Blockmon: Flexible and High-Performance Big Data Stream Analytics Platform and its Use Cases

Maurizio Dusi, Nico d'Heureuse, Felipe Huici, Andrea di Pietro
Nicola Bonelli, Giuseppe Bianchi, Brian Trammell, Saverio Niccolini

Abstract

This paper describes Blockmon, a novel, configurable, software-based Big Data platform for high-performance data stream analytics. Its design allows running applications for a wide range of use cases. When used as network data processing and monitoring platform, it copes with 10 Gb/s of continuous traffic, up to layer 7 (e.g., DPI), on a single commodity server. When used as a server-log processing platform, a fraud detection algorithm built on top of Blockmon can analyze up to 70,000 cps (~250 million BHCA, enough to cope with Japan’s entire phone traffic) on a single machine. Blockmon will be commercialized and applied to analyze operator networks and other applications such as web analytics or financial market analysis, among others.

Keywords

data analysis, modular platform, network and data processing

1. Introduction

The increase of nodes attached to the Internet, from desktop PCs to smartphones, as well as the plethora of applications used to interconnect people, from emails to social networks, have resulted in the amount of data being transferred through the Internet to constantly increase and diversify, doubling every two to three years; this trend is forecast to continue through 2015 [1)].

The variety and the amount of data being exchanged challenge existing mechanisms for network monitoring and analysis, which must be designed to be:

(1) flexible, to easily adapt to emerging applications; and
(2) high-performance, to process data close to real-time, and react to ensure security, quality of service, and prompt operational planning.

We believe Blockmon, a system designed for supporting high-performance composable measurement out of small, discrete blocks, can fulfill those requirements. Blockmon is available open-source under the BSD license [2)].

The rest of this paper is organized as follows. In Section 2 we discuss related work. Section 3 describes the Blockmon architecture. Section 4 shows how to develop Blockmon applications. Section 5 outlines a fraud detection use case, and shows the gain in performance that we achieve when using Blockmon. Section 6 concludes the paper.

2. Related Work

Blockmon takes some of the design principles of previous approaches, enhances them to allow for a wider range of monitoring and analysis applications, and provides tuning mechanisms that allow it to yield high performance when running on modern, multi-core, multi-queue architectures. Further, by allowing runtime reconfiguration of the connections among its blocks, Blockmon enables on-line data analysis of traffic, which adjusts to current network conditions.

Blockmon builds upon previous work in programmable online network measurement and composable networking. CoMo [3)], a passive monitoring system, introduced the concept of a monitoring plugin, even though its monitoring applications rely on a set of limited and strictly pre-defined callback functions.

The Click modular router [4)] has inspired the modular principles in Blockmon. However, Click is intended only for packet processing, with its modules only able to communicate in terms of packets. Further, Click cannot easily do more advanced types of processing such as maintaining TCP connections, interacting with databases, or any number of actions relevant to monitoring and data analysis.

Yahoo! S4 [5)] and TwitterStorm [6)] are two recent frameworks designed with a goal similar to Blockmon, i.e., a distributed platform for stream processing. An in-depth evaluation of how those platforms compare against Blockmon in terms of performance and flexibility is part of our future work.

3. Blockmon: Overview

Blockmon is built upon the notion of blocks, gates and messages.

Blockmon provides a set of units called blocks each carrying out a certain type of processing, for instance counting the number of distinct VoIP users on a link. Blocks can be configured so that two blocks of the same type may be initialized differently, thus ensuring a modular and flexible system. The blocks are then inter-connected via a set of input and output "gates;" the number of gates and what they are used for are defined by the developer of the block. "Messages" are the information units that blocks exchange between each other: a message carries the results of the processing of one block to the next block down the line.

A set of inter-connected blocks representing a monitoring and data analysis application is called "composition" (see **Fig. 1**), which is specified in XML. The Blockmon core and the blocks themselves are implemented in C^{++}, and the system is controlled at runtime through a Python-based command-line interface.

Several blocks are already made available and cover the basic functionality required to design network monitoring compositions. For instance, there are fast capture blocks for packet processing (pcap, pf_ring, pfq), blocks for keeping track of flows, and blocks for statistics (Bloom filters and packet counters). Beyond these, users can implement additional blocks and connect them to existing ones as needed. As Blockmon is written in C^{++}, porting existing functionality from existing C/C^{++} code to it is straight-forward.

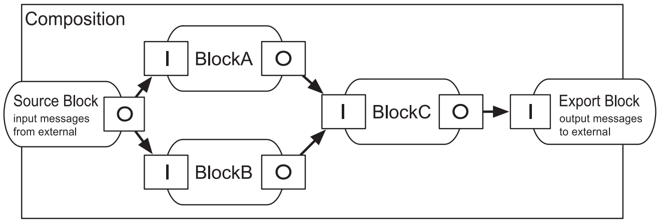


Fig. 1 Blockmon: blocks are inter-connected through input/output gates to form a composition.

