ArchSDN: a reinforcement learning-based autonomous OpenFlow controller with distributed management properties

Carlos Ferreira¹ · Susana Sargento¹ · Arnaldo Oliveira¹

Received: 24 October 2019 / Accepted: 8 August 2020 / Published online: 19 August 2020
© Springer Nature Switzerland AG 2020

Abstract

The scaling capability of network controllers is important for network operators to have the confidence to continue to expand the communication network. Today's networks can use the Software Defined Networks (SDN) concept to separate the control and the switching planes, and use autonomous self-management features to quickly react to any network event, while providing the network with necessary agility to support custom Service Level Agreement (SLA) to its clients, while lowering the capital and operational expenditures. Both SDN and autonomous management concepts have a centralized architecture, requiring a single entity managing every aspect of the network, making it difficult to manage an entire network in the massive IoT world, where services tend to be provided close to the users, and edge-based approaches are key to the support of 5G and beyond services. The support of such dynamic and flexible network requires an approach where the network management responsibilities are distributed through multiple autonomous OpenFlow controllers, which we denote as ArchSDN controllers. By assigning OpenFlow switches to different ArchSDN controllers, the network becomes divided into different sectors, each controlled exclusively by one ArchSDN controller. These controllers, coordinate their actions and use a reinforcement-learning based decision mechanism to explore, learn and implement near-to-optimum end-to-end communication paths. The evaluation results show that the proposed decision system is capable of finding near-to-optimum solutions, learn from the obtained results to improve future service activation results, and quickly adapt the network to the loss of a communication link by responding in less than 100 ms.

Keywords OpenFlow · Reinforcement learning · Distributed management · Routing · Software defined network

1 Introduction

Network management is a difficult task to be maintained by network operators. The capital and operational expenditures are a major concern for network operators, which are required to maintain a high-availability service for their clients, in order to fulfill the SLAs.

Current network services rely upon proprietary appliances and equipments, which are expensive, but essential to maintain the SLA with network operator's clients. This dependence has led to what is known as the ossification of the Internet, where most aspects of a network operation

are fixed and rigid, and the costs of changing minimal parts, such as the introduction of new protocols, becomes so expensive that network operators opt to maintain the current protocols and specifications. Therefore, the difficulty to properly manage a high-availability network, the need to reduce capital and operational expenditures and the need to scale to new types of services, has motivated operators to improve the network management operations and to decouple the network functions from proprietary solutions.

Improving the network management automation allows the reduction of network services

✉ Carlos Ferreira, cmf@ua.pt; Susana Sargento, susana@ua.pt; Arnaldo Oliveira, arnaldo.oliveira@ua.pt | ¹Network Architectures and Protocols (NAP) Group, Instituto de Telecomunicações, University of Aveiro, Aveiro, Portugal.

misconfiguration and downtime caused by human error and by non-deterministic events [1]. By taking leverage on self-* properties, autonomous management systems can be used to provide a constant monitoring over the network events, analysing and detecting anomalous situations which may pose any threat to the normal network operation, and properly reacting to such situations by reconfiguring the network equipments and services. Providing a higher level of automation (the knowledge plane [2]) to the network management systems reduces the management costs. However, this is not enough, since the network functions still depend on proprietary hardware.

SDN is able to enforce network management agility and low-cost operation [3] by separating the forwarding plane from the control plane, merging the control with the centralized autonomous management system and removing the network functions from the network equipments. The separation of the forwarding and control planes opens the possibility to develop and test new network protocols for new types of network services. To address the necessity for the network to provide the required reconfiguration agility, the OpenFlow specification was developed [4]. OpenFlow specifies an open control protocol for network controllers to configure filtering rules with detailed modification actions in different network switches from different hardware vendors. By establishing an open control specification, OpenFlow allows hardware vendors to implement their own OpenFlow compatible hardware without requiring to open the underlying hardware or software implementation, while granting the network operators with the assurance that the switches will remain compatible when used with other OpenFlow controllers. A complement to OpenFlow, P4 [5] a high-level language for programming protocol, was later developed to work in conjunction with SDN control protocols such as OpenFlow, with the intention of providing a higher refinement for existing communication protocols and new experimental protocols.

Both SDN and autonomous management help network operators to improve their control over the network [6], but the control network scalability still remains an issue [7, 8]. The growth of the network in the number of clients and equipments is a concern regarding the capabilities of a central management system being able to respond efficiently to network events [9]. Instead of having one massive controller which gathers the state of every network equipment, it would be preferable to distribute the network control through multiple controllers, not just due to scalability issues, but also due to the new trends of edge approaches noticing that, in future networks, services tend to be provided close to the users, and edge-based approaches are key to the support of 5G and beyond services.

In this paper we propose an approach to split the network into sectors and distributing the management load through multiple network controllers, where each controller autonomously manages one sector, which is an exclusive partial view of the complete communication network. The distribution of the network control can be further enhanced by extending the controller capabilities with autonomous self-management functions, in order to provide an increased level of intelligence. To demonstrate the viability of this distributed and autonomic SDN management architecture, this paper proposes ArchSDN, an autonomous distributed controller using the OpenFlow specification. The reason of using OpenFlow as the control protocol is based on its specification maturity, protocol matching capabilities, the amount of research projects using OpenFlow with success, and also its successes in the data centers environment [10, 11]. OpenFlow already contains the necessary control to reconfigure IP traffic flows and network configuration protocols such as DHCP and ARP, providing the necessary features to establish end-to-end communication between the clients, services and devices.

ArchSDN follows a distributed management model where each OpenFlow controller exclusively controls one sector of the network. To establish end-to-end communication for the network clients, the ArchSDN controllers cooperate and coordinate with each other to explore and implement end-to-end communication paths, capable of crossing multiple network sectors if required. Instead of exhaustively searching all possible communication path combinations in the network and using the best one, ArchSDN models this problem using a Markov Decision Process (MDP) and uses a reinforcement learning based mechanism, the SARSA algorithm, to explore and learn which network paths are the best to use, taking into account both the path length and the service bandwidth consumption, and improving future path calculation.

The experimental results demonstrate the viability of the ArchSDN controller proposal, as an alternative to the traditional SDN controller approach: (1) they show how the reinforcement learning based decision system is capable of finding near-to-optimum solutions for the activation of communication paths between network hosts, and learn from the obtained results to improve future service activation results; (2) the decision system is able to adapt to unforeseen circumstances, such as the loss of a network link, being able to discover alternative paths while continuously exploring and improving with better alternatives to react to the loss of the network links; (3) the ArchSDN controller is capable of activating communication service between two hosts in less than 100 ms.

The structure of this paper is as follows. Section 2 provides an overview of the related work, which is compared

to the objectives of the architecture here proposed. Section 3 describes the architecture of ArchSDN and the autonomous decision mechanism for the network communication paths. In Sect. 5, the decision quality of the ArchSDN controller is evaluated by analysing the experimental results. Finally, Sect. 6 provides the final remarks and an overview of different possible paths for future ArchSDN enhancements.

2 Related work

Since the dawn of the Internet, management has always been the cornerstone of telecommunication networks. The immense diversity of network protocols in the control and forwarding planes has created extreme complexity challenges for management, not being able to cope with the growth of traffic volume and the multi-service complexities [12, 13]. With the emerging of new services and increased demand by the clients to lower the service costs, communication networks are now required to be agile and intelligent enough to cope with the future network challenges, while reducing the capital and operational expenditures at the same time. The SDN concept brought a different approach to the network control architecture, by enabling programmability, management agility and by providing a better test-ground for the deployment of new protocols. Such capabilities are the benefit of having a centralized control plane architecture, but despite the simplicity of a top-down network control approach, the issue of the control network scalability and reliability still remains to be properly addressed [7]. This has led research to look into better methods on how to physically distribute the control plane and improve its reacting capabilities, while remaining logically centralized. Beyond distributing the control through multiple entities, the autonomous management concept is seen as an option for the scalability of the network control plane.

The autonomic networking concepts aim to bring the autonomic computing principles to communication networks, mainly on management policies which can be disseminated through centralized [14] or decentralized [15, 16] approaches, depending on how complex the managed scenarios are [17]. Such management policies empower the network components to autonomously adjust and adapt to the network environment changes, according to higher level goals [18]. The need for decentralizing the control and management systems to reach self-adaptability is not new, and has already been verified in the past [19, 20].

Self-management systems could also be enhanced by upgrading the management loop with machine learning capabilities [21]. For example, reinforcement learning [22]

concepts could be used to develop autonomous management systems. Self-healing mechanisms use the gathered information from their sensors and from the network itself, such as measurements, probing, tests, to analyse network problems, develop a solution and act to solve the problem automatically [23, 24].

The next sub-sections will deal with distributed SDN and autonomous management.

2.1 Distributing SDN control

The analysis of multiple SDN controllers in recently conducted surveys [7, 8] identified two different types of implementation designs for logically centralized control: (1) the single controller architecture where one machine controls the entire network; and (2) a distributed controller architecture where multiple controllers are used, sharing the load of the network control. The survey also conducted an analysis regarding the scalability of these approaches, taking into account the data synchronization model and the controller coordination model used by each one.

