Not Every Flow is Equal – SMART Discrimination in Redundancy

Pradeeban Kathiravelu
INESC-ID Lisboa / Instituto Superior Técnico
Universidade de Lisboa, Portugal
pradeeban.kathiravelu@tecnico.ulisboa.pt

Luís Veiga
INESC-ID Lisboa / Instituto Superior Técnico
Universidade de Lisboa, Portugal
luis.veiga@inesc-id.pt

Abstract

Software-Defined Data Centers (SDDC) extend virtualization, software-defined networking and systems, and middleboxes to provide a better quality of service (QoS). While many network flow routing algorithms exist, most of them fail to adapt to the dynamic nature of the data center and cloud networks and their users' and enterprise requirements. This paper presents SMART, a Software-Defined Networking (SDN) middlebox architecture for reliable transfers. As an architectural enhancement for network flows allocation, routing, and control, SMART ensures timely delivery of flows by diverting them to a less congested path dynamically in the software-defined data center networks. SMART also clones packets of higher priority flows to route in an alternative path, along with the original flow. Hence SMART offers a differentiated QoS through varying levels of redundancy in the flows.

I. Introduction

Enterprise data centers are designed to offer high-availability and fault-tolerance, abiding to service level agreements (SLA). Efficient and high performance network topologies [1] are often tailored for the specific characteristics and requirements of the data center. SDN offers flexibility and configurability to data center networks [2], while middleboxes manage the load balancing, policy control, and security aspects of the data center [3].

While ensuring lower monetary cost and improving energy and carbon-efficiency (ECE) [4], data should be transferred abiding the SLA [5]. Data locality in the networks is further driven by the geopolitical and customer requirements. Network flows in data centers consist of flows of packets of different priorities and deadlines from multiple users [6]. Priority flows often have stricter SLA deadlines to be met. Existing networks generally utilize routing algorithms that often do not consider any SLA, system policies, and user preferences.

Software and hardware middleboxes provide specific custom functions and important features crucial to the network [7], and hence cannot be eliminated from the data center deployments. Research proposes efficient architectures to mitigate the potential overheads imposed by the middleboxes [8], such as the seamless middlebox deployments enabled by SDN [7], offering complimentary features to the network. Leveraging and extending recent middlebox and SDN research and developments, flows can be tagged with custom information, that can be read and interpreted by the applications deployed on top of the northbound API of the controller [9]. Thus information on SLA, business rules, and policies can be included as custom headers with the packets.

Middleboxes can be part of an SDN, or deployed separately anywhere, with the centralized control offered by SDN [7]. FlowTags proposes an extended SDN and middlebox architecture that offers dynamic functionality to the SDN, by adding custom tags to the packets, to enforce network-wide policies, providing flow tracking capabilities [9]. Slick proposes a control plane for middleboxes, extending the SDN paradigm and architecture to network middleboxes [10]. Convergence of middleboxes and SDN has provided many advantages including flexibility in middlebox placement, effective failure handling, scalability [11], and efficient policy enforcement [12].

This paper presents SMART, an approach of adaptive redundancy in a set of flows, focusing to fulfill differentiated levels of SLA across the flows. Priority flows are tagged to indicate thresholds such as maximum routing time and other user-defined QoS parameters at the origin node by SMART. Tags will be read and interpreted at the intermediate nodes as policies, and controller will be triggered upon violation, adhering to the SDN paradigm. Thus, tags are used in detecting potential network congestion, SLA violation, or delays in routing the priority flows. When the controller is triggered for such violation, the packets from a subflow of the flow, possibly of various lengths is diverted in an alternative route to the destination, or the subflow is cloned and routed in an alternative route along with the original flow.

In the upcoming sections, we will further analyze the proposed SMART approach of differentiated redundancy. Section II discusses the design and solution architecture of SMART and elaborates the prototype implementation. Preliminary evaluations on SMART are discussed in Section III. Section IV briefly discusses the related work. Finally, Section V closes the paper discussing the current state of the research and future work.

II. *SMART* Discrimination of Network Flows

SMART is a software middlebox architecture that enforces a set of enhancements over existing network flow routing algorithms, leveraging the northbound API of an SDN controller and exploiting the functionality of adding tags to the packets proposed by FlowTags [9]. FlowTags is extended to provide SLA-awareness to the flows with minimal overhead, as no other existing SDN-based approach enables per-flow custom policy enforcement in a network with presence of software and hardware middleboxes.

