SP4: Scalable Programmable Packet Processing Platform

Harjot Gill
University of Pennsylvania
Philadelphia, USA

Dong Lin
University of Pennsylvania
Philadelphia, USA

Lohit Sarna
University of Pennsylvania
Philadelphia, USA

Robert Mead
University of Pennsylvania
Philadelphia, USA

Kenton C.T. Lee
University of Pennsylvania
Philadelphia, USA

Boon Thau Loo
University of Pennsylvania
Philadelphia, USA

ABSTRACT

We propose the demonstration of SP4, a software-based programmable packet processing platform that supports (1) stateful packet processing useful for analyzing traffic flows with session semantics, (2) uses a task-stealing architecture that automatically leverages multi-core processing capabilities in a load-balanced manner without the need for explicit performance profiling, and (3) a declarative language for rapidly specifying and composing new packet processing functionalities from reusable modules. Our demonstration showcases the use of SP4 for performing high-throughput analysis of traffic traces for a variety of applications, such as filtering out unwanted traffic and detection of DDoS attacks using machine learning based analysis.

Categories and Subject Descriptors

C.2.1 [**Computer-Communication Networks**]: Network Architecture and Design

Keywords

Packet analysis, multicore, declarative networking

1. INTRODUCTION

As network applications become increasingly complex and heterogeneous, there is an increasing need for extensibility at the data plane, in order to carry out sophisticated functionalities such as traffic flow management, filtering out unwanted traffic, and content-based networking. Emerging software-defined networking platforms such as OpenFlow [6] provides extensibility by enabling packet processing at the software level. However, enabling such functionality in software requires significantly high bandwidth and compute capacity from servers.

One promising direction proposed recently is the use of multi-core machines to enable high-performance programmable software routers [3, 11], that can scale as the number of cores/machines increases. However, existing solutions either lack programming tools to rapidly customize different data plane functionalities, require explicit performance profiling to allocate cores to processing functionalities, or are not capable of performing complex operations beyond stateless per-packet processing, such as basic IP routing and packet encryption. To address these limitations, we present SP4 (stands for *Scalable Programmable Packet Processing Platform*), a software-based router platform that aims to provide data plane customizability and rapid deployment at high throughput and low latency.

We have developed a prototype of SP4 based on Rapidnet declarative networking engine [8], enhanced with task-stealing model of parallelism, and uses component-based dataflow architecture that allows for rapid assembly of packet processing functionalities. SP4 uses SP4LOG, a declarative language inspired by declarative networking [5] for constructing dataflows by composing existing data processing components. SP4LOG increases customizability through the use of high-level programming abstractions for composition. For example, our SMTP, VoIP and DDoS dataflows can be specified in only 7, 10 and 5 rules respectively.

SP4 integrates a threading library based on the task-stealing model [9], that enables us to enable fine-grained parallelism at the level of individual components in the dataflow, hence achieving both automatic load-balancing and high throughput processing. As opposed to naïve per-packet parallelism [3] which is only suitable for stateless workload (e.g. packet-routing, IPSec etc.), SP4 can process stateful multi-packet protocols (e.g. SIP) with high degree of parallelism. To ensure correct packet ordering in the presence of parallel processing, SP4 allows packets to be ordered based on its specified *context* attribute, defined in terms of application-specific semantics (e.g., a SIP session).

2. SYSTEM OVERVIEW

We provide an overview of SP4 by describing its dataflow pipeline. Refer to [4] for more details on SP4LOG.

SP4 uses a component-based dataflow architecture [7] that allows for rapid assembly of packet processing functionalities from reusable components into dataflow pipelines. These component modules are typically wrappers over existing code, that perform functionalities for various operations such as aggregating traffic statistics, or performing deep-packet inspections into specific application traffic.

The dataflow pipeline consists of several components connected in a linear chain, which are directly compiled from SP4LOG programs. Figure 1 shows an example dataflow pipeline that illustrates the execution model of SP4. SP4 supports three types of reusable components (e.g. elements), whose types are specified when the dataflow is constructed:

- **Serial.** Packets are processed in strict FIFO order. This is done for operations where total order is essential. In our example, `Packet Capture` is a serial component.
- **Parallel.** Incoming packets to a parallel component can be processed by several concurrent threads in a manner where ordering does not matter. For instance, once SMTP messages are assembled, regular expression matches on individual email mes-

SIGCOMM'12, August 13–17, 2012, Helsinki, Finland.
ACM 978-1-4503-1419-0/12/08.

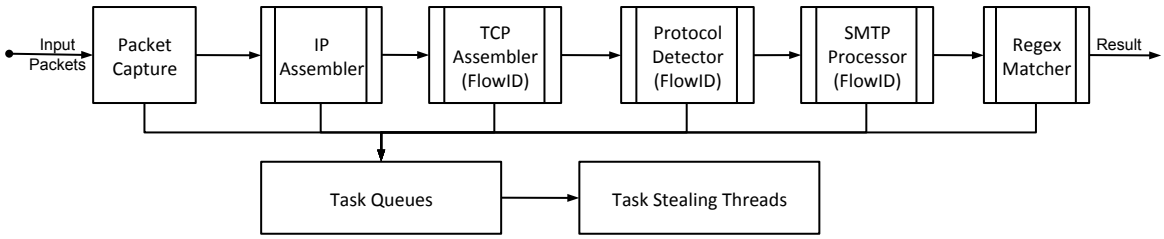

Figure 1: SP4 Dataflow pipeline example based on SMTP analysis. Parallel components have double lines, and context-ordered elements additionally have their context-keys shown in (...).

sages can be parallelized and processed in arbitrary order. `IP Assembler` and `Regex Match` are parallel components.

