Optimal state replication in stateful data planes

Abubakar Siddique Muqaddas, German Sviridov, Paolo Giaccone, Andrea Bianco

Politecnico di Torino, Torino, Italy

Abstract

In SDN stateful data planes, switches can execute algorithms to process traffic based on local states. This approach permits to offload decisions from the controller to the switches, thus reducing the latency when reacting to network events. We consider distributed network applications that process traffic at each switch based on local replicas of network-wide states. Replicating a state across multiple switches poses many challenges, because the number of state replicas and their placement affects both the data traffic distribution and the amount of synchronization traffic among the replicas.

In this paper, we formulate the optimal placement problem for replicated states, taking into account the data traffic routing, to ensure that traffic flows are properly managed by network applications, and the synchronization traffic between replicas, to ensure state coherence. Due to the high complexity required to find the optimal solution, we also propose an approximated algorithm to scale to large network instances. We numerically show that this algorithm, despite its simplicity, well approximates the optimal solution. We also show the beneficial effects of state replication with respect to the single-replica scenario, so far considered in the literature. Finally, we provide an asymptotic analysis to find the optimal number of replicas.

Index Terms

Software Defined Networking (SDN), Stateful data planes, State replication.

I. Introduction

In recent years a major shift of paradigm has been observed in the field of SDN with the introduction of stateful data planes, which address the performance limitations of a complete centralization of the control plane in a canonical SDN architecture, as highlighted in [1], [2]. Indeed, stateful switches, as described for example in [3], [4], can be programmed to execute user-defined code during packet processing, operating on local state variables stored in persistent memories. Thus, stateful data planes provide an additional level of programmability with respect to canonical SDN, whose data plane is instead stateless, according to the original paradigm.

Indeed, stateful switches can take local decisions without relying on the intervention of an SDN controller [5]. This fact has many beneficial effects. First, it greatly improves the reactivity of network applications by reducing the communication and latency overhead due to the interaction with the controller. Second, it reduces the computational burden of the controller to sustain the correct network behavior [6]. Finally, the availability of state variables enables the definition of new fine-grained networking applications [7], as decisions can now be taken on a per-packet basis, contrary to the per-flow basis of canonical SDN.

The availability of local state variables (simply denoted as "states" in the remainder of the paper) and the capability to run local programs (i.e., finite state machines) based on such states open a new perspective, since distributed algorithms can be devised to run in the switches across the network. This permits to extend the scalability of many network applications, thanks to the distributed nature of the approach.

Differently from previous works, we focus on the specific scenario in which the network application runs locally in stateful switches on the basis of some non-local states. Indeed, for applications implementing network-wide policies, the value of a state may be "global" across multiple switches, each switch holding a local replica of the state. Recent works, as [8], [9], have shown the practical feasibility of this approach by leveraging available programmable data planes, such as P4 [3] and Open Packet Processor (OPP) [4].

When a given state is replicated across multiple switches, two fundamental and coupled questions must be addressed: i) How many replicas are needed? ii) In which switches should replicas be placed? To find an optimal solution, several issues should be addressed. First, all traffic flows must traverse at least one switch that holds a state affecting (or affected by) the flow. However, routing a flow possibly not along its shortest path increases the data traffic load on the network. Thus, from the point of view of the data traffic, it would be convenient to increase the number of replicas until at least one replica is present along the shortest path of each flow. At the same time, adopting replicas comes at the cost of keeping the replicas synchronized. This requires the interaction between switches holding the replicas, thus introducing a synchronization traffic, which increases with the number of replicas. This traffic affects the overall offered load on the network. Thus, from this perspective, it would be convenient to reduce the number of replicas as much as possible. In summary, the optimal selection of the number of replicas and their location depends on the tradeoff between the load introduced in the network by data and synchronization traffic.

In this paper, we address all the above mentioned questions and provide the following contributions:

- we propose the optimal state replication problem and formalize it as an ILP problem, that minimizes the overall (i.e., data plus synchronization) traffic load;
- to cope with the limited scalability of the ILP solver, we propose an approximation algorithm, denoted as PLACEMULTIREPLICAS (PMR), able to solve large instances of the problem;
- we numerically evaluate the performance of PMR and show that it well approximates the optimal solution, at least for small instances of the problem. Furthermore, we show that adding few replicas in a network can largely improve the performance with respect to the single-replica scenario;
- we analytically find the optimal number of replicas for unwrapped Manhattan network topologies and characterize its asymptotic behavior; we show that the formula obtained for large networks can be used also for small instances of the network.

The remainder of the paper is organized as follows. In Sec. II, we describe the state replication problem. In Sec. III, we present the ILP formalization of the optimal state replication problem. In Sec. IV, we propose the PMR algorithm. In Sec. V we show the numerical results for the state placement problem. In Sec. VI, we present the asymptotic analysis of the optimal number of replicas in a network. In Sec. VII we discuss the related works. Finally, we draw our conclusions in Sec. VIII.

II. STATE REPLICATION IN STATEFUL SDN

Following the increasing need for highly dynamic network services and policies, the introduction of programmable data planes enables traffic processing policies to be offloaded directly into the switches. New frameworks to embed user-defined network policies to the stateful switches have been proposed [10], [11]. In this paper, we consider SNAP [10] as a reference framework, even if our proposed approach is general and relevant to any programming abstractions for stateful data planes.

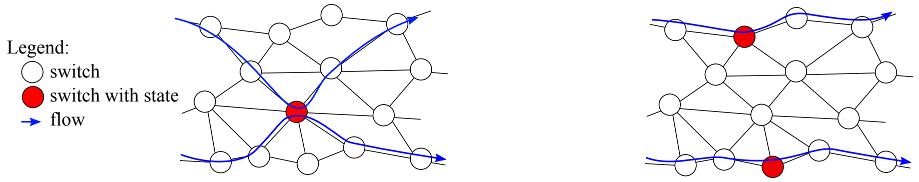
SNAP introduces a one-big-switch (OBS) model as a network abstraction: the whole network (switches and links) is seen as a single “big” switch with a given set of input and output ports, corresponding to the end hosts, and an aggregate list of available resources for traffic processing. Due to the way the OBS abstraction is defined, flow routing between hosts is described on the basis of I/O port pairs. When defining a network application, the programmer is exposed to the OBS abstraction, without any knowledge of the actual underlying composition of the network. The network applications are decomposed by SNAP into an extension of forward decision diagram (xFFD) that incorporates also stateful processing elements available at switches. The placement of the single-replica state affects the application and network performance. Indeed, the xFFD and the traffic matrix between the OBS ports are fed into the SNAP ILP (Integer Linear Programming) optimizer, which selects the switches where to place each state and the corresponding processing logic of the decomposed application. The order in which the traffic traverses the switches storing the states plays a fundamental role, as state dependencies must be preserved to correctly execute the xFDD of the original application. To guarantee the correct execution of a network application, all flows affected by or affecting a state must be routed across the switch storing it. Thus, the routing does not generally follow the shortest path between the input and output OBS port, and the SNAP solver jointly optimizes the placement of the states and the routing to minimize the total data traffic load in the network.

The main limitation of SNAP emerges from the fact that it permits only one replica for each state. This considerably restrains the flow routing, thus precluding a wide range of optimization techniques such as load balancing and traffic engineering.

A. State replication

To cope with the above mentioned SNAP limitations, we consider a scenario in which states are replicated on stateful switches. We address the optimal placement of the replicas of each state, given the knowledge of the traffic demands and of the xFDD defining the network application.

As a toy example, consider a network-wide application that acts on a global counter (e.g., the total traffic entering/leaving the network), which is obviously affected by all flows in the network. SNAP would place a single replica of the state associated with the global counter in a single switch in the topology, likely into the switch in the most "central" position (i.e., with the highest betweenness centrality) in the network topology, as shown in Fig. 1a. As a consequence, all flows are forced to be routed through the single switch storing the state. Due to the "hot-spot" routing, the set of feasible solutions for the capacitated routing problem is significantly reduced. Instead, replicating the global state on multiple switches would lead to a better network utilization, as shown in Fig. 1b, and to a much larger set of feasible routing solutions, with a



(a) Single-replica approach

(b) Multi-replica approach

Fig. 1: Example of routing for single-replica (e.g., SNAP) and multi-replica state placement.

beneficial effect on the maximum amount traffic that can be sustained in the network and/or on the experienced delays.

