Topic preview session 4:

SDN and Workloads

Steve Uhlig (QMUL)

ACM SIGCOMM, August 22, 2018

Talk objectives

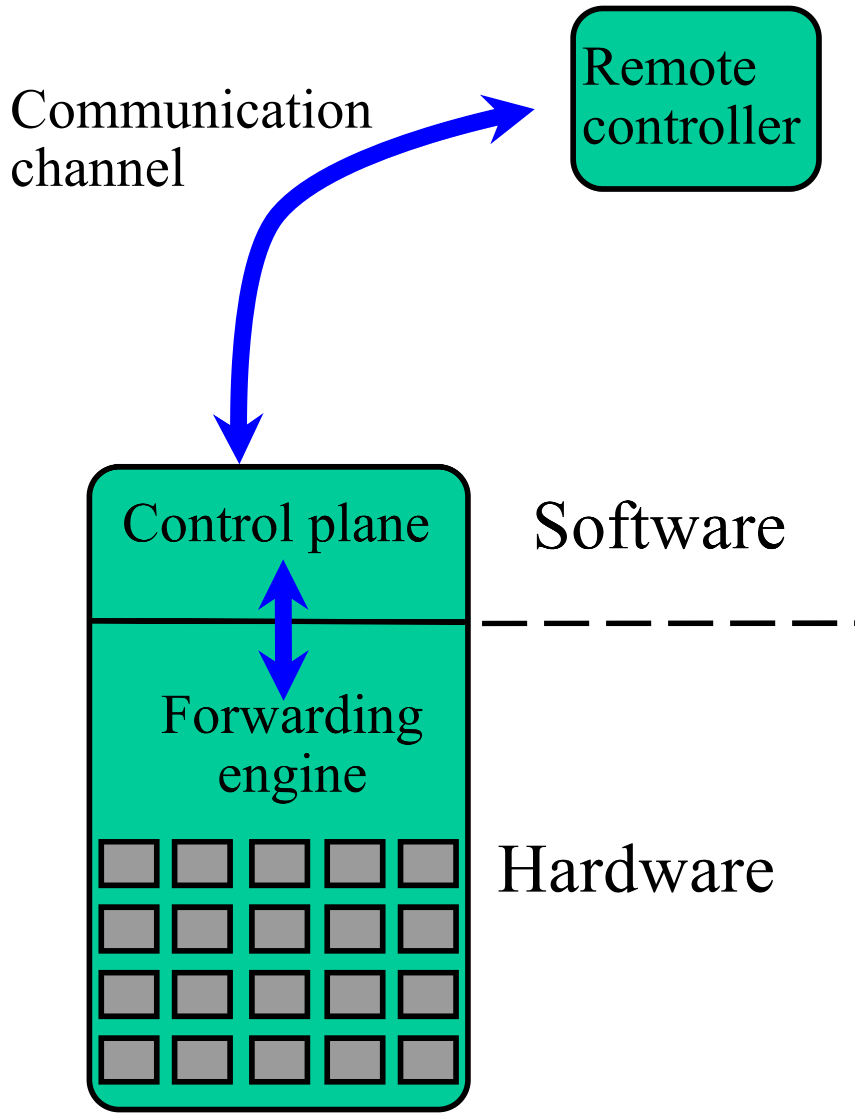
1. Introduce you to SDN
2. Provide context for session, not introduce/explain papers
3. Provide some pointers to literature

Agenda

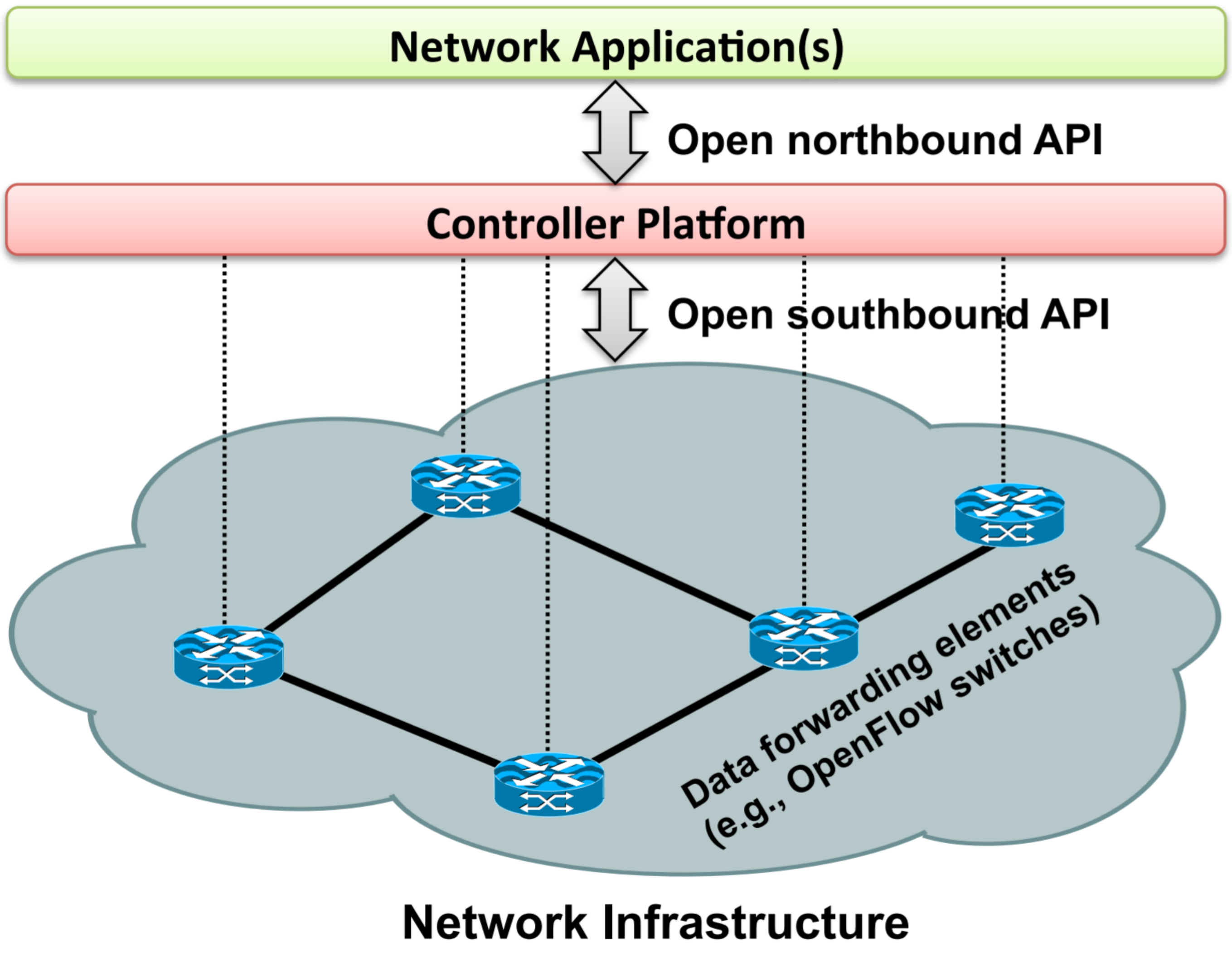
- **SDN primer**
- SDN: open problems
- SDN: applications
- Further reading