The single centralized controller approach, despite using a simpler architecture with a predictive model and capable of slicing the network into multiple virtual networks, still suffers from scalability issues when the network is composed by hundreds of network switches, where the controller becomes overloaded and incapable of responding in real-time. Examples of such centralized controller approach are FlowVisor [25], ADVisor [26], VeRTIGO [27] and OpenVirtex [28, 29].

The distributed approach is more complex to implement, requiring data synchronization and actions coordination between controllers. The works in [7, 8] identified two different models for data synchronization: the hierarchical model and the flat model.

The hierarchical model stipulates that at least one controller knows the global network state, from where the other controllers will access it when needed. Examples of decentralized approaches using the hierarchical model are IRIS [30], Decentralize-SDN [31], Kandoo [32] and FlowBroker [33]. The flat model considers that every controller knows the global network state, requiring constant synchronization using pooling or publish/subscribe mechanisms. Examples of flat approaches using the flat model are ONOS [34], East-West Bridge [35], Elasticon [36], DISCO [37], Hyperflow [38], ONIX [39], Controlling SDN [40] and Ravana [41].

Both models are seen as inefficient. The hierarchical model requires the centralization of an accurate global network state in one selected controller. The root controller and leaf controllers pool the root controller to obtain the necessary information to configure the switches under their control. On the other hand, the flat model creates

copies of the global network state in every network controller, propagating changes through the control network when the state of configuration of a domain changes, which is unlikely to scale.

With respect to the controller coordination model, two different types of coordination in distributed systems are addressed in [7, 8]: Leaderless and Leader-based approach.

With the leaderless approach, there is no leader, where every controller in the network has the same management privileges. Controllers configure switches in other domains, by serializing and sending the OpenFlow configuration messages to the domain controller. For configurations to be implemented with success, the data in the controllers needs to be accurate, so this approach is used together with the data flat model. Examples of controllers using the leaderless approach are the East-West Bridge [35], DISCO [37] and Hyperflow [38]. The leader-based approach requires the election of a control hierarchy with 2 or more levels of leadership, which depend upon the size of the network.

Examples of controllers using the leader-based approach are the Elasticon [36], ONOS [34], Decentralize-SDN [31], IRIS [30], ONIX [39], Kandoo [32], Controlling SDN [40], FlowBroker [33] and Ravana [41] controllers.

Two important approaches worth of mentioning which have taken an attempt at solving the scaling challenges are DIFANE [42] and DevoFlow [43]. DIFANE uses a similar strategy followed by the proposed approach in this paper. It scales by using intermediary switches called Authority switches, to which the central controller distributes the rules evenly by using a partitioning algorithm. The strategy followed by DIFANE prevents the controller from being overwhelmed by packets sent by the switches, being handled instead by the authority switches. DevoFlow instead of following a partition scheme, assumes it is not possible to maintain a complete view over the network and severs the coupling between control and the global view, while maintaining the minimum amount of structure information to maintain a proper operation. DevoFlow achieves this scheme by aggressively using wild-carded OpenFlow matching rules.

2.2 Distributed autonomous management in SDN

SDN needs to have an agile control, capable of quickly reacting to network critical events and adapting to network clients' needs, and the autonomous management concept has the necessary self-management properties to address the challenges. In fact, the GANA architecture reference model [44] already proposes the merge of self-* attributes with OpenFlow enabled networks.

In an OpenFlow enabled network, the core idea is to manage packet streams, commonly known as flows.

OpenFlow controllers are then responsible for managing and maintaining the optimum behaviour of these flows. This is simple when such flows start and end within the same domain, but the flow provision through multiple domains requires a more detailed and continued attention to the flow Quality of Service (QoS). One possibility to address such issue is using real-time autonomous decisions [45]. Network partitioning could also be used to distribute management across sub-domains [46]. Efforts for management unification in both wireless and wired networks are also present for SDN [47]. Also, the trend to unite autonomous management and SDN concepts spawned several research projects, such as AMA [48], SeNET [49], SDNVA [50], AQSDN [51], Beacon [40], ONOS [52], Pratyastha [53], SMaRtLight [54], SDCoN [55], AUTOFLOW [56], SELFNET [57].

Distributed controllers usually offer weak consistency semantics, while controllers with a strong consistency offer a simpler interface to application developers, but not without an impact on system performance. To date, only Onix [39], ONOS [52], Beacon [40] and SMaRtLight [54] provide a strong consistency model. A distributed controller can improve the control plane resilience, scalability and reduce the impact of problems caused by network partition, but not without suffering with performance issues related to the data synchronization, when controlling large clusters of equipments [47]. Still, Beacon [40] results show that the proposed distributed controller is capable of achieving scalability and reliability, even when the controlled network is under heavy load.

Schemes like SDN-DMM (Distributed Mobility Management) [58] provide the necessary solution to address the issues of dynamic routing and IP addresses allocation in environments with distributed mobility management. Driven by the similarities that exist between SDN and DMM, merging both concepts is seen as a solution capable of bringing numerous benefits. The SDN architecture with DMM (S-DMM) [59] provides benefits such as no need to deploy any mobility-related component at the access router, independence of the underlying technologies, and per-flow mobility support. By extending DMM, a Three-Tiered SDNing architecture [60] provides a solution to solve mobility problems and handover in femto-cells and macro-cells, by reducing the delay and increasing throughput over cellular networks. Another example of a 5G network's enhancement with distributed mobility management, is the use of an SDN architecture based on IPv6 protocol with routing header fields [61], to allow the movement between domains and technologies without requiring tunnel mechanisms.

Last but not least, the management of network functions for distributed monitoring is an important factor for the correct operation of SDN-based networks. An

extended analysis was already performed [62], with some proposals for proper data synchronization. However, despite the large amount of contributions in the autonomous management context for communication networks, the proposed models focus its use in centralized management systems. Even the proposals of ONOS [52] and Pratyastha [53] which propose the distribution of work through several systems, the set of computational nodes is still considered and viewed as a centralized management system, controlling the entire network.

2.3 Discussion

This paper proposes a distributed control model using a hybrid mix between the leaderless and leader-based approaches, implemented as the ArchSDN controller. The use of a hybrid approach is not entirely new. For example, the Orion [63] controller follows a mix of hierarchical and domain control, where controllers see an abstract view of the global network, knowing only the details of the domain it handles. Orion also uses a distributed database implemented using NoSQL to establish communication between controllers. Recently, an attempt to evaluate the scalability of Orion was performed, but unsuccessful due to the lack of access to the controller implementation [64].

What differentiates the ArchSDN from other distributed approaches is the capability to establish control over the network without requiring to synchronize the global network state through all of the network controllers. Instead of having the network controllers observe the full network status, ArchSDN sub-divides the network into sectors, by assigning a specific cluster of switches to connect to a specific controller. The segmentation of the network into isolated sectors reduces the amount of information that the controllers need to analyse. Also, by reducing the number of switches connecting to a controller, the number of events the controller needs to respond are also reduced. Therefore, ArchSDN aims to follow the same path of Orion [63] controller, but without requiring the need for neither the existence of an abstract view of the communication network in all of the network controllers, nor the need to use a NoSQL database to establish communication between controllers.

However, segmenting the network into isolated sectors cannot be performed without addressing the fact that the controller only sees a partial view of the network, without information beyond the sector. ArchSDN controllers provide communication services to clients in different sectors, by implementing cross-sector communication paths. However, it is still needed that the controller wanting to implement a specific service in multiple sectors needs to explore communication paths in the network. This path exploration requires a higher level of automation and intelligence

for the controller to be capable of learning from the result of each exploration and improving the quality of future explored communication paths. The level of automation required for ArchSDN has already been used with success for mesh-networks with DistArch [65], which uses the quality of the explored communication paths to reinforce future decisions. ArchSDN aims to push these efforts into the SDN context.

3 ArchSDN network architecture

The ArchSDN network architecture is designed to favour the network scalability, by using a hybrid management control structure. As depicted in Fig. 1, the network structure is splitted into multiple sectors, each managed by a single controller.

At the top of the control hierarchy there is the central manager which contains the network policies. Each sector controller queries the central network policies from the central manager and applies them to their respective sectors, where the aggregate of the network controllers is denoted as the Distributed Control Layer. Each sector is composed of the network controller, the OpenFlow switches and the sector hosts, where the aggregate of the switches is denoted as the Switching Layer. To establish end-to-end communication for the network clients, the ArchSDN controllers cooperate and coordinate with each other to explore and implement end-to-end communication paths, capable of crossing multiple network sectors if required. ArchSDN uses a reinforcement learning based mechanism to explore and learn which network paths are the best to use, taking into account both the path length and the service bandwidth consumption, and improving future path calculation. The next sub-sections detail the architecture and its functionalities.

Fig. 1 Network structure

3.1 ArchSDN central manager

The central manager is the entity that implements the network operator decisions and the network address pool registry policy. It provides the ArchSDN controllers with the means to register the location of the network hosts, and also to retrieve unique IP addresses to answer the hosts Dynamic Host Configuration Protocol (DHCP) requests. The ArchSDN controllers also use the central manager to retrieve information regarding the location of hosts in other networks, based on their IP address.

