# A Distributed Control Plane For Path Computation Scalability in Software-Defined Networks

Mohammed Amine Togou[1], Djabir Abdeldjalil Chekired[2], Lyes Khoukhi[2], and Gabriel-Miro Muntean[1]

[1]Performance Engineering Laboratory, School of Electronic Engineering, Dublin City University
[2]ICD/ERA, University of Technology of Troyes
[1]{mohammedamine.togou, gabriel.muntean}@dcu.ie
[2]{djabir_abdeldjalil.chekired, lyes.khoukhi}@utt.fr

***Abstract*— Given the shortcomings of traditional networks, Software-Defined Networking (SDN) is considered as the best solution to deal with the constant growth of mobile data traffic. SDN separates the data plane from the control plane, enabling network scalability and programmability. Initial SDN deployments promoted a centralized architecture with a single controller managing the entire network. This design has proven to be unsuited for nowadays large-scale networks. Though multi-controller architectures are becoming more popular, they bring new concerns. One critical challenge is how to efficiently perform path computation in large networks considering the substantial computational resources needed. In this paper, we propose DiSC, a distributed high-performance control plane for path computation in large SDNs. It endorses a hierarchical structure to distribute the load of path computation among different controllers, reducing therefore the transmission overhead. In addition, it uses node parallelism to accelerate the performance of path computation without generating high control overhead. Simulation results show that DiSC outperforms existing schemes, including the most recent ones, in terms of path computation time, path setup latency and end-to-end delay.**

## I. Introduction

With the widespread adoption of smartphones and the global endorsement of 3G/4G technologies, mobile data traffic has skyrocketed. According to Cisco Visual Networking Index (VNI) [1], the global mobile data traffic grew from 4.4 exabytes per month in 2015 to 7.2 exabytes per month in 2016 and is expected to reach 49 exabytes per month by 2021. Given the inherent characteristics of nowadays networks (i.e., extremely expensive, manually configured and lacking dynamic scalability), handling the constantly growing data traffic while ensuring high service quality is becoming complex and very laborious.

SDN is a concept that was proposed to improve network performance and management. It decouples the control plane from the data plane to enable their independent evolution [2]. This separation brings about numerous advantages, including high flexibility, programmability and high scalability. The first proposed SDN systems (e.g., NOX [3] and Floodlight [4]) deploy a single centralized entity, called the controller, responsible for managing traffic flows and monitoring switches of the entire network. Despite their simplicity and ease of implementation, these systems have failed to meet the performance requirements of nowadays large-scale networks [5]. The reason is twofold: 1) with the increase of data traffic, the centralized controller will eventually get overwhelmed, leading to performance degradation; and 2) any minor disruption to the controller's activity may jeopardize the availability of the entire network.

Various techniques have been proposed to mitigate these shortcomings. For instance, [6] and [7] used multiple cores to enhance the Input/Output performance and enabled parallelism to support large networks. [8] and [9] redistributed part of the flow requests among switches to alleviate the controller's load. Alternatively, having a physically-distributed control plane is the approach gaining ground among SDN research community. It consists of partitioning the network into multiple areas, each of which is managed by a distinct controller. ONIX [10], ONOS [11], Google's B4 [12] and Espersso [13] are examples of SDN multi-controller architecture.

In a distributed control plane, each controller maintains topology information,including the set of switches assigned to it (i.e., association information), the set of links between them (i.e., link view) and the path between all-pairs of its respective switches (i.e., path view). Failing to synchronize this information, namely link and path views, among all controllers may lead to inconsistent path computation results. This is referred to as the *synchronization problem*. Yet, synchronizing topology information among all controllers may incur high overhead considering the exorbitant number of control messages to be exchanged, particularly when topology changes are frequent [14]. This might impact the efficiency of path computation, which is pivotal to the performance of various SDN applications [15].

