Diolkos: Improving Ethernet Throughput through Dynamic Port Selection

Oceane Bel
obel@ucsc.edu
University of California at Santa Cruz
Santa Cruz, California

Joosep Pata
joosep.pata@cern.ch
NICPB
Tallinn, Estonia

Jean-Roch Vlimant
vlimant@cern.ch
Caltech
Pasadena, California

Nathan Tallent
tallent@pnnl.gov
Pacific Northwest National Lab
Richland, Washington

Justas Balcas
jbalcas@caltech.edu
Caltech
Pasadena, California

Maria Spiropulu
smaria@caltech.edu
Caltech
Pasadena, California

ABSTRACT

In large networked systems, a sudden increase in traffic could slowdown the network significantly, impacting network quality for multiple users. We present Diolkos, a system that leverages smart switches to dynamically re-reroute data flows in response to drops in performance. In contrast to other techniques, our tool predicts the future throughput at each port in a switch if a data flow were to be sent through it, and updates which port should be taken to maximize throughput. We use several techniques to predict network switch performance on a software defined network (SDN) mimicking topologies commonly found in datacenters. Experimentally, we demonstrate the effectiveness of choosing a port to send flows through based on predicted performance. We found that using a distributed predictive technique achieves a 24% improvement over using a traditional heuristic technique.

CCS CONCEPTS

• **Networks** → **Network performance modeling**; *Network simulations*; Network performance analysis.

KEYWORDS

Network port selection, smart switches, performance enhancement

ACM Reference Format:
Oceane Bel, Joosep Pata, Jean-Roch Vlimant, Nathan Tallent, Justas Balcas, and Maria Spiropulu. 2021. Diolkos: Improving Ethernet Throughput through Dynamic Port Selection. In *Computing Frontiers Conference (CF '21), May 11–13, 2021, Virtual Conference, Italy*. ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3457388.3458659

1 INTRODUCTION

In networked systems, there is a need to better understand traffic for efficiently steering demand through a network [23]. Engineers use predictive techniques to determine the return time of a packet in a network, and use this information to modify packet routes to lower return time. These techniques usually send data along shortest paths [43] in the network if possible, and react to performance changes based on best effort routing. Recently, the development of smart switches, as discussed in Section 2.1, have expanded the possibilities of packet routing: with powerful processing capabilities on the switches themselves, we can run sophisticated, rapid-response routing algorithms to dynamically route data.

Applying sophisticated routing algorithms is not a new concept: Bahnasse *et al.* [2] used centralized models that dynamically updated which flow routes to better balance load across the network. In practice, this global-view model has steep scalability limitations: as a model grows the overhead of using these systems increases [38] alongside the risk of leaking sensitive information [26]. These drawbacks can become a considerable hurdle towards improving performance in large networks. To solve this, Diolkos moves the entirety of the decision making to the individual switches, making them responsible for training and predicting future performance of the available ports.

Researchers [27, 42] have used distributed switch-based Reinforcement Learning (RL) models to get a significant increase in performance, however the models relied on dedicated compute nodes to create. Instead, Diolkos places an online-trained model directly on the switches themselves, removing the need for compute nodes. Xiong *et al.* [40], similar to the previous authors, also discussed that using distributed learning with programmable switches can enable faster training and reaction to changes in a network. In contrast, Diolkos demonstrates how to use Online Learning [4] (OL) to select the ports that will have the highest performance over the next few timesteps. Our idea is to treat the process of selecting ports on smart switches as an OL problem, learning how to route data based upon historical data right on the switches themselves.

We propose to utilize an OL model at the switch level that updates the ports taken by flows as performance changes, Diolkos. Diolkos monitors and updates the port taken by network traffic at the switch level to increase the throughput of the network, avoiding contended switches as necessary. We can exploit the underused port parallelism that a network could confer without the need for learning the state of the network as a whole. We experiment with ring and mesh networks, common topologies found in datacenters, running simulated workloads that fluctuated in flow intensity over time. During experimentation, Diolkos is given both full and partial control over the switches in the network, demonstrating its ability

CF '21, May 11–13, 2021, Virtual Conference, Italy

ACM ISBN 978-1-4503-8404-9/21/05.
https://doi.org/10.1145/3457388.3458659

to adapt to varying degrees of network control. We demonstrate that using a distributed model can increase the performance of a network by 24% over using a reactive solution. Our findings can aid in the development of next generation networks that use smart switches as a key component of data routing.

The main contributions of this paper are as follows:

- We describe a port selection system motivated by the use of smart switches with distributed Online Learning that can route data based on historical network performance.
- We present an analysis of Caltech's Tier 2 network for the purpose of determining a neural network architecture suitable for modeling observable network traffic on a specific switch.
- We implement a prototype of our system on a simulated network, and demonstrate that distributing the model gives a reasonable increase in port throughput compared to traditional port selection approaches.

2 BACKGROUND AND RELATED WORK

Heavy workloads on networks can cause contention if data flows are not balanced. Broch *et al.* [5] and Cano *et al.* [6] compared different routing protocols such as DSDV [33], TORA [30, 31], DSR [21, 22] and AODV [32], identifying a need to intelligently route network data based on past traffic data. Traditional best-effort network connectivity services, such as Global First Fit to Simulated Annealing [1], cannot meet the emerging requirements of exascale computing workflows [28], such as the US-CMS scientific networks.

One could use a centralized model [2] that creates routing plans based upon how data flows affect global network performance. But, using a global model on these networks can add unnecessary performance overhead when performance monitoring data is needed from globally distributed switches. We argue that a port selection model on the switch level trained with local switch data enables *smart switches* to make effective routing plans without a global controller.

2.1 Next generation smart switches

The aforementioned smart switch capabilities are motivated by the development of high efficiency machine learning hardware. Traditionally, machine learning is regarded as a computationally burdensome task, with training a neural network requiring dedicated hardware such as GPUs, TPUs [9], and more recently ASICs [18]. Recently, hardware companies such as NVIDIA and Intel have released deep learning compute hardware that cuts the cost, energy, and bulk requirements for machine learning tasks [7, 19]. With this hardware, we can run real-time inferencing on smart switches with relatively low cost and power overheads.

