SPANKING SDN

Anaswara Dinesh
Master in Computer Science & Engineering
RCET,Akkikkavu,Thrissur
anaswarashinil@gmail.com

***Abstract*. Software-defined networking technology is an approach to network management that enables dynamic programmatically efficient network configuration in order to improve network performance and monitoring, making it more like cloud computing than traditional network management. It provides a framework to dynamically adjust and reprogram the data plane with the use of flow rules. The realization of highly adaptive software-defined networks with the ability to respond to changing demands or recover after a network failure in a short period of time hinges on efficient updates of flow rules. To model the time to deploy a set of flow rules by the update time at the bottleneck switch, and formulate the problem of selecting paths to minimize the deployment time under feasibility constraints as a mixed-integer linear program. To reduce the computation time of determining flow rules, we propose efficient heuristics designed to approximate the minimum-deployment-time solution by relaxing the Mixed Integer Programming or selecting the paths sequentially. Through extensive simulations, we show that our algorithms outperform current, shortest path based solutions by reducing the total network configuration time**

Keywords—SDN:Software-Defined Networking,TCAM:Ternary Content-Addressable Memmory,IP:Internet Protocol, IoT:Internet of Things.

I. INTRODUCTION

SDN is widely recognized that flow configuration of flow forwarding rules at the SDN switches, is critical for SDN operation. Configuration updates due to changed flows must be performed consistently and quickly to avoid congestion delays, loops, and policy violations. The need to re-establish disrupted flow caused by failing links as the result of a failure, congestion or attack on the network infrastructure, motivates the requirement to perform fast network reconfiguration. The time required for deploying a given flow configuration is dominated by the time to insert/update flow rules in the ternary content addressable memory of each involved switch. Hardware reigned supreme in the networking world until the emergence of software-defined networking (SDN), a category of technologies that separate the network control plane from the forwarding plane to enable more automated provisioning and policy-based management of network resources. It is an approach to network management that enables dynamic, programmatically efficient network configuration in order to improve network performance and monitoring, making it more like cloud computing than traditional network management.SDN has evolved into a reputable networking technology offered by key vendors including Cisco, VMware, Juniper, Pluribus, and Big Switch. The Open Networking Foundation develops myriad open-source SDN technologies as well.

The idea of programmability is the basis for the most precise definition of what SDN is: technology that separates the control plane management of network devices from the underlying data plane that forwards network traffic.

IDC broadens the definition of SDN by stating: “ Datacenter SDN architectures feature software-defined overlays or controllers that are abstracted from the underlying network hardware, offering intent-or policy-based management of the network as a whole. This results in a data center network that is better aligned with the needs of application workloads through automated (thereby faster) provisioning, programmatic network management, pervasive application-oriented visibility, and where needed, direct integration with cloud orchestration platforms.”

II. BACKGROUND

Software-defined network (SDN) controllers which include mechanisms to globally reconfigure the network in order to respond to a changing environment. As demands arrive or leave the system, the globally optimum flow configuration changes over time. Although the optimum configuration can be computed with standard iterative methods, convergence may be slower than system variations, and hence it may be preferable to interrupt the solver and restart. In this paper, we focus on the class of iterative solvers with an exponential decrease over time in the optimal gap. Assuming dynamic arrivals and departures of demands, the computed optimality gap at each iteration is described by an auto-regressive stochastic process. At each time slot, the controller may choose to 1) stop the iterative solver and apply the best found configuration to the network or 2) allow the solver to continue the iterations keeping the network in its sub-optimal form. Choice 1) reduces the optimality gap leading to smaller routing costs but requires flow reconfiguration which hurts QoS and system stability. To limit the negative impact of reconfigurations, we propose two control policies that minimize the time-average routing cost while respecting a network reconfiguration budget. We experiment with realistic network settings using standard linear programming tools from the SDN industry. In the experiments conducted over the GEANT networks and fat-tree networks, our policies provide a practical means of keeping the routing cost small within a given reconfiguration constraint.

