Software-Defined Inter-networking: Enabling Coordinated QoS Control Across the Internet

George Petropoulos*, Fragkiskos Sardis**, Spiros Spirou* and Toktam Mahmoodi**
* Intracom Telecom, Peania, Greece, {geopet, spis}@intracom-telecom.com
** Department of Informatics, Kings College London, UK, {fragkiskos.sardis, toktam.mahmoodi}@kcl.ac.uk

***Abstract*—Software-Defined Networking can be used to easily provision paths for business applications. However, such provisioning is static and is furthermore lost when application flows cross over to another operator domain. This paper proposes an online mechanism (SDNI) that considers application requirements and uses SDN to negotiate traffic policies between neighbouring operator domains. Hence, SDNI enables the hop-by-hop creation of end-to-end paths that are appropriately provisioned for application flows. A prototype and a small-scale testbed have been implemented to assess SDNI's feasibility. Early results show that SDNI overhead does not affect application performance in various use cases, while the benefits of SDNI are evident.**

I. Introduction

The administrative boundary of a Software-Defined Networking (SDN) deployment is limited by the boundary of the Network Service Provider (NSP) domain. At the interconnection between domains, any local policy set through SDN becomes invalid. This can have a detrimental effect on services and applications that transcend domains because traffic policies are typically different between domains. To address this problem, we propose *SDN Inter-networking (SDNI)*: an interface between SDN Controllers belonging to neighbouring domains for coordinating traffic policies. Given QoS requirements from an application, the neighbouring SDN Controllers use SDNI to negotiate a bilateral policy and its configuration, especially on border SDN Switches. This negotiation takes into account any local policy, as well as network capabilities and does not expose sensitive business information.

SDNI stitches across NSP domains an inter-network that can grow or shrink dynamically and enables inter-domain QoS provisioning. As a use case, an NSP that doubles as a Content Service Provider (CSP) (e.g., an NSP offering intra-domain IPTV) can use SDNI to expand the service footprint to a peering NSP, without deploying extra equipment and without offline re-negotiation of Service-Level Agreements (SLAs). That footprint grows dynamically as more NSPs decide to join. The ensuing, loosely-federated content delivery network can be made available to other CSPs, as an alternative to centrally-managed content delivery networks like Akamai. Thus, the value of SDNI really lies in its empowerment of NSPs to offer more than Internet connectivity.

This work has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 671648 (VirtuWind).

The rest of the paper is outlined as follows: Section II provides background information on SDN and QoS provisioning. Section III discusses the design of SDNI, the functions involved in achieving inter-domain policy negotiations and the related communication interfaces. Section IV presents the experimental implementation and preliminary performance measurements along with a discussion and interpretation of the results. Finally, Section V concludes the paper with a summary of contributions and future work.

II. Related work

In this section we look at some approaches in the areas of application control, inter-domain QoS coordination and SDN interconnection which are relevant to SDNI.

Participatory Networking (PANE) [1] is an API that allows applications to have restricted control over an SDN-enabled network. It provides applications with an interface to apply their policies, dynamically request resources and query the state of the network. Accepted application requests are materialized on the network by configuring flow rules in the SDN Switches. However, PANE is limited to a single administrative domain. SDNI incorporates the semantics of the PANE API and focuses on the negotiation of application requirements between multiple domains.

Bandwidth Brokers (BBs) have been defined within the scope of DiffServ [2] for the purpose of cross-domain resource allocation and traffic differentiation. A BB monitors its domain topology, resources, policies and SLAs and optimizes the use of internal resources in order to satisfy resource allocation requests made by applications or peering networks. BB implementations differ on whether the domain control is centralized, distributed, or hybrid [3]. BBs assume that NSP domains operate with DiffServ, whereas SDNI can work with any traffic differentiation mechanism.

Path Computation Elements (PCEs) [4] perform constraint-based path computation in Multiprotocol Label Switching (MPLS) and Generalized MPLS (GMPLS) networks. A PCE keeps track of its domain topology, resources and policies and computes intra-domain paths. The PCE communication protocol (PCEP) [5] is used for the interaction of PCEs with network devices and adjacent PCEs to compute and manage end-to-end (e2e) paths. The PCEP protocol is currently implemented as an OpenDaylight (ODL) module [6] and allows topology notifications and Label-Switched Path (LSP) setup,

while SDNI aims to be agnostic to each domain's traffic differentiation mechanisms.