SDN: box view

- Beyond today' s monolithic network equipment
- Separation of control and data plane through software modularity, e.g., Linux
- Do not change existing control plane
- Principles
  - Communication channel between forwarding engine and remote controller
  - Expose network equipment capabilities, e.g., TCAM, QoS



SDN: network view



SDN: Benefits

- Uniform API of managing the network from a central software-based controller, e.g., [Trident]
- Extends virtualization techniques at multiple layers, e.g., [FlowVisor]
- Shift from *network hacks* to *applications*, e.g., traffic engineering, network monitoring and security

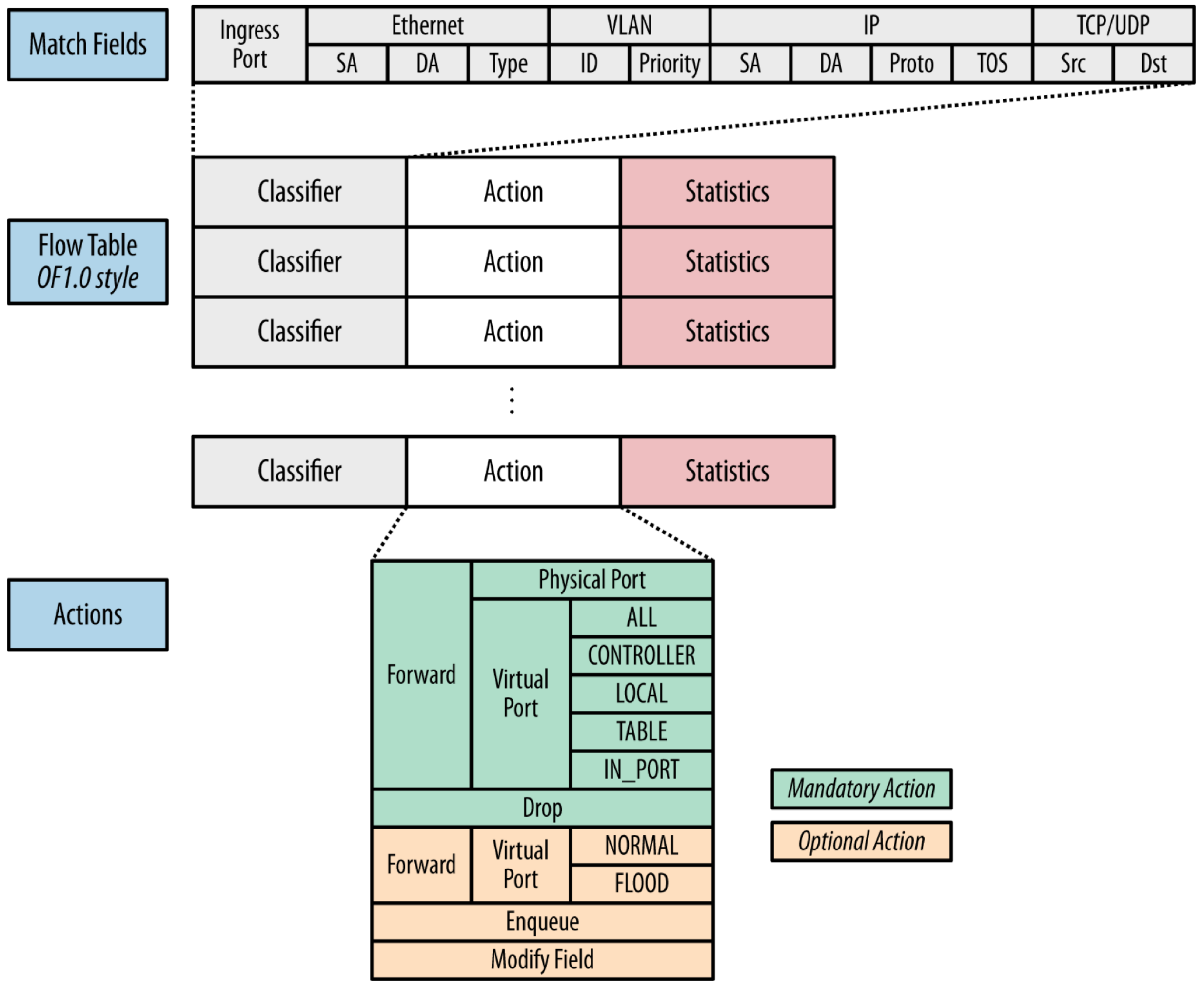
What is OpenFlow?

- The dominant realization of SDN
- Set of protocols and an API:
  - Wire protocol (OF):
    - Establish control sessions
    - Structure of flow modification messages
    - Statistics collection
    - Switch structure (ports and tables)
  - Configuration/management protocol (OF-config):
    - Based on NETCONF
    - Allocate switch ports to controller
    - Availability and failure behavior

