MTS-PolKA: Weighted Multipath Traffic Splitting with Source Routing for Elephant and Mice Flows

Giancarlo O. dos Santos[1], Cristina K. Dominicini[1], Gilmar L. Vassoler[1], Rafael S. Guimarães[1], Isis Oliveira[1], Domingos Jose P. Paraiso[1], Rodolfo S. Villaça[2]

[1]Postgraduate Program in Applied Computing (PPComp)
Federal Institute of Espírito Santo (Ifes)
Serra, ES - Brazil

[2]Postgraduate Program in Informatics (PPGI)
Federal University of Espírito Santo (Ufes)
Vitória, ES - Brazil

{giancarlo, cristina.dominicini, gilmarvassoler, rafaelg}@ifes.edu.br

{isisolip, domingos.paraiso}@gmail.com, rodolfo.villaca@ufes.br

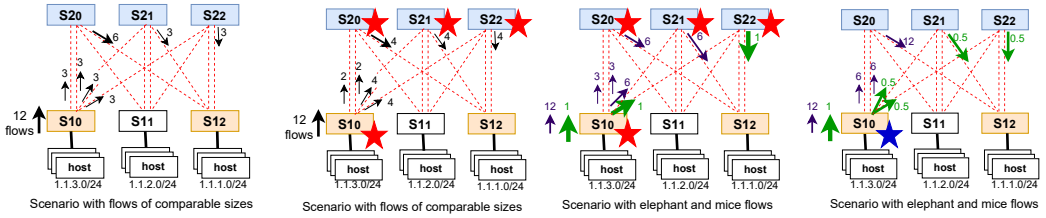
Abstract.

Modern datacenter networks rely on multi-root tree topologies, to ensure efficient and reliable connectivity. However, traditional traffic engineering solutions like Equal Cost Multiple Path (ECMP) and Weighted-Cost MultiPath (WCMP) often fall short in addressing dynamic and heterogeneous traffic conditions. Specifically, these approaches struggle with asymmetric topologies and the allocation of elephant flows, which require more granular and agile traffic management. This paper introduces MTS-PolKA, a novel traffic splitting mechanism designed to enhance traffic engineering in datacenter networks. MTS-PolKA enables packet-level traffic distribution across multiple paths by embedding a source routing label in the packet header that dictates the per-flow traffic division profile. Unlike existing methods that require reconfiguring network tables at each switch, MTS-PolKA enables dynamic adjustments across all switches in the path through simple packet header modifications, significantly enhancing the agility and efficiency of traffic management. The implementation of MTS-PolKA on programmable switches using the P4 language, along with the Residue Number System (RNS) and M-PolKA architecture, showcases its potential through experiments conducted in the Mininet emulator. The results highlight the ability of MTS-PolKA to maintain flow stability, explore network multipaths, and enable rapid reconfigurations of traffic division profiles, enhancing the performance and efficiency of modern datacenter networks.

1. Introduction

In recent years, datacenters have become foundational to the Internet's computing infrastructure. They support a myriad of distributed processing applications, from social collaboration to high-performance computing, across large-scale environments that can host over 100,000 servers [Dixit et al. 2013]. To ensure efficient and reliable connectivity, modern datacenters are typically organized using multi-root tree topologies, such as Fat-Tree and Clos, which provide multiple, cost-effective paths between servers [Huang et al. 2021].

Traffic balancing is essential to fully utilize available bandwidth, particularly as load fluctuations can rapidly alter the traffic conditions on these paths. Consequently,



(a) ECMP **(b) WCMP** **(c) MTS-PolKA**

Figure 1. Asymmetric division of multipath traffic in Clos Tier 2 topologies. (a) ECMP: each flow is assigned to a single equal cost path, without taking topology heterogeneity into account. (b) WCMP: each flow is assigned to a single path, and the flow distribution across paths can be configured by weighted groups tables in each *switch* (red stars). Elephant flows cannot be divided across multiple links. (c) *MTS-PolKA*: flows can be divided across multiple paths according to weights defined by the source (blue star) and inserted as a label in the packet header. Path modifications require only a change in this label and switches require only static traffic profile tables.

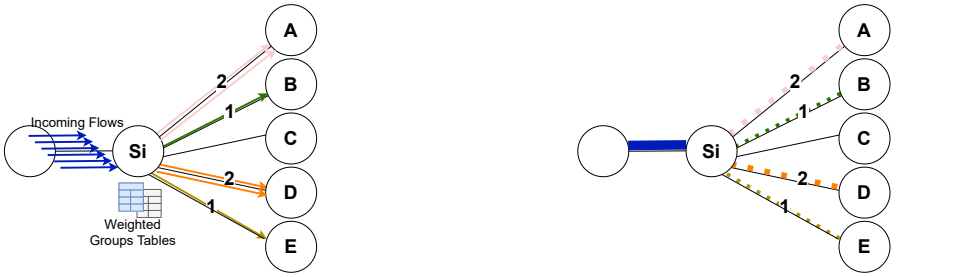
the ability to adjust load distribution on a short timescale according to current traffic patterns is crucial for optimizing performance [Hsu et al. 2020]. Thus, high performance in datacenter network architectures depends on effective traffic engineering. This requires network controllers or distributed agents to agilely select and balance paths to accommodate dynamic traffic patterns [Jyothi et al. 2015].

Traditional algorithms based on Equal Cost Multiple Path (ECMP) only consider symmetric and regular topologies due to their static routing approach, based on paths with equal bandwidth capacity and hop count [Hsu et al. 2020]. This assumption is often unrealistic for modern datacenter networks, where the following factors can invalidate these conditions [Zhou et al. 2014]: i) rapid fluctuations in traffic conditions, ii) failures in network nodes or links, iii) the presence of heterogeneous network components, and iv) asymmetric topologies.

Figure 1 illustrates a multi-root tree topology of the Clos Tier 2 type, which can provide multiple paths of equal length to the same destination but with varying bandwidth capacities. Such variations may arise from differing link capacities or dynamic traffic competition, requiring an asymmetric traffic distribution across the network. As illustrated in Figure 1a, the hashing strategy employed by Equal Cost Multiple Path (ECMP) fails to address this issue effectively.

Other existing approaches to solve this problem are WCMP (Weighted-Cost MultiPath) [Cui et al. 2021], based on the configuration of weights in traffic splitting tables, and *DASH* (Adaptive Weighted Traffic Splitting in Programmable Data Planes) [Hsu et al. 2020], based on an optimized data structure that exploits registers in a multi-stage pipeline to update the distribution in real-time. As shown in Figure 1b, these solutions enable the exploration of multiple paths, dynamically allocate traffic according to the specified weight ratios, and optimize the use of the available bandwidth across the links.

However, modifying path weights requires updating tables or data structures at each switch along the path. For instance, as illustrated in Figure 1b, this entails adjusting all tables on the network devices marked with red stars for each flow that is reorganized.



(a) Related works distribute *n* flows across *p* paths using tables with weights.

(b) Our work allows to divide any flow across *p* paths using labels with weights.

Figure 2. Problem: How to configure *switches* to allow weighted distribution of a flow on multipaths with packet granularity?

The more switches there are on the path and the greater the number of flows being managed, the longer it takes for the device tables to converge, thereby impacting the agility of the operation. Another characteristic of these solutions is that they distribute n flows across p paths using tables with weights, as shown in Figure 2a. In this way, each flow is assigned to a single path, preventing the distribution of one flow across multiple links. This limitation significantly affects elephant flows, which are characterized by large data transfers that consume substantial bandwidth or last for an extended period [Gilliard et al. 2024]. The second scenario in Figure 1b illustrates a situation where mice and elephant flows compete for bandwidth. Even if spare bandwidth is available on other paths, WCMP is limited to allocating only a single path at a time for the elephant flow. Other works employ packet spraying strategies to enable the division of a flow across multiple paths [Dixit et al. 2013, Huang et al. 2021], but they do not offer a way to differentiate policies for heterogeneous traffic.

