P4BFT: Hardware-Accelerated Byzantine-Resilient Network Control Plane

Ermin Sakic*[†], Nemanja Deric*, Endri Goshi*, Wolfgang Kellerer*

*Technical University Munich, Germany, [†] Siemens AG, Germany

E-Mail:*{ermin.sakic, nemanja.deric, endri.goshi, wolfgang.kellerer}@tum.de, [†] ermin.sakic@siemens.com

Abstract—Byzantine Fault Tolerance (BFT) enables correct operation of distributed, i.e., replicated applications in the face of malicious take-over and faulty/buggy individual instances. Recently, BFT designs have gained traction in the context of Software Defined Networking (SDN). In SDN, controller replicas are distributed and their state replicated for high availability purposes. Malicious controller replicas, however, may destabilize the control plane and manipulate the data plane, thus motivating the BFT requirement. Nonetheless, deploying BFT in practice comes at a disadvantage of increased traffic load stemming from replicated controllers, as well as a requirement for proprietary switch functionalities, thus putting strain on switches' control plane where particular BFT actions must be executed in software.

P4BFT leverages an optimal strategy to decrease the total amount of messages transmitted to switches that are the configuration targets of SDN controllers. It does so by means of message comparison and deduction of correct messages in the determined optimal locations in the data plane. In terms of the incurred control plane load, our P4-based data plane extensions outperform the existing solutions by $\sim 33.2\%$ and $\sim 40.2\%$ on average, in random 128-switch and Fat-Tree/Internet2 topologies, respectively. To validate the correctness and performance gains of P4BFT, we deploy *bmv2* and *Netronome Agilio SmartNIC*-based topologies. The advantages of P4BFT can thus be reproduced both with software switches and "commodity" P4-enabled hardware. A hardware-accelerated controller packet comparison procedure results in an average 96.4 % decrease in processing delay per request compared to existing software approaches.

I. INTRODUCTION

State-of-the-art failure-tolerant SDN controllers base their state distribution on crash-tolerant consensus approaches. Such approaches comprise single-leader operation, where leader replica decides on the ordering of client updates. After confirming the update with the follower majority, the leader triggers the cluster-wide commit operation and acknowledges the update with the requesting client. RAFT algorithm [1] realizes this approach, and is implemented in OpenDaylight [2] and ONOS [3]. RAFT is, however, unable to distinguish malicious / incorrect from correct controller decisions, and can easily be manipulated by an adversary in possession of the leader replica [4]. Recently, Byzantine Fault Tolerance (BFT)-enabled controllers were proposed for the purpose of enabling correct consensus in scenarios where a *subset of controllers is faulty due to a malicious adversary or internal bugs* [5]–[7]. In BFT-enabled SDN, multiple controllers act as replicated state machines and hence process incoming client requests individually. Thus with BFT, each controller of a single administrative domain transmits an output of their computation to the target switch. The outputs of controllers are then collected by trusted configuration targets (e.g., switches)

and compared for payload matching for the purpose of correct message identification.

In *in-band* [8] deployments, where application flows share the same infrastructure as the control flows, the traffic arriving from controller replicas imposes a non-negligible overhead [9]. Similarly, comparing and processing controller messages in the switches' software-based control plane causes additional delays and CPU load [7], leading to longer reconfigurations. Moreover, the comparison of control packets is implemented as a proprietary non-standardized switch function, thus unsupported in off-the-shelf devices.

In this work, we investigate the benefits of offloading the procedure of comparison of controller outputs, required for correct BFT operation, to carefully selected network switches. By minimizing the distance between the processing nodes and controller clusters / individual controller instances, we decrease the network load imposed by BFT operation. P4BFT's P4-enabled pipeline is in charge of controller packet collection, correct packet identification and its forwarding to the destination nodes, thus minimizing accesses to the switches' software control plane and effectively outperforming the existing software-based solutions.

II. BACKGROUND AND PROBLEM STATEMENT

BFT has recently been investigated in the context of distributed SDN control plane [5]–[7], [10]. In [5], [6], $3F_M + 1$ controller replicas are required to tolerate up to F_M Byzantine failures. MORPH [7] requires $2F_M + F_A + 1$ replicas in order to tolerate up to F_M Byzantine and F_A availability-induced failures. The presented models assume the deployment of SDN controllers as a set of replicated state machines, where clients submit inputs to the controllers, that process them in isolation and subsequently send the computed outputs to the target destination (i.e., reconfiguration messages to destination switches). They assume trusted platform execution and a mechanism in the destination switch, capable of comparison of the controller messages and deduction of the correct message. Namely, after receiving $F_M + 1$ matching payloads, the observed message is regarded as correct and the containing configuration is applied.

The presented models are sub-optimal in a few regards. First, they assume the collection and processing of controller messages exclusively in the receiver nodes (configuration targets). Propagation of each controller message can carry a large system footprint in large-scale in-band controlled networks, thus imposing a non-negligible load on the data plane. Second, neither of the models detail the overhead of

message comparison procedure in the target switches. The realizations presented in [5]–[7], [10] realize the packet comparison procedure solely in software. The non-deterministic / varied latency imposed by the software switching may, however, be limiting in specific use cases, such as in the failure scenarios in critical infrastructure networks [11] or in 5G scenarios [12]. This motivates a hardware-accelerated BFT design that minimizes the processing delays.