Software-Defined Networking enables efficient utilization of network resources by dynamically adapting the routing configuration over time. In this context, this paper addresses an important question about the interplay between the high degree of configuration flexibility and the computational limits of the SDN controller logic. Specifically, examine the problem where the optimality gap of iterative routing algorithms decreases exponentially fast and we want to minimize the average routing cost subject to a constraint for the average reconfiguration frequency. Furthermore, we present two control policies working on top of the online routing optimization engine to decide whether to apply or not the current yet not optimal global network configuration. Numerical results on the GEANT network and fat-tree network topologies show that our control schemes can effectively track the evolution of the system using a bounded number of reconfiguration, thus pursuing the double objective of optimizing the performance and the system stability. The problem of consistently increasing the flow of traffic between a source and a terminal node, while keeping all other traffic flows intact. Current methods such as RSVP-TE consider unsplittable flows and assign weights according to the importance of the flow. To cope with this, the changing data flows get sent through the network by routing algorithms in a greedy(shortest path) way that favors distributed fault-tolerance over efficiency. While this makes sense in a scenario where the network is controlled by various independent participants. The situation changes when the network is controlled by one entity. With only one logical central controller, it is possible to actually reclaim the control over the general behavior of the network and just leave the menial task of data forwarding localized in the switches and routers. This is one of the fundamental ideas that gave rise to these ideas.

But, while in this system the new network behavior might be optimal, what happens during the migration to the new behavior? Clock synchronization is far from perfect, and even if, some switches will straggle (taking up to 100 longer than average to update in practice) or might not be available to the central controller at all for some time. This inherent asynchrony will lead to over-utilization of links, inducing congestion, and thus packet loss is raised here.

Also studied consistent migration of flows, with a special focus on software-defined networks. Given a current and a desired network flow configuration and give the first polynomial-time algorithm to decide if a congestion-free migration is possible. However, if all flows must be an integer or are unsplittable, this is NP-hard to decide. A similar problem is providing increased
bandwidth to an application while keeping all other flows in the network, but possibly migrating them consistently to other paths. It shows that the maximum increase can be approximated arbitrarily well in polynomial time. Current methods as RSVP-TE consider unsplittable flows and remove flows of lesser importance in order to increase bandwidth for an application: So prove that deciding what flows need to be removed is an NP-hard optimization problem with no PTAS possible unless P = NP. a new algorithm for SDN network updates that preserve forwarding policies. FLIP builds upon the dualism between replacements and additions of switch flow-table rules. It identifies constraints on rule replacements and additions that independently prevent policy violations from occurring during the update. Moreover, it keeps track of alternative constraints, avoiding the same policy violation. Then, it progressively explores the solution space by swapping constraints with their alternatives, until it reaches a satisfying set of constraints. Extensive simulations show that FLIP outperforms previous proposals. It achieves a much higher success rate than algorithms based on rule replacements only, and massively reduces the memory overhead with respect to techniques solely relying on rule additions.

III. SYSTEM DESCRIPTION

software-defined networking (SDN) separates the control from the data plane in network devices, like switches and routers. This new concept suggests the use of a centralized controller that determines the behavior of all forwarding components in the network. Southbound interfaces permit communication between the control plane and the data plane, while northbound interfaces provide enormous possibilities for networking programmability, like creating applications that can automate all networking tasks. Consequently, SDN will enhance creativity, as well as innovation, in the domain of networking.

Three critical requirements are not achievable in an SDN-enabled centralized network, which was the main tendency for early proposed SDN architectures, using just one controller: first, efficiency that is not enough established with just one centralized controller, second, scalability that is one of the most issues that pushes network architects to consider the idea of multi controllers, and, third, high availability, which has two items, redundancy, and security. Redundancy is one of the most significant aspects of any design. One controller could fail anytime and, for this reason, abandon the network without its control plane. Security is considered an important item. If an attacker compromises the controller, subsequently it loses the entire management over the network. Clearly, if we have multiple controllers, we can certainly minimize the issue, because they will team up to identify that another one is misbehaving and for that reason separate the attacker from the network.

Brisk SDN develop an optimization framework and associate flow configuration algorithms to support fast-changing flflow demands in SDNs, by taking into account both the time to compute a new flow configuration at the controller and the time to deploy this configuration at the switches. The focus of this work is on networks with high flow dynamics, such as data centers, corporate networks, or IoT deployments. As a trade-off for faster configuration time, our framework generates slightly longer paths on average, compared to an algorithm which aims at satisfying flows through paths of minimum length.

The main focus of this model is on SDN, network model, and it's performance. The future SDN has a problematic future as it has issues to overcome. The first issue is the console/remote capability with today's security issues many will not want to expose their network to a potential hacker takeover. It is an Open Source Technology. Another issue is the current state of Network Management policies and practices with a single device or single path focus. When a Network Manager looks at SDN he only sees how it can help or hurt his network but SDN is much bigger and a lot of education still remains to be done to make SDN or similar technologies palatable.SDN is a Human Centric Technology where this technology is Device Centric which is and always will be a challenge to get managers to adopt especially when one person can completely change your network, storage, WAN, etc fabric.

