BFT Protocols for Heterogeneous Resource Allocations in Distributed SDN Control Plane

Ermin Sakic*†, Wolfgang Kellerer*
*Technical University Munich, Germany, † Siemens AG, Germany
E-Mail:*{ermin.sakic, wolfgang.kellerer}@tum.de, † ermin.sakic@siemens.com

***Abstract*—Distributed Software Defined Networking (SDN) controllers aim to solve the issue of single-point-of-failure and improve the scalability of the control plane. Byzantine and faulty controllers, however, may enforce incorrect configurations and thus endanger the control plane correctness. Multiple Byzantine Fault Tolerance (BFT) approaches relying on Replicated State Machine (RSM) execution have been proposed in the past to cater for this issue. The scalability of such solutions is, however, limited. Additionally, the interplay between progressing the state of the distributed controllers and the consistency of the external reconfigurations of the forwarding devices has not been thoroughly investigated. In this work, we propose an agreement-and-execution group-based approach to increase the overall throughput of a BFT-enabled distributed SDN control plane. We adapt a proven sequencing-based BFT protocol, and introduce two optimized BFT protocols that preserve the uniform agreement, causality and liveness properties. A state-hashing approach which ensures causally ordered switch reconfigurations is proposed, that enables an opportunistic RSM execution without relying on strict sequencing. The proposed designs are implemented and validated for two realistic topologies, a path computation application and a set of KPIs: switch reconfiguration (response) time, signaling overhead, and acceptance rates. We show a clear decrease in the system response time and communication overhead with the proposed models, compared to a state-of-the-art approach.**

I. Introduction and Problem Statement

Software Defined Networking (SDN) centralizes the decision-making in a dedicated *controller* component. Concepts for achieving crash-fault-tolerance and scalable operation of the controller have been presented in the past [1], [2]. By means of a logical distribution of controller replicas and the state synchronization, the controller instances are able to synchronize the results of their individual computations and come to consistent decisions independent of the instance that handled the client request. However, these approaches are based on weak crash-tolerant algorithms (e.g. RAFT [3] and Paxos [4]) that are unable to cater for malicious and incorrect (e.g., buggy [5]) controller decisions that have an individual controller instance fault as a root cause. Recent works have thus highlighted the importance of deploying Byzantine Fault Tolerance (BFT) protocols for achieving consensus, in scenarios where a subset of controllers is faulty due to a malicious adversary or internal bugs. Realizing a BFT SDN control plane comes with an additional controller deployment overhead, previously shown to range between $2F_M + F_A + 1$ [6] and $3(F_M + F_A) + 1$ [7] controller instances required to tolerate up to F_M strictly Byzantine and F_A fail-crash failures.

To support stateful controller-based applications (i.e., resource-constrained routing, load-balancing, stateful firewalls), the controllers synchronize their internal state updates. Traditional BFT designs [7], [8] require active participation of all replicas in the system. Thus, they leverage an RSM approach to handle the client requests, where a majority of controller instances must come to the agreement about the order of the client requests, before subsequently executing them. Finally, the controllers reach consensus on the output of the computation in order to ensure the causality of subsequent decisions. We have identified two issues with this approach.

First, to preserve causality, the non-faulty replicas always participate in all system operations. In the absence of faults, more replicas execute the decision-making requests than required to make progress, thus strongly limiting the execution throughput of the system. Namely, the application execution is handled by each controller instance in the cluster. In heterogeneous environments, where particular controller replicas can be assigned a higher resource set compared to the others, this leads to an under-utilization of fast replicas, as the system progresses at best at the speed of the $\lfloor \frac{|\mathcal{C}|+1}{2} \rfloor + 1$ fastest replica ($|\mathcal{C}|$ is the number of deployed controllers) [2]. Second, these BFT implementations rely on reaching a successful agreement about the sequence number mapping for each arriving client request, prior to its actual execution. The agreement phase thus necessarily increases the total processing time of individual requests. We claim that the serialization of requests is a *mean to an end* and that the causality of configurations on individual external devices (i.e., switches) is a sufficient constraint.

II. Our contribution

In this work we make a point that an optimal separation of the controller cluster into *sufficiently-sized* agreement and execution (A&E) groups leads to an overall higher utilization in request processing. In our approach, faster replicas may be leveraged in the intersection of different A&E groups, while slower replicas may run at their assigned speed without negatively influencing the faster replicas. To identify the A&E groups, we extend an existing ILP formulation for controller-switch assignment procedure [6]. The solver identifies an A&E group for each deployed switch element, while maximizing the overlap of the members of different groups. The formulation considers the execution capacity of individual controllers, as well as the switch-controller delays as its constraints. The solver executes during runtime, thus optimizing the assignment upon each discovered Byzantine/fail-crash failure.

To cater for the second issue, we adopt the classical Practical BFT (PBFT) approach [8] to realize a distributed sequencer in order to minimize the fail-over time in the case of a leader failure. We additionally introduce a group-based variant of this protocol, that leverages the partitioning of the total controller set into multiple A&E groups. Finally, in addition to the two *agreement-based* designs above, we present an *opportunistic* protocol design. With the opportunistic approach, successful handling of a client request implies reaching a consensus on a *consistent* device reconfiguration while preserving the *causality* of decisions, subsequent to the actual request handling. We

achieve the causality and agreement by reaching consensus: i) on the controller state at the time of application execution; ii) on the actual computed output result (to guarantee the consistency of decisions).

We have implemented these three BFT protocols and have analyzed the overheads of switch reconfiguration time, the communication overhead and the request acceptances rates. We ran our evaluation for emulated Open vSwitch-based Internet2 and Fat-Tree topologies, comprising up to 34 Open vSwitch instances and up to 13 controllers, while considering a varied number of tolerated Byzantine failures.

Paper structure: Sec. III introduces the overall system model. Sec. IV details the proposed BFT protocols. Sec. IV-D discusses the ILP formulation for the optimal controller-switch assignment. Sec. V presents the evaluation methodology. Sec. VI discusses the results. Sec. VII summarizes the related work. Sec. VIII concludes this paper.

