MOZART: Temporal Coordination of Measurement

Xuemei Liu[*] Meral Shirazipour[†] Minlan Yu[*] Ying Zhang[‡]
[*]University of Southern California [†]Ericsson Research [‡]Hewlett Packard Labs

ABSTRACT

In data center and ISP networks, many monitoring tasks are not at a single network device and require coordination across many devices. Because network devices have different views of traffic and different capabilities of monitoring traffic properties, it is useful for one device to tell another one which flows to monitor at which time, rather than monitoring all the flows all the time. In this paper, we present MOZART (MOnitor flowZ At the Right Time), which enables temporal coordination across network devices. MOZART includes two key components: the *selectors* which capture network events and select related flows, and the *monitors* which collect flow-level statistics of the selected flows. We design temporal coordination algorithms and mechanisms across selectors and monitors to maximize the monitoring accuracy while staying within memory constraints. We also optimize the placement of selectors and monitors to support the maximum number of monitoring tasks. We implement MOZART in an Open vSwitch-based testbed and run extensive experiments with real traffic traces. Our results show a reduction of the false negative ratio from 15% to 1% compared to the existing method without coordination.

CCS Concepts

•Networks → Network measurement; Network manageability; Data center networks;

Keywords

Software Defined Networking, Network Measurement.

1. INTRODUCTION

In data center and Internet service provider (ISP) networks where operators own the entire network, monitoring on each device (i.e., a host or switch) should no longer be treated separately. To fully leverage the different views of traffic and different capabilities in monitoring different flow properties across devices, it becomes important to run coordinated measurements across different devices. For example, an ingress switch is a better place to measure per source traffic while an egress is better at measuring per destination traffic. A host can monitor packet losses by collecting TCP-level statistics or packet-level traces, while switches can more easily measure the traffic through a network path.

SOSR '16, March 14-15, 2016, Santa Clara, CA, USA
© 2016 ACM. ISBN 978-1-4503-4211-7/16/03...$15.00
DOI: http://dx.doi.org/10.1145/2890955.2890964

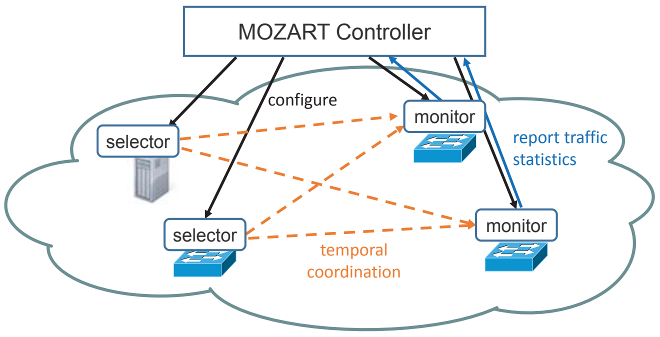

Figure 1: MOZART architecture.

Previous research on network-wide measurement often considers the spatial coordination of network devices. For example, CSAMP [1] uses consistent hashing to ensure different switches monitor a different set of flows. DREAM [2] leverages a centralized controller to decide which switch monitors which flow.

Complementary to spatial coordination, in this paper, we observe the need for temporal coordination of measurement across devices. For example, we only need to start monitoring the per-flow volume at every hop of a path when we observe end-to-end anomalies at hosts (e.g., unexpected large flows, unexpected packet losses). Another example is diagnosing equal cost multi-path (ECMP) hashing problems. Rather than continuously monitoring all the flows at all the ECMP paths, we only need to start monitoring the flows whose traffic across any of the paths becomes large.

To support such temporal coordination, we propose MOZART (MOnitor flowZ At the Right Time). As shown in Figure 1, MOZART configures two key types of components for each monitoring task: the *selectors* which capture network events, select related flows, and send the information to the *monitors* which will collect flow-level statistics for the selected flows. We make the following contributions in MOZART:

Coordinated monitoring at selectors and monitors: To ensure high monitoring accuracy and reduce the memory usage, it is important to coordinate the selectors and monitors in a timely fashion. We introduce a two-mode (*normal* and *event*) coordination algorithms to keep interested flow property data all the time, and leverage coordination information to allocate more memory resource to selected flows in monitors.

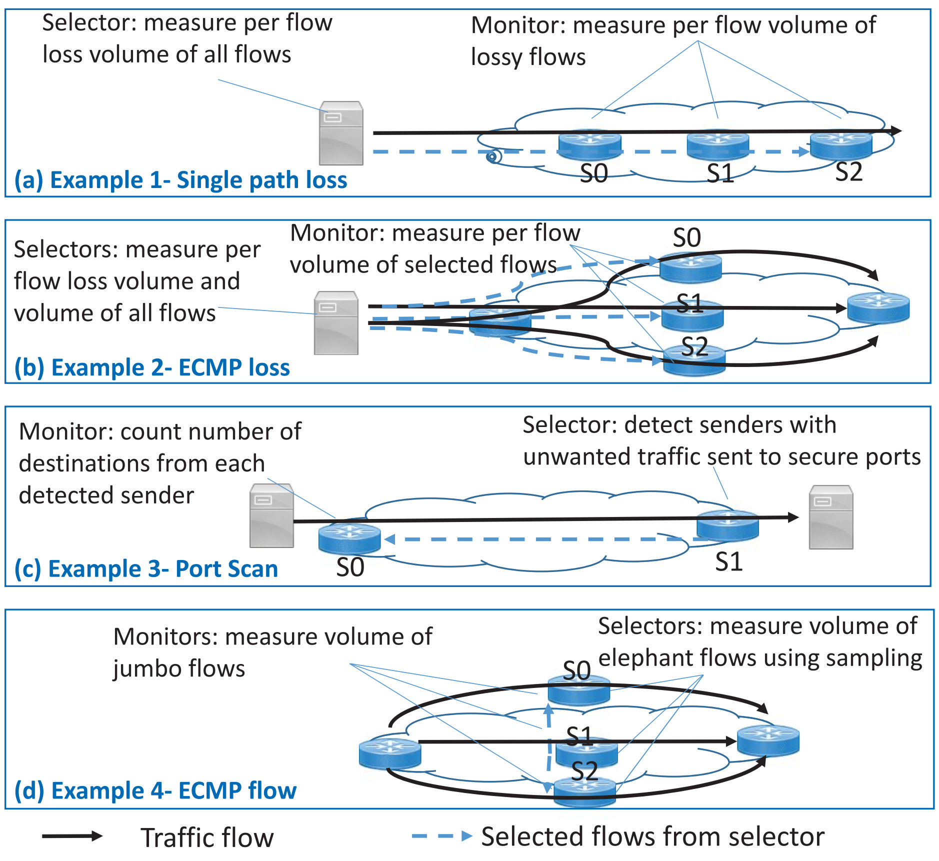
Selector: measure per flow loss volume of all flows
Monitor: measure per flow volume of lossy flows
S0
S1
S2
(a) Example 1- Single path loss
Selectors: measure per flow loss volume and volume of all flows
Monitor: measure per flow volume of selected flows
S0
S1
S2
(b) Example 2- ECMP loss
Monitor: count number of destinations from each detected sender
Selector: detect senders with unwanted traffic sent to secure ports
S0
S1
(c) Example 3- Port Scan
Monitors: measure volume of jumbo flows
Selectors: measure volume of elephant flows using sampling
S0
S1
S2
(d) Example 4- ECMP flow
Traffic flow
Selected flows from selector

Figure 2: Examples of temporal coordination.

Efficient communications between selectors and monitors: Since the selector and monitor may be located in different devices, we introduce packet tagging and explicit messaging to ensure the efficient communication between the selectors and monitors ensuring high monitoring accuracy with low bandwidth overhead.

Joint placement of selectors and monitors: To support the maximum number of monitoring tasks in a network of devices with limited memory, we formulate and solve a joint placement problem that identifies the best locations of selectors and monitors by considering the relation between selectors and monitors and the latency constraint between them.

Extensive experiments based on B4 [3] topology with realistic CAIDA trace data [4] show that our coordination algorithm can significantly decrease the false negative ratio from 15% down to 1% with a large memory capacity, and from 83% to 30% when the memory is very limited. Our placement algorithm can assign 22% more tasks, while reducing the average communication latency between selectors and monitors of the same task from 94 ms to 64 ms.

2. MOTIVATION AND CHALLENGES

In this section, we first give a few examples to motivate the need of temporal coordination of measurement across network devices. Then, we discuss the key challenges in supporting temporal coordination.

