Prediction-based Flow Routing in Programmable Networks with P4

Christoph Hardegen and Sebastian Rieger
Department of Applied Computer Science, Fulda University of Applied Sciences, Germany
Email: {christoph.hardegen, sebastian.rieger}@cs.hs-fulda.de

***Abstract*—Distributing the load of arising network flows with varying characteristics across available topology links and paths efficiently and up to capacity is challenging. To allow for an optimized (e.g., equalized) utilization of multiple links or paths, a flow routing paradigm and architecture leveraging topology state snapshots, predicted flow characteristics and data plane programmability is proposed. This flow steering strategy is compared to existing shortest and multi-path routing methods. Moreover, enhancements to increase the practical applicability of the proposed solution are discussed.**

***Index Terms*—Traffic Engineering, Flow Routing, Programmable Data Plane, P4, Machine Learning, Flow Prediction**

I. Introduction

Effective routing strategies to distribute traffic across links or paths in a network topology efficiently are a cornerstone in computer networks. While established routing techniques (e.g., link state, path vector) primarily use static metrics, more fine-grained flow-based routing is necessary to distribute load over multiple paths evenly and use available capacities up to a maximum. Besides an improved overall network utilization to prevent high load or congestion, optimized load distribution is beneficial for individual flows (latencies and throughput rates).

Traffic control and estimation on network flow level have evolved as a result of modern traffic engineering, e.g., leveraging Software-Defined Networks (SDN) [1]. Mechanisms got even more fine-grained and flexible with programmable data planes, e.g., using Programming Protocol-independent Packet Processors (P4) [2]. Machine learning (ML) allows to detect and predict patterns in flow data. It can be utilized to derive flow characteristics and enable appropriate traffic steering.

This paper proposes *Prediction-based Flow Routing* (*PFR*), a centrally controlled architecture and routing strategy for flows in programmable networks. Topology state snapshots and flow predictions are leveraged to distribute flows efficiently over available links and paths. Experiments are performed using a prototype obtaining results for multi-path (MP) routing scenarios. Benefits are discussed while comparing the approach to other network flow distribution methods.

Using multiple paths efficiently serves as main motivation. A common method for load sharing and fault tolerance is Equal-Cost Multi-Path (ECMP) [3]. Applied round-robin, random or hash-based distribution strategies reach their limits, especially as flow characteristics vary. As network components (stateful middle boxes) depend on communication states, a flow's packets may need to be forwarded on the same path. Thus, reactive flow steering is not always possible. "Suitable" paths need to be determined proactively before any data is exchanged, which is challenging as flow characteristics like throughput and duration are not known ahead of time.

II. Related Work

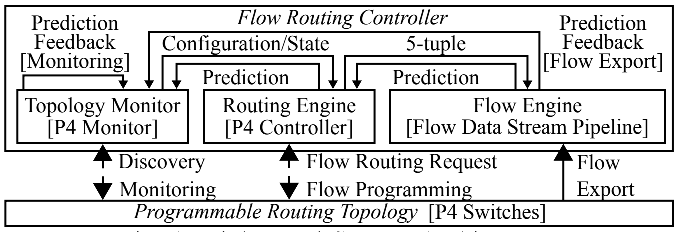
The application and value of ML for network traffic prediction and engineering is focused in several publications and surveys [4] [5]. While [6] and [7] propose concepts for the prediction of flow characteristics, this paper focuses on a flow-based routing strategy to optimize overall link and path utilization leveraging the prediction results. Optimal flow routing is also described in [8] and [9]. The use of machine learning for routing and traffic prediction is also covered in [10] and [11]. However, [10] is limited on using binary classification (elephant/mice flows). [11] focuses on aggregated path traffic and network modeling while not including P4-based flow routing. Fine-grained flow steering holds a significant challenge for network performance, e.g., due to latency added by routing decision support and flow prediction. P4 [2] enhances flow-based routing in SDN by enabling programmable data plane logic and low-latency traffic management. [12] presents the use of P4 for optimized flow routing at Internet Exchange Points, again limited to binary classification of flows (elephant/mice) and ML only being named as possible future work. [13] and [14] describe the use of P4 for flow-based ECMP routing in data centers, also not including traffic prediction. [15] moves in this direction but offers a more general approach, e.g., also covering network security, without presenting a prototype environment and its evaluation.

