Intent-Driven Path State Monitoring to Enable Centralized State-Aware Flow Steering

Christoph Hardegen
Department of Applied Computer Science, Fulda University of Applied Sciences, Germany
Email: christoph.hardegen@cs.hs-fulda.de

***Abstract*—Running state monitoring for network switches and links enables the derivation of a path-based data view. Thereby, various monitoring intents like *utilization* and *latency awareness* can be targeted during data collection. Whereas each of these objectives relies on a particular set of state metrics, different monitoring methods may be run to gather the data basis serving as decision input for subsequent analysis purposes. In addition, path conditions have an impact on the state observed for individual packet streams being forwarded along a specific path. While path level state data is of relevance, e.g., to evaluate past load ratios in order to run state-aware and efficient path determination, flow level state helps to monitor *flow experience* conditions like achieved throughput or perceived latency, e.g., to track the compliance with flow-based requirements. This paper presents a modular architecture for path state monitoring that considers port counter query, network probing and in-band network telemetry as methods for demand-driven data collection and focuses on *utilization* and *latency awareness* as monitoring intents. State data is collected by a centralized controller in collaboration with distributed modules deployed in a switch's data plane to run data tracking, wherefore programmable switches are used as operational basis to ensure a flexible monitoring protocol. Evaluations show that continuously collected data snapshots allow to track accurate path state trends that – w.r.t. path state-aware traffic steering – can be leveraged to improve flow-based load distribution across available path capacities and to resolve inefficiencies like imbalanced path load or congestion.**

***Index Terms*—Path Monitoring, Flow Experience, Monitoring Intent, Flow Steering, Programmable Switches**

I. Introduction

The state of existing network routing and forwarding paths is defined by the state of each contained switch and link. Consequently, to analyze a path-based state view, data has to be collected for each network element and afterwards combined to determine state data describing entire paths. Continuous and recurring data collection at periodic time windows allows to obtain current state data snapshots and to derive past trends from respective data sequences, whereby the considered interval affects data granularity. While state can be represented by various metrics, e.g., utilization ratios or latencies, different monitoring methods may be run for data collection. A particular set of metrics describes a specific state context that corresponds to a data acquisition objective like *utilization* or *latency awareness*. These monitoring intents enable demand-driven state data views that can subsequently be leveraged as analysis input for state-aware network management tasks like flow-based routing and forwarding.

In the context of network performance management, obtained state data snapshots and respective trends for path utilization and latency can be used for proactive flow steering or reactive resteering decisions. For example, because Equal-Cost Multi-Path (ECMP) routing does not consider dynamic path states to map arising flows to available path capacities, traffic volumes assigned to certain paths may vary unevenly, leading to inefficient load distribution. To tackle this, state snapshots can be analyzed to determine more sophisticated flow-based forwarding paths, i.e., by running weighted path selection w.r.t. load levels observed during past collection iterations or choosing the least loaded path, resolving conditions like utilization imbalance, high path load or congestion.

In addition, the state of a particular path affects the transport of flows being routed and forwarded on it. Collecting state metrics for single flows, e.g., achieved throughput rates or experienced latency, allows for more granular monitoring of individual packet streams w.r.t. existing path state conditions. As an example, tracked flow properties provide insights required to rate *flow experiences* impacted by determined routing and forwarding decisions, e.g., to evaluate flow requirements and corresponding compliance.

Since programmable switch architectures offer a reconfigurable data plane, path state monitoring strategies can be flexibly deployed while dynamic control is supported through exposed runtime APIs. Whereas being able to track fine-grained state metrics at packet level, approaches are enabled to combine different data collection methods to provide the data basis for downstream analysis tasks. While programmable switch devices support established state monitoring concepts like port counter query, they also enable mechanisms like network probing and In-band Network Telemetry (INT) due to advanced packet processing capabilities.

This paper presents a path monitoring approach entitled `PathMoni` holding the following main contributions:

- An intent-driven strategy for centralized path state tracking is proposed. Monitoring intents such as *utilization* and *latency awareness* are introduced as data collection objectives enabling to consider specific state contexts as decision input for subsequent state-aware analysis. While each intent relies on particular state metrics, port counter query, network probing and INT are run to provide data views, whereby data acquisition is entirely based on raw packet exchange.
- Based on the use-case of centralized, path state-aware flow steering, the benefit of leveraging state data for proactive and reactive path determination decisions is outlined.
- The code base is publicly available to ensure reproducibility.

978-3-903176-51-5

II. Related Work

[1] proposes an architecture to monitor switch and link states. While INT is run for data collection, obtained network state data views are leveraged for centralized traffic scheduling to reactively eliminate experienced congestion. Compared to [1], this paper introduces a modular architecture that is comprised of different data acquisition methods and supports multiple monitoring intents that can be considered for subsequent state-based analysis like efficient flow steering. [2] presents an architecture for intelligent network management. Whereas INT is considered as data collection technique, traffic engineering, which also includes flow steering tasks, is described as use-case scenario. The path monitoring approach presented in this paper can be integrated into the data collection part of this architecture to respect the concept of monitoring intents while supporting different data acquisition methods.

