IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received July 12, 2021, accepted August 10, 2021, date of publication August 24, 2021, date of current version September 23, 2021.

*Digital Object Identifier 10.1109/ACCESS.2021.3107236*

# Topology Independent Multipath Routing for Data Center Networks

**NATAŠA MAKSIĆ, (Member, IEEE)**
School of Electrical Engineering, University of Belgrade, 11120 Belgrade, Serbia
e-mail: maksicn@etf.rs

**ABSTRACT** Data center communication networks are characterized by fast and intensive traffic changes, to the extent that a statistical approach to routing has significant drawbacks. This paper proposes the first routing solution that takes into account every data center flow. In order to achieve the required performance, the proposed algorithm operates in the data plane, i.e. in the packet processing pipeline of the programmable packet forwarding chips. Route updates are triggered by the arrival of new flows, and are immediately transferred to the upstream switches in order for them to direct newly arriving flows along the best path. This enables the proposed algorithm to handle fast and intensive traffic changes, such as large-scale flow incast communication patterns which historically presented challenges to the data center routing algorithms. With the explicit knowledge of the current path state, the proposed algorithm can improve routing performance in the data center topologies used today. Additionally, being able to operate with high performance in arbitrary topology, the proposed solution can enable the introduction of new topologies into the data center networks, instead of topologies designed with the goal of supporting equal-cost multipath routing which are dominant in modern data centers.

**INDEX TERMS** Routing protocols, communication system traffic control, packet switching, programmable data plane, data center routing, multipath routing.

## I. INTRODUCTION

Data centers have enabled the modern information society. They belong to the biggest and most important technological investments. Data center communication networks are a critical part of data centers, as they transfer data between thousands of computers which perform data processing and storage. Yet, packet routing in the data center networks is performed using ECMP (Equal-Cost MultiPath routing), decades-old algorithm, which is oblivious to traffic properties.

In order to be fast enough to react to fast traffic changes, path reconfigurations must not use CPUs (Central Processing Unit) which traditionally calculate paths within routers. CPU operations are significantly slower than the operations in the data plane: while the control plane is determining where to direct the flow, the congestion will already have occurred. Path selection needs to operate at the same speed as the packet forwarding, and hence, they need to be located in the packet processing pipeline of the high-performance packet forwarding chips in the switches. ECMP itself operates in the packet processing pipeline: it assigns a packet to one of the equal-cost ports by looking at the five fields in the IP packet header (source and destination IP address, transport protocol type, and source and destination transport layer port). In the case of arbitrary topology, this method of forwarding is not sufficient, and we need to transfer information about link utilizations between switches in order to accomplish the goal of multipath routing: even link utilizations and the consequent congestion avoidance.

The simplicity of ECMP has enabled its standardization and implementation. The implementation of more complex routing algorithms in the traditional data center switches would be a lengthy process, which would need to bring together different equipment vendors. However, recent developments in programmable data plane chips have enabled the introduction of new protocols, without the lengthy process of standardization and implementation in the packet forwarding chips.

The main obstacle in using programmable data plane chips is their restricted processing capabilities. The packet processing pipeline needs to process packets at the line rate, and to introduce as short a packet delay as possible.

The associate editor coordinating the review of this manuscript and approving it for publication was Tu Ngoc Nguyen.

Recently, data center routing solutions based on network scanning were proposed in the literature. These solutions will be discussed in the next chapter. In short, they collect network information along the way by using additional headers or special probe packets. Switches use the collected information for directing traffic. The main limitation of this kind of routing is the duration of network scans. Between scans, routing is performed according to the last scan, and it may not be adapted to traffic changes which have occurred in the meantime.

This paper proposes TIMP (Topology Independent Multi-Path routing), a data center routing solution which takes into account each flow and generates routing updates immediately upon detection of new flows. TIMP accomplishes this while keeping the processing short and simple, in order for it to be executed in the packet processing pipeline.

By reacting to every flow, TIMP is able to route traffic in the situations when a large number of new flows appear in the short time. These situations are typical for partition-aggregate data center applications, which are typical for the data centers since they use multiple computers for performing time critical tasks, such as a web search. This can improve the performance of existing data center networks.

The important property of TIMP is that it offers high-performance routing in arbitrary topology. ECMP routing results in an even traffic distribution only in topologies with specific symmetry properties, and this is the reason why certain topologies are dominant in the data centers. However, data center topologies which are not suited to ECMP may have other desirable properties. In this paper we will evaluate the application of TIMP in the torus topology, hypercube topology and the flattened butterfly topology. These topologies have applications for on-chip networks in multiprocessor chips [1]. A reduced design complexity and connection length of these networks could benefit data center performance and price. TIMP can provide high-performance routing in other non-ECMP topologies and also in ECMP friendly topologies, such as widely used leaf-spine topology. The evaluation section also contains the evaluation for leaf-spine topology.

Results presented in this paper are completely reproducible, with the publicly available P4 implementation, the Mininet emulation setup of P4 program and the ns-3 simulation code and scripts which execute simulations and produce graphs [2].

This paper is organized as follows. The second section discusses related work. The third section presents an overview of P4 language features. The fourth section describes TIMP design. The fifth section presents the P4 implementation of TIMP. The sixth section discusses TIMP performance. The seventh section concludes the paper.

## II. RELATED WORK

Published proposals in the research area of the high-speed routing supported by the data plane include CONGA (distributed CONGestion-Aware load balancing for data-centers) [3], HULA (Hop-by-hop Utilization-aware Load balancing Architecture) [4], DASH (Data-plane Adaptive Splitting with Hash threshold) [6] and Contra [5]. These proposals are based on periodic network scanning, either by using additional packet headers or network probe packets.

CONGA, HULA, DASH and Contra perform flowlet-based packet forwarding [7]. Flowlets are more suitable for implementation in the packet processing pipeline, since they are detected according to the ECMP hash value, and not according to the TCP (Transmission Control Protocol) connection state machine. Flowlets expire after the predefined period of inactivity. Hence, one flow may consist of a number of flowlets, if it has periods of inactivity. Since all packets belonging to a flowlet are forwarded along the same path, flowlet based routing guarantees in-order packet delivery.

CONGA is designed for the leaf-spine topology. In this topology the path of the packet is determined by the routing decision in the leaf switch. CONGA operates only in leaf switches. In CONGA, the leaf switch attaches additional packet headers to the packets which are sent in the network. These headers are used to collect the maximal link utilization along the path. When the header reaches another leaf switch then it is returned to the originating switch by attaching it to another packet. The time by which the information returns to the originating switch is approximately the network round trip time and this is the time in which Contra can react to the traffic changes.

HULA, DASH and Contra are based on periodic probes which perform network scanning. While HULA and DASH rely on some other routing protocol for the routing of probes, Contra can operate without additional routing protocols. Thus, Contra, like TIMP, has the ability to operate in arbitrary topology.