III. System Model

In [6], we discussed the often neglected differentiation between state-independent (SIA) and state-dependent (SDA) SDN applications. The SDA require an up-to-date and synchronized application state in order to serve the client requests. In this work, we consider solely the global SDA operations where successfully handled client requests result in stateful write operations to the replicated data-store. The subsequent client request executions that result in new writes to the same state must consider the preceding writes for their correctness. The value of the write operation is determined by an execution of a multi-phase BFT protocol. We distinguish *accepting* and *rejecting* protocol executions. *Rejecting* executions are caused by replicas that interrupt the run because of a missing consensus in one of the protocol *phases* (caused by e.g., conflicting seq. no. proposals, faulty controllers and packet loss). We assume that clients retransmit the requests until a successful execution has been acknowledged by the controllers.

Our SDN architecture is comprised of: i) controllers that individually execute an instance of a BFT process; ii) the switches that implement a comparison mechanism for matching controller configuration messages (as per [6]); iii) the clients; iv) a REASSIGNER component that maintains the switch-controller assignments (as per [6]). The *request-initiating* clients comprise northbound clients (e.g., applications, administrators) and the switches capable of forwarding the client requests as (OpenFlow) *packet-in* messages to the SDN controllers (e.g., routing, load-balancing requests). The *target clients* represent the configuration targets, e.g., switches that are (re)configured as a result of request handling.

We assume a *fair-loss* link abstraction, where a message (re-)transmitted infinitely often is eventually delivered at the recipient. Packets may be arbitrarily dropped, lost, delayed, duplicated and delivered out of order during any of the BFT protocol phases. The SDN control plane is realized in either in-band or out-of-band manner. Control messages exchanged between the controller, switches and clients are assumed to be signed, thus ensuring: i) the integrity of messages exchanged using the SDN data plane; ii) message forging is impossible.

State-updates distribution assumes an eventually synchronous model as per [9], where different replicas possess different views of the current configuration state for a limited time duration. Eventually, given an appropriately long quiescent period, all correct replicas converge to the same state. We assume that a bounded number of controllers may exhibit Byzantine behavior and/or fail-crash failures, respectively.

IV. BFT Consensus Protocols and the Controller-Switch Assignment Methodology

The proposed protocols guarantee the following properties:

- *Uniform Agreement*: When a correct replica commits a particular internal state/switch update (i.e., computes a particular response), all correct replicas eventually commit the same update.
- *Liveness*: All correct replicas eventually finalize the processing of each client request. The resulting run is declared either *accepting* or *rejecting*.
- *Causality*: The updates to the controller data-store and the per-switch configuration updates are executed in a causally dependent order. The controller's decision to reconfigure a switch take into account all preceding configurations of that switch.

We assume a deployment of a total of $2F_M + F_A + 1$ controllers per agreement and execution (A&E) group in order to tolerate an upper bound of individual F_M Byzantine and F_A fail-crash controller failures in that particular A&E group.

In the remainder of this section, we introduce the three BFT protocols: the agreement-based MPBFT and SBFT protocols, and the opportunistic OBFT (ref. Table I).

TABLE I
Overview of presented BFT protocols.

Alg.	Name	Type	No. Rounds
MPBFT	Modified PBFT	Agreement-based	2
SBFT	Serialized A Priori BFT	Agreement-based	3
OBFT	Opportunistic A Posteriori BFT	Opportunistic	2

A. Pre-serialization model MPBFT (agreement-based)

Modified PBFT (MPBFT) imposes a **single** A&E group where each active controller replica is tasked with execution of an agreed command. The workflow of MPBFT is visualized in Fig. 1. A *request-initiating* client initially invokes its application request to all active controller replicas (`REQUEST` phase). For each incoming client request, each controller replica assigns a unique sequence number and distributes this sequence number proposal to the other controllers in the cluster (`PREPARE` phase). The replicas compare the sequence number proposals. If the correct majority of proposals are matching (i.e., the same sequence number is proposed by the majority of correct replicas), successful global agreement has been reached. At the begin of the `COMMIT` phase, each correct replicas executes the client request. The execution output is subsequently broadcasted by each replica to the remainder of the cluster and the collected output responses are once again compared in all replicas. Each controller deduces the correct majority response and eventually commits the output to its local data-store (i.e., a store of reservations) and finally reports the agreed output to the target clients (`REPLY` phase). After collecting F_M+1 consistent output messages, the *target clients* (e.g., switches) decide to apply the new configuration.

MPBFT is a variation of PBFT [8] that requires no leader and is thus tolerant to individual node failures. Compared to PBFT, we shorten the protocol execution by one round.

Whereas PBFT proposes a PRE−PREPARE round, MPBFT skips this round by leveraging a client-initiated atomic multicast execution and a *distributed sequencer*. Namely, each new client request is multicasted to each replica of the system. The replicas propose a new seq. number for the request by incrementing the current counter as per Alg. 1. The seq. numbers for new client requests are assigned based on the current state of a local atomic counter. Following an arrival of a new request, the replicas yield the lowest unallocated seq. number value and propose this seq. number to the remaining replicas. After collecting a *sufficient* amount of matching PREPARE messages, all *correct* replicas decide to accept the seq. number contained in the correct majority proposal as the final seq. number for this request. Table III summarizes the exact amounts of required matching messages to progress the protocol execution.

If no correct majority vote is achieved during the agreement process on either the sequence number or the computed output, the replicas respond with a rejection status. If sufficient rejection messages are collected, the current execution is cancelled and the run is declared *rejecting*. Concurrent client requests can lead to same sequence numbers being assigned to different requests at different replicas, thus resulting in rejecting runs.