In this paper, we propose DiSC, a distributed control plane that seeks to enhance the performance of path computation in multi-controller SDNs. DiSC embraces a hierarchical architecture that partitions the network into multiple domains, each of which is subdivided into several areas. Each area/domain is managed by a distinct controller. In addition, DiSC defines three types of path computation, based on the destination's location, and identifies the role of each controller in every one of them. This is to ensure rapid and scalable path computation without generating high

978-1-5386-4727-1/18/$31.00 ©2018 IEEE

transmission overhead.

The rest of this paper is organized as follows. Section II surveys some related works. Section III presents our proposed architecture and section IV describes the simulation settings and results. Section V concludes the paper.

## II. Related Work

Few approaches have been proposed to optimize the performance of path computation in SDN. For instance, Kouicem et al. [16] proposed a path computation algorithm for centralized SDNs to optimize flow transmission in WAN environment. It deploys BGP-LS [17] to collect information regarding the underlying WAN (e.g., link states and traffic information) and forward it to the centralized controller. Based on a client-server architecture, the centralized controller uses the received information to respond to applications' path computation requests. Synchronization between the controller and the applications is carried out via exchanging report messages.

Cho et al. [18] described a cloud-based approach that enables SDN controllers to delegate the task of path computation to an application, which can be part of a controller or installed on an external server. This application collects, maintains and updates network information within its corresponding domain. When a path is to be established among multiple domains, each application computes the shortest path within its respective domain and shares it with the other applications of the involved domains. The shortest path is then computed using the backward recursive technique (i.e., path computation starts from the destination and goes backwards till the source node is reached).

Qui et al. [19] proposed Paracon, a parallel control plane that seeks to scale up path computation in SDN. ParaCon endorses a strong consistency synchronization for association and link view information (i.e., all controllers must share this information immediately whenever a topology change has occurred). However, it adopts an eventual consistency policy (i.e., allowing for a delay in synchronization) along with an asynchronous parallel algorithm to synchronize path view information among all controllers. This is to reduce message transmission for path computation, minimizing therefore the overall synchronization overhead and enhancing the path computation performance.

Despite their performance, these solutions have different limitations. While [16] suffers from the bottleneck and single point of failure problems, both [18] and [19] might still incur high control overhead due to their flat architecture (i.e., controllers should maintain the global topology information), especially in large networks where path requests are recurrent. This has motivated us to propose DiSC, a distributed control plane that aims at improving the scalability and the performance of path computation via:

- Endorsing a hierarchical structure that partitions the network in multiple domains, each of which is further divided into several areas. Each area is managed by a single controller.
- Disallowing topology information to be shared horizontally and allowing only part of it (i.e., path view) to be shared vertically .
- Enabling parallelism at the level of controllers in the same or different tiers to accelerate the process of path computation.

## III. Distributed and Scalable Control Plane

This section presents DiSC, the proposed distributed control plane that seeks to improve the scalability and performance of path computation in multi-controller SDNs without incurring high transmission overhead. First the DiSC's overall architecture is described. Then how paths are computed and updated in DiSC is outlined. Finally, some security and reliability issues are discussed and how DiSC addresses them is presented.

### A. The Overall Architecture

Fig. 1 illustrates the DiSC architecture. It has three major components: data plane, topology information abstraction and control plane.

*1) Data Plane:* includes switches and links. When a switch (i.e., source) needs to forward a data packet to another switch (i.e., destination), it sends a path request to its respective edge controller. Based on the destination's location, the edge controller can: a) construct the path view, compute the best path and update the source flow table; or b) forward the path request to the domain controller.

*2) Topology information abstraction:* Both link and path views are abstracted as graphs and are represented using adjacency matrices. The link view consists of switches and their links, mirroring the underlying data plane. The path view, on the other hand, illustrates the weights for all-pairs best paths, within an area or a domain, at a given time. Weights are application-dependent and can involve one or multiple metrics such as available bandwidth, delay and jitter.

*3) Control Plane:* is physically distributed over three tiers, each of which is in charge of performing specific tasks, as described in the following subsections.