Advantage of the concept of a multi-controller design, but at the same time, and always consider that we have a single controller. In other words, can take the charge, and can distribute it among the multiple controllers; however, for the underlying layer, it is like there is just one controller that commands the whole network. Another idea was proposed before implementing multiple controllers, which is installing replicated controllers to remove the single point of failure. In a word, a logically centralized architecture stays near to the initial tendency of SDN, which is using a single controller, or a multicore controller to improve the performance. On the other hand, a logically distributed architecture goes away from the first tendency of SDN, by making several controllers have several responsibilities inside the network.

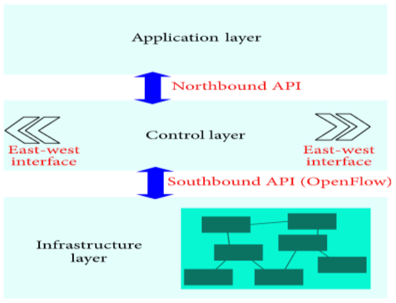
In the SDN model, the splitting of the control and data forwarding functions is referred to as "disaggregation," because these pieces can be sourced separately, rather than deployed as one integrated system. This model gives the applications more information about the state of the entire network from the controller, as opposed to traditional networks where the network is only application-aware.

SDN model generally has three components or groups of functionality. First SDN applications are programs that communicate behaviors and needed resources with the SDN controller via application programming interfaces (APIs). In addition, the applications can build an abstracted view of the network by collecting information from the controller for decision-making purposes. These applications could include networking management, analytic, or business applications used to run large data centers. For example, an analytic application might be built to recognize suspicious network activity for security purposes. The second one is The SDN controller is a logical entity that receives instructions or requirements from the SDN application layer and relays them to the networking components. The controller also extracts information about the network from the hardware devices and communicates back to the SDN applications with an abstract view of the network, including statistics and events about what is happening. And finally he SDN networking devices control the forwarding and data processing capabilities for the network. This includes the forwarding and processing of the data path. The SDN architecture APIs are often referred to as northbound and southbound interfaces, defining the communication between the applications, controllers, and networking systems. A northbound interface is defined as the connection between the controller and applications, whereas the southbound interface is the connection between the controller and the physical networking hardware. Because SDN is a virtualized architecture, these elements do not have to be physically located in the same place.

Through this demonstrated model can find out the network communication delay. After finding queuing, computing, communication & deployment delay it helps to model network time update algorithm.

A,FRAME MODEL

It demonstrates the SDN architecture with single-path routing (i.e., unsplittable flflows) in this work since single-path routing was the technique typically used in the SDN controllers (POX, Floodlight, OpenDayLight) and switches (Brocade ICX 6610) at the time.



Both the SDN control plane and data plane elements of a networking architecture were packaged in proprietary, integrated code distributed by one or more proprietary vendors. The OpenFlow open-source standard, created in 2008, was recognized as the first SDN architecture that defined how the control and data plane elements would be separated and communicate with each other using the OpenFlow protocol. The Open Network Foundation (ONF) is the body in charge of managing OpenFlow standards. There are other standards and open-source organizations with SDN resources, so OpenFlow is not the only protocol that makes up SDN. Model a network as an undirected graph, with a set of switches and the set of links. Each link is associated with capacity. Let H denote the current set of flows on this network. Each flow is associated with a source, a destination, and a demand specifying the flow rate. Depending on the construction of SDN rules, which can be specified by the network operator, a flow can be a single flow identified by a pair of IP addresses, as well as a batch of flows identified by an IP-prefix. The framework will compute an assignment of flflows on a network topology, where the detailed definition of a flow can be set by the network operator.

Model a network as an undirected graph with a set of switches and the set of links. Each link is associated with capacity. Each flow is associated with a source, a destination, and a demand specifying the flow rate. Depending on the construction of SDN rules, which can be specified by the network operator, a flflow can be a single flow identified by a pair of IP addresses, as well as a batch of flows identified by an IP-prefix. The framework will compute an assignment of flows on a network topology, where the detailed definition of a flow can be set by the network operator. In this framework, Assume single-path routing (unsplittable flows) in this work, since single-path routing was the technique typically used in the SDN controllers and switches at the time.