SDN-i [7] was proposed at the IRTF SDN Research Group for interfacing SDN domains that belong to the same NSP domain. It sketched a protocol for coordinating the behaviour between SDN Controllers and for exchanging reachability, capability and flow information across multiple SDN domains, according to information given from applications. In a single NSP domain, SDN Controllers trust each other and can exchange sensitive information. Our work focuses on multiple NSP domains where controllers' negotiation with limited information is required.

DISCO [8] proposed a distributed control plane and designed a message-oriented communication bus which enables SDN Controllers of different domains to exchange topology information and provide e2e services. Its architecture and mechanisms are a solid basis for our design, but it focuses on inter-domain routing over SDN.

Our work goes beyond the state of the art as it proposes protocol and mechanisms for the interconnection of SDN Controllers, offering more than inter-domain routing. It defines an online negotiation scheme which allows dynamic interaction with applications, enables NSPs to negotiate traffic flows and provision high-QoS paths locally, eventually achieving inter-domain e2e service provisioning.

III. Architecture Design

Section I sketched one of the use cases we have considered for SDNI. Based on those use cases, we have identified a number of requirements to drive the design of SDNI. The main requirements are:

1) An NSP should be able to configure its internal policies.

2) Business applications should be able to inform the network about flows and their QoS characteristics and also receive network capabilities.

3) Upon reception of an application request, neighbouring NSPs should negotiate the treatment of the application flow based on their internal policies, without exposing strategic and sensitive information.

4) After successful negotiation, NSPs must configure their network, apply their traffic differentiation mechanism and mark ingress and egress flow packets accordingly.

5) Interfaces must be secure to prevent harmful behaviour.

SDN provides means to differentiate and apply different QoS to certain traffic flows. OpenFlow [9] supports egress queues to prioritize types of traffic and guarantee the minimum bit rate of packets leaving a switch port. Hence, queues can also be applied to limit "greedy" application flows. Meters are another OpenFlow feature that allows rate-monitoring of traffic based on its ingress rate. Depending on the meter status and traffic rates, certain policies can be applied such as packet (re)marking or drop. Meters have the advantage that they can be configured dynamically and added to flows by SDN Controllers while queues are set up on switches bootstrapping.

Each NSP can apply their own traffic differentiation mechanisms such as DiffServ, MPLS, and map them to specific

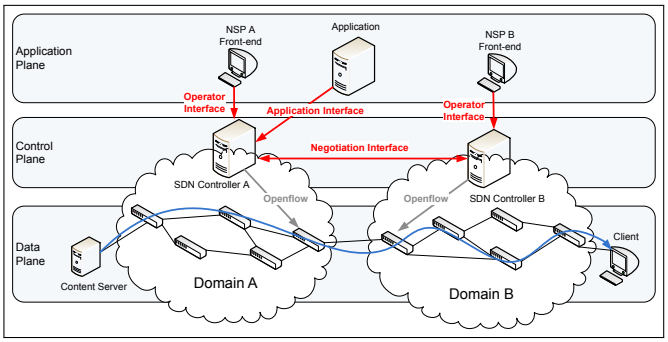

Fig. 1. The SDNI high-level architecture.

queues or meters. NSPs, via their respective SDN Controller, set up long-term flows to the switches to meet local policies and SLAs. New and dynamic application requests could either map to such existing flow rules or require short-term installation of new ones. For fine-grained control of traffic, flows use layer 2 and 3 header fields matching, hence scalability and resilience aspects need to be taken into account with regards to the installation of new flows or aggregation of existing flows in large networks.

The high-level SDNI architecture shown in Figure 1 consists of 3 planes: the *Application plane*, where application and NSP front-ends and servers reside, the *Control plane*, where the SDN Controllers operate and communicate using SDNI and finally the *Data plane* where traffic is forwarded. Our approach exposes 3 interfaces: the *Operator Interface* which is used by the NSP to set up internal policies, the *Application Interface*, allowing applications to convey traffic requirements and characteristics and the *Negotiation Interface*, usually referred in literature as East-West API, allowing peering SDN Controllers to negotiate application requirements and set up e2e paths.