III. Prediction-based Flow Routing

PFR is a flow routing paradigm that aims at distributing traffic efficiently (nearly evenly) over links/paths to avoid high load/congestion. Conditions for flows can be improved by minimizing observed latency/maximizing required throughput.

A. High-Level System Architecture and Workflow

The *Programmable Routing Topology* (Figure 1) is built of *P4 Switches*. Programmable switches are used as they allow a flexible deployment of solutions at the network level enabling operations for *PFR* at the controller level. The logically centralized *Flow Routing Controller* has three modules:


Flow Routing Controller
Prediction Feedback [Monitoring]
Configuration/State
5-tuple
Prediction Feedback [Flow Export]
Prediction
Prediction
Topology Monitor [P4 Monitor]
Routing Engine [P4 Controller]
Flow Engine [Flow Data Stream Pipeline]
Discovery
Monitoring
Flow Routing Request
Flow Programming
Flow Export
Programmable Routing Topology [P4 Switches]
Fig. 1: High-Level System Architecture.

A *P4 Monitor* acts as *Topology Monitor* continuously collecting configuration and state data of deployed *P4 Switches* (*Discovery/Monitoring*) to maintain a state model. A *Flow Engine* implements a *Flow Data Stream Pipeline* [6] [7] collecting flow data (*Flow Export*) and maintaining a prediction model to predict flow characteristics. A *P4 Controller* acts as *Routing Engine* that handles routing requests (*Flow Routing Request*) and performs flow programming accordingly (*Flow Programming*). *PFR* evaluates the maintained topology state (links/paths) combined with predicted flow characteristics to select "suitable" paths to forward a flow's packets based on a predictive topology state. Exported flows and collected path data snapshots allow for a subsequent and continuous update of the state to correct prediction errors (*Prediction Feedback*).

B. System Architecture Modules and Operation

1) Topology Monitor: The *Topology Monitor* (*P4 Monitor*) maintains a topology state model and is built of *P4 Port Counter*, *P4 Probing*, *P4 INT* and *Discovery* instances. *P4 Port Counter* and *Probing* collect load (utilization ratio) and latency metrics for each link and path at fixed time intervals (snapshots). A *P4 INT Monitor* performs flow monitoring (tracking observed flow experiences). *P4 Switches* register themselves to support discovery and link availability monitoring. A *Discovery Monitor* handles corresponding registration requests and polls switch availability/configuration.

2) Flow Engine: A *Flow Data Stream Pipeline* [6] [7] consumes exported flows as input. An ML model (Deep Neural Network [6] [7]) is continuously trained on incoming data while treating the prediction of flow characteristics as multi-class classification problem. As forecasting is carried out as flows start, only features known ahead of time are usable. Besides a continuous model update, an interface to request a prediction for flow 5-tuples is offered. In this paper, predictions are assumed to be provided in advance.

A *P4 Flow Monitor* collects flows streamed from switches.

3) Routing Engine: The *Routing Engine* (*P4 Controller*) receives *Flow Routing Requests*. A flow's 5-tuple is extracted and passed to the *Flow Engine*, which provides the likely throughput and duration. Also, the latest topology state snapshot describing link/path load and latency is requested. A "suitable" flow path is determined and programmed on affected switches (*Flow Programming*). If no path is available, the overall least loaded one is selected providing as much capacity as possible to forward the flow's packets.

Knowing likely flow durations and throughputs ahead of time allows a predictive management of the topology utilization. Thus, prediction results are used to update path load state. Each time an up-to-date snapshot is available, flow data is received or a flow's probable duration is reached and no further activity is observed, this state is updated/corrected accordingly.