Since elephant flows are especially prevalent in large-scale data processing tasks, such as big data analytics and machine learning model training, the ability to agilely and efficiently allocate network resources in these cases is crucial for the performance of modern datacenters [Wassie Geremew and Ding 2023]. However, while elephant flows account for approximately 90% of the total data volume, they represent only about 2% of the total number of flows [Benson et al. 2010]. Therefore, it is equally important to manage mice flows, which are characterized by their short duration and sensitivity to communication delays [Xie et al. 2024]. In summary, effective traffic engineering in datacenter networks requires a traffic splitting mechanism that dynamically and flexibly distributes flows across multiple paths to address heterogeneous traffic. This mechanism should support per-flow policies that allocate traffic proportionally based on the available bandwidth on each path and, when necessary, enable the splitting of a single flow across multiple paths.

To address this need, this paper aims to propose a solution that enables configuring how each switch i in a network path r distributes a flow f across its output ports with packet granularity at the speed required by modern datacenter networks. This distribution must follow pre-configured weight proportions for each subset p of its ports. For instance, as illustrated in Figure 2b, a switch S_i with six ports must allocate the traffic of an incoming flow f, as specified at the network ingress, according to the weight ratio 2:1:2:1 for the four ports $p = \{A, B, D, E\}$. It is important to note that proposing mechanisms for monitoring

flows and path allocation is not part of the scope of this work, which focuses specifically on the design of the traffic splitting mechanism itself.

The solution presented in this work introduces a multipath traffic splitting approach called *MTS-PolKA* (*Multipath Traffic Split Polynomial Key-based Architecture*), which is based on the Multipath Polynomial Key-based Architecture (M-PolKA) [Guimarães et al. 2022] and the Residue Number System (RNS). *MTS-PolKA* enables the selection of a traffic division profile, defined by weight proportions for each network flow, at the source using a label in the packet header. Each switch along the path maintains a static table containing the available traffic splitting static profiles, which are selected by this label.

The innovation in this approach is twofold: (i) its unique integration of weighted traffic splitting with packet-level granularity, which enables the configuration of per-flow policies depending on the selected labels; and (ii) its ability to adjust the distribution of traffic simultaneously across all switches in the path, through a simple modification of the label in the packet header of each flow entering the network. This method eliminates the need to reconfigure tables or data structures in each switch, enabling more agile, efficient, and dynamic traffic engineering in datacenter networks. For example, as illustrated in Figure 1c, a change in the packet header at a single point of traffic generation (indicated by the blue star) will propagate throughout the entire network topology.

As a proof of concept, we developed an efficient implementation for programmable switches using the P4 language and executed several experiments in the Mininet emulator. The results highlight the flexibility and expressiveness of our approach in exploring network multipaths, maintaining flow stability, and facilitating agile reconfigurations of traffic division profiles at the source. These capabilities collectively offer significant potential to improve overall performance and efficiency in data center networks.

The remainder of this paper is organized as follows. Section 2 presents some related work, highlighting the contributions of this work. Section 3 describes the design and implementation of *MTS-PolKA*. Section 4 presents and discusses the results of the experiments. Finally, Section 5 brings conclusions and future works.

2. Related Works

The traffic splitting problem can be categorized into four types based on scheduling granularity: flow, packet, flowlet, and adaptive-granularity [Li et al. 2021]. Flow-based approaches avoid packet reordering, but face long-tailed delays and low link utilization. Packet-based schemes, while enhancing link utilization, can have issues such as hash collisions and packet reordering, which may degrade network performance. Flowlet-based schemes manage bursts of packets with time intervals to reduce packet reordering. However, they can still suffer from low utilization due to their relatively coarse switching granularity. In this section, we compare several works that are more closely related to *MTS-PolKA*.

The most widely used method for balancing flows across different paths is ECMP, which employs hashing to evenly distribute flows among paths of equal capacity. However, it has little application in asymmetric path topologies [Valentim et al. 2019]. WCMP extends ECMP by enabling non-uniform load distributions, as illustrated in Figure 3. WCMP relies on a hashing-based data structure, and updating this structure requires

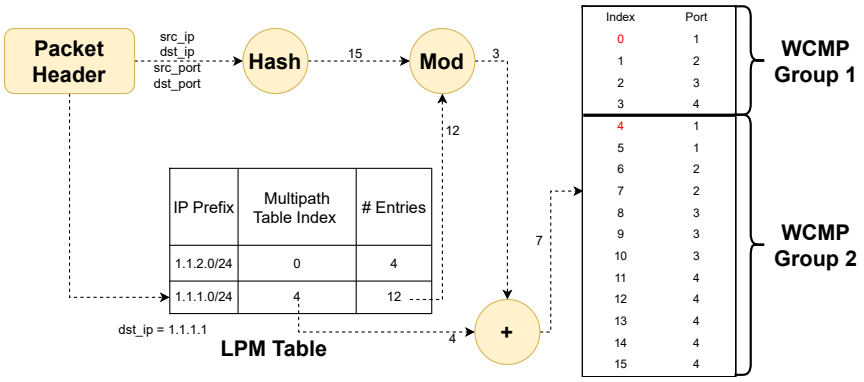


Figure 3. WCMP: Tables with weights.

modifying path IDs across multiple tables. Currently, such updates are impractical due to the limited number of memory accesses per packet at each stage of the pipeline in a programmable device [Rottenstreich et al. 2018].

As outlined in [Hsu et al. 2020], approaches such as CONGA, HULA, and CONTRA focus on load balancing directly within the data plane, dynamically updating the data plane state without requiring control plane intervention. Despite their advancements, these methods have limitations, since they typically consider only one ”best” path at a time, which can lead to underutilization of other potential paths. On the other hand, DASH improves this by distributing flows across multiple paths proportionally to the current load, dynamically adapting to changes [Hsu et al. 2020]. However, DASH is constrained by the number of supported paths [Robin and Khan 2022] and relies on state changes in network devices.

Another solution integrates MultiPath TCP (MP-TCP) with Source Routing (SR). However, this MP-TCP + SR approach requires specific configurations on endhosts [Pang et al. 2017], which may not be feasible in environments where significant changes to the endhost protocol stack are restricted, such as public cloud platforms. Other methods address the multipath traffic division issue by employing packet spraying strategies, such as Random Packet Spraying (RPS)[Dixit et al. 2013]. This technique, however, faces challenges with packet reordering, which can adversely affect TCP congestion control. Queuing Delay Aware Packet Spraying (QDAPS)[Huang et al. 2021] attempts to alleviate this issue but remains sensitive to topology asymmetry.

The Table 1 summarizes the comparison of some multipath traffic splitting related works in datacenter networks, considering as parameters: granularity, presence of a stateless core, routing method, traffic splitting layer and disadvantages associated with each proposal. The presence of a stateless core indicates high agility in (re)configuring paths and weights, which is a distinguishing characteristic of *MTS-PolKA*.

Each approach has trade-offs, with flow-level and flowlet-level schemes struggling with low resource utilization and packet-level schemes experiencing performance degradation due to reordering. To address these issues and combine the advantages of flow-level and packet-level approaches, *MTS-PolKA* was introduced, offering both flow-based and packet-level granularity. It achieves this by employing a weighted traffic splitting mechanism in the core network, which distributes each packet based on a weight label (*weightID*) and a multipath route label (*routeID*). Additionally, the edges classify ingress flows and as-

Table 1. Related Works of Traffic Splitting in Datacenter Networks

Method	Stateless core	Granularity			Routing Method	Layer		Disadvantages
		Flow	Flowlet	Packet		Network	Transport	
ECMP	✗	✓			*Hash Table*	✓		*Collisions in the hash table.*
WCMP	✗	✓			*Hash Table*	✓		*Table scalability*
HULA	✗		✓		*Hash Table*	✓		*Only one best path.*
DASH	✗		✓		*Hash with registers*	✓		*Number of supported paths*
MP-TCP + SR	✓		✓		*SR*		✓	*End host changes.*
RPS	✗			✓	*Random*	✓		*Packet reordering.*
QDAPS	✗			✓	*Random+Hash Table*	✓		*Sensitive to topology asymmetry.*
MTS-Polka	✓	✓		✓	*SR+Hash Table*	✓		*Packet reordering and Header overhead.*