A. Interfaces

SDN Controllers offer an *Operator Interface* for managing the SDNI within the domain. In traditional NSP networks, this could be expressed as the interface towards the NSP's Network Management System (NMS). In this paper, we will define only the interface semantics, which are essential to our design.

Applications can be registered, providing their identifier, type and QoS requirements (bandwidth, jitter, packet loss and packet delay). Applications can also be authorized or blocked per request. Generic application types and their characteristics are also essential for enabling their mapping to internal policies, as well as easing flow aggregation. Network infrastructure management tasks can also be performed, enabling the NSP to set up traffic prioritization mechanisms, such as queues, meters and their maximum rates and to define the protocols, such as MPLS, DiffServ and their mapping to these frameworks. The inter-domain links and adjacent domains are also configured, providing the domain's border switch, queues and meters, the adjacent SDN Controller IP address and billing information.

Business applications will use the provided *Application Interface* to specify traffic flows which require e2e paths with

TABLE I
SDNI INTERFACES, MESSAGES AND SEMANTICS

Interface	Messages	Semantics
Operator	Application	appId, appType, edgeSwitch, protocol, bw, packetLoss, packetDelay, jitter
	Allow Application	appId, true/false
	Application Type	appType, bw, packetLoss, packetDelay, jitter, queue/meter
	Policy	protocol, queues/meters thresholds, strategy
	Border Data	domain, borderSwitch, port, neighbourSdnController, queues/meters, billingInfo
Application	Application Requirements	appId, srcIp, srcPort, dstIp, dstPort, protocol, start, duration, bw, packetLoss, packetDelay, jitter
Negotiation	Negotiation Request	appId, reqId, srcIp, srcPort, dstIp, dstPort, protocol, start, duration, bw, packetLoss, packetDelay, jitter
	Negotiation Response	appId, reqId, ACCEPT/REJECT/NEGOTIATE, diffBw, diffPacketLoss, diffPacketDelay, diffJitter
	Negotiation Notification	appId, reqId, CONFIRM/CANCEL

certain QoS. A unique flow is characterized by {source IP address, destination IP address, source port, destination port, protocol}. Application requests also include the required e2e QoS parameters for a specific flow and its time constraints. Requests trigger the SDNI logic and lead to a unique request ID, which is returned to the business applications to receive the request status and monitoring data. Application requests for specific traffic flows can be sent either proactively or in real-time depending on the application type, the existing utilization of the underlying network or due to an emergency event.

Negotiation Interface is used by the SDN Controllers to exchange application requirements and reserve e2e inter-domain paths. It partially follows patterns proposed within the scope of Generic Autonomic Signaling Protocol (GRASP) [10], specifically its negotiation messages and procedures. Three types of messages are exchanged:

- *Negotiation Request*: This is either the first message of the negotiation protocol or the updated offer after an unsuccessful negotiation round. It is identical to the application request, including the requested flow data, their QoS parameters and time constraints.
- *Negotiation Response*: This message is sent as a response to a Negotiation Request, including the outcome of the request, ACCEPT, REJECT, NEGOTIATE and in the latter case, the bandwidth, packet loss, delay and jitter which can be served in the peer domain, in terms of deltas of the originally requested ones.
- *Negotiation Notification*: This message is used to inform the adjacent domain of the negotiation outcome or other events, in order to reserve or release resources and configure its network accordingly.

Table I presents the key data structures, messages and semantics for each interface. The sections that follow provide technical details on the defined interfaces' supporting mechanisms.

B. Intra-Domain Functionalities

Upon reception of an application request, the intra-domain admission control mechanism takes over. The SDN Controller checks whether the application is registered and authorized to perform such requests or blocked due to harmful or greedy behaviour in the past. The requested QoS parameters are also checked against the already defined profiles for such types of applications. In case that the request violates these profiles, the operation is aborted.

Available configured paths are checked to determine if they can serve the requested traffic flow. In case this is not feasible and the requested e2e path is within the domain, the intra-domain path computation process is executed. The algorithm receives as input the intra-domain graph, latest measurements of QoS metrics, retrieved by Openflow counters, and the requested ones. It then calculates the constrained shortest path, pruning the links that violate the application requirements, and calculating the best available path using the Dijkstra algorithm.