The choice of an appropriate synchronization mechanism is crucial for network performance and for the implementation complexity of the replication scheme. Notably, the CAP theorem [12] states that for a replication scheme, only two properties can be picked at the same time out of Consistency, Availability and Partition tolerance. Considering that network failures may occur, partition tolerance cannot be left out of the design of our replication algorithm, leaving us with the following, well-known, reference models:

a) Strong consistency: A replication algorithm based on strong consistency privileges consistency over availability. This translates into strong guarantees that the same value of a state will be read across all replicas, at the cost of higher delays to access and update the states. The delay penalty is caused by the adopted protocol (e.g., Paxos [13], Raft [14]) requiring intensive interaction among the replicas whenever a read or write transaction is executed. Side effects of the replication protocol are the high overhead in terms of synchronization traffic and its high complexity, typically incompatible with the limited amount of hardware resources available at the switches. Furthermore, the latency due to the communication between replicas requires buffering packets at each switch while waiting for the outcome of the replication transaction. This further makes the scheme too complex to be adopted in practice in high speed networks.

b) Eventual consistency: Replication schemes based on eventual consistency prioritize replicas availability over their consistency. This translates into low latencies during the execution of transactions at the cost of no guarantees on the consistency of the actual values of each replica. Most of eventual consistency algorithms are based on gossip protocols [15]–[17] which incur into small overhead in terms of synchronization traffic. At the same time, due to the simplicity of the adopted communication protocols, these algorithms can be implemented in programmable switches.

Due to the implementation and performance issues highlighted for strong consistency schemes, we assume a replication scheme based on eventual consistency, according to which each replica generates a fixed amount of synchronization traffic towards all the other replicas. As shown in [8], this scheme can be implemented in current state-of-art programmable data plane and, in practice, maintains small errors among the values of the replicas.

III. Optimal state replication problem

Given a network graph, the objective of the state replication problem is to identify the best set of nodes (i.e., switches) where to place the replicas of each state and to compute the optimal routing. Coherently with [10], the nodes are selected to minimize the overall traffic in the network and to guarantee that all flows affecting (or affected by) a given state will traverse at least one state replica. Differently from [10], the traffic in the network is composed not only of data traffic, but also of the traffic introduced by the synchronization protocol required to keep consistent the replicas of a given state.

We propose an integer linear program (ILP) formalization, as in the original SNAP model [10]. The relevant notation is reported in Tab. I. Our formalization takes the following input parameters:

- *Network.* Let $G = (V, E)$ be the network graph with N nodes. Let c_e be the capacity of edge $e \in E$.
- *Traffic flows.* Let $\mathcal{F}$ be the set of all flows. The traffic demands are assumed to be known in advance. In particular: let λ_f be the demand of traffic flow $f \in \mathcal{F}$, being $f_s \in V$ and $f_d \neq f_s \in V$ respectively the source and the destination nodes of the flow.
- *State variables.* Let S be the set of all state variables. Let $S_f \subseteq S$ be the *ordered* sequence of state variables for flow $f \in \mathcal{F}$, obtained from the xFFD of the corresponding application.
- *Maximum number of replicas.* Let C_s be a given upper bound on the number of replicas for a state variable s, chosen by the network designer. Note that the optimal number of replicas for state s, denoted by $\hat{C}_s$, will be computed while satisfying the constraint $\hat{C}_s \leq C_s$.

Let H_f be the set of all possible sequences of state replicas for a flow f. Consider a toy example in which a flow f requires 3 state variables $\mathcal{A}$, $\mathcal{B}$, $\mathcal{C}$, i.e., $S_f = [\mathcal{A}, \mathcal{B}, \mathcal{C}]$. Each state has 2 replicas (denoted as “1” and “2”). Now $H_f = \{[111], [112], [121], [122], [211], [212], [221], [222]\}$, and, as example, the sequence $h = [121]$ implies that f traverses replica 1 of state $\mathcal{A}$, then replica 2 of state $\mathcal{B}$, and finally replica 1 of state $\mathcal{C}$. Let h_s be the replica of state variable s in sequence $h \in H_f$. For the above example with $h = [121]$, $h_{\mathcal{A}} = 1$, $h_{\mathcal{B}} = 2$ and $h_{\mathcal{C}} = 1$.

The output of the solver is described as follows, and the relevant notation is reported in Tab. II:

- *Placement of the replicas of each state.* Let P_{scn} be a binary variable equal to 1 iff replica c of state s is stored at node n. Note that the optimization problem might place multiple replicas on the same node, but this would correspond to a single instance of the state. Thus, the optimal number of distinct replicas $\hat{C}_s$ of state s across the whole network can be computed as follows[1]:
$$\hat{C}_s = \sum_{n \in V} \mathbb{1}_{\Big\{ \sum_{c \leq C_s} P_{scn} > 0 \Big\}}$$
- *Data traffic routing.* Let R_{fhe} be a binary variable equal to 1 iff flow f traverses the sequence of state replicas h on edge e. The set of such variables describes the complete routing of all flows in the network, taking also into account the constraint for the required sequence of traversed replicas. To avoid out-of-sequence problems, we do not permit flow splitting between different sequences of replicas.
- *Synchronization traffic routing.* Let $\hat{R}_{snme}$ be a binary variable equal to 1 iff there are replicas of the state variable s on nodes n and m and the flow from node n to node m traverses edge e. This set of variables describes the routing of the synchronization traffic between different replicas of the same state. Let $\hat{\lambda}_s$ be the traffic generated by each state replica to update each other single replica of the same state.

Finally, Tab. III reports the list of auxiliary variables adopted in the ILP formalization.

In the optimal state replication problem, the total traffic in the whole network is minimized:

$$\min \sum_{e \in E} \sum_{f \in \mathcal{F}} \sum_{h \in H_f} R_{fhe} \lambda_f + \sum_{e \in E} \sum_{s \in S} \sum_{n \in V} \sum_{\substack{m \in V \\ n \neq m}} \hat{R}_{snme} \hat{\lambda}_s \qquad (1)$$

The first term represents the total data traffic in the network. It is obtained by summing all the traffic due to f on all the possible sequences of state replicas and on all of the edges. Instead, the second term is the synchronization traffic between replicas of the same state, summed across all states and edges in the graph. Notably, (1) is similar to the objective function used by the SNAP framework in [10], but with the introduction of the second term that takes into account the synchronization traffic, not included in SNAP.

[1] Let $\mathbb{1}_{\{A\}}$ be the indicator function of A, equal to 1 iff condition A is true.

TABLE I: Input variables

Context	Variable	Description	Range
Network definition	V	set of all nodes	$\{1, \ldots, N\}$
	N	number of nodes (i.e., $\lvert V \rvert$)	$\mathbb{N}$
	E	set of all edges	
	c_e	capacity of edge $e \in E$	> 0
Flow definition	$\mathcal{F}$	set of all the flows	
	λ_f	traffic demand for flow $f \in \mathcal{F}$	> 0
	f_s	source node for flow $f \in \mathcal{F}$	$1, \ldots, N$
	f_d	destination node for flow $f \in \mathcal{F}$	$1, \ldots, N$
State definition	S	set of all state variables	
	C_s	max number of replicas for state s	≥ 1
	S_f	sequence of state variables for flow $f \in \mathcal{F}$	$\subseteq S$
	$\hat{\lambda}_s$	synchronization traffic between any pair of replicas for state $s \in S$	> 0

TABLE II: Output variables

Context	Variable	Description	Range
Data traffic routing	R_{fhe}	1 iff flow f along sequence of replicas h traverses edge e	Binary
Synchronization traffic routing	$\hat{R}_{snme}$	1 iff synchronization traffic from node n to node m containing replicas of state variable s traverses edge e	Binary
Replica placement	P_{scn}	1 iff replica c of state s is stored in node n	Binary

TABLE III: Auxiliary Variables

Variable	Description	Range
$E_I(n)$	set of edges entering node $n \in V$	$\subseteq E$
$E_O(n)$	set of edges leaving node $n \in V$	$\subseteq E$
$E(n)$	set of all edges incident to node $n \in V$	$\subseteq E$
H_f	set of all sequences of replicas for flow $f \in \mathcal{F}$	-
h_s	replica id of state s for flow $f \in \mathcal{F}$ in sequence $h \in H_f$	$1, \ldots, C_s$
P_{fsce}	1 iff flow f on edge e has passed replica c of state s	Binary
X_{fh}	1 iff flow f is assigned $h \in H_f$	Binary
U_{sn}	1 iff at least one replica of state variable s is on node n	Binary
Y_{snme}	1 iff $\hat{R}_{snme} > 0$	Binary

As an alternative, the objective function could be modified to minimize the maximum congestion on a link, obtained by summing data and synchronization traffic, as follows:

$$\min \max_{e \in E} \Big(\sum_{f \in \mathcal{F}} \sum_{h \in H_f} R_{fhe} \lambda_f + \sum_{s \in S} \sum_{n \in V} \sum_{\substack{m \in V \\ n \neq m}} \hat{R}_{snme} \hat{\lambda}_s \Big) \tag{2}$$

and could be easily integrated in the following formalization, using well-known ILP modeling techniques.