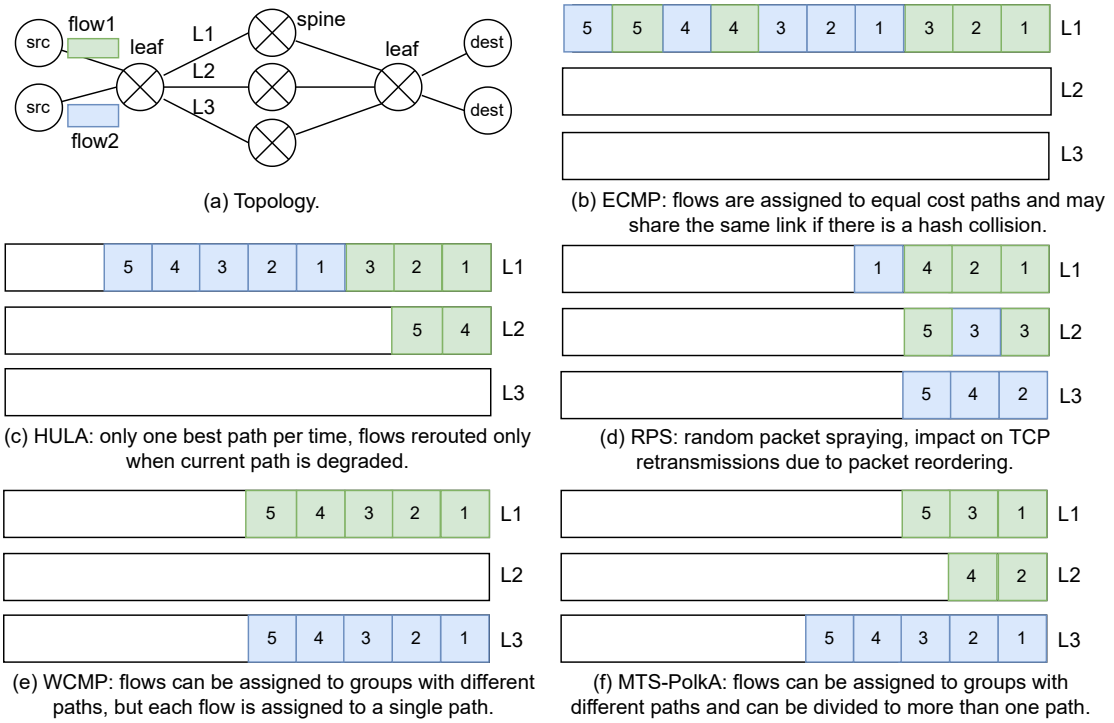


Figure 4. Related works: example illustrating the link queues for a scenario with flows of comparable sizes and symmetric links (Adapted from [Huang et al. 2021]).

sign the appropriate labels according to per-flow policies. Figures 4 and 5 provide a more detailed illustration of this discussion.

Figure 4 illustrates a scenario involving flows of comparable sizes and symmetric links. In ECMP, each flow is assigned to a single equal-cost path, and may end up sharing links with other flows due to hash collisions, even if idle paths are available. HULA, on the other hand, reroutes flows when a path becomes congested but only provides one best path at a time per flow, which can result in link underutilization. RPS distributes packets randomly across links, which can lead to packet reordering and increased TCP retransmissions. Conversely, WCMP assigns flows to different groups and paths, yet each flow is limited to a single path at a time, potentially causing insufficient bandwidth allocation. Finally, MTS-PolkA assigns flows to multiple path groups and allows distribution across more than one path per flow. While this approach offers more controlled spraying compared to RPS, it still faces challenges with packet reordering.

Figure 5 illustrates a scenario with both elephant and mice flows over asymmetric

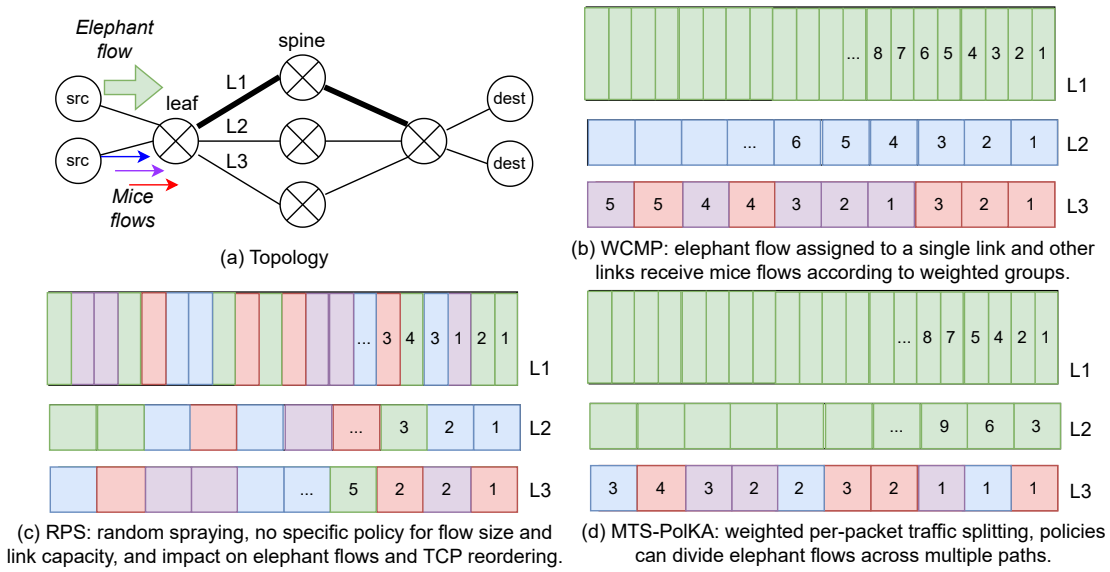


Figure 5. Related works: example illustrating the link queues for a scenario with elephant and mice flows and asymmetric links.

links. In this example, WCMP assigns the elephant flow to a single link with higher capacity, while distributing the mice flows across other links based on weighted group definitions. Although this allocation improves performance compared to ECMP, it may still fall short if the bandwidth-intensive elephant flow could benefit from additional links. In contrast, RPS does not apply specific policies per flow size or link capacity, leading to random and independent spraying of all flows. This approach can significantly impact elephant flows and exacerbate TCP reordering due to both flow size and link capacity asymmetry. Conversely, *MTS-PolKA* facilitates weighted per-packet traffic splitting and can assign elephant flows to multiple paths with higher network utilization. However, it requires more complex queuing mechanisms, as discussed in [Huang et al. 2021], to effectively handle packet reordering when dividing packets over asymmetric paths.

3. Design and Implementation

MTS-PolKA should allow the source to select the desired traffic distribution across multiple paths using labels in the packet headers of the flows. Each switch along the selected paths should maintain a static table with a set of available traffic distribution profiles for selection based on the packet label.

3.1. M-PolKA Routing

MTS-PolKA employs the *M-PolKA* protocol to implement multipath source routing based on RNS (Residue Number System). In this approach, routing decisions within core network nodes rely on polynomial modulo operations applied to a route label. *M-PolKA* was selected because it is the only source routing method identified in the literature that can encode a route label for multipath cycles or trees in a topology-agnostic manner.

As shown in Figure 6, the *MTS-PolKA* architecture involves core switches (`Si`), edge (`Edge`) switches, and a logically centralized controller (`Controller`), allowing multipath routing through 3 (three) polynomial identifiers with binary representation:

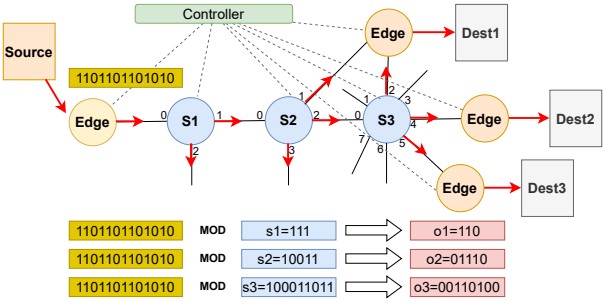


Figure 6. Example of the original M-PolKA routing.