4) P4 Switches: Each switch performs the following tasks:

- manage port counters, process probe packets and INT data
- redirect first packet of a flow, report flow metadata, forward a flow's packets according to programmed paths
- track observed packets and export flow data

IV. Experimental Setup

A. Assumptions for the Experiments

- Topology state is represented by link/path utilization only.
- Network experience is not evaluated for individual flows.
- Flows last for an entire experiment.
- Likely throughputs are assumed to be provided in advance.

Thus, "suitable" paths are determined only based on a flow's predicted throughput and collected link/path load snapshots.

B. P4Environment

P4Environment (*P4Env*) [16] is a framework we developed that is based on [17] and used as testbed for the experiments. *P4Env* provides a *Topology Builder* and *Runner*, *P4 Switches* (bmv2 [18]), *P4 Hosts*, *P4 Controllers* and *P4 Monitors*.

Simulated flows are generated based on prespecified *Traffic Profiles*. *P4Env* provides a *Traffic Profile Generator* leveraging iperf [19] and a *Traffic Manager* to replay flows from *Traffic Profiles*. To support reproducability for flow generation and replay, the processes use seeds for pseudo-random operations.

C. Implementation of Prediction-based Flow Routing

1) Flow Forwarding Program: Each *P4 Switch* has a CPU port and gRPC interface (*Management Channel*, *P4 Runtime*).

a) Ingress Processing: Two hashes of a flow's 5-tuple are computed for each IP packet (Cyclic Redundancy Check 16/32 bit). A flow forwarding table matches on these hashes (*exact match*) to determine the egress port. If a packet belongs to an unknown flow (table miss), it is redirected to the *P4 Flow Forwarding Controller*. A bloom filter (*indirect register*) and the hashes (position indexing) are used to decide whether a packet belongs to a known flow or not. Subsequent packets are dropped until a path is programmed locally. If a packet is part of a known flow (table hit), it is forwarded based on the forwarding table. Link layer addresses, the time-to-live and the IP header checksum are updated (hop-by-hop principle).

b) Egress Processing: If a packet is intended to be redirected via the CPU port, metadata is added (ingress port, flow hashes). Otherwise, egress port counters (*indirect counters*) are updated as these statistics are used to track link load.

2) P4 Port Counter Monitor: A graph-based topology model is maintained and implemented using NetworkX [20]. The port counters (byte count) are obtained for each link (egress switch port) through the gRPC interface of the switches at a fixed time interval (10 s). Byte counters are used along with timestamps to maintain a link and path load model.

3) P4 Flow Forwarding Controller: Asynchronous sniffing on the CPU port of each *P4 Switch* is performed using Scapy [21]. A flow's 5-tuple and provided metadata are extracted from received packets. A forwarding path is determined and programmed by adding an entry to the flow forwarding table on affected *P4 Switches* through their gRPC interface.

D. Multi-Path Flow Routing Use-Case

1) Topology: All simple paths P_1 to P_5 between the routers R_1 and R_7 have uniform capacity and equal cost (Figure 2).

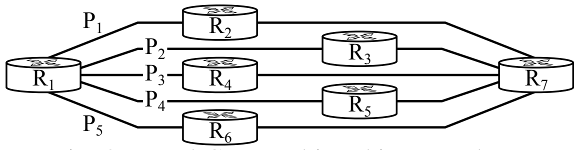


Fig. 2: Equal-Cost Multi-Pathing Topology.

2) Flow Routing Strategies:

- *Shortest Path Flow Routing*: Using hop count metric for path determination. A single path is selected if costs are equal.
- *ECMP Flow Routing*: Selecting a path based on the hash of a flow's 5-tuple, at random or in a round robin manner.
- *Path Load Snapshot Aware Flow Routing*: Analyzing periodically collected path load. The least loaded path is selected for flows in the time frame between consecutive snapshots.
- *PFR*: Path load snapshots and median values of the throughput classes assigned to flows are used for path determination.

