Composable Primitives for SDN Measurements

Paolo Laffranchini
Instituto Superior Técnico, Lisbon
paolo.laffranchini@tecnico.ulisboa.pt

ABSTRACT

Measurement is a crucial concern for network operation. Rather than a principled approach, we see that measurement in SDN has been mostly an ad-hoc activity with poor reuse of abstractions that are hard to combine efficiently.

We argue that network measurements should be supported with first-class artifacts able to address a wide range of measurements tasks in an efficient, flexible and reusable manner.

We have identified a set of primitives that can be easily combined to express well-known techniques and used them to solve three measurement problems that have not been addressed in the literature: monitoring path changes, measuring the latency to flow steering and identifying the top-k congested flows.

1 INTRODUCTION

The introduction of Software Defined Networking (SDN) with its separation of data and control planes has led to significant work on various aspects of programmable network infrastructures. Unfortunately, measurement, a well understood requirement of network management, appears to have been an afterthought in SDN's development.

Historically, measurement's evolution paralleled the growth of the Internet but at a much slower pace. SNMP, *ping*, *traceroute* and sampling constituted the bulk of measurement-related aids for a long time. These tools often lacked sufficient flexibility to provide precise information on low level aspects of the behavior of flows.

The first realization of SDN, OpenFlow[8] provided simple support to measurement activities. Forwarding rules are naturally associated with a counter of packets (and bytes) they match, altogether with a timestamp indicating the time when they were installed in the switch. Numerous research efforts have explored techniques exploit these data to answer a variety of measurement questions: throughput and packet loss[11], anomalies[13], traffic matrix[10] and heavy hitters[6]. However, three main drawbacks emerged. First, measurement data is tightly coupled to the forwarding policy decided by the controller, whereas measurement requirements do not always reflect this assumption. Second, limitations on the available amount of space in the switches' TCAM memory, introduce intrinsic memory/accuracy trade-offs[9], as the quality of the measurements result comes at the expense of an higher number of rules to be deployed on the data plane. Lastly, explicit data collection from the control plane led to high overhead on both controller(s) and switches (additional network traffic, CPU processing, and time)[4].

Subsequent research mitigated these drawbacks using approximation algorithms (*i.e.,* sketches and bloom filters) to keep short summaries of traffic haracteristics[12], with provable bounds on accuracy when assigned a certain amount of resources. But they had limited applicability and/or re-usability, and required special-purpose, hard-coded algorithms.

Recent trends on programmable dataplanes[3] allow the definition and implementation of arbitrary measurement algorithms[5, 7] that can be installed on protocol-independent switch architectures. However, it is not clear how programmable dataplanes can foster re-usability for arbitrary run-time measurement needs.

2 PROBLEM STATEMENT

Network measurement spans different areas often characterized by non-overlapping requirements: from network-wide traffic monitoring (*e.g.,* heavy hitters, traffic matrix), to fine-grained monitoring flow properties (*e.g.,* throughput, latency, loss), to verification (routing correctness *w.r.t.* operator's intent) and debugging (*e.g.,* troubleshooting network issues) and various aspects of security (*e.g.,* anomalies, DDoS).

State-of-the-art approaches adopt ad-hoc algorithms that are only able to cover a subset of these use cases. Many techniques overlap in intent and functionality but cannot be easily re-purposed to address different questions. Adaptation to a slightly different version of a problem require significant development effort due to the monolithic nature with which these solutions are realized. Some of the low-level operations exported by a switch (*e.g.,* counters) may be shared between different solutions, but their actual usage is dependent on the specifics of the algorithms themselves. This is due to the lack of high-level abstractions to flexibly express how a measurement is being realized, leading to hard-coded operations that cannot be reused efficiently. Furthermore, a thorough review of the existing literature revealed that a measurement problem can be approached using a variety of ad-hoc algorithms, that often optimize different objectives, and whose accuracy depends upon workload characteristics. These solutions are therefore only amenable under particular assumptions.