Smart switches are currently used in small instances to improve research networks. The TAEP project [20] was an optimization effort done by Caltech and AT&T researchers into applying predictive techniques to Tofino smart switches on the Tier 2 [10] network node within the Caltech campus, represented in Figure 1. Two experiments were conducted: a reinforcement learning-based load balancer and a heavy hitter detector. From their initial experimentation, they concluded that additional inputs must be considered for better load balancing. It is not sufficient to only know that a heavy hitter is occurring without an intelligent solution to resolve the heavy hitter. We envision that a smart switch of the future will have capabilities to train, save, and utilize a neural network to aid its normal operation of routing data flows using this hardware. With this, the requirement of a centralized training service is removed, and we have the opportunity to distribute the task of learning how a network responds to demand shifts directly to the hardware that is best suited to handle them.

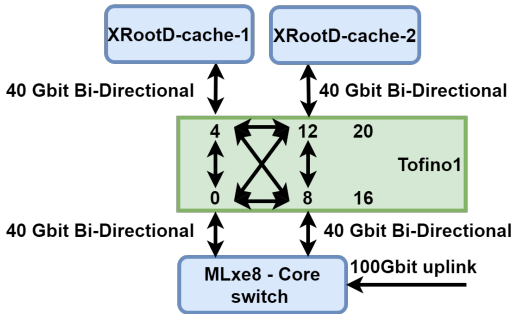

Figure 1: Placement of a Tofino switch in Caltech Tier 2 network. All links are bi-directional. The ports shown in the figure are the available ports used during the TAEP project.

2.2 Software Defined Networks

Smart switches can be part of a Software Defined Network (SDN). An SDN is a network architecture technique that allows the network to be centrally controlled for dynamic management of the network. SDNs allow the behavior and architecture of a network to be controlled with software, rather than hardware. Diolkos is designed primarily for smart switches that are controlled by an SDN.

Voellmy *et al.* [39] describes how SDNs allow for simpler controls over the network; however, the simplicity comes at the cost of scalability. Gholami *et al.* [14] used heuristics to lower the congestion of SDNs with mininet. This inspires our usage of heuristics as a comparison baseline. Additionally, they mention that applying neural networks could be an upcoming technique for lowering congestion on networks. Rexford *et al.* [34] describe the challenges of finding features that describe routing metrics, and modeling them for useful applications to routing. This work is one of the reasons we focus on switch level controllers instead of whole network controllers.

3 DESIGN

Diolkos, shown in Figure 2, is a tool that dynamically updates the ports taken by data flows in the network depending on forecasted network throughput. Diolkos must run on a smart switch, where all training, data gathering, data storage, and predictions are done. To prevent switches from working against each other, each switch has a map of where the hosts are in the system. At the site level, flows are tagged with their source and destination host address, and the switches use this tagged information and their own map prevent flows from infinite looping in the system.

Every r_c seconds, a smart switch collects the number of bytes that has flowed through every port in the switch, referred to as

throughput. These measurements are stored in a database on the switch for training the prediction model that learns how network traffic changes over time. The model is trained in parallel to throughput metric gathering, and generates an updated port selection for each switch every $(N + O) \times r_c$ seconds. N and O are part of a set of hyperparameters detailing prediction output, described later in Section 4. The model selects the port with the highest performance and applies it if the current port has a lower predicted performance. Doing so, data flows only through the ports that the model predicts to have the highest performance.

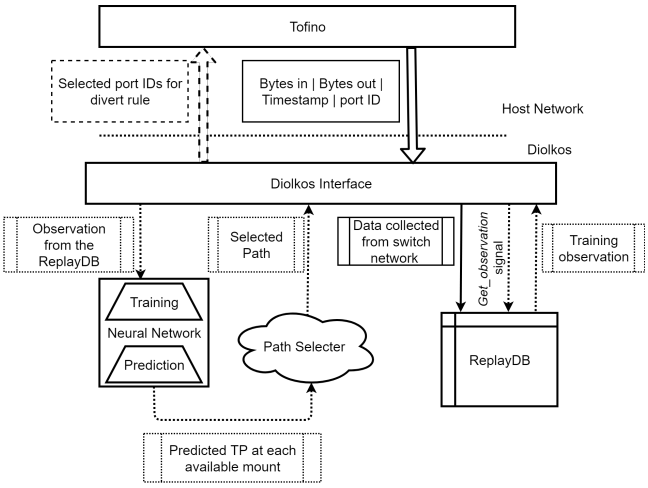

Figure 2: High level design of the Diolkos system and target system. Outlined arrows are data exchanged between the network of switches and Diolkos. The thick dotted arrow is the flow taken by the new calculated port by the model. Thin arrows are data paths between the components of Diolkos. The thin dotted arrow is the path taken by data during the training round. Undotted thick and thin arrows are data collected from the target system.

3.1 Training data

The training data used to train the model is the throughput at each port of the switches and the time this data was taken. The throughput ($TP_{i,j,t}$) of port j of switch i at time t is calculated using Equations 1 and 2, where BR_i and BT_i are the number of bytes received and transmitted respectively at port j of switch i, and T is the measurement period in seconds. Each throughput value is referred to as a *snapshot* of the switch's performance, measurements of the throughput at each port taken at approximately five-second intervals to stay in line with other related work about network sampling [3, 44].

$$NB_{i,j,t} = BR_{i,j,t} + BT_{i,j,t} \tag{1}$$

$$TP_{i,j,t} = \frac{NB_{i,j,t} - NB_{i,j,t-T}}{T} \tag{2}$$

The on-switch database is referred to as the *ReplayDB*. The ReplayDB is used to replay past performance fluctuations as training data for the prediction engine. If the model is trained with only recent observations, it wouldn't learn from historical trends, decreasing prediction accuracy. The data stored in the dataset must have structure between timesteps to usefully train the model. To determine structure, we calculate the auto-correlation structure over time, as seen in Section 4.1.

To format the data, we define a set of hyperparameters: *BS* (Batch Size), N (Number of timesteps in the future a prediction is), and O (number of predicted timesteps), all determined and described in Section 4. The choice of *BS*, N, and O is crucial for a functioning system. Predictions created for timesteps too close to the training data may not be relevant once applied to the system because predictions need to be far enough in the future to allow it to propagate in the network. Further, the model should make predictions with a reasonable amount of training information, determined by the *BS* value.

All training data is split by time into three partitions. The first 20% of the data is used for testing the model, the second 60% of the data is used for training, and the third 20% of the data is used for validation. This data is also normalized to prevent noise from adversely affecting training. In addition, a moving average of size 75 is applied to all training data to reduce the affect of noise. The moving average window represents approximately six minutes and 25 seconds of real-time data collection.