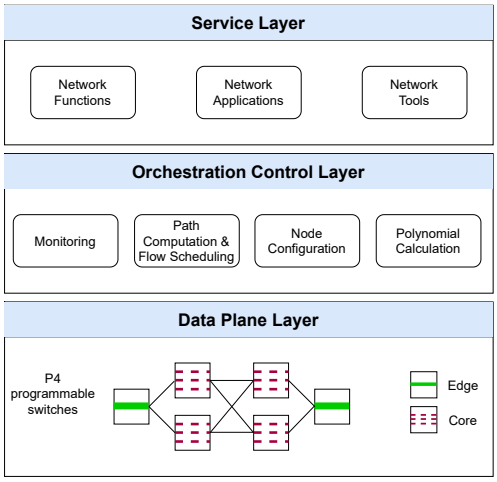


Figure 7. *MTS-PolKA* Architecture.

routeID, *nodeID* and *portID*. The transmission state of the output ports (*portID*) is given by the remainder of the binary polynomial division (i.e., a modulo operation `MOD`) between the route identifier (*routeID*) and the node identifier (*nodeID*). As a result, *M-PolKA* clones the ingress packet to all active output ports selected in the *portID*.

In the example, the controller installs flow entries at the edge to embed the route identifier, *routeID* = 1101101101010, in the packets of a flow. The result of module operations on each of the core switches is represented by red arrows. For example, the remainder of $R(t) = 1101101101010$ when divided by $s_3(t) = 100011011$ is $o_3(t) = 00110100$. Thus, on s_3, ports 2, 4 and 5 forward this flow to the next hop, while the other ports do not forward it.

The M-PolKA routing allows the source to select the multiple paths of a flow according to a multipath cycle or tree. However, M-PolKA only clones packets for all ports that belong to the selected path, without allowing traffic splitting. This gap was filled by *MTS-PolKA*, which proposes the application of *M-PolKA* to divide traffic with weight proportion between output ports. This allows exploring path diversity to provide load balancing.

3.2. *MTS-PolKA* Architecture

MTS-PolKA adds traffic splitting functionalities to the M-PolKA architecture (shown in Figure 6), which is composed of core nodes (in blue), edge nodes (in yellow), and a logically centralized controller. The layers of the *MTS-PolKA* architecture are described below and depicted in Figure 7:

- **Data Plane Layer:** Assumes a physical topology composed of P4-enabled programmable *switches*, which act as edge nodes for classifying flows and adding labels to the packets, or core nodes for splitting multipath traffic by weight ratio.
- **Control and Orchestration Layer:** Composed of four modules: i) `Monitoring`: discovers the topology and collects data on the network state, such as link failure and congestion, to provide a feedback cycle for orchestration decisions; ii) `Path computation and flow scheduling`: manage and distribute network

traffic across multiple paths or links to optimize resource utilization and performance; iii) `Node configuration`: configures *nodeID*s and traffic profile tables on core nodes, in addition to configuring entries on edge nodes for flow classification; and iv) `Polynomial calculation`: calculates *weightID*s, *routeID*s and *nodeID*s for the defined paths and traffic distributions.

- **Service Layer:** Responsible for implementing network functions, user applications, and other tools used in the architecture implementation and validation stages.

The *MTS-PolKA* controller calculates three polynomial identifiers: i) ***nodeID***, an identifier assigned to the core *switches* in the network configuration; ii) ***routeID***, a multipath route identifier; and iii) ***weightID***, a weight identifier that defines the traffic splitting profile at each *switch* in the path.

Both *routeID* and *weightID* are calculated using the Chinese Remainder Theorem (CRT, *Chinese Remainder Theorem*) [Dominicini et al. 2020] and incorporated into packets by edge *switches*, which have flow entries that can be proactively installed by the controller, without the need to interact with the control plane for each packet that enters the network.

On the data plane, the polynomial *mod* implementation exploits the CRC (Cyclic Redundancy Check) operation supported by the equipment. As detailed in [Guimarães et al. 2022], the size of the *routeID* is given by multiplying the topology's maximum number of hops by 2 bytes (for use of the CRC16 algorithm in topologies with up to 4,080 nodes). The same logic can be adopted to calculate *weightID*. The implementation of this article adopted a maximum number of 10 hops with a 20-byte header for each *routeID* and *weightID* field. Thus, there is a cost in terms of bits in the packet header to exploit the RNS system in routing, but this can be reduced by applying some known coding techniques [Dominicini et al. 2020]. Although not implemented in this version, it is also feasible to encode both labels (*routeID* and *weightID*) into a single 20-byte field to reduce this overhead.

Furthermore, two polynomial identifiers are calculated, in the data plane, for each packet: i) ***portID***, an identifier that represents the active output ports of each core switch, i.e., which ports will transmit a proportion of the traffic to be forwarded; and ii) ***profileID***, an identifier obtained by the modulo operation between the *weightID* of the packet and the *nodeID* of the switch, which represents the traffic division ratio profile on the active output ports.

3.3. Control Plane

As illustrated in Figure 8, the *MTS-PolKA* controller has two main tasks: i) defining the route, based on the definition of a *multicast* tree that identifies, for each one of the *switches* on the path, which are the active ports that should forward packets, as represented by this *multicast* tree; and ii) define the weights for dividing traffic on each active port of each of the nodes in the *multicast* tree.

At the core of the network, the controller is responsible for configuring the *switches* with the *nodeID*s, and the static tables of each *switch* with the available traffic division profiles. At the edge, the controller provides a set of multipaths and traffic division profiles through *routeID*s and *weightID*s, which are assigned to incoming flows for performing the load balance according to the network state. This solution exploits the diversity of

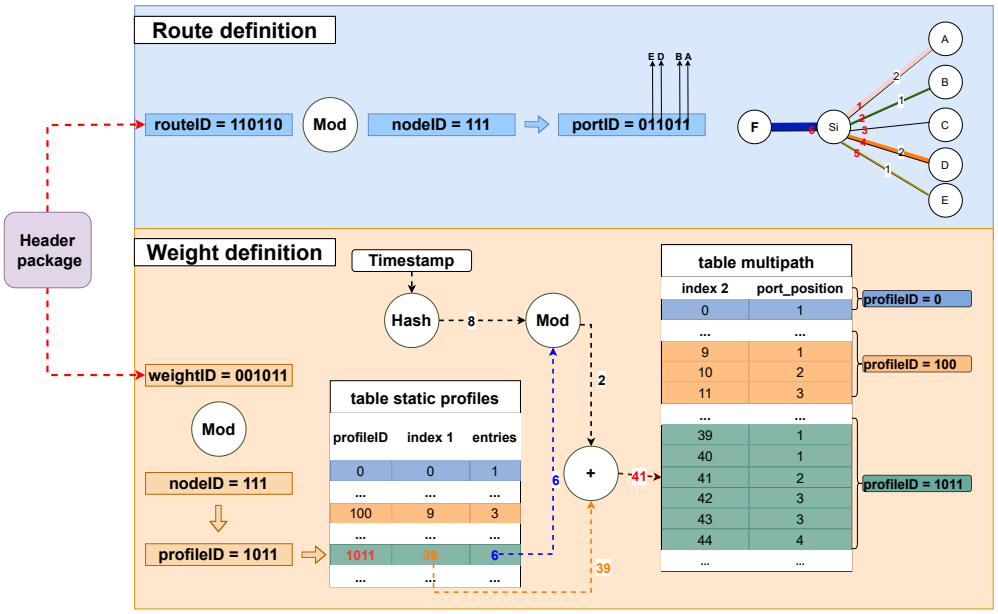


Figure 8. *MTS-PolKA*: Example of how a traffic division profile is selected by the source using a label in the packet header. Both tables are previously configured by the controller.

paths available on the network, resulting in efficient use of the bandwidth available on the network.

3.4. Data Plane

Each edge *switch* has a flow table, previously populated by the controller, which inserts the *routeID* and *weightID* labels into the packets of each flow. This table maps information about the flow (e.g., destination IP address, ports) into routing and load balancing decisions. Subsequently, these labels are used in each core *switch* to determine the state of the output ports and select the corresponding traffic division profiles, as shown in Figure 8.

Figure 8 exemplifies the operation of the data plane for a core switch *Si* that has 6 ports and a traffic distribution profile with a weight ratio of 2:1:2:1 for forwarding to switches A, B, D and E, respectively. When the packet arrives, the `Route Definition` operation must be performed using the modulo operation (`MOD`) between *routeID* = 110110 and *nodeID* = 111. The result of this operation defines the active output ports (*portID* = 011011) that will be used to forward the packet. In this way, traffic is forwarded through the ports corresponding to *bits* with a value of 1 in *portID* (from right to left). Thus, packets are transmitted to switches A, B, D and E.