4. Creating Blockmon Applications

(1) Developer's Perspective: The Block

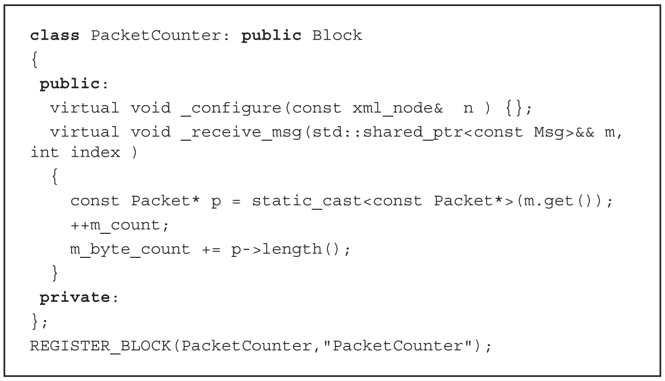
To run inside Blockmon, a block inherits from the core Block class, and implements at least two methods, called when appropriate by the framework:

- **_configure():**
  called with a reference to the XML element used to configure the block, at composition startup time;
- **_receive_msg():**
  called with a reference to the message and the gate on which it was received.

For instance, implementing a block for counting packets only requires overloading the _receive_msg function to get the message and increment a counter. **Fig. 2** shows code for the packet counter block.

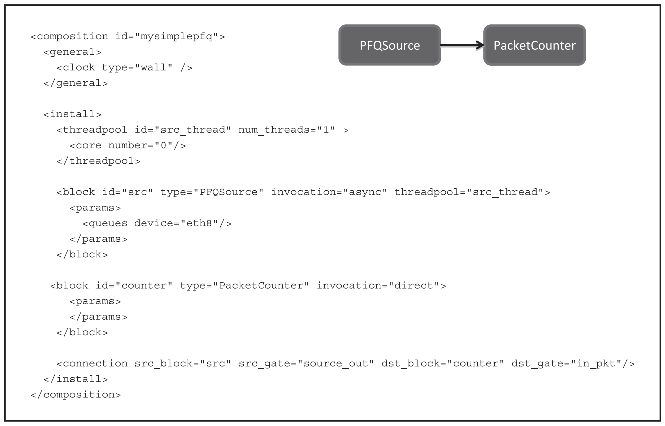
(2) User's Perspective: The Composition

To create a Blockmon application, users write a composition XML file and specify the blocks that compose the application, together with the inter-connection between them. **Fig. 3** outlines an XML composition file: a block that listens for packets coming to an interface and a block that counts those packets. While specifying the composition, Blockmon gives the user full control over the behavior of the blocks and their interaction in order to achieve the best performance. For instance, Blockmon allows blocks to be mapped to particular threads, and for these threads to be mapped to one or more CPU cores. In this way, users decide which cores run which bits of code, perhaps to optimize factors like cache effects, memory accesses, or CPU utilization.



```
class PacketCounter: public Block
{
 public:
  virtual void _configure(const xml_node&  n ) {};
  virtual void _receive_msg(std::shared_ptr<const Msg>&& m,
int index )
   {
    const Packet* p = static_cast<const Packet*>(m.get());
    ++m_count;
    m_byte_count += p->length();
   }
 private:
};
REGISTER_BLOCK(PacketCounter,"PacketCounter");
```

Fig. 2 Developing a Blockmon block.



```
<composition id="mysimplepfq">
  <general>
    <clock type="wall" />
  </general>

  <install>
    <threadpool id="src_thread" num_threads="1" >
      <core number="0"/>
    </threadpool>

    <block id="src" type="PFQSource" invocation="async" threadpool="src_thread">
      <params>
        <queues device="eth8"/>
      </params>
    </block>

   <block id="counter" type="PacketCounter" invocation="direct">
      <params>
      </params>
    </block>

    <connection src_block="src" src_gate="source_out" dst_block="counter" dst_gate="in_pkt"/>
  </install>
</composition>
```

Fig. 3 A Blockmon composition.

5. Use Case: VoIPSTREAM

To show the advantages of using Blockmon as a data analytics platform, we port an existing VoIP fraud detection application called VoIPSTREAM to it. VoIPSTREAM has proven to be an effective method for detecting potential telemarketers in real-time while protecting the privacy of normal (i.e., non-telemarketing) users [7] .

The reason for using VoIPSTREAM to evaluate the performance of Blockmon is two-fold. First, VoIPSTREAM works on textual data: this allows us to show the flexibility of Blockmon to deal with data other than network packets. Second, the results we show should be of interest to VoIP and phone operators.