HULA operates in each switch of the topology. HULA probes carry the maximum link utilization encountered on the path towards the switch which generated the probe. Each switch in the topology generates probes periodically. When a switch receives a probe on a port, it calculates the maximum link utilization on the path over that port. This utilization is calculated as a maximum of local utilization on the port on which the probe arrived and the utilization received in the probe. For the forwarding of the newly detected flowlet, HULA uses the path with the minimal utilization.

Instead of using the best path for the new flowlet, DASH selects one of the available paths. Path selection probabilities are calculated according to the path utilizations. This reduces the effect of routing all flowlets along the best path during the inter-probe period. DASH uses both flowlet-based routing and packet-based routing.

Contra is the generalization of the probe-based data plane routing to any topology and various path selection criteria. Contra uses probes both for path discovery and for carrying path metric. We will discuss the most significant differences between Contra and TIMP.

The difference between Contra and TIMP from the performance standpoint is the speed of reaction to the arrival of new flows. Contra will receive the path information upon the

arrival of the probe and then it will take into account traffic changes since the last probe. TIMP reacts to each new flow immediately and informs other switches about the change of path state. These properties determine the reaction to the fast traffic changes. If we observe the TCP incast traffic pattern, which is typical for data center applications, TIMP will be able to respond as the TCP flows arrive, and Contra will react only when the next probe arrives, which may be too late to prevent congestion. In Contra, the probe period cannot be shorter than half of the maximal round trip time in the network. This limit reduces the minimal probe period as the network diameter increases.

The second significant difference is the approach to the prevention of control message loops. For this purpose Contra uses mechanisms for loop prevention of control messages from ad-hoc routing protocols DSDV (Destination-Sequenced Distance-Vector Routing) and Babel. These mechanisms are based on keeping track of update identifiers. On the other hand, TIMP route updates do not need mechanisms for loop prevention. This results in significantly simpler implementation.

Despite the more complex algorithm for the prevention of control message loops, Contra has the inherent possibility of the occurrence of the transient loops, while TIMP cannot have transient loops. For Contra, the authors propose detection of transient loops by monitoring the packet TTL (Time To Live) and reacting if it starts increasing. Such detection may last long enough for the flow to miss the flow completion deadline in the data center applications.

TIMP is in essence a different routing protocol from CONGA, HULA, DASH and Contra because it reacts to the change of the network state instead of scanning the network state using additional headers or periodic probes. Periodic probes are typically used in ad-hoc routing protocols for wireless networks. Ad-hoc routing protocols have features adapted to simple devices with low power and processing capabilities which should result in a simpler implementation needed by the hardware processing pipeline. This makes their solutions attractive for P4 implementations. However, as this paper will show, it turns out that TIMP can be efficiently implemented in P4. Routing updates triggered by the state change have the major advantage of faster reaction to the creation of new flows, since they are sent when the change is detected and the probes are sent periodically. The time period by which probes are sent limits the responsiveness of the ad-hoc routing protocol. In ad-hoc networks this is not the issue, but in the data center networks it is critical.

In order to keep the P4 implementation simple, the metric used by the proposed TIMP implementation is the number of flows on links. Balancing based on the number of flows has proven to be successful in data center networks since it is used by ECMP. With congestion control algorithms such as Swift [8], load balancing based on the number of flows is successfully used in data center networks. On the other hand, CONGA, Contra, HULA and DASH use estimated link utilization as a metric. Contra and DASH can use different metrics but the evaluation in the paper [5] is performed using an estimation of the link utilization. Estimation of link utilization is more complex to implement since it requires multiplication. Multiplication may not be available in P4 implementation. Additionally, TCP incast can be detected by the increased number of flowlets before the traffic throughput is increased. This makes the number of flows a better metric for the detection of TCP incast.

In other proposals, authors have used different techniques in order to improve data center packet forwarding.

DRILL (Distributed Randomized In-network Localized Load-balancing) [9] chooses output queue for each packet based on the current local queue states. The paper concludes that there is little reordering due to the small variance of queue lengths in the network provided by DRILL. However, for routing in asymmetric networks the local state is not sufficient [3]. TIMP performs routing based on the state of the entire paths.

Presto [10] improves ECMP performance by splitting the flows into chunks. Servers direct these chunks across preconfigured paths and handle packet reordering. Expeditus [11] is designed for a 3-tier Clos network and it uses two stage path selection based on the monitoring of the link utilization and on the control message exchange between ToR (Top of Rack) switches during the start of the new flow. Both Presto and Expeditus are designed for special topologies, and TIMP can operate in arbitrary topology.

LB-SPR (Load Balanced Shortest Path Routing) [12] and LB-ECR (Load Balanced Equal-Cost Routing) [13], [14] propose congestion avoidance in the data center networks by using optimized two-phase load balancing. This enables the application of non-ECMP topologies in the data center networks by providing load balancing parameters based on the current communication demands in the network. Since the load balancing parameters are calculated in the control plane, these protocols do not have the ability to react to traffic changes at the level of the individual flows.

The main motivation behind TIMP is to provide a data center routing algorithm which can update routing information in the switches fast enough to be able to follow the fast changing state of the data center network. With the ability to follow the quickly arriving data center flows, routing protocol would provide higher level of determinism in the data center routing, and enable the data center network to operate more consistently as a part of the one large computer, which data centers aim to be.

## III. P4 LANGUAGE FEATURES

P4 program defines operations in the packet processing pipeline. In order to provide parallel processing, operations of the algorithm are assigned to pipeline stages. As more operations are added, more space in the chip is used, and the chip becomes more expensive. Additionally, longer pipelines introduce longer packet delays.

In Fig. 4 we can observe packet flow within the TIMP implementation in the Portable Switch Architecture (PSA).

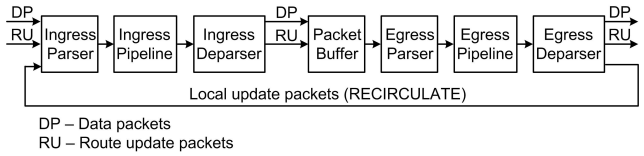



**FIGURE 1.** TIMP packet flow in the portable switch architecture.

PSA defines the model of the P4 programmable switch. Data packets and route update packets enter the switch and are processed in the ingress pipeline. The ingress pipeline has the ability to create new TIMP update packets by using the packet cloning operation. Packets are then stored in the packet buffer, and processed in the egress pipeline. Finally, data packets and route update packets are sent to output ports.

P4 program defines the operations performed on the packet in the ingress pipeline and in the egress pipeline. The processing of each packet in the pipeline is described by the P4 standard as a logical thread of execution. P4 program is expressed as a sequence of operations conducted on the thread-local memory state which represents the packet. Additionally, each pipeline stage can read and write its own memory state, which is preserved as the packets traverse that stage. For that, pipeline stages use registers in the so-called global memory.