[3] and [4] introduce a network telemetry system that relies on probe packets to collect state data for switches and links. Based on telemetry requirement policies specified for a particular network management application, probes traverse an intended path using source routing to gather corresponding state metrics on it. The data acquisition approach outlined in this paper also includes probing as one collection method to provide path state data views, whereby switch-local probe replication is run to achieve timely full topology coverage. The concept of demand-driven data tracking and gathering is also the same as for [3] and [4], wherefore monitoring intents describing a particular data collection objective are introduced.

Data plane counters are not only used to track switch port states but also to measure per-flow ones [5] [6]. This paper adapts the use of counters to account volume metrics for both link and flow level. While probing and INT are inherently based on raw packet exchange to gather data, port counter queries are also decoupled from runtime APIs such that the entire data acquisition relies on pure packet sharing.

[7] and [8] propose distributed utilization-aware traffic scheduling schemes to improve ECMP. Both methods run at data plane level entirely, whereby [7] selects forwarding paths based on least utilization and [8] applies weighted path determination w.r.t. observed load levels. In addition, approaches like [9] and [10] confirm that a centralized state data view, e.g., based on collected flow statistics or link utilization, helps to resolve inefficiencies that arise from ECMP-based traffic distribution, e.g., to rebalance observed load levels and avoid congestion. While the motivation for efficient traffic load distribution is common with aforementioned works, this paper runs proactive and reactive utilization- and latency-aware flow steering at a centralized controller, which is considered as use-case for centrally collected path state data snapshots.

III. Use-Case: Path State-Aware Flow Steering

A set of programmable switches acting as routers in collaboration with an assigned controller maintain five paths P_1 to P_5 between edge switches $S_{E,1}$ and $S_{E,2}$ connecting multiple host networks and forwarding a variety of flows (Figure 1).

Since each path has one transit hop $S_{T,1}$ to $S_{T,5}$, paths are of equal costs in terms of hops and enable multi-path routing.

The centralized controller platform consists of a Path State Monitor and a Flow Steering Engine. The first runs state monitoring for switches and their links to derive path level views, whereby data is tracked and collected in collaboration with associated switches. The second operates flow-based path determination and programming on affected switches.

First, a routing decision is required if a flow arises and the controller receives a Flow Steering Request from an edge switch (proactive behavior). Second, the controller continuously evaluates obtained path state data at periodic time windows and can send a Flow Resteering Request if non-optimal network state conditions like high load, imbalance or congestion are revealed (reactive behavior).

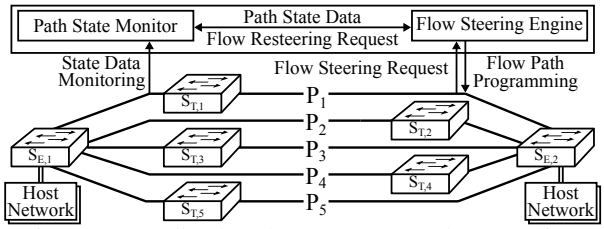


Fig. 1: Centralized Path State-Aware Flow Routing.

Consider a set of flows that has to be forwarded from $S_{E,1}$ to $S_{E,2}$. Since ECMP leverages multiple paths of equal costs but does not respect existing state conditions, the distribution of flow loads across available capacities may be unequal and hence inefficient. As a consequence, in case of experiencing high load on a particular path, alternatives may remain considerably less utilized or even entirely unused (load imbalance).

Running `PathMoni` allows to include obtained state data metrics into the flow steering and resteering process to achieve a more efficient traffic distribution. Collected path states serve as decision basis for a centralized and state-aware flow control that supports multi-path routing using paths of equal cost. Different routing objectives relying on a particular monitoring intent are distinguished while selecting forwarding paths: Whereas observed path load w.r.t. *utilization awareness* is considered as primary routing target, experienced path delay w.r.t. *latency awareness* can be included as secondary objective. The rationale behind this two-staged approach is based on the interdependence of both state metrics.

Also, flow-based state conditions like achieved throughput or perceived latency that arise in response to programmed steering decisions can be measured using `PathMoni`.

IV. PathMoni Approach

A. High-Level System Overview

`PathMoni` performs distributed data tracking and centralized collection to provide path state data views. Therefore, the system architecture is comprised of a central `PathMoni` Monitor that is assisted by `PathMoni` Modules deployed as part of the packet processing pipeline on switches (Figure 2). The monitor combines three sub-monitors each having an individual switch module assigned and operating a particular monitoring method to collect different state metrics.