Finally new flow rules are configured to the selected SDN Switches. These flow rules consist of the source and destination IP prefix, protocol and ports and include the mapping to the domain's traffic differentiation mechanism and are configured for the specified duration. The SDN Controller keeps track of configured flow rules and requested resources and is able to remove and release them or renew their duration upon relevant request. In the case the destination address is in a foreign domain, the negotiation process has to be triggered, which will be described further in the following section.

SDN Controllers periodically monitor their underlying network infrastructure and measure QoS parameters, to ensure that active application requests' requirements are continuously met internally. In addition, they listen for internal topology updates, equipment failures and react in such cases by triggering the path computation process to recalculate optimal paths.

C. Inter-Domain Functionalities

Multi-domain e2e paths are constructed in a hop-by-hop fashion, with bilateral negotiation taking place in each hop. An alternative would be to assume the existence of a centralized platform (an orchestrator), in which all underlying domains agree to participate and share information with and coordinates the calculation and establishment of multi-domain e2e paths. However, this might not be desirable for NSPs, which do not want to expose sensitive information, such as their status and policies to third parties. In addition it's hard to decide who will own the orchestrator. Our distributed approach has the advantage that negotiation parties exchange only application request parameters and decide their next-hop based on their own information. Which one of these architectures is more feasible, efficient and could be applied in existing NSP domains is a matter for evaluation and future work.

The negotiation takes over when the application request includes external source or destination IP addresses. Using

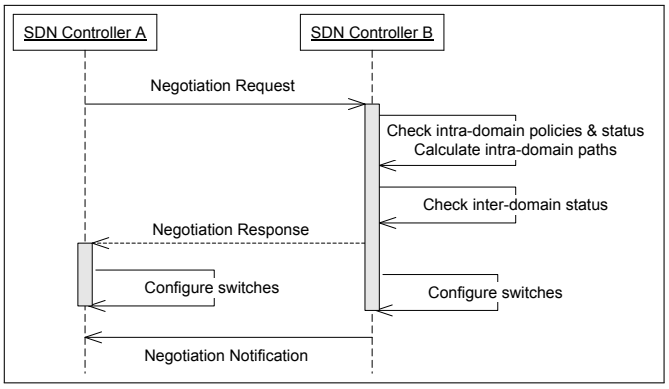

Fig. 2. Two-hop negotiation sequence diagram.

available routing and border information, the SDN Controller selects the next hop and starts negotiating with it to get the required resources. The lifecycle of an inter-domain negotiation, which is loosely based on GRASP negotiation procedures [10], is described below and illustrated in Figure 2.

The Controller of Domain A identifies that the destination address of the application request is external. Based on BGP data, it identifies that one of the best next-hops is Domain B, retrieves the border data from its local data store and sends a Negotiation Request to the SDN Controller of Domain B which includes the e2e application requirements.

When Controller B receives such a message, it compares the request to Domain B's internal policies and performs intra-domain admission control as described in section III-B. If there are configured internal paths which can be applied to the request, it then updates them to meet the additional requirements. Otherwise, it will calculate new paths, retrieve measured QoS and compare it to the requested. Furthermore, it will also check the peering link statistics. If there are available resources and the request is compliant with internal policies, it then replies to Controller A with an ACCEPT Negotiation Response, otherwise with a REJECT. There is also the option that Controller B responds with a NEGOTIATE message, when there are available resources but initial application requirements cannot be fully met. Then the response will include the alternative values which are feasible internally.

If Controller A receives ACCEPT, then the two-domain negotiation is considered successful. If the response is REJECT, the application will be served with best effort QoS. When NEGOTIATE is received, Controller A could either accept the offered values, considering its latest links statistics and the generic QoS requirements for the application type, or propose new values with another Negotiation Request message. The bidding process between two domains should not take more than two rounds, to prevent delaying the e2e negotiation.

When the two negotiation parties agree on the specified values, they can both configure their intra-domain paths and peering link. Each controller configures its SDN Switches with appropriate flow rules and applies its internal traffic differentiation mechanism, using Openflow or OVSDB protocol. If the destination address is not part of Domain B, then it is assumed that Controller B will repeat the process.

Eventually, a multi-domain e2e path is set up. SDN Controllers will inform their peers that the reservations are still valid, by sending a CONFIRM Notification. If negotiation between any parties has failed and there was no alternative, CANCEL Notifications are sent backwards so that SDN Controllers can release resources and delete inserted flows.