Parallel processing in the pipeline stages introduces an important limitation regarding access to memories which keep the pipeline state. The thread-local memory can only be accessed by operations on one packet as it traverses the pipeline, while the global memory is accessible by all packets traversing the pipeline. The limitation on the global memory is that each register in that memory has to be accessed by only one of the pipeline stages, in order to simplify memory accesses and to improve switch performance. Due to this feature, it is not possible to read the value from a location in the global memory, then perform a calculation that requires multiple pipeline stages, and finally to update the same location in the global memory. For TIMP this is the case for counting new flowlets: registers containing flow counts of the pre configured ports should be read from the global memory, then used to calculate output port for the new flowlet, and finally the register for the selected output port should be incremented.

Such operation may be performed using another P4 language feature, recirculation, which allows for the packet to be returned from the end of the egress pipeline to the beginning of the ingress pipeline. In the case of counting flowlets on ports, an update can carry the identifier of the port for which the flow was added or removed. As the recirculation introduces new packets to the pipeline, it can increase latency of data packets and should be used carefully. We discuss the impact of updates in TIMP in sections IV-E and IV-F.

P4 implementations also impose constraints on arithmetic operations: multiplication and division are reduced or completely removed from implementations. The only arithmetic operations that TIMP uses are addition and subtraction, which enables its implementation on available P4 programmable chips.

## IV. TIMP DESIGN

TIMP needs to be simple enough for the data plane implementation and it has to provide high-performance routing. This section explains the design decisions involved in developing TIMP.

If we observe data center traffic, there is a general need for flows to be transmitted as fast as possible. However, data center traffic is not uniform in time. An important traffic pattern that needs to be handled is the TCP incast, which occurs in partition-aggregate applications typical for data centers.

In order to be able to handle the TCP incast, TIMP switches generate route update packets when they detect new flowlets, and send them to the upstream switches on the paths to the destination of the new flowlet. The upstream switches are then able to select the best path for the arriving flows.

Obviously, this is the fastest possible update which can be sent to the upstream switches. The update is sent upstream as the first packet of the new flowlet is detected, before any other packet is processed. If required, the update interval can be shortened by using priority queues for the route update packets.

The question that arises is how many initial packets of the new flows can already be queued to be processed in the switch which generates the update. If we assume a queue size of 100 kB per port and a link speed of 10 Gb/s, the entire queue will be processed in 80 $\mu$s. In the partition-aggregate applications, one aggregation server will partition the tasks to many worker servers, which need to answer within a given time, which is usually expressed in tens of milliseconds [15]. Processing in the worker servers involves using services of the server's operating system, such as disk access and network access. Just the task switching within these services can take tens of microseconds [16], and the access to the hard drive, network, as well as the access to the processing time, depend on the activity of other applications on that server. Thus, the variations of the processing time in the worker servers will most likely be larger than 80 $\mu$s. Additional variations of flowlet delay will occur in the network, since the packets from worker servers may traverse links with different delay due to queueing, or a different number of links on their path. Hence, TIMP will be able to perform load balancing of the flows belonging to TCP incast.

### A. PATHS

ECMP requires graphs containing equal-cost paths to be precalculated. Possible ports for each destination need to be configured within the ECMP forwarding function in the switches. In most cases, path calculation is performed by routing protocols such as OSPF (Open Shortest Path First) or IS-IS (Intermediate System to Intermediate System). In the data center networks, BGP (Border Gateway Protocol) is commonly applied for the path computation [17].

Similarly, TIMP also requires a precalculated graph of links leading towards each destination switch. P4 program needs to be as simple as possible and to contain only elements necessary for path selection. Calculation of these graphs can be performed by control plane, either by the distributed routing protocol, for example BGP, or by the centralized controller. The implementation presented in this paper uses a centralized controller because it is simpler to implement, and there are no disadvantages of using the centralized controller over BGP for this purpose. Pre-calculated graphs have to be loop free.

Precalculated graphs provide TIMP with another advantage: simplified processing of route update packets. For each destination, updates will be forwarded along a pre-calculated graph towards predecessor switches. With this functionality, TIMP does not need a mechanism to prevent the route update loops. Mechanisms for the loop prevention of routing updates are generally present in routing protocols but they would increase complexity of P4 implementation.

The task of pre-calculating a graph of possible paths towards one destination is surprisingly simple. This task is equivalent to the task of assigning one arrow to each link, and making sure that arrows do not create closed paths. The task of assigning arrows can even be performed by a human operator who wants to add a possible packet traffic detour.

In practice, calculation of TIMP forwarding graph and checking for loops will be automated with more or less sophisticated algorithms. In this evaluation, the following simple algorithm is used:

- for each destination calculate depth first search paths from each node to that destination,
- sort paths by length,
- add paths one by one to the set of selected paths. Each path is added if it does not create a loop with previously added paths.

This algorithm is one of the many possible path selection algorithms, and its purpose is to illustrate the performance of TIMP in this evaluation. Figure 2 shows switches in a simple network, and paths towards switch 0 for ECMP and TIMP.

We can notice that both ECMP and TIMP have the same principle: arrows which lead towards the destination do not create paths with loops. The difference is that ECMP will use only links which lead to a destination with equal-cost, and TIMP can use all links.

In TIMP, as in ECMP, the information on ports which lead to each destination switch is preconfigured in the appropriate P4 table in the switch. TIMP will use this information during the path selection.

Github repository with the ns-3 TIMP implementation [2] contains simulation of partition-aggregate application in the network shown in Fig. 2, along with the script which executes simulation and generates graphs.

The partition-aggregate architecture is dominant in applications which distribute processing and data storage across many servers [15]. Large data sets used in these applications cannot be stored on one server. Hence, the processing will

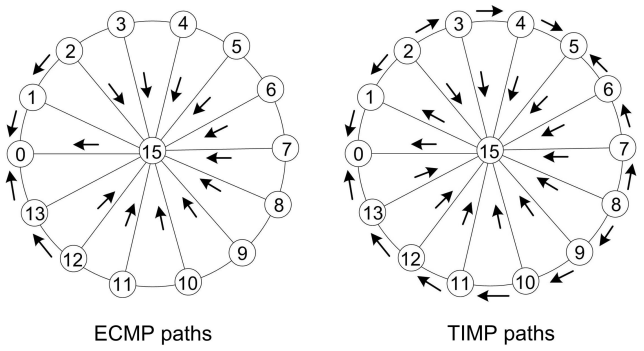



**FIGURE 2.** ECMP and TIMP paths. Arrows represent graph of paths leading to the switch 0. Arrows define ports between which switches can choose to forward flowlets to the switch 0. ECMP decides based on the hash value of the packet, and TIMP decides based on path state.

simultaneously be executed on many worker servers, and the results will be returned to the aggregation server for further processing. In the evaluation, we simulate the flows which carry the results from the worker servers to the aggregation server. The critical goal of a data center routing algorithm is to be able to route these flows which are generated practically simultaneously, and need to finish as fast as possible in order to enable partition-aggregate applications to meet their processing deadlines.