B.UPDATE CONFIGURATION

After construct the topology find the delay between the flow arrival and the time the network is reconfigured to support it includes four components:

- Queuing delay: Amount of time or a certain number of flow demands to be processed in a batch.Avoid combining SI and CGS units, such as current in amperes and magnetic field in oersteds. This often leads to confusion because equations do not balance dimensionally. If you must use mixed units, clearly state the units for each quantity that you use in an equation.
- Computation delay: The controller calculates the time for new configuration.
- Deployment delay : Which is the time to deploy the rules at the switches.

The queuing delay the time an SDN controller waits until a batch of flows is processed and the network configuration is updated. In highly dynamic environments such as data-centers or IoT deployments where the network is based on SDN, it is beneficial to set the queuing delay to a specified time threshold such as the lower bound of TCP retransmission time. The queuing delay can also be set as a threshold to the number of arriving packet-in messages in the SDN controller. Therefore it is recommended to use a combination of both approaches and trigger a new network configuration whenever the first threshold condition, waiting time or number of packet in messages, is satisfied.Then analyze the impact of flow configuration on the deployment time. In SDN, deploying a new configuration requires updating the corresponding rules in the flow tables of the switches. After the observation then update time of a flow table tends to grow with the number of updated rules, as updates on a given switch must be performed sequentially to ensure consistency.

After insert/update a flow rule in the TCAM of SDN-enabled switches varies significantly depending on different hardware vendors and rule priorities.

The goal is to minimize the network update time by selecting a path for each flow in the set of flows which minimizes the maximum number of updated rules per switch, weighted by its per-rule update time. Then represent the path selection by a decision variable, which indicates if flow traverses undirected link, the traversals of links are directed. The decision variables must satisfy the flow conservation constraint. Then ensures that each flow is routed along a single path, and constraint ensures that the objective values are no smaller than the maximum update time of any switch.

In the case of non-disruptive the solution to the MinUpdateTime problem can reroute existing flows, which can cause temporary disruption to applications relying on these flows. For disruption-sensitive applications, it is desirable that the configuration is calculated for the new flows only, leaving the existing flow undisrupted.

Identify applicable sponsor/s here. If no sponsors, delete this text box (*sponsors).*

C.SHORTEST PATH DETERMINATION

The first step is a set of paths between each pair of nodes with minimum overlap, such that they will provide enough diversity to route the flflows to avoid excessive updates at any single switch. Then precompute paths per switch pair, where the path is a design parameter that controls the trade-off between complexity and optimality. Specifically, given a parameter path and a switch source-destination pair, then want to and critical paths from source to a destination such that the maximum number of paths traversing the same switch is minimized.

In the case of the Non-Disruptive MinUpdateTime problem is precisely to minimize the maximum number of flows in the flow set that traverse the same switch. The difference is that the link capacity constraint is no longer applies since the pre-computing paths instead of routing flflows. Cycle avoidance constraints are also removed to reduce the complexity without affecting the optimal solution, then it can remove cycles in a given solution without increasing the objective value. Thus, the problem of computing candidate paths referred to as the CandidatePath problem.

The second step aims at selecting one path per flow from the candidate paths to minimize the network update time under link capacity constraints. Formulating this problem as optimization requires rewriting the optimization in terms of a new decision variable, which indicates whether path p is selected to carry flow. Ensure that the flow path is the shortest path and the conservation constrain is minUpdateTime.DN can be programmed with the help of python programming whereas SD-WAN is programmed by the vendor. SDN python page helps to enable network functions virtualization (NFV) within a closed system. SD-WAN, on the other hand, offers application routing that runs on an SD-WAN appliance or that can be virtualized. Typically in this SDN environment, customers can see all of their devices and TCP flows, which means they can slice up the network from the data or management plane to support a variety of applications and configurations. The queuing delay the time an SDN controller waits until a batch of flows processed and the network configuration is updated. In highly dynamic environments such as data-centers or IoT deployments where the network is based on SDN, it is beneficial to set the queuing delay to a specified time threshold such as the lower bound of TCP retransmission time.