2.1 The case for temporal coordination

Today, there have been many independent measurement solutions at hosts and switches. At hosts, we can capture the volume of packets and packet loss by collecting TCP-level statistics [5] or packet-level traces. At switches, we can measure sampled per-flow or per-packet information using NetFlow [6, 7], sFlow [8], sample and hold [7], or OpenSketch [9]. Both hosts and switches have resource constraints when running these measurement tasks. Hosts need to devote most of their resources to the revenue-generating applications, leaving fewer processing resources for measurement. Switches have limited memory to store measurement data from all flows.

Given the resource limitations at individual hosts and switches, instead of monitoring all the flows all the time, we observe the benefits of coordinating these devices to monitor the right flows at the right time. We call this *temporal coordination* in this paper. With temporal coordination, we can leverage limited resources at hosts and switches to only capture the important flows of interest. For this, we need *selectors* which decide when and which flows to monitor and *monitors* which collect information for the selected flows at the right time. We now give a few examples highlighting the benefits of temporal coordination:

Example 1 (*Single path loss*): High loss diagnosis on a single path. Hosts may have many flows traversing a network path. It is too expensive to track all the flows at all the switches all the time. Instead, we only need to track flows on a given path when the host detects higher packet loss on some of the flows. Given the flows with higher loss experience (called *lossy* flows in this paper), we can capture their traffic volumes at each hop along the path, in order to identify the switches who contribute the most to the loss. As shown in Figure 2a, we can set up the selector at hosts that identify flows with high loss. Meanwhile, each switch along the path monitors the traffic volume of only these lossy flows. At the end of the measurement interval, the per-hop volume measurement will be sent to the controller to localize the cause of packet loss (e.g. congestion).

Example 2 (*ECMP loss*): High loss diagnosis on ECMP paths. The diagnosis of lossy flows becomes more challenging when these flows go through multiple ECMP paths. In addition to switch problems (e.g., buffer overflows, hardware deficits), packet loss can also be caused by unbalanced splits across ECMP paths, for example when a hash function does not split the flows equally across all the ECMP paths [10]. Such imbalance can cause waste of computing resources or performance problems. However, when there are too many lossy flows, the hosts have to focus on diagnosing only the larger flows. Therefore, as shown in Figure 2b, the hosts have two selectors, one for loss measurement and one for volume measurement. When a flow has both high packet loss and large volume, the selector notifies three monitors (switches S_0, S_1, and S_2), one on each ECMP path, to monitor the volume for the selected flow. If the volume on one path is significantly higher than others, the operator may decide to further investigate the ECMP algorithm.

Example 3 (*Port scan*): Port scan detection. Port scan detection is critical for cloud security [11]. One strawman approach to detect port scanning is to count the number of ports that each sender uses. However, it takes too much memory to monitor all the flows from all the senders. Instead, we choose to only monitor those suspicious senders who have been sending unusual traffic as reported by the receiver end. As illustrated in Figure 2c, we can first install selectors on the egress switches or receiver hosts to identify those flows trying to access unused ports at S_1 (e.g., if the receiver is a web server, it should only use port 80 and 443 and all the other ports are unused). We then install a monitor at the ingress switch near the sender (S_0) to only capture those flows identified by the receiver, and count the ports the sender uses.

Example 4 (*ECMP flow*): Elephant flow distribution on ECMP paths. Another way to monitor ECMP imbalance is to monitor the same flow at all the paths and compare their volume differences. However, monitoring every flow on every path is not scalable. Instead, we can have a switch on each ECMP path run a selector and a monitor. The selector measures the volume of elephant flows using a sampling technique. If the volume of the elephant flows on one path exceeds certain threshold, the selector asks the monitors on other ECMP paths to monitor the volume of these jumbo flows. The monitor will simply measure the volume of selected flows without any sampling. Note that the elephant flow on one path may not be an elephant flow on other ECMP paths. The goal is to examine whether the jumbo flows are balanced across the ECMP paths. For illustration, Figure 2d shows S_1's selector sending selected flows to S_0 and S_2's monitors.

2.2 Challenges

There are four key challenges in coordinating the selectors and monitors:

Coordinate in a timely fashion: The first question is how to coordinate between the selectors and monitors. One strawman solution is whenever a selector captures an event it immediately notifies the monitor to capture other properties of the selected flows. In the *single path loss* example, once the selector detects a high loss, the monitor needs to be notified to start the volume measurement. Similarly, in the *port scan detection* example, when the egress switch detects suspicious hosts, it notifies the monitor on the ingress switches to capture other properties of flows from those hosts.

However, if it takes too long for the selector's signal to arrive at the monitor, the flows for which we wish to capture other properties at the monitor may have already passed by. In the *single path loss* example, a large portion of the lossy flows may have already passed by the switches, so the monitors will fail to report the accurate flow volume.

Pre-signal monitoring is needed to increase accuracy: As described above, for some queries, the monitor may need to start the measurement even before the arrival of the selector's signal. In the *single path loss* example, if we only start monitoring a flow's volume when its loss becomes higher than a threshold (e.g. 6 Kbytes), we may face inaccuracy in the volume measurement at the monitor, because we have lost the information about the packets of the flow that already passed by during the selector measuring and signaling process. In such cases, the selector and the monitor may need to perform the measurement simultaneously.

To address this challenge, we propose to have the monitor capture the flows in different modes. In the normal mode, the monitor treats all the flows as of equal importance. When the selector sends event information about selected flows, the monitor switches to the event mode and devotes more resources to monitor those selected flows. For example, before the event (e.g., loss $<$ 6 Kbytes) happens, the monitor can allocate memory resources equally to all the flows (e.g., through uniform sampling). Once the selector identifies the selected flows satisfying the event, the monitor can devote more resources to these selected flows (e.g., set higher sampling rates).

Reduced coordination overhead: There is a tradeoff between coordination overhead and measurement accuracy. The selector can coordinate with the monitor at different levels of information and at different frequencies. For example, the selector can just send one triggering signal or send the full list of flows it identifies as being important to capture. The selector can either piggyback the information in-band or generate new packets and send them to all the monitors. Sending more information more frequently can enhance the measurements accuracy. However, it also takes more bandwidth overhead. We design packet tagging and explicit messaging mechanisms for different monitoring scenarios that minimize such bandwidth overhead while ensuring high monitoring accuracy.

Efficient network-wide resource usage: There are often many concurrent monitoring tasks in a network. The key question is how to allocate the limited memory resources at different hosts and switches to these monitoring tasks. Although there have been many works on resource allocations [1, 2], the new challenge here is that we need to ensure there are resources for both selectors and monitors of one task. If there are only resources at the selector to capture an event but not enough resources to measure the corresponding flows at the monitor, it is better to devote the selector's resources to other monitoring tasks. We design a new resource allocation solution that maximizes the number of assigned monitoring tasks considering the bundles of selectors and monitors and associated constraints.

3. MOZART OVERVIEW

In this section, we give an overview of MOZART as shown in Figure 1. We describe step by step how operators can initialize a monitoring task. Then, we give a brief introduction of novel methods that enable the coordination between selectors and monitors, and how challenges in Section 2 are solved.

Example	Selector		Monitor	
	event	location	property	location
1) Single path loss	loss volume > 6 Kbytes	host	total volume	$S_0\ AND\ S_1\ AND\ S_2$
2) ECMP loss	loss volume> 6 Kbytes AND total volume> 120 Kbytes	host	total	$S_0\ AND\ S_1\ AND\ S_2$
3) Port scan detection	dst port$\in$ Secure Ports	S_1	distinct destination	S_0
4) ECMP flow	total volume$> 1K$	$S_0\ AND\ S_1\ AND\ S_2$	total volume	$S_0\ AND\ S_1\ AND\ S_2$

Table 1: Specification of example monitoring tasks.

3.1 APIs to specify monitoring tasks

In order to actively collect data for targeting and analyzing events in the network, the operator can initialize MOZART tasks. Below we describe part by part what the operator needs to include in a MOZART task description.

As an initial step, the operator needs to specify the granularity of flows to be monitored. Flows, or flow aggregates, can be specified by any combination of packet header fields, such as source and destination IP addresses, ports, protocol, as well as MAC addresses. For example, in the *single path loss* example, the flow is defined based on the source and destination IP addresses. In the *port scan* example, the flows can be defined as the source IP, i.e., the IP addresses of suspicious senders. The flow definition will be consistent in the selector and monitors of a given MOZART task.