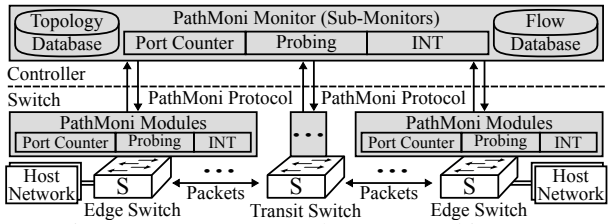


Fig. 2: `PathMoni` Deployment Architecture.

While a *Port Counter* and *Probing Monitor* allow to track path load states enabling *utilization awareness*, the latter also permits to consider latency measures for *latency awareness*. Whereas an *INT Monitor* supports both path monitoring purposes as well, it also allows to track flow level state conditions treated as *flow experience*. The aforementioned objectives are understood as monitoring intents that focus on specific state contexts. Each intent is described by a set of state metrics gathered during data collection, whereby the obtained data basis serves as decision input for state-aware analysis tasks.

Therefore, any combination of the proven monitoring methods port counter query, network probing and INT can be deployed. Each strategy not only enables to collect different state data metrics but also has individual deployment properties that can be respected for certain management scenarios. For example, network probing and INT run in-band, either leveraging explicit packets or using regular ones as carriers implicitly, whereas counter query can be run out-band.

`PathMoni` runs window-based monitoring, whereby data collection is continuously initiated at periodic time intervals. State data obtained during a monitoring iteration corresponds to snapshots that describe a path element at a particular point in time or on average for a continuous timeframe. Consecutive snapshots enable to derive past data trends over time while the used collection window length defines data granularity.

To permit communication between sub-monitors and associated modules, management channels for API calls towards exposed switch runtime environments used for module configuration and for raw packet exchange used for data sharing are established over trusted and secure out-of-band networks. The latter channel corresponds to a direct physical or virtual link, i.e., connected to a programmable device's CPU port, whereas the first only requires socket endpoint connectivity.

Besides the sub-monitors, `PathMoni` Monitor has a *Discovery System* as well as a *Topology* and *Flow Database*. The first handles switch registration and configuration sharing. While the second maintains switch and link states as part of a digraph-based network model, the third holds flow level states.

B. Path State Metrics

Path state data metrics are gathered for each switch and link individually. Afterwards, values are combined to form path level ones. Table I lists the metrics that are supported to respect the monitoring intents of *utilization* and *latency awareness*.

Besides packet timestamps and metadata provided by the switch platform, `PathMoni` relies on tracked counter objects for data collection. Packet and byte counters are maintained for each switch port while separate instances for ingress and egress direction are used. Each directed counter allows to derive the state of the ingress or egress link a particular port belongs to. In addition, in case of tracking flow level state conditions, counter objects hold data for single packet streams.

TABLE I: `PathMoni` State Metrics.

Metric Type	State Metric	State Metric Source	`PathMoni` Sub-Monitor		
			Port Counter	Probing	INT
Raw Values	Packet Count	Switch Counter Objects		✓	
	Byte Count			✓	
	Packet Timestamps	Switch Clock	✗	✓	✓
	Hop Latency	Switch Metadata	✗	✓	✓
Derived Values	Average Throughput / Packet Rate / Bit Rate	Packet & Byte Counter, Data Collection Interval		✓	
	Utilization Ratio	Throughput, Capacity		✓	
	Link Latency	Packet Timestamps	✗	✓	✓

C. Sub-Monitor Operation

1) Port Counter Monitor & Module: While the switch module maintains counter objects based on observed packet data, the monitor collects tracked values for each port on associated switches at a predefined time interval w_{PC}. Based on w_{PC} and counter data, average link throughput rates and utilization ratios for the last collection window are calculated, whereafter combined path level states are derived.

Decoupled from any runtime API, one port counter request packet per switch is asynchronously injected by the monitor to query counters from the module running packet replication and manipulation before returning response packets per switch port. Besides this bidirectional exchange, the module can also push packets containing counter values directly after w_{PC} elapses without previously receiving a request packet.

2) Probing Monitor & Module: At periodic time frames denoted by w_{PB}, the monitor injects probing packets into each associated switch. After the module receives the packet, it is replicated for each egress port to collect local state data and store it in the packet before it is forwarded to a neighbor switch. Accordingly, packet timestamps for ingress and egress processing, measured hop latency as well as maintained packet and byte counter values are determined and added. As soon as the packet arrives at the neighbor switch, it is redirected to the assigned monitor after collecting and adding state metrics for the local ingress port as well. Back at the monitor, link throughputs, utilization ratios and latencies are determined, whereby the latter are computed as difference between the egress and ingress timestamps tracked for successive switches. Subsequently, derived path state metrics for the last probing window are determined using hop and link state data.