Figure 3 displays the difference in TIMP and ECMP performance in the case of two partition-aggregate applications in which response flows are sent from computers attached to switches 3-12 to two computers attached to switch 0. Figure 3 shows flow completion times for each of 20 flows. The graphs illustrate that, in this case, TIMP flow durations are half of the ECMP flow durations.

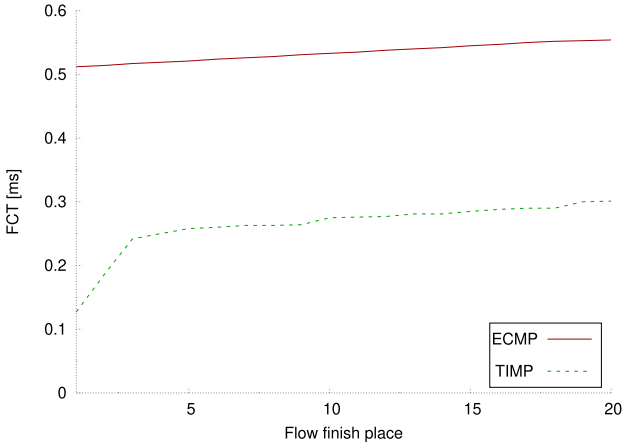



**FIGURE 3.** Flow completion times for 20 simultaneous flows in the topology from Figure 3. The destinations of the flows are servers attached to switch 0, and sources are servers attached to other switches in the network. TIMP provides significantly better load balancing.

In this topology, TIMP has more paths available and flow completion time is about half of flow completion times with ECMP. Flow completion times are important for partition-aggregate applications since they set a limit for

acquiring answers from computers that perform subtasks. Finally, due to loop-free precomputed paths, micro-loops for data packets cannot be created either.

### B. DESTINATIONS

TIMP destinations are defined with respect to the packet IP addresses. The IP address is translated to the destination identifier by performing the longest prefix match. TIMP destination will typically include the address range of servers connected to one ToR switch. TIMP paths are calculated for each destination.

Translation from the packet destination address to the destination identifier is performed using the packet processing pipeline table populated by the controller. This table performs the longest prefix match on the destination IP address. In a separate table, the controller provides the list of ports which belong to the paths toward destination. Data packets are sent to the output ports on their way to their destinations, and routing updates are sent to the input ports along the precomputed path. More details of the implementation are presented in Section V.

### C. FLOWLETS

The common approach with flow-based routing is to use flowlets [7], instead of flows. Instead of tracking active time of flow on the TCP level, which is a complex operation, flowlets are limited to periods of flow in which packets are transmitted at high enough rates. If a pause occurs in the packet transmission, the flowlet is considered to be finished. In that case, the following packets can be routed along different paths as part of the new flowlet.

### D. METRIC

ECMP successfully provides data center routing by distributing the flowlets among equal-cost paths evenly. Since this method of load balancing has been proven in practice, the goal of TIMP is also to distribute flowlets evenly across paths. The path metric used in TIMP is the maximal number of flowlets on any of the links along the path to the destination. TIMP will direct the new flowlet along the path with minimal metric.

In the P4 implementation described in Section V, the number of flowlets on a link is calculated by counting hash table entries.

### E. INTER-SWITCH UPDATES

TIMP update packets are exchanged between neighbouring switches. Each update packet carries the identifier of the destination switch and a changed price for that destination. This simple packet structure simplifies packet generation and processing in order to keep the P4 program simple.

The destination switch in the update packet is the destination of the new flowlet, which triggered the update. Upon a change in the distance to the destination, a switch sends updates only to neighbours which precede that switch on the paths toward the destination of that flowlet. A change in the distance to the destination can be triggered by the detection of a new flowlet or by the reception of the update packet.

Probe-based routing algorithms, Contra and DASH, have the limitation that the probe period has to be longer than half of the maximal round trip time between switches in the network. This results in a longer update time than in TIMP, and in the limited ability of these protocols to handle TCP incast.

In order to obtain insight into the update time of TIMP in the data center, we need to understand data center traffic patterns. Since TIMP updates are triggered on the arrivals of the flows, we need information on the flow inter-arrival times. Such information for one of the Facebook data centres is available in [18]. This paper reports median interarrival times of flows of 2-8ms per server. If we assume an interarrival time of 2ms per server and 40 servers per switch, we obtain a flow-interarrival time of 50 $\mu$s per switch. Thus, in such a network, the routing prices for each switch will be refreshed on 50 $\mu$s or less, depending on the number of flowlets within flows. As discussed, the upstream switches will be updated on changes as the first packet of the flowlet is forwarded toward the destination switch. When it reaches the destination switch it will initiate the update of the switches which do not belong to the path tree on which the packet is forwarded. The update is forwarded through the switch only if its price toward destination has changed since the last update.

### F. INTRA-SWITCH UPDATES

In order to satisfy the locality of accessing the global memory, TIMP introduces local update packets. The problem that arises is storing a number of flowlets on switch ports in the global memory. This number is needed as an input to the algorithm which chooses the port for the new flowlet. After the port is selected, the number of flowlets needs to be incremented. If the global memory register is read at the start of the processing and updated at the end of the processing, that would break the condition that the global memory register can be accessed from one stage. For that reason, read/write access to this register is performed only at the beginning of the pipeline, and local update packets are used to carry the information about the addition and removal of flowlets from the end of the input pipeline to its beginning.

TIMP local update packets use the recirculate operation to be returned to the start of the ingress processing. Local update packets solve the problem of the localization of the memory accesses to the registers containing local port utilizations. As the next section explains, this problem arises since the local port utilizations need to be read before the calculation of the output port of the new flowlet, as it needs to be written after the output port is selected.

## V. P4 IMPLEMENTATION

The P4 code of the implementation, along with the Mininet emulation setup, is available at Github [2].

### A. INGRESS PIPELINE PROCESSING

High-level P4 program flow diagram for ingress pipeline is shown in Fig. 4. Each step in the figure is a group of logically related operations. The following paragraphs discuss the processing steps shown in Fig. 4.