Although this entity is unique in the network, it is perfectly possible to replicate the ArchSDN central manager and embed it in each ArchSDN controller. Previously published work on dissemination of information through multiple agents in a network provides the base-ground for such replication [66–68].

3.2 Sector topology management

The distributed nature adopted by the ArchSDN controller to manage the networks requires the definition of which are the elements composing the overall network. Since the ArchSDN architecture splits the network into sectors, each controller has a specific, partial view of the network, which is the sector it controls.

The sector view is composed of 4 different types of entities (Fig. 2):

- Local controller: unique within the sector.
- Local switch: one or more within the sector.
- Local host: one or more within the sector.
- Adjacent sector: zero or more connected one or multiple times using different border links, to one or more local switches.

The sector topology is composed of local switches and local hosts. To discover the local sector topology, the local controller must implement an internal discovery process which will allow it to discover the links between the local switches, and the border links connecting the local switches to the adjacent sectors. It is important for the local sector to know which adjacent sectors it will have access to, since it will be through them that the local controller will establish end-to-end communication from the local hosts to hosts located in others sectors. This process of discovering local switches, local hosts and adjacent sectors is continuous and permanent.

To implement the topology discovery process, the local switches are reconfigured immediately after connecting with the necessary flows for the process. These flows will filter specific packets, allowing the local controller to accurately detect the network links in the local sector topology,

Fig. 2 Sector internal structure

and also allows the detection of new local hosts connecting to the local switches.

3.2.1 Discovering adjacent sectors and border links

To discover the links between local hosts and from local hosts to the adjacent sectors, the controller injects into every connected port of every local switch, a MAC layer packet with specific payload information, using the ethertype (0xAAAA) with the aim to be easily filtered by the switches. The discovery packet payload contains the identification of the sector represented as the controller ID, and an encoded MD5 [69] hash value of the switch data path ID and the port ID concatenation (Fig. 3a), used as the key in a hash table to map the switch data path ID and port ID data pair.

When sent through the port switch, the discovery packet is captured by the receiving switch and sent to the controller, which analyses the controller ID field to know the origin of the packet. If the controller ID field is the same as the ID of the receiving controller, this means that the discovery packet was sent by the controller itself and that both sending and receiving switches belong to the local sector, and the link which was used is a link of the local sector (Fig. 3b). The controller uses the MD5 hash value to query the hash map to know which switch and port was used to send the discovery packet, to register the link in the local sector topology. However, if the controller ID field is different, then it means that the discovery packet was sent by an adjacent controller using a switch under

Fig. 3 Sector topology discovery. **a** Discovery packet payload; **b** local switch to local switch; **c** local switch to adjacent sector

its control and connected through a border link to a local switch. In this case, the local controller registers the link as a border link connected to the adjacent sector with the ID provided in the discovery packet (Fig. 3c).

The hash value present in the discovery packet is used by the local controller to identify the border link in the local topology. When establishing communication paths

that cross the adjacent sector, the local controller will communicate with the adjacent controller and use the hash value to uniquely identify the border link which should be used to implement the communication path. This is necessary because the local controller has no information about the topology of the adjacent sectors, requiring a way to distinguish between different border links connecting the local sector to an adjacent sector, during the activation of communication paths.

3.2.2 Discovering local hosts

To detect new local hosts connecting to the local switches, the controller implements flows at the sector OpenFlow switches to capture and redirect DHCP discovery packets sent by new local hosts, during their request for IP network addresses. This allows the local controller to register the location of new local hosts, at the ArchSDN Central Manager. After registering the local hosts, the local controller will answer to the local hosts with a DHCP Offer packet, containing the IP address offer and the remaining network configuration information.

3.2.3 Controller cooperation

Controller cooperation is essential to implement an end-to-end service in the ArchSDN architecture. ArchSDN controllers communicate with each other using a dedicated control network, as seen in Fig. 1, to implement an end-to-end service. Controllers use peer-to-peer communication to coordinate with each other. When the controller service activation is triggered, the controller evaluates the requirements of the service, and if the target host is not located within the local sector, it contacts the ArchSDN Central Manager to obtain the sector ID where it is located.

3.3 Communication paths

Similar to other traditional centrally managed networks, the implementation of communication paths requires the configuration of routing policies in the network equipments to flow the network traffic from one end to the other. In the ArchSDN architecture, such routes are known as communication paths.

The ArchSDN controller activates communication paths (Fig. 4) by encapsulating packets with MultiProtocol Label Switching (MPLS) headers, being essentially a process which activates MPLS paths. MPLS allows the controllers to implement in-sector and cross-sector communication paths, with enough resources to accommodate the required end-to-end services. By using MPLS labels, controllers can implement flows in their local switches to match packets with specific MPLS label IDs, and switch the

Fig. 4 Communication path

packets between ports efficiently along the provisioned path.

In a centralized managed network, MPLS paths connect two end-points, by simply switching the packets with a specific MPLS label ID from one end to the other. But in ArchSDN network, end-points may be connected in different network sectors, which means that packets are required to cross one or more sector boundaries. The more boundaries a communication path crosses, the more sector controllers will be involved in the process, where each one will be responsible for implementing a partial view of the communication path. This requires controllers to cooperate with each other to communicate to negotiate which MPLS label IDs will the packets have and through which switches and ports will the packets be sent, when crossing a sector boundary.

In an ArchSDN enabled network, communication paths have three different types of complexity:

1. In-Sector Path (Fig. 5): A path connecting end-points in the same sector.
2. Cross-Sector Path (Fig. 6): A path connecting end-points in different sectors, crossing a single border.
3. Cross-Multi-Sector Path (Fig. 7): A path connecting end-points in different sectors, crossing multiple borders.

Depending on the location of the end-points in the network, a different type of path will be implemented.

3.3.1 In-sector path (Fig. 5)

The activation of a communication path between two hosts in the same local sector does not require the cooperation of other network controllers. Since the sector controller knows the exact state of the local sector, it is

Fig. 5 In-sector path example

perfectly capable of implementing an optimum communication path between the two hosts, and as so, the in-sector path is implemented using a single and local MPLS path.

3.3.2 Cross-sector path (Fig. 6)

When the sector controller (blue sector) connects a local host with a target host residing in an adjacent sector (red sector), the sector controller will need to contact the adjacent sector controller to request the activation of a communication path to the target host, using a specific border link as the source end-point in the adjacent sector. If the

Fig. 6 Cross-sector path example

Fig. 7 Cross-multi-sector path example

adjacent sector controller is capable to fulfil the request of the local sector controller, the resultant path is a cross-sector path, which starts in the local sector and terminates in the adjacent sector.

The cross-sector path is implemented using two MPLS paths, one path used for the first-half inside the local sector (blue sector) and a second path used for the second-half inside the adjacent sector (red sector). The first-half of the path is the ingress MPLS path in the local sector. It encapsulates the packets from the host of origin with an MPLS header, delivering them to the adjacent sector through the selected border link. The second-half of the path is the egress MPLS path in the adjacent sector. It receives the packets from the local sector with a specific MPLS label ID arriving at the selected border link, changing the packets MPLS label ID value to the ID value of the second-half MPLS path. When the packet arrives to the target host end-point, the MPLS encapsulation is removed from the packets which are then delivered to the local host.

3.3.3 Cross-multi-sector path (Fig. 7)

There are two situations where a cross-multi-sector communication path is required to be activated: (1) when the local sector (blue sector) controller connects a source local host with a target host residing not in an adjacent sector, but in a sector somewhere else in the network (yellow sector), to which the cooperation of the adjacent controllers will be necessary (red sectors); and (2) when the adjacent controller failed to activate a communication path to the target host located in its sector using the selected border link, and as so, the local controller is obliged to find an alternative path through a different sector. In both situations, the local controller will need to delegate the responsibility of activating the communication path to an

adjacent controller, meaning the communication path will need to cross more than one sector.

The cross-multi-sector path is implemented using three or more MPLS paths. One for the first path section in the local sector (blue sector), one for the last path section in the sector of the target host (yellow sector), and one or multiple middle path sections which will redirect the packets between the middle-sectors (red sectors), from one sector to the other.

The first and the last sections of the path are equal to first-half and second-half of the cross-sector path, respectively. The middle parts of the path switch the packets between one middle sector from the inbound border link to the outbound border link. For this to happen, the packets sent from the adjacent sector into the middle sector of the path are matched by looking at the MPLS label ID and changing the ID to the one of the corresponding to the activated middle path, which will lead the packets to the outbound border link and into the next adjacent sector.

Communication paths implemented by the ArchSDN controllers support two different types of directional communication: unidirectional and bidirectional. The majority of services do not require packets flowing from the destination back to the origin, to use the same path which the packets sent from the origin to the destination have used. In fact, services like streaming usually have higher bandwidth requirements in the downstream direction, rather than in the upstream direction. However, services require that reply packets flow through the network using the same path which the request packets have used.

3.4 Communication management

ArchSDN controllers are reactive, executing actions as a reaction to an event from the network. The controllers react to two different events: (1) OpenFlow events; and (2) adjacent controller events.