The absence of a principled approach with which to address measurement needs, burdens network operators with the need to understand many subtle nuances between different solutions to a problem. The exposure to low-level details, along with the need to manually craft algorithms for a large spectrum of scenarios, increase both deployment time and management costs, while possibly harming correctness due to the complexity of the tasks at hand.

Currently, no existing system is capable to tackle a comprehensive set of different measurement requirements from a higher-level perspective, while coping with ever-changing run-time needs.

With this PhD proposal, we are the first to address SDN measurements with a set of general-purpose, programmable building blocks, called primitives. Our approach is rooted on the identification of a set of primitives that are orthogonal and composable, able to express a broad range of measurement in a concise manner. These primitives will export an API that can be used to combine them and realize flexible measurements. The high-level composition of measurement tasks with our primitives will provide a service layer through which to manage and deploy measurements

Primitive	API	Purpose
Timestamp	*Timestamp(x)*	Timeouts, latency
Counter	*Counter(λ_u, c)*	#packets/bytes, sequence numbers
Bloom filter	*BloomFilter(key, λ_u, bf)*	Set membership
Sketch	*Sketch(key, λ_u, sketch)*	Approximate counting
Sample	*Duplicate("virtual_stream")* *Collect("collector")*	Data reduction, collection
Tag	*Tag(λ_u, header_field)*	Information piggybacking
Match	*Match(condition)*	Flow selection, triggers

Table 1: Measurement primitives & API.

in a dynamic fashion. Our goal is to propose a comprehensive SDN measurement architecture capable to optimize, distribute and coordinate network-wide measurement operations. Integration with the forwarding policies and coordination with the SDN controller will allow to exploit decision making strategies to maximize efficiency and reliability of measurement, while respecting resource constraints.

3 RESEARCH PLAN

The research plan is laid out as follows:

i) Identification of a suitable set of measurement primitives with which a P4-programmable data plane can be equipped to perform basic measurement operations. Definition of a general-purpose expressive API with which operators can program these primitives and combine them together to implement complex measurement techniques. Implementation of a compiler to translate an high-level measurements to a P4 switch pipeline configuration.

ii) Analyze and model the primitives' memory usage and execution cost. Limited memory resources and the tight time constraints to perform per-packet operations represent important objectives to assess the feasibility of data-plane measurement tasks.

iii) Explore network-wide distribution and coordination of measurement. Some measurements use cases cannot be performed on a singular network entity, due to the need for multiple points of observation. Moreover, placement of primitives is a core challenge to optimally deploy measurement tasks, considering as well the resource limitations discussed above.

iv) Integrate measurement in the control plane and define interactions with the data plane for run-time primitive configuration and tuning. Measurement activities are expected to be either short or long-lived depending on their nature. However, dynamic reconfiguration of programmable data planes algorithms expressed in P4 is still an open research question. As well, we plan to explore abstractions and protocols to model the interactions between the switches and controller.

v) Propose a comprehensive SDN system, integrating traffic control and measurement requirements. While we believe the data plane configuration for the two should be independent, an high degree of synergy is expected between measurement and control in SDN, due to its highly dynamic nature. Whereas forwarding decisions are affected by measurements, the latter need to adapt to maintain reliability and resource efficiency. Integration of these two aspects is a crucial concern for the management of SDN networks.

4 CURRENT STATUS

At the current state, we completed the first step of the plan described in Section 3. We approached the identification of the primitives for measurement on the basis of their *breadth* of applicability and the ability for *maximal reuse*. We argue that switches should provide support to perform these primitive measurement actions. A compiler translates a measurement, expressed via a composition of these primitives, into a P4 pipeline configuration to run on programmable data planes.

4.1 Measurement primitives

The core set of measurement primitives we have identified are the following: *timestamps*, *counters*, *samples*, *tags*, *bloom filters*, *sketches*, and *matching*. Each primitive exports an API through which operators can express measurement activities.