*Edge Controllers:* are the entry point to the control plane. They are deployed near end users to enhance the quality of service of various applications. Instead of relaying application requests to remotely deployed controllers, edge controllers can process them locally, enabling therefore quick response times while reducing the computation overhead. Each edge controller is responsible for managing its own area. This includes maintaining and updating topology information and updating flow tables of its assigned switches.

*Domain Controllers:* are responsible for managing their respective domains. This includes supervising edge controllers and assisting them in accomplishing specific tasks. Each domain controller maintains an inter-area gateway table (IAGT) that keeps track of the switches connecting neighboring areas (see Fig. 1). Each entry in IAGT includes the switch's address, the address of the edge controller to which it is assigned (i.e., the area), its status (i.e., normal or special) and the set of inter-area gateways to which it is

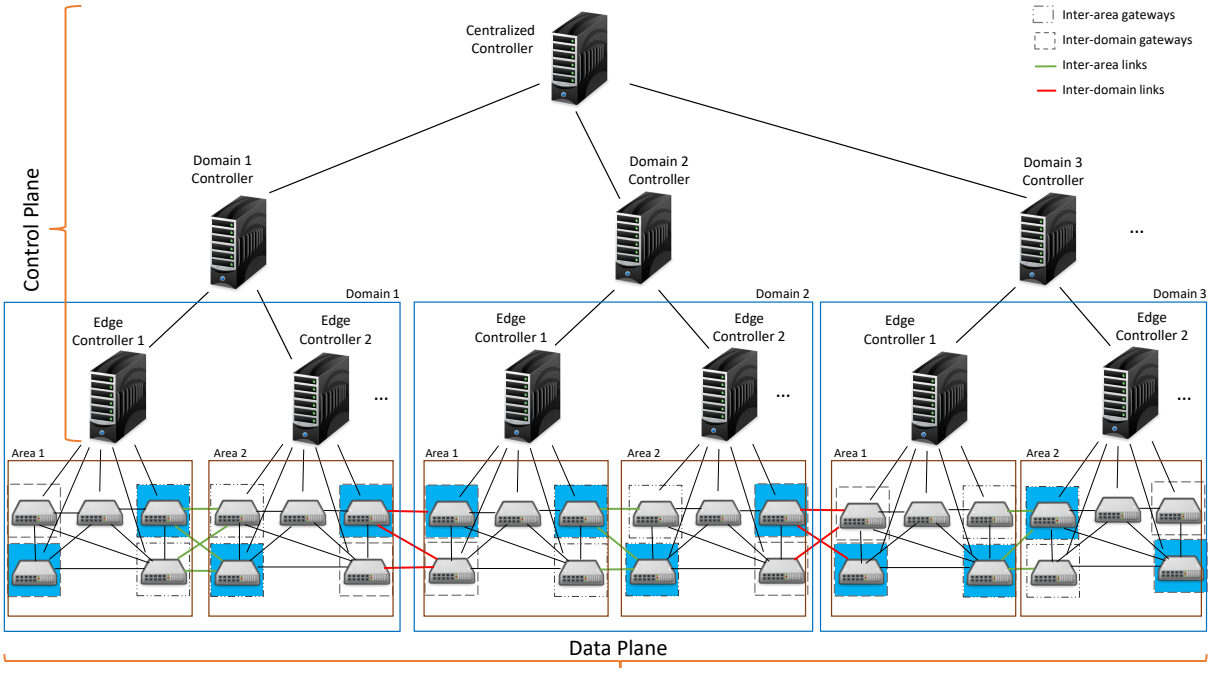


Fig. 1. DiSC architecture. The control plane is made of 3-tiers: edge controller, domain controller and centralized controller.

connected. The use of the status field is described in Section III.D. Domain controllers use IAGT to construct the path view of their respective domain.