Using this local packet replication-based probing method, each probe traverses exactly one link and two connected switches. While the first switch acts as *probing source* and the second as *probing sink*, both run *probing transit* behavior.

3) INT Monitor & Module: Besides state data tracking, the module runs the following role-specific processing behavior:

An *INT source* decides on whether to apply INT-based data collection to a packet or not, whereby only packets that ingress on host network ports are considered. Therefore, an INT flag is set in packet headers to indicate subsequent INT processing. An INT duration timer w_{INT} that defines a recurring collection interval serves as trigger mechanism. Thus, each time the timer expires, state data is collected for the

path or flow that a particular packet is forwarded on or belongs to respectively. For this, the INT duration timer is individually maintained on each egress port or observed flow. While the first are identified by port IDs, hashes that are computed based on a flow's 5-tuple data are used for tracking of the latter.

An *INT transit* processes incoming INT packets and writes local state data to it before forwarding the packet to a neighbor. Thereby, ingress and egress port counters as well as packet timestamps and experienced hop latency are recorded.

After reaching an *INT sink*, the regular packet is replicated to build an *INT report* including collected state data that is shared with the associated monitor while previously added INT data is removed from the original packet before it is forwarded to the intended destination. At the monitor, report data is used to determine link latencies, throughputs and utilization ratios before combined path state data metrics are derived.

V. Path State-Aware Flow Steering

First, the approach acts proactively, whereby maintained path state views, i.e., trends reflected in the last collected data snapshot, are analyzed and respected each time a flow steering decision is required (Algorithm 1). Path determination depends on the chosen routing target: For *utilization awareness*, either the least loaded path is selected or a weighting method w.r.t. utilization levels included in the last state snapshot is applied (weighted ECMP). In case of considering both *utilization* and *latency awareness*, a candidate set of paths having similar load is determined, whereby ΔU states the permitted deviation level w.r.t. the least loaded path, whereafter the one ensuring smallest latency is selected from remaining candidates.

Algorithm 1 Proactive Path State-Aware Flow Steering.

```
get simple paths between source & destination switch
query path state views from maintained data
if utilization and not latency awareness then
    select least loaded path or run weighted ECMP
if utilization and latency awareness then
    obtain utilization ratio of least loaded path
    for all path candidates do
        if load ratio exceeds ΔU w.r.t. least loaded path then
            filter path from candidates
    select path with least latency from remaining candidates
program flow path on affected switches
```

On the one hand, the primary steering target respects path utilization ratios and aims at balanced and thus efficient load distribution across available path capacities. On the other hand, the secondary target additionally respects latency conditions.

Second, since proactive flow steering may still result in imbalanced load levels for alternative paths, the approach also operates reactively in response to high path utilization ratios (Algorithm 2). To handle these potential inefficiencies and avoid or solve congestion scenarios, each time a state snapshot is collected, the included data view is evaluated to reveal paths whose load levels exceed the threshold T_U.

Flow metadata is assumed to be available for analysis, which can be accomplished by running a flow monitoring system.

Algorithm 2 Reactive Path State-Aware Flow Resteering.

```
query switch link utilization from maintained data
rank links w.r.t. load ratios in descending order
for all ranked switch links do
    if link utilization ratio > T_U then
        query flow metadata for streams traversing the link
        sort flows based on throughput in descending order
        for all sorted flows do
            get simple paths from source to destination switch
            rank candidates based on load in ascending order
            if least loaded path has sufficient capacity and
               no overload w.r.t. T_U results then
                if utilization and not latency awareness then
                    select path with least utilization as alternative
                if utilization and latency awareness then
                    get paths with close load w.r.t. ΔU and
                    able to handle shifted load w.r.t. T_U
                    select path with least latency as alternative
                program alternate flow path on affected switches
                update load ratio for links of old and new path
            if reduced link utilization ≤ T_U then
                continue analysis with next switch link
```

VI. Evaluation

A. Experimental Environment

The network given in Figure 1 is deployed using behavioral model version 2 (bmv2) as virtual switch. Python scripts run the behavior of sub-monitors and a flow routing controller, whereas switch modules are developed as $P4_{16}$ program[1]. The topology is reconstructed as digraph used to initialize the *Topology Database* implemented using NetworkX library while a dictionary-like object serves as *Flow Database*.

A CPU port link and exposed runtime APIs, i.e., thrift and gRPC message interfaces, are leveraged as management channels between monitors and modules. While Scapy library is used for packet sniffing and parsing as well as construction and sending operated at monitors, a runtime library enables module configuration. Traffic profile generation and replay are implemented in separate scripts with iperf being used to generate flows with constant rates respecting mean throughputs.