The OpenFlow events are sent by the local switches. Switches send events either because a local link state has changed, because a flow previously set by the sector controller has expired, or because a packet has been captured due to a match from a flow also previously set by the sector controller.

3.4.1 Service activation

To activate services across the network, ArchSDN controllers set flows in the switches where sector hosts are connected, to capture packets which match a specific signature, allowing the initial detection of a specific service request with specific bandwidth requirements. For example, for the case of generic IPv4 services, using TCP or UDP

protocols with specific source and destination ports, flows with specific matching signatures can be matched.

The capture of an ICMP Request packet sent by a network host to another network host triggers the activation of a bi-directional communication service, while the capture of a generic IPv4 packet will trigger the activation of a unidirectional communication service. To simplify the analysis, in this document all IPv4 packets are considered to be generic traffic, with the exception of DHCP and ICMP network packets.

The ArchSDN controller starts its service activation by analysing the captured packet sent by a local switch, and detecting the type of end-to-end service which the local host aims to use.

If the target host resides within the local sector, the controller does not need the cooperation of the adjacent sectors and proceeds to activate the optimum communication path between the two end-points. But, if the target host resides in some other sector in the network, the controller will contact the central manager to know the sector which the target host resides. After obtaining the target host location, it initiates the contact with the controllers from the adjacent sectors. When there are multiple adjacent controllers, the decision system will decide which controller will be used to try to establish the communication path. The decision and learning processes are addressed in Sect. 4.

After selecting which adjacent controller will be contacted, the local controller sends to it a service implementation request message. The message contains the details regarding the end-to-end service, such as: (a) the service implementation unique ID; (b) the type of service; (c) the source host IP address; (d) the destination IP address; (e) the bandwidth requirements.

Upon receiving the request message, the adjacent controller, which is now the current local controller in the path implementation process has two options. Either completes the path if the target host is present in its sector, or it repeats the process by continuing to explore a path to the target host, using the available adjacent sectors to continue the path implementation. To prevent circular message passing between the network controllers, every time a controller receives a request, it compares the identification of the request with the internal registry of active requests. If the active request identifier is already registered, the controller returns an error to the controller requesting the service implementation. The active request identifier is removed from the controllers' local registrations when the path implementation is complete.

The path implementation either ends when the request reaches the sector where the target host is present, or when the process ends without finding an available path. Not finding a path may happen due to the lack of available

resources for the service, or the absence of an open path to the target host.

3.4.2 Service deactivation

Service deactivation happens when the controller receives an event from a sector switch, announcing the expiration of a flow previously set to switch packets related to a specific previously activated service. When the ArchSDN controller activates a service in the network, the flow set to match the source host of the service traffic is also set with an expiration timer. If the flow does not match any service traffic originating from the source host during a specific interval of time, the flow expires and the switch informs the controller regarding the flow expiration. The controller will then declare that the source host is no longer using the activated service, proceeding to deactivate the service in the network in order to release the network allocated resources.

3.5 Service recovery

Service recovery is an important feature in the ArchSDN controller. It allows the re-establishment of a lost service activation in the event of a link-failure in the network. When a link-state changes in the network, the switch detecting the state change sends a link-state change event to the ArchSDN controller. Upon reception of the event, the ArchSDN controller checks the event to see what link changed state and if the link state is now *down*. If the link state has indeed changed to *down*, the controller creates a manifesto of all the currently active services in the sector that use the link. For the active services which have been requested by adjacent sectors, it sends a message via P2P communication. The message informs the adjacent sectors that the services requested by them are no longer available, and that the sector controller which initially started the service activation should retry again to recover the service loss.

4 Cross-sector path learning and decision mechanism

When a local controller establishes a communication path to a target host located in a different sector, the local controller will resort to the cooperation of its adjacent controllers and learn from previous results. Since a local controller only has information about its network, it needs to find the best path with incomplete network information. The purpose of the decision mechanism is to find the optimal path when there is more than one border link leaving the

sector, and several sectors are crossed between the source and destination.

The autonomous decision management concepts were first demonstrated using a virtualized environment [70], and also in a static wireless mesh network environment with DistArch [65]. Now, the same concept is pushed into the SDN environment.

4.1 Reinforcement learning and SARSA

The base approach for the decision mechanism is the reinforcement learning process which defines, in a system, which actions to take in order to maximize a numerical reward, collected from the previous result of equally taken actions.

Each ArchSDN controller contains an active decision maker, understood as the *Agent*, which implements the reinforcement learning process. Every time it is necessary to decide which border link should be used to establish the communication path, the agent is consulted. The agent is capable to visualize its own state in the network and to take an action with the aim to modify its state (transit to another state). Whenever an action is taken by the controller, the environment responds with a reward in the form of a positive feedback, or in the form of a negative penalty. The agent sees the environment time as a discrete $t \in \mathbb{N} \cap 0$ and $s_t \in S$ represents the state of the agent at time t , where S is the set of all possible states. $a_t \in \mathcal{A}(s_t)$ represents the action selected by the agent at time t , where $\mathcal{A}(s_t)$ is the set of available actions in state s_t [22].

When the agent in state s_t decides to use the available action a_t , the environment clock ticks, the reward $r_{t+1} \in \mathcal{R}$ is received, and the agent transitions to the next state s_{t+1} . This problem is modelled using a Markov Decision Process (MDP) where the state and reward in the next time step depend only on the current state and action. As so, the reward and next state are sampled from their respective probability distributions, $p(r_{t+1}|s_t, a_t)$ and $P(s_{t+1}|s_t, a_t)$, which is a Markov system. Considering that $Q(s_t, a_t)$ denotes how good it is to perform action a_t when in state s_t , $Q(s_t, a_t)$ denotes the expected cumulative reward of action a_t taken in state s_t [22].

The on-policy version of Q-Learning is the SARSA algorithm [22], which chooses the next action a' based on the Q-Value. In previous work conducted for virtual networks and wireless static mesh networks [65, 70], this algorithm was chosen due to its on-policy properties, since the exploration process will use it to choose the next hop when exploring a path. Based on the previous work, the SARSA algorithm now used in a SDN based network environment.

The Q-Values are updated by the SARSA mathematical Eq. 1 each time an action is used.

$$Q(a_t, s_t) = Q(a_t, s_t) + \alpha(r_{t+1} + \gamma Q(a_{t+1}, s_{t+1}) - Q(a_t, s_t)) \quad (1)$$

The Learning Rate ($\alpha \in]0, 1[$) defines the flexibility of the agent when learning from the result of a recent chosen action. The more near-to-zero the learning rate is, the more liberal the agent is causing recent actions to have more impact in future decisions. The more near-to-one the learning rate is, the more conservative the agent is causing recent actions to have less impact in future decisions.

The Discount Factor ($\gamma \in]0, 1[$) determines how much will the next-state reward count. Near-to-zero discount factor values make the agent more greedy to short-term rewards, looking to obtain more reward from the local chosen action. Near-to-one the discount factor values make the agent more far sighted, looking to obtain more reward from the following next-state.

The Reward (r) is the feedback obtained from applying the chosen action a in the state s . If the s_{t+1} is terminal, then the $Q(a_{t+1}, s_{t+1})$ is usually zero.

In the context of the ArchSDN implementation, the reward is not deterministic. In Sect. 4.4, it is described how the reward collected by the ArchSDN controller will depend on the bandwidth available, and why the competition for network resources prevents a network controller to precisely determine the reward for a specific action in a specific state.

4.2 Ranking border links

Instead of randomly selecting a border link, the ArchSDN controller prioritises the selection using the decision mechanism, which implements the SARSA algorithm. The decision mechanism works by continuously calculating and reinforcing the experience value associated with the use of a specific border link, with each service request. This experience value represents the historical quality of the communication paths implemented by using a specific border link, when implementing a communication path to a target host. The experience value allows the possibility to rank the border links by their experience value.

If a specific border link has a history of continuously providing high quality communication paths to a specific host, the rank will increase. But, if the border link has a history of continuously providing low quality communication paths or does not provide communication paths at all to a specific host, the rank will decrease. The border link with the highest rank will be selected, until its rank is decreased or until the rank of the remaining border links increases.

The rank can increase if the exploration process decides to randomly select a different border link, other than the one with the highest rank. The exploration process is necessary to prevent the border link with the highest rank to

be the one selected indefinitely. This way, from time to time, other border links can be tried out to check for the existence of better paths.

4.3 Distributed decisions

When two network hosts are located in different sectors, network controllers are required to cooperate to activate the communication service between the hosts. This cooperation distributes the service activation through the controllers involved in the process.

The process distribution occurs when the communication service activation process is delegated from one controller to another along the path being established, as seen in Fig. 8. The delegation process will continue until the process reaches the controller which controls the sector where the target host is located.

Each time the process is delegated, the receiving controller uses the decision mechanism to choose a border link and delegate the process to the next controller. When the process reaches the sector where the target host is present, the sector controller concludes the process returning the results to the previous controller in line, backtracking the information down to the controller which started the process in the network. This continued chain of delegations distributes the decision mechanism processing load through the network.