3.2 Per-Switch Prediction Engine

The *Prediction Engine* uses either a model or a heuristic to predict future throughput fluctuations. To determine the models for Diolkos, we started by recording the throughput at each port of the Tier 2 node located at Caltech. We then use this data as the regression target for training the models and heuristics. We compare techniques that are commonly used for time series modeling, such as Dense neural network models, Simple Recurrent Neural Networks (SRNNs), Gated Recurrent Units [8] (GRU) and Long Short Term Memory [15] (LSTM) models. Although SRNNs have fallen out of favor for GRUs and LSTMs, we include them as a base case for recurrent model experimentation. The output layer of the models contains the same number of neurons as there are ports. All the neural network models we experiment with use the Adam optimizer [24] to update the weights of the model during back-propagation.

We also implement three heuristics that can make throughput predictions: Linear Fit, Exponentially Weighted Moving Average [17] (EWMA), and Holt-Winter [11]. Linear fit is a simple heuristic that has low computational overhead. EWMA uses a sliding window to record the change of average over time. Holt-Winter uses a combination of the sliding window of EWMA with seasonality data present in the training data. Both our models and heuristics are online models since they apply changes to the system in real-time.

Diolkos predicts the I/O throughput value at every port on a switch for $N+O$ timesteps in the future. This prediction lets Diolkos select the port with highest potential average throughput value over all $N+O$ timesteps to send data through. We can also set $N+O$ to be how long the change is port should persist before it is able to change. In practice, changing the port taken by flows must be

done in regard to when the flows can safely changed while limiting packet loss.

3.3 Port selection

Once the model predicts the throughput of each port, Diolkos selects the port with the highest potential throughput based on all the flows previously observed at that port. The current flow is sent through that port for the next $(N + O)$ seconds. Before training is done, certain ports may have never been utilized because they are not considered to be on the shortest path. Hence,Diolkos will never predict that port to have any throughput because it was never included as training data. To prevent this, we use a hyperparameter α, which is the threshold for how low throughput at a port can be before it is considered to not have any traffic. If a port falls below α it is considered over the high throughput ports to enable Diolkos to start testing the ports that have never had data flows. The value of α changes depending on the network and the current load of the network.

3.4 Site-level coordination

In an SDN, the path selection module has a table relating ports to switches and a table relating host address to ports. During port selection, the tables and packet flooding are used to ensure that Diolkos produces a viable path that does not cause a loop, even if the first path may have higher throughput. For example, in the mesh network described in Section 5.1, if a flow goes from host 1 to host 2, ports 3', 2', 3, and 2 are always prioritized over 1 and 1' since sending the flow through port 1 causes the flow to go back to the source host. In case of aggregated links, flooding the packets during start up lets Diolkos know that multiple ports can be taken to go to the same location. If the aggregation of ports is seen to have the highest predicted throughput and there is no difference between the ports, Diolkos selects one at random.

To prevent packets from livelocking, all flows are tagged with information about where they originated from and its destination. In case a prediction tries to cause a loop, Diolkos looks through any other available port and chooses the one that has the next highest predicted performance that does not cause a loop. We assume that Diolkos eventually finds a port that data can flow through without causing a loop.

4 MODELING CHALLENGES

To experimentally demonstrate the Diolkos project, we begin with a model accuracy study and a hyperparameter study to determine which technique accurately predicts future performance changes. We select one model and one heuristic based upon each algorithm's predictive accuracy when modeling port throughput on a real switch. Then we determine the hyperparameters BS, N and O that allow these models to perform well.

4.1 Structured and unstructured data flows

Data without trends is considered to be *unstructured*, and attempting to predict a trend in it with machine learning will likely result in overfitting. To determine if neural networks are capable for modeling scientific network data flows and generalizing the trend to new data, we calculate the partial autocorrelation for the throughput of each port on the Tofino switches in the Caltech Tier 2 network. When looking at the autocorrelation of the data, we observe that ports 8, 12 and 0 has structure up to a lag of 10 timesteps. This indicates that the type of data flowing through ports 0, 8 and 12 have structure, and thus the trend can be modeled with neural networks. With this finding, we argue that neural network models can effectively model the data flows on ports 0, 8, and 12.

4.2 Selecting the Neural Network Model

For experimentation, we determine the heuristic and neural network model with the lowest predicting error when calculating throughput on a network switch with static BS, N and O values. We chose the model using the measured data since we simulate a similar network architecture at a smaller scale. We start by initiating BS, N and O to $BS = 5$, $N = 1$, and $O = 1$. If each port of a switch has the possibility of affecting another, then our approach should use those interactions to accurately model the network throughput through that switch. To measure a model's effectiveness, we measure the Mean of the Absolute relative Error (MAE) and the Standard Deviation of the Absolute relative Error (SDAE). If the predictions ever grow in error, it indicates that the model is impacted by noise. We calculate Absolute relative Error (AE) between predicted value P and target value T and at timestep i as: $AE_i = \frac{|T_i - P_i|}{P_i}$

For this selection process, data is collected from ports 0, 4, 8, and 12 from the Caltech Tofino switches. During data collection, we monitor the traffic flowing through each one of these ports as the number of bytes transmitted. We sample every five seconds until we obtain 20,000 snapshots of information in the ReplayDB, approximately 2.3 days of real life performance values for the model searches. We only collect 20,000 entries because this amount of training data adequately trained the models we tested. This limit was removed during the Study 2, and should not be present in an actual implementation of Diolkos. We attempt to predict the throughput of a singular port (e.g. 0, 4, 8, or 12) using data from the same port or data from all ports. For example, one experiment uses data from only port 4 to train a single dense hidden layered model to predict throughput at port 4. Then, we use data from all ports to predict port 4's throughput again. We do this to quantitatively prove that port interaction exists: if there is increased prediction accuracy when using data from all ports instead of one port, then we must use all port data as training data.

Table 1: MAE for structured data observed on the Caltech Tofino for the 7 models with the lowest prediction error out of the 19 tested. Our selected model is in bold.

Modeling technique	MAE (%)
Dense 1HL A	**0.45 ± 0.30**
GRU 1HL A	0.45 ± 0.32
Dense 0HL A	0.47 ± 0.34
SimpleRNN 1HL A	0.47 ± 0.35
GRU 2HL A	0.55 ± 0.40
EWMA	**0.54 ± 0.41**
Holt-Winter	0.63 ± 0.45

Accuracy measurements for all tested models are reported in Table 1. *HL* represents the number of hidden layers, *I* represents models trained with individual port data, and *A* represents models trained with all port data. Heuristics such as linear fit often mispredict by a significant margin, resulting in high MAE and SDAE. The GRU model with one hidden layer trained with all port data and the dense model with one hidden layer trained with all port data performed the best, with similarly low MAE and SDAE. However, the dense model has slightly lower MAE. We use the *Dense* model to determine optimal hyperparameters for it prior to applying it to a simulated network. For heuristics, we use EWMA since it is the highest scoring heuristic.