*The Centralized Controller:* is responsible for supervising the domain controllers and helping them carrying out their duties. It maintains an inter-domain gateway table (IDGT) that contains the list of switches connecting neighboring domains (see Fig. 1). Each entry in IDGT keeps track of: the switch's address, the address of the domain controller to which it is assigned and the set of inter-domain gateways to which it is connected. IDGT is used by the centralized controller to construct the network's path view. This will be described in detail in the next section.

Note that the DiSC architecture can adapt to network scalability in two ways: 1) by adding new domain and edge controllers (i.e., horizontal scalability) to accommodate the new deployed switches; and 2) by adding a new controller, labeled SDN controller, on top of the centralized controller (i.e., vertical scalability). Under these circumstances, the network will be partitioned into regions. Each region is managed by a centralized controller and is divided into multiple domains, each of which is subdivided into several areas. The SDN controller will be responsible for supervising and assisting the centralized controllers.

### B. Path Computation

Assume that a source switch, labeled source, needs to forward data packets to a destination switch, labeled destination. Based on DiSC, there exists three path computation possibilities: intra-area, inter-area and inter-domain.

*1) Intra-area path computation:* both the source and the destination are within the same area. In this case, the source sends a path request to the area's edge controller. This latter deploys an optimized version of the Bellman-Ford algorithm [20] to construct the area's path view. It then selects the best path between the source and the destination considering the application's requirement (e.g., shortest path, shortest delay, highest bandwidth) and updates the flow tables of all the switches in the selected path.

*2) Inter-area path computation:* the source and the destination are within the same domain, but in different areas. In this case, the source sends a path request to its respective edge controller. Given that the destination is not within its respective area, the edge controller forwards the path request to the domain controller. When received, the domain controller localizes the area to which the destination belongs and requests edge controllers managing the areas between the source and the destination to construct and transmit their path views (i.e., this is done in parallel). These views are combined with the information in IAGT to construct the path view between the source and the destination. The domain controller then selects the best path, taking into account the application's requirements. Finally, it updates the flow tables of all the switches in the selected path with the support of the involved edge controllers.

*3) Inter-domain path computation:* The source and the destination are located in different domains. In this case, the source sends a path request to its respective edge controller, which forwards it to the domain controller. Unable to find the destination in its domain, the domain controller forwards the path request to the centralized controller. Once received, the centralized controller localizes the domain to which the destination belongs and requests the domain controllers monitoring the domains between the source and the desti-

nation to construct and transmit their path views (i.e., by combining IAGT information with path views received from edge controllers). This is done also in parallel. These views are then combined with the information in IDGT to construct the path view between the source and the destination and to select the best path. The centralized controller then updates the flow tables of all the switches in the selected path with the support of all involved domain and edge controllers.

### C. Path Update

Whenever a topology change occurs (e.g. a link between two switches becomes unavailable or link weights are modified), the edge controller is the first to get notified. It starts by updating the link view and computing the new path view of its respective area using *Algorithm 1*, which is based on the centralized algorithm in [19]. Each edge controller maintains a queue, labeled $Q_u$, to store the switches that need to be checked for path weight update. The edge controller starts by dequeuing the node at the head of the queue, labeled $x$, and checks whether the total weights of existing paths can be optimized when including $x$. If yes, $x$'s neighbors are added to the queue; otherwise, the edge controller dequeues the next node in the queue. This process continues until the queue becomes empty.

**Algorithm 1** Path Update Algorithm

**Input:** Link View, $Q_u$
**Output:** Path view

```
while Q_u not empty do
    x ← Q_u.head()
    if path weights are optimized by including x then
        Update wieghts in path view matrix
        Q_u ← x'sneighbors
    else
        x ← Q_u.head()
```

Edge controllers will transmit the new path view to the domain controllers and the centralized controller only when there are either inter-area or inter-domain path computation requests. In this way, we reduce the transmission overhead needed to reflect the changes in topology. Indeed, instead of involving all or most of the controllers in the network whenever a topology change occurs (i.e., the case of [19] and [18]), DiSC can implicate at most $h-1$ controllers, where $h$ denotes the number of tiers in the architecture. This implies that DiSC can incur at most $h-1$ transmissions per topology change.