Then, it is necessary to find out which traffic division profile will be used to forward this packet to each active output port, according to the *weightID* defined in the packet. In a previous network configuration phase, the controller configured the *nodeID*s of the switches and two static tables (`table static profiles` and `table multipath`), which define, respectively, the supported traffic profiles and how traffic should be distributed across active ports according to the selected profile. In this way, there are several traffic profiles available on each core switch, which can be selected by the edge for the packets of each flow, without any additional configuration on the core switches. The number and

variety of profiles supported is a control plane decision.

Following the example illustrated in Figure 8, the table `table static profiles` determines "how" the next table (`table multipath`) should be accessed according to the division profile selected for that flow. By employing the modulo operation between *weightID* = 001011 and *nodeID* = 111, the switch *Si* obtains the *profileID* = 1011.

For this profile, in the table `table multipath` there are 6 entries, starting from position 39 (*index1*). In the table `table multipath`, the profile *profileID* = 1011 is represented in green and the number of lines defines the weights for each active output port: 2 entries for the first active port ($port\_position = 1$), 1 entry for the second active port ($port\_position = 2$), 2 entries for the third active port ($port\_position = 3$) and 1 entry for the fourth active port ($port\_position = 4$), with a total of 6 entries. Therefore, the profile *profileID* = 1011 must divide the traffic in the following proportion: the first active port must receive 2/6 of the traffic, the second active port must receive 1/6 of the traffic, the third active port must receive 2 /6 of the traffic and the fourth active port should receive 1/6 of the traffic. The number of entries in the `multipath table` represents exactly the expected proportion and will be explored by a *hashing* function.

To distribute the packets of a flow across the active output ports according to the proportions defined in the profile, switch *Si* performs a hashing function with the packet's ingress timestamp. In this hypothetical example, consider that the result of hashing is 8, and that this value is subjected to an integer modulo operation by the number of entries in green ($entries = 6$ in `table multipath`), whose result is = 2. This result must be added to the index ($index1 = 39$), obtained in the `table static profiles` table, resulting in the index $index2 = 41$ of the `table multipath`. Finally, this $index2 = 41$ line indicates that the second active port ($port\_position = 2$) should be chosen as the output port for this specific packet.

In this example, as *portID* = 011011 was previously calculated in the route definition step, the second active port means that the packet must be forwarded through port index 2 (port indexes start at 1, from right to left in *portID*). In the figure, port 2 represents the output port that forwards the packet to node B. Applying the port selection algorithm described so far, and considering the random variation of the hashing of the ingress timestamp, the traffic associated with the profile *profileID* = 1011 will be spread among the active ports with a weight ratio equal to 2:1:2:1.

Figure 9 presents the same logic as the example of Figure 8, but representing the implementation of the data plane pipeline of the core switches, using the P4 language. In the parsing step, if the `etherType` field is `TYPE SR`, indicating the presence of the *MTS-PolKA* header, the next step is to read the *routeID* and the *weightID*, via `lookahead`. Then, the `ingress` block performs polynomial modulo operations between *routeID* and *nodeID*, and between *weightID* and *nodeID*, to calculate the states of the transmission ports (*portID*) and the traffic division profile (*profileID*), respectively.

Afterwards, the *profileID* is used to search the table `table static profiles` and retrieve the values of the variables *entries* and *index1*. These values are used with the hashing of the ingress timestamp to obtain the variable $index2 = index1 + hash(tstamp) \bmod entries$. Then, this last index is used to access the `table multipath` table and identify which active port should transmit that specific packet ($port\_position$). Finally,

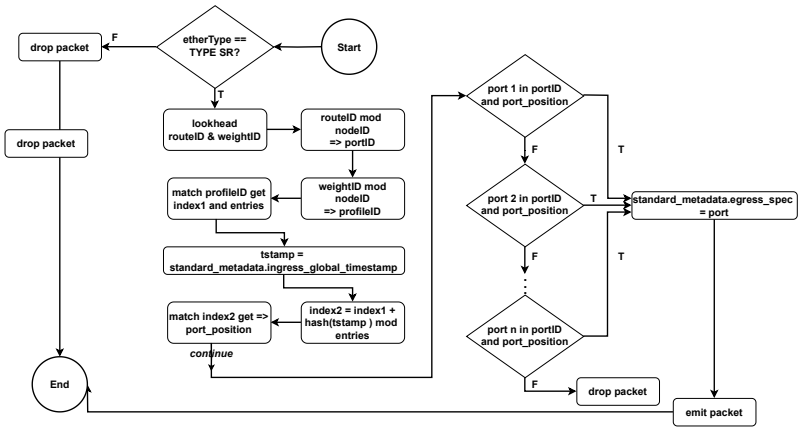


Figure 9. *MTS-PolKA*: Pipeline of the core switches using the P4 language.

a search is conducted in the state vector for the active port that matches the value of $port\_position$. Once the corresponding port is identified, the packet is transmitted through it.

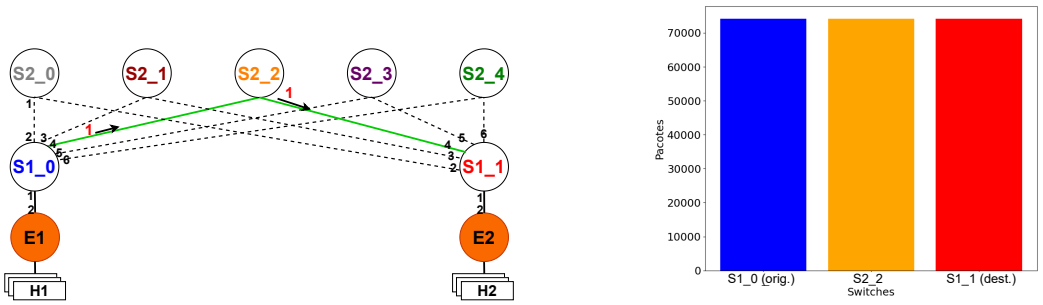
4. Experimental evaluation

The development and testing environment used a *PowerEdge* R650 server, with 32 GB of *RAM* and a 2.4 GHz Intel *Xeon Silver* 4314 processor. The *Mininet* emulator was used to conduct experiments, allowing the compilation of P4-16 programs and configuration of *switches bmv2*. To perform polynomial calculations, such as the *RNS* calculation of *nodeID*s and *routeID*s, we use a *Python* library, with the *galoistools* package from the *sympy* library. Furthermore, a control plane application was implemented that interacts with the *switches* through the *bmv2 simple_switch* CLI and connects via the *Thrift RPC* port. This application implements the Control and Orchestration layer, as in Section 3.2, except for the monitoring module, which is outside the scope of this article.

Four experiments were carried out: i) a functional test, to demonstrate the application of different traffic division profiles; ii) an agile reconfiguration test of traffic division profiles at the source; iii) a test with multiple concurrent flows; and iv) a flow completion time comparison with related works. The test involved measurements using the *bwm-ng* tool and was generated by *iperf3*, with one *host* as server and another as client. In tests 1 and 2, UDP packets were sent with a throughput of 10 Mbps. In test 3, UDP flows were sent with a throughput of 4 Mbps.