4.3 Hyperparameter search

Data formatting parameters must be determined, which include batch size (*BS*), the number of timesteps between training data and prediction (*N*), and the number of timesteps that are predicted (*O*). *BS* controls the size of the training data batch, potentially increasing accuracy. *N* controls the time to implement the result of a prediction, at the expense of prediction accuracy. *O* controls the number of predictions generated in one cycle, allowing the system to cache predictions at the cost of accuracy. We experiment with different combinations of *BS*, *N*, and *O* to discover each model's predictive performance (MAE) in predicting throughput of the Caltech Tier 2 data. *BS*, *N*, and *O* are varied between 3-7, 1-10, 1-5, respectively. The higher the MAE, the lower we consider each model's accuracy for that *BS*, *N*, and *O* combination. For our experimental testbed, we determined that the best values are $BS = 5$, $N = 1$, and $O = 1$ for Dense and EWMA, and any values beyond the ones we found contributed to higher MAE.

5 EXPERIMENTATION

To experimentally demonstrate Diolkos, we simulate a network with a similar topology to the one in the Caltech Tier 2 network and rerouted flows of a simulated workload. We also simulate topologies similar to ATCA-based topologies [41] used in data acquisition in large physics experiments. During experimentation, flows are sent through ports chosen by either a heuristic or a neural network. To simulate these network, we use Mininet [25], a network simulator that can simulate networks with loops in it. Mininet is a proven testing platform for network experiments when hardware is not a viable solution [12]. Although it is impractical to simulate hundreds of interconnected nodes with Mininet, researchers [13] have been able to reproduce performance benefits from simulated routing algorithms developed in mininet onto real hardware. Because of this translatability, we have chosen this simulator to conduct experiments.

The network architecture is defined with Ryu (described in Section 5.2). SDNs allow us to delegate the model training duties to the SDN controller host device. For Diolkos, it is done on the host of the simulation, with the predictive models sent to the simulated switches to make routing decisions. In a real implementation, the models will be trained and executed right on the switches.

We simulate traffic observed on the Caltech Tier 2 Tofino to demonstrate the performance benefit of applying our selected neural network against our selected heuristic. Although we have workload traces from Caltech Tier 2, we cannot replay them in the simulator because Mininet has no direct equivalent of a Tofino switch. We generate flows that have similar autocorrelation values to the ones observed on the Caltech Tier 2 Tofino switch. To reduce the noise present in the flows, Diolkos applies a moving average to the throughput values collected at each port in the simulated network. This is done to smooth out noise and clarify the trends in the throughput data for proper training. We then measure the performance benefits of applying our different approaches to determine which method produces the highest performance gain.

5.1 Topology

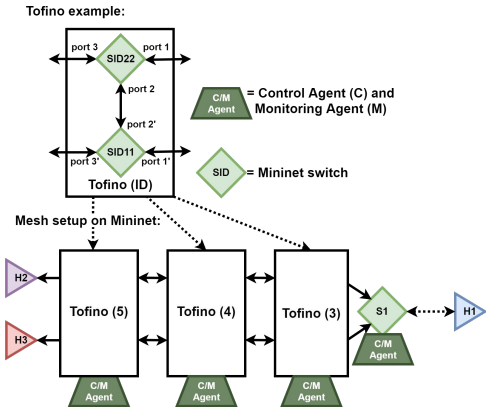

Figure 3: *ID* is the number of the Tofino. *22* and *11* serve as example names for the Mininet switches that make up the Tofino switch. Incoming links into switch 1 (s1) are 100Mbit links. The rest of the links are all 40 Mbit links. We are using a mesh network to represent a single path on the Caltech Tier 2, which uses a star topology.

Figures 3, and 4 illustrate the network topologies used in experimentation. We simulate both mesh and ring networks to demonstrate how Diolkos scales to networks with a higher number of connections and various topologies. Additionally, the mesh network represents the network topology the Tofino was part of in our model and hyperparameter search, at a larger scale. Switch 1 represents the MLxe8-Core switch, Host 2 represents the XRootD-cache-1, and Host 3 represents XRootD-cache-2 in the the Caltech Tier 2 network. Host 1 simulates the inbound traffic that would traverse the 100Gbit link. There is underutilized port parallelism through the Tofino switches, and has the potential for higher performance if the flows are dynamically sent through new ports. Boxed pairs of mininet switches represent one simulated Tofino switch (e.g. switches 311 and 322 represent one Tofino switch with 4 ports similar to a Tofino switch in the Tier 2 network), each containing an instance of Diolkos. We have chosen to simulate Tofinos in this manner since Mininet has no built-in smart switch emulation. Mininet sets a hard upper limit for the bandwidth of the links in a simulated network to 1000 Mbits per second, which prevents us

from simulating 100 GBit and 40 GBit links. Thus, all links in our simulated network run at 40 MBits, except the dotted link between S1 and H1, which runs at 100 MBits. We do this to preserve the ratio of bandwidth available in the Tier 2 network. We also experiment with a ring topology, as seen in Figure 4, where all the links are 100 Mbits. Both topologies use the same TCP protocol for all experiments.

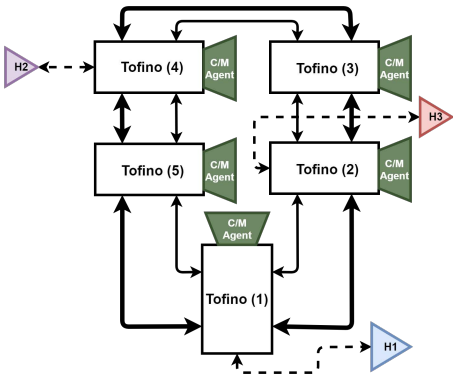

Figure 4: The Tofino switches are similar to those in Figure 3. Each network link is 100 Mbit links. Thick solid links are the outer paths numbered 00 and thinner solid links are inner paths numbered 11. The dotted link is the link between the hosts and switches

The goal of our simulated network is to deliver data produced from Host 2 (H2) and Host 3 (H3) to Host 1 (H1), and vice versa. Data can flow through any path as long as it reaches its target destination. Although the model type was determined from analyzing Tier 2 network data, the model type readily applies to a simulated network since we are modeling similar network traffic.