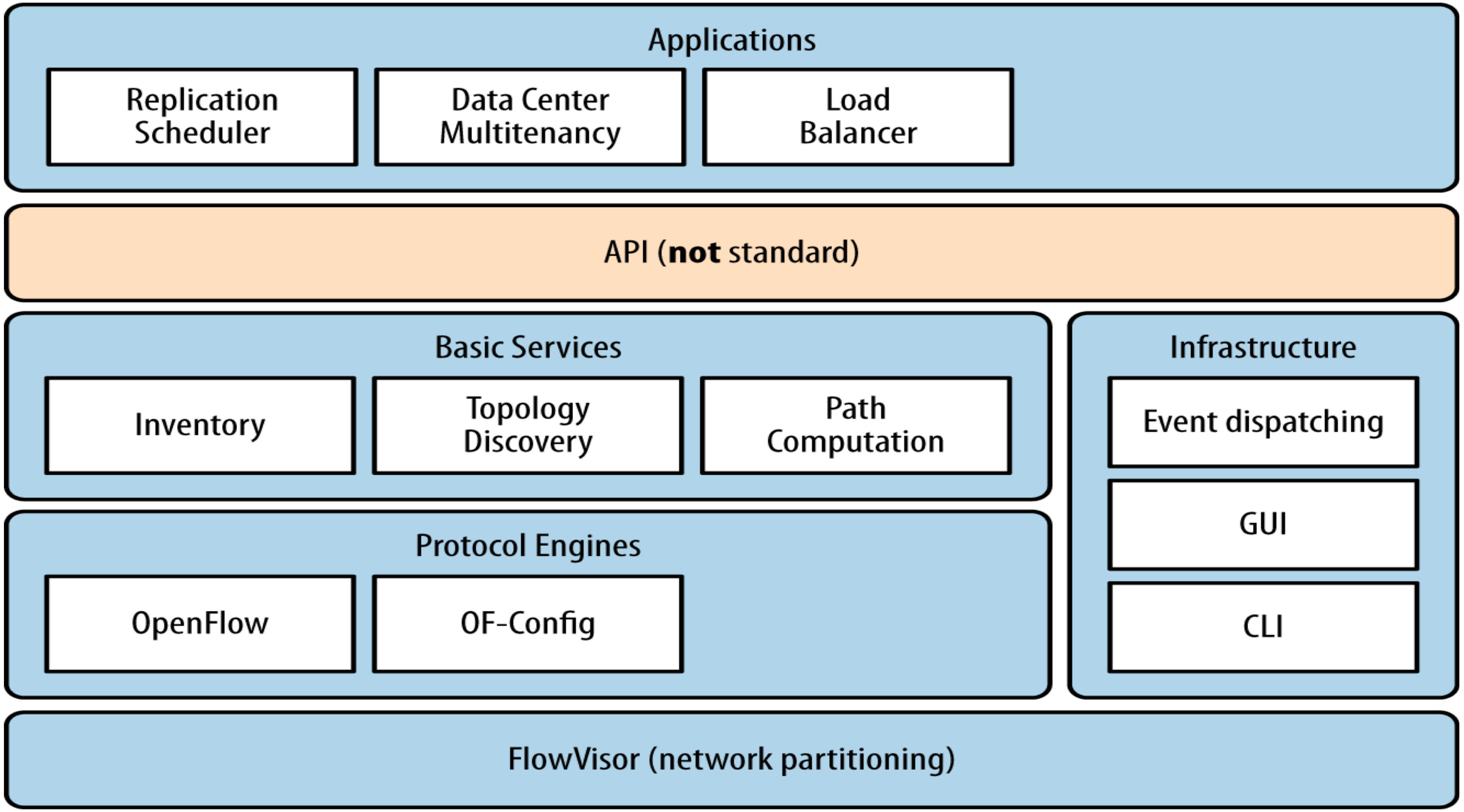
OpenFlow 1.0

Principle: match-action-(monitor)

- Match headers: defines flows
- Act on packets
- Monitor: counters



OpenFlow controller architecture

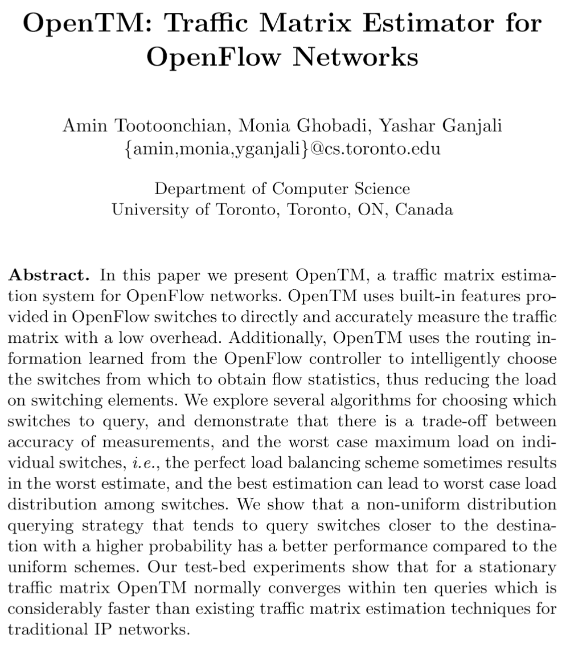


Agenda

- SDN primer
- **SDN: open issues**
- SDN: applications
- Further reading

Open issues: monitoring

- Issue: collect and analyse traffic at scale and in real-time, e.g., [OpenTM]
- Monitoring also called "telemetry", see tomorrow's session [Sonata, Snapshot]

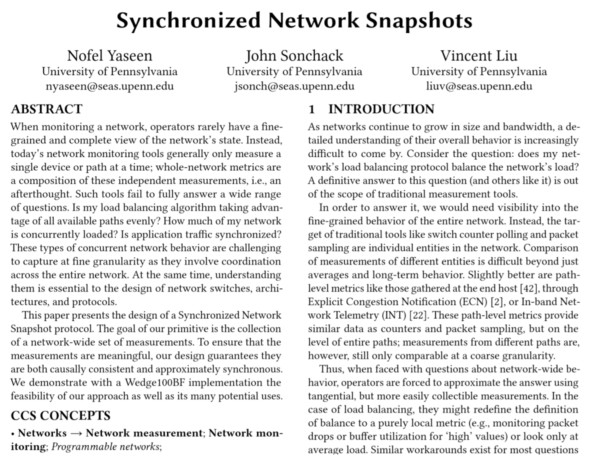


OpenTM: Traffic Matrix Estimator for OpenFlow Networks

Amin Tootoonchian, Monia Ghobadi, Yashar Ganjali
{amin,monia,yganjali}@cs.toronto.edu

Department of Computer Science
University of Toronto, Toronto, ON, Canada

Abstract. In this paper we present OpenTM, a traffic matrix estimation system for OpenFlow networks. OpenTM uses built-in features provided in OpenFlow switches to directly and accurately measure the traffic matrix with a low overhead. Additionally, OpenTM uses the routing information learned from the OpenFlow controller to intelligently choose the switches from which to obtain flow statistics, thus reducing the load on switching elements. We explore several algorithms for choosing which switches to query, and demonstrate that there is a trade-off between accuracy of measurements, and the worst case maximum load on individual switches, *i.e.*, the perfect load balancing scheme sometimes results in the worst estimate, and the best estimation can lead to worst case load distribution among switches. We show that a non-uniform distribution querying strategy that tends to query switches closer to the destination with a higher probability has a better performance compared to the uniform schemes. Our test-bed experiments show that for a stationary traffic matrix OpenTM normally converges within ten queries which is considerably faster than existing traffic matrix estimation techniques for traditional IP networks.



Synchronized Network Snapshots

Nofel Yaseen
University of Pennsylvania
nyaseen@seas.upenn.edu

John Sonchack
University of Pennsylvania
jsonch@seas.upenn.edu

Vincent Liu
University of Pennsylvania
liuv@seas.upenn.edu

ABSTRACT

When monitoring a network, operators rarely have a fine-grained and complete view of the network's state. Instead, today's network monitoring tools generally only measure a single device or path at a time; whole-network metrics are a composition of these independent measurements, i.e., an afterthought. Such tools fail to fully answer a wide range of questions. Is my load balancing algorithm taking advantage of all available paths evenly? How much of my network is concurrently loaded? Is application traffic synchronized? These types of concurrent network behavior are challenging to capture at fine granularity as they involve coordination across the entire network. At the same time, understanding them is essential to the design of network switches, architectures, and protocols.