Generally, one MOZART task consists of selector(s) and monitor(s). Selector(s) are responsible for identifying the flows satisfying the event(s), and notifying the monitor(s) about them to measure their intended properties. Selectors are configured with the event to be captured at the associated location(s). Monitors are configured with the property to be measured at the associated location(s).

Given a flow definition, the first step in creating a MOZART task is to decide the event(s) to be monitored by the selector(s). An event can be a network state change (e.g., link failure, routing changes, etc.) or any condition on the value of the flow's properties. Typical flow properties are throughput, loss volume, loss rate, latency, and jitter within a time interval, or the statistics of these properties (e.g., average, maximum, standard deviation, or changes between intervals). For example, in the *port scan* case, the event is attempts to access unused ports at the servers. In the *single path loss* example, the event is large loss volume (loss $> threshold1$). Moreover, one MOZART task can require multiple events. They can be combined using AND/OR operators, where AND means the events must all be satisfied, and OR means satisfying one event is enough to select the flow. In the *ECMP loss* example, two events are configured, i.e., large loss volume (event1: loss $> threshold1$) AND large total volume (event2: volume $> threshold2$), and only flows satisfying both events are selected.

The second step in creating a MOZART task is to specify the candidate location(s) where selector(s) can be placed to capture the event(s) of interest identified in step 1. The locations of the selectors to measure different events can be explicitly specified or loosely specified using AND/OR operators. The AND operator means the selectors have to be deployed at those locations, and the OR operator can be used to specify a list of candidate locations. In the *ECMP flow* example, there are three selectors that must be placed in the three ECMP paths (AND) to detect the only event of elephant flows. For each selector, it can be placed in any disjoint switch of the local ECMP path (OR). In data center topologies like fat-tree [12] and VL2 [13], the candidate disjoint locations on each ECMP path are usually the core or aggregate switching devices.

The third step in creating a MOZART task is to specify the properties to be measured at the monitor(s) upon notification from the selector(s). This depends on what flow properties the operator is interested in. The possible flow properties in monitors are the same as those available in selectors, as listed in step 1.

The fourth and last step in creating a MOZART task is to specify the candidate location(s) for monitor(s). Without doubt, devices that have the resources to observe flow properties of interest and that are on the path of the selected flows at strategic locations are ideal for acting as monitors. For example, in the *port scan* case, monitors can be placed at ingress switches near the hosts that are suspected to execute port scans. Moreover, sometimes flow property data at diverse locations needs to be collected to achieve the overall goal of the MOZART task. In the *ECMP loss* example, flow volume through different ECMP paths should be collected to analyze the imbalance distribution of the flow.

Table 1 shows the MOZART task descriptions for the examples in Section 2.1. We take *single path loss* as an example to explain in more detail. The location of the selector is at the host, and the event for triggering the selector's notification to the monitor is that detected loss is larger than 6 Kbytes. The property measure at the monitors is the total traffic volume and the location is at all switches along the path.

3.2 MOZART design

In order to achieve obtain better measurement performance with the configured MOZART tasks, we propose the following design to address the challenges described in Section 2.

Coordinated measurement at selectors and monitors (Section 4): Rather than independently monitoring at individual hosts and switches, we use selectors to capture the events. Based on these events from selectors, we ensure the monitors capture the right flows at the right time. There are two key challenges for the coordination: First, part of the flow may have already passed by before being selected at the selector side. How to ensure the monitor captures that part of the flow is a challenge and is required to improve measurement performance. The second challenge is to efficiently use the limited memory at monitors to maximize the opportunities to capture selected flows, both before and after flows are selected by selectors. To address the first challenge, we propose a two-mode coordination scheme: In the *normal* mode, monitors employ sampling techniques to keep the flow properties for non-selected flows. In the *event* mode, monitors capture all of following flow properties. In order to address the second challenge, we leverage coordination information between selectors and monitors to make best use of limited memory in hash table in monitors.

Efficient communication between selectors and monitors (Section 5): Since the selectors and monitors may be located in geographically separate devices, we need an efficient network-wide communication method for their coordination. We introduce packet tagging and explicit messaging to ensure the efficient communications between the selectors and monitors with low latency and low bandwidth overhead while ensuring high monitoring accuracy.

Coordinated placement of selectors and monitors (Section 6): To support the maximum number of MOZART tasks in a network with devices of limited memory, we should identify the best locations of selectors and monitors. There are two key concerns: First, we can only support a task when we install all its selectors and monitors. Second, since the selectors need to send the events to the monitors in a timely fashion, the distance between them has an impact on the measurement results. Also, we need to consider that one selector may send selected flows to multiple monitors, while one monitor may receive selected flows from multiple selectors.

As a result, we develop a joint placement algorithm that identifies the best locations for selectors and monitors that maximizes the number of MOZART tasks while respecting latency constraints between selector/monitor pairs and abiding to node constraints in terms of monitoring functionality and resource capacity.

4. COORDINATION ALGORITHMS

In this section, we investigate ways to coordinate the measurement between one selector and their associated monitors. We design a two-mode schema to capture flow properties both before and after one flow is selected, and propose a memory management mechanism in monitors to make the best use of limited memory. In order to explain our algorithm better, we studies two example cases. Finally, we discuss designs to handle more complicated scenarios with multiple monitors and selectors.

4.1 Monitor based on selected flows

In order to better explain our coordination algorithms, we first focus on one selector in one MOZART task, and then extend to multiple selectors in Section 4.3. Most of the traffic properties (e.g., throughput, loss rates, latency, jitter) can be easily collected using existing switch primitives (e.g., tcpdump at hosts, [7], [14] at switches). We can then maintain these collected properties in hash tables, where each entry includes a flow identifier (e.g., 5 tuples) and its counters (e.g., per flow packet and byte counters, etc.).

Given the limited memory for hash tables, we need to select the important flows to monitor rather than monitoring all the flows. However, it may not often be possible to identify the important flows at the monitor itself. This is because we may need to capture some flow properties that are only available at another location, or we may need to start capturing some flows at the monitor only when other switches along the path detected a network event. In the *ECMP loss* example, operators want to capture the flows with high volume and packet loss across all ECMP paths (which are not necessarily the large flows that will be captured at the monitors).

It is important to coordinate between the selector and the monitors. We define the important flows as *selected* flows identified as the set of flows satisfying the defined events in a MOZART task. One strawman coordination solution is to have the selector detect and send the information of the selected flows to the monitors; and have the monitors measure desired properties of the selected flows. There are two key problems in such an approach. First, due to the delay of capturing and transmitting the events, by the time the monitor receives the event notifications for the selected flows, the monitor may no longer be able to capture the full information of the selected flows because part of the flows may have already passed by. Transient events are hard to be captured due to similar reasons. Second, efficiently using the limited memory resources is a challenge. If the monitor only measures selected flows' properties, the memory space at the monitors is considered as wasted during the time periods when the monitor has not received enough selected flows. In the reverse scenario, selected flows may be overwritten by non-selected flows competing to use the hash table at the monitors in case of collisions.

In order to solve the first problem, we propose a two-mode schema, i.e., the *normal* mode for non-selected flows and the *event* mode for selected flows. In order to fix the second problem, we introduce memory management mechanisms in monitors.

***Normal* and *event* modes:** On the selector side, *normal* mode applies to flows not satisfying associated event yet. In this mode, a selector keeps measuring flow properties and checking whether a flow have satisfied the associated event or not. Once a flow has satisfied the event, it enters *event* mode and the selected signal is sent to all monitors. In the *single path loss* example, the set of flows with loss $> 6Kbytes$ enter *event* mode.

On the monitor side, *normal* mode applies to non-selected flows. As non-selected flows may become selected in the future as well, it is meaningful to capture flow property data even in this mode. Due to limited resources in monitors, we propose to employ sampling techniques (e.g., [9, 7, 6, 8]) to capture the flow properties of non-selected flows. This makes each non-selected flow have a probability to be captured. Depending on the purpose of the MOZART task and the properties measured at the monitor side, associated sampling techniques can be utilized. For example, sample and hold [7] can be employed to capture flows of large volume with larger probability. OpenSketch [9] can identify superspreader flows with high probability. Once a non-selected flow becomes selected in the future, the already sampled flow properties can compensate for the part of the already lost information before being selected. When a flow is signalled as selected from the selector, it enters *event* mode in the monitor. After that, the monitor definitely captures the selected flows. This makes sure that the following property data for the selected flows are captured with 100% probability.