The execution capacity of MPBFT is limited by the slowest replica in the system. Consider the scenario $F_M = 1$, $F_A = 0$ depicted in Fig. 1. Each controller C_i is able to service request workload up to a capacity of P_i per observation interval. The portrayed system is thus able to service computations up to $max(\sum_{i=1..N} \lambda_i) \leq min(P_i)$, or 500 requests/interval (imposed by the capacity of $C4$ and $C5$). Thus, Client 1 (with processing requirement of $\lambda_1 = 500$) and Client 2 ($\lambda_2 = 400$) cannot be serviced concurrently. One can alternatively portray the depicted rates as continuous execution workloads. While active participation of $C4$ and $C5$ in the system is unnecessary to tolerate a single Byzantine fault, they are included in execution and signaling and are necessary to progress the system state. MPBFT's communication overhead is quadratic (ref. Table IV). With alternative protocol designs SBFT and OBFT, we next leverage the additional execution capacity by partitioning the control plane into multiple A&E groups.

B. Pre-serialization model SBFT (agreement-based)

With *Serialized A Priori BFT* (SBFT), agreement and execution processes are administered by multiple A&E groups. We assign for each request-initiating client (i.e., a northbound application, an edge switch) an A&E group according to the algorithm presented in Sec. IV-D. To tolerate F_M Byzantine and F_A fail-crash failures in the scope of a single A&E group, each group must comprise $2F_M + F_A + 1$ controllers. Multiple execution groups can process the client requests concurrently. SBFT design is depicted in Fig. 2. Compared to MPBFT, SBFT introduces the PRE−PREPARE step, where the replicas belonging to the A&E group propose and subsequently notify the remainder of the replicas of an assigned sequence number. In an *accepting* run, the group replicas collect the responses in the PREPARE phase and reach consensus by collecting $\lceil \frac{|\mathcal{C}|+F_M+1}{2} \rceil$ matching sequence number proposals. Finally, the replicas of the A&E group execute the request in the COMMIT phase and broadcast the response to all remaining replicas. If $F_M + 1$ matching outputs are received, the replicas apply the internal state reconfiguration and notify the target clients of the final result during REPLY. The communication overhead

Algorithm 1 Logical Sequencer: Ordering of client requests

Notation:
M_P Client request (e.g. flow request)
M_C Replica message (seq. no. proposal) initiated at a remote controller
$\mathcal{C}$ Set of available SDN controllers
R_{ID} Unique client request identifier
$\mathcal{R}_{mappings}$ Mapping of client request ids to unique seq. numbers
S_{atomic} Atomic sequencer that yields the current seq. number

1: **upon event** *on-client-request* $< M_P, R_{ID} >$ **do**
2: ...
3: proposed_seq_no = *propose_seq_no*(R_{ID})
4: ...
5:
6: **upon event** *on-new-replica-sync-update* $< M_C, R_{ID} >$ **do**
7: ...
8: **switch** PHASE **do**
9: **case** MPBFT-PREPARE:
10: *propose_seq_no*(R_{ID})
11: **case** SBFT-PRE-PREPARE:
12: *propose_seq_no*(R_{ID})
13: ...
14:
15: **function** PROPOSE_SEQ_NO(R_{ID})
16: **if** $R_{ID} \in \mathcal{R}_{mappings}$ **then**
17: *return* $\mathcal{R}_{mappings}[R_{ID}]$
18: **else**
19: **while** $S_{atomic} \in \mathcal{R}_{mappings}.values()$ **do**
20: $S_{atomic} = S_{atomic} + 1$
21: $\mathcal{R}_{mappings}[R_{ID}]$ = S_{atomic}
22: *return* $\mathcal{R}_{mappings}[R_{ID}]$

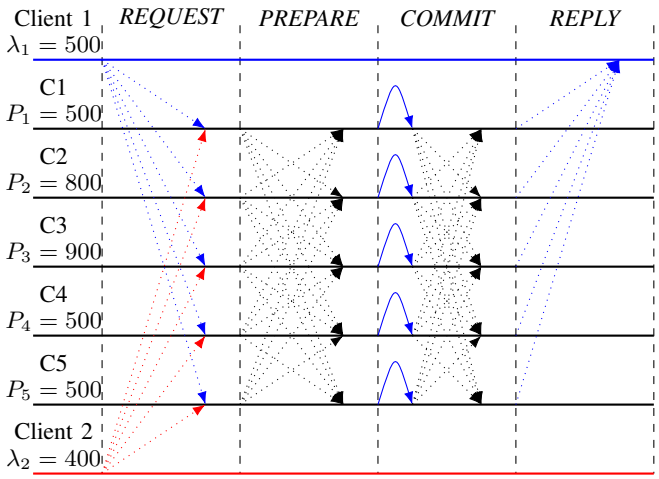


Fig. 1. MPBFT Model: In REQUEST phase, the clients initiate new executions. During PREPARE, the controller replicas agree on the execution order by reaching consensus on the assigned sequence number for the clients' requests. Each controller executes the request in the COMMIT phase. During REPLY, target clients are notified of reconfigurations. Client 2's requests cannot be serviced as a result of a limited processing capacity of the controllers.

of SBFT is bounded $\mathcal{O}(3|\mathcal{A}||\mathcal{C}|)$, and grows linearly for a fixed A&E group size.

Causality: To ensure that the causality property holds in MPBFT and SBFT, the controllers execute the sequenced request in order agreed during PREPARE. The replicas execute the COMMIT phase only if the outputs (i.e., the added reservations) for the preceding requests were seen by the executing replica. Thus, before handling subsequent requests, the status of preceding runs (*accepting*/*rejecting*) must be determined.