This paper presents the design of a Synchronized Network Snapshot protocol. The goal of our primitive is the collection of a network-wide set of measurements. To ensure that the measurements are meaningful, our design guarantees they are both causally consistent and approximately synchronous. We demonstrate with a Wedge100BF implementation the feasibility of our approach as well as its many potential uses.

CCS CONCEPTS

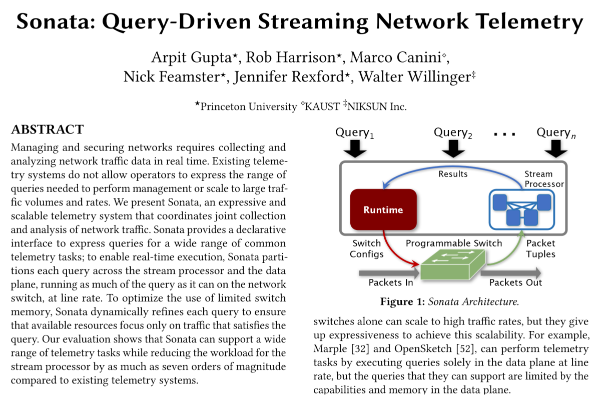
• **Networks** → **Network measurement**; **Network monitoring**; *Programmable networks*;

1 INTRODUCTION

As networks continue to grow in size and bandwidth, a detailed understanding of their overall behavior is increasingly difficult to come by. Consider the question: does my network's load balancing protocol balance the network's load? A definitive answer to this question (and others like it) is out of the scope of traditional measurement tools.

In order to answer it, we would need visibility into the fine-grained behavior of the entire network. Instead, the target of traditional tools like switch counter polling and packet sampling are individual entities in the network. Comparison of measurements of different entities is difficult beyond just averages and long-term behavior. Slightly better are path-level metrics like those gathered at the end host [42], through Explicit Congestion Notification (ECN) [2], or In-band Network Telemetry (INT) [22]. These path-level metrics provide similar data as counters and packet sampling, but on the level of entire paths; measurements from different paths are, however, still only comparable at a coarse granularity.

Thus, when faced with questions about network-wide behavior, operators are forced to approximate the answer using tangential, but more easily collectible measurements. In the case of load balancing, they might redefine the definition of balance to a purely local metric (e.g., monitoring packet drops or buffer utilization for 'high' values) or look only at average load. Similar workarounds exist for most questions



Sonata: Query-Driven Streaming Network Telemetry

Arpit Gupta[⋆], Rob Harrison[⋆], Marco Canini[⋄],
Nick Feamster[⋆], Jennifer Rexford[⋆], Walter Willinger[‡]

[⋆]Princeton University [⋄]KAUST [‡]NIKSUN Inc.

ABSTRACT

Managing and securing networks requires collecting and analyzing network traffic data in real time. Existing telemetry systems do not allow operators to express the range of queries needed to perform management or scale to large traffic volumes and rates. We present Sonata, an expressive and scalable telemetry system that coordinates joint collection and analysis of network traffic. Sonata provides a declarative interface to express queries for a wide range of common telemetry tasks; to enable real-time execution, Sonata partitions each query across the stream processor and the data plane, running as much of the query as it can on the network switch, at line rate. To optimize the use of limited switch memory, Sonata dynamically refines each query to ensure that available resources focus only on traffic that satisfies the query. Our evaluation shows that Sonata can support a wide range of telemetry tasks while reducing the workload for the stream processor by as much as seven orders of magnitude compared to existing telemetry systems.

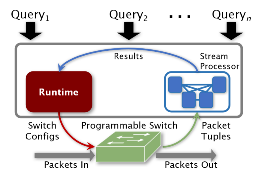


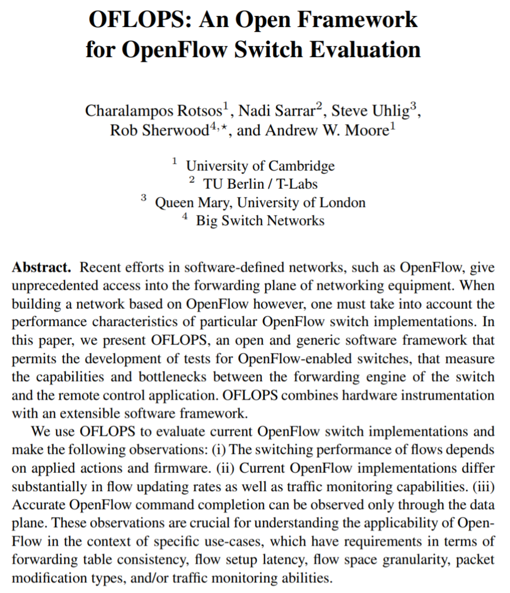
Figure 1: *Sonata Architecture.*

switches alone can scale to high traffic rates, but they give up expressiveness to achieve this scalability. For example, Marple [32] and OpenSketch [52], can perform telemetry tasks by executing queries solely in the data plane at line rate, but the queries that they can support are limited by the capabilities and memory in the data plane.

Open issues: control/data plane behavior

Issue: proper behaviour despite decoupling of control and data plane

- Goal: Control/data plane congruence [Consistency]
- Performance/overhead, e.g., [DevoFlow]
- Testing and troubleshooting, e.g., [OFLOPS,CASTAN]

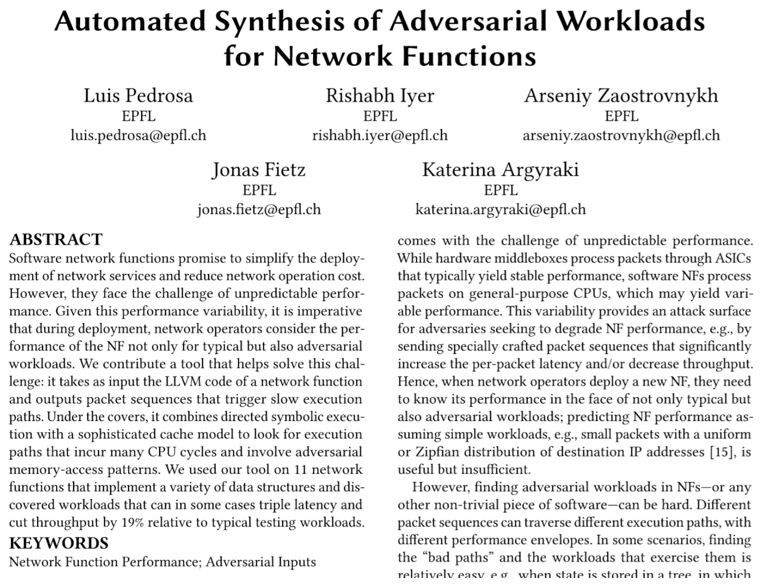

OFLOPS: An Open Framework for OpenFlow Switch Evaluation

Charalampos Rotsos[1], Nadi Sarrar[2], Steve Uhlig[3], Rob Sherwood[4,⋆], and Andrew W. Moore[1]

[1] University of Cambridge
[2] TU Berlin / T-Labs
[3] Queen Mary, University of London
[4] Big Switch Networks

Abstract. Recent efforts in software-defined networks, such as OpenFlow, give unprecedented access into the forwarding plane of networking equipment. When building a network based on OpenFlow however, one must take into account the performance characteristics of particular OpenFlow switch implementations. In this paper, we present OFLOPS, an open and generic software framework that permits the development of tests for OpenFlow-enabled switches, that measure the capabilities and bottlenecks between the forwarding engine of the switch and the remote control application. OFLOPS combines hardware instrumentation with an extensible software framework.