A. Our contribution

We introduce and experimentally validate the P4BFT design, which builds upon [5]–[7] and adds further optimizations:

- It allows for collection of controllers' packets and their comparison in *processing* nodes, as well as for relaying of deduced correct packets to the destinations;
- It selects the optimal processing nodes at per-destination-switch granularity. The proposed objective minimizes the control plane load and reconfiguration time, while considering constraints related to the switches' processing capacity and the upper-bound reconfiguration delay;
- It executes in software, e.g., in P4 switch behavioral model (*bmv2*[1]), or in a physical, e.g., Netronome Smart-NIC[2] environment. Correctness, processing time and deployment flexibility are validated in both platforms.

We present the evaluation results of P4BFT for well-known and randomized network topologies and varied controller and cluster sizes and their placements. To the best of our knowledge, this is the first implementation of a BFT-enabled solution on a hardware platform, allowing for accelerated packet processing and low-latency malicious controller detection time.

Paper Structure: Related work is presented in Section III. Section IV details the P4BFT co-design of the control and data plane as well as the optimization procedure. Section V presents the evaluation methodology and discusses the empirically measured performance of P4BFT in software- and hardware-based data planes. Section VI concludes this paper.

III. RELATED WORK

1) BFT variations in SDN context: In the context of centralized network control, BFT is still a relatively novel area of research. Reference solutions [5]–[7], assume the comparison of configuration messages, transmitted by the controller replicas, in the switch destined as the configuration target. With P4BFT, we investigate the flexibility advantages of message processing in any node capable of message collection and processing, thus allowing for a footprint minimization. [6] and [7] discuss the strategy for minimization of no. of matching messages required to deduce correct controller decisions, which we adopt in this work as well. [10] discusses the benefit of disaggregation of BFT consensus groups in the SDN control plane into multiple controller cluster partitions, thus enabling higher scalability than possible with [6] and [7]. While compatible with [10], our work focuses on scalability enhancements and footprint minimization by means of data-plane reconfiguration for realizing more efficient packet comparison.

2) Data Plane-accelerated Service Execution: Recently, Dang et al. [13] have portrayed the benefits of offloading coordination services for reaching consensus to the data plane, on the example of a Paxos implementation in P4 language. In this paper, we investigate if a similar claim can be transferred to BFT algorithms in SDN context. In the same spirit, in [14], end-hosts partially offload the log replication and log commitment operations of RAFT consensus algorithm to neighboring P4 devices, thus accelerating the overall commit time. In the context of in-network computation, Sapio et al. [15] discuss the benefit of data aggregation offloading to constrained network devices for the purpose of data reduction and minimization of workers' computation time.

IV. SYSTEM MODEL AND DESIGN

A. P4BFT System Model

We consider a typical SDN architecture allowing for flexible function execution on the networking switches for the purpose of BFT system operation. The flexibility of in-network function execution is bounded by the limitation of the data plane programming interface (i.e., the $P4_{16}$ [16] language specification in the case of P4BFT). The control plane communication between the switches and controllers and in-between the controllers is realized using an in-band control channel [8]. In order to prevent faulty replicas from impersonating correct replicas, controllers authenticate each message using Message Authentication Codes (assuming pre-shared symmetric keys for each pair) [17]. Similarly, switches that are in charge of message comparison and message propagation to the configuration targets must be capable of signature generation using the processed payload and their secret key.

In P4BFT, controllers calculate their decisions in isolation from each other, and transmit them to the destination switch. Control packets are intercepted by the *processing* nodes (i.e., processing switches) responsible for decisions destined for the target switch. In order to collect and compare control packets, we assume packet header fields that include the `client_request_id`, `controller_id`, `destination_switch_id` (e.g., MAC/IP address), the `payload` (controller-decided configuration) and the optional `signature` field (denoting if a packet has already been processed by a processing node). Clients must include the `client_request_id` field in their controller requests.

Apart from distinguishing correct from malicious/incorrect messages, P4BFT allows for identification and exclusion of *faulty* controller replicas. P4BFT's architectural model assumes three entities, each with a distinguished role:

1) Network controllers enforce forwarding plane configurations based on internal decision making. For simplification, each controller replica of an administrative domain serves each client request. Each *correct* replica maintains internal state information (e.g., resource reservations) matching to that of other *correct* instances. In the case of a controller with diverged state, i.e., as a result of corrupted operation or a malicious adversary take-over, the *incorrect* controllers' computation outputs may differentiate from the correct ones.

---

[1]P4 Software Switch - https://github.com/p4lang/behavioral-model
[2]Netronome Agilio®CX 2x10GbE SmartNIC Product Brief - https://www.netronome.com/media/documents/PB_Agilio_CX_2x10GbE.pdf

2) P4-enabled switches forward the control and application packets. Depending on the output of Reassigner's optimization step, a switch may be assigned the *processing* node role, i.e., become in charge of comparing outputs computed by different controllers, destined for *itself* or *other* configuration targets. A processing node compares messages sent out by different controllers and distinguishes the correct ones. On identification of a faulty controller, it declares the faulty replica to the Reassigner. In contrast to [5]–[7], P4BFT enables control packet comparison for packets destined for remote targets.