SMART exploits the monitoring capabilities offered by SDN, while extending the complimentary features offered by middleboxes to mark the priority flows. Packets of priority flows consist of the information on SLA parameters, in the form of tags attached to the packets extending and

leveraging FlowTags to indicate parameters relevant for the SLA enforcement of the flow. Packets of the priority flows are tagged at the origin node by the distributed SMART software middlebox architecture consisting of FlowTagger deployed on the nodes, and the tags are read at the nodes en route destination, by the FlowTagger. Priority flows can be all the flows of a given user, all the flows originating at a given node, or a set of flows that meet a certain user-defined custom criteria.

SLA parameters contain hard limits or thresholds such as the maximum permissible routing time, and also soft limits that are fractions of the respective hard limits. Upon encountering a soft limit, based on the policy and the length and the priority level of the flow, either the flow is replicated and rerouted from its origin to the destination in an alternative route, or it is cloned or diverted from a break point node partially, to mitigate the potential SLA violation by certain malfunctioning or congested nodes in the initial route. With a little redundancy, SMART attempts to meet the deadlines of the priority flows.

A. Software Architecture and Design

Figure 1 depicts the higher level deployment architecture of SMART. OpenDaylight is extended and used as the base SDN controller, physically distributed across a cluster of computers, with a logically centralized view.

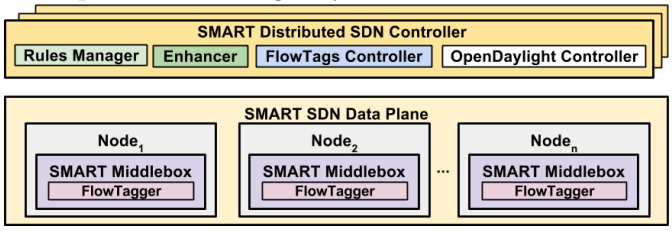

Fig. 1. SMART Deployment Architecture

FlowTags architecture has been extended and adapted as a software middlebox inside each node of the data plane as well as a FlowTags controller. The SMART middlebox consisting of the FlowTagger resides inside the nodes that are the origins of the flows. FlowTagger reads and writes tags to the packets. Tags include current time stamp to track the time consumed in routing so far which can be used to estimate other optional information such as estimated monetary cost and energy consumption.

FlowTags controller is designed as an extension to OpenDaylight SDN controller to parse the tags from the packets forwarded to the controller, and also to control and invoke the FlowTagger to tag the packets of the flows originating from a node. Policies, thresholds, and business rules are read and stored into the SMART controller from the configuration files, as defined in the network by the system administrators or the users, and managed by the rules manager. Rules manager reads the rules from the tags, and triggers the SMART enhancer according to the defined policies. SMART enhancer consists of the enhancement algorithms on top of the base routing algorithms, as an extension to the controller.

Along with the other rules set by the SDN controller, the custom user-defined tags in the packets read from FlowTagger are interpreted as policies, which the packets should respect. Upon the violation of the policies in any of the nodes, the first packet of the violating flow is sent to the SMART controller and triggers it. Then controller sets a break point in the flow on the packet that triggered the controller and the location node of the packet when the violation occurred. The break point node or packet can also be chosen algorithmically.

TABLE I. Time and Bandwidth Overhead

Approach	Duplicate Packets	Time Overhead
Divert(n)	(n-1) * (0 – 100)%	Possible
Clone(n)	n * (0 – 100)%	No/Negligible
Replicate(n)	n * 100%	No/Negligible

B. SMART Enhancement Mechanisms

SMART offers 3 alternative approaches - divert, clone, and replicate, based on the priority level of the flow, to provide an SLA-aware data center network. Algorithmic improvements handle these cases based on the nature of the flow and the defined policies. Table I presents the potential overhead for the priority flows in completion time and bandwidth usage. Time overhead is measured as a potential delay that may happen in the flow delivery compared to the original flow. Here n refers to the number of times the packets are cloned, diverted, or replicated.

Flow Diverting Approach: In the diverting approach, subflows of a few selected priority flows are routed in an alternative path to the destination, when an SLA violation is expected due to a congested node or link in the original route. The subflow is diverted in a single or multiple alternative routes except the original route. Choosing multiple alternative routes in divert approach will be useful for higher priority flows, when there is no certain alternative route that can be considered the best alternative. As the chosen routes may be longer or suboptimal than the original route, and as there is a need to reconstruct the flow, there is a potential time overhead or delay. Divert approach does not have duplicate packets if the subflow is diverted in just one alternative direction. However, diverting to multiple alternative directions will have duplicate packets, just like the cloning approach.