4.1. Test 1: Functional Validation

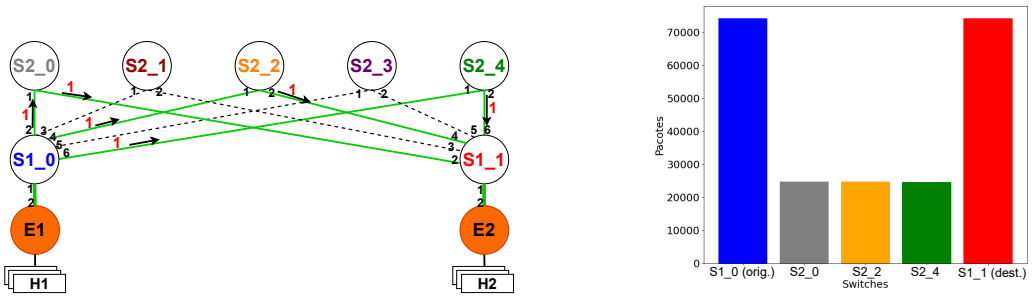
Consider the following scenario: three different traffic profiles were selected at the source for a specific core *switch* (`S1_0`), which represents, for example, an access *switch* in a *datacenter* network with hierarchical topology. The three profiles are represented in Figures 10a, 11a, and 12a which will be further detailed later in this section. The remaining *switches* maintained a constant traffic distribution using Profile 0, which represents, in this example, directing all traffic through the first active port. As already explained, the traffic division profile was determined by the origin, and inserted into the flow by the edge *switch*, through the use of two labels in the packet header: i) the *routeID*, to select the active ports



(a) Profile scenario 0

(b) Number of packets per switch

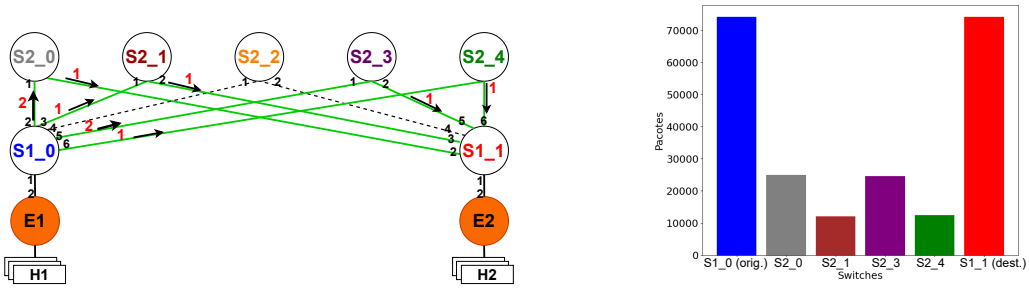
Figure 10. Test 1: profile 0



(a) Profile 100 scenario

(b) Number of packets per switch

Figure 11. Test 1: profile 100



(a) Profile scenario 1011

(b) Number of packets per switch

Figure 12. Test 1: 1011 profile

on each *switch* in the path and assemble the transmission tree; and ii) *weightID*, to select the proportion of weights.

Test with Profile 0 (Figure 10): Using the indicated transmission tree (Figures 10a) and a weight ratio of 1 on the first active port, all outgoing traffic was routed to port `S1_0-eth4` on *switch* `S1_0`. The results (Figure 10b) show the balanced division of UDP traffic according to profile 0 on *switches* `S1_0`, `S2_2` and `S1_1`.

Test with Profile 100 (Figure 11): Using the indicated transmission tree (Figure 11a) and a weight ratio of 1 :1:1, the *switch* `S1_0` divided outgoing traffic equally across its three active ports (`S1_0-eth2`, `S1_0-eth4` and `S1_0-eth6`). Figure 11b presents the results, illustrating the distribution of UDP packets following profile 4 (1:1:1) in *switches* `S1_0`, `S2_0`, `S2_2`, `S2_4` and `S1_1`.

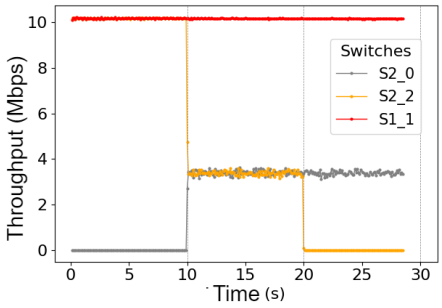


Figure 13. Test 2: Result

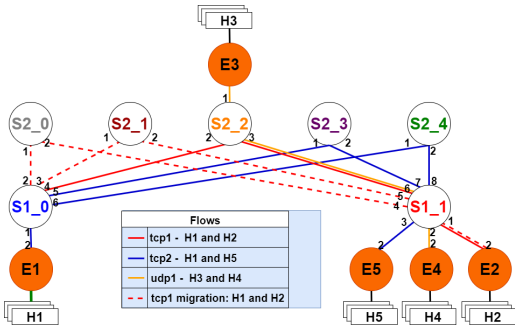


Figure 14. Test 3: Scenario

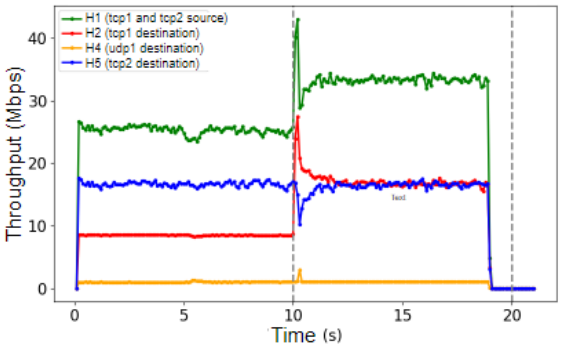


Figure 15. Test 3: Result.

Test with Profile 1011 (Figure 12): Using the indicated transmission tree (Figure 12a) and a weight ratio of 2: 1:2:1, *switch* `S1_0` distributed traffic variously across its four active ports. The results, represented in Figure 12b, demonstrated the distribution of UDP packets aligned with profile 11 (2:1:2:1) in *switches* `S1 _0`, `S2_0`, `S2_2`, `S2_4` and `S1_1`.

4.2. Test 2: Agile Reconfiguration

The experiment consisted of agile reconfiguration between traffic division profiles 0, 100, and 1011, already presented in Figures 10, 11 and 12. The division of traffic was determined by the origin and, again, *switch* $E1$ is responsible for inserting the labels *routeID* and *weightID* into the packet headers. Traffic on the active ports of switches $S2\_0$, $S2\_2$, and $S1\_1$ was captured. It is important to note that during the experiment, the values of *routeID* and *weightID* varied depending on the selected profile. However, due to space limitations, and to simplify the description of the experiments, the values were omitted. The test carried out consists of three phases: use of Profile 0 for 10 *s*, then migration to Profile 100 for another 10 *s*, and new migration to Profile 1011 for another 10 *s*.

The flow obtained on the active ports of the *switches* during the experiment is represented in Figure 13, which shows a stable and constant flow during the transitions between profiles, without fluctuations or interruptions. Thus, based on the results in Figure 13, it can be stated that with *MTS-PolKA*, the traffic division can be quickly reconfigured simply by defining new labels on the edge, without the need to update the core node tables. In table-based approaches, there is a large latency for configuration and convergence, as the controller needs to apply changes to all affected nodes along the path [Jyothi et al. 2015].

4.3. Test 3: Multiple flows and competition

In this experiment, we demonstrate how multiple flows operate simultaneously and how agile traffic profile migration can optimize resource utilization in competitive environments. The following were sent: a TCP flow between *H1* and *H2* (*tcp1*, in red), a TCP flow between *H1* and *H5* (*tcp2*, in blue) and a UDP flow with a throughput of 4Mbps between *H3* and *H4* (*udp1*, in yellow), as shown in Figure 14. The traffic division was determined by the edge *switches* $E1$ and $E3$, which are responsible for inserting the *routeID* and *weightID* labels into the packet headers. Two traffic division profiles are used: profile 0 (sends all traffic to the first active port) and profile 1 (1:1 weights, sends half of the traffic to the first active port and half of the traffic to the second active port).

In the period from **0 s to 10 s**, the flow *tcp1* followed a single path ($S1\_0 - S2\_2 - S1\_1$) with traffic profile 0, the flow *tcp2* used traffic profile 1, dividing traffic equally between two paths ($S1\_0 - S2\_3 - S1\_1$ and $S1\_0 - S2\_4 - S1\_1$), and the flow *udp1* followed a single path ($S2\_2 - S1\_1$) with traffic profile 0. As the flow *tcp1* starts the experiment competing with the flow *udp1* on link $S2\_2 - S1\_1$ and there are paths with idle links, the experiment performed a migration of the path and traffic profile to the flow *tcp1*. In the period from **10 s to 20 s**, the flow *tcp1* was migrated, just by changing the flow entry that inserts packet labels referring to the flow *tcp1* in $E1$, and started to use traffic profile 1 with the new transmission tree, dividing traffic equally between the two paths that were idle ($S1\_0 - S2\_0 - S1\_1$ and $S1\_0 - S2\_1 - S1_1$).