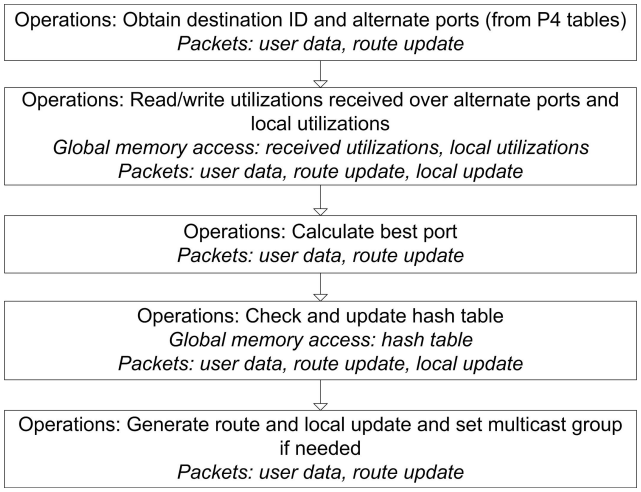



**FIGURE 4. Ingress pipeline - high-level program flow. The figure presents processing steps, which types of packets are involved in those steps, and which registers in the global memory are accessed.**

In the first step from Fig. 4, the program uses P4 tables to translate the destination IP address into the destination identifier using the longest prefix match. This is performed for each data packet. For TIMP control packets, the destination identifier is extracted from the packet. Then, for all packets, the second P4 table is used to obtain the list of alternate next hop ports based on the destination identifier. The destination identifier and the list of alternate ports are stored in the packet thread-local variables for use in the following stages of the pipeline.

In the second step from Fig. 4, the P4 program handles access to the registers which contain the path cost received from the neighbours and the local port costs. The path costs received from the neighbours are updated according to the received TIMP route update packets. The local port costs are updated using local update packets which are created at the end of the ingress pipeline and recirculated to its beginning. Finally, path costs for alternate ports are stored in the thread-local variables for further processing of the packet.

The third step in the program flow presented in Fig. 4 is the calculation of the best port. This part of the P4 program is implemented using thread-local variables since all the needed data was transferred to those variables in the previous two steps. The calculation firstly uses the path costs received from the neighbours and the local port costs to calculate the price over each of the alternate ports for the destination identifier obtained in the first step. Then the program finds the alternate port with the smallest price.

The fourth step of the program flow presented in Fig. 4 performs operations on the hash table. The hash function is performed on the packet header fields, similarly to ECMP. The value of the hash function is the index into register arrays which comprise the hash table. Register arrays contain output port identifiers and the time at which the last packet belonging to this flowlet traversed the switch. The detection of the new flowlet occurs when the hash table entry calculated for the packet is not valid. The detection of the flowlet expiration is determined by the definition of the flowlet: the flowlet will expire after a predefined period of time without packets. Upon the change of the number of flowlets on any port, the program will generate the local update packet which will update pipeline global memory, as introduced earlier in this chapter.

P4 pipeline executes processing only when the packet is traversing the switch. Hence, the check for flowlet expiration also has to be performed during the packet processing. The TIMP P4 implementation uses each data packet to check hash table entries for expiration. According to the rule that each part of the global memory can be accessed from a single pipeline stage, the implementation splits the hash table in two parts in order to check two hash table entries for expiration. The first of these two entries is determined by the hash value of the packet. The second hash table entry is checked in the other part of the hash table memory. If the processed packet is control packed instead of a traffic packet, both parts of the hash table memory are checked, based on the round robin counters.

In the final step of the ingress pipeline processing, the P4 program checks if there is a need to send local update packets or route update packets, i.e. if the local port costs or best path to the observed destination have changed.

Generation of the route update packets is accomplished by using multicast groups. Multicast groups are defined with the goal of creating an exact number of replicated packets.

The size of the hash table is determined according to the expected number of flows and the available memory. However, a small probability of a hash collision remains. This probability is small enough not to affect flowlet load balancing, but it has to be handled in order not to send packets on the wrong path. TIMP has protection against misdirected traffic due to hash collisions. The P4 program has the list of alternate ports which lead to the destination, and it detects if the hash table directs the packet to a port that is not on that list. If that happens, the P4 program will direct the packet on the correct alternate path with the smallest utilization.

### B. EGRESS PIPELINE PROCESSING

The goal of the egress pipeline processing is to perform header selection for packets which are generated using multicast groups.

If the packet which arrives at the egress pipeline has a valid IP header, the egress pipeline checks if the port to which the packet is directed is the port to which the data packet should be sent. In that case, possible update headers are set to invalid, and the data packet is forwarded on the next hop on its path to the destination switch. The information to which port the data

packet should be sent is added to the custom packet metadata in the ingress pipeline.

If the port to which a packet is directed is a special recirculate port, then all headers except local update header are disabled.

In other cases, all headers except route update header are disabled, and the packet is sent as a route update packet. This is possible because multicast groups are defined so that they contain only ports to which any of the data packet, local update packet or route update header should be sent.

### C. TIMP PIPELINE LATENCY

TIMP introduces advanced routing protocol features to the packet processing pipeline, which can cause increased length of the packet processing pipeline. We will shortly assess the complexity of the TIMP P4 program available at [2] in terms of the number of required operations. If we observe Fig. 4, only the second and third steps do not have a fixed length. The second step scales linearly with the number of switch ports, with two if-else constructs per port. The third step scales linearly with the selected maximal number of alternate paths. The egress pipeline has a fixed and small number of operations.

In the implementation [2], we can observe that all steps except the second step perform the number of operations usual for P4 solutions. The second step will contain more operations for high-radix switches. Each port introduces two if-else blocks per port, which translates to two pipeline stages per port. If we assume a pipeline clock frequency of 1GHz, which results in the delay of 1ns per stage, each port will introduce a delay of 2ns. The ports toward servers are not included in TIMP. Packets arriving at their destination ToR switch are forwarded towards the server using P4 table [2].

By observing P4 implementation available at [2], we can conclude that TIMP delay will not exceed typical pipeline latency of data center switch of 500ns [19].

### D. DETECTION OF FLOWLET EXPIRATION

TIMP enables immediate detection of new flowlets and generation of update packets which transfer network state to other switches.

Flowlet expires after a period of inactivity, i.e. after the predefined time period $T_f$ has expired since the last packet of the flowlet traversed the switch. In this section we analyze the additional time needed to detect flowlet expiration after the time period $T_f$ has expired.

TIMP checks hash table entries for the flowlet expiration in a round-robin fashion, one entry with the traversal of each data packet and two entries per update packet. For data packets, TIMP also checks hash table location determined by this packet. If we take into account guaranteed expiration detection using round robin check we can simply calculate the detection time based on the hash table size and the packet rate passing through the pipeline. With the assumed pipeline clock of 1GHz, with high traffic intensity, one packet would be processed every nanosecond and a hash table with 65536 entries would be completely scanned in 65 $\mu s$. This value is close to the typical flow interarrival time per switch of 50 $\mu s$, discussed in section IV-E. This implies that most of expired flowlets will be detected between two flows towards any of the destination switches.

Additionally, for many flowlets expiration detection time will be shorter than the time needed by the periodic scan to traverse the hash table. Detection time is reduced since the hash table entry corresponding to the processed packet is checked in addition to the round robin check.

## VI. PERFORMANCE EVALUATION

We have implemented Contra, DASH and TIMP in the ns-3 network simulator. This implementation is made publicly available at [2]. Results presented in this paper can be reproduced by following simple steps described in the github page [2].