Each strategy is controlled centrally for time scale comparison.

3) Traffic Profile: A uniform link capacity of $1\,\mathrm{Mbit/s}$ is employed. Flows belong to UDP traffic between hosts connected to R_1 and R_7. Load of flows remains steady during an entire experiment. Each of the 50 UDP flows has a throughput between $15\,\mathrm{Kbit/s}$ and $165\,\mathrm{Kbit/s}$, which is related to $15\,\%$ of the link capacity. The lower limit is $15\,\mathrm{Kbit/s}$ due to a boundary for bandwidth throttling in iperf. *Traffic Profile* parameters (number of flows, throughputs) ensure the topology is not used beyond capacity and overall path capacities are theoretically sufficient for arising flow loads. While link speeds are intentionally low, relativity of results is preserved.

Ten classes for a flow's throughput are evenly defined over the selected fraction of the link capacity. The more classes, the more fine-grained the flow routing but also the more challenging the flow prediction. First, the number of flows is distributed over the fraction to match a nearly "inverted" Gaussian normal distribution to form a challenging MP scenario. Considerably more flows with low or high than medium load is also a common practical scenario. Second, flow throughputs are randomly selected from the space belonging to the assigned class. The overall flow volume is $\approx 4.5\,\mathrm{Mbit/s}$ for the actual and for the median throughputs for the classes assigned to each flow ($<$ topology capacity of $5\,\mathrm{Mbit/s}$). Flows are replayed in batches of 5 in an interval of $10\,\mathrm{s}$. A batch is generated after a monitoring iteration, in which a load snapshot is obtained.

While a flow is replayed using its generated throughput value, flow prediction provides the median of the value range for a class label assigned to the flow. Thus, this representative value is used to maintain predictive path load states.

V. Experiments

Experiments were run 10 times. Mean results were obtained.

A. Flow Routing

1) Shortest Path Flow Routing (Router Hops): Paths have an imbalanced utilization (Figure 3a). Flows are routed over P_2 (single shortest path), which is under high load. Its capacity is not sufficient. P_1 and P_3 to P_5 with equal cost remain unused.

2) ECMP Flow Routing (Flow Hashes): Path utilization trends are instable because load ratios vary significantly over time (Figure 3b). Final path loads are not nearly close to each other. Two paths are fully loaded. Others have remaining capacities. One path is even only about half-way utilized.

3) ECMP Flow Routing (Round Robin): Path load trends are less instable but final load is not nearly evenly distributed over available path capacities (Figure 3c). A round robin strategy is a rather static method and not suitable in practice.

4) ECMP Flow Routing (Random): Flows are not distributed efficiently across available paths over time, leading to imbalanced utilization (Figure 3d). While three paths have a high load, two are considerably less loaded. A pseudo-random strategy is not deterministic and not suitable in practice.

5) Path Load Aware Flow Routing (Snapshots): Final path utilization is not nearly close for all paths and the trends highly vary over time (Figure 3e). This is due to using only least loaded paths for arising flows between subsequent snapshots.

6) Prediction-based Flow Routing: Flows are distributed efficiently across available paths (Figure 3f). Utilization trends are quite stable and final path loads are close to each other.

Flow prediction is associated with potential errors (varying accuracy levels). As prediction results are used to maintain predictive path states between collected utilization snapshots, errors impact the distribution of flows within this time interval. Newer collected load snapshots are assumed to correct errors in the predictive state for further routing decisions.