3) Reassigner is responsible for two tasks:

Task 1: It dynamically reassigns the controller-switch connections based on the events collected from the detection mechanism of the switches, i.e., upon their detection, it excludes faulty controllers from the assignment procedure. It furthermore ensures that a minimum number of required controllers, necessary to tolerate a number of availability failures F_A and malicious failures F_M, are loaded and associated with each switch. This task is also discussed in [6], [7].

Task 2: It maps a *processing* node, in charge of controller messages' comparison, to each destination switch. Based on the result of this optimization, switches gain the responsibility of control packets processing. The output of the optimization procedure is the *Processing Table*, necessary to identify the switches responsible for comparison of controller messages. Additionally, the Reassigner computes the *Forwarding Tables*, necessary for forwarding of controller messages to processing nodes and reconfiguration targets. Given the no. of controllers and the user-configurable parameter of max. tolerated Byzantine failures F_M, Reassigner reports to processing nodes the no. of *necessary matching messages* that must be collected prior to marking a controller message as correct.

B. Finding the Optimal Processing Nodes

The optimization methodology allows for minimization of the experienced switch reconfiguration delay, as well as the decrease of the total network load introduced by the exchanged controller packets. When a switch is assigned the *processing* node role for itself or another target switch, it collects the control packets destined for the target switch and deduces the correct payload on-the-fly, it next forwards a single packet copy containing the correct controller message to the destination switch. Consider Fig. 1a). If control packet comparison is done only at the target switch (as in prior works), a request for S4 creates a total footprint of $F_C = 13$ packets in the data plane (the sum of Cluster 1 and Cluster 2 utilizations of 4 and 9, respectively). In contrast, if the processing is executed in S3 (as depicted in Fig. 1b)), the total experienced footprint can be decreased to $F_C = 11$. Therefore, in order to minimize the total control plane footprint, we identify an optimal processing node for each target switch, based on a given topology, placement of controllers and the processing nodes' capacity constraints. If we additionally extend the optimization to a multi-objective formulation by considering the delay metric, the total traversed critical path between the controller furthest away from the configuration target would equal $F_D = 3$ in the worst case (ref. Fig. 1c)), i.e., 3 hops assuming a delay

weight of 1 per hop. Additionally, this assignment also has the minimized communication overhead of $F_C = 11$.

TABLE I
PARAMETERS USED IN THE MODEL

Symbol	Description
$\mathcal{V} : \{S_1, S_2, ..., S_n\}, n \in \mathbb{Z}^+$	Set of all switch nodes in the topology.
$\mathcal{C} : \{C_1, C_2, ..., C_n\}, n \in \mathbb{Z}^+$	Set of all controllers connected to the topology.
$\mathcal{D} : \{d_{i,j,k}, \forall i, j, k \in \mathcal{V}\}$	Set of delay values for path from i to k, passing through j.
$\mathcal{H} : \{h_{i,j}, \forall i, j \in \mathcal{V}\}$	Set of number of hops for shortest path from i to j.
$\mathcal{Q} : \{q_i, \forall i \in \mathcal{V}\}$	Set of switches' processing capacity.
$\mathcal{C}^j \subseteq \mathcal{C}$	Set of controllers connected to the node j.
$\mathcal{M} \subseteq \mathcal{V}$	Set of switches connected to at least one controller.
T	Maximum tolerated delay value.
$x(i,k)$	Binary variable that equals 1 if i is a processing node for k.

We describe the processing node mapping problem using an integer linear programming (ILP) formulation. Table I summarizes the notation used.

Communication overhead minimization objective minimizes the global imposed communication footprint in the control plane. Each controller replica generates an individual message sent to the processing node i, that subsequently collects all remaining necessary messages and forwards a resulting single correct message to the configuration target k:

$$M_F = \min \sum_{k \in \mathcal{V}} \sum_{i \in \mathcal{V}} (1 \cdot h_{i,k} \cdot x(i,k) + \sum_{j \in \mathcal{M}} |\mathcal{C}^j| \cdot h_{j,i} \cdot x(i,k)) \quad (1)$$

Configuration delay minimization objective minimizes the *worst-case* delay imposed on the critical path used for forwarding configuration messages from a controller associated with node j, to the potential processing node i and finally to the configuration target node k:

$$M_D = \min \sum_{k \in \mathcal{V}} \sum_{i \in \mathcal{V}} x(i,k) \cdot \max_{\forall j \in \mathcal{M}} (d_{j,i,k}) \quad (2)$$

Bi-objective optimization minimizes the weighted sum of the two objectives, w_1 and w_2 being the associated weights:

$$\min w_1 \cdot M_F + w_2 \cdot M_D \quad (3)$$

Processing capacity constraint: Sum of messages requiring processing on i, for each configuration target k assigned to i, must be kept at or below i's processing capacity q_i:

$$\text{Subject to: } \sum_{k \in \mathcal{V}} x(i,k) \cdot |\mathcal{C}| \leqslant q_i, \quad \forall i \in \mathcal{V} \quad (4)$$