In order to help the a flow enter the *event* mode earlier in the monitor (which leads to capture flow properties earlier), the selector may mark a larger set of flows, which may have not satisfied the event yet, but have a large chance to satisfy the event in the future. For example, in the *single path loss* example, flows with high loss but not exceeding $6Kbytes$ yet can be marked as event satisfied.

One key concern for the mode switching is that we may not be able to capture some transient events if the switching between the two modes is slow. Fortunately, the combination of *normal* and *event* modes also makes monitoring transient events possible. This is because the monitor has a chance to capture flow property even before flows are identified as selected flows, by sampling in *normal* mode in monitors, and by signalling flows as event satisfied earlier than expected in selectors.

Memory management at monitors: At the monitor, we introduce a signal bit in each entry in the hash table to indicate if a flow has been flagged as selected or not. When a monitor receives a list of selected flows from the selector, the monitor sets the corresponding signal bits to 1. For every incoming packet, we store the related properties in the hash table entry if the entry is not currently occupied, regardless of the flow is selected or not. In this way, we can make the vacant memory used by non-selected flows even before there are enough selected flows. As non-selected flows may become selected in the future, the already stored flow properties can count as a part of whole flow properties data. Most of time, there are both selected and non-selected flows stored in the hash table. When two flows collide in the same entry, we choose to store the selected flow over the non-selected one. This makes sure we allocate more resources to flows once selected. For example, in Figure 3, the signal bit of selected flow $f1$ is set. If an incoming packet of $f5$ is hashed in the same entry, it cannot overwrite $f1$. The same applies to $f2$ even when incoming packet is from selected flow $f6$.

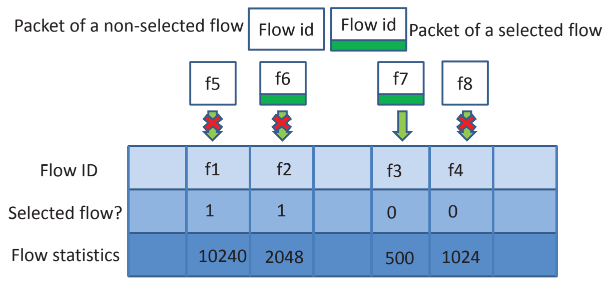

Figure 3: Memory management of hash table in monitors.

In contrast, suppose an entry already keeps properties for $f3$ and $f3$ is not a selected flow. When an incoming packet of selected flow $f7$ is hashed into the same entry with $f3$, $f6$ can take this entry and overwrite $f3$. Moreover, an entry taken by non-selected flow (e.g., $f4$) will not be overwritten by another non-selected flow (e.g., $f8$).

4.2 Case studies

We now take two examples to demonstrate the coordination between the selectors and their associated monitors.

In the *single path loss* example, during the *normal* mode, the monitors at switches employ *sample and hold* [7] to capture the non-selected flow, which is a memory efficient way to capture big flows. However, small flows (they also might be selected flows in the future) could be captured as well. In the *event* mode, the selector at the sender side signals all monitors along the path to capture the flow with 100% probability. As a result, both the *normal* and *event* modes contribute to capture more accurate selected flows' volumes at the monitors.

Similarly, in the *ECMP path loss* example, we also utilize the two-mode coordination methodology to capture flows in different ECMP paths. Moreover, we propose to customize the *sample and hold* algorithm in the *normal* mode, to achieve its goal of capturing large flows with higher probability in a more elegant way. The intuition comes from the observation that the sender witnesses the entire traffic flow splitting into each ECMP path, while one switch can just observe the traffic through the local ECMP path. Thus, in *sample and hold*, we propose to execute the *sample* process at the sender, and the *hold* process at switches. *Sample* at sender can more accurately identify large flows and notify monitors to *hold* those flows' volume property.

4.3 Handling multiple monitors or selectors

We first extend our basic solution to a single MOZART task with multiple selectors, and then discuss the management of monitors belonging to different tasks.

When one task has multiple events (combined by AND/OR operators), there would be multiple selectors sending signals to one monitor. In this case, one monitor needs to use a *bitmap* to indicate whether certain event is satisfied or not, and it needs to execute the logic operations (AND/OR) to decide whether one flow is selected or not. In order to make the logic operation possible in hardware, the MOZART controller needs to configure the monitors with the list of valid combinations of events. For example, if operators need to capture flows with either *high loss* rate OR *high volume*, the controller configures the list of $[01, 10, 11]$ in the monitors, and the monitors execute $(bitmap \wedge 01)|(bitmap \wedge 10)|(bitmap \wedge 11)$ to decide whether one flow is selected or not. Then, the memory management strategies in Section 4.1 can be applied to allocate more memory to selected flows.

At any given time, there are many MOZART tasks running in the network. The monitors of different tasks may be located at the same switch. In this case, we can maintain a single hash table for all the monitors at a switch. The switch may receive signals from different selectors. As described above, it will replace an existing non-selected flow in an entry if the flow conflicts with an incoming selected flow. If multiple selected flows are hashed in the same entry, operators can configure the right policies to decide which flow from which task takes priority. In this paper, we take the first-come-first-serve policy approach which maintains the entry for the first selected flow. We leave the question of how to support policy and fairness that gives each task an equal number of slots in the hash table for future work.

5. COMMUNICATIONS BETWEEN SELECTORS AND MONITORS

In this section, we discuss the communications between the selectors and their associated monitors to ensure that the information of selected flows is delivered in a timely fashion and with low bandwidth overhead.

MOZART employs different coordination mechanisms depending on whether the selector and the monitor are located on the same path or on different paths. We discuss two mechanisms in this section: packet tagging and explicit messaging.

5.1 Packet tagging

If the selector and the monitor are located on the same path, they can communicate by tagging the packets traversing the path. We now discuss what information to tag and which packets to tag.

What information to tag? There are three pieces of information that the selector should include in the message to the monitor:

(1) Selected flows: The selector needs to inform the monitor on the set of flows to monitor. If the packets which carry the tag belong to the selected flow, the selector simply needs to mark one bit on the packets to designate them as belonging to a selected flow, which is identified by the 5-tuple of the marked packet.

However, sometimes, the selector may choose to select a broader range of flows to monitor. In the *port scan detection* example, once the selector detects a suspicious port usage, it asks the monitor to monitor all the flows from the particular source IP address. Thus, the tag should indicate which of the header fields of the packet are used to identify the selected flows. We use a bit map as a mask to designate packet header fields. The semantics of the tag is defined by the controller. For example, to mask an IP header with 5-tuple information, we use five bits, each of which represents the source IP, destination IP, source port, destination port, and protocol. For example, a tag of "10100" means to capture all the flows that share the same source IP and source port as the packet carrying the tag. Note that the semantics of a tag only needs to be consistent between the selector and monitor pairs. Thus, we can have different definition of tags for different measurement tasks.

(2) Task ID: Since different tasks may need different pairs of selectors and monitors, it is important for each monitor to recognize the tagged packets that are related to its tasks. Therefore, the controller pre-configures each selector and monitor about the set of tasks it is responsible for. The selector tags the task ID in the packet. Upon receiving a tagged packet, the monitor first checks the task ID to see if it is responsible for that task. If the tag is targeted at the current monitor, the monitor extracts the selected flow information from the tag and maintains traffic properties for these flows.

(3) Event ID: Because one task can detect multiple events, a tag from a selector should also include an event ID. Upon receiving a tag, the monitor mark the associated bit in bitmap to indicate the event is satisfied.

Which packets to tag? When the selector and the monitor are located on the same path, the selector captures traffic information, identifies the selected flows, and tags all the following packets in the selected flows. If the selector is at the upstream of the monitor, it simply tags all the packets for the flow. In the *single path loss* example, the selected flow traverses from the host (where the selector is located) through the intermediate switches (where the monitor runs).

If the selector is at the downstream of the receiver, we can tag the reverse flows (e.g., the ACK packets in the TCP flow), if there are enough reverse packets in reverse stream. Otherwise, we use explicit messages as discussed in Section 5.2. In the *port scan detection* example, the selector runs at the egress switch and the monitor runs at the ingress switch of the selected flow. Thus we need to tag the TCP reset (RST) packets to notify the monitor, assuming that the scanner's packet triggers a TCP RST response from the destination. Note that tagging the reverse flows may not be timely in notifying the monitor to start monitor selected flows, because it depends on the time the reverse flow is generated.