5.2 Base case

We compare the most accurate heuristic and neural network from our model search against a default Ryu controller with no updates to the switch's tables. Our base case is where the switches act as standard ethernet switches and the MAC table is not updated during the round. To do route discoveries, we use a Ryu controller [35], which allows user-configurable data flows in an SDN network topology where loops can exist. Ryu uses tables, represented as a dictionary-based MAC table, to associate flow IDs with the ports they are sent through. When a host sends a packet to the network, the packet will carry the MAC of the host that sent it and the destination host. The first switch and port that receives that packet associates the MAC address of the sender to the port that received the packet. If the destination MAC address is not found, the switch will flood the packet to all available ports until the destination host is found. This process is repeated continuously to account for any changes in the topology.

No training is done on the simulated switches, rather they do predictions on a pre-trained model created by the SDN controller. With an SDN controller, we can load a chosen data routing model onto the smart switch to change its routing routing behavior based on that switch's performance. In the scope of Diolkos, the Ryu controller trains the predictive model with the training data gathered at the smart switches it controls.

5.3 Studies

To conduct the studies described in Table 2, we use a synthetic workload that preserved the correlation structure observed on the workload measured on the Caltech Tier 2 node. Our goal is to show that Diolkos can handle surges of varying severity. The structure of the initial workload is preserved so we can apply the models and hyper-parameters selected in the previous section and measure the performance benefit of applying Diolkos to a network. With a synthetic workload, we simulate cases when Diolkos has control over the entire network and when it only controls intermediate Tofino switches. If Diolkos does not control all the switches, there is one instance of a model per Tofino switch and the instance sees all of the available ports on only the Tofino switch which demonstrates Diolkos's ability to work sections of the network.

The synthetic workload creates TCP flows between H2 and H1, and H3 and H1. We focus on TCP flows since they are very common in scientific networks as a way to guarantee data delivery. Any host can initiate a flow, and each flow sends packets of 128 KB in bursts of 10 seconds to the destination host. This data is continuously transmitted for 380 seconds, repeating over seven hours. The types of flows generated and redirected are IPv4 flows and ARP flows, common flows seen in most networks. Other flows do occur in our simulation, such as healthchecks, and we do not redirect these flows because they happen rarely and do not react well to delays in arrival. All flows are generated with iPerf [16], a data flow generation program often used to benchmark performance.

For our performance studies, we set our high traffic flow count to be 200 since Benson *et al.* [3] demonstrated that some private datacenters can receive 100 to 10,000 flow arrivals per switch per second. We observed that simulating more than 200 flows in a virtual machine-based Mininet caused simulation errors, preventing any useful experimentation from being done. Although full-scale networks will experience more than 200 flows per second, we believe that this is a representative scale model of datacenter workloads given simulator constraints.

Studies 1a and 1b demonstrate Diolkos's ability to run on a mesh network that has a surge in demand that then decreases. In **study 1a**, the hosts generate 10 flows per second, increasing to 20 flows per second after 33% of the experiment has run, returning to 10 flows per second after 66% of the experiment has elapsed. **Study 1b** starts with 100 flows per second, increasing to 200 flows per second after 33% of the experiment is run, and returns back to 100 flows per second after 66% of the experiment is run. **Study 2a and 2b** uses a ring topology to demonstrate Diolkos's effectiveness on a different topology while preserving performance benefits. **Study 2a** shows how a ring topology performs with a small number of flows and **study 2b** shows how a ring topology performs with 10X flows.

These case studies must show that an accurate predictive model can increase network throughput at the port level, and that it can scale. Additionally, we demonstrate how prediction accuracy plays an important role when determining paths in the network by letting

Table 2: *partial* means that Diolkos will only control the Tofino switches, while *full* means that Diolkos controls all switches. For mesh networks, Diolkos will only control the Tofino switches. In ring toplogies, Diolkos will only partially control the Tofino switches (switches with no host). When all the switches are controlled, there is a model per switch.

Study number	Number of Tofinos	Flows per second	Control type	α	Topo
1a	4	10-20-10	partial	0.01	mesh
1b	4	100-200-100	full	0.005	mesh
2a	5	10	partial	0.01	ring
2b	5	100	partial	0.02	ring

Diolkos control all the switches in some studies. Performance is not only determined by throughput, but also by successful packet arrival.

We measure network throughput in bytes per second at each switch, the average network throughput over all switches' individual ports, and the standard error of the average throughput. Throughput is useful for performance modeling because it is a clear target for optimization: higher throughput is the model's goal. We use standard error instead of standard deviation to compare the average benefit that updating the port taken by flows had on each port of the network. We implement throughput monitoring software in the Ryu controller that reports the speed at each port of the network.

During the performance studies with the heuristic and model, all switches gather performance data for 25,000 timesteps, with a 1 second gap between measurements, before training is initialized and Diolkos begins to suggest changes to the ports taken by data flows. Predictions take 0.2 seconds to be made, and a few milliseconds are required to apply the change. Hence, our predictions are valid in the network for about 2 seconds of network traffic.

6 PERFORMANCE RESULTS

Using the *BS*, *N* and *O* values determined from the hyperparameter search, we apply the Dense model, EWMA heuristic and the base Ryu controller to different simulated variations of the network path topology from the Caltech Tier 2 network. All graphs reported represent the mean at 95% confidence to compare mean performance. We quantify the benefit of an approach by how often it increases the throughput of a switch since it demonstrates how much of the network was used by that approach.

6.1 Study 1 results

In a situation where demand fluctuates, the Dense model has an average throughput of 2.86 MB/s ± 0.02 MB/s per port, 28% higher than the throughput per port achieved using EWMA's predictions (2.23 MB/s ± 0.02 MB/s). The Dense model's performance is also 49% higher than that of the base controller (1.92 MB/s ± 0.02 MB/s). At this small scale, we observe that, when the flow intensity doubles, both EWMA and Dense maintain a relatively high throughput.