Maximum delay constraint: For each configuration target k, the delay imposed by the controller packet forwarding to node i, responsible for collection and packet comparison procedure and forwarding of the correct message to the target node k, does not exceed an upper bound T:

$$\text{Subject to: } \sum_{i \in \mathcal{V}} x(i,k) \cdot \max_{\forall j \in \mathcal{M}} (d_{j,i,k}) \leqslant T, \quad \forall k \in \mathcal{V} \quad (5)$$

Single assignment constraint: For each configuration target k, there exists *exactly* one processing node i:

$$\text{Subject to: } \sum_{i \in \mathcal{V}} x(i,k) = 1, \quad \forall k \in \mathcal{V} \quad (6)$$

Note: The assignment of controller-switch connections for the purpose of control and reconfiguration is adapted from existing formulations [7], [10] and is thus not detailed here.
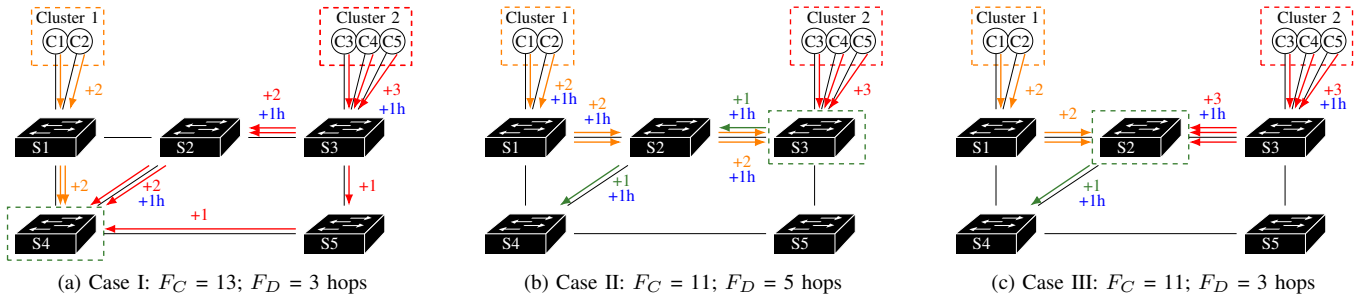
Fig. 1. For brevity we depict the control flows destined only for configuration target S4. The orange and red blocks represent an exemplary cluster separation of 5 controllers into groups of 2 and 3 controllers, respectively. The green dashed block highlights the processing node responsible for comparing the controller messages destined for S4. Figure (a) presents the unoptimized case as per [5]–[7], where S4 collects and processes control messages destined for itself, thus resulting in a control plane load of $F_C = 13$ and a delay on critical path (marked with blue labels) of $F_D = 3$ hops (assuming edge weights of 1). By optimizing for the total communication overhead, the total F_C can be decreased to 11, as portrayed in Figure (b). Contrary to (a), in (b) processing of packets destined for S4 is offloaded to the processing node S3. However, additional delay is incurred by the traversal of path S1-S2-S3-S2-S4 for the control messages sourced in Cluster 1. Multi-objective optimization according to P4BFT, that aims to minimize both the communication overhead and control plane delay instead selects S2 as the optimal processing node (ref. Figure (c)), thus minimizing both F_C and F_D.

C. P4 Switch and Reassigner Control Flow

Processing node data plane: Switches declared to process controller messages for a particular target (i.e., for itself, or for another switch) initially collect the control payloads stemming from different controllers. Each processing node maintains counters for the number of observed and matching packets for a particular (re-)configuration request identifier. After sufficient matching packets are collected for a particular payload (more specifically, hash of the payload), the processing node *signs* a message using its private key and forwards one copy of the correct packet to its own control plane for required software processing (i.e., identification of the correct message and potentially malicious controllers), and the second copy on the port leading to the configuration target. To distinguish processed from unprocessed packets in destination switches, processing nodes refer to the trailing signature field.

Processing node control plane: After determining the correct packet, the processing node identifies any incorrect controller replicas (i.e., replicas whose output hashes diverge from the deduced correct hash) and subsequently notifies the Reassigner of the discrepancy. Alternatively, the switch applies the configuration message if it is the configuration target itself. The switch then proceeds to clear its registers associated with the processed message hash so to free the memory for future requests.

Reassigner control flow: At network bootstrapping time, or on occurrence of any of the following events: i) a detected malicious controller; ii) a failed controller replica; or iii) a switch/link failure; Reassigner reconfigures the processing and forwarding tables of the switches, as well as the number of required matching messages to detect the correct message.

D. P4 Tables Design

Switches maintain *Tables* and *Registers* that define the method of processing incoming packets. Reassigner populates the switches' Tables and Registers so that the selection of processing nodes for controller messages is optimal w.r.t. a set of given constraints, i.e., so that the total message overhead or control plane latency experienced in control plane is minimized (according to the optimization procedure in Section IV-B). The Reassigner thus modifies the elements whenever a controller is identified as incorrect and is hence excluded from consideration, resulting in a different optimization result. P4BFT leverages four P4 tables:

1) *Processing Table*: It holds identifiers of the switches whose packets must be processed by the switch hosting this table. Incoming packets are matched based on the destination switch's ID. In the case of a table hit, the hosting switch processes the packets as a processing node. Alternatively, the packet is matched against the *Process-Forwarding Table*.