Figure 5 presents topologies used in the evaluation. Switches are represented with circles. In the leaf-spine topology, the servers are connected only to the switches in the bottom row. In the other topologies each switch is used to connect servers to the network.

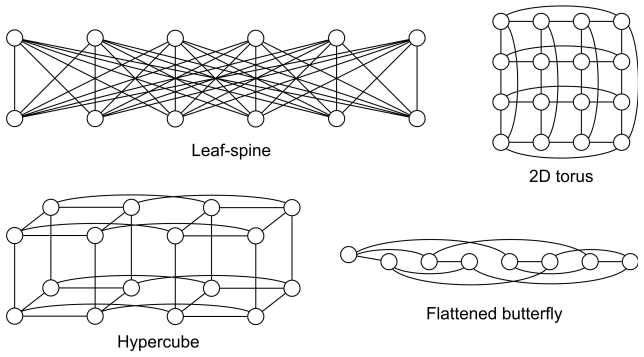



**FIGURE 5. Data center topologies used in the evaluation. Circles represent switches. In leaf-spine topology servers are attached only to servers in the lower row. In other topologies servers are attached to each switch.**

The goal of the evaluation is to measure flow completion times (FCT) which are important indicators of the data center performance. As mentioned before, partition-aggregate applications set a time limit until the aggregation server waits for responses from the worker servers, and FCT can show the ability of routing algorithms to meet these deadlines. In addition to that, the shorter FCT indicates better load balancing, i.e. more even network utilization and shorter paths of packets.

### A. LEAF-SPINE TOPOLOGY

Firstly, we evaluate the performance of data-plane routing protocols in the leaf-spine topology. Leaf-spine is the dominant topology in modern data-centers. The leaf-spine topology used in the evaluation is shown in Figure 5. This topology has twelve switches, with six leaf switches in the lower row and six spine switches in the upper row. Each switch of the

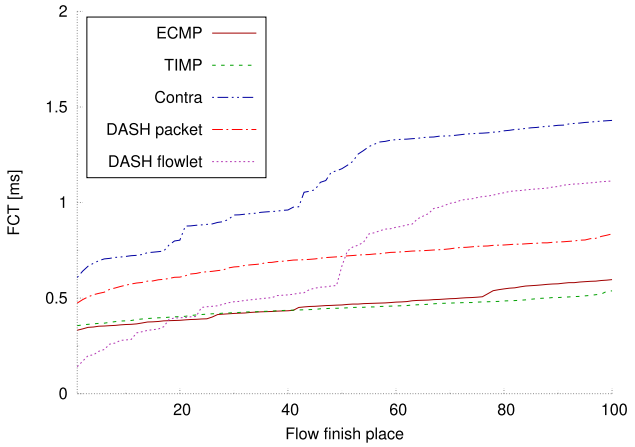
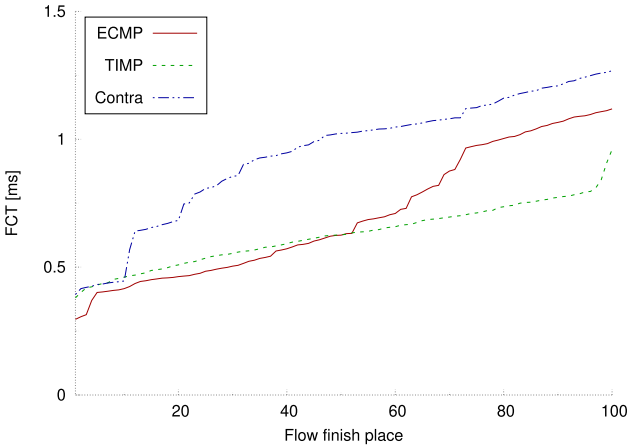
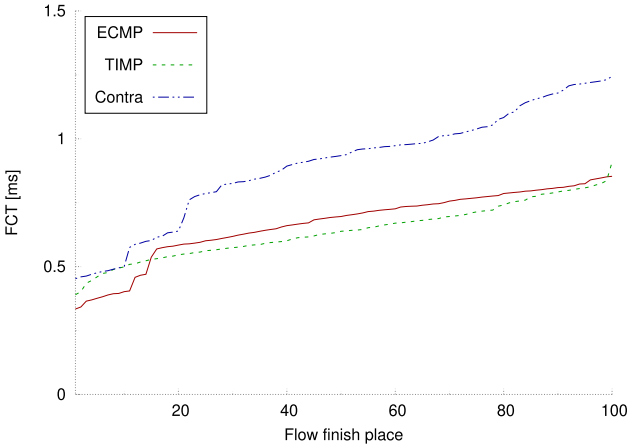


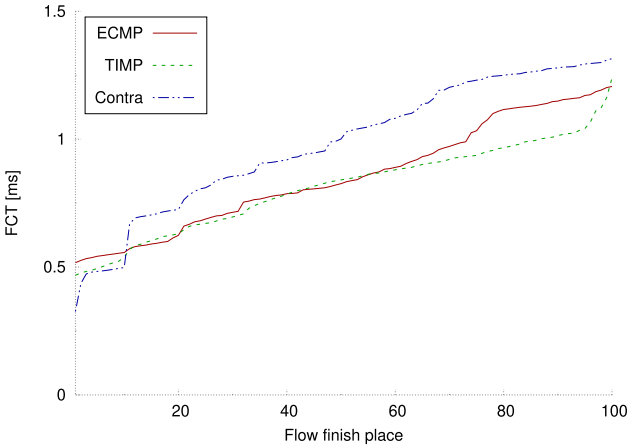

**FIGURE 6.** Flow completion times of partition-aggregate applications in the leaf-spine topology. TIMP has shorter flow completion times than other protocols.




(a) Torus topology




(b) Hypercube topology




(c) Flattened butterfly topology

**FIGURE 7.** Flow completion times of partition-aggregate applications in three non-ECMP topologies. TIMP performs significantly better than ECMP and Contra in torus topology. In hypercube and flattened butterfly topologies the difference between TIMP and ECMP is smaller.

lower row is connected to each of the switches in the top row. Servers are connected only to the leaf switches, and paths between any two leaf switches have the same length of 2. Thus, ECMP can perform load balancing over six paths between two leaf switches.

Leaf switches have the role of top-of-the-rack (ToR) switches, to which servers located in one data center rack are connected. In the simulation 42 servers are connected to each of the leaf switches. All links in the simulation have 10 Gb/s transmission speeds.

The background traffic is generated by random selection of communicating servers using uniform distribution. The flow sizes from the distribution of Web search workload used in the HULA paper [4]. The flow sizes in Web search workload are [5 kB, 7 kB, 10 kB, 20 kB, 30 kB, 50 kB, 100 kB, 500 kB], with probabilities [0.5, 0.1, 0.1, 0.1, 0.1, 0.05, 0.04, 0.01] respectively.

