Hardware-Based "on-the-fly" Per-flow Scan Detector Pre-filter (Poster)

Salvatore Pontarelli, Simone Teofili, and Giuseppe Bianchi

Università degli Studi di Roma Tor Vergata
Via del Politecnico, 1 00133 Rome, Italy
{salvatore.pontarelli,simone.teofili,giuseppe.bianchi}@uniroma2.it

Abstract. Pre-filtering monitoring tasks, directly running over traffic probes, may accomplish a significant degree of data reduction by isolating a relatively small number of flows (likely to be of interest for the monitoring application) from the rest of the traffic. As these filtering mechanisms are conveniently run as close as possible to the data gathering devices (traffic probes), and must scale to multi-gigabit speed, the feasibility of their implementation in hardware is a key requirement. In this paper, we document a hardware FPGA implementation of a recently proposed network scan pre-filter. It leverages processing stages based on Bloom filters and Counting Bloom Filters, and it is devised to detect, through on-the-fly per-packet analysis, the flows which potentially exhibit a network/port scanning behaviour. The framework has been implemented in a modular manner. It suitably combines two different general-purpose modules (a rate meter and a variation detector) likely to be reused as building blocks for other monitoring tasks. In the following presentation, we further discuss some lessons learned and general implementation guidelines which emerge when the goal is to efficiently implement run-time updated (*i.e.*, dynamic) Bloom-filter-based data structures in hardware.[1]

1 Introduction

In high-speed (multi-gigabit) networks, traffic analysis and network monitoring functions based on the traditional gather-first-process-later paradigm appear inconvenient. Indeed, the relative number of flows which exhibit a behaviour worth of detailed investigation can be small with respect to the total amount of traffic carried over a network link, and monitoring solutions which mandate to deliver *all* the captured traffic to a remote device for analysis and inspection appear to pose a huge and unnecessary demand on the network monitoring infrastructure.

Starting from the seminal paper from Estan and Varghese [1], a new direction in traffic monitoring emerged. The idea is to delegate some traffic processing tasks to traffic probes. Specifically, probes may support traffic analysis functions whose goal is to detect and isolate flows which exhibit a potentially anomalous

[1] This work has been partially supported by the European Commission in the frame of the DEMONS project http://fp7-demons.eu/

J. Domingo-Pascual, Y. Shavitt, and S. Uhlig (Eds.): TMA 2011, LNCS 6613, pp. 181–184, 2011.

behaviour, where said "anomalous behaviour" is suitably defined in correspondence of the specific monitoring application goals. As a result, monitoring infrastructure scalability is accomplished by dramatically reducing the amount of data ultimately delivered to remote monitoring applications.

An extensive amount of literature has shown that several meaningful traffic analysis tasks, such as heavy flows detection [1], traffic classification [2], worm fingerprinting [3], scan detection [4][5], and so on, can be conveniently performed "on-the-fly" over a memory/resource-constrained device such as a hardware traffic probe. When such analysis function are implemented as pre-filters, the accuracy requirements are somewhat reduced (the goal to provide a final answer is delegated to the remote monitoring application), and approximate low-resource consuming data filters and structures, such as those based on Bloom Filters and their extensions [6], seems very appealing.

Among these papers, our previous work [5] presents a novel method for detecting and accounting for *variations*, which are usually a component of a more comprehensive network scan activity. Specifically, we aim at detecting flows which exhibit a significant difference, in time, with respect to one or more parameters (for instance traffic which originates by a same IP source address and "hits" a large number of IP destinations; ARP requests showing a large IP target variation; TCP packets addressed to a large amount of ports, etc). It was shown that this approach may achieve a significant reduction ($95 - 98\%$) in terms of number of flows which require further fine-grained analysis, meanwhile using a relatively small amount of memory resources over the traffic probe (order of a few hundreds of kbits, dependent on the accuracy target).

This work complements our former work [5] by showing how the specific filtering mechanism therein proposed can be implemented on a reprogrammable hardware device (*i.e.* FPGA), thus proving the the viability of that approach in a high speed network monitoring scenario.

2 Architecture

Figure 1 shows the two-stage filtering approach proposed in [5]. For reasons of space, the reader is referred to the original paper for a detailed functional explanation. In what follows, we focus on the hardware implementation decisions concerning each stage, indeed influenced by the need to handle Bloom-type filters dynamically updated on a per-packet basis.