2) *Process-Forwarding Table*: Declares which egress port the packets should be sent out on for further processing. If an unprocessed control packet is not to be processed locally, the switch will forward the packet towards the correct processing node, based on forwarding entries maintained in this table.

3) *L2-Forwarding Table*: After the processing node has processed the incoming control packets destined for the destination switch, the last step is forwarding the correctly deduced packet towards it. Information on how to reach the destination switches is maintained in this table. Contrary to forwarding to a processing node, the difference here is that the packet is now forwarded to the destination switch.

4) *Hash Table with associated registers*: *Processing* a set of controller packets for a particular request identifier requires evaluating and counting the number of occurrences of packets containing the matching payload. To uniquely identify the decision of the controller, a hash value is generated on the payload during processing. The counting of incoming packets is done by updating the corresponding binary values in the register vectors, with respective layout depicted in Table II.

On each arriving unprocessed packet, the processing node computes a previously seen or i-th initially observed hash $h_i^{request\_id}$ over the acquired payload. Subsequently, it sets the

Msg Hash	Request ID 1				...		Request ID K			
h_0	$b_{C_1}^{h_0}$	$b_{C_2}^{h_0}$	...	$b_{C_N}^{h_0}$		...	$b_{C_1}^{h_0}$	$b_{C_2}^{h_0}$	...	$b_{C_N}^{h_0}$
...	$b_{C_1}^{\cdots}$	$b_{C_2}^{\cdots}$	...	$b_{C_N}^{\cdots}$		...	$b_{C_1}^{\cdots}$	$b_{C_2}^{\cdots}$	...	$b_{C_N}^{\cdots}$
h_{F_M}	$b_{C_1}^{h_{F_M}}$	$b_{C_2}^{h_{F_M}}$	...	$b_{C_N}^{h_{F_M}}$		...	$b_{C_1}^{h_{F_M}}$	$b_{C_2}^{h_{F_M}}$	...	$b_{C_N}^{h_{F_M}}$

binary flag to 1, for source controller `controller_id` in the i-th register row at column [`client_request_id`·$|\mathcal{C}|$ + `controller_id`]. $|\mathcal{C}|$ represents the total no. of deployed controllers. Each time a client request is fully processed, the binary entries associated with the corresponding `client_request_id` are reset to zero. To detect a malicious controller, the controller IDs associated with hashes distinguished as incorrect, are reported to the Reassigner.

Note: To tolerate F_M Byzantine failures, a maximum of $F_M + 1$ unique hashes for a single request identifier may be detected, hence the corresponding $F_M + 1$ pre-allocated table rows in Table II.

V. Evaluation, Results and Discussion

A. Evaluation Methodology

We next evaluate the following metrics using P4BFT and state-of-the-art [5]–[7] designs: i) control plane load; ii) imposed processing delay in the software and hardware P4BFT nodes; iii) end-to-end switch reconfiguration delay; and iv) ILP solution time. We execute the measurements for random controller placements and diverse data plane topologies: i) random topologies with fixed average node degree; ii) reference Internet2 [18]; and iii) data-center Fat-Tree ($k = 4$). We also vary and depict the impact of no. of switches, controller instances, and disjoint controller clusters. To compute paths between controllers and switches and between processing and destination switches, Reassigner leverages the Constrained Shortest Path First (CSPF) algorithm. For brevity, as an input to the optimization procedure in Reassigner, we assume edge weights of 1. The objective function used in processing node selection is Eq. 3, parametrized with $(w_1, w_2) = (1, 1)$.

P4BFT implementation is a combination of P4$_{16}$ and P4 Runtime code, compiled for software and physical execution on P4 software switch *bmv2* (*master* check-out, December 2018) and a *Netronome Agilio SmartNIC* device with the corresponding firmware compiled using `SDK 6.1-Preview`, respectively. Apache Thrift and gRPC are used for population of registers and table entries in *bmv2*, respectively. Thrift is used for both table and registers population for the *Netronome SmartNIC*, due to the current SDK release not fully supporting the P4 Runtime. HTTP REST is used in exchange between P4 switch control plane and the Reassigner. The Reassigner and network controller replicas are implemented as Python applications.

B. Communication Overhead Advantage

Figure 2 depicts the packet load improvement in P4BFT over the existing reference solutions [5]–[7] for randomly generated topologies with average node degree of 4. The footprint improvement is defined as $1 - \frac{F_C^{P4BFT}}{F_C^{SoA}}$, where F_C

denotes the sum of packet footprint for control flows destined to each destination switch of the network topology as per Sec. IV-B and Fig. 1. P4BFT outperforms the state-of-the-art as each of the presented works assumes an uninterrupted control flow from each controller instance to the destination switches. P4BFT, on the other hand, aggregates control packets in the processing nodes that, subsequently to collecting the control packets, forward a single correct message towards the destination, thus decreasing the control plane load.
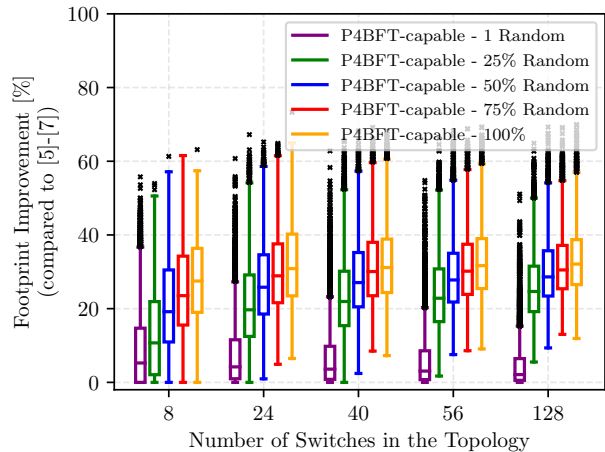