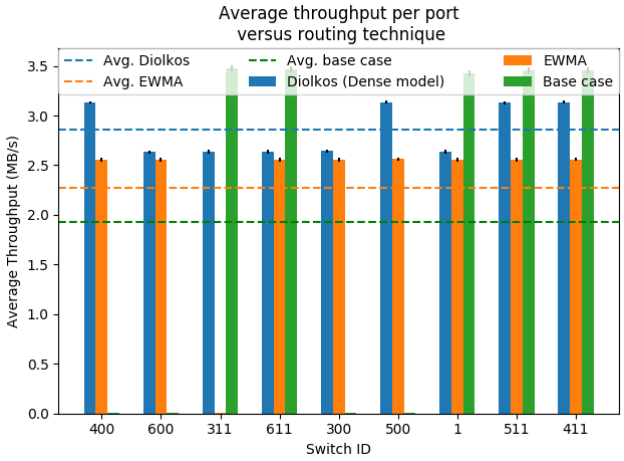

Figure 5: In Study 1, Diolkos maintains 2.86 MB/s ± 0.02 MB/s, the base case run measures 1.92 MB/s ± 0.02 MB/s, and EWMA measures 2.23 MB/s ± 0.02 MB/s.

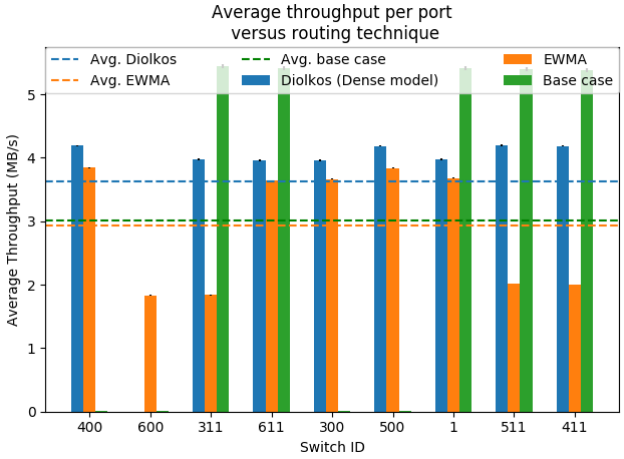

Figure 6: Diolkos measures a 3.63 MB/s ± 0.01 MB/s throughput, the base case measures 3.01 MB/s ± 0.02 MB/s and EWMA measures 2.93 MB/s ± 0.01 MB/s. Rises and falls in demand do not cause the Dense model to bottleneck in performance.

Increasing the severity of the surge and number of flows in the network, while letting Diolkos control all the switches of the network, improves results more, as seen in Figure 6. The Dense model obtains an average throughput per port of 3.63 MB/s ± 0.01 MB/s, 21% higher than that of the base case controller (3.01 MB/s ± 0.02 MB/s) and 24% higher than that of EWMA (2.93 MB/s ± 0.01 MB/s). If Diolkos has full control over the entire network, the accuracy of the prediction methods heavily influences the resulting performance benefit of the approach.

We also observe that both the base case and the Dense model did not push packets to the host on switch 600. We improved our design to allow the user to add additional information about switches that

may not have been covered by the base case through the *path selector*. Once the path selector was aware of the missing data, it was able to evenly spread the flows across the system.

6.2 Study 2 results

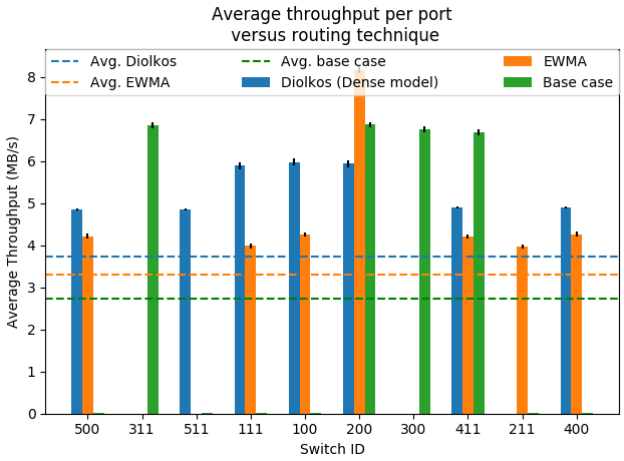

Figure 7: Dense model predictions has an average throughput of 3.73 MB/s ± 0.04 MB/s, 13% greater than the throughput of the EWMA model (3.31 MB/s ± 0.04 MB/s) and 37% greater than that of the base case (2.72 MB/s ± 0.03 MB/s).

This experiment's network topology has lower connectivity compared to the mesh network. When 10 flows were run on the network, the network had an average throughput of 3.73 MB/s ± 0.04 MB/s when the Dense model made predictions, 37% higher than that of the base controller (2.72 MB/s ± 0.03 MB/s) and 13% greater than that of the EWMA heuristic (3.31 MB/s ± 0.04 MB/s), as seen in Figure 7. The base controller struggled to use the network to its fullest extent, unlike the Dense model and EWMA. The Dense model manages to use more of the network than other techniques, and achieves the highest total throughput over all ports.

Increasing the number of flows in the network from 10 to 100 flows demonstrates the Dense model's superiority, as seen in Figure 8. The Dense model's predictions achieved an average throughput per port of 7.88 MB/s ± 0.05 MB/s, 31% greater than the throughput of the EWMA model (6.02 MB/s ± 0.04 MB/s) and 60% greater than that of the base case (4.92 MB/s ± 0.04 MB/s). Both models and our base controller opted to not use the entire network, thus leaving room for other flows in the network. Even with lower connectivity, Diolkos was able to maintain its superiority over the other methods, demonstrating that Diolkos has potential in improving network performance of an SDN of smart switches.

6.3 Overheads and current limitations

Diolkos's overhead is measured by subtracting the timestamp before the training/prediction and after it is done. Any model takes 130.48± 3.40 seconds to train on a simulated switch, and each inference on the model takes 0.20 ± 0.08 seconds. These numbers are reported from our simulated switches, and are likely to be lower in a real

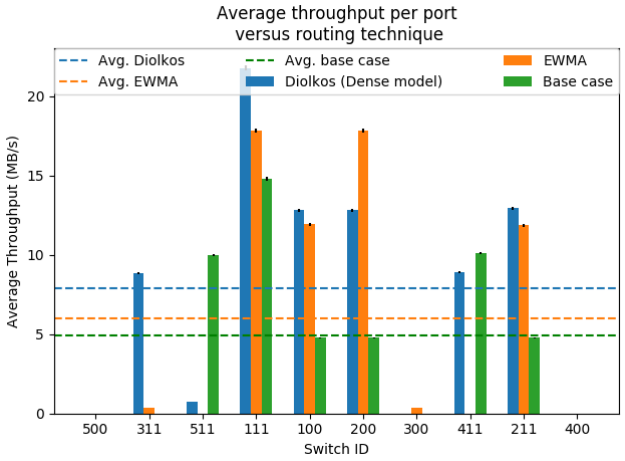

Figure 8: Diolkos is network topology agnostic: routes made with the Dense model measured an average throughput 31% greater than that of the EWMA model and 60% greater than that of the base case on a ring topology.

hardware implementation of Diolkos. In Study 1b, 335 out of 8800 connection requests failed to connect when Diolkos was running. Thus, Diolkos lost 4% of the total flows that were in the network. In the future, we can use another simulator, such as NS3, to track details at the packet level for measuring exact packet loss.