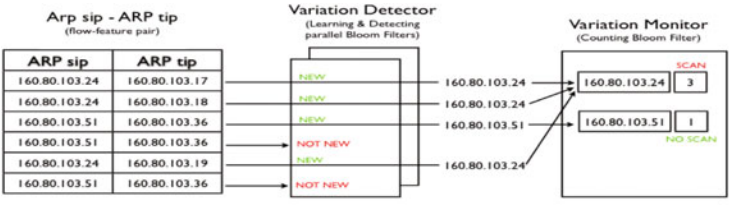

Fig. 1. Two stage filtering - example for ARP traffic

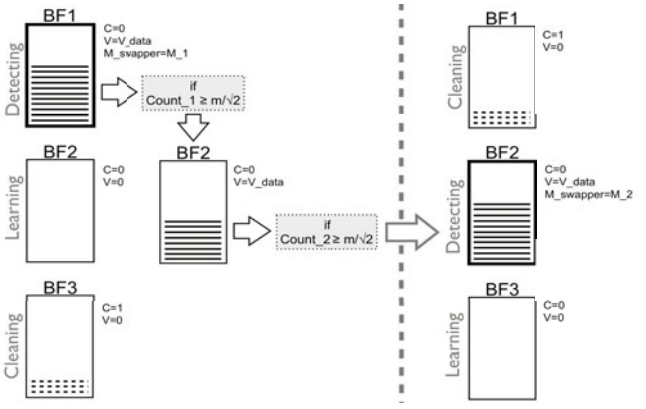

Fig. 2. variation detector: Swapping Bloom Filters

2.1 Stage 1: Variation Detector

We recall from [5] that a variation detector can be constructed using two self-clocked swapping Bloom Filters. The major hardware implementation concern we had to face consisted in the time needed to reset one of the two Bloom filters upon filter swapping. Indeed, such reset procedure needed to be accomplished "instantaneously", *i.e.* in the short time frame elapsing between the arrival of two consecutive packets.

As a solution, we have resorted to implement three Bloom filters instead of the original two. Such three Bloom filters are in three different operating states: Detecting, Learning, Cleaning. The filter in Detecting state, for each received packet, checks if the string obtained by combining the selected flow key (e.g. the IP source address for network scan) and the selected feature key which is monitored to identify a variation in the flow (e.g. the IP destination address) is already stored in the Bloom Filter; if not, it is inserted and the flow is accounted in the next variation monitor module. The filter in Learning state is updated according to the same rule, but its output is discarded. The only role of the Cleaning state is to wipe the content of the filter. Specifically, when the Detecting filter is full (according to the rules defined in [5]), filter swapping occurs so that the previously Learning filter becomes the Detecting one, and this latter enters into Cleaning state.

Note that the development of a supplementary Cleaning filter was a necessary consequence of our design decision of developing Bloom filters in the FPGA Block RAM: RAM zeroing in fact requires a number of clock cicles linearly increasing with the filter dimension, and in any case always greater (for practical filter sizes) than the time available between two consecutive packets. Specifically we need a clock cicle to reset 32 bits (*i.e.* 1000 clock cicles for Bloom filter of 32000 bits). The alternative solution of implementing the Bloom filters in LUTs was not considered viable because of FPGA area consumption. The implementation of 3 Bloom Filters of 32000 bits each would have required the usage of 93% of the logic resources of a Xilinx Virtex 5 [7].

2.2 Stage 2: Variation Monitor

The variation monitor is composed of a Counting Bloom Filter (CBF) Block that receives, as inputs, the flow string signalled by the variation detector as carrying a new flow/feature pair. Such CBF is transformed into a rate monitor [5] by periodically decrementing all the filter bins. Again, this specific additional functionality requires a careful hardware implementation, when (as per our choice), also the CBF is implemented using the Block RAM. Note that a periodic decrement operation, for the same reasons described above, applied to all the CBF bins requires a large number of memory accesses (further slowed by the need to perform arithmetic operations over the bin values themselves). However, in this case, there is no possibility to deploy an associated cleaning filter; rather, the hardware must make possible to decrement the filter while its state is maintained and eventually updated. As a result,we have introduced a pointer to the last decremented counter together with a compensation circuitry subtracting 1 to the value taken from the bins not yet decremented.

3 Results

The presented paper proposes an implementation of the framework presented in [5] on a reprogrammable hardware device (*i.e.* FPGA). Our single-pipe implementation, on a low-end Virtex II pro [8] reaches the frequency of $183MHz$ overcoming the minimum frequency required to process data over a $1Gb/sec$ link ($125MHz$). The achievement of higher rates (in the order of few tens of Gbps) appears easy by 1) exploiting more performing FPGA, and 2) by designing multiple parallel processing pipe. The implemented framework exploits a very limited amount of physical resources. In a Virtex II pro [8] that is a low-end FPGA we need only 20% of the Block RAM (50 of 232) and the 4% of the Flip-Flop (1920 of 46632). In a Virtex 5 [7] we exploit 15% of the Block RAM and less then 2% of the Flip-Flop.

References

1. Estan, C., Varghese, G.: New Directions in Traffic Measurement and Accounting. In: SIGCOMM 2002 (August 2002)
2. Dharmapurikar, S., Song, H., Turner, J., Lockwood, J.: Fast Packet Classification Using Bloom Filters. In: Proceedings of the 2006 ACM Symposim on Architectures for Networking and Communications Systems (December 2006)
3. Singh, S., Estan, C., Varghese, G., Savage, S.: Automated Worm Fingerprinting. In: 6th Usenix Symposium on Operating Systems and their Applications (2004)
4. Kong, S., He, T., Shao, X., Li, X.: A Double-Filter Structure Based Scheme for Scalable Port Scan Detection. In: IEEE ICC 2006, Istanbul, Turkey (2006)
5. Bianchi, G., Boschi, E., Teofili, S., Trammell, B.: Measurement Data Reduction through Variation Rate Metering. In: INFOCOM 2010 (March 2010)
6. Broder, A., Mitzenmacher, M.: Network Applications of Bloom Filters: A Survey. Internet Mathematics 1(4), 485–509 (2005)
7. Virtex-5 Family Overview, `http://www.xilinx.com/support/documentation/data_sheets/ds100.pdf`
8. Virtex-II Pro and Virtex-II ProX FPGA User Guide, `http://www.xilinx.com/support/documentation/user_guides/ug012.pdf`