Fig. 2. Packet load improvement of P4BFT over the reference works [5]–[7] for 5000 randomly generated network topologies per scenario, with 7 controllers distributed into 3 disjoint and randomly placed clusters. In addition to the 100% coverage where each node may be considered a P4BFT processing node, we include scenarios where only the random [1, 25%, 50%, 75%] nodes of all available nodes in the infrastructure are P4BFT-enabled. Thus, even in the topologies with limited programmable data plane resources, i.e., in brownfield-scenarios involving OpenFlow/NETCONF+YANG non-P4 configuration targets, P4BFT offers substantial advantages over existing SoA.

Fig. 3 (a) and (b) portray the footprint improvement scaling with the number of controllers and disjoint clusters. P4BFT's footprint efficiency generally benefits from the higher number of controller instances. Controller clusters, on the other hand, aggregate replicas behind the same edge switch. Thus, with the higher number of disjoint clusters, the degree of aggregation and the total footprint improvement decreases.

C. Processing and Reconfiguration Delay

Fig. 4 depicts the processing delay incurred in the processing node for a single client request. The delay corresponds to the P4 pipeline execution time spent on identification of a correct controller message, comprising the i) hash computation over controller messages; ii) incrementing the counters for the computed hash; iii) signing the correct packet and; iv) propagating it to the correct egress port. When using the P4-enabled SmartNIC, P4BFT decreases the processing time compared to *bmv2* software target by two orders of magnitude.

Fig. 5 depicts the total reconfiguration delay imposed in SoA and P4BFT designs for $(w_1, w_2) = (1, 1)$ (ref. Eq. 3). It considers the time difference between issuing a switch reconfiguration request, until the correct controller message is determined and applied in the destination. Related works process the reconfiguration messages stemming from controller replicas in the destination target, their control flows
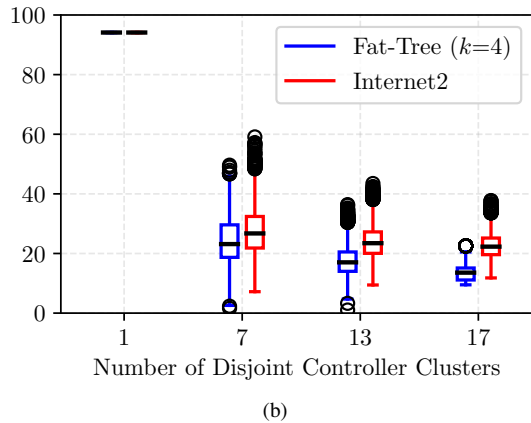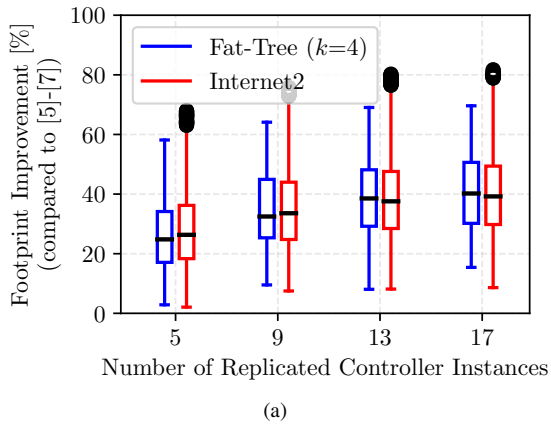
(a)

(b)

Fig. 3. The impact of (a) controllers and; (b) disjoint controller clusters on the control plane load footprint in Internet2 and Fat-Tree ($k = 4$) topologies for 5000 randomized controller placements each. (a) randomizes the placement but fixes the no. of disjoint clusters to 3; (b) randomizes the no. of disjoint clusters between [1, 7, 13, 17] but fixes the no. of controllers to 17. The resulting footprint improvement scales with the number of controllers but is inversely proportional to the number of disjoint clusters.

Fig. 4. The CDF of processing delays imposed in a P4BFT's processing node for a scenario including 5 controller instances. 3 correct packets and thus 3 P4 pipeline executions are necessary to confirm the payload correctness when tolerating 2 Byzantine controller failures.

traversing shortest paths in all cases. On average, P4BFT's reconfiguration delay is comparable with related works, the overall control plane footprint being substantially improved.