A. *Constraints in the optimization problem*

We now discuss all the constraints considered in the ILP model. In some cases, we will get products of binary variables, but the corresponding constraint can be easily linearized according to well-known techniques.

1) Data routing constraints: Constraints (4)-(7) are similar to the constraints for the classic multi-commodity flow problem. However, our modification consists of assigning a commodity for each sequence $h \in H_f$ of state variable replicas directly at the source of the flow f, to model the sequence of states required by each flow.

We introduce an auxiliary variable, which is an indicator function X_{fh} equal to 1 if sequence $h \in H_f$ is assigned to flow $f \in \mathcal{F}$.

$$X_{fh} = \sum_{e \in E_O(f_s)} R_{fhe} - \sum_{e \in E_I(f_s)} R_{fhe} \tag{3}$$

Indeed, whenever a particular sequence h is adopted, similar to (4), the net outgoing data traffic from source f_s is 1. Notably, the second term considers the special case in which the flow is re-entering (and leaving) f_s in the path to reach the state and then the destination. We now force only one sequence h to be assigned to flow f. $\forall f \in \mathcal{F}$:

$$\sum_{h \in H_f} X_{fh} = 1 \tag{4}$$

A similar constraint is defined for flow f's destination f_d, but now the net incoming flow should be 1. $\forall f \in \mathcal{F}$:

$$\sum_{h \in H_f} \Big(\sum_{e \in E_I(f_d)} R_{fhe} - \sum_{e \in E_O(f_d)} R_{fhe} \Big) = 1 \tag{5}$$

The sum of all the data and synchronization traffic passing an edge must not exceed its capacity. $\forall e \in E$:

$$\sum_{f \in \mathcal{F}} \sum_{h \in H_f} R_{fhe} \lambda_f + \sum_{s \in S} \sum_{n \in V} \sum_{\substack{n^* \in V \\ n \neq n^*}} \hat{R}_{snn^*e} \hat{\lambda}_s \leq c_e \tag{6}$$

Finally, the standard flow conservation condition must be satisfied at any node. $\forall h \in H_f, \forall f \in \mathcal{F}$:

$$\sum_{e \in E_I(n)} R_{fhe} = \sum_{e \in E_O(n)} R_{fhe} \quad \forall n \in V \setminus \{f_s, f_d\} \tag{7}$$

2) Placement constraints: Each replica can only be placed at one switch. $\forall s \in S,\ \forall c \leq C_s$:

$$\sum_{n \in V} P_{scn} = 1 \tag{8}$$

We now constrain the flows to be routed through the corresponding states, i.e., all flows dependent on a state must traverse the node where the replica of such state is located (except at source f_s and destination f_d). $\forall n \in V \setminus \{f_s, f_d\}, \forall f \in \mathcal{F}, \forall h \in H_f, \forall s \in S_f$:

$$\sum_{e \in E_I(n)} R_{fhe} \geq P_{sh_sn} + X_{fh} - 1 \tag{9}$$

Indeed, if a particular sequence h is adopted for f, then (9) becomes $\sum_{e \in E_I(n)} R_{fhe} \geq P_{sh_sn}$ and in the case the node contains a replica h_s of the state s, then $\sum_{e \in E_I(n)} R_{fhe} \geq 1$, which forces at least one R_{fhe} variable to be one on the incoming edges to e. Otherwise, if the sequence h is not adopted for f, then (9) becomes a useless bound.

We now define a variable that tracks the fact that a flow has already traversed a particular state along its path. For a flow f traversing a replica h_s of state s, we define $P_{fsh_se} = 0$ for all edges along the path before entering the node with replica h_s of s, and $P_{fsh_se} = 1$ for all edges on the path after h_s. It is initialized to zero for all unused replica sequences h. $\forall f \in \mathcal{F}, \forall s \in S_f, \forall h \in H_f, \forall e \in E$:

$$P_{fsh_se} \leq R_{fhe} \tag{10}$$

To model the fact that P_{fsh_se} changes from 0 to 1 whenever the flow leaves a node where the state is stored, we set: $\forall f \in \ \mathcal{F}, \forall s \in S_f, \forall h \in H_f, \forall e \in E, \forall n \in V \setminus \{f_s, f_d\}$:

$$P_{sh_sn} X_{fh} + \sum_{e \in E_I(n)} P_{fsh_se} = \sum_{e \in E_O(n)} P_{fsh_se} \tag{11}$$

Indeed, only when $P_{sh_sn} X_{fh} = 1$ (i.e., node n has replica h_s and f exploits h including it), the net flow of P_{fsh_se} entering n is 0 and the corresponding one leaving n is 1.

We now impose that the data flow reaches the destination f_d after having traversed all the states required in h, i.e. $P_{fsh_se} = 1$ for one edge entering f_d. $\forall f \in \mathcal{F}, \forall s \in S_f, \forall h \in H_f$:

$$P_{sh_sf_d} X_{fh} + \sum_{e \in E_I(f_d)} P_{fsh_se} = X_{fh} \tag{12}$$

So far, the constraints (10)-(12) force the flows to pass through all the required state variables, but not necessarily in sequence. We model here the correct sequence of traversed states, if the flow f has to cross $h_s \in H_f$ of s, followed by replica $h_{s'} \in H_f$ of s'. $\forall f \in \mathcal{F}, \forall s, s' \in S_f, \forall h \in H_f, \forall n \in V$

$$P_{sh_sn} + \sum_{e \in E_I(n)} P_{fsh_se} \geq P_{s'h_{s'}n} + X_{fh} - 1 \tag{13}$$

Indeed, if either flow f has been assigned sequence h, i.e., $X_{fh} = 1$, or replica $h_{s'} \in H_f$ exists at node n, or replica $h_s \in H_f$ does not exist at node n, then (13) becomes $\sum_{e \in E_I(n)} P_{fsh_se} \geq 1$. This forces P_{fsh_se} to be 1 before entering node n, which means that the flow must have traversed h_s before entering the node containing $h_{s'}$. This ensures that the flow traverses the correct sequence of states as dictated by h.

Constraint (14) ensures that if flow has traversed state variable replica h_s on edge e, i.e., $P_{fs'h_{s'}e} = 1$, then it must have already crossed state variable replica h_s, which ensures $P_{fsh_se} = 1$. $\forall f \in \mathcal{F}, \forall s, s' \in S_f, \forall h \in H_f, e \in E$:

$$P_{fsh_se} \geq P_{fs'h_{s'}e} \tag{14}$$

3) State synchronization: State synchronization implies the generation of synchronization traffic between any pair of replicas of the same state. Thanks to the routing variable $\hat{R}_{snme}$, we can model the traffic between any pair of nodes n and m containing replicas of the state variable s and consider its contribution in the total traffic, as in (1) and (2), and in the constraint (6) regarding the edge capacity.

In the optimization model, multiple replicas of the state variable can be hosted on the same node n. Hence, to track that there is at least one replica at node n, we define the variable U_{sn} in (15). $\forall c \in C_s$, $\forall s \in S$, $\forall n \in V$:

$$U_{sn} \geq P_{scn} \tag{15}$$

For the synchronization traffic from node n to node m, the routing variable $\hat{R}_{snme}$ is treated as a commodity from node n such that $U_{sn} = 1$ to node m such that $U_{sm} = 1$. We constrain the routing to ensure the standard flow conservation equation at the intermediate node.

We define a new intermediate variable Y_{snme}, set to 1 iff $\hat{R}_{snme} > 0$. This is ensured using the big-M method [18] as in (16) where M is sufficiently larger than $\hat{R}_{snme}$. $\forall s \in S$, $\forall n \in V$, $\forall m \neq n \in V$, $\forall e \in E$

$$0 \leq -\hat{R}_{snme} + MY_{snme} \leq M - 1 \tag{16}$$

To fix a large enough value for M, assume $\hat{R}_{snme} = 1$, $\forall e \in E_O(n)$, then $Y_{smne} = 1$ from (16). In this case, for the condition $M \geq \hat{R}_{snme}$ to be true, M must be equal to or greater than the maximum degree of G:

$$M \geq \Delta_G \tag{17}$$

with $\Delta_G = \max_{n \in V} |E_O(n)|$.