We can reuse free bits in packet headers to set the tags. For IPv4 packets, the candidate fields include the 6-bit DS field, or the 16-bit IP_ID field. For IPv6 packets, we can use the 20-bit flow label field to carry the tagging information. Other tunneling header fields could be leveraged as well.

5.2 Explicit messaging

While the tagging method can help reduce the overhead on the network, it may not be applicable when the selector and monitor are located on different paths. In the *ECMP flow* example, the selector is in one switch in one of the ECMP paths while the monitors are on the other ECMP paths. The flows traversing the selector do not go through all the monitors. In this case, the selector constructs an explicit message with three fields: a set of selected flows, the task ID and the event ID, and sends explicit messages to the monitors using an out-of-band communication or relay the messages through the controller.

If we send one message for each selected flow, there would be too many messages that could overload the network or the controller. Instead, the selector should send a batch of selected flows in given time window. The time window should be not too large so that the monitor gets notified in time; but should not be too small to reduce the number of messages. To further reduce the number and size of messages, the selector can send the delta set of the selected flows each time compared to the previous time window.

6. JOINT PLACEMENT OF SELECTORS AND MONITORS

Operators configure the MOZART tasks as selectors and associated monitors. This often comes with some flexibility in the locations of the monitors and selectors. In the *ECMP loss* example, the selector should be at the host, but the monitors can be any switch on each ECMP path. In the *ECMP flow* example, the selector can be any switch on one ECMP path, and the monitors can be any switch on other ECMP paths.

The MOZART controller is responsible for allocating the incoming MOZART tasks among the network resources. Selectors and monitors of one MOZART task should all be allocated or none at all, otherwise they are of no use. Network devices can spare limited resources to support measurement besides normal packet processing duties, and usually they cannot support all MOZART modules (either selector or monitor). Moreover, placing the selectors and monitors closer to each other could help reduce the signal propagation delay, and can potentially help improve measurement performance.

N	set of network nodes
K	set of tasks to be placed
M	set of selector/monitor modules from tasks in K
ρ_{st}	=1 if selector s and monitor m belong to the same task
ξ_{km}	=1 is module m is a selector or monitor of task k
β_i	monitoring capacity of node $i \in N$
γ_m	required resource of module $m \in M$
λ_{mi}	=1 if node i can support module m
Δ_{ij}	Latency between node i and j
δ_{st}	Latency limit between selector s and monitor m
ω	A very large positive number
g_{mi}	Binary variable to designate module m was assigned to node i
Φ_k	Binary variable to designate taks k was placed

Table 2: Notations used in BIP

Given such flexibility, a natural question is how to place the selectors and monitors so that we can support as many MOZART tasks as possible, while meeting the resource and latency constraints.

We formulate a binary integer linear program (BIP) which maximizes the number of placed tasks while considering all above factors as follows (Table 2 presents the notations):

$$\text{Max} \sum_{k \in K} \Phi_k \tag{1}$$

subject to

$$g_{mi} \leq \lambda_{mi}\ , \quad \forall m \in M, i \in N \tag{2}$$

$$\sum_{i \in N} g_{mi} \leq 1\ , \quad \forall m \in M \tag{3}$$

$$\sum_{m \in M} g_{mi} \cdot \gamma_m \leq \beta_i\ , \quad \forall i \in N \tag{4}$$

$$\Phi_k \begin{cases} \leq 1, & \sum_{m \in M} \sum_{i \in N} \xi_{km} \cdot g_{mi} = \eta_k \\ = 0, & otherwise \end{cases} \quad ,\forall k \in K \tag{5}$$

$$\begin{aligned} &\rho_{st} \cdot \Delta_{ij} \leq \delta_{st}, \;\; if\ g_{si} + g_{tj} = 2, \\ &\forall\ s, t \in M, i, j \in N \end{aligned} \tag{6}$$

Objective function: We want to maximize the number of assigned MOZART tasks. The key difference with previous measurement placement solutions like CSAMP [1] is that here a task is successfully placed only when all its selectors and monitors are placed in the network. Let the variable Φ_k denote whether all selectors s and monitors t that belong to the same MOZART task k are successfully placed, where $k \in K$, $s, t \in M$ for K tasks and M modules (set of all selectors and monitors to be placed). The objective function is described in Eq.1 and maximizes the total number of placed tasks.

Assignment constraints: We set $\lambda_{mi} = 1$ if module m from set M can be assigned to node i (in terms of functionality, traffic view, etc.). Let g_{mi} denote whether module m is placed at node i. Eq.2 ensures that any module m is placed at a candidate node that can support it. Eq.3 ensures that a given module m is only placed once.

Resource constraints: Each node i in the network can only store statistics for β_i flows because of its limited memory and CPU for measurement. γ_m denotes an upper bound of the number of flows each module requires. For selectors, γ_m is the number of flows traversing the selector; for monitors, since it is hard to know ahead of time which flows are selected, we set γ_m as the number of flows traversing the selector. In the case where multiple modules colocate at the same node, we simply sum up γ_m because the resource usage is proportional to the total number of flows we monitor. Eq.4 assures that the node capacity is respected, i.e., the flows monitored by all tasks placed at node i cannot exceed its capacity β_i.

Constraints on placing all the selectors and monitors of the same task: For each MOZART task k, we add its selectors s and monitors t into a unified set M, and set $\rho_{st} = 1$ to indicate modules s and t are the selector and monitor modules of the same MOZART task. Thus, there can be multiple monitors t with $\rho_{st} = 1$ for one selector s. The same applies to selectors. Eq.5 derives the value of Φ_k. η_k designates the total number of modules (selector and monitor) for task k. Φ_k can be 1 only when all its modules s and t are placed.

Latency constraint: Ensuring timely condition signaling between selectors and monitors is critical for result accuracy.Therefore we should choose the assigned locations to each monitor and selector pair (belonging to the same MOZART task) to ensure the latency does not exceed a given bound $\delta_{s,t}$. Suppose the latency from node i to node j is Δ_{ij} in the network. Eq.6 describes the latency constraint.

Eq.5 and Eq.6 can be transformed and replaced by Eq.7 and Eq.8 respectively, to make the formulation linear. Here ω is a large positive number.

$$\eta_k \cdot \Phi_k \le \sum_{m \in M} \sum_{i \in N} \xi_{km} \cdot g_{mi}, \; \forall k \in K \tag{7}$$

$$\rho_{st} \cdot \Delta_{ij} \le \delta + \omega \cdot (2 - g_{si} - g_{tj}), \; \forall s, t \in M, i, j \in N \tag{8}$$

As new tasks arrive, we periodically rerun the above BIP to reoptimize the placement of selectors and monitors already in place in the network.

7. EVALUATION

In this section, we describe the experiments for evaluating both the coordination algorithm and the placement algorithm. We find that the coordination can significantly decrease the false negative ratio from 15% down to 1% with a larger memory capacity for *single path loss* example. For the *ECMP loss* example, our method reduces false negative by at least 40% and even improves the accuracy by 2% with a large memory size. Compared with a strawman algorithm that does not maximize the number of assigned tasks, our placement algorithm can place 22% more tasks while bringing down average communication delay from 94 ms to 64 ms.

7.1 Experiment setup

Network setup: We run our experiments on Google's B4 topology [3]. The network topology we use contains 12 switches and 12 data centers to which we assign disjoint IP prefixes, i.e., the first 8 bits of the IP address are used as the identification of the data center, and the remaining 24 bits are used to identify the servers in a given data center. We set up the topology in Mininet [15], running Open vSwitch (OVS) [16] switches, and having each data center represented as one host.

Forwarding rules: We install forwarding rules in OVS to forward packets based on shortest path routing. For cases where there are multiple shortest paths between two data centers, we extended OVS to run packet level ECMP.

Traffic generation: As we do not have access to the traffic matrix data in B4, we use the Caida trace [4] from the *equinix-sanjose* 10 Gbps link with an average 2 Gbps load, during 2012-9-20 13:00 to 15:00 GMT, to generate the traffic among data centers. We use the Caida trace in the following way. We divide all the IP addresses in the trace into 12 subsets, and assign each to a data center. Packets with source IP from one subset are treated as being sent out from the assigned data center, and those with destination IP in one subset are treated as being received by the respective data center. In order to simplify the number of forwarding rules installed in OVS, IP addresses in Caida trace are mapped to the IP domains of each data center.