Flow Cloning Approach: Clone approach clones the subflow to a single or multiple alternative routes. Cloning approach is employed for higher priority flows, where flows are cloned partially or fully, instead of merely diverting. The original flow is left to continue in its route, while subflows are cloned and routed in an alternative route towards the destination. The updated rule in the break point ensures sending the packets in the original route as well as an alternative route. As the original flow is left to continue in its original route unmodified, clone approach does not have a time overhead.

Flow Replicating Approach: In the cloning and diverting approaches, controller clones or diverts the packets that follow the break point packet respectively, by changing the routing rules for the packets of the priority flows in the break point node. Alternatively, the entire violating flow can be replicated from the origin to destination in a single or multiple alternative routes. Flow cloning and replicating are further enhancements to flow diverting, as the original route could be a better choice, if the congested links or nodes recovered during the transmission. Similar

to the cloning approach, replicate approach do not have a time overhead as well. Replicating entire flows imposes 100% of duplicate packets till the routing is complete. Replicate approach can also be extended to drop the original flow, as in the divert approach. While reducing the duplicate packets, this may introduce a time overhead.

C. SMART Enhancer Algorithms

SmartRoute, the core routing procedure is described in Algorithm 1. The algorithm diverts or clones sub sets of priority flows, known as subflows, when the current routing fails to complete the transmission of the flow within the stipulated soft limit. These limits, set by the controller on the switches will trigger a communication to the controller from the switches when a violation is imminent. Soft limit parameters are often modeled as a fraction of the respective hard limit parameters, such as routing time. Tags such as priority and SLA parameters are added to the packets of the flows to provide the addition information required in accomplishing this.

Algorithm 1 SMART Enhancement

```
1:  procedure SmartRoute(flow, origin, destination)
2:      repeat
3:          BaseRoutingAlgorithm(flow, origin, destination)
4:          flow.status.update()
5:          if (flow.policies.isThresholdMet()) then
6:              cloneOrigin ← markBreakPoint(flow, origin,
                                                 destination)
7:              cloneDestination ← findCloneDestination(
                                          flow, flow.status)
8:              clonedFlow ← cloneFlow(flow, cloneOrigin,
                                              cloneDestination)
9:          end if
10:     until (flow.allReceived(cloneDestination) or
                                  flow.allReceived(destination))
11:     mergeFlows(flow,clonedFlow)
12:     dropPacketsOnTransmission(flow.parentID)
13: end procedure
```

The *BaseRoutingAlgorithm* refers to any underlying routing algorithm such as Dijkstra's shortest path algorithm [13] or equal-cost multi-path (ECMP) algorithm [14], which is to be enhanced by SMART. The thresholds can be defined as system-wide policies, such as minimal throughput and latency, in network system and individual flow level.

Statistics when routing through each link is monitored to offer fault-tolerance to the data center network. Nodes or links that take much longer time to route the flows or packets than the average time to route, those consume unconventionally large amount of energy or computing resources in routing, or those who exhibit a similar behavior that may lead to exceeding the threshold specified in the SLA, are considered to be functioning poorly, and acted upon. The status of the flow is updated to the controller as the tags are read by the middlebox architecture.

SmartRoute routes the flows from origin to destination entirely using BaseRoutingAlgorithm, unless the threshold is met. The SMART software middlebox is triggered to reroute the subflow in the new alternative route towards the destination, or is invoked to forward the packets of the subflow to both the original and alternative route by the SDN controller, when an SLA violation for the flow is imminent according to the policies, or when the threshold defined in the flow policies is met.

A node and a packet are chosen as the break point node and packet respectively. Having the break point node as the origin, a subflow is cloned or diverted starting from the break point packet to the rest of the flow. The destination of the cloned or diverted subflow is defined as the clone destination, where the subflow is merged with the rest of the flow to reconstruct the original flow.

Clone Destination: Clone destination is either the destination of the original flow, or the next node following the congestion. The procedure findCloneDestination() decides the clone destination based on the flow and its status, which consists of further information which can be used to find the nature of the policy violation. When a congestion is encountered, the exact destination is decided based on the characteristics of the congestion. In a large data center with a few nodes identified to be contributing to the congestion, the cloned or diverted subflow is routed towards the node that immediately follows the congested link to avoid routing in a sub-optimal path when the congestion affects just one or a few of the nodes in the original route. This also minimizes redundant packets by enabling early recomposing of the original flow. If there is no such nodes identified to be contributing to the congestion, the cloned or diverted subflow is routed towards the original destination in an alternative route.