We require the egress synchronization flow from a state replica containing node to use only one outgoing edge. This can be done by exploiting Y_{snme} as in (18). $\forall s \in S$, $\forall n \in V$, $\forall m \neq n \in V$:

$$\sum_{e \in E_{O(n)}} Y_{snme} \leq 1 \tag{18}$$

The following constraints (19)-(22) model the multi-commodity flow problem for the synchronization traffic. Specifically, constraints (19) and (20) are for the originating synchronization flow from the source node n and the sink flow in the destination node m containing the state replicas respectively. $\forall s \in S$, $\forall n \in V$, $\forall m \neq n \in V$:

$$\sum_{e \in E_{O(n)}} Y_{snme} \geq U_{sn} \tag{19}$$

$$\sum_{e \in E_{I(m)}} Y_{snme} \geq U_{sm} \tag{20}$$

Instead, constraints (21)-(22) are for the flow conservation at intermediate nodes. $\forall s \in S, \forall n \in V$, $\forall m \neq n \in V$:

$$\sum_{e \in E_{O(n)}} Y_{snme} \leq \sum_{e \in E_{I(n)}} Y_{snme} + U_{sn} \leq 1 \tag{21}$$

$$\sum_{e \in E_{I(n)}} Y_{snme} \leq \sum_{e \in E_{O(n)}} Y_{snme} + U_{sm} \leq 1 \tag{22}$$

B. Computational complexity

The complexity to solve an ILP model is $O(2^{2^{k_v+2}} k_c)$ [19], where k_v is the number of variables and k_c is the number of constraints. As a worst case, assume that all flows $f \in \mathcal{F}$ require to traverse all state variables $s \in S$, where each $s \in S$ has C replicas. In this case, it can be shown that $k_v = O(\max(N^2 C^{|S|}, |S|N^4))$ and $k_c = O(\max(N|S|C^{|S|}, |S|N^4))$. In a simple scenario when only one state variable required by all the flows, $k_v = O(N^4)$ and $k_c = O(N^4)$. Thus, the final complexity is lower bounded by $O(2^{2^{N^4+2}} N^4)$. Clearly, the presented ILP formalization does not scale for large instances of the problem. This advocates the design of approximation algorithms to solve the optimal replication problem in real scenarios, as addressed in the following section.

IV. Approximation algorithm for single state replication

We address specifically the problem of state replication for a single state variable. To address the limited scalability of the ILP solver, we propose PLACEMULTIREPLICAS (PMR) algorithm which is computationally scalable and will be shown in Sec. V to approximate well the optimal solution obtained by the ILP solver for small problem instances.

The pseudocode of PMR is given in Algorithm 1. It takes as input the network graph G, the state variable s and the maximum number of replicas C_s of s and the set of flows $\mathcal{F}$ requiring s. As output, the algorithm returns: the routing variables of the data flows R_{fhe} and of the state synchronization flows $\hat{R}_{smne}$ and the replicas placement variables P_{scn}. The algorithm works through 3 phases:

- *Phase 1.* The network graph G is partitioned into C_s clusters, in order to minimize the maximum distance among the elements within a cluster. This allows to distribute the replicas across the whole network in a balanced way, exploiting the spatial diversity offered by each cluster.
- *Phase 2.* In each cluster, a replica is placed in the "most central" node, i.e., the one with the highest betweenness centrality, in order to minimize the data traffic for each flow.
- *Phase 3.* The position of each replica is perturbed at random using a local search to improve the solution with respect to one obtained in the previous two phases.

Algorithm 1 comprises all the mentioned phases. After having initialized the routing and the replica placement variables (lines 2-4), Phase 1 is executed in line 5 by calling COMPUTEPARTITIONS. This method solves the k-means clustering problem [20] with $k = C_s$ using Lloyd's algorithm [21] in which the node with the highest betweenness centrality is chosen as center of the partition.

As part of Phase 2 (lines 6-9), within each subgraph G_c the node n' with the highest betweenness centrality is assigned a state variable replica through NODEWITHHIGHESTBC. As a reminder, betweenness centrality of a node v is proportional to the number of shortest paths crossing it.

Lines 11 to 18 refer to a local search procedure with I iterations. Within each iteration, ROUTEFLOWS is used to route flows through the location of the replicas identified in Phase 2, following two sub-paths: one from the flow source node to the closest replica and one from this replica to the destination node. The procedure works on the set of flows $\mathcal{F}$ and the location of state variables P_{scn} and returns the routing variables for data flows R'_{fce} and for state synchronization $\hat{R}'_{smne}$, and the corresponding total traffic T' in the network. Lines 23 to 39 route the data flows from their source f_s to the destination f_d while traversing the replica c_b which has the minimum path length among all other replicas. For each flow, in lines 25 and 26, the replica c_b and the path $\mathcal{P}_{best}$ traversing it are initialized. Then for each replica (in lines 27-34), first, the shortest path $f_s \rightarrow n_c \rightarrow f_d$ is computed. n_c is the vertex for which $P_{scn} = 1$. If the path length $\mathcal{P}$.length is less than the previous minimum minDist in line 29, then the current path $\mathcal{P}$ is stored as the best path $\mathcal{P}_{best}$ and the current replica c as the best replica c_b. In lines 35-38, for each edge in $\mathcal{P}_{best}$, the routing as well as the traffic value is updated. Lines 40 to 48 generate flows from each state replica c to all the other state replicas g for state synchronization using the shortest path. This includes the synchronization flows $\hat{R}_{scge}$ being updated in line 44 for each edge in the path $\mathcal{P}_{cg}$ before updating the total traffic in line 45. If T' is less than the previous minimum, then the minimum traffic value and all the decision variables are updated (lines 14-15). In Phase 3 (line 17), a local search procedure perturbs the existing state replica locations. This proceeds by randomly selecting one node where a replica is located and moving it to one of its neighbor nodes. This new solution is then compared with the current one (line 13) after having evaluated the corresponding routing and total traffic.

Algorithm 1 PlaceMultiReplicas (PMR)

```
procedure $[\{R_{fhe}\}, \{\hat{R}_{smne}\}, \{P_{scn}\}]$ = PLACEMULTIREPLICAS($G$, $s$, $C_s$, $\mathcal{F}$)
    $R_{fhe} = 0, \forall f \in \mathcal{F}, h \in H_f, \forall e \in E$                                ▷ Init routing
    $\hat{R}_{smne} = 0, \forall c, g \neq c \leq C_s, \forall e \in E$                                ▷ Init state sync
    $P_{scn} = 0, \forall c \leq C_s, \forall n \in V$                                ▷ Init state $s$ location
    $\{G_c\} \leftarrow$ COMPUTEPARTITIONS($G, C_s,$)                                ▷ **Phase 1:** Graph partitions $\{G_c\}$
    for $c \leq C_s$ do                                ▷ **Phase 2:** Replica placement
        $n' \leftarrow$ NODEWITHHIGHESTBC($G_c$)                                ▷ Find best candidate in partition $G_c$
        $P_{scn'} = 1$                                ▷ Store the state replica location
    end for
    $T_{\min} = \infty$                                ▷ Init minimum traffic
    for $I$ iteration do                                ▷ **Phase 3:** Local search
        $[T', \{R'_{fhe}\}, \{\hat{R}'_{smne}\}] \leftarrow$ ROUTEFLOWS($\mathcal{F}, \{P_{scn}\}$)                                ▷ Route flows through the replicas
        if $T' < T_{\min}$ then                                ▷ Check if the traffic is smaller
            $T_{\min} = T'$                                ▷ Store current best solution
            $R_{fhe} = R'_{fhe}$ $\hat{R}_{smne} = \hat{R}'_{smne}$, $P'_{scn} = P_{scn}$, $\forall f \in \mathcal{F}$, $\forall h \in H_f$, $\forall c, g \neq c \leq C_s$, $\forall e \in E$, $\forall n \in V$
        end if
        $\{P'_{scn}\} \leftarrow$ PERTURBREPLICALOCATION($\{P_{scn}\}$)                                ▷ Change existing location of state replicas
    end for
return $[\{R_{fhe}\}, \{\hat{R}_{smne}\}, \{P_{scn}\}]$
end procedure

procedure $[T_{\text{CURRENT}}, R'_{fce}, \hat{R}'_{smne}]$ = ROUTEFLOWS($\mathcal{F}, P_{scn}$)
    $T_{\text{current}} = 0$                                ▷ Init total traffic
    for $f \in \mathcal{F}$ do                                ▷ For each flow
        minDist $= \infty$                                ▷ Init minimum distance
        $c_b \leftarrow$ null                                ▷ Init best replica for current flow
        $\mathcal{P}_{best} \leftarrow$ null                                ▷ Path with minimum length for $f_s \rightarrow n_c \rightarrow f_d$
        for $c \in C_s$ do                                ▷ For all state replicas
            $\mathcal{P} =$ SHORTESTPATH($f_s, n_c$) $\cup$ SHORTESTPATH($n_c, f_d$)
            if $\mathcal{P}$.length $<$ minDist then
                minDist $= \mathcal{P}$.length                                ▷ Update minimum distance
                $\mathcal{P}_{best} \leftarrow \mathcal{P}$                                ▷ Store path with minimum length
                $c_b \leftarrow c$                                ▷ Store best replica for this flow
            end if
        end for
        for $e \in \mathcal{P}_{best}$ do                                ▷ For each edge in the minimum length path
            $R'_{fc_be} = R'_{fc_be} + \lambda_f$                                ▷ Store the routing
            $T_{\text{current}} = T_{\text{current}} + \lambda_f$                                ▷ Store the traffic value
        end for
    end for
    for $c \in C_s$ do                                ▷ For each c$^{\text{th}}$ replica of state variable $s$
        for $g \neq c \in C_s$ do                                ▷ For each g$^{\text{th}}$ replica of state variable $s$
            $\mathcal{P}_{cg} \leftarrow$ SHORTESTPATH($n_c, n_g$)                                ▷ Shortest path from $n_c \rightarrow n_g$
            for $e \in \mathcal{P}_{cg}$ do                                ▷ For each edge in the path $n_c \rightarrow n_g$
                $\hat{R}_{smne} = \hat{R}_{smne} + \alpha$                                ▷ Store the state sync flow
                $T_{\text{current}} = T_{\text{current}} + \alpha$                                ▷ Update total traffic
            end for
        end for
    end for
return $[T_{\text{current}}, R'_{fce}, \hat{R}'_{smne}]$
end procedure
```