The flow interarrival times are taken from the exponential distribution. By adjusting the mean of the exponential distribution we obtain the target mean total throughput. In this simulation, we select flow interarrival times typical for data centers, as discussed in Section IV-E. With the flow interarrival of 50 $\mu s$ per switch and six switches which have servers attached, we obtain network flow interarrival time of 8.333 $\mu s$. With the distribution of flow sizes used in the simulation, this results in the average total network traffic of 19.86 Gb/s.

In the simulation we measure the flow completion times of flows belonging to partition-aggregate applications. In the applications the flows are transmitted from the worker servers to the aggregation server. Each application has one aggregation server and ten worker servers. The flows from the worker servers start simultaneously and each flow has the size of 30 kB. We start ten partition-aggregate applications. All aggregation servers are located in the same rack, i.e. they are attached to the same leaf switch. The worker servers are attached to other switches in the network. The probe period for Contra and HULA is set to 256 $\mu s$. All simulation parameters are available in the published implementation [2].

Figure 6 shows the measured flow completion times for six routing schemes. The arrival time of the i$^{th}$ flow is averaged

over five statistically independent runs. TIMP has lower flow completion time than Contra, DASH and ECMP. This result indicates that TIMP provides better performance than ECMP, which is still the main routing protocol in the data center networks. In this simulation Contra and DASH have longer flow completion times. That can be attributed to imbalance in flowlet distribution across spine switches, as a consequence of inter-probe intervals during which routes are not updated. As TCP incast flows arrive during time which is shorter than the probe period, the same forwarding parameters in switches will be used on all flows. The exception is when the probe arrives during TCP incast, which will provide one reconfiguration during the arrivals of the TCP incast flows.

### B. NON-ECMP TOPOLOGIES

The goal of the simulations in this section is to assess the performance of the partition-aggregate applications in the alternative data center network topologies. We evaluate the torus topology, flattened butterfly topology, and hypercube topology, which are shown in Figure 5. In these topologies we cannot evaluate DASH since it relies on the second routing protocol for the underlying routing. Thus, we evaluate topology independent data-plane routing protocols Contra and TIMP. Along with them, we evaluate ECMP in order to examine the performance gains of the data-plane routing protocols in these topologies.

In these simulations we use the same traffic setup as in the leaf-spine simulation, described in the previous section. As in the leaf-spine simulation, the mean value of the exponentially distributed flow interarrival times is selected to obtain the average flow interarrival time of 50 $\mu s$ per switch, which is typical for data center networks [18]. This results in the average network flow interarrival times of 3.125 $\mu s$ for hypercube topology and torus topology and 6.25 $\mu s$ for flattened butterfly topology used in the simulation. the average total network traffic amounts to 52.96 Gb/s for hypercube and torus, and 26.48 Gb/s for flattened butterfly.

Figure 7 presents simulation results. In torus topology, TIMP has significantly shorter FCT than Contra and ECMP. The difference between ECMP and TIMP is smaller in hypercube and flattened butterfly topologies.

## VII. CONCLUSION

TIMP has achieved the balance between the computational constraints of the programmable packet processing pipelines, and the need to detect, transfer and process network changes as fast as possible. This is enabled by preparing loop-free graphs of possible paths, and configuring them by the control plane. With this information, the routing algorithm which executes in the packet processing pipeline can limit the size of its data structures and the number of operations.

Using precalculated graphs of possible paths, TIMP routing protocol operates on a set of simpler graphs, instead of one network graph. One switch will store only information on which of its ports belong to which graph. Then it selects routes among a smaller number of ports, and it sends updates only to upstream ports since the other switches do not need them. Updates sent upstream are likely to encounter smaller queue sizes during TCP incast compared to the queue sizes in the downstream direction of the TCP incast flows. Finally, processing of route update packets is simplified since one packet contains information on one destination.

The precalculated graph can encompass all paths of practical importance. Hence, in the case of TCP incast, TIMP can direct flows to longer paths in order to achieve more even network utilization and avoid congestion. TIMP will direct flow along a path in the preconfigured graph of alternative paths. As stated in the section IV-A, the graph of alternative paths can be configured using a centralized network controller. The alternative paths may be static and symmetrical, or they may be adjusted dynamically, as the network controller would play a role of a slower, second level traffic optimizer. The algorithm in the TIMP switch is ready to accept changes in the alternative paths without the interruption of routing. Additionally, in case of link failure, an algorithm in the switch embedded software would be able to remove a port from a list of ports belonging to the graphs. This action would ensure that new flows are not directed to the failed link. It will also instantaneously redirect traffic from the failed link to another port, using the protection from hash collision described in Section V-B. For each packet, this protection checks if the output port belongs to the list of output ports and redirects the packet to one of the ports from the list when needed. This procedure doesn't require any additional processing in the packet processing pipeline.

The high performance of TIMP is accomplished by generation of the updates locally, in the switch which detects a new flow. This results in the shortest distance to the switches which need this information in order to properly direct flows. Such transfer of information may be observed as a reservation procedure. In this procedure a switch performs reservation as a new flow appears and immediately updates other switches which may select a path through this switch for some following flow. The advantage of this reservation procedure is its simplicity and speed. The simplicity enables implementation in the packet processing pipeline. The speed of such reservation procedure stems from the fact that it does not require coordination which involves flow endpoints and switches along the path of the flow. Typically, reservation involves communication protocol in which flow endpoints generate messages which establish state of reservation along the path. Such coordination would postpone the start of the time critical flow and degrade the performance. The implementation in the packet processing pipeline enables TIMP to inform other switches at the speed at which new flows arrive, thus being able to direct the new fastly arriving flows along the least used path. This introduces determinism in data center routing with the goal of handling traffic patterns generated by the distributed applications. Since bursts of flows can be properly routed using this procedure, applications are not constrained by the possibility of congestion due to TCP incast. TIMP can be combined with advanced congestion control algorithms

such as Swift [8]. TIMP's ability to distribute TCP incast flows evenly across available paths combined with Swift's ability to regulate congestion window in order to achieve line throughput without losses would provide high-performance routing in the case of large-scale TCP incast.

By taking into account each flow, TIMP can perform well at the times when data center applications generate flow patterns for which the statistical approach to routing has significant drawbacks. If a routing scheme would require applications to provide traffic demands, that would introduce additional delay. Instead, the applications communicate freely, and TIMP switches exchange network state information as they direct the first packet of the flow through the network.

The performance evaluation has shown that TIMP improves network performance in the widely used leaf-spine data center topology. The performance evaluation has also shown that TIMP enables the introduction of non-ECMP topologies in data centers by providing high-performance multipath routing. This would remove the requirement that data center topologies need to have as many equal-cost paths as possible, and enable benefits which alternative topologies may provide. With its ability to provide routing in arbitrary topology TIMP can facilitate the introduction of topologies which may bring advantages such as shorter cable length, ability for modular build using pre-built containers and other.