- ***Timestamps***: provide the ability to derive time-related information, essential to perform a variety of measurement tasks, such as detecting timeouts or estimating latency and packets inter-arrival times.
- ***Counters***: can be used to keep track of any measurable quantities such as bytes, packets, flows and sequence numbers.
- ***Bloom filters***: permit to compactly encode a group of elements, making them ideal to test set memberships. Counting variations can also be exploited to handle dynamic deletions.
- ***Sketches***: allow to collect approximate data using compact data structures providing provable accuracy when equipped with fixed memory resources. Useful to gather measurement data for an high number of independent flows with known error bounds. Typical usages are volume (amount of data) and cardinality (*e.g.,* number of flows) estimation.
- ***Samples***: permit to duplicate a packet and forwarding it to a collector component. Can be used to offload measurement algorithms outside of the data plane, if the switch resources are insufficient to execute all measurement operations.
- ***Tags***: allow to modify the content of packet header fields, enriching the information they carry with contextual information as the packet travels in the network.
- ***Matching***: tests conditions on the packet header fields and the status of the switch stateful memory. It's intended to detect specific packets or select groups of them (*e.g.,* coming from a specific source) so to apply the measurement operations to the target portion of the traffic. Conditionals on the value of the switch registers, holding the current state, serve the purpose to trigger events.

These primitives cope with different needs in measurements: i) store state, either with a 1-to-1 mapping from flows to registers (timestamps and counters) or approximately via a N-to-1 relation via hash-based algorithms (sketches and bloom filters); ii) disseminating and piggybacking information via packet marking (tags); iii) reduce the data volume and enable off-network collection (samples) and iv) selection and grouping of packets or flows to which measurements need to be applied.

4.2 Primitives composition

To express complex measurement tasks, the primitives described above can be composed using two basic forms of composition:

Sequential composition. Primitives can be composed in serial order using the sequential operator ≫. A composition: $primitive_1 \gg primitive_2 \gg \ldots \gg primitive_N$ executes the primitives in order, where the results of the execution of a primitive are made visible to the following. A primitive sequentially composed to a *Match* operation is only executed upon a positive evaluation of its condition.

Parallel composition. Primitives can be parallelized via the operator +, the expression $primitive_1 + primitive_2 + \ldots + primitive_N$ executes the primitives independently and applies multiple disjoint actions to the packet. A parallel composition of mutually exclusive *Match* operations allows the specification of if-else alike behavior.

At high level, a composition expresses a chain of operations, working on an abstraction of a stream of packets fed as input. Primitives execute on a per-packet basis, updating the switch memory and manipulating packets according to the primitives' semantics.

The proposed set of primitives can, when opportunely configured and assembled, encompass a wide range of measurement needs. We have successfully expressed many well-known techniques from the literature by means of a combination of these building blocks. Additionally, we addressed use cases that, at the best of our knowledge, have not yet been considered in the literature. We have so far identified three examples of such measurements. In Section 4.4, we will discuss concretely one of them: measuring path changes.

4.3 Compilation to programmable dataplanes

We have built a compiler from a high-level specification of measurement into a P4 switch pipeline configuration. The compiler analyzes the structure of a measurement composition and translates each involved primitive into a set of independent tables and actions. These are linked together as specified via the parallel and sequential operators, resulting in an ordered sequence of steps to be applied at runtime on each incoming packet.

Targeting multiple architectures Although our approach is not tied to any particular hardware switch implementation, we chose as target a PISA (Protocol-Independent Switch Architecture)[2] switch programmed in P4. This is a straightforward choice for our solution given their high degree of programmability. Currently our solution is capable of compiling to the behavioral model of a P4 software switch[1]. We are currently improving the compiler to generate the P4 configuration for an hardware Tofino switch.

Extending our solution to target different switch architectures or P4 switch implementations is a possible future direction to further enhance its generality. As well, programmable smartNICs are an alternative candidate to provide even broader applicability of our approach.

A unified framework for the implementation, compilation and deployment of measurement primitives would indeed be beneficial to improve the portability of network measurement algorithms.

4.4 Applicability of primitives

In this paragraph we present a measurement of particular relevance for SDN: detecting and counting path changes. As SDN allows for fast and frequent changes in network behavior, the data collected with this measurement can help with both traffic steering decisions and performance debugging.