D. Optimization procedure in Reassigner

1) Impact of optimization objectives: Figure 6 depicts the Pareto frontier of optimal processing node assignments w.r.t. the objectives presented in Section IV-B: the total control plane footprint (minimized as per Eq. 1) and the reconfiguration delay (minimized as per Eq. 2). From the total solution
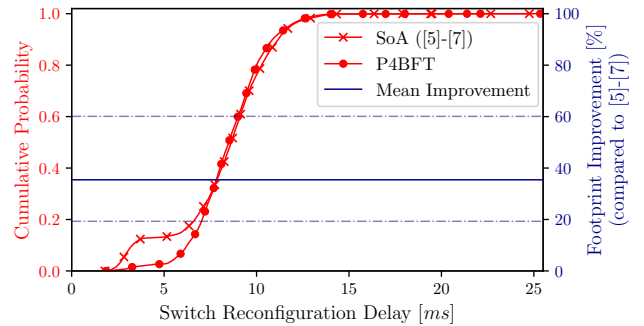

Fig. 5. CDFs of time taken to configure randomly selected switches in SoA and P4BFT environments for Internet2 topology, 10 random controller placements for 5 replicas and 1700 individual requests per placement. SoA works [5]-[7] collect, compare and apply the controllers' reconfiguration messages in the destination switch thus effectively minimizing the reconfiguration delay at all times. P4BFT, on the other hand, may occasionally favor footprint minimization over the incurred reconfiguration delay and thus impose a longer critical path, leading to slower reconfigurations. On average, however, P4BFT imposes comparable reconfiguration delays at a much higher footprint improvement (depicted blue), mean being 38%, best and worst cases at 60% and 19.3%, respectively, for evaluated placements.

space, depending on the weights prioritization in Eq. 3, either $(26.0, 3.0)$ or $(28.0, 2.0)$ solutions can be considered optimal. Comparable works implicitly minimize the incurred reconfiguration delay but fail to consider the control plane load. Hence, they prefer the $(30.0, 2.0)$ solution (encircled red).
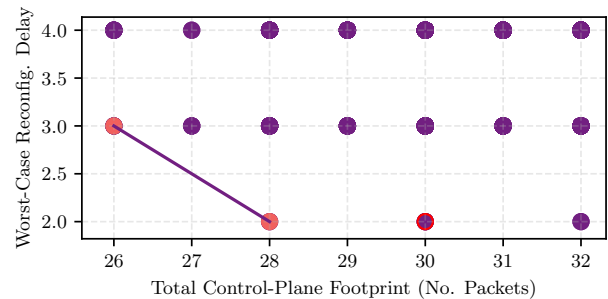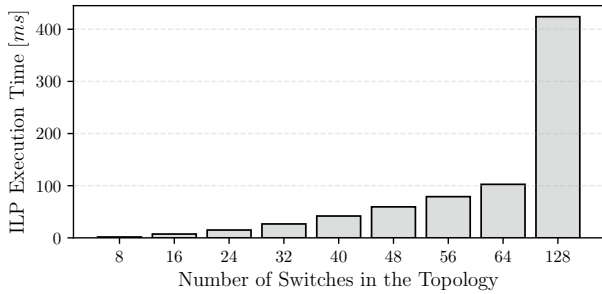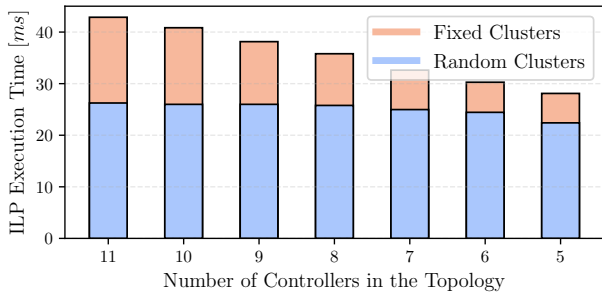

Fig. 6. Pareto Frontier of P4BFT's solution space for the topology presented in Fig. 1. The comparable works tend to minimize the incurred reconfiguration delay, but ignore the imposed control plane load. [5]–[7] hence select $(30.0, 2.0)$ as the optimal solution (encircled in red) while P4BFT selects $(26.0, 3.0)$ or $(28.0, 2.0)$ thus minimizing the total overhead as per Eq. 3.

2) ILP solution time - impact of topology, amount of controller and disjoint clusters: The solution time for the optimization procedure considering random topologies with average network degree of 4 and a fixed no. of randomly placed controllers is depicted in Fig. 7 (a). The solution time scales with number of switches, peaking at $420ms$ for large 128-switch topologies. The reassignment procedure is executed in few rare events: during network bootstrapping, on malicious / failed controller detection and following a switch / link failure. Thus, we consider the observed solution time short and viable for online mapping. Fig. 7 (b) depicts the ILP solution time scaling with the number of active controllers. The lower the number of active controllers, the shorter the solution time. In "Fixed Clusters" case, each controller is placed in its disjoint cluster (worst-case for the optimization).

(a)

(b)

Fig. 7. (a) depicts the impact of network topology size on the ILP solution time for random topologies. (b) depicts the impact of controller number and cluster disjointness in the case of Internet2 topology. The results are averaged over 5000 per-scenario iterations. The higher the cluster aggregation of controllers, the lower the ILP solution time. "Fixed Clusters" considers the worst-case, where each controller is randomly placed, but disjointly w.r.t. the other controller instances. Clearly, the ILP solution time scales with the amount of deployed switches and controllers. We used Gurobi 8.1 optimization framework configured to execute multiple solvers on multiple threads simultaneously, and have chosen the ones that finish first.