We use OFLOPS to evaluate current OpenFlow switch implementations and make the following observations: (i) The switching performance of flows depends on applied actions and firmware. (ii) Current OpenFlow implementations differ substantially in flow updating rates as well as traffic monitoring capabilities. (iii) Accurate OpenFlow command completion can be observed only through the data plane. These observations are crucial for understanding the applicability of OpenFlow in the context of specific use-cases, which have requirements in terms of forwarding table consistency, flow setup latency, flow space granularity, packet modification types, and/or traffic monitoring abilities.


Automated Synthesis of Adversarial Workloads for Network Functions

Luis Pedrosa, EPFL, luis.pedrosa@epfl.ch
Rishabh Iyer, EPFL, rishabh.iyer@epfl.ch
Arseniy Zaostrovnykh, EPFL, arseniy.zaostrovnykh@epfl.ch
Jonas Fietz, EPFL, jonas.fietz@epfl.ch
Katerina Argyraki, EPFL, katerina.argyraki@epfl.ch

ABSTRACT
Software network functions promise to simplify the deployment of network services and reduce network operation cost. However, they face the challenge of unpredictable performance. Given this performance variability, it is imperative that during deployment, network operators consider the performance of the NF not only for typical but also adversarial workloads. We contribute a tool that helps solve this challenge: it takes as input the LLVM code of a network function and outputs packet sequences that trigger slow execution paths. Under the covers, it combines directed symbolic execution with a sophisticated cache model to look for execution paths that incur many CPU cycles and involve adversarial memory-access patterns. We used our tool on 11 network functions that implement a variety of data structures and discovered workloads that can in some cases triple latency and cut throughput by 19% relative to typical testing workloads.

KEYWORDS
Network Function Performance; Adversarial Inputs

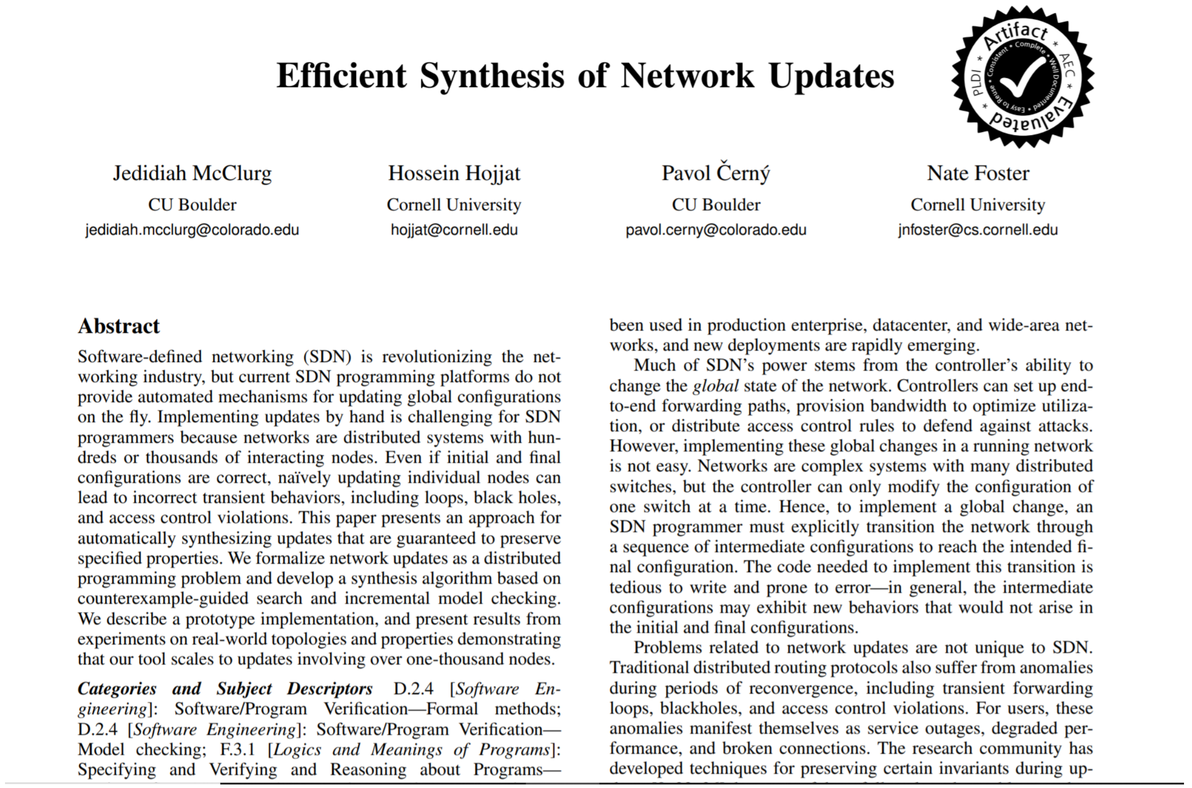
comes with the challenge of unpredictable performance. While hardware middleboxes process packets through ASICs that typically yield stable performance, software NFs process packets on general-purpose CPUs, which may yield variable performance. This variability provides an attack surface for adversaries seeking to degrade NF performance, e.g., by sending specially crafted packet sequences that significantly increase the per-packet latency and/or decrease throughput. Hence, when network operators deploy a new NF, they need to know its performance in the face of not only typical but also adversarial workloads; predicting NF performance assuming simple workloads, e.g., small packets with a uniform or Zipfian distribution of destination IP addresses [15], is useful but insufficient.