V. Performance comparison

We evaluate the performance of PMR presented in Sec. IV. The local search in PMR runs with $I = 1000$ iterations. In the case of small instances of the problem, we run an ILP solver, coded using IBM CPLEX optimizer [22], implementing the optimization model in Sec. III. Notably, whenever the number of replicas is set to 1, $\hat{\lambda}_s = 0$ and the solver obtains a solution equivalent to the one achieved by SNAP. We compute the *approximation ratio*, i.e., the ratio between the total traffic obtained by PMR and the optimal traffic obtained by the ILP solver. We consider two standard topologies for the network graph:

- *Unwrapped Manhattan* is a $\sqrt{N} \times \sqrt{N}$ grid.
- *Watts-Strogatz* [23] adds a few long-range links to regular graph topologies to reduce the distances between pairs of nodes and emulate a small-world model. It is generated by taking a ring of N nodes, where each node is connected to k nearest neighbors. In each node, the edge connected to its nearest clockwise neighbor is disconnected with probability p and connected to another node chosen uniformly at random over the entire ring. Thus, the final topology maintains the original average degree k while being connected. In the following, we will use $p = 0.1$ and $k = 8$.

We utilize random traffic matrices with the number of flows equal to the number of nodes in the graph ($|\mathcal{F}| = N$) and with unity demands ($\lambda_f = 1$). The source-destination pairs for the flows were generated according to two models. In the case of *uniform traffic*, all the source nodes were associated to a random permutation of nodes as destination; thus each node is source and destination of exactly one flow. In the case of *clustered uniform traffic*, we partitioned the nodes of the graph in half and generated a random permutation between the nodes of the same partition; thus all the flow are local within the same partition. All the results were obtained with 1000 different runs to get very small 95% confidence intervals (in all cases within 4.2% accuracy).

A. Synchronization traffic and optimal number of replicas

In Fig. 2 we evaluate the effect of varying the number of replicas for state s and of the synchronization rate $\hat{\lambda}_s$, through the optimal ILP solver. We consider a 4×4 Manhattan graph and set $C_s = 7$. As expected, when increasing the traffic required to synchronize the replicas ($\hat{\lambda}_s$), the optimal number of replicas reduces, since the higher costs of synchronization compensates

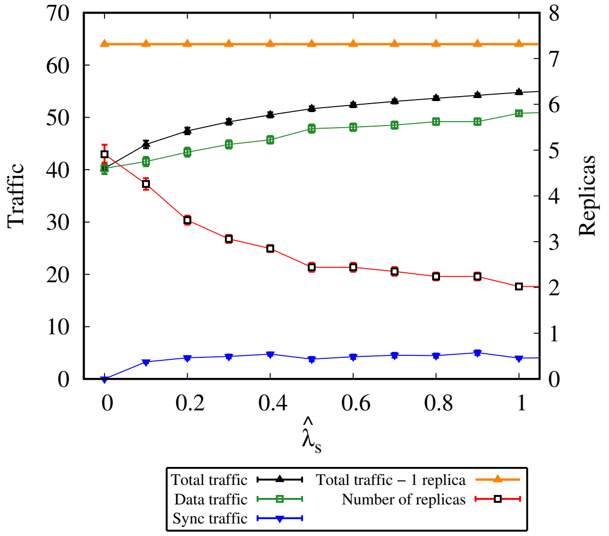


Fig. 2: Optimal traffic and number of replicas in a 4×4 Manhattan graph for uniform traffic, using the ILP solver.

the beneficial effect of multiple replicas on the data traffic. Instead the synchronization traffic is almost constant, since, for smaller number of replicas, their relative distances grows, to "cover" a larger area of the network. As a term of comparison, we report the total traffic for one single replica allowed in the network, equivalent to the solution obtained by SNAP.

Fig. 3 extends Fig. 2 for larger values of $\hat{\lambda}_s$. Due to the higher cost for synchronization, for $\hat{\lambda}_s \geq 6.1$, the optimal number of replicas becomes one, i.e., it is not anymore convenient to replicate states due to the high synchronization cost and the final solution is equivalent to the one achieved by SNAP.

B. Comparison of PMR with ILP

Figs. 4-5 show the approximation ratio for different number of nodes N, of replicas C_s and different values of $\hat{\lambda}_s$, under uniform traffic. The two graphs refer to Manhattan and Watts-Strogatz graphs, respectively. The approximation ratio in all cases is always ≤ 1.15, thus PMR approximates well the ILP solution. For larger graphs, we could not provide the results as the ILP solver is not computationally feasible.

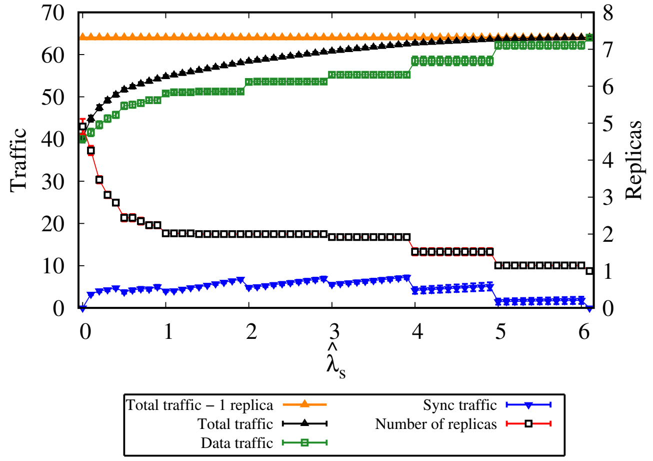


Fig. 3: Optimal traffic and number of replicas in a 4×4 Manhattan graph for uniform traffic, using the ILP solver, for large values of $\hat{\lambda}_s$.

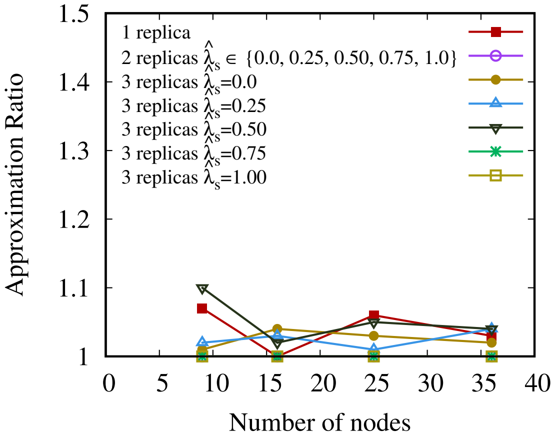


Fig. 4: Approximation ratio of PMR in a Manhattan graph under uniform traffic.