The "Random Clusters" case considers a typical clustering scenario, where a maximum of $[1..3]$ clusters are deployed, each comprising a uniform number of controller instances. The higher the cluster aggregation, the lower the ILP solution time.

VI. CONCLUSION

P4BFT introduces a switch control-plane/data-plane co-design, capable of malicious controller identification while simultaneously minimizing the control plane footprint. By merging the control channels in P4-enabled *processing* nodes, the use of P4BFT results in a lowered control plane footprint, compared to existing designs. In a hardware-based data plane, by offloading packet processing from general purpose CPU to the data-plane NPU, it additionally leads to a decrease in request processing time. Given the low solution time, the presented ILP formulation is viable for on-line execution. While we focused on an SDN scenario here, future works should consider the conceptual transfer of P4BFT to other application domains, including stateful web applications and critical industrial control systems.

ACKNOWLEDGMENT

REFERENCES

[1] H. Howard, M. Schwarzkopf, A. Madhavapeddy, and J. Crowcroft, "Raft refloated: Do we have consensus?" *ACM SIGOPS Operating Systems Review*, vol. 49, no. 1, 2015.

[2] J. Medved, R. Varga, A. Tkacik, and K. Gray, "OpenDaylight: Towards a model-driven SDN controller architecture," in *Proceedings of IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks 2014*. IEEE, 2014, pp. 1–6.

[3] P. Berde, M. Gerola, J. Hart, Y. Higuchi, M. Kobayashi, T. Koide, B. Lantz, B. O'Connor, P. Radoslavov, W. Snow *et al.*, "ONOS: Towards an open, distributed SDN OS," in *Proceedings of the third workshop on Hot topics in software defined networking*. ACM, 2014, pp. 1–6.

[4] C. Copeland *et al.*, "Tangaroa: A Byzantine Fault Tolerant Raft," http://www.scs.stanford.edu/14au-cs244b/labs/projects/copeland_zhong.pdf, [Accessed March-2019].

[5] H. Li, P. Li, S. Guo, and A. Nayak, "Byzantine-resilient secure software-defined networks with multiple controllers in cloud," *IEEE Transactions on Cloud Computing*, vol. 2, no. 4, pp. 436–447, 2014.

[6] P. M. Mohan, T. Truong-Huu, and M. Gurusamy, "Primary-backup controller mapping for Byzantine fault tolerance in software defined networks," in *GLOBECOM 2017 - 2017 IEEE Global Communications Conference*. IEEE, 2017, pp. 1–7.

[7] E. Sakic, N. Ðerić, and W. Kellerer, "MORPH: An adaptive framework for efficient and Byzantine fault-tolerant SDN control plane," *IEEE Journal on Selected Areas in Communications*, vol. 36, no. 10, pp. 2158–2174, 2018.

[8] L. Schiff, S. Schmid, and P. Kuznetsov, "In-band synchronization for distributed SDN control planes," *ACM SIGCOMM Computer Communication Review*, vol. 46, no. 1, pp. 37–43, 2016.

[9] A. S. Muqaddas, A. Bianco, P. Giaccone, and G. Maier, "Inter-controller traffic in ONOS clusters for SDN networks," in *2016 IEEE International Conference on Communications (ICC)*. IEEE, 2016, pp. 1–6.

[10] E. Sakic and W. Kellerer, "BFT protocols for heterogeneous resource allocations in distributed SDN control plane," in *2019 IEEE International Conference on Communications (IEEE ICC'19)*, Shanghai, P.R. China, 2019.

[11] E. Sakic and W. Kellerer, "Response time and availability study of RAFT consensus in distributed SDN control plane," *IEEE Transactions on Network and Service Management*, vol. 15, no. 1, 2018.

[12] University of Surrey - 5G Innovation Centre, "5G Whitepaper: The Flat Distributed Cloud (FDC) 5G Architecture Revolution," 2016.

[13] H. T. Dang, M. Canini, F. Pedone, and R. Soulé, "Paxos made switch-y," *ACM SIGCOMM Computer Communication Review*, vol. 46, no. 2, pp. 18–24, 2016.

[14] Y. Zhang, B. Han, Z.-L. Zhang, and V. Gopalakrishnan, "Network-assisted raft consensus algorithm," in *Proceedings of the SIGCOMM Posters and Demos*. ACM, 2017, pp. 94–96.

[15] A. Sapio, I. Abdelaziz, A. Aldilaijan, M. Canini, and P. Kalnis, "In-network computation is a dumb idea whose time has come," in *Proceedings of the 16th ACM Workshop on Hot Topics in Networks*. ACM, 2017, pp. 150–156.

[16] P. Bosshart, D. Daly, G. Gibb, M. Izzard, N. McKeown, J. Rexford, C. Schlesinger, D. Talayco, A. Vahdat, G. Varghese *et al.*, "P4: Programming protocol-independent packet processors," *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 3, pp. 87–95, 2014.

[17] M. Eischer and T. Distler, "Scalable byzantine fault tolerance on heterogeneous servers," in *2017 13th European Dependable Computing Conference (EDCC)*. IEEE, 2017, pp. 34–41.

[18] Internet2 Consortium, "Internet2 Network Infrastructure Topology," https://www.internet2.edu/media_files/422, [Accessed March-2019].