- **Parallel context-ordered.** These components are processed in partial order. A *context-key* is specified, in which all packets with the same key should be processed in order of their arrival into the system. But the ordering of packets with different keys is not required. For instance, `TCP Assembler` have to assemble message in a partial order (based on TCP flows). `Protocol Detector` and `SMTP Processor` are also context-ordered components.

3. DEMONSTRATION PLAN

Our demonstration will showcase dataflow parallelization techniques used in SP4 to support automatic load-balancing and fine-grained parallelism, and present concrete use-cases of the system for high-throughput analysis of network traces. Our demonstration consists of the following use cases:

SMTP analysis. As our first use-case, we will demonstrate a naive regex based data-leak detection system that monitors SMTP traffic. The dataflow is shown in Figure 1. The setup consists of SMTP protocol based email traffic generator, a network tap and a SP4 node. To emulate real network traffic, we will use an email generator written in JavaMail API that sends emails with preconfigured content to a mail server. A SP4 node deployed on a linux machine will execute a dataflow for analyzing the SMTP traffic. Specifically, this dataflow will assemble TCP flows, detect SMTP protocol, assemble SMTP messages and red-flag if message matches customizable regex based rules.

VoIP call interception. As our second use-case, we will demonstrate VoIP traffic capture of a live SIP call. Our experimental setup consists of SIP clients (running on Android smartphones), a network tap and a SP4 node. To emulate real network traffic load, we will use SIPp traffic generator tool[10] in addition to real SIP clients. The workflow will monitor and track all active SIP sessions based on preconfigured interception rules. The RTP stream interceptor component will decode and dump the RTP stream to local disk if the corresponding SIP session is matched by the SIP transaction processor component. Finally the dumped stream will be available for playback as sound media, post session termination. SP4 will maintain call-states while dynamically load-balancing the processing across threads.

DDoS attack detection. Our third use-case evaluates the performance and reliability of SP4 in detecting Distributed Denial of Service Attack (DDoS) patterns by using computationally intensive Support Vector Machine algorithm (SVM), a popular general-purpose algorithm used for machine learning. The dataflow includes two additional components for feature extraction and SVM classification. Our experimental setup consists of a DDoS traffic generator and a SP4 node. The SVM Classification module is implemented using libsvm [1], and is trained outside of SP4 on DARPA intrusion detection dataset [2] and PREDICT dataset. In our demonstration, SP4 node will tap network traffic generated by a DDoS trace-driven traffic generator, extract and update features that are significant for machine learning for each active connection, and red-flag potential DDoS incidents if the set of feature value is classified as positive by SVM classification component.

In all use cases, we will demonstrate the speedup and throughput obtained by SP4 by running the system on multiple quad-core laptops deployed at the conference venue. If network connectivity allows, we will also connect to powerful desktop machines at Penn with multiple cores (and hyperthreading enabled) to demonstrate throughput and speedup capabilities of SP4.

Acknowledgements. This project is supported in part by NSF grants CCF-0820208, IIS-0812270, CNS-0845552, and CNS-1040-672, and the DARPA SAFER award N66001-C-4020.

4. REFERENCES

[1] C.-C. Chang and C.-J. Lin. LIBSVM: A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology*, 2:27:1–27:27, 2011. Software available at `http://www.csie.ntu.edu.tw/~cjlin/libsvm`.

[2] DARPA Intrusion Detection Data Sets. `https://www.predict.org/`.

[3] M. Dobrescu, N. Egi, K. Argyraki, B.-G. Chun, K. Fall, et al. Routebricks: Exploiting parallelism to scale software routers. In *SOSP*, Oct 2009.

[4] H. Gill, D. Lin, T. Kothari, and B. T. Loo. Declarative multicore programming of software-based stateful packet processing. In *Declarative Aspects and Applications of Multicore Programming*. `netdb.cis.upenn.edu/papers/sp4_damp12.pdf`.

[5] B. T. Loo, T. Condie, M. Garofalakis, D. E. Gay, J. M. Hellerstein, P. Maniatis, R. Ramakrishnan, T. Roscoe, and I. Stoica. Declarative Networking. *CACM*, 2009.

[6] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, and J. Rexford. Openflow: Enabling innovation in campus networks. In *ACM SIGCOMM*, April 2008.

[7] R. Morris, E. Kohler, J. Jannotti, and M. F. Kaashoek. The click modular router. In *ACM Transactions on Computer Systems*, Aug 2000.

[8] RapidNet: A Declarative Toolkit for Rapid Network Simulation and Experimentation. `http://netdb.cis.upenn.edu/rapidnet/`.

[9] J. Reinders. Intel thread building blocks. In *OâĂŹReilly Associates*, 2007.

[10] SIPp Open Source test tool / traffic generator for the SIP protocol. `http://sipp.sourceforge.net/`.

[11] T. Wolf, N. Weng, and C.-H. Tai. Runtime support for multicore packet processing systems. *Network, IEEE*, 21(4):29 –37, July-August 2007.