Flow Monitoring in High-Speed Networks with 2D Hash Tables

David Nguyen, Joseph Zambreno, and Gokhan Memik

Department of Electrical and Computer Engineering
Northwestern University
Evanston, Illinois 60208
{dnguyen, zambro1, memik}@ece.northwestern.edu

Abstract. Flow monitoring is a required task for a variety of networking applications including fair scheduling and intrusion/anomaly detection. Existing flow monitoring techniques are implemented in software, which are insufficient for real-time monitoring in high-speed networks. In this paper, we present the design of a flow monitoring scheme based on two-dimensional hash tables. Taking advantage of FPGA technology, we exploit the use of parallelism in our implementation for both accuracy and performance. We present four techniques based on this two-dimensional hash table scheme. Using a simulation environment that processes packet traces, our implementation can find flow information within 8% of the actual value while achieving link speeds exceeding 60 Gbps for a workload with constant packet sizes of 40 bytes.

1 Introduction

There is a tremendous growth in the complexity of networking applications. Many applications (such as QoS, fair packet scheduling, intrusion/anomaly detection, firewalls, traffic engineering) require flow information[1] [5]. Because of increasing wire speeds, there is a need for hardware-based flow monitoring techniques in high-speed networks. However, most routers do not implement flow monitoring. The existing solutions, which also include software solutions, are either too slow or inaccurate.

In this paper, we present four novel techniques utilizing a two dimensional hash table to gather flow information. We implement our Flow Monitoring Unit (FMU) using a Xilinx Virtex II XC2V8000 [8] chip and achieve throughput speeds up to 73 Gbps without sacrificing accuracy. We observe different designs perform better for certain workloads/requirements. Because network traffic profiles are all unique, FPGAs are an attractive implementation choice for their reconfigurable properties.

In Section 2, we explain the flow monitoring techniques. Section 3 presents the FPGA implementation. In Section 4, we discuss the experimental results and Section 5 concludes the paper.

[1] In this paper, the terms flow and session are used interchangeably; both correspond to a TCP session. Hence, flow information is statistics (such as total traffic generated) about a TCP connection collected at a router.

J. Becker, M. Platzner, S. Vernalde (Eds.): FPL 2004, LNCS 3203, pp. 1093–1097, 2004.

2 Flow Monitoring Unit (FMU)

2.1 Queries

The host machine uses the FMU unit by sending two types of queries.

UPDATE(k, v) : Increase the value of the key k by v.

GET(k) : Return the value for the key k.

The key k is a combination of five TCP packet header fields: source IP, destination IP, source port, destination port, and protocol. In our design, the FMU stores the number of packets for each flow. Note, while tracking different flows, it is possible for collisions to occur. Therefore, the GET query does not always return the correct value. We will discuss this in Section 4 with error analysis.

2.2 Flow Monitoring Techniques

As mentioned, our FMU is based on hashing. We chose the Jenkins hash function [4] for this study for its proven performance for hash tables. For a two-dimensional hash function with dimensions NxS, there are N hash tables each with S elements. Each of these tables is addressed by a different hash function[2]. The GET(k) function takes the results from each hash table N and uses one of the following techniques:

Min FMU: The simplest technique is called the *min FMU (MIFMU)*. MIFMU reads the corresponding values from the tables and returns the smallest value. This method is most accurate for large flows.

$$\min \{T_i[h_i(k)]\}, \quad \forall i \in \{0,..,N-1\}$$

Median FMU: The second technique is the *median FMU (MEFMU)*. As stated, this method returns the median of the values (corrected with a balanced hash factor sum/S) from the tables for a key k.

$$\text{median } \{T_i[h_i(k)] - \frac{sum}{S}\}, \quad \forall i \in \{0,..,N-1\}$$

Collision Estimate FMU: The *collision estimate FMU (CEFMU)* estimates the number of collisions for each hash bucket and returns the output values according to a collision counter. The collision counter is incremented when the current access does not match the last access to the hash table. An additional table ($C_i[h_i(k)]$) stores the collision counters. This method is most accurate for small flows.