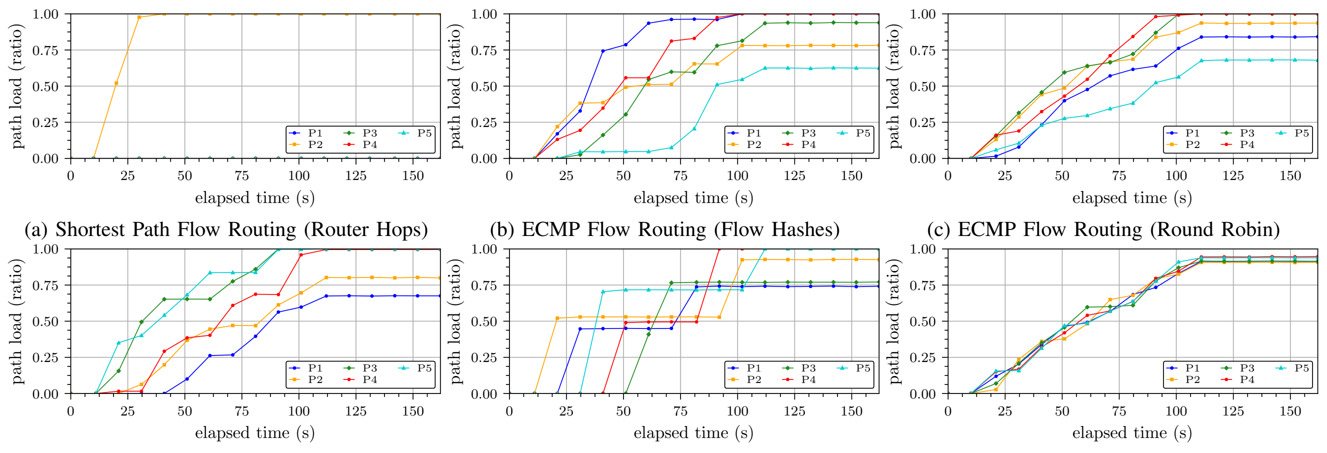
The relative error is of interest because routing flows while assuming too much or not enough capabilities has a negative impact on the predictive state. Even an accuracy of $100\,\%$ holds little error because *PFR* works with the median values of a predicted class associated with a flow.

TABLE I: Overview of Final Path Loads (%).

Result	**Shortest Path**	**ECMP Strategies**			**Path Load**	**PFR**
		Hash	**Round Robin**	**Random**		
Minimum	0.00	61.38	66.34	66.43	74.06	89.53
Maximum	100.00	100.00	100.00	100.00	100.00	93.47
Mean	20.00	86.30	88.34	88.82	87.92	91.45
Median	0.00	92.01	92.60	99.06	92.61	90.87
Volume (Mbit/s)	1.00	4.31	4.42	4.44	4.40	4.57

Table I summarizes the final path utilization ratio results. Mean loads after the tenth monitoring iteration are shown. All flows were replayed at this time. Not only path load trends vary significantly over time, but also final path utilization.

Using another profile (similar distribution for 70 flows distributed over the first $10\,\%$ of link/path capacity) provides comparable results. The same applies for using multiple seeds (10) and a varying batch size for replaying flows (10).



(a) Shortest Path Flow Routing (Router Hops) (b) ECMP Flow Routing (Flow Hashes) (c) ECMP Flow Routing (Round Robin) (d) ECMP Flow Routing (Random Selection) (e) Path Load Snapshot Aware Flow Routing (f) Prediction-based Flow Routing

Fig. 3: Path Utilization Trends over Time.

B. *Time Measurements*

Table II summarizes the average delays of controller operations. Measured delay in our testbed allows an approximation for production networks depending on realistic conditions.

TABLE II: Elapsed Time for Controller Operation (ms).