Sending and receiving traffic: In Mininet, we employ Tcpreplay [17] at hosts as the senders in data centers to generate unlimited number of flows. As Tcpreplay does not build real connections, we implement a customized receiver program at hosts to generate ACK packets to senders. We implement the loss measurement module on the hosts by counting the un-ACKed packets. This module is added into Tcpreplay to get the real time volume and loss information of the flows.

Encoding and decoding tags: In our testbed, we encode the *selected flows* and the *task id* information in the VLAN field. The packet tagging process is also implemented in Tcpreplay. We also modified OVS to decode the tags at user plane. The measurement module is also added to OVS at the user plane.

Flows and measurement interval: The flows in our evaluation are of two-tuple, i.e., srcip and dstip. The selectors detect flows and monitors measure their volume in the same time interval. In order to accumulate enough volume for two-tuple flows, we set the length of one time interval as 30 seconds. In one time interval, there are around 1.5 million two-tuple flows in the network, and multiple MOZART tasks are running in the system simultaneously.

Compared algorithms: We compare two methods: *Coordination*, which uses our method to coordinate between selectors and monitors; and *No-coordination*, where the monitors only perform standalone *sample and hold* [7].

Evaluation metrics: We evaluate the performance of our method using two metrics. The *false negative ratio* is the number of missed selected flows that should have been captured by the monitor, divided by that of all selected flows. The *accuracy* of a flow is the captured volume divided by its total volume in the interval. For the latter, we calculate the *average accuracy* for all captured selected flows as metric of interest. For simplicity, we use the term *accuracy* to designate *average accuracy* in the following sections.

7.2 Benefit of coordination

We evaluate the benefit of coordination by implementing the first two examples in Section 2.1. For *single path loss*, *Coordination* reduces the false negative ratio from 83% to 30% when the resource is very limited, and reduces from 15% to 1.3% when the node capacity is large. For *ECMP loss*, *Coordination* reduces the false negative ratio from 98% to 40% when the resource is very limited, and reduces from 30% to 1.7% when the node capacity is large.

7.2.1 Case study: Single path loss

We set the loss threshold of selected flows to be 6 Kbytes. We randomly select 58 origin/destination (OD) pairs. One task is run-

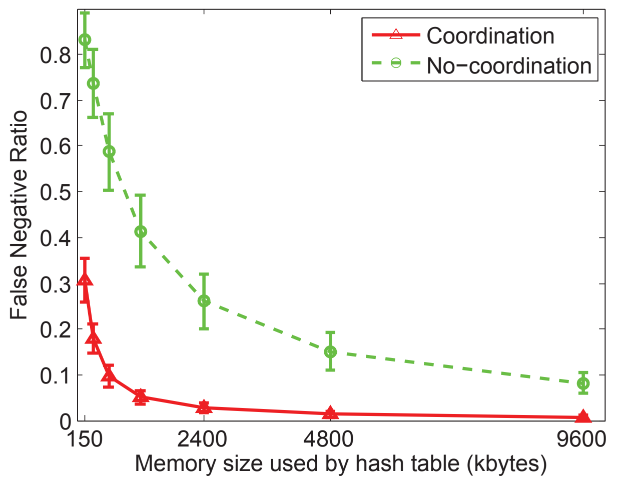

Figure 4: False negative ratio of *single path loss* example with 6 Kbytes loss threshold.

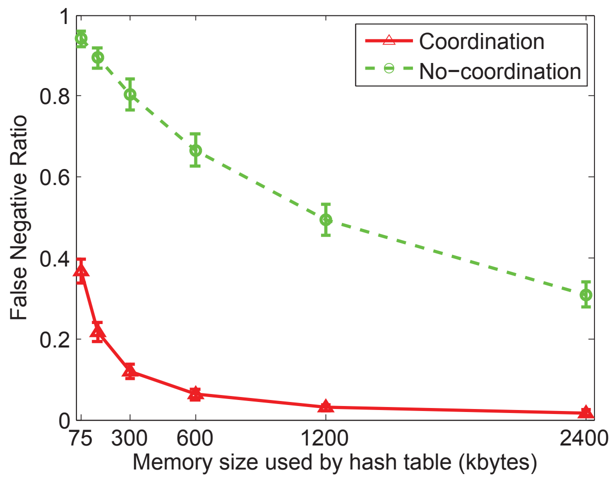

Figure 6: False negative ratio of *ECMP loss* example with 120 Kbytes volume and 6 Kbytes loss thresholds.

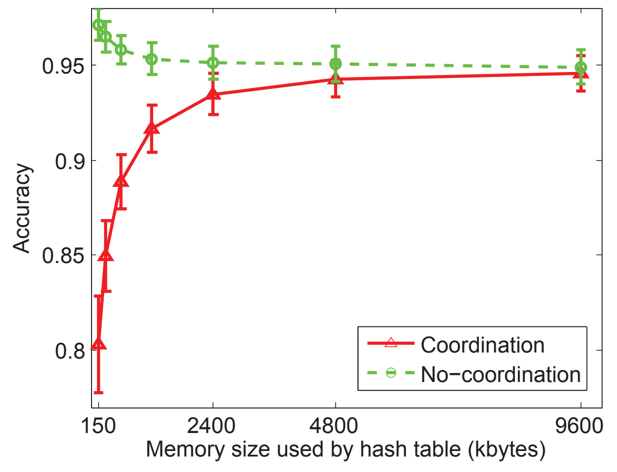

Figure 5: Accuracy of *single path loss* example with 6 Kbytes loss threshold.

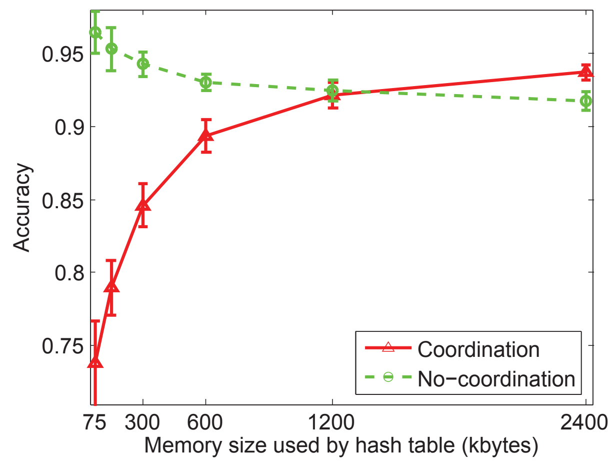

Figure 7: Accuracy of *ECMP loss* example with 120 Kbytes volume and 6 Kbytes loss thresholds.

ning on each OD pair: the host running as selector and all the switches along the path running as monitors. The number of selected flows per task is around 400.

Figure 4 shows *Coordination* has much lower false negative than *No-coordination*. The large flows captured by sample and hold in *No-coordination* are not necessarily the selected flows, and this makes the false negative ratio of *No-Coordination* larger. In contrast, in *Coordination*, the selector can signal the monitors which are the selected flows, and monitors can allocate memory resources to selected flows rather than to other flows. With the increased memory size, the false negative ratio for both algorithms decreases. When the memory size is 4.8 Mbytes, the false negative ratio of *Coordination* is 1.3%, while that of *No-coordination* is as large as 15%.

Figure 5 shows the accuracy. The accuracy of *Coordination* increases with memory size, while that of *No-coordination* decreases. When the memory size is small, *No-coordination* just captures few large flows which will be kept in the hash table once they are sampled. This is the reason why the accuracy of *No-coordination* is very high initially in the graph. When memory size is increased, *No-coordination* captures more selected flows due to fewer collision. The volume of the extra captured flows is smaller and the accuracy is lower due to the essence of sample and hold (flows with higher volume are captured with higher accuracy). Thus the average accuracy of all captured selected flows decrease when memory size increases.

In contrast, for *Coordination*, when the memory size is small, the accuracy is not as high as *No-Coordination*, as *Coordination* captures more selected flows, including both large and small flows. Similarly, larger memory size helps capture more selected flows due to fewer collisions. *Coordination* cannot help increase the upper bound of accuracy in this case because *sample and hold* is used to capture flow volume in the *normal* mode, which is the same as the *No-Coordination* case.

7.2.2 *Case study: ECMP loss*

For *ECMP loss* example, the volume threshold is set to 120 Kbytes, and the loss threshold is set to 6 Kbytes. We randomly select 28 OD pairs with ECMP paths between them, and allocate one *ECMP loss* monitoring task for each OD pair. The number of selected flows per task is around 400.