Besides the proof-of-concept implementation framework, a script-based simulation approach was developed due to limited bmv2 performance. While the testbed is sufficient to run exemplary experiments for the proactive and reactive flow routing methods (Section VI-B3), the decoupled scripts allow a more scalable evaluation for the proactive one (Section VI-B2).

B. Experiments

1) Path State and Flow Experience Tracking: Port counter query and probing are run to monitor path load and latency trends, wherefore intervals w_{PC} and w_{PB} are set to 1 s. While a simple traffic pattern is replayed, flow hash-based ECMP is used to distribute flows across paths P_1 to P_5 that have artificially imposed delays of 100 to 500 ms. The traffic profile

[1] https://gitlab.cs.hs-fulda.de/flow-routing/cnsm2022/path_monitoring

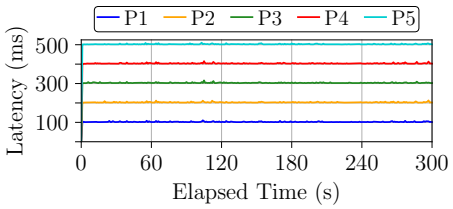


Fig. 4: Path Latency Monitoring (Probing).

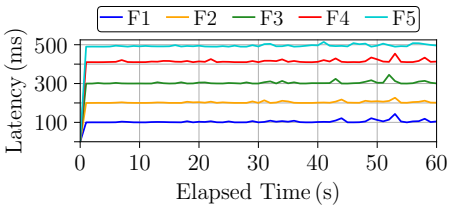


Fig. 5: Flow Latency Monitoring (INT).

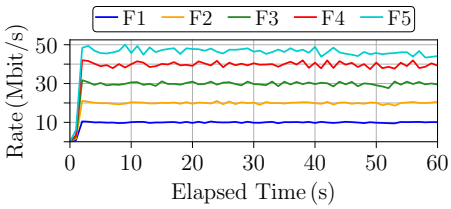


Fig. 6: Flow Throughput Monitoring (INT).

specifies a constant rate of 10 flows/s for an experiment time of 300 s, whereby duration and volume properties are sampled from an exponential decreasing distribution with mean values of 10 s and 10 Mbit/s. Though parameters are exemplary, they are sufficient to prove the feasibility of sub-monitor methods.

As Figures 3 and 4 illustrate, path load and latency are precisely tracked over time. Thus, `PathMoni` sub-monitors enable an accurate data view for path state-aware analysis.

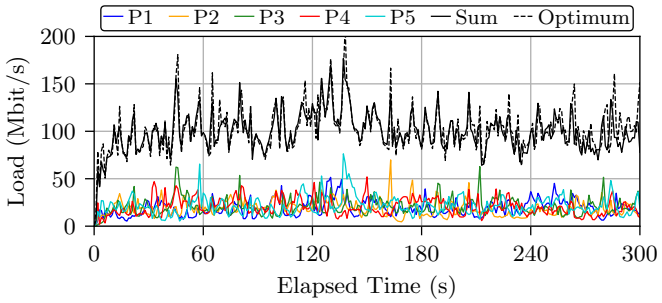


Fig. 3: Path Utilization Monitoring (Port Counter Query).

In addition, INT is run to exemplarily measure the flow experience in terms of achieved throughput and observed latency, whereby w_{INT} is specified as 1 s. Therefore, 5 flows F_1 to F_5 are bound to respective paths P_1 to P_5 using an ECMP round robin distribution strategy. Again, delays of 100 to 500 ms are imposed on the paths. Flows are generated with intended throughput rates of 10 to 50 Mbit/s.

Figures 5 and 6 show that `PathMoni` sub-monitors not only allow to track path- but also flow-related state conditions. Since flows with higher throughputs are affected by increased delays, measured trends outline a more instable behavior.

Using data collection intervals of 5 and 10 s is associated with coarser-grained trends hence impacting data granularity. Running probing and INT to track path loads provides similar results compared to port counter query. Likewise, running INT for latency estimation delivers similar outcomes as probing.

2) Proactive Path State-Aware Flow Steering:

Flow Set Generation

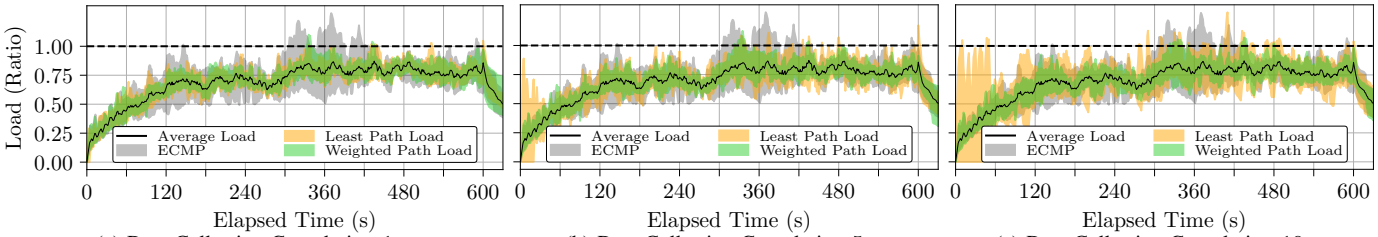
To respect real-world traffic characteristics for flows fed as analysis input, the following steps are performed to build a realistic flow set: First, about 58 mio flows are extracted from two flow datasets that were collected during capture periods of one week at central switches in the data center of a university campus network [11] [12]. Therefore, nearly equal shares of flow data describing packet streams observed during day- and nighttime (starting 2 pm and 2 am respectively) are selected for each weekday in the capture periods and afterwards merged to form a comprehensive and diversified flow set. Second, to focus on multi-packet flows, those that transmit just one packet and thus do not have a duration and average throughput rate as well as ones individually exceeding assumed path capacity limits of 1 Gbit/s are filtered. As a consequence, the overall set is reduced to approximately 17 mio flow data samples.

The rate of flows that are generated each second follows a Poisson distribution, whereby the considered mean is set to 4 000 flows/s. While this is the approximate average number of exported flow data records per second observed during peak times at daytime on most weekdays in the aforementioned university network [11], it allows to estimate the arising flows per second in a practical network environment. At nighttime, this rate drops to about 500 such that a mix of both relates to ≈2 250 flows/s. Having the flow count to be replayed each second, the starting times of individual flows are evenly distributed within the interval of 1 s. Flow 5-tuple data, duration and volume as well as derived average throughput properties are sampled from the above flow set. Because an experiment lasts 600 s, about 2.4 mio flows are taken from the overall set. Since an experiment is run 10 times, 10 diversified subsets are extracted based on individual randomization seeds. Although these sets may have a particular proportion of common flows included, their order of occurrence regarding assigned starting times varies and thus preserves decent diversity.

Flow Set Distribution

Obtained flow sets are replayed and distributed across 5 non-overlapping paths. Besides selecting the least loaded path and running weighted ECMP w.r.t. utilization trends observed in gathered path state data snapshots, flow hash-based ECMP is run as reference for comparison. During analysis, the data collection interval is varied between 1, 5 and 10 s to evaluate different granularity levels for utilization-aware path determination. Outcomes from 10 experiment iterations with separate flow sets are averaged, hence providing aggregated and smoothed results. To enable comparison, the minimum and maximum path load levels over time are determined and their derived differences illustrated as trend in Figure 7 for each considered flow distribution strategy separately. In addition, Table II summarizes mean, median and standard deviation scores for the measured differences over time.

Compared to ECMP results, taking past utilization with a data collection granularity of 1 s into account for flow-based path determination helps to significantly balance load levels on average over time and to reduce the occurence of congestion. While closer path saturation is ensured, choosing the least loaded path and leveraging the utilization-aware weighting method achieve quite similar trends. As a consequence, respecting tracked utilization during path determination enables to distribute arising flow loads more evenly and efficiently.



(a) Data Collection Granularity: 1 s. (b) Data Collection Granularity: 5 s. (c) Data Collection Granularity: 10 s.

Fig. 7: Minimum & Maximum Path Load Ratio Differences Over Time.

First, for weighted path determination, increasing the time frame between obtained state data snapshots to 5 or 10 s shows quite stable outcomes with slight variations compared to utilizing an interval of 1 s. Second, for selecting least loaded paths, increased windows provide more instable and hence imbalanced path load trends. This is due to the fact that the higher the interval, the higher the burstiness w.r.t. putting high load on a single path because only one path is considered for all decisions until the next snapshot becomes available.

TABLE II: Minimum & Maximum Path Load Differences (%).

Score	ECMP	Least Path Load			Weighted Path Load		
		1s	5s	10s	1s	5s	10s
Mean	28.34	15.01	23.03	33.80	15.80	16.80	18.24
Median	26.54	14.35	21.29	29.75	14.55	15.94	17.37
Std Dev	12.33	6.04	11.02	17.25	6.80	6.55	7.13

Although the considered flow rate and the mixed data proportions from both day- and nighttime correspond to the more challenging scenario, running above experiments for both times of day separately using respective data and individually related flow rates provides similar result trends though with varying dimension. The same applies to running the experiments not for data from multiple but single capture days.