The flow rates obtained during the experiment are illustrated in Figure 15. The data highlights a significant increase in the flow rate of *tcp1* (red line) following the migration, which eliminated competition with *udp1* and enabled the exploration of two paths instead of just one. Additionally, there is a noticeable increase in aggregate throughput on host H1 (green line), which generates traffic. This improvement results from the enhanced total bandwidth made possible by exploiting multipath routing. Thus, it can be concluded that *MTS-PolKA* enables easy reconfiguration of traffic division by defining new labels at the edge, facilitating the operation of multiple flows. Furthermore, this feature allows for agile traffic profile migration, optimizing the aggregation of multiple TCP flows to effectively utilize multiple paths in competitive environments.

4.4. Test 4: Comparison with Related Works on Flow Completion Time for Elephant Flows and Latency Analysis for Mice Flows

In this experiment, we implemented ECMP, WCMP, and RPS using the P4 language within the Mininet emulator to conduct a comparative study. For elephant flows, we focused on the Flow Completion Time (FCT) metric, which measures the duration required for a data flow to traverse the network from source to destination until fully received. This metric is crucial for evaluating networking performance in modern datacenters. For mice flows, we analyzed latency during transmission, as these flows are characterized by their short duration and sensitivity to delays.

Figure 16 shows the test scenario. A 1 ms delay was introduced across the network. Core network links were set to 1 Mbps, while edge links were configured to 10 Mbps. The following TCP flows were sent using the *iperf* tool: a 100MB elephant flow between H2 and H6 (in green), a 100MB elephant flow between H3 and H5 (in blue), and several 10KB mice flows (4 flows per second following a Poisson distribution) between H1 and H4 (in

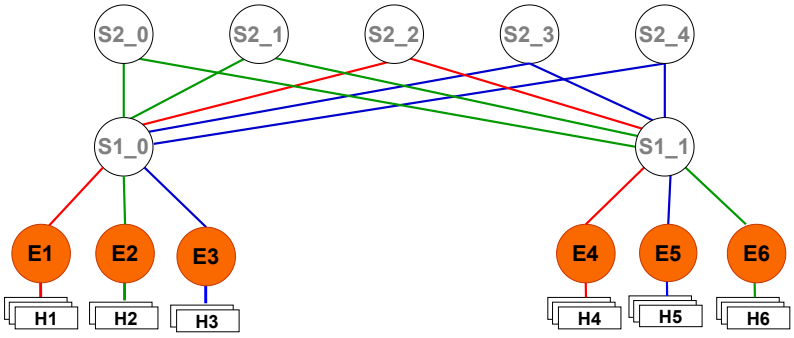
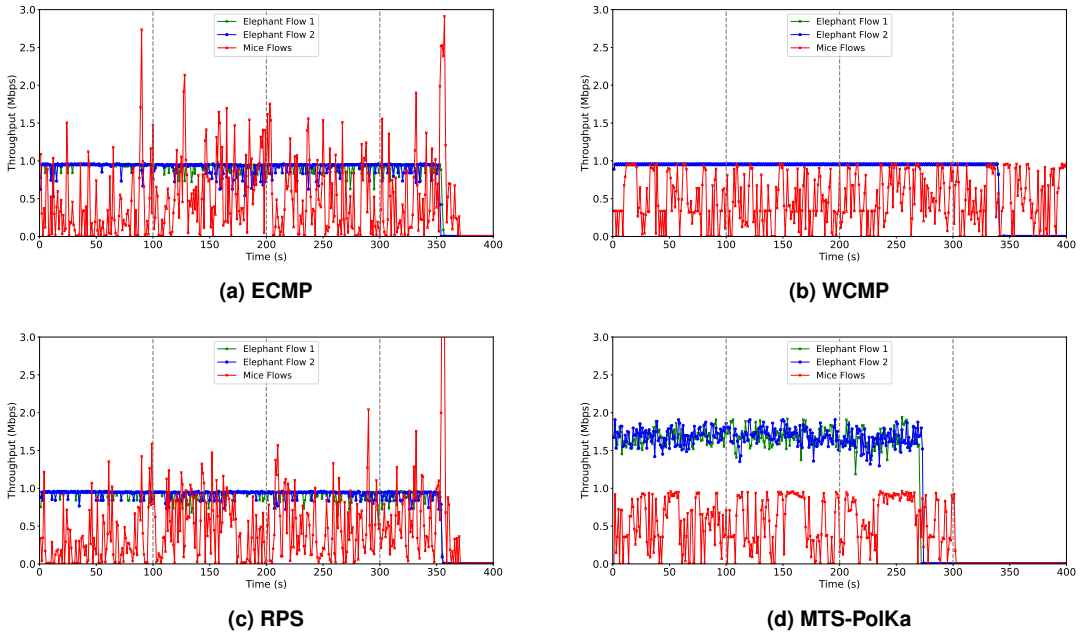


Figure 16. Test 4: Scenario (*MTS-PolKA* traffic splitting configuration).



(a) ECMP **(b) WCMP**

(c) RPS **(d) MTS-PolKa**

Figure 17. Test 4: Flow Completion Time comparison with related works.

red). Traffic was captured at the destination hosts H4, H5, and H6, which are connected to *switch* `S1_1`. In addition, latency data was collected using the *ping* tool between H1 and H4.

Figure 16 specifically illustrates the traffic splitting configuration selected by *MTS-PolKA*. Two traffic splitting profiles were used: elephant flow 1 (in green) and elephant flow 2 (in blue) followed profile 1, which splits the traffic equally between the two active ports, while the mice flows (in red) followed profile 0, which sends all traffic to one active port. For each flow, Edge switches E1, E2, and E3 are responsible for inserting the *routeID* and *weightID* labels into the packet headers, according to the selected active ports and traffic splitting profiles.

Additionally, ECMP, WCMP, and RPS have been implemented to distribute both elephant and mice flows across multiple paths. With ECMP, each flow is assigned to a single path of equal cost, leading to a uniform traffic distribution, without considering the specific characteristics of the flows. RPS randomly distributes packets among available links without considering flow sizes or link capacities. With WCMP, each flow is assigned to a single path, but the traffic distribution among these paths can be adjusted using

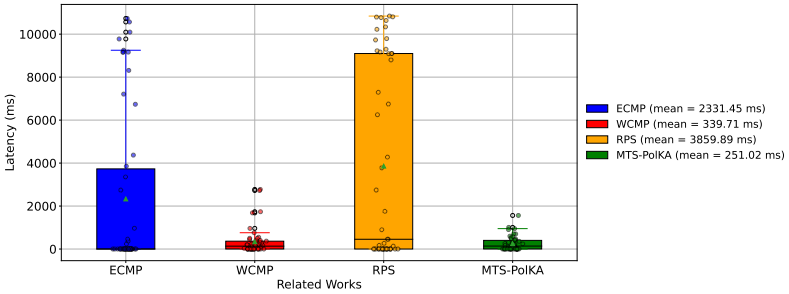


Figure 18. Test 4: Mice flow latency comparison with related works.

weighted group tables at each switch. Consequently, the optimal configuration with WCMP involved reserving one link for each elephant flow, while allocating the remaining links for mice flows.

The FCT comparison in Figure 17 demonstrated that MTS-PolKA outperformed the other schemes. MTS-PolKA was able to significantly reduce the FCT (approximately 270s) compared to ECMP (approximately 355s), WCMP (approximately 340s), and RPS (approximately 360s), because it allows elephant flows to explore more than one link. While improving bandwidth utilization, the latency comparison in Figure 18 shows that *MTS-PolKA* simultaneously optimizes latency for mice flows more effectively (average latency of only 251.02 ms) than ECMP (average latency of 2331.45 ms) and RPS (average latency of 3859.89 ms). This happens because both *MTS-PolKA* and WCMP can separate latency-sensitive mice flows from elephant flows, resulting in improved latency performance. However, while WCMP also achieves this separation and minimizes latency (average latency of 339.71 ms), it results in longer FCT due to its restriction on splitting elephant flows across multiple links. RPS proved to be the least efficient method among those tested due to its impact on TCP reordering and its random distribution of all flows across all links when spraying packets.