As shown in Figure 6, the improvement of false negative ratio is

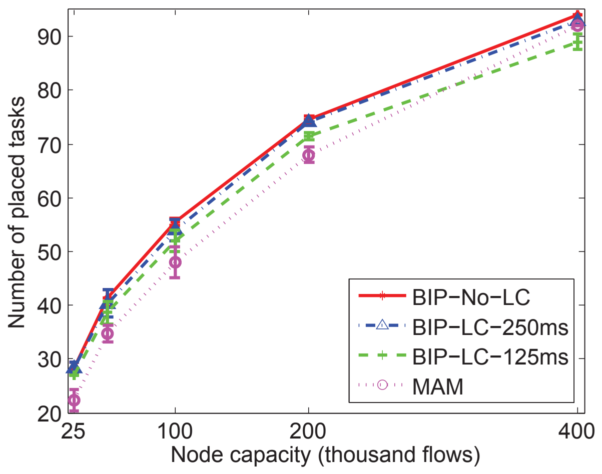

Figure 8: Number of assigned tasks of placement algorithms: 1 selector to 1 monitor.

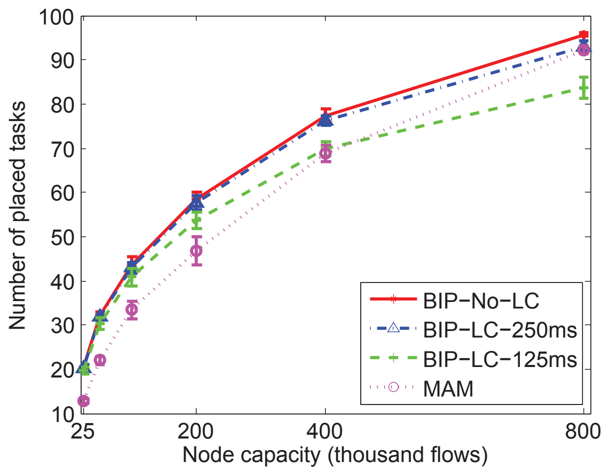

Figure 10: Number of assigned tasks of placement algorithms: 1 selector to 2 monitors.

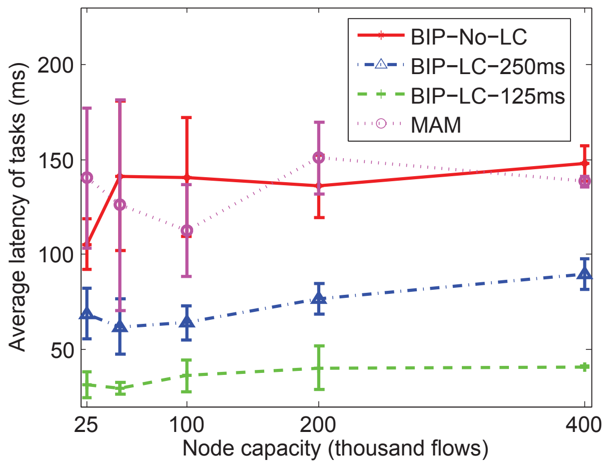

Figure 9: Average latency of placement algorithms: 1 selector to 1 monitor.

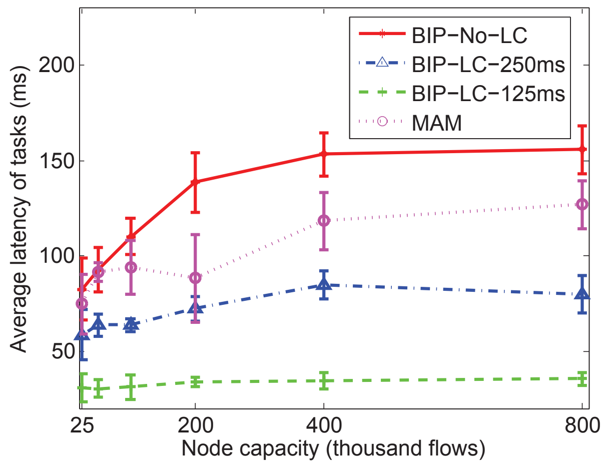

Figure 11: Average latency of placement algorithms: 1 selector to 2 monitors.

much larger for *Coordination*. When the memory is 75 Kbytes, *Coordination* reduces the false negative ratio from 98% to 40%, which means it manages the memory in a more efficient way. When memory size is 2.4 MBytes, the false negative ratio of *Coordination* is 1.7% while that of *No-Coordination* is still over 30%. In Figure 7, the accuracy of *Coordination* catches up when the memory is large enough. This is the benefit of our customized *sample and hold* algorithm. Sender of one OD pair that has ECMP paths can observe all traffic for one flow, while one switch can only observe partial traffic on the local ECMP path. Thus, *sampling* at host can help *sample and hold* improve the upper limit of accuracy in *normal* mode. Note that it is different for the *single path loss* example, where all the switches on the path can observe all traffic of the flow.

7.3 Evaluation of the placement algorithm

In order to demonstrate the benefit of binary linear program in Section 6, we compare with a strawman method that uses linear programming to Maximize the Assigned number of Modules, referred as *MAM* in the rest of this section. We compare the two methods with two metrics: the number of assigned tasks and the average latency of assigned tasks. If one task has multiple selectors and monitors, we calculate the average latency between all selectors and monitors. In our BIP algorithm, there is a tunable parameter, *latency constraint (LC)*, that controls the balance between the two objectives: maximizing the assigned tasks and minimizing the latency. In order to get the upper bound of assigned tasks, we also introduce BIP with no latency constraint (No-LC).

We first evaluate the scenario of $k = 1$, i.e., each task has one selector and one monitor module. Figure 8 shows the number of assigned tasks with increasing node capacity. Figure 9 shows the average latency of each task with different node capacities. Obviously, *BIP-No-LC* has the maximum number of assigned tasks because it only tries to maximize the number of assigned tasks. Meanwhile as it does not take latency into consideration, it has the largest latency. *BIP-LC-250ms* and *BIP-LC-125ms* both achieve good balance between assigned tasks and latency. When the node capacity is 100K flows, *BIP-LC-250ms* assigns 98% of the tasks compared to *BIP-No-LC*, and 64ms latency, compared to 87% and 112ms achieved by *MAM*. *MAM* can get larger number of placed tasks when the node capacity is large, because it has no latency constraint. However, its latency is always larger than BIP with latency constraint.

Similarly, we show the evaluation results for the scenario of $k = 2$, i.e., each task has one selector and two monitor modules. From Figure 10 and Figure 11, we can observe even a higher benefit with *BIP-LC*. When the node capacity is 100K flows, *BIP-LC-250ms* can assign 98% of our upper bound on the number of tasks, and achieve 64 ms latency, compared to 77% and 94 ms achieved by *MAM*. *BIP-LC-250ms* can assign 22% more tasks. This improvement is because for larger k values more attention is needed to find the best balanced location for selector and monitor modules within the same task.

8. RELATED WORK

There have been many traffic measurement solutions at hosts and switches/routers for monitoring per-packet or per-flow properties. Solutions such as NetFlow [6], sFlow [8], sample and hold [7], as well as works like [18], focus on basic sampling techniques for coarse-grained traffic flows. Other works like burst measurement [19], and latency measurement [14] started looking in more fine-grained solutions. Since fine-grained measurement faces quick resource exhaustion, various recent works followed on improved dataplane schemes for per-flow traffic measurement. For example, OpenSketch [9] is based on a three stage packet processing pipeline for SDN. It allows the development of more expressive traffic measurement applications by proposing an API to the packet processing pipeline. Another example is DREAM [2], which does dynamic adjustments of the resources devoted to each measurement task, while ensuring a user-specified level of accuracy. DREAM and OpenSketch perform in data plane notification without controller or operator involvement. These works serve as the key components in MOZART which focuses on how to glue these measurement modules together to enable temporal coordination on multiple flow properties across devices. CSAMP [1] presents a framework for network wide monitoring based on sampling and optimized resource utilization with hash-based flow selection instead of coordination. While MOZART has similarities with CSAMP, its scope is different as we work with monitoring tasks that are tightly interdependent in time and location.