3) Reactive Path State-Aware Flow Resteering: State data collection is run at periodic intervals of 10 s and subsequently triggers utilization snapshot analysis. Whereas path capacities are intentionally limited to 100 Mbit/s, an individual set of 4 flows is generated after 10 and 30 s likewise to cause exemplary aggregated traffic levels (Figure 8). In the first batch, flows F_1 to F_4 having constant throughputs of 30, 10, 40 as well as 10 Mbit/s are distributed and bound to paths P_1 to P_4 respectively. The resulting load ratios remain below the considered threshold $T_U = 0.9$. Once the second batch with flows F_5 to F_8 having intended rates of 70, 10, 60 as well as 10 Mbit/s is started and again distributed across P_1 to P_4, P_1 and P_3 experience high load since capacity is entirely filled.

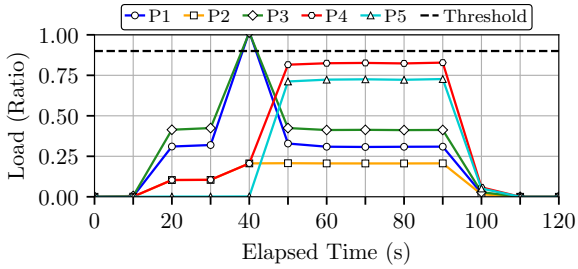


Fig. 8: Path Load Ratios for Reactive Flow Resteering Scenario.

After utilization analysis is run at 40 s, arising congestion on both paths is discovered. Hence, flows F_5 and F_7 are placed on alternative paths P_5 (that is entirely unused) and P_4 (that is less utilized) able to carry the new flow load.

As a result, the evaluation of obtained path state data snapshots and downstream reactive flow resteering allow to reveal paths experiencing high utilization and partly shift observed traffic loads to alternatives if possible. This enables to balance load differences and to avoid or resolve congestion conditions, which in turn is also associated with improved flow experiences w.r.t. forwarded packet streams.

VII. Discussion

`PathMoni` is an intent-driven approach for centralized path state monitoring. The inclusion of different sub-monitors allows to track various state metrics at switch and link level that are combined to form path level metrics. In addition, flow-based state conditions that apply as packet streams are forwarded along topology paths can be measured to evaluate the observed flow experience. During data acquisition, state metrics are collected with regard to monitoring intents that focus on a specific state context like *utilization* or *latency awareness*. While an intent enables state-aware analysis for particular network management tasks, the corresponding data basis serves as decision input. Since, besides a modular monitor architecture, programmable switch platforms are used as operational basis to support centralized controller operation, a flexible state monitoring protocol is ensured. This way, the set of tracked state data can be dynamically adapted to respect further monitoring intents like *packet loss* or *drop awareness*.

As `PathMoni` runs state data tracking based on maintained counter objects, packet processing is slightly delayed.

Because metrics are gathered at regular intervals, the used window length affects data granularity available for subsequent snapshot analysis. Experiments for different switch and topology sizes, i.e., port and device counts, confirm that timely data collection at second scale is possible for both path and flow level tracking while running `PathMoni` sub-monitors.

`PathMoni` leverages direct and raw packet exchange between a monitor and switch module for data collection. While this is a lightweight and portable solution, i.e., decoupled from any particular runtime API, reliable operation depends on efficient packet sniffing and processing behavior at controller level. Whereas probing and INT are based on this data retrieval method inherently, for port counter query, compared to using runtime APIs, this pure packet sharing reduces the number of exchanged messages on the control path and provides stable query times for increased switch port counts. For network probing, compared to traversing a Eulerian path, the included local probe replication-based strategy enables stable query times for increased switch counts. At the same time, the

number of individual probing packets that have to be injected and subsequently processed at the monitor rises. This again requires scalable processing capabilities at the monitor to ensure reliable operation and thus consistent state data views.

The centralized path state-aware flow steering approach outlines a clear network management use-case that is based on collected state data views. Evaluation experiments confirm the individual benefit of the proactive and reactive behavior:

First, leveraging collected path state data snapshots for utilization-aware path determination enables more sophisticated steering decisions that allow to balance arising traffic loads more efficiently across available path capacities. Thereby, the granularity of obtained state data trends affects the distribution quality w.r.t. resulting path utilization ratios. As a consequence, the data collection interval has to be specified as low as possible to ensure stable and balanced path load levels. At the same time, running centralized path selection and programming is associated with additional delay. Whereas initial packet redirection and intermediate processing at the controller contribute to this, the latter can be optimized by efficient control flow implementation. However, streams may be subject to slight proportions of initial packet drops until a forwarding path is available.

Second, the reactive operation in response to high path load allows to resolve or avoid congestion and to rebalance inefficient traffic load distributions that still may stem from initially derived proactive decisions. Shifted path loads also contribute to increased flow experiences for packet streams that are forwarded along intensively utilized paths. One challenge of leveraging flow data to identify streams that offer potential for efficient resteering is that available data views may be distorted in case of congestion, hence moving flows might lead to overload on alternative paths as well. Also, shifting particular flows enables others that were previously slowed down due to high utilization to increase their rate which may again cause congestion. This calls for an early analysis of potential overload conditions, which can be achieved by specifying the load ratio threshold accordingly.