$$\min \{T_i[h_i(k)] - C_i[h_i(k)]\}, \quad \forall i \in \{0,..,N-1\}$$

Hybrid FMU: The final technique is the *hybrid FMU (HYFMU)*, which runs both CEFMU and MIFMU techniques in parallel and returns one value. Again by virtue of FPGA design, there is a negligible performance hit. A threshold value is used to select the hybrid value. HYFMU subtracts the output of CEFMU from the output of

[2] In our implementation, to achieve equal timing, we utilized the same hash function with different hash seeds to generate the effect of different hash functions.

MIFMU. Given a threshold, MIFMU is used for larger estimates and CEFMU for smaller estimates.

3 FPGA Implementation

We implemented the FMU using the Synplify Pro synthesis tool [7] and Xilinx Design Manager implementation tools. We chose Xilinx VirtexII XC2V8000 FPGA as our target chip. The FMU architecture is presented in Figure 1. According to the FMU type (MIFMU, MEFMU, CEFMU, HYFMU), the selection mechanism returns the corresponding output. The resource limit for this chip was memory (S=80,000 entries) for the hash tables. This fact favors using two-dimensional hash tables. For one, there is smaller access latency because multiple smaller tables are accessed in parallel. Also, the two-dimensional hash table design inherently performs better than a single hash table.

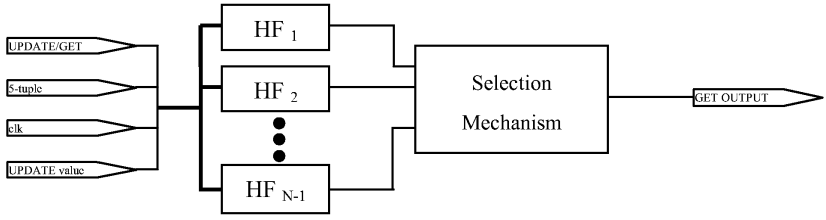

Fig. 1. Overview of the FMU

Table 1 presents the critical path for different FMU designs. Extensive pipelining increases the overall throughput significantly. The rightmost column of Table 1 presents the corresponding maximum bandwidth supported for GET queries. This value is calculated for constant packet sizes of 40 bytes.

Table 1. The latency of critical paths of various FMU components

Configuration	Critical Path Delay	Max. Bandwidth
N = 1, S = 8K	6.63 ns.	48.3 Gbps
N = 4, S = 2K, MIFMU	4.33 ns	73.9 Gbps
N = 4, S = 2K, MEFMU	4.99 ns.	64.1 Gbps
N = 4, S = 2K, CEFMU	4.33 ns.	73.9 Gbps
N = 4, S = 2K, HYFMU (combining min and CE)	4.99 ns.	64.1 Gbps

4 Experimental Results

In this section, we present the simulation results for the different FMU techniques (MIFMU, MEFMU, CEFMU, and HYFMU as explained in Section 2.2) we have developed. We implemented a simulator that processes NLANR packet traces [6] and

executes the FMU techniques. For error analysis, the simulator finds the exact number of packets for each flow in a packet trace. We report the *error rate*, which is the average error for finding flow size of all the flows in a trace.

First, we compare different FMU techniques. Figure 2 presents the average error rates for the four FMU techniques and varying table size S=500 to S=16000 entries. We set the number of parallel hash functions to N=4. The CEFMU technique has the best overall performance for varying configurations. For the largest setup, N=4 and S=16,000, HYFMU gives the best performance with an error rate of 7.3% obviously because it employs both MIFMU and CEFMU methods.

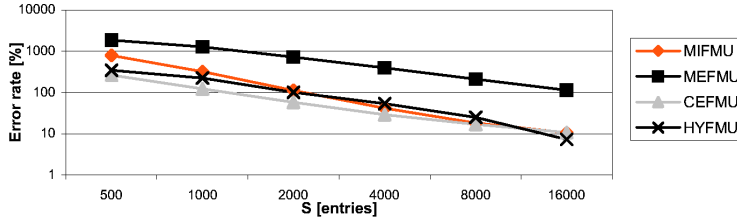

Fig. 2. Average error rates of various FMU techniques.