Controller Operation	**Shortest Path**	**ECMP Strategies**			**Path Load**	**PFR**
		Hash	**Round Robin**	**Random**		
Packet Disassembly	0.45	0.50	0.53	0.44	0.48	0.44
Flow Prediction	—	—	—	—	—	0.50
Path Determination	0.01	1.25	1.26	1.22	1.37	1.82
Path Programming	7.25	7.72	7.52	7.32	7.47	7.27
Packet Reassembly	3.26	2.40	2.51	2.28	2.52	2.47
Flow Routing	10.97	11.87	11.82	11.26	11.84	12.50

The delay for extracting packet features (*Packet Disassembly*) is nearly constant. The same applies to the time to rebuild the packet (*Packet Reassembly*) and to program a flow path (*Path Programming*). Requesting a flow prediction (*Flow Prediction*) is associated with delay (0.5 ms [7]). Considering topology and flow data for path selection (*Path Determination*) also causes delay (≈1-2 ms). Both aspects are the main cause for variations in overall delays (*Flow Routing*).

VI. Discussion

PFR achieves a nearly equal flow distribution across multiple paths (improved overall utilization) while having stable load trends over time. As path loads and probable flow throughputs are considered, it is not limited to static routing metrics. Since the first packet of each flow has to be analyzed for path determination, *PFR* causes delay. A general application is challenging, e.g., w.r.t. scalability. Hence, a hybrid approach is pursued, using *PFR* only for a subset of flows (e.g., likely long lasting) and otherwise regular routing by default.

The likely flow duration is important together with the throughput to manage the predictive path state between load snapshots on a timely basis. While a flow's predicted throughput provides its likely load, the prediction of its duration allows to estimate how long this load is expected to last. Moreover, it enables the conduction of extensive experiments with increasing and dropping path loads caused by a variety of starting and ending flows. This is essential to further evaluate the generalizability, scalability and prediction error impacts.

PFR can also be employed to reserve path capacities. While some paths are used for flows with likely low/moderate load, the capacities of others are preserved for flows with likely high load. A nearly "inverted" Gaussian normal distribution was used for the number of flows per class, ensuring a relatively high number of small and large flows competing for path saturation. *PFR* can offer an improvement in such scenarios by spreading flow load more equally or preserving capacities.

Conceptually, *PFR* proposes to collect link/path utilization and latencies. While both properties are interrelated (high loads normally cause higher latencies), the latency is critical for several applications. Having multiple paths that are nearly equally used but are of unequal cost (number of hops) causes varying latencies for traversing flows. To also allow an evaluation at the flow level, flow experiences need to be tracked.

A software-based evaluation is a flexible solution sufficient for a proof-of-concept. As it holds its own challenges like performance issues, a hardware switch focused investigation that considers existing capabilities and limitations is planned.

VII. Conclusion and Future Work

PFR is a routing strategy and architecture to steer flows in networks supporting multiple paths and programmable switches. Common routing methods are based on static metrics. *PFR* considers dynamic topology and flow data to determine forwarding paths. Predicted flow characteristics and collected path load snapshots are utilized to maintain predictive path states used for path selection. Compared to other routing strategies, employing *PFR* achieves a more efficient distribution of flows across available paths, whereby a stable and nearly equal path load trend is ensured over time.

Flow durations need to be considered to dynamically maintain the predictive state for path loads during consecutively collected utilization snapshots (timely dynamics). Extensive experiments are necessary to analyze the generalizability, scalability and prediction error impacts. *P4Env* will be extended to support *P4 Probing*, *P4 INT* and *P4 Flow Monitor*. An integration of *P4Env* and *PFR* with P4 hardware switches is required to evaluate practical application scenarios. The analysis of distributed/hybrid *PFR* deployments is also planned.

References

[1] D. Kreutz, F. M. Ramos, P. Verissimo, C. E. Rothenberg, S. Azodolmolky, and S. Uhlig, "Software-Defined Networking: A Comprehensive Survey", *Proceedings of the IEEE*, 2015.

[2] P. Bosshart, D. Daly, G. Gibb, M. Izzard, N. McKeown, J. Rexford, C. Schlesinger, D. Talayco, A. Vahdat, G. Varghese *et al.*, "P4: Programming Protocol-Independent Packet Processors", *ACM SIGCOMM Computer Communication Review*, 2014.

[3] C. Hopps, "Analysis of an Equal-Cost Multi-Path Algorithm", Request for Comments 2992, 2000.