C. Number of replicas in large topologies

For large topologies, we run just the PMR algorithm. Figs. 6-7 show the total traffic, normalized by the number of flows, for Manhattan and Watts-Strogatz graphs, under clustered uniform traffic. We set $\hat{\lambda}_s = 0.5$. For comparison, we also report the result of the traffic obtained by routing each flow from its source to its destination along the shortest path, obliviously of the placement of the state replicas; this provides a lower bound on the total traffic in the network obtained for the optimal solution of the ILP problem (which cannot be computed in this case).

As expected, the highest amount of traffic is given by the single-replica case, because of the

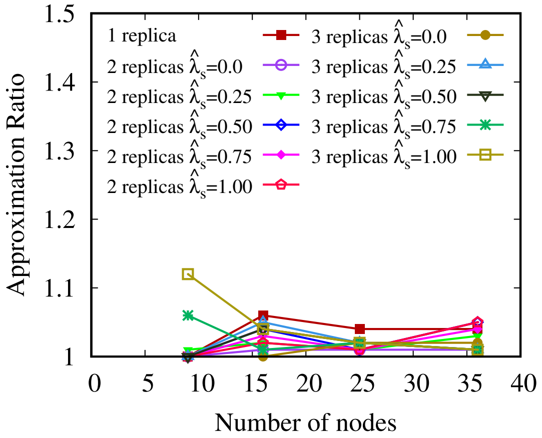


Fig. 5: Approximation ratio of PMR in Watts-Strogatz graph under uniform traffic.

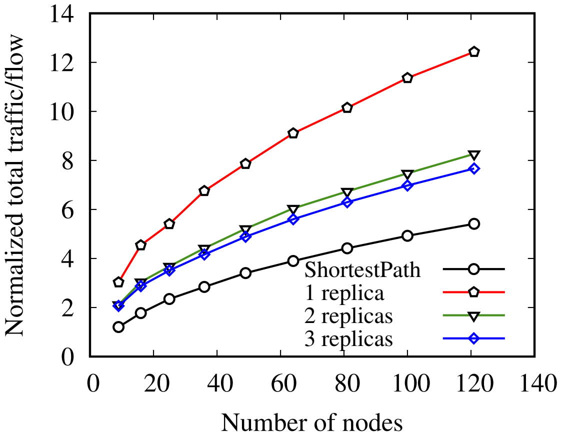


Fig. 6: Performance of PMR in Manhattan graph under clustered uniform traffic.

longer path to reach the state location targeted by all the flows. Now adding one replica provides a beneficial effect, since the spatial diversity of 2 replicas can be exploited to route the flows and minimize the total traffic. The gain is generally around 30% for Manhattan graph and grows up to 20% in Watts-Strogatz graph. If increasing again the number of replicas from 2 to 3, then the gain is very limited (around 5%), since the higher spatial diversity is compensated by a higher synchronization traffic. Thus, in general we can expect that allowing few replicas has a strong beneficial effects on the overall traffic with respect to the single-replica scenario.

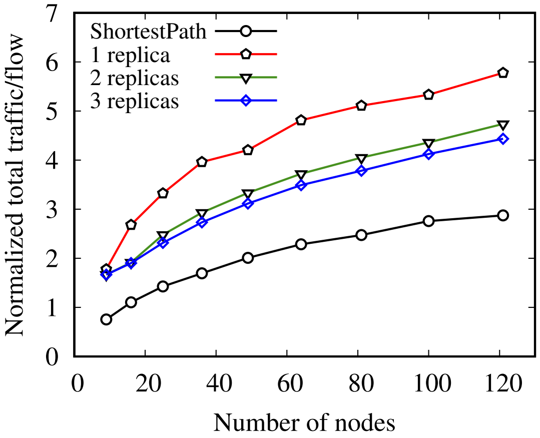


Fig. 7: Performance of PMR in Watts-Strogatz graph under clustered uniform traffic.

VI. Asymptotic analysis for number of replicas

We now present an asymptotic analysis, i.e., for very large network graphs, to estimate the optimal number of replicas. We will consider specifically an unwrapped Manhattan topology since amenable to analytical modeling. Furthermore, for simplicity we assume a single state.

A. Methodology

We consider a unit square as shown in Fig. 8, representing the boundary of an unwrapped Manhattan topology containing N nodes, with $N \to \infty$. Thus, any position within the unit square is associated to a network node, and any line within the unit square represents a routing path across a sequence of nodes in the original topology.

We now assume that the number of replicas C is a perfect square, i.e. $\sqrt{C} \in \mathbb{N}$. The unit square is divided into individual C squares, each of them of size $1/\sqrt{C} \times 1/\sqrt{C}$ and with a center point P_c^{ctr}, where $c \in \{1, \dots, C\}$ is an index identifying the square, as shown in Fig. 8. Here, P_c^{ctr} denotes the location of the c-th state replica in the network. We now evaluate the optimal number of replicas that minimizes the total traffic in the topology.

The total traffic is composed of the data traffic and the synchronization traffic, coherently with the cost function in (1). Consider now a given flow $f \in \mathcal{F}$. We assume that the traffic demand λ_f is routed in a straight line between two points in the square, since its approximates well the step-wise stair-like routing in the original Manhattan topology, for $N \to \infty$. The total traffic generated by the flow is $\lambda_f h$ where h is the corresponding distance of the routing path

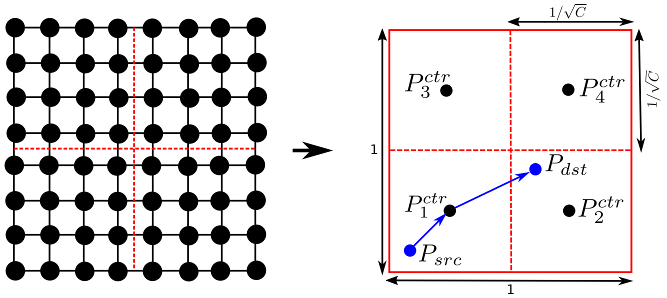


Fig. 8: Unwrapped Manhattan topology (left) and its unit square representation with 4 replicas ($C = 4$) (right).

in terms of hops in the Manhattan topology. The following bound can be easily shown, relating the distance d between two points in the unit square and the corresponding routing distance in terms of hops:

$$d\sqrt{N} \leq h \leq d\sqrt{2}\sqrt{N} \tag{23}$$

Now recall that a flow from a source node P_{src} to a destination node P_{dst} must traverse at least one replica P_c^{ctr}, as shown in Fig. 8, in order to affect (or being affected by) the state replica.

We start by evaluating the overall data traffic. We assume uniform traffic between any pair of nodes in the original topology, with a total number of flows equal to $|\mathcal{F}| = N$ and all flows with rate λ_f, coherently with Sec. V. Based on (23), we can define the average routing distance as:

$$\hat{h} = \hat{d}\sqrt{N}\beta \tag{24}$$

where β is a constant value less than $\sqrt{2}$. Thus, the overall data traffic generated in the network can be computed as the total generated data traffic $\lambda_f N$ times the average distance $\hat{h}$:

$$T_{data} = \lambda_f \hat{d}_{data} N\sqrt{N}\beta \tag{25}$$

where $\hat{d}_{data}$ is the average total distance between two randomly generated points in the unit graph passing through the closest replica.

To evaluate $\hat{d}_{data}$, we utilize a Monte Carlo method. We generate pairs of points with uniform random coordinates in the unit square, which are P_{src} and P_{dst} for source and destination nodes respectively, as in Fig. 8. Assume now the following case holds: the distance between P_{src}

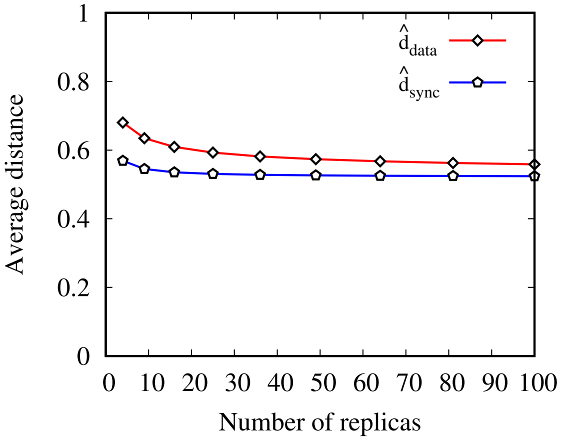


Fig. 9: The total average distance $\hat{d}_{data}$ and $\hat{d}_{sync}$ in function of the number of replicas C for a unit square.

and its closest replica P_c^{ctr} is smaller than between P_{dst} and its closest replica. Now the total distance between P_{src} and P_{dst} is computed by summing two terms: the distance from P_{src} to the closest replica P_c^{ctr}, and the one from such replica P_c^{ctr} to P_{dst}. If the considered case does not hold, the result is identical for symmetry. Fig. 9 shows the average total distance $\hat{d}_{data}$ obtained by randomly generating 10^7 pairs of nodes. When the number of replicas is large, $\hat{d}_{data}$ asymptotically approaches 0.5412 coherently with well-known theoretical results [24].

