Blockmon: A High-Performance Composable Network Traffic Measurement System

Felipe Huici*, Andrea di Pietro±, Brian Trammell§, Jose Maria Gomez Hidalgo†, Daniel Martinez Ruiz†, Nico d'Heureuse*

*NEC Europe Ltd, ±CNIT, †OPTENET, §ETH Zurich

ABSTRACT

Passive network monitoring and data analysis, crucial to the correct operation of networks and the systems that rely on them, has become an increasingly difficult task given continued growth and diversification of the Internet. In this demo we present Blockmon, a novel composable measurement system with the flexibility to allow for a wide range of traffic monitoring and data analysis, as well as the necessary mechanisms to yield high performance on today's modern multi-core hardware. In this demo we use Blockmon's GUI to show how to easily create Blockmon applications and display data exported by them. We present a simple flow meter application and a more involved VoIP anomaly detection one.

1. INTRODUCTION

Two salient trends have dominated Internet-scale measurement over the past decade: the continued growth of the Internet, both in terms of attached nodes and total data transferred; and the diversification of devices attached to and applications running over the network.

These challenges point to the need for a high-performance, yet easily-extensible solution. To this end we demonstrate Blockmon, a system for supporting high-performance *composable measurement*: building fast network measurement applications out of small, discrete blocks.

Systems work in network measurement is by no means a new field. The modular principles in Blockmon were inspired by the Click modular router [4], though Click is oriented towards packet-based processing. Other programmable tools such as CoMo [3] or PF_RING [2] are either less flexible than or do not perform as well as Blockmon.

Blockmon contributes some key innovations to this wide base of work, focusing on applying programmable measurement to today's traffic loads and diversity while maximizing the use of modern, multi-core commodity hardware to ensure high performance. Blockmon is available as open-source software at `http://blockmon.github.com/blockmon`.

2. SYSTEM DESCRIPTION

At a high-level, Blockmon provides a set of units called *blocks*, each of which performs a certain discrete processing action, for instance parsing a DNS response, or counting the number of distinct VoIP users on a link. The blocks communicate with each other by passing *messages* via *gates*; one block's output gates are connected to the input gates of other blocks, which allows runtime indirection of messages. A set of inter-connected blocks implementing a measurement application is called a *composition*; a generic composition is shown in figure 1.

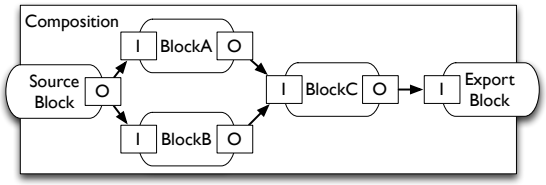

Figure 1: Example composition

The Message class provides a basic interface for identifying message types, and for supporting import and export of messages in order to connect compositions across nodes. Messages are constant in order to ensure that they can be shared without contention among multiple blocks concurrently, and provide a tagging mechanism to allow Blocks to add small bits of data to a message in a thread-safe manner. This latter mechanism allows blocks to process high volume messages (e.g., packets) in a pipeline without incurring the overhead of having to allocate new message objects.

The assignment of activities to threads and threads to CPU cores can have a large impact on performance [1]. To leverage this, Blockmon schedules work in thread pools. Each block is assigned to a pool via the composition, and pools can be pinned to specific cores. This model allows flexibility in terms of which block is executed on which CPU core.

Input gates are implemented either via direct invocation, where messages are passed via method call; or indirect invocation, where messages are passed via a lock-free rotating queue. Direct invocation causes the downstream block to run within the thread of the upstream block, while indirect invocation separates threads from each other without lock contention. Whether blocks are invoked directly or indirectly is another method for tuning performance.

Additional performance improvements include efficient message passing via C++11 shared pointers and object-move semantics, batch allocation of memory for messages, and

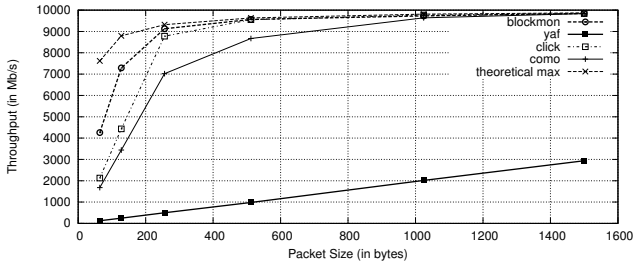

Figure 2: Performance comparison between Blockmon and other existing systems.

support for fast capture blocks using novel packet capture engines such as PFQ [5]. These improvements combine to produce the results in figure 2, which show Blockmon outperforming other systems when they all run a simple application that keeps per-flow statistics (byte and packet counts).