6. VoIPSTREAM's Blockmon Design

At an abstract layer, the application can be split into three main parts: feature extraction, processing and reasoning; **Fig. 4** outlines the design of VoIPSTREAM.

The feature-extraction part reads from a set of call logs (or possibly also from live traffic) and gathers the features based on which the user behavior is analyzed.

The processing part exploits the features extracted from each

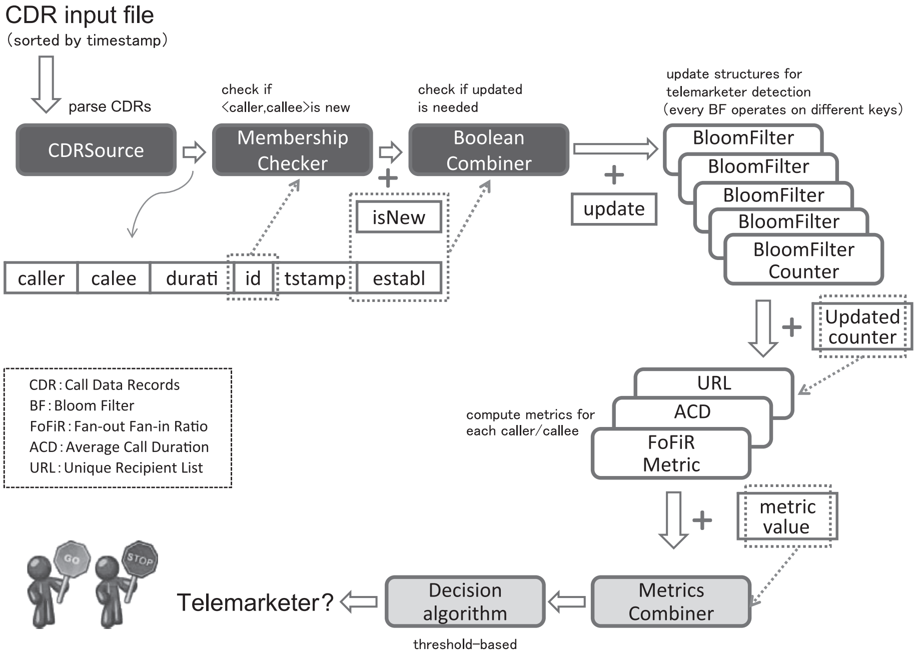


Fig. 4 Blockmon implementation of VoIPSTREAM: feature-extraction part is in red; processing part in blue; reasoning part in orange.

call to keep track of the overall behavior of each user within a time-window of configurable length. In VoIPSTREAM, the overall behavior of each user is derived by computing five quantities: per-user received call rate, per-user established outgoing call rate (over two different time-window lengths), per-user new callee rate, and per-user total outgoing call time. These quantities are computed by means of time-decaying Bloom filters, and then combined to compute three risk assessment modules over a time-window: FoFiR, ACD and URL. FoFiR is the ratio between number of outgoing and incoming established calls; ACD is the ratio between number of outgoing calls established towards distinct callees (new callees) and total number of established calls; URL is the ratio between global average call duration and user's average call duration. Using these ratios, each module returns an anomaly score between 0 and 1.

Finally, the reasoning part combines the risk assessment modules above to associate the user with an anomaly score: in our case, the combination is a weighted sum of the FoFiR, ACD and URL scores.

A threshold-based decision algorithm states whether the user is acting like a telemarketer in a given time-window based on the user's anomaly score.

Each part described above is implemented in Blockmon by means of a series of blocks: blocks exchange messages to ensure that each block has the information needed to perform its task.

The parameters of each block and of each bloom filter are configurable and can be specified while creating the block through the composition file. For instance, it is possible to specify the size of the hash-table in each Bloom- filter block (on which the false positive ratio depends), or to specify the threshold of the decision algorithm, as well as the weights to assign to each risk assessment module in the reasoning part.

7. Results

We implemented each block in Fig. 4 and ran the application on an x86 server with eight-core 2.4GHz Xeon CPU cores and 24GB of RAM.

Standalone (i.e., non-Blockmon) VoIPSTREAM processes a file containing 10 million phone calls at a mean rate of 46. 9kCall/s over three trials. As shown in **Fig. 5** , VoIPSTREAM on Blockmon using two cores (one for decoding phone calls and one for the rest of the processing) resulted in a rate of 41.6 kCall/s. Simply by increasing the number of cores devoted to processing to four (one for decoding phone calls, and three for the stages in the feature pipeline, with the metric blocks directly invoked), the rate of calls processed jumps to 67.6kCall/s, an improvement of 44% over standalone VoIPSTREAM. By using eight cores, the rate of calls improves incrementally to 70.9kCall/s, an improvement of 51%. This demonstrates the performance advantages of porting standalone applications to Blockmon. It is worth pointing out that porting the original code to Blockmon took only two weeks.