This paper is accompanied by the publicly available ns-3 simulation code and P4 implementation of TIMP [2]. Since TIMP uses standard P4, it can be implemented in existing switches that support P4, and used in existing data center networks. Beside the flow completion time metrics presented in this paper, the published ns-3 simulation code has prepared scripts which generate results for the following additional metrics: number of dropped packets, queue sizes seen by arriving packets, routing overhead and end-to-end packet delay. In addition to measurement of flow completion times in the applications, the implementation provides the graphs of flow completion times measured in the transport layer.

If we consider the number of data centers in use today, it is clear that any improvement in the efficiency of the data center network would result in large savings. By improving algorithms, instead of improving and adding hardware, data centers would become more energy efficient and less expensive.

## REFERENCES

[1] J. Kim, J. Balfour, and W. J. Dally, "Flattened butterfly topology for on-chip networks," *IEEE Comput. Archit. Lett.*, vol. 6, no. 2, pp. 37–40, Feb. 2007.
[2] *Implementation of the TIMP Data Center Multipath Routing Protocol*. Accessed: Sep. 16, 2021. [Online]. Available: https://github.com/nmaksic/timp
[3] M. Alizadeh, T. Edsall, S. Dharmapurikar, R. Vaidyanathan, K. Chu, A. Fingerhut, V. T. Lam, F. Matus, R. Pan, N. Yadav, and G. Varghese, "CONGA: Distributed congestion-aware load balancing for datacenters," in *Proc. ACM Conf. SIGCOMM*, Aug. 2014, pp. 503–514.
[4] N. Katta, M. Hira, C. Kim, A. Sivaraman, and J. Rexford, "HULA: Scalable load balancing using programmable data planes," in *Proc. Symp. SDN Res.*, Mar. 2016, pp. 1–12.
[5] K. F. Hsu, "Contra: A programmable system for performance-aware routing," *Proc. Networked Syst. Design Implement.*, Feb. 2020, pp. 701–712.
[6] K. F. Hsu, P. Tammana, R. Becket, A. Chen, J. Rexford, and D. Walker, "Adaptive weighted traffic splitting in programmable data planes," in *Proc. ACM SOSR*, San Jose, CA, USA, Mar. 2020, pp. 103–109.
[7] S. Sinha, S. Kandula, and D. Katabi, "Harnessing TCPs burstiness using flowlet switching," in *Proc. 3rd ACM SIGCOMM Workshop Hot Topics Netw. (HotNets)*, Nov. 2004, p. 84.
[8] G. Kumar, N. Dukkipati, K. Jang, H. M. G. Wassel, X. Wu, B. Montazeri, Y. Wang, K. Springborn, C. Alfeld, M. Ryan, D. Wetherall, and A. Vahdat, "Swift: Delay is simple and effective for congestion control in the datacenter," in *Proc. Annu. Conf. ACM Special Interest Group Data Commun. Appl., Technol., Archit., Protocols Comput. Commun.*, Jul. 2020, pp. 514–528.
[9] S. Ghorbani, Z. Yang, P. B. Godfrey, Y. Ganjali, and A. Firoozshahian, "DRILL: Micro load balancing for low-latency data center networks," in *Proc. Conf. ACM Special Interest Group Data Commun.*, Los Angeles, CA, USA, Aug. 2017, pp. 225–238.
[10] K. He, E. Rozner, K. Agarwal, W. Felter, J. Carter, and A. Akella, "Presto: Edge-based load balancing for fast datacenter networks," in *Proc. ACM Conf. Special Interest Group Data Commun.*, London, U.K., Aug. 2015, pp. 465–478.
[11] P. Wang, H. Xu, Z. Niu, D. Han, and Y. Xiong, "Expeditus: Congestion-aware load balancing in clos data center networks," in *Proc. 7th ACM Symp. Cloud Comput.*, Santa Clara, CA, USA, Oct. 2016, pp. 442–455.
[12] M. Antic, N. Maksic, P. Knezevic, and A. Smiljanic, "Two phase load balanced routing using OSPF," *IEEE J. Sel. Areas Commun.*, vol. 28, no. 1, pp. 51–59, Jan. 2010.
[13] N. Maksic and A. Smiljanic, "Improving utilization of data center networks," *IEEE Commun. Mag.*, vol. 51, no. 11, pp. 32–38, Nov. 2013.
[14] N. Maksia, "Two-phase load balancing for data center networks using OpenFlow," *Telfor J.*, vol. 10, no. 1, pp. 8–13, 2018.
[15] C. Wilson, H. Ballani, T. Karagiannis, and A. Rowtron, "Better never than late: Meeting deadlines in datacenter networks," in *Proc. ACM SIGCOMM Conf.*, Toronto, ON, Canada, 2011, pp. 50–61.
[16] D. B. de Oliviera, D. Casini, R. S. de Oliviera, and T. Cucinotta, "Demystifying the real-time Linux scheduling latency," in *Proc. 32nd Euromicro Conf. Real-Time Syst. (ECRTS)*, Jul. 2020, pp. 1–4.
[17] A. P. Lapukhov and J. Mitchell, *Use of BGP for Routing in Large-Scale Data Centers*, document RFC 7928, IETF, Aug. 2016.
[18] A. Roy, H. Zeng, J. Bagga, G. Porter, and A. C. Snoeren, "Inside the social network's (datacenter) network," *Proc. SIGCOMM*, London, U.K., Aug. 2015, pp. 123–137.
[19] P. Goyal, P. Shah, N. K. Sharma, M. Alizadeh, and T. E. Anderson, "Backpressure flow control," in *Proc. Workshop Buffer Sizing*, Palo Alto, CA, USA, Dec. 2019, pp. 1–3.
[20] B. Vamanan, J. Hasan, and T. N. Vijaykumar, "Deadline-aware datacenter TCP (D2TCP)," in *Proc. SIGCOM*, Helsinki, Finland, Aug. 2012, pp. 115–126.

**NATAŠA MAKSIĆ** (Member, IEEE) was born in Belgrade, Serbia in 1983. She graduated from the Mathematical Gymnasium in Belgrade, in 2002. She received the B.Sc. and M.Sc. degrees in electrical engineering from Belgrade University, Serbia, in 2007 as the best student in her class, with the maximum average grade of 10. She received the Ph.D. degree in electrical engineering from Belgrade University, in 2014.

From 2008 to 2014, she was a Researcher with the Innovation Center of the School of Electrical Engineering at Belgrade University. Since 2014, she has been an Teaching Assistant at the Telecommunications Department at the School of Electrical Engineering. Currently, she works as an Assistant Professor at the School of Electrical Engineering.

Her research interests are communication networks and protocols. In particular, she is interested in improving performance of the data center communications networks.