3. DEMONSTRATION DESCRIPTION

We will demonstrate two applications built on top of Blockmon: a simple monitoring application that keeps per-flow statistics such as byte and packet counts, and a more complex application that can detect anomalies or abuse in VoIP CDR (call data record) traces. The demonstration will focus on the detection of telemarketing activity. We will show the operation of Blockmon together with a Python-based back-end daemon and a web GUI front-end implemented in PHP and Javascript.

The Blockmon GUI consists of two views. The first is the composition view, which allows users to graphically design a composition and run it in the system. This view connects to the Blockmon back-end via a JSON-based protocol and retrieves a list of available blocks, as well as information about them (e.g., what they do, what parameters and gates they have, etc). The user then drags and drops blocks onto the main canvas, configures their parameters and connects them. In addition, the GUI allows blocks to be mapped to threadpools, and threadpools to be mapped to the available CPUs (see figure 3).

Once this is all done, the GUI sends a command to the back-end to start the composition running. While running, the user can inspect internal variables of blocks. For instance, the *PacketCounter* block has a variable *pktcnt* which returns the number of packets the block has seen so far; writing to the *reset* variable resets this count.

The GUI's second view is a dashboard, which is in charge of displaying the actual data produced by the running Blockmon compositions. The dashboard provides a number of display types based on the Google Chart Tools[1] and Highcharts JS[2] (e.g., line graphs, bar and pie charts, etc.) which are constantly updated according to the output of the compositions. The user can configure the most appropriate graphs to monitor the output of a composition, which is temporarily stored in an internal buffer to allow the dashboard graph to read and update the results on the screen.

The simple flow statistics application is intended as a "Hello, World" demonstration of Blockmon and the concepts of composable measurement. To demonstrate it, we draw its composition in the GUI, then start it. The composition

[1] `https://google-developers.appspot.com/chart/interactive/docs/index`
[2] `http://www.highcharts.com/`

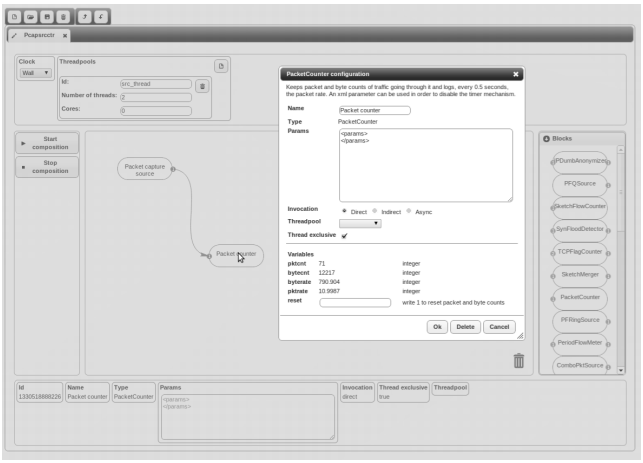

Figure 3: Blockmon GUI showing a composition.

keeps track of flows, periodically exporting data in IPFIX [6] format about the N flows with the largest byte and packet counts. We then show how to set up the dashboard to display a graph showing the moving byte rates of the top N flows. We finally run the drawn composition and show the tracked flows statistics in the configured dashboard graphs.

We then turn to a VoIP anomaly detection application, intended as a demonstration of the ease of porting an existing non-trivial application to Blockmon. This application detects anomalies such as telemarketers based on CDR (call data record) traces. Once again we show how to draw up its composition and set up the dashboard in the GUI. This time the dashboard displays a listing of the most anomalous users, including time-varying anomaly detection scores for the system's various anomaly detection algorithms. We will also show how to use the composition view to inspect the value of a block's variable by displaying the rate at which the block *CDRSource* is processing CDRs.

4. REFERENCES

[1] Egi, N., Greenhalgh, A., Handley, M., Hoerdt, M., Huici, F., and Mathy, L. Towards high performance virtual routers on commodity hardware. In *Proceedings of ACM CoNEXT 2008* (Madrid, Spain, December 2008).

[2] Fusco, F., and Deri, L. High speed network traffic analysis with commodity multi-core systems. In *Proceedings of the 10th annual conference on Internet measurement* (2010), IMC '10, ACM, pp. 218–224.

[3] Iannaccone, G. Fast prototyping of network data mining applications. In *Passive and Active Measurement Conference 2006* (Adelaide, Australia, March 2006).

[4] Kohler, E., Morris, R., Chen, B., Jahnotti, J., and Kasshoek, M. F. The click modular router. *ACM Transaction on Computer Systems 18*, 3 (2000), 263–297.

[5] N.Bonelli, Pietro, A. D., Giordano, S., and Procissi, G. On multi–gigabit packet capturing with multi–core commodity hardware. In *Passive and Active Measurement conference (PAM)* (2012).

[6] Trammell, B., and Boschi, E. An introduction to ip flow information export. *IEEE Communications Magazine 49*, 4 (Apr. 2011).