Flow Reconstruction: Once the entire packets of the flow, regardless whether from original or cloned flows, are received at clone destination, the original flow is reconstructed. If the clone destination is different from the original destination, the recomposed flow is left to continue in its original route towards the destination. Leveraging the FlowTags custom tags, duplicate packets are detected and dropped in the clone destination upon receiving the entire flow, ensuring end-to-end transmission guarantee.

Cloning approach minimizes the necessity to reconstruct the flow, if the entire packets from the original flow are received before the packets from the clone, hence dropping the clone. For the diverting approach, and for the cloning approach if the packets of the cloned flows arrived earlier, the flow will be reconstructed by merging the packets from the cloned or diverted subflow to the packets of the original flow that have already arrived.

The following priority flows of the same path may be replicated and rerouted, or diverted in the origin, in an alternative route. Thus, while the initial flows that are identified to violate the SLAs may still violate SLAs due to the time taken in cloning the flow, following flows will be able to avoid the violating route altogether. A replicate approach resends the entire flow from the origin to the destination in one or more alternative routes, which avoids the necessity for recomposing and packet-level manipulation, with more redundancy.

Break Point: Break point is a pointer to the node and flow where the subflow is cloned. The controller chooses the break point programmatically, and writes rules on the break point nodes to divert or clone the upcoming packets of the priority flows. While break point is crucial in the SMART enhancement algorithm, it is used just for the subflow construction, and information on break points are not stored statically in the flows or the controller beyond the time frame of subflow construction.

Algorithm 2 elaborates marking a break point for the flow, which first needs deciding the exact node to be the break point node, and also find the exact packet from which the flow is to be included in the diverted or cloned subflow. If a specific node or link is estimated to be responsible for the policy violation, the node will be marked as the breakpoint node. If there is no such specific or explicit malfunctioning link or node to be blamed, the delay may be due to other factors such as network congestion across multiple nodes and links or the flows being much larger than the average flows in the data center and hence taking longer than expected. Here, the break points depend on policies or are decided statistically.

Algorithm 2 Marking the Break Point

```
1:  procedure markBreakPoint(flow, origin,
                             destination,policies, links)
2:      for (link in flow.route) do
3:          if (policies.isThresholdMet(link.param)) then
4:              ▷ A clearly visible malfunctioning link exists
5:              breakPoint.node ← current.node
6:              breakPoint.packet ← current.packet
7:              Return breakPoint
8:          end if
9:      end for
10:     breakPoint ← flow.estimate(policies.breakPolicy)
11:     Return breakPoint
12: end procedure
```

As the origin of the diverted or cloned subflow, break point node reroutes the packets to the destination in an alternative route as they arrive at the node. All the following packets arriving to the break point node will be diverted in the alternative route, while the packets of the original flow following the break point is left to continue in the clone approach.

D. Prototype Implementation

A prototype of the proposed solution is implemented leveraging the OpenDaylight controller, while exploiting simulation and emulation environments to provide the network. A distributed controller environment is created with an Infinispan [15] cache over a distributed network cluster. An elastic in-memory cluster architecture proposed in our previous work has been extended to provide a distributed adaptive execution of the controller [16].

Network flow routing algorithms that are commonly used in data center networks, such as the shortest path algorithm, are implemented as the base algorithms. SMART Algorithmic improvements were then applied on top of these base algorithms. As OpenDaylight follows the OSGi (Open Service Gateway Initiative) [17] specification and offers a componentized modular architecture deployed on top of Apache Karaf, the controller extensions are developed as independent OSGi bundles and deployed alongside with the controller core bundles. The Model-Driven Software Engineering (MDSE) principles offered by the model-driven service abstraction layer (MD-SAL) [18] of OpenDaylight Lithium was leveraged in integrating the controller extensions and middlebox controllers. Algorithmic enhancements and extensions are deployed similarly. Due to the loose coupling in the design, SMART can be made to work with other controllers with minimal changes.

III. Preliminary Evaluation

SMART prototype was evaluated in a distributed simulation and emulation environment, on a cluster with 6 identical nodes (Intel® Core™ i7-2600K CPU @ 3.40GHz processor and 12 GB memory). Prototype implementation of SMART enhancements was compared with base algorithms commonly used, to assess SLA fulfillment regarding priority flows, by extending the *xSDN* software-defined networking enabled platform for network flow simulations [19]. Experiments were carried out on multiple routing scenarios with the different SMART enhancement approaches, to evaluate the QoS, efficiency, and potential time and bandwidth overheads of SMART. Different type and number of flows with multiple different policies and intents were evaluated.