### D. Security and Reliability

In order to mitigate the problem of isolated areas due to edge controllers' failure (i.e., hardware faults or security attacks), DiSC compels inter-area gateways with the status field set to *special* (i.e., the blue shaded boxes in Fig. 1) to maintain the link view of their respective areas. When an edge controller becomes unavailable, the domain controller will use IAGT to reach one of these gateways to retrieve the link view of the disconnected area. The domain controller will then assign the switches of that area to another edge controller. This latter will update its link view to accommodate the new assigned switches and compute a new path view for its respective area.

In case a domain controller becomes unavailable, the centralized controller uses IDGT to reach the edge controllers within the disconnected domain and retrieve the link views of their respective areas. These areas are then assigned to one or multiple domain controllers, allowing for load balancing between different domains. The involved domain controllers will then update their IAGTs to accommodate the new inter-area gateways. By doing so, DiSC can ensure service continuity while strengthening the system's security.

## IV. Performance Evaluation

In this section, we present a simulation-based evaluation of DiSC and compare it to the state of the art solutions: POX [21], a widely used controller in SDN research community, and ONOS v1.3, a popular and stable distributed OpenFlow controller, ParaCon [19].

### A. DiSC Implementation

We implemented DiSC using virtual machines (VM) as controllers, each one is implemented on Dell OptiPlex 7050 (Intel Core i5 CPU 2.71 GHz with 08G RAM). We have 10 VMs, each of which is used as a controller. Three DisC versions were implemented:

- DiSC-One(i): consists of one domain and a variable number of areas *i* (i.e., from 2 to 8).
- DiSC-Two: consists of two domains, each of which contains two areas (i.e., two edge controllers). Both domains are managed by one centralized controller.
- DiSC-Three: consists of three domains. The first has one area, the second has two while the third has four areas. All domains are managed by one centralized controller.

The data plane topology for each area is provided by Mininet. Each controller is based on a modified POX where we replace the model of the path computation by our proposed path computation mechanism.

### B. Simulation Results and Analysis

Fig. 2(a) depicts the path computation time with respect to the number of switches in each area. We observe that the path computation time increases with the increase in the number of switches. We also observe that DiSC variants incur the shortest path computation time compared to all other schemes. For instance, DiSC-Three achieves an average path computation time that is $133\%$, $169\%$ and $180\%$ shorter than Paracon, ONOS and POX, respectively. Instead of maintaining a global topology information (i.e., the case of ParaCon and ONOS) and sharing it with all controllers, DiSC compels edge controllers to sustain topology information of their respective areas and share only part of it (i.e., path view) with their respective domain controllers. In this way, edge controllers can quickly compute the path view of their areas while reducing the transmission overhead.

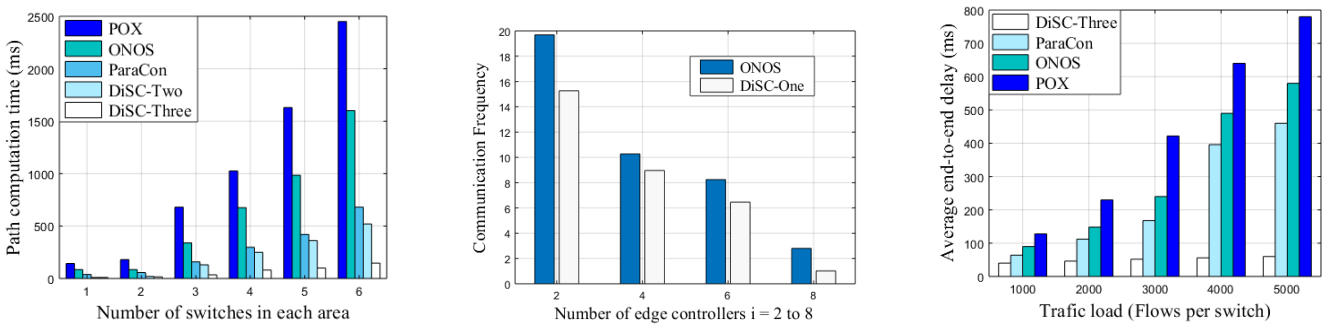


Fig. 2. Performance evaluation of the various schemes in terms of: a) path computation time; b) communication frequency; and c) end-to-end delay

