.

4. RESOURCE UTILIZATTION

The resource utilization of the original NetFlow probe architecture and the current architecture is shown in Table 1 and Table 2 respectively. Even though three different technologies of virtual interfaces (GRE, L2TP and MPLS) have been incorporated in the current design, yet there is only an average increase of 3% in the overall utilization of the resources.

Table 1: Resource utilization of the Current Architecture with Virtual Interfaces

Resources	**XC2VP50 Utilization**	**Utilization Percentage**
Slices	18276 out of 23616	77%
4 - Input LUTS	25165 out of 47232	53%
Flip Flops	21244 out of 47232	44%
Block RAMs	200 out of 232	86%

Table 2: Resource utilization of the Original NetFlow probe Architecture

Resources	**XC2VP50 Utilization**	**Utilization Percentage**
Slices	17617 out of 23616	74%
4 - Input LUTS	23319 out of 47232	49%
Flip Flops	19504 out of 47232	41%
Block RAMs	200 out of 232	86%

5. CONCLUSION

We propose a protocol analysis layer for NetFlow probe pipelined architecture. The architecture is implemented on the NetFPGA platform. The implemented architecture targets all the present and future IPv4 and IPv6 protocols. The GRE and L2TP protocol are especially selected in order to demonstrate the flexibility and the scalability of the architecture. The architecture can be used for applications involving deep packet inspection such as VoIP monitoring and NextGen firewalls besides NetFlow probe. The proposed architecture is scalable and can be implemented on any FPGA board. The newer protocols can be independently added in the protocol analyzer layer without modifying the already implemented protocols in the layer.

6. REFERENCES

[1] Cisco: NetFlow Protocol http://www.cisco.com/en/US/products/ps6601/products_ios_protocol_group_home.html

[2] Wiki: NetFlow Protocol http://en.wikipedia.org/wiki/Netflow

[3] Cisco White Paper. NetFlow on Logical Interfaces: Frame Relay, Asynchronous Transfer Mode, Inter-Switch Link, 802.1 q, Multilink Point to Point Protocol, General Routing Encapsulation, Layer 2 Tunneling Protocol, Multiprotocol Label Switching VPNs, and Tunnel, March 2006.

[4] G. A. Covington, G. Gibb, J. Naous, J. Lockwood, and N. McKeown. Methodology to contribute netfpga modules. In *International Conference on Microelectronic Systems Education (submitted to)*, 2009.

[5] G. Gibb, J. W. Lockwood, J. Naous, P. Hartke, and N. McKeown. Netfpga: An open platform for teaching how to build gigabit-rate network switches and routers. In *IEEE Transactions on Education*, August 2008.

[6] J. W. Lockwood, N. McKeown, G. Watson, G. Gibb, P. Hartke, J. Naous, R. Raghuraman, and J. Luo. Netfpga - an open platform for gigabitrate network switching and routing. In *International Conference on Microelectronic Systems Education*, 2007.

[7] G. Watson, N. McKeown, and M. Casado. Netfpga - a tool for network research and education. In *2nd Workshop on Architecture Research using FPGA Platforms (WARFP)*, Febuary 2006.

[8] G. A. Covington, G. Gibb, J. Naous, J. Lockwood, and N. McKeown. A Packet Generator on the NetFPGA Platform. In *IEEE Symposium on. Field-Programmable Custom Computing Machines (FCCM)*, April 2009.

[9] NetFPGA: NetFlowProbe http://www.netfpga.org/foswiki/bin/view/NetFPGA/OneGig/NetFlowProbe

[10] NetFPGA: Guide http://netfpga.org/netfpgawiki/index.php/Guide

[11] CESNET: NetFlow probe on NetFPGA http://www.liberouter.org/~xzadni00/netflowprobedoc.pdf