After a successful negotiation, inter-domain link status and QoS metrics are periodically checked, ensuring that application QoS requirements and SLA parameters are met. Applications may also monitor their end-points, check the received e2e QoS and react with new requests. In case of inter-domain link failure, a few measures are taken. If the affected domains have other additional links, then CANCEL negotiation notifications are sent, in order to release resources and delete configured flows. Then the next best one is selected based on its current status and statistics and the two controllers renegotiate using the aforementioned process in order to agree on the inter-domain interface and reconfigure their internal paths. When the negotiation is not successful, then the whole inter-domain e2e path has to be released, using CANCEL negotiation notifications and calculated from the beginning. A complete e2e negotiation has to be executed, which might lead to lost packets until e2e paths are set up. Another approach assumes that domains might keep a proactive plan for such cases, with less-prioritized flow rules, activated for such cases.

IV. Implementation and Evaluation

We implemented a proof-of-concept prototype for SDNI and used a virtualised testbed to validate and demonstrate the feasibility of the approach designed. We estimated the performance of the prototype by measuring the time it takes to negotiate resources between two domains.

A. Prototype

SDNI implementation is based on ODL Lithium SR3, a versatile, production-quality controller supported by the Linux Foundation. Its modular architecture allows us to use only the services we need and load them as modules that perform specific network functions and implement southbound protocols.

Functionalities of the implemented prototype include the logic for the Operator, Application and Negotiation interfaces and depend on certain ODL modules to monitor the underlying topology, retrieve link statistics, calculate intra-domain paths, QoS metrics and configure SDN Switches. The three interfaces are exposed as extensions of ODL's HTTP REST API and are defined and generated using YANG configuration files. Minimal resilience implementation has been done to detect intra-domain link failures and setup new paths in real-time.

B. Evaluation Setup

The evaluation topology shown in Figure 3 comprises of two physical servers, each one emulating a single domain, interconnected through a virtual router which in our case is passive and does not participate in the resource allocation scheme. Each domain consists of 5 virtual machines (VMs),

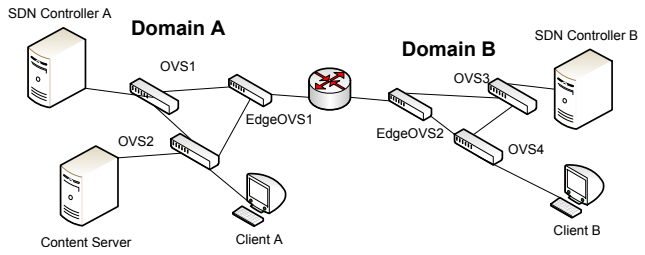

Fig. 3. The SDNI testbed.

three SDN Switches, one SDN Controller and one client. All VMs are configured with 1 vCPU, 10GB HDD and 1GB RAM, (except SDN Controllers which have 4 vCPUs and 6GB RAM) and have Ubuntu 14.04 installed. Each SDN Switch is an Open vSwitch (OvS) version 2.0.2 [11], acting as a single bridge with as many ports as their virtual interfaces and provides an additional management interface to connect with their attached SDN Controller. Two queues have been set up on all the switches: one for best effort and one for premium traffic. This mode of operation aims to show the validity of our approach with real and generated application traffic.

For the evaluation of our approach, we validate that application requests are negotiated between the two domains using a Python script which performs thousands of them. It measures the time needed for two peering domains to negotiate and eventually configure the inter-domain switches. Besides measuring the internals of our implemented ODL module, OpenFlow packets are captured in the management interface of inter-domain switches using Wireshark [12] and the additional configuration time from SDN Controller to OvS is monitored.

C. Results

We consider the signalling time for two domains to negotiate and setup a path as the key performance indicator for SDNI. Three time intervals are identified and measured in order to evaluate the proposed approach:

1) The time between the application request arriving and the negotiation request being sent to the peer domain, t_{intra}. This time includes the checking of intra-domain and inter-domain link capabilities and path calculation.

2) The time between sending a negotiation request and receiving a response, t_{neg}.

3) The time needed to configure the border switch with the relevant flow rules, t_{switch}.