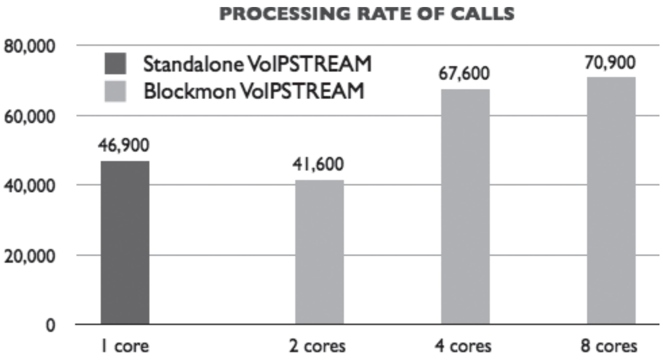


Fig. 5 VoIPSTREAM performance: standalone vs. Blockmon.

8. Conclusion and Future Use Cases

Blockmon is a flexible and high performance big data stream analytics platform. When used for network traffic analysis, it can cope with 10 Gb/s of network data, up to layer 7 (e.g., DPI), on a single commodity server. Used for server-log processing it can detect fraud in VoIP traffic at a rate of 70,000 cps (~250 million BHCA [*1]), enough to deal with all of Japan's phone traffic. We are now in the process of implementing a distributed version of Blockmon that scales performance out over a number of commodity servers; the objective of this work is to compare Blockmon with other platforms used for big data processing in different contexts (e.g., Yahoo! S4 and Twitter Storm). Further, this work will also implement, in addition to a distributed version of VoIPSTREAM and a content popularity estimation application, two web analytics applications that demonstrate Blockmon's flexibility and high performance: clickthrough rates (for measuring the success of an online ad-

[*1] Busy Hour Call Attempt

vertising campaign for a particular website) and hashtag counts (for detecting popular themes currently being discussed on Twitter).

9. Acknowledgment

This work was partially supported by DEMONS, a research project supported by the European Commission under its 7th Framework Program (contract no. 257315). The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the DEMONS project or the European Commission. The authors would like to thank all the people involved in Blockmon development activities.

*Yahoo! is a registered trademark or trademark of Yahoo! Inc.

*Twitter is a registered trademark of Twitter, Inc.

*Python is a registered trademark of Python Software Foundation.

*Intel and Xeon are trademarks of Intel Corporation in the U.S. and other countries.

References

1) CISCO SYSTEMS: "Cisco Visual Networking Index: Forecast and Methodology," http://www.cisco.com/en/US/netsol/ns827/networking_solutions_sub_solution.html, 2011.6.

2) BLOCKMON, the source code. https://github.com/blockmon/blockmon

3) G. Iannaccone: "Fast prototyping of network data mining applications," In Passive and Active Measurement Conference, 2006.

4) R. Morris, E. Kohler, J. Jannotti, M. F. Kaashoek: "The Click modular router," SIGOPS Oper. Syst. Rev. 33, 5 (1999), pp. 217 - 231.

5) L. Neumeyer et al.: "S4: Distributed Stream Computing Platform. Data Mining Workshops (ICDMW), 2010 IEEE International Conference on. http://ieeexplore.ieee.org/xpl/mostRecentIssue.jsp?punumber=5691154

6) Twitter Storm, the source code. https://github.com/nathanmarz/storm

7) G. Bianchi et al.: "On-demand time-decaying bloom filters for telemarketer detection," ACM SIGCOMM Computer Communication Review Volume 41, Number 5, pp. 6-12, October 2011. http://www.sigcomm.org/ccr/papers/2011/October/2043165.2043167

Authors' Profiles

Maurizio Dusi
Research Scientist
NEC Laboratories Europe
NEC Europe Ltd.

Nico d'Heureuse
Research Scientist
NEC Laboratories Europe
NEC Europe Ltd.

Felipe Huici
Senior Researcher
NEC Laboratories Europe
NEC Europe Ltd.

Andrea di Pietro
CNIT Research Assistant
Department of Information Engineering
University of Pisa

Nicola Bonelli
CNIT Research Assistant
Department of Information Engineering
University of Pisa

Giuseppe Bianchi
Full Professor
CNIT/ University of Roma Tor Vergata, Italy

Brian Trammell
Researcher
Communications Systems Group
Swiss Federal Institute of Technology Zurich.

Saverio Niccolini
Manager
NEC Laboratories Europe
NEC Europe Ltd.

Information about the NEC Technical Journal

Thank you for reading the paper.
If you are interested in the NEC Technical Journal, you can also read other papers on our website.

Link to NEC Technical Journal website

Japanese | English

Vol.7 No.2 Big Data

Vol.7 No.2

September, 2012

Special Issue TOP