Fig. 2(b) illustrates the communication frequency as a function of the number of edge controllers. By communication frequency we mean the number of times that edge controllers were solicited. We observe that the communication frequency decreases with the increase in the number of controllers. This is rational since augmenting the number of edge controllers implies fewer switches per area. We also observe that DiSC-One outperforms ONOS (i.e., DiSC-one incurs an average communication frequency that is 25% lower than ONOS's). This is because ONOS adopts a flat architecture that requires each controller to inquire all the other controllers to construct and maintain the network's global topology information. DiSC, however, defines three path computation scenarios and involves only controllers that are concerned. For instance, in case of an inter-area path computation, DiSC compels edge controllers to compute the path view of their respective areas and implicates domain controllers to build the path view of the respective domains. In this way, edge controllers do not become overwhelmed and their resources can be used to handle new intra-area path computation requests.

Fig. 2(c) shows the end-to-end delay as a function of traffic load. We observe that except for DiSC-Three, the end-to-end delay of all schemes increases with the increase in traffic load. As expected, POX generates the highest end-to-end delay since it uses a single centralized controller. Even though ONOS and ParaCon are distributed multi-controller architectures, they still incur high end-to-end delay. In fact, both ONOS and ParaCon incur an average end-to-end delay that is 130% and 143% higher than DiSC-Three's. The reason is that DiSC computes paths quickly compared to the other schemes as it distributes traffic load over different controllers according to the type of the path request (i.e., intra-area, inter-area and inter-domain) and allows for the sharing of path views with upper-tier controllers only, therefore avoiding network congestion.

To examine the path computation scalability of DiSC and ParaCon, we set up three distinctive topologies, as illustrated in Table I. *CERNET* and *USCarrier* are provided by Topology Zoo [22]. As the files of the two topologies were in GML/GraphML format, we used Python and the iGraph library to convert these files into an adjacency matrices and edge lists. We have also designed a scalable topology and named it *ScTop*. While *CERNET* is a small topology with a small diameter, *USCarrier* is a small topology with a large diameter. *ScTop* is a large topology with a small diameter, involving a massive number of switches.

TABLE I
SIMULATED TOPOLOGIES

| Topology | Node | Link | Diameter |
|---|---|---|---|
| CERNET | 41 | 57 | 6 |
| USCarrier | 158 | 189 | 35 |
| ScTop | 600 | 30000 | 5 |

Fig. 3 depicts the path setup latency with respect to the number of path computation requests for both DiSC and Paracon in the various topologies. We observe that the path setup latency decreases with the increase in the number of path computation requests. We also observe that ParaCon incurs the highest path setup latency in all topologies. For instance, ParaCon achieves an average path setup latency that is 54%, 72% and 135% higher than DiSC-Three's in *CERNET*, *USCarrier*, and *ScTop*, respectively. The fact that ParaCon requires all controllers to maintain the global topology information makes its path computation performance contingent to the graph's diameter alongside the number of switches and links in the network. Indeed, while ParaCon performs as good as DiSC-Two when the diameter and the size of the network are small (i.e., see Fig. 3(a)), its performance is dreadful in large size networks with small diameter (i.e., see Fig. 3(c)).