4.1 Sensitivity Analysis

For sensitivity analysis, we fix the total size for the hash tables (S) and vary the number of parallel hash functions (N) to find the optimal parallelism. The results for S_{total}=32K entries are presented in Figure 3. For all techniques except MEFMU, increasing the number of parallel hash functions initially results in a reduction in the error rate. Particularly, for S_{total}=32K, the error rate reduces from 69% to 17%, from 59% to 11%, and from 63% to 19% for MIFMU, CEFMU, and HYFMU, respectively. The CEFMU technique, on the other hand, improves its performance slightly until N=4. This is because CEFMU is more immune to collisions, which occur more frequently in smaller hash tables. CEFMU is designed to perform well under these circumstances.

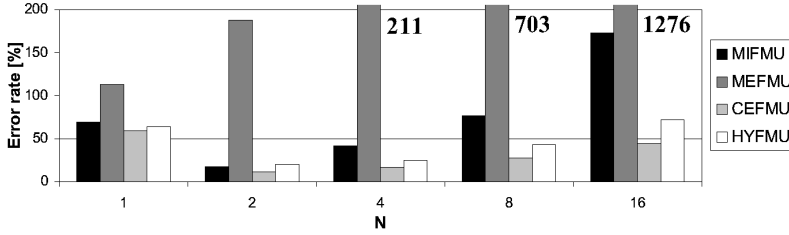

Fig. 3. Error rates for a total table size S_{total}=32K.

5 Conclusions

Flow monitoring is an important task in computer networks. However, almost all techniques are implemented in software and designing hardware for them is very hard if not impossible. With the increase in the link speeds and the wider usage of flow information, hardware flow monitoring is becoming essential for most state-of-the-art routers. In this paper, we presented a hardware flow monitoring design implemented on FPGAs. The FMU unit is based on two-dimensional hashing. With hashing, it has two major advantages over alternative techniques. First, high speeds can be achieved. And second, the access latency to any data is constant. Clearly, any inaccuracies are a result of depending on the performance of the hash function. Multiple hash functions running in parallel on an FPGA alleviates the inherent accuracy penalty. We have applied four different techniques to address this problem. The best technique (HYFMU), which combines CEFMU and MIFMU, can process 200M packets per second (corresponding to a link speed of 64 Gbps for 40-byte packets), while having an average error rate of 7.3%.

References

1. Arsham, H. Time Series Analysis and Forecasting Techniques, http://obelia.jde.aca.mmu.ac.uk/resdesgn/arsham/opre330Forecast.htm
2. Carter, J. and M. Wegman, *Universal classes of hash functions.* Journal of Computer and System Sciences, 1979, 18: p. 143--154.
3. Cheung, O. Y. H., P. H. W. Leong. Implementation of an FPGA Based Accelerator for Virtual Private Networks. In *IEEE International Conference on Field-Programmable Technology (FPT' 02)*, Dec. 2002. Hong Kong, China.
4. B. Jenkins, Hash Functions and Block Ciphers, http://www.burtleburtle.net/bob/hash/index.html
5. C. Madson, , L. Temoshenko, C. Pellecuru, B. Harrison, and S. Ramakrishnan, *IPSec Flow Monitoring MIB Textual Conventions*. Mar. 2003, Internet Engineering Task Force.
6. NLANR Project. Network Traffic Packet Header Traces, http://moat.nlanr.net/Traces
7. Synplicity Inc. Synplify Pro: The Most Powerful HDL Synthesis Solution for Multi-Million Gate Programmable Logic Designs, 2000, www.synplify.com
8. Xilinx Inc., *SPEEDRouter v1.1 Product Specification*. Oct. 2001.