C. Post-negotiation model OBFT (opportunistic)

Opportunistic A Posteriori BFT (OBFT) is a speculative take on SBFT, where computations of the client requests

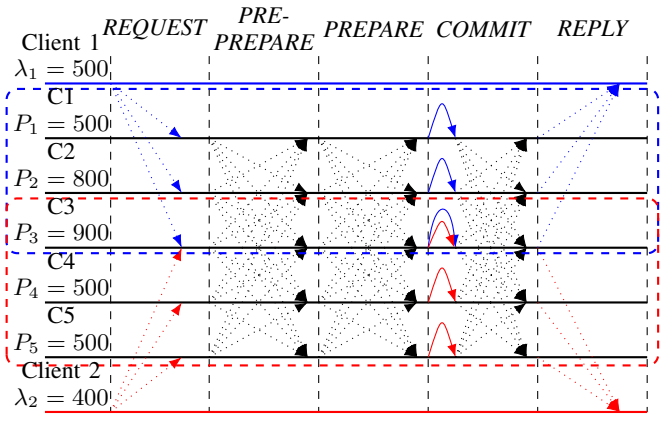


Fig. 2. SBFT Model: Compared to MPBFT, SBFT allows for more efficient allocation of execution resources, since execution is separated into multiple A&E groups. This comes with an overhead of a PRE−PREPARE step, required to reach consensus on the sequence number allocated to the request.

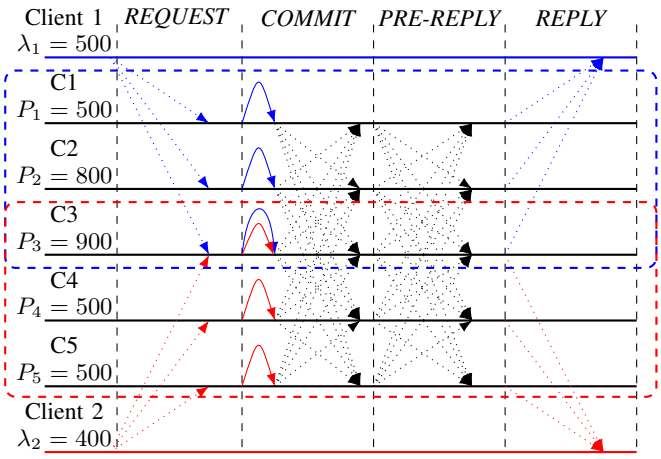


Fig. 3. OBFT model: An opportunistic protocol variation, where A&E group members execute their clients' requests prior to the distribution of reference state configurations based on which the computations were executed. The internal controller state and the target clients are updated only if the consensus on the reference state configurations could be reached for the *correct* majority of global controller instances (ref. Table III).

execute prior to reaching consensus about the computed output values. A global sequencer is not used in OBFT and thus PRE−PREPARE and PREPARE phases are omitted. Instead, each replica maintains the hashes of current switch configurations, as well as a state array containing the hashes of the configurations of the switches at the time of request executions (*TORC* hashes). Following the output computation in the COMMIT phase, the replicas come to consensus about the updated switch state in the PRE−REPLY phase. This workflow is depicted in Fig. 3.

In contrast to MPBFT and SBFT, in their COMMIT phase, the replicas belonging to the same A&E group compute the outputs, and in addition to the computed response outputs, they broadcast the hash arrays denoting their view of the target clients' configurations. Each accepting replica *that is not part of the serving A&E group* evaluates its actual current local view of the switch states, and *iff*: i) $F_M + 1$ matching output values have been computed by the A&E replicas; and b) their current view of switch configuration hashes is matching with those of the A&E replicas; they answer with an *accepting* status. The execution replicas (belonging to the A&E group), instead compare the proposed hash array with their local TORC hashes for the target client (i.e., target switches) and notify other A&E replicas of their status. If *sufficient* (ref. Table III) positive confirmations have been collected at the end of PRE−REPLY phase, each active controller internally commits the output proposed by the correct majority of the A&E group. The A&E group members then notify the configuration targets of the agreed output in REPLY phase. OBFT's comm. overhead is quadratic and grows with $|\mathcal{C}|$.

D. Dynamic Controller-Switch (Re)Assignment Procedure

In our design, each request-initiating client (i.e., a northbound client or a switch) is assigned a unique controller agreement and execution (A&E) group. Groups assigned to different switches are allowed to partially or fully overlap. Only the assigned controllers are required to contact the target clients and apply reconfigurations. Similarly, only these controllers are contacted by the request-initiating clients with new application requests. Our ILP formulation of the assignment problem aims to minimize the total overlap between the members of the active A&E groups, so to minimize the synchronization delay during the consensus executions. The proposed reassignment

Algorithm 2 Hash comparison in the OBFT-COMMIT phase

Notation:
R_{ID} Unique client request identifier
$\mathcal{HV}$ Current configuration hashes for the switches
$\mathcal{HVC}[\mathcal{R}_{\mathcal{ID}}]$ Switches' config. hashes prior request computation (TORC)
find-path() An exemplary SDN application logic operation
consensus() Returns consensus message according to the number of minimum required confirmations (ref. Table III)
$m^C_{R_{ID}}$ A COMMIT message for round R_{ID}
$M^C_{R_{ID}}$ Set of buffered COMMIT messages for R_{ID}

1: **procedure** HANDLE NEW CLIENT REQUEST
2: **upon event** *on-received-client-request* ($CL_{R_{ID}}$) **do**
3: $\mathcal{R}$ = *find-path*($CL_{R_{ID}}$.routing_request)
4: **for all** SW $\in \mathcal{R}$ **do**
5: current-hash[SW] = *hash*(SW.state)
6: $m^C_{R_{ID}}$.hash, $m^C_{R_{ID}}$.path = current-hash, $\mathcal{R}$
7: *broadcast-to-cluster-members*($m^C_{R_{ID}}$)
8:
9: **procedure** HANDLE INCOMING COMMIT MESSAGE
10: **upon event** *new-replica-sync-message*($m^C_{R_{ID}}$) **do**
11: $P^{R_{ID}}_C$ = *consensus*($M^C_{R_{ID}}$, <val, state-hash-array>)
12: *on-init-obft-pre-reply*($P^{R_{ID}}_C$,inline-with-replica-view($P^{R_{ID}}_C$))
13:
14: **function** INLINE-WITH-REPLICA-VIEW($P^{R_{ID}}_C$)
15: **for all** SW $\in P^{R_{ID}}_C$.path **do**
16: **if** $\mathcal{HV}$[SW] == $P^{R_{ID}}_C.hash$[SW] **then**
17: *pass ()*
18: **else if** $\mathcal{HVC}[R_{ID}]$[SW] == $P^{R_{ID}}_C.hash$[SW] **then**
19: *pass ()*
20: **else** *return* (REJECT)
21: *return* (ACCEPT)

mechanism, the objective function and the constraints extend the formulation presented in [6]. For brevity, we do not discuss each constraint in detail here, but refer the reader to the summary in Table V and [6]. The procedure is executed once at the system startup and dynamically during runtime, on each detected controller failure.