However, finding adversarial workloads in NFs—or any other non-trivial piece of software—can be hard. Different packet sequences can traverse different execution paths, with different performance envelopes. In some scenarios, finding the "bad paths" and the workloads that exercise them is relatively easy, e.g., when state is stored in a tree, in which

Open issues: network updates

Issue: handling network updates, e.g., [PLDI'15], while guaranteeing specific properties

- Correctness
- Consistency
- Ordering
- Data plane



Efficient Synthesis of Network Updates

Jedidiah McClurg, CU Boulder, jedidiah.mcclurg@colorado.edu
Hossein Hojjat, Cornell University, hojjat@cornell.edu
Pavol Černý, CU Boulder, pavol.cerny@colorado.edu
Nate Foster, Cornell University, jnfoster@cs.cornell.edu

Abstract

Software-defined networking (SDN) is revolutionizing the networking industry, but current SDN programming platforms do not provide automated mechanisms for updating global configurations on the fly. Implementing updates by hand is challenging for SDN programmers because networks are distributed systems with hundreds or thousands of interacting nodes. Even if initial and final configurations are correct, naïvely updating individual nodes can lead to incorrect transient behaviors, including loops, black holes, and access control violations. This paper presents an approach for automatically synthesizing updates that are guaranteed to preserve specified properties. We formalize network updates as a distributed programming problem and develop a synthesis algorithm based on counterexample-guided search and incremental model checking. We describe a prototype implementation, and present results from experiments on real-world topologies and properties demonstrating that our tool scales to updates involving over one-thousand nodes.

Categories and Subject Descriptors D.2.4 [*Software Engineering*]: Software/Program Verification—Formal methods; D.2.4 [*Software Engineering*]: Software/Program Verification—Model checking; F.3.1 [*Logics and Meanings of Programs*]: Specifying and Verifying and Reasoning about Programs—

been used in production enterprise, datacenter, and wide-area networks, and new deployments are rapidly emerging.

Much of SDN's power stems from the controller's ability to change the *global* state of the network. Controllers can set up end-to-end forwarding paths, provision bandwidth to optimize utilization, or distribute access control rules to defend against attacks. However, implementing these global changes in a running network is not easy. Networks are complex systems with many distributed switches, but the controller can only modify the configuration of one switch at a time. Hence, to implement a global change, an SDN programmer must explicitly transition the network through a sequence of intermediate configurations to reach the intended final configuration. The code needed to implement this transition is tedious to write and prone to error—in general, the intermediate configurations may exhibit new behaviors that would not arise in the initial and final configurations.

Problems related to network updates are not unique to SDN. Traditional distributed routing protocols also suffer from anomalies during periods of reconvergence, including transient forwarding loops, blackholes, and access control violations. For users, these anomalies manifest themselves as service outages, degraded performance, and broken connections. The research community has developed techniques for preserving certain invariants during up-

Open issues: security

Issue: several points to be secured against attacks, e.g., [Secsurvey, SecSDN]

- Controller
- Communication channels: southbound (network) and northbound (applications)
- Virtual planes, e.g., [FlowVisor]
- Data plane

See "Network Verification" session

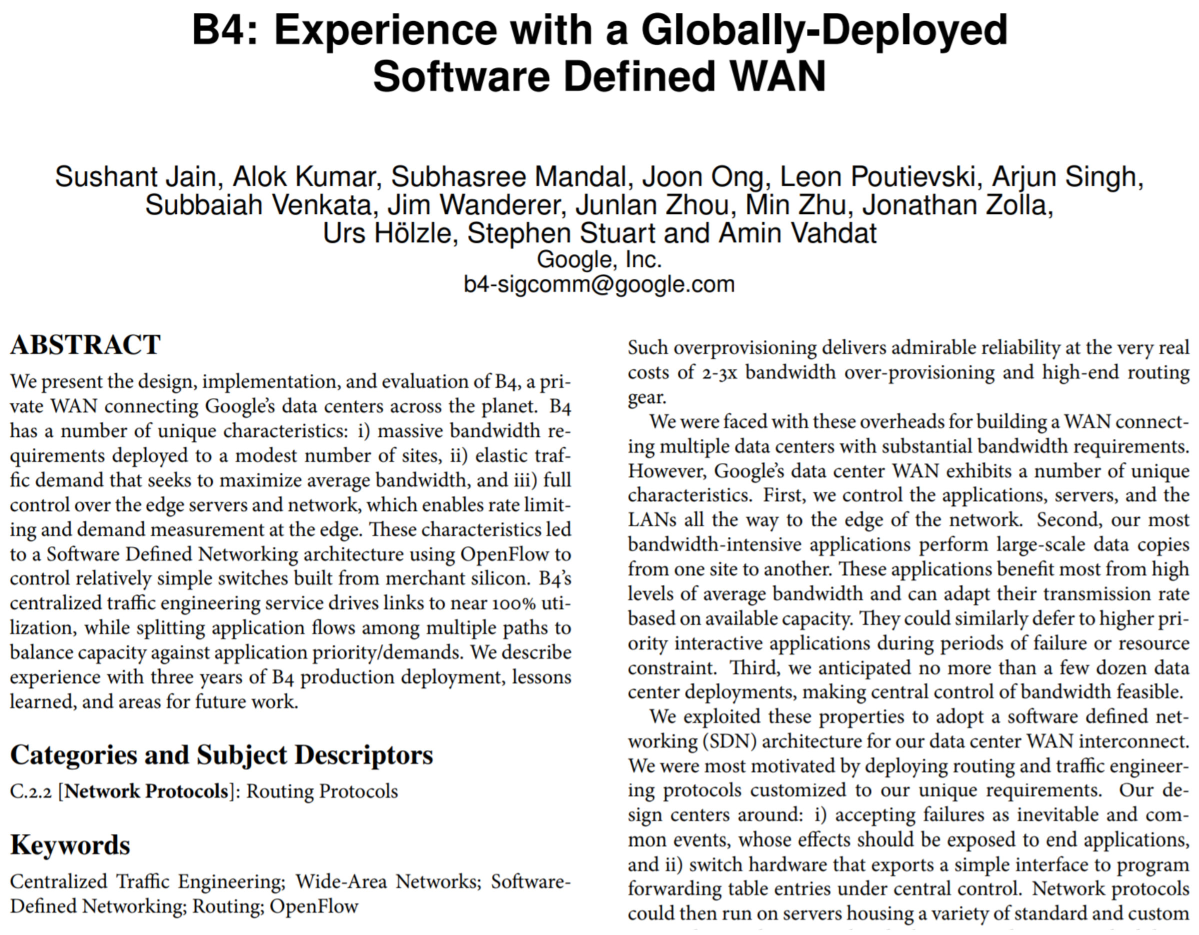
Agenda

Queen Mary
University of London

- SDN primer
- SDN: open issues
- **SDN: applications**
- Further reading

Applications: traffic engineering

- Using SDN to steer traffic inside a multi-site network, e.g., [B4]



B4: Experience with a Globally-Deployed Software Defined WAN

Sushant Jain, Alok Kumar, Subhasree Mandal, Joon Ong, Leon Poutievski, Arjun Singh, Subbaiah Venkata, Jim Wanderer, Junlan Zhou, Min Zhu, Jonathan Zolla, Urs Hölzle, Stephen Stuart and Amin Vahdat
Google, Inc.
b4-sigcomm@google.com

ABSTRACT

We present the design, implementation, and evaluation of B4, a private WAN connecting Google's data centers across the planet. B4 has a number of unique characteristics: i) massive bandwidth requirements deployed to a modest number of sites, ii) elastic traffic demand that seeks to maximize average bandwidth, and iii) full control over the edge servers and network, which enables rate limiting and demand measurement at the edge. These characteristics led to a Software Defined Networking architecture using OpenFlow to control relatively simple switches built from merchant silicon. B4's centralized traffic engineering service drives links to near 100% utilization, while splitting application flows among multiple paths to balance capacity against application priority/demands. We describe experience with three years of B4 production deployment, lessons learned, and areas for future work.