4.4 Defining states and rewards

The decision mechanism uses the adapted recursive equation (Eq. 2) to reinforce the adjacent controllers experience with a reward. $Q(CE_t)$ is the current experience value calculated by the local controller which is associated with the use of a specific border link of the network, to implement a communication path to a specific target host. By default, the experience value for every border link is 0. $Q(CE_{t+1})$ is the experience value returned by the adjacent controller

when implementing the forward section of the communication path, using the selected border link by the local controller.

$$Q(CE_t) = Q(CE_t) + \alpha(\text{reward} + \gamma Q(CE_{t+1}) - Q(CE_t)) \quad (2)$$

The reward is calculated using the rewarding equation (Eq. 3) with the properties of the implemented communication path.

$$\text{reward} = \frac{\frac{100.0}{|P|} \times \sum_{I \in P} R_I}{E_{pl} \times K_{spl}} \quad (3)$$

The rewarding equation (Eq. 3) is composed of:

1. $|P|$, which is the length of the activated P path section in the local sector.
2. The *Explored Path Length* E_{pl} , which is the length of the complete communication path from the source host to the target host.
3. The *Known Shortest Path Distance* K_{spl} , which is the minimum distance known by the controller determining the reward, from either the source host or the inbound border link to the target host.
4. The *Remaining Bandwidth* R_I value, which represents the percentage of available bandwidth (multiplied by 100) at the network link I , used to activate the section in the local sector of the complete communication path.

The output of the reward equation provides a ratio between the amount of available bandwidth at the local communication path within the sector, and the length of the explored communication path (E_{pl}) multiplied with the current known shortest path length (K_{spl}). The aim of the rewarding Eq. 3 is to provide higher rewards to communication paths capable of providing shorter paths using less congested links.

4.5 Exploring and learning

The decision mechanism is responsible for the selection of a border link, which is to be used as the outbound link from the local sector (Fig. 9). To determine which border link has the highest rank, the decision mechanism needs to have a historical background regarding the use of the available border links to activate communication paths to specific network hosts. The historical dataset is created by the continued use of border links in the activation of communication paths, allowing the decision system to be trained by constantly reinforcing

Fig. 8 Service request delegating process

Fig. 9 Selecting the border link

the experience value associated to a border link with the specific target host $Q(CE_t)$.

When a controller discovers its own sector, the border links are new and without historical experience, and so, without a defined rank. By default, the decision mechanism prioritizes the selection of unranked border links to activate the communication path. If there are multiple unranked border links, the decision mechanism randomly selects one border link from the unranked set, giving all the available unranked border links the same selection probability (uniform distribution). By prioritizing unranked border links, the decision mechanism guarantees that every available border link will be used at least once. During the delegation process, the highest ranked border link will only be discarded if there are any existent unranked border links.

After building an experience dataset, the decision mechanism can either use the border link with the highest rank, or explore the remaining border links with the aim to find alternative paths capable of providing a better experience. When all of the border links are ranked, the decision mechanism gives the process a 75% probability of using the border links with best ranking ($Q(Ac_t)$), and a 25% probability for exploring other border links to find better alternative paths. Since the decision mechanism is distributed, the decision of using the best border link or exploring the rest needs to be defined at the start of the communication activation process, otherwise, the controllers to whom the activation process is delegated, might decide to change the decision. Therefore, the ArchSDN controller that is responsible for starting the service activation in the network is the one deciding if the communication path to be activated is going to use the highest ranking border links along the cross-sector

communication path, or if it is going to be an exploratory one.

The learning process begins when the delegation process reaches the controller managing the sector where the target host is located, and proceeds during the rollback of the resulting information in the delegation chain (Fig. 8). After activating the final section of the path, the controller returns to the previous controller in the delegation chain the details of the complete path, along with the value for the forward experience ($Q(Ac_{t+1})$). Since the last controller in the delegation process is the one finalizing the activation process, it does not select any border link and, as so, it does not proceed to update ranks. The remaining action is to return the forward experience value of 1 to the previous controller in the delegation chain, indicating the conclusion of the communication path.

When a controller involved in the delegation chain receives an answer from the forward controller, it uses the received forward experience value ($Q(Ac_{t+1})$) to feed the learning equation (Eq. 2) along with the old experience value for the selected border link ($Q(Ac_t)$), and the reward value. The reward value is calculated by feeding the reward equation (Eq. 3) with the received properties of the complete communication path, plus the properties of the path section implemented in the local sector. The result from the application of the learning equation (Eq. 2) is the reinforcement of the experience value ($Q(Ac_t)$) for the selected border link. The rollback process continues with the new experience value ($Q(Ac_t)$) being returned to the previous controller in the delegation chain, along with the properties of the complete communication path. The rollback process ends when it reaches the first controller in the delegation chain, terminating the activation of the communication service. After this, the packets will flow from the source host to the target host, in the case of uni-directional communication paths, or between source and target hosts, in the case of bi-directional communication paths.

5 Experimental evaluation

Having access to the complete network state and information, a centralized SDN controller has the possibility to know at a moment in time, the optimum routes to activate communication paths between network hosts. This allows the SDN central controller to always provide the best route, at the cost of centralizing the management of every switch in a single controller and increasing the possibility of overloading it. This experimental evaluation aims to demonstrate that it is possible to obtain near-to-optimum routing solutions based on incomplete information, by distributing the network

management responsibilities through multiple SDN controllers enhanced with autonomous management properties. The ArchSDN controller decision system aims to improve the quality of the activated communication paths after multiple activations in sequence, by decreasing the length of the communication path. While the initial activations may not provide paths with the optimum length, the shortest path, the decision system aims to provide near-to-optimum solutions after several activations.

The experimental evaluation will consist on performing repeatedly and in sequence, the actions of: (1) activating bidirectional services between two network hosts; and (2) the destruction of the services after timing out. The bidirectional service request will trigger the activation of a bi-directional communication tunnel between the two network hosts, while reserving 100 Kbps of network bandwidth in both directions for the service. After three seconds without receiving any data from the source host to the target host, the network controller managing the sector where the source host is located removes the service activation. Then, it waits for the source host to send a new service request to the target host, to activate again the bidirectional service.

During the evaluation, two different metrics are registered: (1) the length of the activated bi-directional path between two network hosts, and how close it is to the optimal path after learning and exploring the network; and (2) the delay of the service activation in an SDN environment, which should remain near the 50 ms response time. A 50 ms response time is considered to be the baseline metric in telecommunications for acceptable round-trip delay in voice communications.

5.1 Experimental scenarios

Two different network scenarios are created in a real environment with different structural complexities, following the architecture concept observed in Fig. 1. Each scenario is sub-divided into multiple sectors, and an instance of the ArchSDN controller is created to manage each sector. The OpenFlow switches of each sector are configured to connect to the respective sector controller. An instance of the ArchSDN Central Manager is also created in each of the scenarios. For each scenario, the ArchSDN controllers and the ArchSDN Central Manager use a dedicated control network to communicate with each other. The control port of the network switches is also connected to this dedicated control network. The communication data between network hosts is handled by the network switches.

5.1.1 Realistic network topology

The first scenario (Fig. 10) is generated using the NPart tool [71], which uses the Berlin algorithm to generate graphs representing communication networks with a realistic structure in order to simulate a realistic communication network. This network is sub-divided into 6 sectors, each controlled by a different ArchSDN controller instance, containing a total of 31 OpenFlow switches.

5.1.2 Matrix-shaped network topology

The second scenario is a matrix-shaped network (Fig. 11), which aims to recreate a worst case scenario, where the scenario provides a large amount of possible path combinations between two different hosts. This scenario provides the necessary testing ground to observe how the decision system handles in an environment with a large number of possible combinations with different qualities. This network is sub-divided into 9 sectors, each controlled by its own instance of the ArchSDN controller, containing a total of 81 OpenFlow Switches.

Fig. 10 NPart (Berlin Algorithm) topology—6 sectors 37 switches

Fig. 11 Matrix shaped topology—9 sectors 81 switches

5.2 Evaluation description

In each scenario, 10 rounds of bidirectional service activation are conducted where, per round, 50 sequential activations are performed with a 5 second interval per activation. Each service activation remains active during 3 seconds, the enough time for the source host to send a service request, and for the target host to reply. After the 3 seconds time interval, the service is removed from the network.

A service activation consists in finding an available path in the network between two hosts and reserving bandwidth in the links used by the path. Sector controllers do not actively monitor the bandwidth of each network link. Instead, each sector controller maintains a book of links usage by active paths within their sector. This way, each sector controller decides which links should be considered for new service activation, depending if they have enough bandwidth to accommodate the service. To maintain a focus on the decision model, this paper assumes that the hosts are well-behaved and will not abuse from the bandwidth allocated for the activated service.

The experimental evaluation will also assess how the decision system performs when the two network hosts are at different hop distances from each other. To determine

how the decision system performs for both scenarios, 3 different distances will be evaluated ranging from 10 hops to 14 hops of minimum distance between the two network hosts.