For each switch S_i we can derive a bitstring R_{S_i} comprised of *ones* for replicas actively assigned to S_i and *zeros* for the unassigned replicas. We then formalize the objective function:

TABLE II
NOTATION USED IN TABLES III, IV AND V.

Symbol	Meaning
$\mathcal{C}$	Set of active controllers in the system
F_M	No. of tolerated Byzantine faults in a single A&E group
$Req(t)$	Time-variant no. of controllers [6] that must be assigned to each switch, to tolerate the Byzantine failures
$\mathcal{S}$	Set of switches in the system
P_{C_i}	Total available controller C_i's capacity.
L_{CL_k}, L_{S_j}	Request processing load stemming from the northbound client CL_k and edge switch S_j, respectively.
$D_{C,S}$	Max. tolerable delay for controller-switch communication.
$\mathcal{A}$	Controller replicas belonging to a single A&E group
$\lvert\mathcal{M}_{agr}\rvert$	Sum of the tolerated Byzantine failures and the majority of correct replicas per A&E group: $\lceil\frac{\lvert\mathcal{A}\rvert+F_M+1}{2}\rceil$
$\lvert\mathcal{M}_{glob}\rvert$	Sum of the tolerated Byzantine replicas and the majority of all correct active replicas: $\lceil\frac{\lvert\mathcal{C}\rvert+F_M+1}{2}\rceil$
CMP	Comp. overhead of executing the packet comparison
E	Comp. overhead of executing SDN application operation

TABLE III
THE AMOUNT OF MATCHING MESSAGES REQUIRED TO REACH CONSENSUS IN THE RESPECTIVE PROTOCOL PHASE (WORST-CASE).

Algorithm	PRE-PREPARE	PREPARE	COMMIT	PRE-REPLY	REPLY
MPBFT	N/A	$\lvert M_{glob}\rvert$	F_M+1	N/A	F_M+1
SBFT	$\lvert M_{agr}\rvert$	$\lvert M_{glob}\rvert$	F_M+1	N/A	F_M+1
OBFT	N/A	N/A	$\lvert M_{agr}\rvert$	$\lvert M_{glob}\rvert$	F_M+1

$$min \sum_{S_j \in \mathcal{S}} \sum_{S_i \in \mathcal{S}, S_i \neq S_j} HD(R_{S_j}, R_{S_i}) \quad (1)$$

where $HD(R_{S_j}, R_{S_i})$ denotes the Hamming distance between the assignment bitstrings for S_j and S_i. Combined with the adapted *minimum assignment* constraint depicted in Table V, we ensure the building of minimum-sized A&E groups that fulfill the capacity and delay constraints of the clients.

V. EVALUATION

To evaluate the different BFT protocols, we realized a centralized path computation application that executes in each of the deployed controller replicas. Based on the sequence and current state of link reservations, the routing algorithm leverages Dijkstra algorithm to choose the optimal (cheapest) path w.r.t. bandwidth resource consumption, and thus implicitly load-balances the embedded flows in the given topology.

TABLE IV
COMPUTATIONAL AND COMMUNICATION OVERHEAD OF THE INTRODUCED BFT PROTOCOLS.

Alg.	Computational Overhead	Communication Overhead
MPBFT	$\mathcal{O}(2\lvert\mathcal{C}\rvert\text{CMP} + \lvert\mathcal{C}\rvert\text{E})$	$\mathcal{O}(2\lvert\mathcal{C}\rvert\lvert\mathcal{C}\rvert)$
SBFT	$\mathcal{O}(\text{CMP}(2\lvert\mathcal{C}\rvert+\lvert\mathcal{A}\rvert)+\lvert\mathcal{A}\rvert E)$	$\mathcal{O}(3\lvert\mathcal{A}\rvert\lvert\mathcal{C}\rvert)$
OBFT	$\mathcal{O}(2\lvert\mathcal{C}\rvert\text{CMP} + \lvert\mathcal{A}\rvert E)$	$\mathcal{O}(\lvert\mathcal{A}\rvert(\lvert\mathcal{C}\rvert+1)+\lvert\mathcal{C}\rvert(\lvert\mathcal{C}\rvert-1))$

TABLE V
CONSTRAINTS USED IN BUILDING THE A&E GROUPS.

Constraint	Formulation
Min. Assignment	$\sum_{C_i \in \mathcal{C}} A_{C_i,S_j} == Req(t), \forall S_j \in \mathcal{S}$
Unique Assignment	$A_{C_i,S_j} \leq 1, \forall C_i \in \mathcal{C}, S_j \in \mathcal{S}$
Bounded Capacity	$\sum_{S_j \in \mathcal{S}} A_{C_i,S_j} * L_{S_j} \leq P_{C_i} - \sum_{CL_k \in \mathcal{CL}} L_{CL_k}, \forall C_i \in \mathcal{C}$
Delay Bounds	$A_{C_i,S_j} * d_{C_i,S_j} \leq D_{C,S}, \forall C_i \in \mathcal{C}, S_j \in \mathcal{S}$

The BFT protocol executions take the source-destination pair and the required bandwidth as an input for the service request. Subsequently, the protocol computes the optimal path in the COMMIT phase and notifies the switches on the path of new reservation in the REPLY phase. To evaluate the designs of all three protocols, we consider the following performance metrics: i) time required to apply a new switch reconfiguration, measured from the time of a client request arrival until the confirmation of the last switch reconfiguration; ii) the acceptance rate for the new arrivals; ii) the total communication overhead.