Intent-based networking (IBN) has a variety of components but basically is about giving network administrators the ability to define what they want the network to do, and having an automated network management platform creates the desired state and enforce policies to ensure what the business wants happens. If a key tenet of SDN is abstracted control over a fleet of infrastructure, then the provisioning paradigm and dynamic control to regulate infrastructure state is necessarily a higher level. The policy is closer to declarative intent, moving away from the minutia of individual device details and imperative and reactive commands.IDC helps that intent-based networking represents an evolution of SDN to achieve even greater degrees of operational simplicity, automated intelligence, and closed-loop functionality. To reconfigure the network under changes in flow demand, consider the following algorithms:

- A randomized rounding algorithm that solves the optimization via linear programming (LP) relaxation followed by randomized rounding.The MILP formulation of the MinUpdateTime problem and its non-disruptive version allows us to leverage standard techniques to the approximation of MILPs.
- A minimax path algorithm that routes the flows sequentially on the feasible path that minimizes the maximum per-rule update time among the traversed switches.
- A shortest path algorithm that routes the flows sequentially such that each flow uses the path with the smallest hop count with sufficient residual capacity to carry this flow.

D.EXPERIOR OF NETWORK

To evaluate dynamic flow demands, flow with a value of one arrive in our network every computation round, while previous flows depart the source and destination nodes of a flflow are selected based on random distribution. A number of flows the candidate paths for every node mean switch pair is used for the restricted algorithms. To emphasize the evaluation of the algorithms and models on network configuration time is less or not which is run with the help of python. After evaluating the metrics associate with each round of the simulation, evaluate the flow configuration algorithms in terms of how promptly they update the network and how well they satisfy the flow demands. The promptness is measured by both the time to compute the set of new rules and the time to deploy these rules at the switches. Based on these metrics, evaluate the total network configuration time total that is the total of configuration time and demand t time. In the simulations, the configuration is directly measured, but demand time is calculated based on the number of updated rules per switch. Then ignore the queuing delay and the southbound communication delay as they are negligible compared to Configuration delay and demand delay. Moreover, since the algorithms also differ in how much flow demand they can satisfy and measure the packet loss in the percentage of the overall demand. To compute the percentage of packet loss, calculate the ratio between the total demand of the dropped flow and the total amount of demand from all flows, in the number of packets. Here, the parameter of destination can be seen as the number of packets transferred by a flow. Then run the algorithms in two different modes. The first one is disruptive, where rules of existing flow can be updated in addition to accommodating new flow and the second one is non-disruptive, where only rules for newly arriving flows are added. In the presented evaluations we use the unrestricted shortest path algorithm as a comparison baseline for our proposed algorithms and models.

In the Rocketfuel topology, the best performing algorithm in the non-disruptive case randomized random rounding; it shows the lowest total configuration time and has a negligible packet loss. The packet loss in the random rounding based algorithms is due to the rounding step which may cause congestion. consider the disruptive case. Unlike the minimax algorithm, which is always non-disruptive, the shortest path and the randomized rounding algorithms show worse computation time in the disruptive case since more flows have to be processed in each round. The deployment time for the unrestricted random rounding algorithm is increasing in the disruptive case in both topologies.

Both baseline algorithms do not perform well in scenarios with failing links. The unrestricted shortest path algorithm shows an increasing configuration time since fewer shortest paths become available with an increasing number of failing links. This causes congestion on a number of paths and results in increased deployment time. Once the network gets close to its saturation, the deployment time drops since only a small number of new flows can be accommodated and fewer disrupted flows recovered while the packet loss increases. The restricted shortest path algorithm shows a decreasing network configuration time caused by significant packet losses since only a restricted number of candidate paths is considered which does not make it a good approach for a scenario with flow disruption. After providing an overview of the average results of the evaluated algorithms. Considering the evaluated metrics total time and deployment time, our unrestricted minimax algorithm shows the best performance in networked environments with failing links.As a trade-off for shorter network configuration time, the algorithms may compute longer paths compared to the shortest path baseline algorithm. This effect is smaller on structured topologies where path lengths tend to be uniform, such as in the case of the fat-tree, while it is more evident for the topologies of the Rocketfuel data-set. expected, the shortest path based algorithms show the smallest path lengths, since their main objective is to minimize the number of hops between a source and a destination node. Our proposed minimax and randomized rounding algorithms follow a different objective and do not minimize the path length between network endpoints. As a trade-off, for faster network configuration time they produce longer paths. On a larger topology, this effect is slightly stronger on average due to the higher path diversity available between different nodes.