In the scope of our paper we focus on performance, however we acknowledge potential security implications of our technique. Changing the ports taken by flows increases the attack surface of the network for distributed denial of service attacks and internet worms. Worms can be filtered out with packet sifting techniques [29, 36]. Denial of service attacks can be mitigated by Zhang *et al.'s* framework [45] to defend networks from distributed attacks, and their approach [37] can be applied at the switch level.

7 CONCLUSION

Using next generation smart switches, we can completely distribute an online learning model that routes packets without the use of external compute nodes. Using a Mininet simulator, we are able to simulate performance increases with a distributed online learning model placed on smart switches. We found for a dynamically changing workload, a Dense neural network prediction model balanced data flows for a 21% increase in throughput over baselines, and a 24% increase over a EWMA prediction heuristic. Our findings demonstrate a useful application of smart switches and distributed model-based data routing for SDNs.

In future work we will apply Diolkos with our selected dense network to larger networks such as the Caltech Tier 2 network, dragonfly networks and InfiniBand (IB) networks. An application of Diolkos to real networks will let us to test scalability beyond a few hosts and flows. We will determine if any new networking coordination needs to be done to ensure proper flow control. Additionally, we will experiment with dynamically updating the α threshold in response to network changes.

ACKNOWLEDGEMENTS

We thank our collaborators Wenji Wu, Soren Telfer and Michael Jensen for their help in reviewing this paper. We also thank NVIDIA Corporation for their donation of a TITAN Xp GPU used as part of the development of Diolkos. Part of this work was conducted at "*iBanks*", the AI GPU cluster at Caltech. We acknowledge NVIDIA, SuperMicro and the Kavli Foundation for their support of "*iBanks*". We acknowledge support from Caltech's Intelligent Quantum Networks and Technologies (INQNET) research program, AT&T's Palo Alto Foundry and funding support from the U.S. Department of Energy's (DOE) Office of Advanced Scientific Computing Research as part of "Integrated End-to-end Performance Prediction and Diagnosis." This work is partially supported by a DOE/HEP QuantISED program grant, QCCFP/Quantum Machine Learning and Quantum Computation Frameworks (QCCFP-QMLQCF) for HEP, Grant No. DE-SC0019219. This work is partially supported by the U.S. DOE, Office of Science, Office of High Energy Physics under Award No. DE-SC0011925 and DE-AC02-07CH11359.

REFERENCES

[1] Mohammad Al-Fares, Sivasankar Radhakrishnan, Barath Raghavan, Nelson Huang, Amin Vahdat, et al. 2010. Hedera: dynamic flow scheduling for data center networks.. In *Proceedings of the 7th USENIX Symposium on Networked Systems Design and Implementation (NSDI 10)*. 89–92.

[2] Ayoub Bahnasse, Fatima Ezzahraa Louhab, Azeddine Khiat, Abdelmajid Badri, Mohamed Talea, and Bishwajeet Pandey. 2020. Smart Hybrid SDN Approach for MPLS VPN Management and Adaptive Multipath Optimal Routing. *WIRELESS PERSONAL COMMUNICATIONS* (2020).

[3] Theophilus Benson, Aditya Akella, and David A Maltz. 2010. Network traffic characteristics of data centers in the wild. In *Proceedings of the 10th ACM SIGCOMM conference on Internet measurement*. 267–280.

[4] Léon Bottou. 1998. Online learning and stochastic approximations. *On-line learning in neural networks* 17, 9 (1998), 142.

[5] Josh Broch, David A Maltz, David B Johnson, Yih-Chun Hu, and Jorjeta Jetcheva. 1998. A performance comparison of multi-hop wireless ad hoc network routing protocols. In *Proceedings of the 4th annual ACM/IEEE international conference on Mobile computing and networking*. 85–97.

[6] J-C Cano and Pietro Manzoni. 2000. A performance comparison of energy consumption for mobile ad hoc network routing protocols. In *Proceedings 8th International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (Cat. No. PR00728)*. IEEE, 57–64.

[7] Maurizio Capra, Beatrice Bussolino, Alberto Marchisio, Muhammad Shafique, Guido Masera, and Maurizio Martina. 2020. An updated survey of efficient hardware architectures for accelerating deep convolutional neural networks. *Future Internet* 12, 7 (2020), 113.

[8] Junyoung Chung, Caglar Gulcehre, Kyunghyun Cho, and Yoshua Bengio. 2015. Gated feedback recurrent neural networks. In *Proceedings of the International Conference on Machine Learning*. 2067–2075.

[9] Google Cloud. 2020. Using TPUs to train your model. https://cloud.google.com/ai-platform/training/docs/using-tpus.

[10] Computing, Mathematical Sciences, and High Energy Physics at Caltech. [n.d.]. CMS Caltech Tier2. https://tier2.hep.caltech.edu/

[11] Roger Davies and Alan Huitson. 1967. A sales forecasting comparison. *Journal of the Royal Statistical Society. Series D (The Statistician)* 17, 3 (1967), 269–278.

[12] Rogério Leão Santos De Oliveira, Christiane Marie Schweitzer, Ailton Akira Shinoda, and Ligia Rodrigues Prete. 2014. Using mininet for emulation and prototyping software-defined networks. In *2014 IEEE Colombian Conference on Communications and Computing (COLCOM)*. IEEE, 1–6.

[13] Müge Erel, Emre Teoman, Yusuf Özçevik, Gökhan Seçinti, and Berk Canberk. 2015. Scalability analysis and flow admission control in mininet-based SDN environment. In *2015 IEEE Conference on Network Function Virtualization and Software Defined Network (NFV-SDN)*. IEEE, 18–19.

[14] Masoumeh Gholami and Behzad Akbari. 2015. Congestion control in software defined data center networks through flow rerouting. In *2015 23rd Iranian Conference on Electrical Engineering*. IEEE, 654–657.

[15] Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural computation* 9, 8 (1997), 1735–1780.