A. Long Running Flows

The network was modeled as a small-world data center (SWDC) topology [1] with 1024 nodes and shortest path as the base routing algorithm, and flows were routed between chosen origin and destination nodes. Properties of the links, nodes, and flows were uniformly distributed, and congestion uniformly randomized. Network congestion was modeled by dynamically making certain links slower to route. The routing process was repeated with SMART enhancements applied over the base routing algorithm, where the subflows of the priority flows were diverted. SLA was defined as the maximum time to complete the transfer of the priority flows. Slow links were marked from the descriptors of the simulation environment and failures and congestions were randomized across the links.

SMART was initially evaluated for priority flows of longer duration. In a control experiment with no congestion, the flows took up to around 2 minutes (120 secs.) to complete, which is used as the approximate value for the soft-threshold in the experiment. The SLA limit was indicated as 250 secs. Figure 2 shows the time taken for the routing with, and without the SMART enhancements for individual priority flows of equal lengths. This, across different origin and destination in multiple routing paths with or without congestion in certain links across the paths occurring randomly.

The complete time taken for the flow to reach the destination from origin is measured as the routing time. Though routing time is increased compared to the base routing algorithm without congestion, results show that SMART was able to avoid SLA violations in the data center, where the base algorithm violated the SLA for around 33% of the priority flows in the presence of congestion in a few links. SLA violations were minimized for the selected priority flows by dynamically avoiding the slow routes as

they are monitored and reported to the controller, based on previous packets of the flow. Moreover, limited additional bandwidth consumption and controller CPU and memory load were observed.

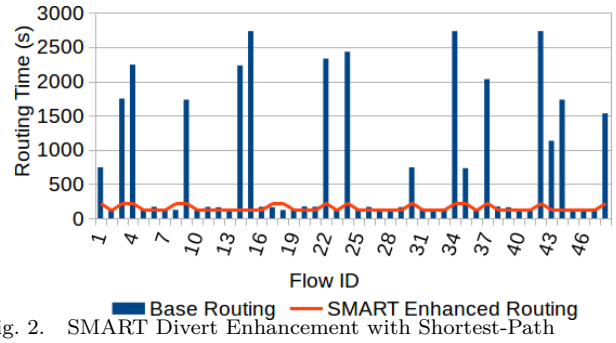

Fig. 2. SMART Divert Enhancement with Shortest-Path

In the regular routing, when a slow or congested route is encountered, the flow path cannot be changed dynamically, and the rest of the flow continues its routing in the original path regardless of a potential better alternative. In the SMART approach, after the specified time limit, SMART enhancer diverts or clones the subflow following the break point in an alternative route, which enhances the chosen base algorithm. While SMART depends on the availability of alternative routes in the network between the chosen node pairs, highly connected networks such as a mesh network typically offer a higher potential of finding an alternate route, that is as good as the shortest path, and should thus be leveraged.

Overheads of Flow Cloning: Similar routing times were observed when the experiments were repeated with cloning the subflows following the break point, instead of just diverting. Redundancy in packets was monitored with flow cloning, and overhead imposed by SMART was measured them. Figure 3 shows the routing time of the priority flows and time of routing with redundant packets due to the cloned subflows. It also shows the estimated overhead imposed by SMART on the routing.

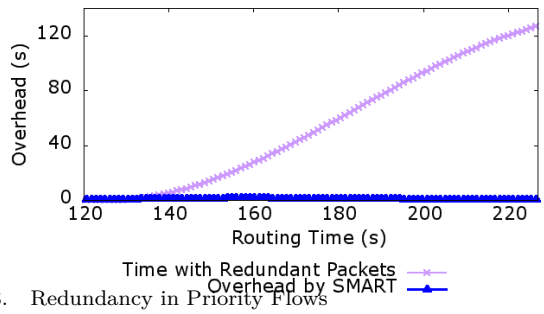

Fig. 3. Redundancy in Priority Flows

There were no redundant packets till the soft limit is met and following that, the cloned packets assume an alternative route while the original flow continues. Hence a redundancy of up to 50% of the entire routing time till the flow is recomposed, was observed, as shown by Figure 3. Further, Figure 3 also indicates the overhead in routing time caused by SMART atop the base routing time without congestion. The overheads imposed by the SMART enhancements on the controller and switches are minimal for large flows, around 0.2% of the routing time, which is often relatively a constant and negligible compared to the enhancements offered by SMART beyond 100% in a congested route.

B. Short Running Flows

The experiment was repeated in the same data center simulation, modeled with up to 100,000 of short flows each consuming less than 1 second to complete its routing. Figure 4 shows the time taken for a flow to route using shortest path as the base routing algorithm, as well as with SMART enhancements to clone the flows that exceed the soft-threshold. Thick and solid filled blocks in the diagrams indicate clustered outcomes for the pairs of base routing time vs SMART enhanced routing times, where thin and white blocks indicate single or less repeating pairs of observed values. During the congestion, an immediate overhead of around hundred milliseconds caused by SMART was observed. Yet SMART offered a speed up of up to 500% in the presence of congestion.