III . RECENT STUDIES

Destounis et al[1] proposed a paper "Minimum cost SDNrouting with reconfiguration frequency constraints" model Software-defined network (SDN) controllers which include mechanisms to globally reconfigure the network to respond to a changing environment. As demands arrive or leave the system, the globally optimum flow configuration changes over time. Although the optimum configuration can be computed with standard iterative methods, convergence may be slower than system variations, and hence it may be preferable to interrupt the solver and restart. In this paper, we focus on the class of iterative solvers with an exponential decrease over time in the optimal gap. Assuming dynamic arrivals and departures of demands, the computed optimality gap at each iteration is described by an auto-regressive stochastic process.

S. Brandt et al[2] proposed a paper " On consistent migration of flows in SDNs" studied the problem of migrating flows consistently, that without congestion or rate-limiting, with special focus on software-defined networks.

X.Wen et al.[3] proposed a paper "RuleTris: Minimizing rule update latency for TCAM based SDN switches". They highlighted the RuleTris, the First SDN update optimization framework that minimizes rule update latency for TCAM-based switches. RuleTris employs the dependency graph(DAG) as the key abstraction to minimize the update latency. RuleTris efficiently obtains the DAGs with novel dependency preserving algorithms that incrementally build rule dependency along with the compilation process. Then, in the guidance of the DAG, RuleTris optimizes the rule updates in TCAM to avoid unnecessary entry moves, which are the main cause of TCAM update inefficiency. Here prove that RuleTris generates TCAM updates with the minimum number of TCAM entry moves.

In addition, S. Vissicchio[5], L. Cittadini[6], S. Vissicchio[7], and L. Cittadini [8] studied about Safe, efficient, and robust sdn updates by combining rule replacements and addition. It implies that disruption-free updates are a key primitive to effectively operate SDN networks and maximize the benefits of their programmability. In this needs to study how to implement this primitive safely (with respect to forwarding correctness and policies), efficiently (in terms of consumed network resources) and robustly to unpredictable factors, such as delayed message delivery and processing. First, analyze the fundamental limitations of prior proposals, which either: progressively replace initial flow rules with new ones or instruct switches to maintain both initial and final rules. Second, It shows that safe, efficient, and robust updates can be achieved by leveraging a more general approach.

Indeed unveil a dualism between rule replacements and additions that opens new degrees of freedom for supporting SDN updates. Third, demonstrate how to build upon this dualism. For that propose FLIP, an algorithm that computes operational sequences combining the efficiency of rule replacements with the applicability of rule additions. FLIP identifies constraints on rule replacements and additions that independently prevent safety violations from occurring during the update. Then, it explores the solution space by swapping constraints that prevent the same safety violations, until it reaches a satisfiable set of constraints. Fourth, perform extensive simulations, showing that FLIP can significantly outperform prior work. In the average case, it guarantees a much higher success rate than algorithms only based on rule replacements, and massively reduces the memory overhead needed by techniques solely using rule additions.

IV. CONCLUSION

In this paper the problem of minimizing the network configuration time in an SDN in response to changing demands, by computing a flow configuration that minimizes the worst-case update time across switches. That empirically show that our heuristics can reduce the update time of a shortest-path baseline algorithm on average packet loss. Further, shows that the proposed algorithms are able to reestablish disrupted flflows in scenarios with failed links are faster on average compared to shortest-path based algorithms. The amount of automation can leverage out of a Networking process that can help in various ways. It's the best way to invest speed in the overall networking operations. Offer a centralized view of an organization's entire network, making it easier to streamline enterprise management and provisioning. As VLANs become a more prominent part of physical LANs, the number of links and dependencies can easily create confusion. SDNs can speed service delivery and provide more

REFERENCES

[1] S. Brandt et al., "On consistent migration of flows in SDNs," IEEE INFOCOM, 2016,Page.no 1-14.

[2] A. Destounis et al., "Minimum cost sdn routing with reconfiguration frequency constraints," IEEE/ACM Transactions on Networking, 2018 Page.no 1-13.

[3] T. Mizrahi et al., "Software Defined Networks: It's about time," IEEE INFOCOM, 2016 Page.no 1-9.

[4] X. Wen et al., "RuleTris: Minimizing rule update latency for TCAMbased SDN switches," IEEE ICDCS, 2016Page.no 1-14.

[5] S. Vissicchio et al., "Flip the (flow) table: Fast lightweight policypreserving sdn updates," in INFOCOM 2016 Page.no 1-7.

[6] S. Vissicchio, L. Cittadini, S. Vissicchio, and L. Cittadini, "Safe, efficient, and robust sdn updates by combining rule replacements and additions," IEEE/ACM Transactions on Networking (TON), 2017 Page.no.1-14.