To validate our claims in a realistic environment, we have emulated the Internet2 topology, as well as a fat-tree datacenter topology, encompassing 34 and 20 switches, respectively. The controllers in the Internet2 scenario were placed so to maximize the system coverage against failures as per [6], [10]. The controllers of the fat-tree topology were placed on the leaf-nodes as per [6], [11]. The state synchronization between the controllers and the resulting switch reconfigurations occur in in-band control mode. To provide for realistic delay emulation, we derive the link distances from the publicly available geographical Internet2 data[1] and inject the propagation delays using Linux's *tc* tool. A single client was placed at each switch of the Internet2 topology, while two clients were placed at each leaf-switch of the fat-tree topology. The arrivals for incoming service requests are modeled using n.e.d. [11].

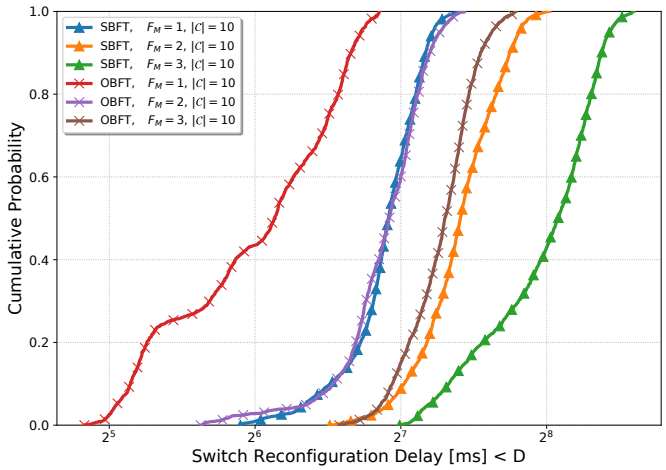
To generate the hashes for per-switch configuration state (ref. Sec. IV-C), we used Python's *hashlib* implementation and the SHA256 secure hash algorithm, defined in FIPS 180-2 [12]. We used Gurobi to solve the ILP formulated in Sec. IV-D. The measurements were executed on a commodity PC equipped with AMD Ryzen 1600 CPU and 32 GB RAM.

VI. DISCUSSION

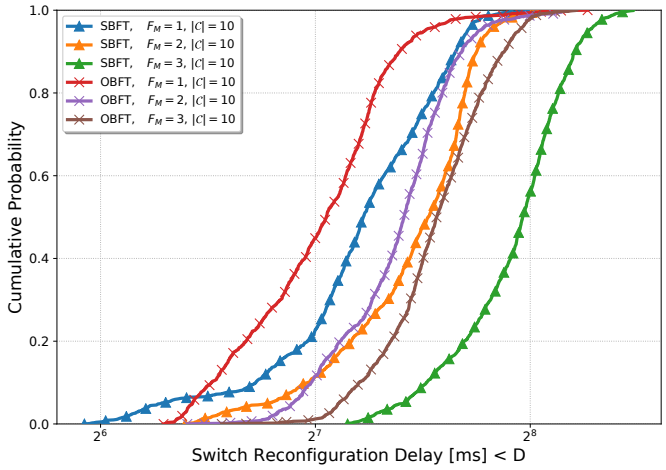
1) Total reconfiguration time for the internal controller and the switch state: Fig. 4a and Fig. 4b depict the accumulated response time starting with the reception of a client request at a controller replica until the last reconfiguration in one of the switches on the detected path. The total number of active controllers was fixed to $|\mathcal{C}| = 10$ and the measurement was executed for A&E group sizes varying between $|\mathcal{A}| = 3$ and $|\mathcal{A}| = 7$ ($F_M = 1$ and $F_M = 3$, respectively) controllers. Rejecting executions were not considered. Both Fat-Tree and Internet2 topologies depict the benefit of opportunistic execution and a lower number of phases in OBFT in all scenarios.

In Fig. 5a and Fig. 5b, we vary the total number of deployed active controllers. The figures portray how MPBFT provides equal performance for the controller constellations where the A&E group size in SBFT and OBFT approximately equals the total number of active controllers (all controllers belong to the same A&E group). After provisioning additional replicas (case for $|\mathcal{C}| = [7..13]$), the performance of MPBFT starts to suffer compared to both SBFT and OBFT, as it requires interactions between all instances of controllers for successful request handling, whereas SBFT and OBFT continue to operate at the level of a constant A&E group size. OBFT offers the best performance in both topologies. This is due to SBFT and MPBFT requiring additional rounds to handle the request sequencing, compared to OBFT, that ensures the causality property holds per-switch, even in the case of unordered executions. MPBFT

[1]Internet2 topological data (provided by POCO project) - https://github.com/lsinfo3/poco/tree/master/topologies