We now evaluate the overall synchronization traffic between the replicas, by knowing the pre-defined positions of the replicas in the unit square. The average distance between any two replicas $\hat{d}_{sync}$ asymptotically approaches 0.5221 as shown in Fig. 9. Thanks to (23), the synchronization traffic between the C replicas can be computed as follows:

$$T_{sync} = \hat{\lambda}_s \hat{d}_{sync} C(C-1)\sqrt{N}\beta \tag{26}$$

where the last term considers the pair-wise synchronization between replicas. Note that T_{sync} is independent from the data traffic.

Combining (25) and (26), we can finally claim:

Property 1: The total traffic for an unwrapped Manhattan topology of size N is given by:

$$T_{TOT} = \sqrt{N}\beta(\lambda_f N \hat{d}_{data} + \hat{\lambda}_s \hat{d}_{sync} C(C-1)) \tag{27}$$

where $\beta < \sqrt{2}$, and both $\hat{d}_{data}$ and $\hat{d}_{sync}$ depend on C as shown in Fig. 9.

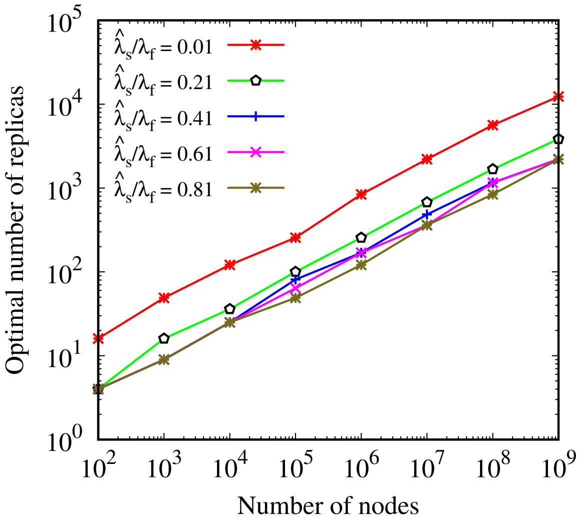


Fig. 10: Optimal number of replicas for different values of $\hat{\lambda}_s/\lambda_f$.

B. Optimal number of replicas and its approximation

We now evaluate numerically (27) and, through a dichotomic search, we find the optimal number of replicas that minimizes T_{TOT}. Fig. 10 shows the optimal number of replicas for different values of N and $\hat{\lambda}_s/\lambda_f$. Note that for higher values of N, more replicas are required to cover the network. For higher values of $\hat{\lambda}_s/\lambda_f$, the number of replicas decreases because of the higher cost in terms of synchronization traffic.

The curves in Fig. 10 can be fit by a function in the following form:

$$\log_{10} C_{opt} = x + y \log_{10} N + z \log_{10} \left(\frac{\hat{\lambda}_s}{\lambda_f} \right) \tag{28}$$

with x, y, z the fitting parameters. Using standard least-square fitting procedure, we numerically evaluated the best fitting parameters and obtained the following claim:

Property 2: The optimal number of replicas C_{opt} in an unwrapped Manhattan topology of size N can be approximated as follows

$$\bar{C}_{opt} = \left\lceil 0.47 N^{0.40} \left(\frac{\lambda_f}{\hat{\lambda}_s} \right)^{0.40} \right\rceil \tag{29}$$

which implies that $\bar{C}_{opt}$ grows as $\theta(N^{2/5})$.

Fig. 11 shows the optimal number of replicas $\bar{C}_{opt}$ obtained according to (29). As expected, if $\hat{\lambda}_s$ is small, then the number of replicas is large and for small networks correspond almost to one replica per node. For large values of synchronization traffic ($\hat{\lambda}_s = \lambda_f$), the number of replicas is

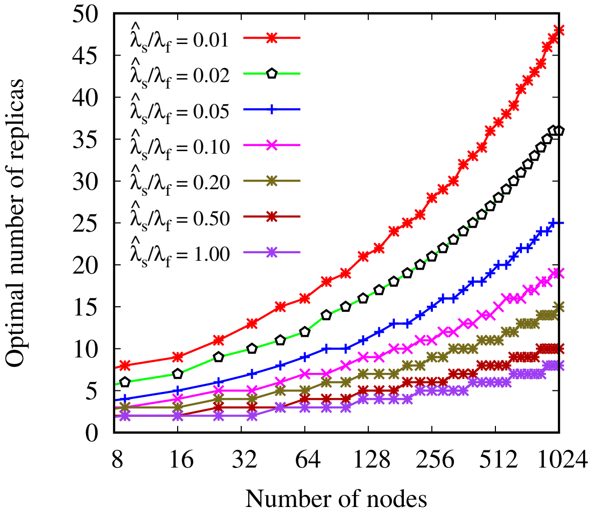


Fig. 11: Optimal number of replicas $\bar{C}_{opt}$ according to Property 2.

kept at the minimum, and 8 replicas are enough for networks with $N = 1024$ switches. We now evaluate the error introduced by Property 2. We evaluated (i) C_{opt} by solving the optimization problem described in Sec. III, (ii) $\bar{C}_{opt}$ by computing (29), and (iii) the optimal number of replicas C_{PMC} obtained by running PMR. We considered the same uniform traffic pattern described in Sec. V for the unwrapped Manhattan topology. All the results were obtained with 1000 different runs.

Fig. 12 shows the maximum error between $\bar{C}_{opt}$ and C_{opt} for N that varies between 9 and 36. In all cases, the maximum error is bounded by one, i.e., $\bar{C}_{opt}$ overestimates by at most one the optimal number of replicas. This result shows that the formula in (29) is also a good approximation for small Manhattan networks.

Due to scalability restraints we could not run the optimal solver to evaluate the error for larger networks. For this reason we had to refer to the optimal number of replicas obtained by PMR. Fig. 13 shows the error between $\bar{C}_{opt}$ and C_{PMC} for N varying between 9 and 121. Also in this case, the maximum error is bounded by one. Thus, the expression in (29) appears to be a reliable approximation even for larger unwrapped Manhattan topologies.

VII. Related works

The works in [8], [9] propose the programming abstractions to define network applications based on global states, as assumed in this work. Furthermore, they show the practical feasibility

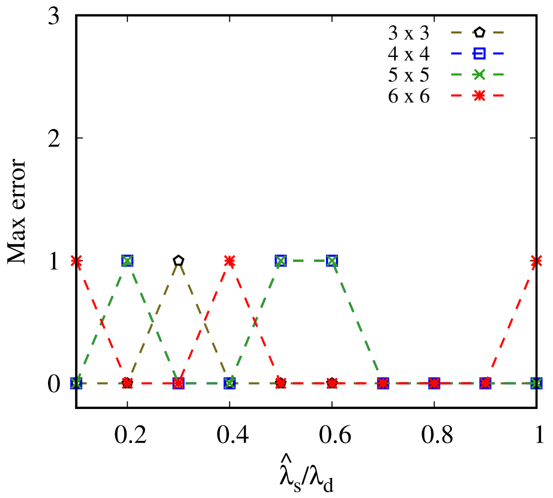


Fig. 12: Maximum error $\bar{C}_{opt} - C_{opt}$ in number of replicas between the approximated formula and the optimization model.

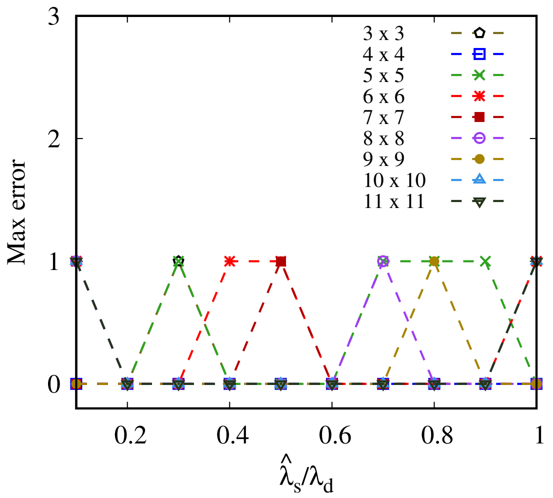


Fig. 13: Maximum error $\bar{C}_{opt} - C_{PMR}$ in number of replicas between the approximated formula and the one obtained with PMR.

of replicating the states by describing and testing an implementation based on programmable data planes, such as P4 [3] and Open Packet Processor (OPP) [4]. Both [8], [9] assume to know the number of replicas and their placement within the network, i.e., they need an optimization engine which solves the multi-replica placement problem addressed here. On the other end, our work needs a practical implementation scheme to support the state replication as described by the two cited papers. Thus, this work and [8], [9] are complementary.