Categories and Subject Descriptors

C.2.2 [**Network Protocols**]: Routing Protocols

Keywords

Centralized Traffic Engineering; Wide-Area Networks; Software-Defined Networking; Routing; OpenFlow

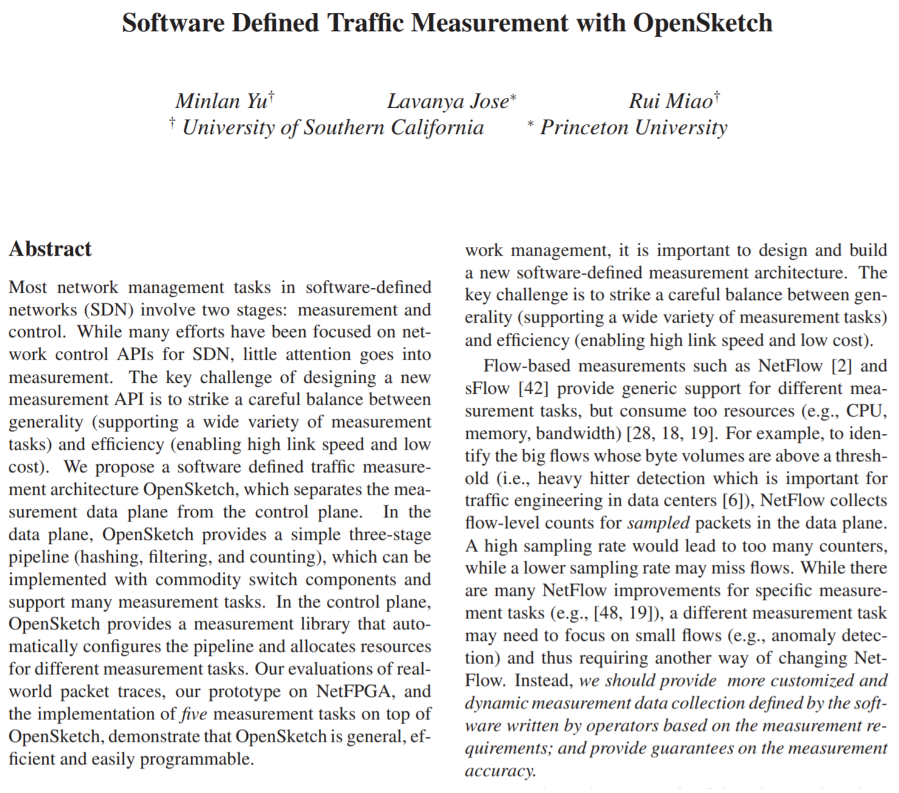
Such overprovisioning delivers admirable reliability at the very real costs of 2-3x bandwidth over-provisioning and high-end routing gear.

We were faced with these overheads for building a WAN connecting multiple data centers with substantial bandwidth requirements. However, Google's data center WAN exhibits a number of unique characteristics. First, we control the applications, servers, and the LANs all the way to the edge of the network. Second, our most bandwidth-intensive applications perform large-scale data copies from one site to another. These applications benefit most from high levels of average bandwidth and can adapt their transmission rate based on available capacity. They could similarly defer to higher priority interactive applications during periods of failure or resource constraint. Third, we anticipated no more than a few dozen data center deployments, making central control of bandwidth feasible.

We exploited these properties to adopt a software defined networking (SDN) architecture for our data center WAN interconnect. We were most motivated by deploying routing and traffic engineering protocols customized to our unique requirements. Our design centers around: i) accepting failures as inevitable and common events, whose effects should be exposed to end applications, and ii) switch hardware that exports a simple interface to program forwarding table entries under central control. Network protocols could then run on servers housing a variety of standard and custom

Applications: monitoring

- Requires advanced data structures, e.g., Sketches ([OpenSketch] and [ElasticSketch])



Software Defined Traffic Measurement with OpenSketch

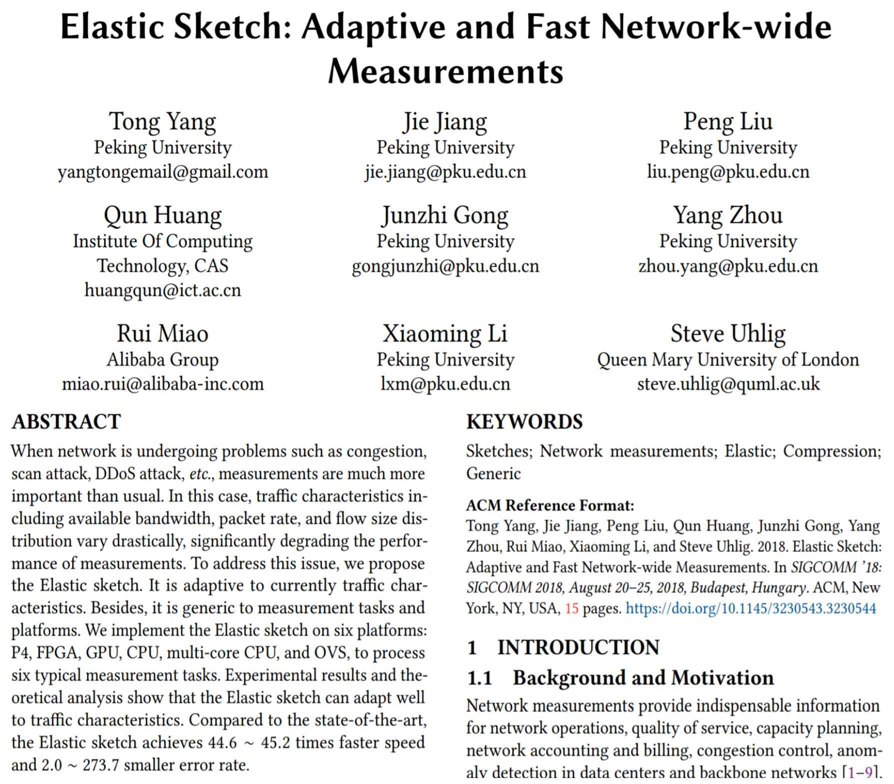
Minlan Yu[†] Lavanya Jose[*] Rui Miao[†]
[†] University of Southern California [*] Princeton University