The code snippet in Listing 1 demonstrates how to tackle it:

```
1 (BloomFilter(key:{switch_id}, λu:{1}, location_bf)
2   ≫ Tag(pkt.bf_tag | location_bf, pkt.bf_tag))
3 +
4 (Match(is_last_hop(pkt))
5   ≫ Match(pkt.bf_tag != paths_sketch{key:flow_id})
6   ≫ Sketch(λu:{n_changes + 1}, n_changes{key:flow_id})
7   ≫ Sketch(λu:{pkt.bf_tag}, paths_sketch{key:flow_id}))
```

Listing 1: Measuring path changes

The measurement consists of two parallelized parts (Lines 1-2 and Lines 4-7). First, at every hop, we encode the packet location (*i.e.,* the switch id) in a bloom filter (Line 1) and tag the result back into the packet (Line 2), by or'ing the local computed value with that carried in the packet. As the packet travels to destination, the packet tag will evolve into a compact representation of the followed path. Second, at the last hop, we use a Sketch to track the last path followed. On the last hop (Line 4), we compare the value of the tag in the current packet against the value of the previous packet(s) of the same flow (Line 5). If the values mismatch, we increment a Count-Min Sketch tracking the number of times a flow changed its path (Line 6) and update the latest value seen (Line 7). Both sketches are indexed using the flow 5-tuple as key, while the bloom filter is indexed at each hop with a switch-unique identifier; the λ_u functions provided as second parameter to both primitives are user-supplied lambda function specifying how to update the data structures.

ACKNOWLEDGMENTS

This work is supported by the EMJD-DC program (Erasmus Mundus Joint Doctorate in Distributed Computing), under the supervision of Luis Rodrigues (Instituto Superior Técnico) and Marco Canini (KAUST), with the collaboration of Balachander Krishnamurthy from AT&T Labs–Research.

REFERENCES

[1] The P4 Software Switch Behavioral Model. https://github.com/p4lang/behavioral-model.
[2] P. Bosshart, G. Gibb, H.-S. Kim, G. Varghese, N. McKeown, M. Izzard, F. Mujica, and M. Horowitz. Forwarding metamorphosis: Fast programmable match-action processing in hardware for sdn. In *SIGCOMM*, 2013.
[3] Bosshart, Pat et al. P4: Programming protocol-independent packet processors. *SIGCOMM Comput. Commun. Rev.*, 2014.
[4] A. R. Curtis, J. C. Mogul, J. Tourrilhes, P. Yalagandula, P. Sharma, and S. Banerjee. Devoflow: Scaling flow management for high-performance networks. In *SIGCOMM*, 2011.
[5] M. Ghasemi, T. Benson, and J. Rexford. Dapper: Data plane performance diagnosis of tcp. In *SOSR*, 2017.
[6] L. Jose, M. Yu, and J. Rexford. Online measurement of large traffic aggregates on commodity switches. In *Hot-ICE*, 2011.
[7] Y. Li, R. Miao, C. Kim, and M. Yu. Flowradar: A better netflow for data centers. In *NSDI*, 2016.
[8] McKeown, Nick et al. Openflow: Enabling innovation in campus networks. *SIGCOMM Comput. Commun. Rev.*, 2008.
[9] M. Moshref, M. Yu, and R. Govindan. Resource/accuracy tradeoffs in software-defined measurement. In *HotSDN*, 2013.
[10] A. Tootoonchian, M. Ghobadi, and Y. Ganjali. Opentm: Traffic matrix estimator for openflow networks. In *PAM*, 2010.
[11] N. L. Van Adrichem, C. Doerr, and F. A. Kuipers. Opennetmon: Network monitoring in openflow software-defined networks. In *NOMS*, 2014.
[12] M. Yu, L. Jose, and R. Miao. Software defined traffic measurement with opensketch. In *NSDI*, 2013.
[13] Y. Zhang. An adaptive flow counting method for anomaly detection in sdn. In *CoNEXT*, 2013.