For 5674 requests performed with a rate of 10 requests/second, average values were $t_{intra} = 2.5ms$, $t_{neg} = 8.59ms$ and $t_{switch} = 1.62ms$, resulting in average overall setup time of $t_{overall} = 11.16ms$, with a 95% confidence interval 10.99 to 11.33ms. Negotiation time includes the time required by the peering SDN Controller to monitor and setup its internal path and configure its switches and the RTT between the two domains. Average ping time between the two Controllers was measured to 3.77ms, hence the internal operations in the adjacent domain require similar time compared to the one which started the negotiation. We admit that the negotiation time is affected by the number of AS hops and RTT between controllers. Complex intra-domain topologies would also impact our results. Initial findings for two domains would not affect most business applications. Future work will evaluate our approach under more realistic conditions.

V. Conclusion

We presented a lightweight protocol that employs SDN Controllers belonging to neighbouring NSP domains in order to negotiate the reservation of e2e resources for business applications. Negotiation is performed in a hop-by-hop fashion, exchanging only the required QoS and traffic characteristics. SDN Controllers map those characteristics to domain-local policies and resources, without exposing sensitive information to other parties. A proof-of-concept prototype has been implemented and a testbed emulating two domains has been set up. Preliminary results show that negotiation overhead is small enough to leave most business applications unaffected.

As future work, the authors will look into resilience and scalability aspects, such as properly handling inter-domain link failures and aggregating flow rules. Those will be evaluated for the multi-domain case in order to understand how they affect application traffic. In addition, the existing testbed will be extended to emulate more domains and inter-domain links, as well as more complex intra-domain topologies. The aim is to evaluate and validate the hop-by-hop approach under more realistic conditions and to compare it with the orchestrator-based approach. A longer-term goal includes submitting a draft to the IRTF SDN Research Group.

References

[1] A. D. Ferguson, A. Guha, C. Liang, R. Fonseca, and S. Krishnamurthi, "Participatory Networking: An API for Application Control of SDNs," *SIGCOMM Comput. Commun. Rev.*, vol. 43, no. 4, pp. 327–338, Aug. 2013. [Online]. Available: http://doi.acm.org/10.1145/2534169.2486003

[2] K. Nichols, S. Blake, F. Baker, and D. Black, "Definition of the Differentiated Services Field (DS Field) in the IPv4 and IPv6 Headers," December 1998, RFC2474. [Online]. Available: http://tools.ietf.org/rfc/rfc2474.txt

[3] S. Sohail and S. Jha, "The Survey of Bandwidth Broker," School of Computer Science and Engineering, University of New South Wales, Sydney, Australia, Tech. Rep., 2002.

[4] A. Farrel, J.-P. Vasseur, and J. Ash, "A Path Computation Element (PCE)-Based Architecture," August 2006, RFC4655. [Online]. Available: http://tools.ietf.org/rfc/rfc4655.txt

[5] J. Vasseur and J. L. Roux, "Path Computation Element (PCE) Communication Protocol (PCEP)," March 2009, RFC5440. [Online]. Available: http://tools.ietf.org/rfc/rfc5440.txt

[6] "Opendaylight." [Online]. Available: https://www.opendaylight.org

[7] H. Yin, H. Xie, T. Tsou, D. Lopez, P. Aranda, and R. Sidi, "SDNi: A Message Exchange Protocol for Software Defined Networks (SDNS) across Multiple Domains," IRTF, Internet-Draft, 2012. [Online]. Available: https://tools.ietf.org/html/draft-yin-sdn-sdni-00

[8] K. Phemius, M. Bouet, and J. Leguay, "DISCO: Distributed Multi-domain SDN Controllers," *CoRR*, vol. abs/1308.6138, 2013. [Online]. Available: http://arxiv.org/abs/1308.6138

[9] "The OpenFlow Switch Specification." [Online]. Available: http://OpenFlowSwitch.org

[10] B. E. Carpenter, B. Liu, and D. C. Bormann, "A Generic Autonomic Signaling Protocol (GRASP)," IETF, Internet-Draft, 2016. [Online]. Available: https://tools.ietf.org/html/draft-ietf-anima-grasp-04

[11] "Open vSwitch." [Online]. Available: http://openvswitch.org

[12] "Wireshark." [Online]. Available: https://www.wireshark.org