[16] Chung-Hsing Hsu and Ulrich Kremer. 1998. IPERF: A framework for automatic construction of performance prediction models. In *Workshop on Profile and Feedback-Directed Compilation (PFDC), Paris, France*. Citeseer.

[17] J Stuart Hunter. 1986. The exponentially weighted moving average. *Journal of quality technology* 18, 4 (1986), 203–210.

[18] Insidehpc. 2020. Inspur Unveils GX4 Ai Accelerator. https://insidehpc.com/2017/06/inspurs-unveils-gx4-ai-accelerator/.

[19] Intel. 2020. Intel Distribution of OpenVINO Toolkit. https://software.intel.com/content/www/us/en/develop/hardware/neural-compute-stick.html.

[20] Michael Jensen. [n.d.]. Traffic analysis and experimentation platform (TAEP). https://indico.hep.caltech.edu/event/128/contributions/155/ Published: 14 Nov 2017 at 14:00.

[21] David B Johnson. 1994. Routing in ad hoc networks of mobile hosts. In *1994 First Workshop on Mobile Computing Systems and Applications*. IEEE, 158–163.

[22] David B Johnson and David A Maltz. 1996. Dynamic source routing in ad hoc wireless networks. In *Mobile computing*. Springer, 153–181.

[23] Srikanth Kandula, Sudipta Sengupta, Albert Greenberg, Parveen Patel, and Ronnie Chaiken. 2009. The nature of data center traffic: measurements & analysis. In *Proceedings of the 9th ACM SIGCOMM conference on Internet measurement*. 202–208.

[24] Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014).

[25] Bob Lantz, Brandon Heller, and Nick McKeown. 2010. A Network in a Laptop: Rapid Prototyping for Software-defined Networks. In *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks* (Monterey, California) *(Hotnets-IX)*. ACM, New York, NY, USA, Article 19, 6 pages. https://doi.org/10.1145/1868447.1868466

[26] He Li, Peng Li, Song Guo, and Amiya Nayak. 2014. Byzantine-resilient secure software-defined networks with multiple controllers in cloud. *IEEE Transactions on Cloud Computing* 2, 4 (2014), 436–447.

[27] Youjie Li, Iou-Jen Liu, Yifan Yuan, Deming Chen, Alexander Schwing, and Jian Huang. 2019. Accelerating distributed reinforcement learning with in-switch computing. In *2019 ACM/IEEE 46th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 279–291.

[28] Drew Margolin, K Ognyanoya, Meikuan Huang, Yun Huang, and Noshir Contractor. 2012. Team formation and performance on Nanohub: A network selection challenge in scientific communities. *Networks in social policy problems* (2012), 80–100.

[29] Chanho Park, Seong Woon Kim, and Sun Wook Kim. 2011. Apparatus and method for preventing network attacks, and packet transmission and reception processing apparatus and method using the same. US Patent App. 12/701,253.

[30] Vincent Park and Scott Corson. 2001. *Temporally-ordered routing algorithm (TORA)*. Technical Report. IETF internet draft.

[31] Vincent Douglas Park and M Scott Corson. 1997. A highly adaptive distributed routing algorithm for mobile wireless networks. In *Proceedings of INFOCOM'97*, Vol. 3. IEEE, 1405–1413.

[32] Charles Perkins, Elizabeth Belding-Royer, and Samir Das. 2003. RFC3561: Ad hoc on-demand distance vector (AODV) routing.

[33] Charles E Perkins and Pravin Bhagwat. 1994. Highly dynamic destination-sequenced distance-vector routing (DSDV) for mobile computers. *ACM SIGCOMM computer communication review* 24, 4 (1994), 234–244.

[34] Jennifer Rexford. 2006. Route optimization in IP networks. In *Handbook of Optimization in Telecommunications*. Springer, 679–700.

[35] Ryu. 2019. Ryu. https://osrg.github.io/ryu/.

[36] Sumeet Singh, George Varghese, Cristi Estan, and Stefan Savage. 2012. Detecting Public Network Attacks using Signatures and Fast Content Analysis. US Patent 8,296,842.

[37] Steven R. Snapp, James Brentano, Gihan V. Dias, Terrance L. Goan, L. Todd Heberlein, Che-Lin Ho, Karl N. Levitt, Biswanath Mukherjee, Stephen E. Smaha, Tim Grance, Daniel M. Teal, and Doug Mansur. 1997. *DIDS (Distributed Intrusion Detection System)—Motivation, Architecture, and an Early Prototype*. ACM Press/Addison-Wesley Publishing Co., USA, 211–227.

[38] Shams ur Rahman, Geon-Hwan Kim, You-Ze Cho, and Ajmal Khan. 2017. Deployment of an SDN-based UAV network: Controller placement and tradeoff between control overhead and delay. In *2017 International Conference on Information and Communication Technology Convergence (ICTC)*. IEEE, 1290–1292.

[39] Andreas Voellmy and Junchang Wang. 2012. Scalable software defined network controllers. In *Proceedings of the ACM SIGCOMM 2012 conference on Applications, technologies, architectures, and protocols for computer communication*. 289–290.

[40] Zhaoqi Xiong and Noa Zilberman. 2019. Do Switches Dream of Machine Learning? Toward In-Network Classification. In *Proceedings of the 18th ACM Workshop on Hot Topics in Networks*. 25–33.

[41] H Xu, Z-A Liu, Q Wang, J Zhao, D Jin, W Kühn, S Lang, and M Liu. 2012. An ATCA-based high performance compute node for trigger and data acquisition in large experiments. *Physics Procedia* 37 (2012), 1849–1854.

[42] Liangcheng Yu, John Sonchack, and Vincent Liu. 2020. Mantis: Reactive Programmable Switches. In *Proceedings of the Annual conference of the ACM Special Interest Group on Data Communication on the applications, technologies, architectures, and protocols for computer communication*. 296–309.

[43] F Benjamin Zhan and Charles E Noon. 1998. Shortest path algorithms: an evaluation using real road networks. *Transportation science* 32, 1 (1998), 65–73.

[44] Chuanting Zhang, Haixia Zhang, Jingping Qiao, Dongfeng Yuan, and Minggao Zhang. 2019. Deep transfer learning for intelligent cellular traffic prediction based on cross-domain big data. *IEEE Journal on Selected Areas in Communications* 37, 6 (2019), 1389–1401.

[45] Guangsen Zhang and Manish Parashar. 2010. Cooperative detection and protection against network attacks using decentralized information sharing. *Cluster Computing* 13, 1 (2010), 67–86.