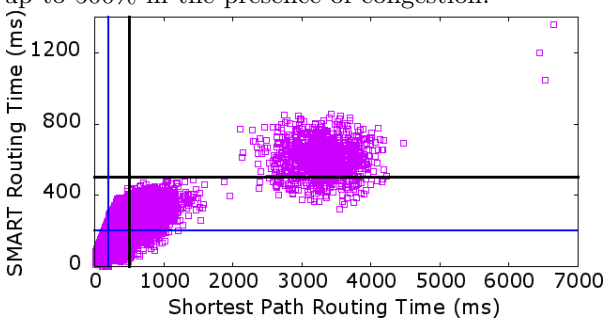

Fig. 4. SMART Clone Enhancements with Shortest-Path

SMART was configured to replicate the following flows at the origin when a flow of the same path reported a violation and cloned. Figure 5 indicates the time taken for SMART configured with this adaptive behavior, indicating there was no SLA violation with SMART enhancements.

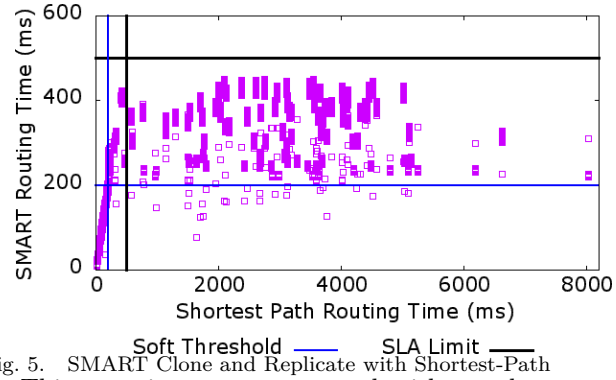

Fig. 5. SMART Clone and Replicate with Shortest-Path

This experiment was repeated with equal-cost multi-path (ECMP) routing algorithm. Figure 6 shows the time taken to route the flows in ECMP as the base algorithm in a congested network, with and without SMART enhancements. As a base algorithm, ECMP distributes the flows across the alternatives. However, it is not aware of the congestion. Hence, SMART was able to enhance its performance by cloning the priority flows in an alternative route, which was readily available in ECMP, further replicating the following flows of the same path, that originally was found to be congested in an alternative route.

Considering all the cases, SLA violations were avoided by SMART by up to 95%. The majority of the flows that originally violate SLA, abide to the SLA with the SMART enhancements. Performance of the controller and

switches in detecting the violations, and updating the rules, contributes to the potential SLA violations. However, there is no flow which has an SLA violation with SMART enhancement, which is not also violated with the base routing. Unless the soft threshold was met, SMART enhancements were not invoked, as it indicated that the existing route was good enough to meet SLA and no potential congestion was foreseen.

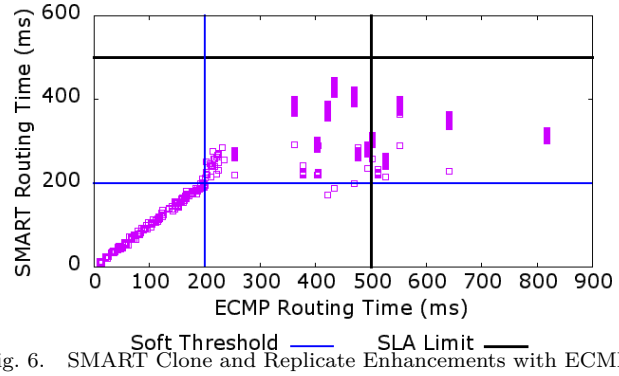

Fig. 6. SMART Clone and Replicate Enhancements with ECMP

Assessment of Overheads: Bandwidth overhead was shown to be depending on multiple factors, such as the topology and size of the data center network, average length between any node pair, length of the congestion in the route (how many nodes and links are congested), location of the congestion and break point in the route and the flow, and the number of alternative routes available between any two nodes.

SMART exhibits an adaptive behavior to the nature of the congestion, finding the right time to clone or divert. The contribution to congestion from cloning the subflows is minimal, as only around 16.7% of the packets of the higher priority flows were found to be cloned in the typical data center network modeled, and hence the overall redundancy will be further smaller, depending on the fraction of the flows that are marked as higher priority to be cloned.