DiSC does not endure this problem as its variants incur a stable path setup latency (i.e., see Fig. 3(c)). The reason is threefold: 1) DiSC endorses a hierarchical multi-tier control plane that requests controllers to maintain topology information for their respective areas or domains only; 2) DiSC also allows for path views to be shared with upper-tier controllers; and 3) DiSC enables load balancing among controllers by defining different types of path computation and involving different controllers in each one of them. Observe that DiSC-Two incurs a high path setup latency (i.e., 46%) compared to DiSC-Three in *CERNET*. This is because DiSC-Three has areas with smaller sizes, implying that edge controllers can rapidly construct their path views. When the size of these areas increases, DiSC-Two incurs an average path setup

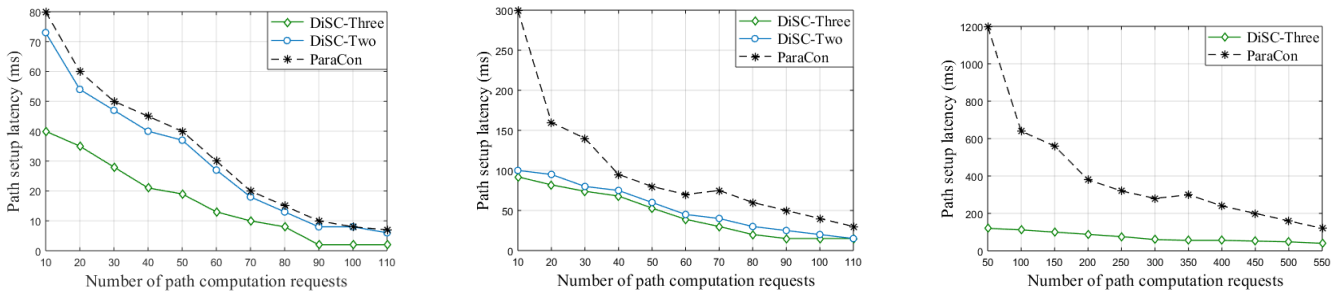


Fig. 3. Path setup latency in terms of path computation requests in: a) CERNET topology; b) USCarrier topology; and c) Scalable topology

latency that is only 15% higher than DiSC-Three's (i.e, see Fig. 3(b)).

## V. Conclusion

In this paper, we propose DiSC, a hierarchical and distributed control plane that improves the performance of path computation in SDN. DiSC partitions the network into domains, each of each is subdivided into areas. Each area is managed by an edge controller and each domain is supervised by a domain controller. All controllers are responsible for maintaining all or part of the topology information. DiSC deploys a centralized controller to monitor the domain controllers and to build the network's path view. It also defines three types of path computation and identifies the role of each controller in every one of them. This is to accelerate the path computation process, using parallelism, and to balance the load (i.e., path computation request) among all controllers. Simulation results show that DiSC outperforms existing schemes in terms of path computation time, path setup latency and end-to-end delay.

For future work, we will examine the performance of DiSC in scenarios with significant traffic load. We are also planning to consider scenarios where topology changes are recurrent and propose a mechanism to quickly adapt to these abrupt changes without the need for computing the paths once again.

## Acknowledgement

This work was supported by the European Union's Horizon 2020 Research and Innovation program under Grant Agreement no. 688503 for the NEWTON project (http://newtonproject.eu).

## References

[1] "Cisco visual networking index: Global mobile data traffic forecast update, 2016-2021 [white paper]," https://www.cisco.com/mobile-white-paper-c11-520862.html, March 2017, accessed: 2018-04-05.

[2] F. Bannour, S. Souihi, and A. Mellouk, "Distributed sdn control: Survey, taxonomy, and challenges," *IEEE Communications Surveys Tutorials*, vol. 20, no. 1, pp. 333–354, Firstquarter 2018.

[3] N. Gude *et al.*, "Nox: Towards an operating system for networks," *SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 3, pp. 105–110, Jul. 2008.

[4] "Floodlight project," http://www.projectfloodlight.org/floodlight/, accessed: 2018-03-05.