[4] M. Wang, Y. Cui, X. Wang, S. Xiao, and J. Jiang, "Machine Learning for Networking: Workflow, Advances and Opportunities", *IEEE Network*, 2018.

[5] R. Boutaba, M. A. Salahuddin, N. Limam, S. Ayoubi, N. Shahriar, F. Estrada-Solano, and O. M. Caicedo, "A Comprehensive Survey on Machine Learning for Networking: Evolution, Applications and Research Opportunities", *Journal of Internet Services and Applications*, 2018.

[6] C. Hardegen, B. Pfülb, S. Rieger, A. Gepperth, and S. Reißmann, "Flow-based Throughput Prediction using Deep Learning and Real-World Network Traffic", in *Proceedings of the 15th International Conference on Network and Service Management (CNSM)*, 2019.

[7] 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.

[8] S. Sen, D. Shue, S. Ihm, and M. J. Freedman, "Scalable, Optimal Flow Routing in Datacenters via Local Link Balancing", in *Proceedings of the Ninth ACM Conference on Emerging Networking Experiments and Technologies (CoNEXT)*, 2013.

[9] M. Al-Fares, S. Radhakrishnan, B. Raghavan, N. Huang, and A. Vahdat, "Hedera: Dynamic Flow Scheduling for Data Center Networks", in *Proceedings of the 7th USENIX Conference on Networked Systems Design and Implementation (NSDI)*, 2010.

[10] P. Poupart, Z. Chen, P. Jaini, F. Fung, H. Susanto, Y. Geng, L. Chen, K. Chen, and H. Jin, "Online Flow Size Prediction for Improved Network Routing", in *IEEE 24th International Conference on Network Protocols (ICNP)*, 2016.

[11] A. Lazaris and V. K. Prasanna, "Deep Learning Models For Aggregated Network Traffic Prediction", in *15th International Conference on Network and Service Management (CNSM)*, 2019.

[12] M. V. B. da Silva, A. S. Jacobs, R. J. Pfitscher, and L. Z. Granville, "IDEAFIX: Identifying Elephant Flows in P4-Based IXP Networks", in *IEEE Global Communications Conference (GLOBECOM)*, 2018.

[13] N. Katta, M. Hira, C. Kim, A. Sivaraman, and J. Rexford, "Hula: Scalable Load Balancing Using Programmable Data Planes", in *Proceedings of the Symposium on SDN Research*, 2016.

[14] C. H. Benet and A. J. Kassler, "FlowDyn: Towards a Dynamic Flowlet Gap Detection using Programmable Data Planes", in *IEEE 8th International Conference on Cloud Networking (CloudNet)*, 2019.

[15] T. Swamy, A. Rucker, M. Shahbaz, and K. Olukotun, "Taurus: An Intelligent Data Plane", *arXiv preprint*, 2020.

[16] "P4Environment (P4Env)", 2020. [Online]. Available: https://gitlab.cs.hs-fulda.de/flow-routing/cnsm2020/p4environment

[17] P4 Language Project, "P4 Language Tutorials", 2020, accessed: 2020-07-24. [Online]. Available: https://github.com/p4lang/tutorials

[18] ——, "behavioral model version 2 (bmv2)", 2020, accessed: 2020-07-24. [Online]. Available: https://github.com/p4lang/behavioral-model

[19] ESnet/iperf Project, "iPerf - The ultimate speed test tool for TCP, UDP and SCTP", 2020, accessed: 2020-07-24. [Online]. Available: https://iperf.fr

[20] A. A. Hagberg, D. A. Schult, and P. J. Swart, "Exploring Network Structure, Dynamics, and Function using NetworkX", in *Proceedings of the 7th Python in Science Conference*, 2008.

[21] P. Biondi and Scapy Community, "Scapy", 2020, accessed: 2020-07-24. [Online]. Available: https://github.com/secdev/scapy