The logically centralized controller in an enterprise data center is a physically distributed cluster of high performance servers. Hence, the controller computations, such as determining the break point node and packet, monitoring the network flows for thresholds, imposing/changing the flow tables and policies in the relevant switches, and enforcing the SLA for the priority flows in the congested network based on the tags, are executed in the scale of microseconds. The overhead was estimated to be lower than a hundred milliseconds in switches when the break points are manipulated and flow tables are updated, with a minimal overhead in the bandwidth. As the base routing and FlowTagger are integrated with the SDN architecture, no overhead was caused by the deployment of SMART.

Mininet emulations of an about 1000-node data center with a distributed controller deployment of OpenDaylight over 6 nodes and SMART enhancements showed that the controller can handle the routing, rerouting, and reconstruction of flows and subflows effectively. This, without creating a bottleneck, as the majority of the decisions are handled by the nodes themselves with minimal intervention from the controller, unless a violation is triggered. Subflows still respect the ordering of packets. Hence, reconstruction of the original flow at the destination is straightforward, dropping the duplicate packets. The enhancements are adaptive to minimize the overhead even for much smaller flows, where if the performance improvement is minimal by cloning subflows, entire following/downstream flows of the same priority, in the same path, will be replicated and routed in an alternative route along with the original route, or just rerouted in an alternative route omitting the slow route.

IV. Related Work

While hosts in the data center networks are connected through multiple paths, TCP limits the connection to a single path. Multipath TCP (MPTCP) is a transport protocol that uses the available multiple paths between the nodes concurrently to route the flows across the nodes. MPTCP is proposed and implemented as an enhancement to TCP to improve the performance, bandwidth utilization, and congestion control through a distributed load balancing [20]. MPTCP uses subflows in routing the flows, leveraging the multiple paths between the nodes in a network, and reconstruct the data in the destination in the original order [21].

Conga offers congestion-aware load balancing for data center networks through flowlet switching [22]. Flowlets are defined as the bursts or chunks of packets of a flow, that is separated with the other bursts of chunks by a gap [23]. Flows are often composed of flowlets and gaps between the flowlets, enabling an efficient partitioning of flows as flowlets and routing them in multiple alternative routes. While dynamically rerouting the network flows to optimize the bandwidth consumption has been proposed in the previous work [24], further research is necessary to enhance the existing networks and flow scheduling by leveraging the availability of the entire view in the central controller consisting of large computational power.

Though data center networks are efficiently orchestrated and scaled with SDN, SLAs cannot be promised without dedicated and replicated resources. The existing work that leverages MPTCP or flowlets do not use redundant subflows for a reliable transfer of flows, or prioritize the flows based on user preferences to satisfy SLAs. Resending or cloning the flowlets if a previous flowlet has not reached the destination within the stipulated time should be researched. SLA-aware data center networks should be designed by exploiting the functionality offered by the middleboxes with minimally replicated resources and redundancy to ensure timely delivery of priority content.

V. Conclusion and Future Work

SMART is developed as a fully functional middlebox-based approach for software-defined data center networks, by diverting or cloning subflows of priority flows for a timely delivery in a network with congested links. Preliminary evaluations on simulation and emulation platforms showed the efficiency of SMART in offering SLA-awareness to data center networks. As FlowTags effectively enforces policies regardless of the presence of middleboxes that modify the flow headers in the network, *SMART* deployment is orthogonal to the presence of middleboxes. An ongoing development effort implements SMART on a real data center network and evaluate against CONGA and congestion-aware data center networks.

References

[1] J.-Y. Shin, B. Wong, and E. G. Sirer, "Small-world datacenters," in *Proceedings of the 2nd ACM Symposium on Cloud Computing*. ACM, 2011, p. 2.

[2] S. Sezer, S. Scott-Hayward, P.-K. Chouhan, B. Fraser, D. Lake, J. Finnegan, N. Viljoen, M. Miller, and N. Rao, "Are we ready for sdn? implementation challenges for software-defined networks," *Communications Magazine, IEEE*, vol. 51, no. 7, pp. 36–43, 2013.

[3] D. A. Joseph, A. Tavakoli, and I. Stoica, "A policy-aware switching layer for data centers," *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 4, pp. 51–62, 2008.

[4] A. Khosravi, S. K. Garg, and R. Buyya, "Energy and carbon-efficient placement of virtual machines in distributed cloud data centers," in *Euro-Par 2013 Parallel Processing*. Springer, 2013, pp. 317–328.