A final experimental evaluation will be conducted to assess the capability of the decision system to adapt to the loss of a network link, used by the best solution to a communication path. The testing scenario is the same used for the Realistic Network Topology, where the hosts are separated at a minimum distance of 14 hops. After 50 sequential service activations, a link used by the best bi-directional communication path is disabled, and again, new 50 sequential service activations are executed. The length of the bi-directional communication path for all the service activations is recorded for further analysis.

The delay of the service activation and the bi-directional communication path length for the underlying service are collected as experimental data.

5.3 Experimental results

Section 5.3.1 compares the analysis of the results of the realistic and matrix-shaped scenario. Section 5.3.2 depicts the analysis of the results for the link-loss scenario. Section 5.3.3 presents the analysis of the service activation delays for both scenarios.

In the service activation rounds it is determined the length of the underlying bi-directional communication path, from the source host to the target host. To better understand the improvement in the quality provided by the decision system, the resultant length of the communication path is compared with the value of the minimum distance determined through the shortest-path algorithm.

The plots in Sects. 5.3.1 and 5.3.2 show the difference between the length of the communication path activated by the ArchSDN controller and the shortest path provided by a centralized controller. The plot in Sect. 5.3.3 provides the average delay for a service activation in both realistic and the matrix-shaped scenario.

5.3.1 Realistic and matrix network topology results

The realistic scenario provides the testing ground to understand how the ArchSDN controller reacts when used in a network with a structure that can be found in the real world. To test the ArchSDN controller, 50 sequential service activations were performed for different network host pairs located at different minimum distances. Figures 12 and 13 contain the experimental results for realistic and matrix-based scenarios with 10, 12 and 14 minimum hop distances, respectively. Each sub-plot contains, per activation (axis x), the difference between the lengths of the underlying bi-directional communication path and the

(a)

(b)

(c)

Fig. 12 Realistic scenario results with different hop configuration: **a** 10 Hops; **b** 12 Hops; **c** 14 Hops

optimum shortest path between the network hosts (axis y), where the zero value means a service activation using the optimum shortest-path.

The results of the realistic scenario show how fast is the decision system to adjust its decision, per activation. The decision system quickly adjusts and reinforces its decision, providing and maintaining a reinforced

(a)

(b)

(c)

Fig. 13 Matrix scenario results with different hop configuration: **a** 10 Hops; **b** 12 Hops; **c** 14 Hops

decision upon the best communication path it has found, while continuously trying new paths to detect better paths in the network. It is also possible to observe the quickness of the system to adapt the decision, for all of the pairs of network hosts located at different distances. It is possible to conclude that the decision system is capable of finding near-to-optimum and

even the optimum solution to be used as viable communication paths, only with local information. It is also possible to conclude that the decision system is capable of improving its decisions with the same quality, even when increasing the distance between network hosts.

The matrix-shaped scenario provides the testing ground to understand how the ArchSDN controller reacts in a network with a large amount of different possibilities for the activation of communication paths.

Figure 13 demonstrates the adaptability of the decision system to adjust its decision, per activation. Although it takes more time than in the realistic scenario, the slowdown is expected due to the amount of possible combinations for the activation of communication paths, which are possible with a matrix-shaped topology. Still, the decision system adjusts and reinforces its decision as expected, providing and maintaining a reinforced decision upon the best communication path it has found, while continuously trying new paths.

These results show that, even when facing a high number of possible different solutions, the decision system is capable of finding near-to-optimum solutions to be used as viable communication paths.

5.3.2 Link loss results

Recovering from a link-loss event is a required function in a SDN controller. The ArchSDN controller provides the recovering capability from the loss of a network link, by destroying the service whose underlying communication path requires the failing link to work, and by consequentially recovering the link.

To test this feature, the realistic scenario was used with two network hosts located in the network at a minimum distance of 14 hops. Between those two hosts, 50 service activations were started and destroyed in sequence, similarly to what was performed in Sect. 5.3.1. After finishing the first 50 service activations, the link of the shortest path between the two network hosts is disabled, and another round of 50 service sequential activations is started. Figure 14 shows the length of the underlying communication path between the two network hosts provided by the decision mechanism. It is possible to observe that the decision system adapts to the new network environment and constraints after the link being disabled (dashed line in the plot). The proposed distributed approach was able to recover and to continuously adapt its decision quality, by exploring new alternatives through different border links with each new activation, and continuously finding communication paths with better quality.

Fig. 14 Length of the underlying communication path for each service activation. Before (blue) and after (red) the link-loss (dashed vertical line)

5.3.3 Service activation delay

The service activation delay provides a view on how fast is the proposed system capable of providing a solution for the service, and how fast the controllers populate the OpenFlow switches in the network with the required flows. For a SDN network to provide fast and agile response to network events, having a low delay for service activation is imperative.

Figure 15 depicts two different plots with the average service activation delays, for both the realistic and matrix scenarios, using different minimum distances for the network hosts. The completion of a service activation requires the full configuration of the bi-directional communication path between two network hosts, which includes the configuration of the OpenFlow switches by their respective sector controller.

The delay of the full configuration of a bi-directional communication path in the network remains below the 100 ms mark. Also, comparing the activation delays from the two tested scenarios, it becomes clear that changing the network topology causes little effect in the response delay of the decision system: the delay variation is below 20 ms. These results allow to conclude that the reaction time is maintained at very low values for both scenarios, providing the necessary agility required for SDN Networks.

6 Conclusion and future work

This paper proposed ArchSDN, an autonomous controller model for SDN Networks using OpenFlow Switches. Instead of centralizing the management of an SDN based network, the ArchSDN controller restructures the network by splitting it into sectors and instantiating an ArchSDN

Fig. 15 Service activation delay: **a** realistic scenario; **b** matrix-shaped scenario

controller to manage each sector. Using a decision system based in reinforcement learning concepts, the instantiated controllers in the network cooperate with each other to provide near-to-optimum quality host-to-host communication, without knowing the complete network state.

The network isolation into sectors allows the distribution of the OpenFlow switches management through multiple controllers, providing the ground for an efficient and scalable management architecture, at the cost of a minimal quality loss in the host-to-host communication.

The experimental results demonstrate the viability of the ArchSDN controller proposal, as an alternative to the traditional SDN controller approach. They show how the reinforcement learning based decision system is capable of finding routing solutions for the activation of communication paths between network hosts, and learn from the obtained results to improve future service activation results. The results also demonstrate the capability of the decision system to adapt to unforeseen circumstances, such as the loss of a network link. It is demonstrated that the decision system is capable of discovering alternative

paths, while continuously exploring and improving with better alternatives to react to the loss of the network link. Finally, the results have shown that the ArchSDN controller is capable of activating communication services between two hosts in less than 100 ms. Moreover, even with significant changes to the network topology and with the increase of the hop distance between two network hosts, the service activation delay remained below the 100 ms mark.

Currently, the ArchSDN controller is a reactive controller: it answers to the hosts' requests and activates services based on the hosts data traffic requests. Improvements could be performed in the future using Artificial Intelligence concepts, by continuously analysing the data exchanged between two or more network hosts and to establish the hosts data traffic profile. By detecting the traffic profile, the ArchSDN controller would be able to coordinate with the other network controllers, to activate alternative paths which although would not be the best available communication paths, they would be the communication paths with a lower load with the necessary quality to answer the requested service requirements.

Compliance with ethical standards

Conflict of interest On behalf of all authors, the corresponding author states that there is no conflict of interest.

References

1. Fiebig T (2017) An empirical evaluation of misconfiguration in internet services, Ph.D. dissertation, Technische Universität Berlin, Technische Universität Berlin. <https://doi.org/10.14279/depositonce-6140>
2. Clark DD, Partridge C, Ramming JC, Wroclawski JT (2003) A knowledge plane for the internet. In: Proceedings of the 2003 conference on applications, technologies, architectures, and protocols for computer communications—SIGCOMM'03. ACM Press, New York, p 3. <https://doi.org/10.1145/863955.863957>
3. McKeown N (2009) Software defined networking. INFOCOM Keynote Talk 17(2):30–32
4. McKeown N, Anderson T, Balakrishnan H, Parulkar G, Peterson L, Rexford J, Shenker S, Turner J (2008) OpenFlow: enabling innovation in campus networks. ACM SIGCOMM Comput Commun Rev 38(2):69. <https://doi.org/10.1145/1355734.1355746> [Online]
5. Bosshart P, Daly D, Gibb G, Izzard M, McKeown N, Rexford J, Schlesinger C, Talayco D, Vahdat A, Varghese G et al (2014) P4: programming protocol-independent packet processors. SIGCOMM Comput Commun Rev 44(3):87–95. <https://doi.org/10.1145/2656877.2656890> [Online]
6. Poullos G, Tsagkaris K, Demestichas P, Tall A, Altman Z, Destre C (2014) Autonomics and SDN for self-organizing networks. In: 2014 11th international symposium on wireless communications systems (ISWCS). IEEE, pp 830–835. <https://doi.org/10.1109/ISWCS.2014.6933468>