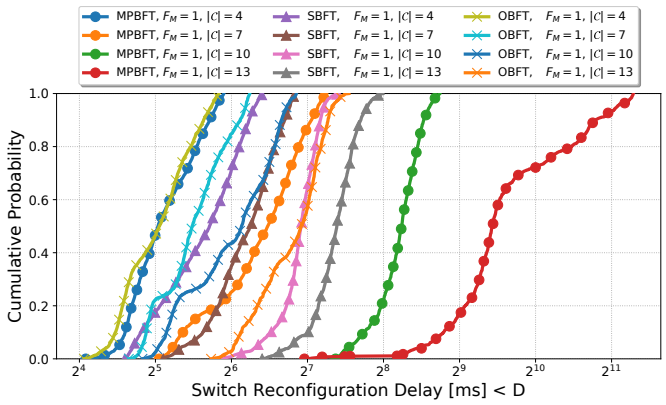


(a) Fat-Tree topology

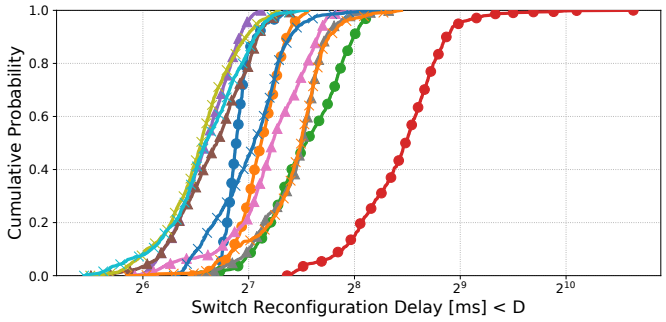


(b) Internet2 topology

Fig. 4. Total accumulated switch reconfiguration (system response) time for varied sizes of A&E groups for max. tolerated Byzantine failures $F_M = [1..3]$, $F_A = 0$ and a fixed total number of active controllers $|\mathcal{C}| = [10]$. OBFT shows dominantly lower commit delays in all depicted scenarios.



(a) Fat-Tree Topology



(b) Internet2 Topology

Fig. 5. Total accumulated switch reconfiguration (system response) time for varied numbers of active controller replicas $|\mathcal{C}| = [4..13]$, $F_A = 0$ and an A&E group size of $|\mathcal{A}| = 3$. While OBFT portrays the lowest reconfiguration delays, its performance is similar to SBFT and MPBFT for small control planes (especially for Internet2), slightly better compared to SBFT and largely dominant compared to MPBFT for larger topology sizes.

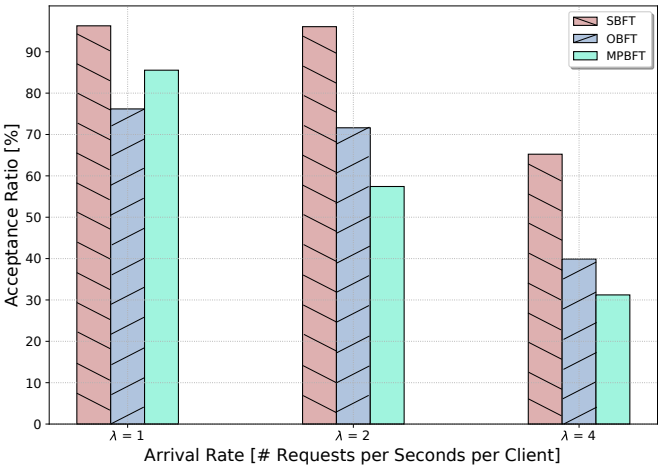


Fig. 6. Acceptance rates for incoming client requests in fat-tree topology and $F_M = 1, |\mathcal{C}| = 4$. SBFT tends to execute a higher number of *successful* runs compared to: i) MPBFT, due to its larger number of active replicas involved in sequencing process and ii) OBFT, due to its opportunistic design, where consistency of outputs is agreed upon after execution has finished.

suffers further since the commands execute on each of the controller replicas. Hence, its consensus requires on average an inclusion of a larger number of replicas compared to SBFT and OBFT. Internet2 topology depicts a lower discrepancy between SBFT and OBFT and highlights the benefit of sequencing in geographically distributed scenarios where network delays cause a longer asynchronous period and thus a higher probability of execution overlaps (confirmed by Fig. 6). The maximum path lengths are higher for Internet2 topology, thus resulting in a higher number of overlapping reservations that cause execution rejections/stalling period in opportunistic OBFT.

2) Acceptance rates for arriving requests: In Fig. 6 we vary the per-client arrival rates λ for incoming client requests. In the case of $\lambda = 4$, up to 64 requests/second are processed by the cluster in Internet2 topology. Opportunistic execution of OBFT and subsequent hash comparison tends to result more often in rejecting runs, compared to SBFT that serializes all requests prior to their processing. MPBFT results in a relatively high percentage of rejections, due to a higher chance of conflicting sequence number handouts that may occur concurrently since all replicas are involved in proposals during `PREPARE` phase.

3) Communication overhead: Fig. 7 depicts the scaling of communication overhead with the increase of the total number of active controllers. Controller-to-Switch (C2S) communication overhead increases with the number of controllers that execute the operation and communicate their result to the target switches. Thus, following an output response computation, in MPBFT each controller distributes the newly computed configurations to switches, hence the linear overhead increase. Since the size of the A&E group remains unchanged throughout all depicted scenarios, SBFT and OBFT show a constant low C2S

overhead. The Controller-to-Controller (C2C) overhead scales with the number of active controllers involved in the A&E group. For MPBFT and OBFT, this increase is quadratic. For SBFT, the C2C overhead increase is linear. It should be noted that the linear evolution holds only for constant A&E group sizes, i.e., for fixed F_M and F_A.

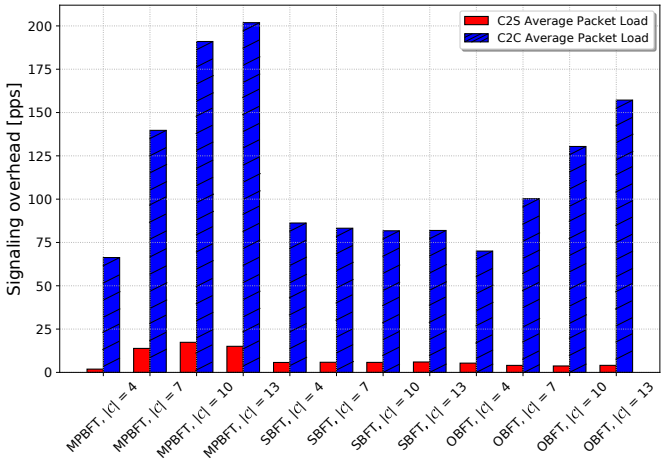


Fig. 7. Signaling overhead [pps] when serving 16 requests/second for a varied number of controllers $|\mathcal{C}| = [4..13]$ and a fixed A&E group size $|\mathcal{A}| = 3$. SBFT possesses the lowest overhead (linear growth), followed by OBFT and MPBFT, that show a quadratic growth scaling with $|\mathcal{C}|$.