Abstract

Most network management tasks in software-defined networks (SDN) involve two stages: measurement and control. While many efforts have been focused on network control APIs for SDN, little attention goes into measurement. The key challenge of designing a new measurement API is to strike a careful balance between generality (supporting a wide variety of measurement tasks) and efficiency (enabling high link speed and low cost). We propose a software defined traffic measurement architecture OpenSketch, which separates the measurement data plane from the control plane. In the data plane, OpenSketch provides a simple three-stage pipeline (hashing, filtering, and counting), which can be implemented with commodity switch components and support many measurement tasks. In the control plane, OpenSketch provides a measurement library that automatically configures the pipeline and allocates resources for different measurement tasks. Our evaluations of real-world packet traces, our prototype on NetFPGA, and the implementation of *five* measurement tasks on top of OpenSketch, demonstrate that OpenSketch is general, efficient and easily programmable.

work management, it is important to design and build a new software-defined measurement architecture. The key challenge is to strike a careful balance between generality (supporting a wide variety of measurement tasks) and efficiency (enabling high link speed and low cost).

Flow-based measurements such as NetFlow [2] and sFlow [42] provide generic support for different measurement tasks, but consume too resources (e.g., CPU, memory, bandwidth) [28, 18, 19]. For example, to identify the big flows whose byte volumes are above a threshold (i.e., heavy hitter detection which is important for traffic engineering in data centers [6]), NetFlow collects flow-level counts for *sampled* packets in the data plane. A high sampling rate would lead to too many counters, while a lower sampling rate may miss flows. While there are many NetFlow improvements for specific measurement tasks (e.g., [48, 19]), a different measurement task may need to focus on small flows (e.g., anomaly detection) and thus requiring another way of changing NetFlow. Instead, *we should provide more customized and dynamic measurement data collection defined by the software written by operators based on the measurement requirements; and provide guarantees on the measurement accuracy.*



Elastic Sketch: Adaptive and Fast Network-wide Measurements

Tong Yang
Peking University
yangtongemail@gmail.com

Jie Jiang
Peking University
jie.jiang@pku.edu.cn

Peng Liu
Peking University
liu.peng@pku.edu.cn

Qun Huang
Institute Of Computing Technology, CAS
huangqun@ict.ac.cn

Junzhi Gong
Peking University
gongjunzhi@pku.edu.cn

Yang Zhou
Peking University
zhou.yang@pku.edu.cn

Rui Miao
Alibaba Group
miao.rui@alibaba-inc.com

Xiaoming Li
Peking University
lxm@pku.edu.cn

Steve Uhlig
Queen Mary University of London
steve.uhlig@quml.ac.uk

ABSTRACT

When network is undergoing problems such as congestion, scan attack, DDoS attack, *etc.*, measurements are much more important than usual. In this case, traffic characteristics including available bandwidth, packet rate, and flow size distribution vary drastically, significantly degrading the performance of measurements. To address this issue, we propose the Elastic sketch. It is adaptive to currently traffic characteristics. Besides, it is generic to measurement tasks and platforms. We implement the Elastic sketch on six platforms: P4, FPGA, GPU, CPU, multi-core CPU, and OVS, to process six typical measurement tasks. Experimental results and theoretical analysis show that the Elastic sketch can adapt well to traffic characteristics. Compared to the state-of-the-art, the Elastic sketch achieves 44.6 ~ 45.2 times faster speed and 2.0 ~ 273.7 smaller error rate.

KEYWORDS

Sketches; Network measurements; Elastic; Compression; Generic

ACM Reference Format:
Tong Yang, Jie Jiang, Peng Liu, Qun Huang, Junzhi Gong, Yang Zhou, Rui Miao, Xiaoming Li, and Steve Uhlig. 2018. Elastic Sketch: Adaptive and Fast Network-wide Measurements. In *SIGCOMM '18: SIGCOMM 2018, August 20–25, 2018, Budapest, Hungary*. ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/3230543.3230544

1 INTRODUCTION

1.1 Background and Motivation

Network measurements provide indispensable information for network operations, quality of service, capacity planning, network accounting and billing, congestion control, anomaly detection in data centers and backbone networks [1–9].

Agenda

- SDN primer
- SDN: open issues
- SDN: applications
- **Further reading**

Further reading

[Trident] Kai Gao et al. *Trident: Toward a Unified SDN Programming Framework with Automatic Updates*. Proc. of ACM SIGCOMM, 2018.

[FlowVisor] R. Sherwood et al. *Can the production network be the testbed?* Proc. of ACM OSDI, 2010.

[OpenTM] A. Tootoonchian et al. *OpenTM: traffic matrix estimator for OpenFlow networks*. Proc. of PAM, 2010.

[Sonata] A. Gupta et al. *Sonata: Query-Driven Streaming Network Telemetry*. Proc. of ACM SIGCOMM, 2018.

[Snapshot] N. Yaseen et al. *Synchronized Network Snapshots*. Proc. of ACM SIGCOMM, 2018.

[PLDI'15] J. McClurg et al. *Efficient synthesis of network updates*. Proc. of ACM PLDI, 2015.

[OFLOPS] C. Rotsos et al. *OFLOPS: an open framework for openflow switch evaluation*. Proc. of PAM, 2012.

Further reading

[CASTAN] L. Pedrosa et al. *Automated Synthesis of Adversarial Workloads for Network Functions*. Proc. of ACM SIGCOMM, 2018.

[B4] S. Jain et al. *B4: experience with a globally-deployed software defined WAN*. Proc. of ACM SIGCOMM, 2013.

[OpenSketch] M. Yu et al. *Software Defined Traffic Measurement with OpenSketch*. Proc. of USENIX NSDI, 2013.

[Elastic] T. Yang et al. *Elastic sketch: Adaptive and fast network-wide measurements*. Proc. ACM SIGCOMM, 2018.

[Consistency] P. Zhang et al. *Mind the Gap: Monitoring the Control-Data Plane Consistency in Software Defined Networks*. Proc. of ACM CoNEXT, 2016.

[DevoFlow] A. Curtis et al. *DevoFlow: scaling flow management for high-performance networks*. Proc. of ACM SIGCOMM, 2011.

Further reading

[Survey] D. Kreutz et al. *Software-defined networking: A comprehensive survey*. Proc. of the IEEE, 2015.

[Secsurvey] A. Akhunzada et al. *Securing software defined networks: taxonomy, requirements, and open issues*. IEEE Communications Magazine, 2015.

[SecSDN] S. Ali et al. *A Survey of Securing Networks Using Software Defined Networking*. IEEE Transactions on Reliability, 2015.