Although performed evaluation experiments for proactive and reactive flow steering just consider the primary routing objective of *utilization awareness*, a combined approach that also takes *latency awareness* as secondary one into account is enabled. Therefore, `PathMoni` provides the data basis that allows to respect both metrics and corresponding conditions during path determination and subsequent programming.

In addition, the centralized flow steering approach can be combined with distributed schemes to form a hybrid solution: On the one hand, header criteria- or hash-based filtering can be applied at switch level to selectively request proactive routing decision support for application types that are subject to a particular monitoring intent. This differentiated behavior enables to ensure packet forwarding over a certain path while respecting best possible flow conditions and tracking their compliance. On the other hand, continuous state evaluation and reactive traffic resteering can help to balance occurring inefficiencies like incompliance with flow requirements.

VIII. Conclusion and Future Work

`PathMoni` enables an intent-driven monitoring approach to collect path-based state data views. While state metrics are collected as periodic data snapshots, gathered data trends serve a particular monitoring intent considered as decision input for subsequent state-aware data analysis. Therefore, a controller architecture combining the data acquisition strategies of port counter query, network probing and in-band network telemetry is introduced. Centralized sub-monitors run a flexible data tracking and collection protocol in conjunction with distributed modules deployed in a switch's data plane. While granular monitoring is ensured, data sharing is based on raw and direct packet exchange. Besides path level, state conditions for packet streams at flow level can be tracked and evaluated.

For example, obtained state data can be leveraged for centralized path state-aware flow steering: First, proactive path determination based on past state trends to distribute arising traffic load across available path capacities efficiently is enabled, i.e., by selecting the least loaded path or applying a utilization-aware weighting method. Second, reactive resteering in case of experiencing high load or congestion on particular paths helps to rebalance or resolve these inefficiencies. Similar state-aware analysis is possible while respecting not only utilization but also latency measures in a combined manner.

`PathMoni` is planned to be ported onto hardware switches run in a real-world network to analyze operational behavior.

References

[1] J. Geng, J. Yan, Y. Ren, and Y. Zhang, "Design and Implementation of Network Monitoring and Scheduling Architecture Based on P4", *ACM International Conference on Computer Science and Application Engineering*, 2018.

[2] J. Hyun and J. W. K. Hong, "Knowledge-Defined Networking using In-band Network Telemetry", *Asia-Pacific Network Operations and Management Symposium*, 2017.

[3] Z. Liu, J. Bi, Y. Zhou, Y. Wang, and Y. Lin, "NetVision: Towards Network Telemetry as a Service", *IEEE International Conference on Network Protocols*, 2018.

[4] Y. Lin, Y. Zhou, Z. Liu, K. Liu, Y. Wang, M. Xu, J. Bi, Y. Liu, and J. Wu, "NetView: Towards On-Demand Network-Wide Telemetry in the Data Center", *IEEE International Conference on Communications*, 2020.

[5] Y. Li, R. Miao, C. Kim, and M. Yu, "FlowRadar: A Better NetFlow for Data Centers", *USENIX Symposium on Networked Systems Design and Implementation*, 2016.

[6] L. Castanheira, R. Parizotto, and A. E. Schaeffer-Filho, "FlowStalker: Comprehensive Traffic Flow Monitoring on the Data Plane using P4", *IEEE International Conference on Communications*, 2019.

[7] N. Katta, M. Hira, C. Kim, A. Sivaraman, and J. Rexford, "HULA: Scalable Load Balancing Using Programmable Data Planes", *ACM Symposium on SDN Research*, 2016.

[8] J.-L. Ye, C. Chen, and Y. Huang Chu, "A Weighted ECMP Load Balancing Scheme for Data Centers Using P4 Switches", *IEEE International Conference on Cloud Networking*, 2018.

[9] M. Al-Fares, S. Radhakrishnan, B. Raghavan, N. Huang, and A. Vahdat, "Hedera: Dynamic Flow Scheduling for Data Center Networks", *USENIX Symposium on Networked Systems Design and Implementation*, 2010.

[10] O. M. Mon and M. T. Mon, "Flow based Traffic Scheduling in Software Defined Network", *IEEE International Conference on Advanced Information Technologies*, 2020.

[11] C. Hardegen, B. Pfülb, S. Rieger, A. Gepperth, and S. Reißmann, "Flow-based Throughput Prediction using Deep Learning and Real-World Network Traffic", *IEEE International Conference on Network and Service Management*, 2019.

[12] C. Hardegen, B. Pfülb, S. Rieger, and A. Gepperth, "Predicting Network Flow Characteristics using Deep Learning and Real-World Network Traffic", *IEEE Transactions on Network and Service Management*, 2020.