Additional notes: While SBFT and MPBFT ensure a single execution and validation of inputs for client requests (i.e., each client sequence number is mapped to a unique request), OBFT executes client requests speculatively, prior to reaching consensus. Thus, Byzantine clients may attempt affecting the order of execution, or generate execution contentions. Metering mechanisms for misbehaving clients and their exclusion could cater for this case. They are, however, not in the scope of this work.

VII. Related Work

Agreement-based approaches have focused on the optimization of sequencing procedure by minimizing the number of replicas that actively participate in sequence proposals [13], [14]. *REBFT* [13] keeps only a subset ($2F + 1$ of a total of $3F + 1$) replicas active during normal case operation. It activates the passive replicas only after a detected replica fault. Such approaches rely on a trusted counter implementation to prevent *equivocation*, the capability of a malicious replicas to send conflicting proposals to other members. Since we do not assume a centralized proposer, we prevent equivocation by deciding new seq. numbers individually, without the overhead of a trusted counter nor passive replica activation delay.

Speculative BFT protocols have been investigated in [15], [16]. However, these approaches conclude about the consensus of the computed decisions based on the comparison of the instantaneous outputs and assume a *stateless* operation. In the contrast, in OBFT we leverage the agreement procedure that relies on external outputs, i.e., *stateful* per-switch configurations that are inherent to network management scenarios.

Omada [17] is a sequencing-based BFT design that assigns replicas with either agreement or execution roles and parallelizes the agreement phase. It highlights the benefit of selecting a configuration with the lowest number of agreement groups. Contrary to our work, the authors assume a centralized sequencer per agreement group. Distinguishing causality property per configuration target is not discussed nor leveraged in their protocol. Similarly, Omada does not provide an insight into opportunistic approaches to execution handling.

VIII. Conclusion

We have implemented two agreement-based and an opportunistic BFT protocol for the purpose of SDN controller state synchronization, and have analyzed their overheads in an emulated environment using software switches and emulated network delays. The evaluated KPIs include the switch reconfiguration times, the request acceptance rates and the communication overhead. We have shown how our opportunistic BFT approach leverages agreement of switch state at the time of request computation to ensure the causality during request reconfiguration. It offers considerably lower response time compared to the sequencing-based approaches. However, this benefit comes at the expense of a lower acceptance rate and quadratic communication overhead. For those metrics, the A&E group-based sequencing approach SBFT presents a better alternative. Both approaches result in a higher throughput compared to MPBFT, which adapts the PBFT protocol.

Acknowledgment

This work has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement number 780315 SEMIOTICS. We are grateful to Nemanja Deric, Arled Papa, Johannes Riedl and the reviewers for their useful feedback and comments.

References

[1] D. Suh *et al.*, "On performance of OpenDaylight clustering," in *NetSoft Conference and Workshops (NetSoft), 2016 IEEE*. IEEE, 2016.
[2] E. Sakic *et al.*, "Response Time and Availability Study of RAFT Consensus in Distributed SDN Control Plane," *IEEE Transactions on Network and Service Management*, 2017.
[3] H. Howard *et al.*, "Raft Refloated: Do we have Consensus?" *ACM SIGOPS Operating Systems Review*, vol. 49, no. 1, 2015.
[4] L. Lamport *et al.*, "Paxos made simple," *ACM Sigact News*, vol. 32, no. 4, 2001.
[5] P. Vizarreta *et al.*, "Mining Software Repositories for Predictive Modelling of Defects in SDN Controller," in *IFIP/IEEE International Symposium on Integrated Network Management*, 2019.
[6] E. Sakic *et al.*, "MORPH: An Adaptive Framework for Efficient and Byzantine Fault-Tolerant SDN Control Plane," *IEEE Journal on Selected Areas in Communication*, 2018.
[7] H. Li *et al.*, "Byzantine-resilient secure software-defined networks with multiple controllers in cloud," *IEEE Transactions on Cloud Computing*, vol. 2, no. 4, 2014.
[8] M. Castro *et al.*, "Practical Byzantine fault tolerance," in *OSDI*, vol. 99, 1999.
[9] A. Miller *et al.*, "The honey badger of BFT protocols," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2016.
[10] D. Hock *et al.*, "POCO-framework for Pareto-optimal resilient controller placement in SDN-based core networks," in *Network Operations and Management Symposium (NOMS), 2014 IEEE*. IEEE, 2014.
[11] X. Huang *et al.*, "Dynamic Switch-Controller Association and Control Devolution for SDN Systems," *arXiv preprint arXiv:1702.03065*, 2017.
[12] National Institute of Standards and Technology, "FIPS 180-2 with change notice, "Secure Hash Standard"," 2004.
[13] T. Distler *et al.*, "Resource-efficient Byzantine fault tolerance," *IEEE Transactions on Computers*, vol. 65, no. 9, 2016.
[14] J. Liu *et al.*, "Scalable Byzantine Consensus via Hardware-assisted Secret Sharing," *IEEE Transactions on Computers*, 2018.
[15] R. Kotla *et al.*, "Zyzzyva: speculative byzantine fault tolerance," *ACM SIGOPS Operating Systems Review*, vol. 41, no. 6, 2007.
[16] P. Mohan *et al.*, "Primary-Backup Controller Mapping for Byzantine Fault Tolerance in Software Defined Networks," in *GLOBECOM 2017-2017 IEEE Global Communications Conference*. IEEE, 2017.
[17] M. Eischer *et al.*, "Scalable Byzantine Fault Tolerance on Heterogeneous Servers," in *Dependable Computing Conference (EDCC), 2017 13th European*. IEEE, 2017.