MTS-PolKA takes a hybrid approach that combines flow and packet granularity, along with source-based routing, allowing it to dynamically optimize bandwidth usage and minimize network congestion. This smart traffic distribution eases congestion on specific routes, letting large flows (elephant flows) use less overloaded paths, which reduces the completion time for transfers. At the same time, this strategy ensures that smaller flows (mice flows) are transmitted quickly without directly competing with larger flows for the network's limited resources. By leveraging multiple links, *MTS-PolKA* helps achieve a more balanced bandwidth usage, improving network performance for all types of traffic. Moreover, this test scenario can be extrapolated to more complex traffic loads, where per-flow policies selected by the source can be individually selected to optimize overall performance.

4.5. Prototype limitations

The solution presented in this work has proven to be viable and functional compared to traditional methods. However, there are some limitations that need to be highlighted, which will be the focus of research and development in future works:

- The software switch `bmv2 simple_switch` with the `v1model` architecture was selected as the target for this prototype, because it supports all the functional-

ities required by *MTS-PolKA*, such as the configuration of CRC polynomials. It is important to highlight that this software switch is a user space implementation with focus on feature testing. There are other high performance implementations of P4 software switches and compilers (e.g., PISCES[1], P4ELTE[2], and MACSAD[3]), but they do not yet cover all the features required by *MTS-PolKA*. As these implementations evolve, it will be possible to test our prototype with higher loads. For the time being, the solution was to limit the number of flows and the link rates to 10Mbps in our emulated prototype to avoid reaching the processing capacity limits of `bmv2 simple_switch`.

- Due to the described limitations of the software switch, this paper was unable to perform a deeper performance and scalability comparison with related works. We plan to address this in future research by implementing our proposal and related solutions on physical devices, such as the Tofino switch, and on the ns-3 simulator. The latter will be particularly important for gaining more insights into TCP reordering issues.
- This paper focused on designing the traffic splitting mechanism within the data plane. Other important functionalities, such as monitoring, flow scheduling, and path allocation, were treated as predefined control plane inputs in the experiments, and were not within the scope of this work. These aspects will be addressed in future research.
- Specifically, we plan to further investigate queue management within flow scheduling algorithms to mitigate packet reordering caused by spraying packets across different paths. For example, we aim to select paths for packets based on the queuing delay of the output buffer, ensuring that packets arriving earlier are forwarded before later packets. This approach would help avoid packet reordering, similar to the method used by QDAPS [Huang et al. 2018].

5. Conclusions and Future Work

In this paper, we introduced MTS-PolKA, **a weighted multipath traffic splitting solution that uniquely combines the benefits of per-flow and per-packet granularity with source routing**. A key point of our approach is that MTS-PolKA is the only solution to integrate these features, providing unmatched flexibility and control over traffic distribution. This combination allows for more precise adjustments to traffic flows, which, in turn, has the potential to significantly optimize bandwidth traffic engineering in datacenter networks.

Unlike traditional methods like ECMP and WCMP, which struggle with dynamic traffic conditions, MTS-PolKA enables agile traffic distribution across all switches in the path through simple modifications to the packet header at the source, without requiring changes in the network core. Its advantages become particularly evident during network overload scenarios, where real-time adaptability enhances traffic engineering efficiency. MTS-PolKA not only improves Flow Completion Time (FCT) but also strengthens network robustness, making it a strong candidate for data center environments that demand high efficiency and speed in flow processing. Our experiments using the Mininet emulator and P4-programmable switches show that MTS-PolKA maintains stable flows during

[1] `http://pisces.cs.princeton.edu/`
[2] `http://p4.elte.hu/`
[3] `https://github.com/intrig-unicamp/macsad/`

profile transitions and boosts aggregate throughput by fully exploiting available bandwidth through multipath routing.

Future work will focus on enhancing the performance evaluation and scalability of *MTS-PolKA* across diverse scenarios, including packet reordering analysis and assessments of memory and bandwidth overhead. Additionally, we plan to conduct experiments with Tofino programmable switches to validate the effectiveness of the approach in real-world environments.

Acknowledgements

This work has partial funding from: Ifes, Fapes (#941/ 2022, #2023/ RWXSZ, #2022/ ZQX6), Fapesp/ MCTI/ CGI.br (#2020/ 05182-3) and CAPES (process 2021-2S6CD, nº FAPES 132/2021) through the PDPG (Programa de Desenvolvimento da Pós-Graduação, Parcerias Estratégicas nos Estados).

References

Benson, T., Anand, A., Akella, A., and Zhang, M. (2010). Understanding data center traffic characteristics. *SIGCOMM Comput. Commun. Rev.*, 40(1):92–99.

Cui, Z., Hu, Y., and Hou, S. (2021). Adaptive weighted cost multipath routing on pisa. In *2021 IEEE International Conference on Artificial Intelligence and Industrial Design (AIID)*, pages 541–544.

Dixit, A., Prakash, et al. (2013). On the impact of packet spraying in data center networks. In *2013 Proceedings IEEE INFOCOM*, pages 2130–2138.

Dominicini, C. et al. (2020). Polka: Polynomial key-based architecture for source routing in network fabrics. In *2020 6th IEEE Conference on Network Softwarization (NetSoft)*, pages 326–334.

Gilliard, E., Liu, J., Aliyu, A. A., Juan, D., Jing, H., and Wang, M. (2024). Intelligent load balancing in data center software-defined networks. *Transactions on Emerging Telecommunications Technologies*, 35(4):e4967.

Guimarães, R. S. et al. (2022). M-polka: Multipath polynomial key-based source routing for reliable communications. *IEEE Transactions on Network and Service Management*, 19(3):2639–2651.

Hsu, K.-F., Tammana, et al. (2020). Adaptive weighted traffic splitting in programmable data planes. In *Proceedings of the Symposium on SDN Research*, SOSR '20, page 103–109, New York, NY, USA. Association for Computing Machinery.

Huang, J., Lv, W., Li, W., Wang, J., and He, T. (2018). Qdaps: Queueing delay aware packet spraying for load balancing in data center. In *2018 IEEE 26th International Conference on Network Protocols (ICNP)*, pages 66–76.

Huang, J., Lyu, et al. (2021). Mitigating packet reordering for random packet spraying in data center networks. *IEEE/ACM Transactions on Networking*, 29(3):1183–1196.

Jyothi, S. A., Dong, M., and Godfrey, P. B. (2015). Towards a flexible data center fabric with source routing. In *Proceedings of the 1st ACM SIGCOMM Symposium on Software Defined Networking Research*, pages 1–8.

Li, W., Liu, J., Wang, S., Zhang, T., Zou, S., Hu, J., Jiang, W., and Huang, J. (2021). Survey on traffic management in data center network: from link layer to application layer. *IEEE Access*, 9:38427–38456.

Pang, J., Xu, G., and Fu, X. (2017). Sdn-based data center networking with collaboration of multipath tcp and segment routing. *IEEE Access*, 5:9764–9773.

Robin, D. D. and Khan, J. I. (2022). Clb: Coarse-grained precision traffic-aware weighted cost multipath load balancing on pisa. *IEEE Transactions on Network and Service Management*, 19(2):784–803.

Rottenstreich, O., Kanizo, et al. (2018). Accurate traffic splitting on commodity switches. In *Proceedings of the 30th on Symposium on Parallelism in Algorithms and Architectures*, SPAA '18, page 311–320, New York, NY, USA. Association for Computing Machinery.

Valentim, R. et al. (2019). Rdna balance: Balanceamento de carga por isolamento de fluxos elefante em data centers com roteamento na origem. In *Anais do XXXVII Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos*, pages 1000–1013, Porto Alegre, RS, Brasil. SBC.

Wassie Geremew, G. and Ding, J. (2023). Elephant flows detection using deep neural network, convolutional neural network, long short-term memory, and autoencoder. *Journal of Computer Networks and Communications*, 2023(1):1495642.

Xie, S., Hu, G., Xing, C., and Liu, Y. (2024). Online elephant flow prediction for load balancing in programmable switch-based dcn. *IEEE Transactions on Network and Service Management*, 21(1):745–758.

Zhou, J., Tewari, et al. (2014). Wcmp: Weighted cost multipathing for improved fairness in data centers. In *Proceedings of the Ninth European Conference on Computer Systems*, EuroSys '14, New York, NY, USA. Association for Computing Machinery.