[5] J. Simao and L. Veiga, "Flexible slas in the cloud with a partial utility-driven scheduling architecture," in *Cloud Computing Technology and Science (CloudCom), 2013 IEEE 5th International Conference on*, vol. 1. IEEE, 2013, pp. 274–281.

[6] C. Wilson, H. Ballani, T. Karagiannis, and A. Rowtron, "Better never than late: Meeting deadlines in datacenter networks," in *ACM SIGCOMM Computer Communication Review*, vol. 41, no. 4. ACM, 2011, pp. 50–61.

[7] A. Gember, P. Prabhu, Z. Ghadiyali, and A. Akella, "Toward software-defined middlebox networking," in *Proceedings of the 11th ACM Workshop on Hot Topics in Networks*. ACM, 2012, pp. 7–12.

[8] M. Walfish, J. Stribling, M. N. Krohn, H. Balakrishnan, R. Morris, and S. Shenker, "Middleboxes no longer considered harmful." in *OSDI*, vol. 4, 2004, pp. 15–15.

[9] S. K. Fayazbakhsh, L. Chiang, V. Sekar, M. Yu, and J. C. Mogul, "Enforcing network-wide policies in the presence of dynamic middlebox actions using flowtags," in *Proc. USENIX NSDI*, 2014.

[10] B. Anwer, T. Benson, N. Feamster, D. Levin, and J. Rexford, "A slick control plane for network middleboxes," in *Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking*. ACM, 2013, pp. 147–148.

[11] Z. Qazi, C.-C. Tu, R. Miao, L. Chiang, V. Sekar, and M. Yu, "Practical and incremental convergence between sdn and middleboxes," *Open Network Summit, Santa Clara, CA*, 2013.

[12] Z. A. Qazi, C.-C. Tu, L. Chiang, R. Miao, V. Sekar, and M. Yu, "Simple-fying middlebox policy enforcement using sdn," in *ACM SIGCOMM Computer Communication Review*, vol. 43, no. 4. ACM, 2013, pp. 27–38.

[13] E. W. Dijkstra, "A note on two problems in connexion with graphs," *Numerische mathematik*, vol. 1, no. 1, pp. 269–271, 1959.

[14] C. E. Hopps, "Analysis of an equal-cost multi-path algorithm," 2000.

[15] F. Marchioni, *Infinispan data grid platform*. Packt Pub., 2012.

[16] P. Kathiravelu and L. Veiga, "An adaptive distributed simulator for cloud and mapreduce algorithms and architectures," in *Utility and Cloud Computing (UCC), 2014 IEEE/ACM 7th International Conference on*. IEEE, 2014, pp. 79–88.

[17] T. Gu, H. K. Pung, and D. Q. Zhang, "Toward an osgi-based infrastructure for context-aware applications," *Pervasive Computing, IEEE*, vol. 3, no. 4, pp. 66–74, 2004.

[18] J. Medved, R. Varga, A. Tkacik, and K. Gray, "Opendaylight: Towards a model-driven sdn controller architecture," in *2014 IEEE 15th International Symposium on*. IEEE, 2014, pp. 1–6.

[19] P. Kathiravelu and L. Veiga, "An expressive simulator for dynamic network flows," in *Software Defined Systems (SDS), 2015 IEEE 2nd International Workshop on*. IEEE, 2015, pp. 311–316.

[20] C. Raiciu, S. Barre, C. Pluntke, A. Greenhalgh, D. Wischik, and M. Handley, "Improving datacenter performance and robustness with multipath tcp," *ACM SIGCOMM Computer Communication Review*, vol. 41, no. 4, pp. 266–277, 2011.

[21] A. Ford, C. Raiciu, M. Handley, and O. Bonaventure, "Tcp extensions for multipath operation with multiple addresses," Tech. Rep., 2013.

[22] M. Alizadeh, T. Edsall, S. Dharmapurikar, R. Vaidyanathan, K. Chu, A. Fingerhut, F. Matus, R. Pan, N. Yadav, G. Varghese *et al.*, "Conga: Distributed congestion-aware load balancing for datacenters," in *Proceedings of the 2014 ACM conference on SIGCOMM*. ACM, 2014, pp. 503–514.

[23] S. Kandula, D. Katabi, S. Sinha, and A. Berger, "Dynamic load balancing without packet reordering," *ACM SIGCOMM Computer Communication Review*, vol. 37, no. 2, pp. 51–62, 2007.

[24] M. Al-Fares, S. Radhakrishnan, B. Raghavan, N. Huang, and A. Vahdat, "Hedera: Dynamic flow scheduling for data center networks." in *NSDI*, vol. 10, 2010, pp. 19–19.