7. Karakus M, Durrresi A (2017) A survey: control plane scalability issues and approaches in software-defined networking (SDN). *Comput Netw* 112:279–293. <https://doi.org/10.1016/j.comnet.2016.11.017> [Online]
8. Oktian YE, Lee S, Lee H, Lam J (2017) Distributed SDN controller system: a survey on design choice. *Comput Netw* 121:100–111. <https://doi.org/10.1016/j.comnet.2017.04.038> [Online]
9. Stallings W (2013) Software-defined networks and openflow. *Internet Protocol J* 16(1):2–14
10. Trestian R, Katrinis K, Muntean G (2017) Ofload: an openflow-based dynamic load balancing strategy for datacenter networks. *IEEE Trans Netw Serv Manag* 14(4):792–803. <https://doi.org/10.1109/TNSM.2017.2758402> [Online]
11. Alsaeedi M, Mohamad MM, Al-Roubaiey AA (2019) Toward adaptive and scalable openflow-sdn flow control: a survey. *IEEE Access* 7:107 346–107 379. <https://doi.org/10.1109/ACCESS.5.2019.2932422> [Online]
12. Gazis V, Kousaridas A, Polychronopoulos C, Raptis T, Alonistioti N (2009) Self-management capacities in future internet wireless systems. In: 2009 Computation world: future computing, service computation, cognitive, adaptive, content, patterns. IEEE, pp 9–14. <https://doi.org/10.1109/ComputationWorld.2009.36>
13. Prieto AG, Gillblad D, Steinert R, Miron A (2011) Toward decentralized probabilistic management. *IEEE Commun Mag* 49(7):80–86. <https://doi.org/10.1109/MCOM.2011.5936159> [Online]
14. Sekar V, Reiter MK, Willinger W, Zhang H, Kompella RR, Andersen DG (2008) CSAMP: a system for network-wide flow monitoring. *Usenix—The Advanced COmputing Systems Association*. http://usenix.org/events/nsdi08/tech/full_papers/sekar/sekar_html/index.html
15. Koch FL, Westphall CB (2001) Decentralized network management using distributed artificial intelligence. *J Netw Syst Manag* 9(4):375–388. <https://doi.org/10.1023/A:1012976206591> [Online]
16. Nobre JC, Granville LZ (2010) Consistency maintenance of policy states in decentralized autonomic network management. In: 2010 IEEE network operations and management symposium—NOMS 2010. IEEE, pp 519–526. <https://doi.org/10.1109/NOMS.2010.5488469>
17. Nobre JC, Granville LZ, Clemm A, Prieto AG (2013) Coordination in P2P management overlays to improve decentralized detection of SLA violations. In: 2013 IEEE international conference on communications (ICC). IEEE, pp 3456–3460. <https://doi.org/10.1109/ICC.2013.6655084>
18. Jennings B, van der Meer S, Balasubramaniam S, Botvich D, Foghlú MÓ, Donnelly W, Strassner J (2007) Towards autonomic management of communications networks. *IEEE Commun Mag* 45(10):112–121. <https://doi.org/10.1109/MCOM.2007.4342833> [Online]
19. Dobson S, Zambonelli F, Denazis S, Fernández A, Gaïti D, Gelenbe E, Massacci F, Nixon P, Saffre F, Schmidt N (2006) A survey of autonomic communications. *ACM Trans Auton Adapt Syst* 1(2):223–259. <https://doi.org/10.1145/1186778.1186782> [Online]
20. Huebscher MC, McCann Ja (2008) A survey of autonomic computing degrees, models, and applications. *ACM Comput Surv* 40(3):1–28. <https://doi.org/10.1145/1380584.1380585> [Online]
21. Thomas R, Friend D, Dasilva L, Mackenzie A (2006) Cognitive networks: adaptation and learning to achieve end-to-end performance objectives. *IEEE Commun Mag* 44(12):51–57. <https://doi.org/10.1109/MCOM.2006.273099> [Online]
22. Sutton RS, Barto AG (1998) Reinforcement learning: an introduction. *IEEE Trans Neural Netw* 9(5):1054. <https://doi.org/10.1109/TNN.1998.712192> [Online]
23. Bokareva T, Bulusu N, Jha S (2005) SASHA: toward a self-healing hybrid sensor network architecture. In: Second IEEE workshop on embedded networked sensors, EmNetS-II, vol 2005, pp 71–78. <https://doi.org/10.1109/EMNETS.2005.1469101>
24. Dai Y, Xiang Y, Li Y, Xing L, Zhang G (2011) Consequence oriented self-healing and autonomous diagnosis for highly reliable systems and software. *IEEE Trans Reliab* 60(2):369–380. <https://doi.org/10.1109/TR.2011.2136490> [Online]
25. Sherwood R, Gibb G, K.-K. Yap, Appenzeller G, Casado M, McKeown N, Parulkar G (2009) Flowvisor: a network virtualization layer. *OpenFlow Switch Consortium, Tech Rep* 1:132. <https://pdfs.semanticscholar.org/64f3/a81fff495ac336dcccdd63136d451852eb1c9.pdf>
26. Salvadori E, Doriguzzi Corin R, Gerola M, Broglio A, De Pellegrini F (2011) Demonstrating generalized virtual topologies in an openflow network. In: ACM SIGCOMM computer communication review, vol 41(4). ACM, pp 458–459. <https://doi.org/10.1145/2018436.2018520>
27. Doriguzzi Corin R, Gerola M, Riggio R, De Pellegrini F, Salvadori E (2012) Vertigo: network virtualization and beyond. In: 2012 European workshop on software defined networking, pp 24–29. <https://doi.org/10.1109/EWSN.2012.19>
28. Al-Shabibi A, De Leenheer M, Gerola M, Koshibe A, Parulkar G, Salvadori E, Snow B (2014) Openvirtex: make your virtual SDNS programmable. In: Proceedings of the 3rd workshop on hot topics in software defined networking, ser. HotSDN'14. Association for Computing Machinery, New York, pp 25–30. <https://doi.org/10.1145/2620728.2620741>
29. Al-Shabibi A, De Leenheer M, Gerola M, Koshibe A, Snow W, Parulkar G (2014) Openvirtex: a network hypervisor. In: Open networking summit 2014 (ONS 2014). <https://www.usenix.org/system/files/conference/ons2014/ons2014-paper-al-shabi bi.pdf>
30. Lee B, Park S H, Shin J, Yang S (2014) IRIS: the openflow-based recursive SDN controller. In: 16th international conference on advanced communication technology. Global IT Research Institute (GIRI), pp 1227–1231. <https://doi.org/10.1109/ICACT.2014.6779154>
31. Santos MAS, Nunes BAA, Obraczka K, Turletti T, de Oliveira BT, Margi CB (2014) Decentralizing SDN's control plane. In: 39th annual IEEE conference on local computer networks. IEEE, pp 402–405. <https://doi.org/10.1109/LCN.2014.6925802>
32. Yeganeh SH, Ganjali Y, Hassas Yeganeh S, Ganjali Y (2012) Kandoo. In: Proceedings of the first workshop on Hot topics in software defined networks—HotSDN'12. ACM Press, New York, p 19. <https://doi.org/10.1145/2342441.2342446>
33. Marconett D, Yoo SJB (2015) FlowBroker: a software-defined network controller architecture for multi-domain brokering and reputation. *J Netw Syst Manag* 23(2):328–359. <https://doi.org/10.1007/s10922-014-9325-5> [Online]
34. Bianco A, Giaccone P, Mashayekhi R, Ullio M, Vercellone V (2017) Scalability of ONOS reactive forwarding applications in ISP networks. *Comput Commun* 102:130–138. <https://doi.org/10.1016/j.comcom.2016.09.007> [Online]
35. Lin P, Bi J, Wang Y (2013) East-west bridge for SDN network peering. In: Frontiers in internet technologies, pp 170–181. https://doi.org/10.1007/978-3-642-53959-6_16
36. Dixit A, Hao F, Mukherjee S, Lakshman T, Kompella R (2013) Towards an elastic distributed SDN controller. In: Proceedings of the 2nd ACM SIGCOMM workshop on hot topics in software defined networking—HotSDN'13, vol 43(4), p 7. <https://doi.org/10.1145/2491185.2491193>
37. Phemius K, Bouet M, Leguay J (2014) DISCO: distributed multi-domain SDN controllers. *IEEE/IFIP NOMS 2014—IEEE/IFIP network operations and management symposium: management in a software defined world*. <https://doi.org/10.1109/NOMS.2014.6838330>