Regarding the optimization problem addressed in this paper, the Virtual Network Embedding (VNE) problem finds the optimal placement of chains of VNFs under various optimization metrics. VNE can be closely mapped to the problem mentioned in this paper, if we consider network functions to be states and chains to be dependency graphs as computed by SNAP. Several ILP formulations and heuristics for VNE were proposed (an extensive survey is available in [25]), some of which are similar to the one proposed here. However, to the best of our knowledge, none of them consider the possibility of having replicated virtual functions, the peculiar feature of this work.

SNAP [10] solves the problem of the optimal placement of the states across network switches, taking into account the dependency between states and the traffic flows. However, by design, SNAP enables only one replica of each state within the network. This limits SNAP applicability, and may impair network performance, as discussed in Sec. II-A. To overcome this issues, we extend SNAP by enabling multiple replicas of the same state.

Several other network programming abstractions were proposed [26]–[28]. However, most of them keep the states at the controller, with few existing works exploiting stateful data planes to store states. NetKAT [11] focuses on stateful data planes and provides a native support for replicated states, but, by design, the replicas are placed at the network edge (i.e., entry and exit switches) for all flows. Thus, the placement is not optimized with respect to the traffic matrix. However, our methodology could be directly applied to NetKAT. Furthermore, the synchronization traffic is carried in piggybacking over the data traffic. Thus, both the synchronization and the data traffic must traverse all state replicas. Instead, our proposal decouples data traffic and synchronization traffic, thus leading to more flexibility for the routing strategy.

Swing State [29] introduces a mechanism for state migrations entirely in the data plane but, similarly to SNAP, assumes only a single replica of a state which can be migrated across the network, on demand.

VIII. Conclusions

We consider stateful data planes, with state replication in multiple switches. We define an ILP formalization of the problem that identifies the optimal placement for the state replicas and the optimal routing for the data and synchronization traffic. To cope with the limited scalability of the ILP solver, we propose the PMR algorithm and we show that it well approximates the optimal solution. We also numerically show the beneficial effect of state replication in the reduction of the overall traffic load in the network. Finally, we provide an asymptotic analysis to compute the optimal number of state replicas in unwrapped Manhattan topology and show its applicability also to small graphs. Our results advocate the adoption of replicated states when the network application is distributed and the states are "global" across multiple switches. Notably, our work is complementary to the works showing the feasibility of implementing replicated states in state-of-art programmable data planes.

REFERENCES

[1] D. Kreutz, F. M. V. Ramos, P. Esteves Veríssimo, C. Esteve Rothenberg, S. Azodolmolky, and S. Uhlig, "Software-Defined Networking: A comprehensive survey," *Proceedings of the IEEE*, vol. 103, no. 1, pp. 14–76, Jan 2015.

[2] S. H. Yeganeh, A. Tootoonchian, and Y. Ganjali, "On scalability of software-defined networking," *IEEE Communications Magazine*, vol. 51, no. 2, pp. 136–141, 2013.

[3] P. Bosshart and al., "Forwarding metamorphosis: Fast programmable match-action processing in hardware for SDN," in *ACM SIGCOMM CCR*, 2013.

[4] M. Bonola, R. Bifulco, L. Petrucci, S. Pontarelli, A. Tulumello, and G. Bianchi, "Implementing advanced network functions for datacenters with stateful programmable data planes," in *LANMAN*. IEEE, 2017, pp. 1–6.

[5] A. Bianco, P. Giaccone, S. Kelki, N. M. Campos, S. Traverso, and T. Zhang, "On-the-fly traffic classification and control with a stateful SDN approach," in *IEEE ICC*, May 2017, pp. 1–6.

[6] K. He, J. Khalid, A. Gember-Jacobson, S. Das, C. Prakash, A. Akella, L. E. Li, and M. Thottan, "Measuring control plane latency in SDN-enabled switches," in *ACM SIGCOMM SOSR*, 2015.

[7] C. Kim, P. Bhide, E. Doe, H. Holbrook, A. Ghanwani, D. Daly, M. Hira, and B. Davie, "In-band Network Telemetry (INT)," 2016. [Online]. Available: https://p4.org/assets/INT-current-spec.pdf

[8] G. Sviridov, M. Bonola, A. Tulumello, P. Giaccone, A. Bianco, and G. Bianchi, "LODGE: LOcal Decisions on Global statEs in programmable data planes," in *IEEE NetSoft*, June 2018, pp. 257–261.

[9] ——, "LODGE: LOcal Decisions on Global statEs in programmable data planes," *arXiv preprint arXiv:2001.07670*, 2020. [Online]. Available: http://arxiv.org/abs/2001.07670

[10] M. T. Arashloo, Y. Koral, M. Greenberg, J. Rexford, and D. Walker, "SNAP: Stateful network-wide abstractions for packet processing," in *ACM SIGCOMM*, 2016.

[11] J. McClurg, H. Hojjat, N. Foster, and P. Černỳ, "Event-driven network programming," in *ACM SIGPLAN Notices*, vol. 51, no. 6, 2016, pp. 369–385.

[12] E. Brewer, "CAP twelve years later: How the "rules" have changed," *Computer*, vol. 45, no. 2, pp. 23–29, Feb. 2012.

[13] L. Lamport, "Paxos made simple," *ACM Sigact News*, 2001.

[14] D. Ongaro and J. K. Ousterhout, "In search of an understandable consensus algorithm." in *USENIX Annual Technical Conference*, 2014.

[15] K. Birman, "The promise, and limitations, of gossip protocols," *ACM SIGOPS Operating Systems Review*, vol. 41, no. 5, pp. 8–13, 2007.

[16] M. Shapiro, N. Preguiça, C. Baquero, and M. Zawirski, "Conflict-free replicated data types," in *Symposium on Self-Stabilizing Systems*. Springer, 2011, pp. 386–400.

[17] K. Petersen, M. Spreitzer, D. Terry, and M. Theimer, “Bayou: replicated database services for world-wide applications,” in *ACM SIGOPS European workshop*, 1996, pp. 275–280.

[18] D. G. Luenberger and Y. Ye, *Linear and Nonlinear Programming*. Springer Publishing Company, Incorporated, 2015.

[19] N. Megiddo, “Linear programming in linear time when the dimension is fixed,” *Journal of ACM*, vol. 31, no. 1, pp. 114–127, Jan. 1984.

[20] S. E. Schaeffer, “Survey: Graph clustering,” *Computer Science Review*, vol. 1, no. 1, pp. 27–64, Aug. 2007.

[21] K. Ruddel and A. Raith, “Graph partitioning for network problems,” in *Joint NZSA ORSNZ Conference*, no. 107, 2013, pp. 1–10.

[22] “CPLEX Optimizer.” [Online]. Available: https://www.ibm.com/analytics/cplex-optimizer

[23] D. J. Watts and S. H. Strogatz, “Collective dynamics of ’small-world’ networks,” *Nature*, vol. 393, no. 6684, p. 440, 1998.

[24] B. Gaboune, G. Laporte, and F. Soumis, “Expected distances between two uniformly distributed random points in rectangles and rectangular parallelpipeds,” *Journal of the Operational Research Society*, vol. 44, no. 5, pp. 513–519, 1993.

[25] A. Fischer, J. F. Botero, M. T. Beck, H. De Meer, and X. Hesselbach, “Virtual network embedding: A survey,” *IEEE Communications Surveys & Tutorials*, vol. 15, no. 4, pp. 1888–1906, 2013.

[26] H. Kim, J. Reich, A. Gupta, M. Shahbaz, N. Feamster, and R. Clark, “Kinetic: Verifiable dynamic network control,” in *USENIX NSDI 15*, 2015, pp. 59–72.

[27] Y. Yuan, R. Alur, and B. T. Loo, “NetEgg: Programming network policies by examples,” in *ACM SIGCOMM HotNets*, 2014, p. 20.

[28] R. Beckett, M. Greenberg, and D. Walker, “Temporal NetKAT,” *ACM SIGPLAN Notices*, vol. 51, no. 6, pp. 386–401, 2016.

[29] S. Luo, H. Yu, and L. Vanbever, “Swing State: Consistent updates for stateful and programmable data planes,” in *ACM SIGCOMM SOSR*, 2017.