Other solutions like NetSight [20] aim for holistic traffic view at the expense of being offline. They collect all the packet headers in all switches and send them to the controller, allowing offline analysis of network properties. Instead, MOZART works with coordinated queries and measures traffic across hosts and switches, only sending to the controller the properties of selected flows that match the query's description. At last, the work in [21] supports real time measurement based on queries that allow the selective measurement of traffic flows of interest. Complementary to this work, MOZART allows complex queries relating conditions detected at selectors to trigger measurements performed at monitors.

Finally, regarding measurement module placement, there is a long list of prior work in optimal probe placement for network monitoring. The more relevant ones to MOZART are the works in [1, 22, 23, 24, 25, 26, 27]. None of these proposals can be used for selector and monitor placement in MOZART as our architecture has tight constraints relating the selectors and monitors. However, our proposed solution was developed after careful study of these works.

9. DISCUSSION

We now discuss the key concerns of deploying MOZART on commodity switches and hosts, and propose the direction of multiple selected priorities.

Monitoring capabilities: MOZART can be built on commodity switch components. This is because MOZART mainly works with hash tables that match flows to counters, which are common in commodity switches. The actual traffic properties a switch or host can collect may differ. Thus, MOZART can only work with those traffic properties that the devices provide. For example, if a switch can support measuring the number of packets that it drops during buffer overflow, we can count the number of packet losses at the switch. If not, we can only count the number of packet losses at the hosts. MOZART can deploy the right monitoring functions at hosts and switches based on their capabilities.

Communications between selectors and monitors: MOZART requires the communications between selectors and monitors through tagging or explicit messaging. Commodity switches can already add tags to packet header fields such as VLAN fields. Switches can also send explicit messages (e.g., using in-band network telemetry (INT) [28]).

Multiple selected priorities: In order to coordinate, the selectors of one MOZART task send the selected flows to monitors. Generally, if more information besides selected-or-not is coordinated, the monitors can apply better strategy for flow property measurement. For example, in *single path loss* example, the selector can send three levels of selected priorities for flows, i.e., no-loss (*LOW*), loss smaller than threshold (*Middle*), loss over threshold (*High*). *Low* flows are captured by monitors under *normal* mode, and *Middle* and *High* ones are captured under *event* mode. They have ascending priorities of using hash table when collision happens. Preliminary experiments show three-level coordination outperforms just selected-or-not. However, it is non-trivial to design multiple selected priorities for MOZART tasks monitoring more than one event, and determine the conditions for different priorities.

10. CONCLUSION

Traffic measurement is a fundamental problem for efficient network and data center management. In this paper we observe that it may be more efficient to monitor the right flows at the right time instead of monitoring all the flows all the time. This requires careful coordination in time and location of the monitoring. MOZART enables network operators to achieve selective monitoring of flows of interest and hence improve on overall monitoring resource utilization while achieving real time monitoring of important events. MOZART ensures high accuracy for tasks, while taking network-wide resource and timing constraints into account.

11. ACKNOWLEDGEMENTS

We thank SOSR reviewers and Srinivas Narayana for their helpful feedbacks. This paper is partially supported by NSF CNS-1453662, CNS-1423505, CNS-1413972, and Ericsson.

12. REFERENCES

[1] V. Sekar, M.K. Reiter, W. Willinger, R. Rao Zhang, H.and Kompella, and D.G. Andersen, "csamp: A system for network-wide flow monitoring.," in *5th USENIX Symposium on Networked Systems Design and Implementation (NSDI'08)*, 2008.

[2] M. Moshref, M. Yu, R. Govindan, and A. Vahdat, "Dream: dynamic resource allocation for software-defined measurement," in *ACM Special Interest Group on Data Communication (SIGCOMM'14)*, 2014.

[3] S. Jain, A. Kumar, S. Mandal, J. Ong, L. Poutievski, A. Singh, S. Venkata, J. Wanderer, J. Zhou, M. Zhu, et al., "B4: Experience with a globally-deployed software defined

wan," in *ACM Special Interest Group on Data Communication (SIGCOMM'13)*, 2013.

[4] "Caida anonymized internet traces 2012.," http://www.caida.org/data/passive/passive_2012_dataset.xml.

[5] "The web10g project:taking tcp instrumentation to the next level," http://www.web10g.org.

[6] B. Claise, "Cisco systems netflow services export version 9," 2004.

[7] C. Estan and G. Varghese, "New directions in traffic measurement and accounting," in *ACM Special Interest Group on Data Communication (SIGCOMM'02)*, 2002.

[8] P. Phaal, S. Panchen, and N. McKe, "InMon Corporation's sFlow: A Method for Monitoring Traffic in Switched and Routed Networks," RFC 3176, Sept. 2001.

[9] M. Yu, L. Jose, and R. Miao, "Software defined traffic measurement with opensketch.," in *10th USENIX Symposium on Networked Systems Design and Implementation (NSDI'13)*, 2013.

[10] "Solving the mystery of link imbalance: A metastable failure state at scale," https://code.facebook.com/posts/1499322996995183.

[11] R. Miao, R.l Potharaju, M. Yu, and N. Jain, "The dark menace: Characterizing network-based attacks in the cloud," in *Proceedings of the 2015 ACM Conference on Internet Measurement Conference*, 2015.

[12] M. Al-Fares, A. Loukissas, and A. Vahdat, "A scalable, commodity data center network architecture," *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 4, pp. 63–74, 2008.

[13] A. Greenberg, J. R Hamilton, N. Jain, S. Kandula, C. Kim, P. Lahiri, D. A Maltz, P. Patel, and S. Sengupta, "Vl2: a scalable and flexible data center network," *ACM SIGCOMM computer communication review*, vol. 39, no. 4, pp. 51–62, 2009.

[14] R.R. Kompella, K. Levchenko, A.C. Snoeren, and G.e Varghese, "Every microsecond counts: tracking fine-grain latencies with a lossy difference aggregator," in *ACM Special Interest Group on Data Communication (SIGCOMM'09)*, 2009.

[15] "Mininet," http://mininet.org/.

[16] "Open vswitch website," http://www.openvswitch.org.

[17] "Tcpreplay," http://tcpreplay.appneta.com/wiki/overview.html.

[18] N. Duffield, "Sampling for passive internet measurement: A review," *Statistical Science*, 2004.

[19] F. Uyeda, L. Foschini, F. Baker, S. Suri, and G. Varghese, "Efficiently measuring bandwidth at all time scales," in *8th USENIX Symposium on Networked Systems Design and Implementation (NSDI'11)*, 2011.

[20] N. Handigol, B. Heller, V. Jeyakumar, D. Mazieres, and N. McKeown, "I know what your packet did last hop: using packet histories to troubleshoot networks," in *11th USENIX Symposium on Networked Systems Design and Implementation (NSDI '14)*, 2014.

[21] S. Narayana, J. Rexford, and D. Walker, "Compiling path queries in software-defined networks," in *ACM SIGCOMM Workshop on Hot Topics in Software Defined Networks (HotSDN'14)*, 2014.

[22] C.-W. Chang, G. Huang, B. Lin, and C.-N. Chuah, "Leisure: Load-balanced network-wide traffic measurement and monitor placement," *IEEE Transactions on Parallel and Distributed Systems*, vol. 26, no. 4, 2015.

[23] C. Chaudet, E. Fleury, I.e Lassous, H. Rivano, and M.-E. Voge, "Optimal positioning of active and passive monitoring devices," in *ACM conference on Emerging network experiment and technology*. ACM, 2005.

[24] Y. Gao, W. Dong, W. Wu, C. Chen, X.-Y. Li, and J. Bu, "Scalpel: Scalable preferential link tomography based on graph trimming," *IEEE/ACM Transactions on Networking*, vol. PP, no. 99, 2015.

[25] L. Ma, T. He, K.K. Leung, A. Swami, and D. Towsley, "Monitor placement for maximal identifiability in network tomography," in *IEEE International Conference on Computer Communications (INFOCOM'14)*. IEEE, 2014.

[26] E. Salhi, S. Lahoud, and B. Cousin, "Heuristics for joint optimization of monitor location and network anomaly detection," in *IEEE International Conference on Communications (ICC'11)*, 2011.

[27] K. Suh, Y. Guo, J. Kurose, and D. Towsley, "Locating network monitors: complexity, heuristics, and coverage," *Computer Communications*, vol. 29, no. 10, 2006.

[28] "In-band network telemetry (int).," http://p4.org/wp-content/uploads/fixed/INT/INT-current-spec.pdf.