38. Tootoonchian A, Ganjali Y (2010) Hyperflow: a distributed control plane for openflow. In: Proceedings of the 2010 internet network management conference on Research on enterprise networking, pp 3. <http://dl.acm.org/citation.cfm?id=1863133.1863136>
39. Koponen T, Casado M, Gude N, Stribling J, Poutievski L, Zhu M, Ramanathan R, Iwata Y, Inoue H, Hama T, Others, Shenker S (2010) Onix: a distributed control platform for large-scale production networks. OSDI, Oct, pp 1–6. <http://dl.acm.org/citation.cfm?id=1924943.1924968>
40. Yazici V, Sunay MO, Ercan AO (2014) Controlling a software-defined network via distributed controllers. arXiv preprint [arXiv:1401.7651](https://arxiv.org/abs/1401.7651), 90(11580):6.
41. Katta N, Zhang H, Freedman M, Rexford J (2015) Ravana. In: Proceedings of the 1st ACM SIGCOMM symposium on software defined networking research—SOSR'15. ACM Press, New York, pp 1–12. <https://doi.org/10.1145/2774993.2774996>
42. Yu M, Rexford J, Freedman MJ, Wang J (2010) Scalable flow-based networking with DIFANE. ACM SIGCOMM Comput Commun Rev 40(4):351. <https://doi.org/10.1145/1851275.1851224> [Online]
43. Curtis AR, Mogul JC, Tourrilhes J, Yalagandula P, Sharma P, Banerjee S (2011) Devoflow: scaling flow management for high-performance networks. In: Proceedings of the ACM SIGCOMM 2011 conference, ser. SIGCOMM'11. Association for Computing Machinery, New York, pp 254–265. <https://doi.org/10.1145/2018436.2018466>
44. Chaparadza R, Ben Meriem T, Radier B, Szott S, Wodczak M, Prakash A, Jianguo Ding, Soulhi S, Mihailovic A (2013) SDN enablers in the ETSI AFI GANA reference model for autonomic management & control (emerging standard), and virtualization impact. In: 2013 IEEE globecom workshops (GC Wkshps). IEEE, pp 818–823. <https://doi.org/10.1109/GLOCOMW.2013.6825090>
45. Civanlar S, Lokman E, Kaytaz B, Murat Tekalp A (2015) Distributed management of service-enabled flow-paths across multiple SDN domains. In: 2015 European conference on networks and communications (EuCNC). IEEE, pp 360–364. <https://doi.org/10.1109/EuCNC.2015.7194099>
46. Caria M, Jukan A, Hoffmann M (2016) SDN partitioning: a centralized control plane for distributed routing protocols. IEEE Trans Netw Serv Manag 13(3):381–393. <https://doi.org/10.1109/TNSM.2016.2585759> [Online]
47. Kreutz D, Ramos FMV, Esteves Verissimo P, Esteve Rothenberg C, Azodolmolky S, Uhlig S (2015) Software-defined networking: a comprehensive survey. Proc IEEE 103(1):14–76. <https://doi.org/10.1109/JPROC.2014.2371999> [Online]
48. Li H, Que X, Hu Y, Xiangyang G, Wendong W, Hongyun L, Que X, Hu Y, Xiangyang G, Wendong W (2013) An autonomic management architecture for SDN-based multi-service network. In: 2013 IEEE globecom workshops (GC Wkshps), no 2011. IEEE, pp 830–835. <https://doi.org/10.1109/GLOCOMW.2013.6825092>
49. Galis A, Clayman S, Mamatas L (2013) Towards autonomic management of software enabled networks. In: 2013 8TH international symposium on advanced topics in electrical engineering (ATEE). IEEE, pp 1–3. <https://doi.org/10.1109/ATEE.2013.6563356>
50. Qi Q, Wang W, Gong X, Que X (2014) A SDN-based network virtualization architecture with autonomic management. In: 2014 IEEE globecom workshops (GC Wkshps). IEEE, pp 178–182. <https://doi.org/10.1109/GLOCOMW.2014.7063427>
51. Wang W, Qi Q, Gong X, Hu Y, Que X (2014) Autonomic QoS management mechanism in software defined network. China Commun 11(7):13–23. <https://doi.org/10.1109/CC.2014.6895381> [Online]
52. Berde P, Snow W, Parulkar G, Gerola M, Hart J, Higuchi Y, Kobayashi M, Koide T, Lantz B, O'Connor B, Radoslavov P (2014) ONOS—towards an open, distributed SDN OS. In: Proceedings of the 3rd workshop on hot topics in software defined networking—HotSDN'14, 2014, pp 1–6. <https://doi.org/10.1145/2620728.2620744>
53. Krishnamurthy A, Chandrabose SP, Gember-Jacobson A (2014) Pratyastha: an efficient elastic distributed SDN control plane. In: Proceedings of the 3rd workshop on hot topics in software defined networking—HotSDN'14. ACM Press, New York, pp 133–138. <https://doi.org/10.1145/2620728.2620748>
54. Botelho F, Bessani A, Ramos FMV, Ferreira P (2014) On the design of practical fault-tolerant SDN controllers. In: Proceedings—2014 3rd European workshop on software-defined networks, EWSDN 2014, pp 73–78. <https://doi.org/10.1109/EWSDN.2014.25>
55. Ahmad I, Namal S, Ylianttila M, Gurtov A (2015) Towards software defined cognitive networking. In: 2015 7th international conference on new technologies, mobility and security (NTMS). IEEE, pp 1–5. <https://doi.org/10.1109/NTMS.2015.7266528>
56. Foteinos V, Poullos G, Tsagkaris K, Logothetis M, Foteinos V, Poullos G, Michaloliakos M, Demestichas P (2015) Customizable autonomic network management: integrating autonomic network management and software-defined networking. IEEE Veh Technol Mag 10(1):61–68. <https://doi.org/10.1109/MVT.2014.2380633> [Online]
57. Neves P, Calé R, Costa MR, Parada C, Parreira B, Alcaraz-Calero J, Wang Q, Nightingale J, Chirivella-Perez E, Jiang W, Schotten HD, Koutsopoulos K, Gavras A, Barros MJ (2016) The SELFNET approach for autonomic management in an NFV/SDN networking paradigm. Int J Distrib Sensor Netw 2016:1–17. <https://doi.org/10.1155/2016/2897479> [Online]
58. Yang H, Kim Y (2016) SDN-based distributed mobility management. In: 2016 international conference on information networking (ICOIN), vol 2016-March. IEEE, pp 337–342. <https://doi.org/10.1109/ICOIN.2016.7427127>
59. Nguyen T-T, Bonnet C, Harri J (2016) SDN-based distributed mobility management for 5G networks. In: 2016 IEEE wireless communications and networking conference, no. April. IEEE, pp 1–7. <https://doi.org/10.1109/WCNC.2016.7565106>
60. Elgendi I, Munasinghe KS, Jamalipour A (2016) A three-tier SDN based distributed mobility management architecture for DenseNets. In: 2016 IEEE international conference on communications (ICC). IEEE, pp 1–6. <https://doi.org/10.1109/ICC.2016.7511020>
61. Idri M (2017) Mobility management based SDN-IPv6 routing header. In: 2017 4th international conference on software defined systems (SDS). IEEE, pp 150–155. <https://doi.org/10.1109/SDS.2017.7939156>
62. Adami D, Donatini L, Foddiss G, Giordano S, Roma S, Topazzi S (2014) Design and development of management functions for distributed monitoring based on SDN-based network. In: 2014 Euro med telco conference (EMTC). IEEE, pp 1–5. <https://doi.org/10.1109/EMTC.2014.6996655>
63. Fu Y, Bi J, Gao K, Chen Z, Wu J, Hao B (2014) Orion: a hybrid hierarchical control plane of software-defined networking for large-scale networks. In: Proceedings—international conference on network protocols, ICNP, pp 569–576. <https://doi.org/10.1109/ICNP.2014.91>
64. Yang H, Ivey J, Riley GF (2018) Scalability comparison of SDN control plane architectures based on simulations. In: 2017 IEEE 36th international performance computing and communications conference, IPCCC 2017, vol 2018-January, pp 1–8. <https://doi.org/10.1109/IPCCC.2017.8280465>
65. Ferreira C, Sargento S, Oliveira A (2017) An architecture for a learning-based autonomic decision system. J Comput Sci. <https://doi.org/10.1016/j.jocs.2017.04.010>

66. Guardalben L, Sargento S, Salvador P, Mirones V (2010) A cooperative hide and seek discovery over in network management. In: 2010 IEEE/IFIP network operations and management symposium workshops, ser. 2010 IEEE/IFIP network operations and management symposium workshops, NOMS 2010. IEEE, pp 217–224. <https://doi.org/10.1109/NOMSW.2010.5486570>
67. Guardalben L, Gomes T, Pinho A, Salvador P, Sargento S (2012) Nodes discovery in the in-network management communication framework. In: Mobile networks and management. Springer, Berlin, pp 145–157. https://doi.org/10.1007/978-3-642-30422-4_11
68. Guardalben L, Condeixa T, Gomes T, Salvador P, Sargento S (2013) On the analysis of dissemination management information through an eyesight perspective. In: 2013 IEEE globe-com workshops (GC Wkshps). IEEE, pp 1001–1006. <https://doi.org/10.1109/GLOCOMW.2013.6825122>
69. Rivest RL (1992) The md5 message-digest algorithm. Internet Requests for Comments, RFC Editor, RFC 1321. <https://doi.org/10.17487/rfc1321>
70. Ferreira C, Sargento S, Oliveira A (2016) Learning-based automatic decision system for bandwidth-aware routing. In: 2016 IEEE symposium on computers and communication (ISCC), pp 56–63. <https://doi.org/10.1109/ISCC.2016.7543714>
71. Milic B, Malek M (2009) NPART—node placement algorithm for realistic topologies in wireless multihop network simulation. In: 2nd international ICST conference on simulation tools and techniques, 2009. <https://doi.org/10.4108/ICST.SIMUTOOLS2009.5669>

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.