[5] R. Trestian, K. Katrinis, and G. M. Muntean, "Ofload: An openflow-based dynamic load balancing strategy for datacenter networks," *IEEE Transactions on Network and Service Management*, vol. 14, no. 4, pp. 792–803, Dec 2017.

[6] A. Tootoonchian *et al.*, "On controller performance in software-defined networks," in *Proc. of the 2Nd USENIX Conference on Hot Topics in Management of Internet, Cloud, and Enterprise Networks and Services*, ser. Hot-ICE'12, 2012, pp. 1–10.

[7] Z. Cai, A. L. Cox, and T. S. E. Ng, "Maestro: A system for scalable openflow control," 2010.

[8] M. Yu, J. Rexford, M. J. Freedman, and J. Wang, "Scalable flow-based networking with difane," *SIGCOMM Comput. Commun. Rev.*, vol. 41, no. 4, pp. 351–362, Aug. 2010.

[9] A. R. Curtis *et al.*, "Devoflow: Scaling flow management for high-performance networks," in *In ACM SIGCOMM*, 2011.

[10] T. Koponen *et al.*, "Onix: A distributed control platform for large-scale production networks," in *Proc. of 9th USENIX Conf. on Operating systems design and implementation (OSDI'10)*, 2010.

[11] P. Berde *et al.*, "Onos: Towards an open, distributed sdn os," in *Proceedings of the Third Workshop on Hot Topics in Software Defined Networking*, ser. HotSDN '14, 2014, pp. 1–6.

[12] S. Jain *et al.*, "B4: Experience with a globally-deployed software defined wan," *SIGCOMM Comput. Commun. Rev.*, vol. 43, no. 4, pp. 3–14, Aug. 2013.

[13] K.-K. Yap *et al.*, "Taking the edge off with espresso: Scale, reliability and programmability for global internet peering," in *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, ser. SIGCOMM '17, 2017, pp. 432–445.

[14] G. DeCandia *et al.*, "Dynamo: Amazon's highly available key-value store," *SIGOPS Oper. Syst. Rev.*, vol. 41, no. 6, pp. 205–220, Oct. 2007.

[15] M. Wichtlhuber, R. Reinecke, and D. Hausheer, "An sdn-based cdn/isp collaboration architecture for managing high-volume flows," *IEEE Transactions on Network and Service Management*, vol. 12, no. 1, pp. 48–60, March 2015.

[16] D. E. Kouicem, I. Fajjari, and N. Aitsaadi, "An enhanced path computation for wide area networks based on software defined networking," in *2017 IFIP/IEEE Symposium on Integrated Network and Service Management (IM)*, May 2017, pp. 664–667.

[17] O. G. de Dios *et al.*, "First multi-partner demonstration of bgp-ls enabled inter-domain eon control with h-pce," in *2015 Optical Fiber Communications Conference and Exhibition (OFC)*, March 2015, pp. 1–3.

[18] H. Cho, J. Park, J.-M. Gil, Y.-S. Jeong, and J. H. Park, "An optimal path computation architecture for the cloud-network on software-defined networking," *Sustainability*, vol. 7, no. 5, pp. 5413–5430, 2015.

[19] K. Qiu, S. Huang, Q. Xu, J. Zhao, X. Wang, and S. Secci, "Paracon: A parallel control plane for scaling up path computation in sdn," *IEEE Transactions on Network and Service Management*, vol. 14, no. 4, pp. 978–990, Dec 2017.

[20] T. H. Cormen, C. Stein, R. L. Rivest, and C. E. Leiserson, *Introduction to Algorithms*, 2nd ed. McGraw-Hill Higher Education, 2001.

[21] "Sdn hub, pox controller tutorials," http://sdnhub.org/tutorials/pox/, accessed: 2018-04-05.

[22] S. Knight, H. X. Nguyen, N. Falkner, R. Bowden, and M. Roughan, "The internet topology zoo," *IEEE Journal on Selected Areas in Communications*, vol. 29, no. 9, pp. 1765–1775, October 2011.