A Decentralized SDN Framework and Its Applications to Heterogeneous Internets

Mateus Santos, Bruno Nunes Astuto, Bruno de Oliveira, Cintia Margi, Katia Obraczka, Thierry Turletti

A Decentralized SDN Framework and Its Applications to Heterogeneous Internets

Mateus A. S. Santos
University of Sao Paulo, Brazil

Bruno Astuto A. Nunes
INRIA, France

Bruno T. de Oliveira
University of Sao Paulo, Brazil

Cintia B. Margi
University of Sao Paulo, Brazil

Katia Obraczka
UC Santa Cruz, USA

Thierry Turletti
INRIA, France *

Abstract

Motivated by the internets of the future, which will likely be considerably larger in size as well as highly heterogeneous and decentralized, we propose Decentralize-SDN, a Software-Defined Networking (SDN) framework that enables both physical– as well as logical distribution of the SDN control plane. D-SDN accomplishes network control distribution by defining a hierarchy of controllers that can "match" an internet's organizational and administrative structure. By delegating control between *main controllers* and *secondary controllers*, D-SDN is able to accommodate administrative decentralization and autonomy, as well as possible disruptions that may be part of the operation of future internets. D-SDN specifies the protocols used for communication between *main controllers* as well as for *main controller-secondary controller*– and *secondary controller-secondary controller* communication. Another distinguishing feature of D-SDN is that it incorporates security as an integral part of the framework and its underlying protocols.

This paper describes our D-SDN framework as well as its protocols. It also presents our prototype implementation and proof-of-concept experimentation on a real testbed in which we showcase two use cases, namely network capacity sharing and public safety network services.

1. INTRODUCTION

Some of the challenges future internets will face, i.e., scale and complexity, are a direct result of the enormous success of their predecessor, the Internet. Most of the "action" fueling these challenges has been taking place at the network "edges" driven by ever increasing use of "smart" mobile devices (e.g., smart phones, tablets, etc.) and ubiquitous access to the communication infrastructure (especially through wireless communication). Another major driving force behind this "evolutionary step" towards future internets is the development and deployment of increasingly sophisticated network applications ranging from cloud-based services, "smart" spaces (e.g., smart home, smart buildings, smart neighborhoods and grids, etc), health care delivery, law enforcement and emergency services, as well as the so-called Internet of Things (or Internet of Everything).

Motivated by the need to facilitate networks, in particular, the Internet, to evolve, the Software-Defined Networking (SDN) paradigm has been introduced. SDN's premise is to decouple the network control- and data planes and thus make deploying new network services and protocols viable especially in production networked environments. The most widely recognized SDN model to-date is the one adopted by the OpenFlow Standard [15] and endorsed by the Open Networking Foundation[1] (ONF), a consortium bringing together industry and academia to explore how SDN can play a role in network innovation and evolution. SDN in general, and OpenFlow in particular, have been receiving considerable attention from both academia and industry. However, as it will become clear in our overview of related work in Section 5, SDN techniques to-date, including OpenFlow, have mostly targeted "managed networks" and thus promote logically centralized control which is ill-suited not only to the scale but also to the level of decentralization and disruption that may be present in future internets.

Scalability and performance degradation resulting from offloading control decisions from the underlying network hardware infrastructure to SDN "controllers" have been studied [27, 19] and proposed solutions have addressed this issue in different ways. For example, DIFANE [28] and DevoFlow [2] offload to the switches new functionalities in order to reduce the load in the control plane. Some such solutions devolve back to the switches control of some flows. However, one can argue that moving

*E-mail addresses: {mateus,btrevizan,cbmargi}@larc.usp.br (M.A.S. Santos, B.T. de Oliveira, C.B. Margi), bruno.astuto-arouche-nunes@inria.fr (B.A.A. Nunes), thierry.turletti@inria.fr (T. Turletti), katia@soe.ucsc.edu (K. Obraczka).

[1]http://www.opennetworking.org

decisions back to the switches is undesirable and conflicting with the principles of SDN.

An alternative is to move control decisions closer to the datapaths. This can also be accomplished by adopting a "physically" distributed SDN control model, which does not contradict decoupling the control– from the forwarding plane. Moreover, a physically centralized controller represents a single point of failure. Onix [11] and HyperFlow [26] propose to physically distribute the SDN controller by means of a distributed file system to maintain a consistent network-wide view in all the controller "replicas". In contrast, Kandoo [5] presents a physically distributed control plane with a two-level control hierarchy. It has at its top layer a logically centralized controller that maintains network-wide state. Most approaches aiming at increasing the scalability and robustness of the SDN control plane have targeted "managed" networks, e.g. data centers and intranets, where it is reasonable to assume the existence of a single, logically centralized administrative authority, as shown in the top part of Figure 1. However, this assumption does not hold in heterogeneous and administratively decentralized internets that may include a variety of autonomously administered networks, such as infrastructure-less self-organizing networks (as illustrated in Figure 1).

In this paper, we propose Decentralize-SDN, or D-SDN, an SDN framework that allows SDN control distribution both physically and logically by defining a *control hierarchy* of *main controllers (MCs)* and *secondary controllers(SCs)*. The Decentralize-SDN hierarchy can be of arbitrary depth, e.g., matching an internet's administrative boundaries. In "smart spaces" type applications, for example, devices within the home will be controlled by the "home" controller independent of the "smart neighborhood" controller and the ISP's controller. Decentralize-SDN enables logically decentralized control through control delegation between different levels of the control hierarchy as shown in the bottom part of Figure 1. This is accomplished using D-SDN's *MC-SC* communication. The proposed framework also enables services such as inter-domain routing that require cooperation and coordination among MCs via a *MC-MC* communication protocol. D-SDN also defines a *SC-SC* protocol that allows SCs to communicate. For instance, we use SC-SC communication to address control plane fault tolerance. Another distinguishing feature of D-SDN is that it incorporates security as integral part of the framework and its underlying protocols.

To-date only a few efforts have explored logically decentralizing SDN control. One notable example is DISCO [10], which mainly targets control decentralization to address inter-domain routing. The goal of D-SDN goes beyond enabling a specific network function or application; it aims at providing a general framework that can be used to enable a wide range of current- as well as future network services and applications. To this end, D-SDN supports control distribution both physically– as well as logically.

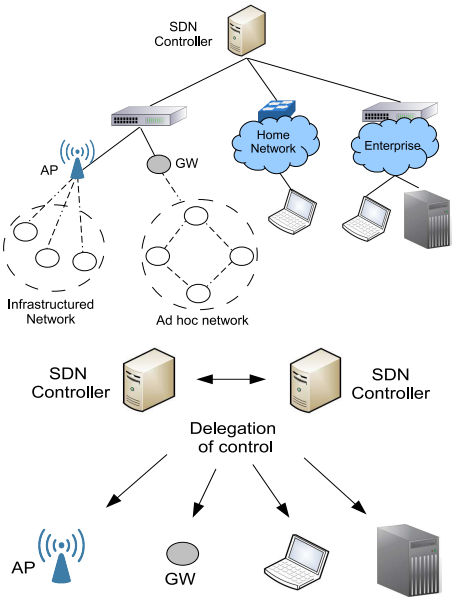

Figure 1: SDN control distribution

As proof of concept, we apply the D-SDN framework in two use cases, namely: (1) network capacity sharing, in which control decentralization enables nodes in a infrastructure-less network to connect to the Internet via other (connected) nodes, and (2) public safety network (PSN) scenario which showcases the need for control decentralization in emergency response services.

The remainder of this paper is organized as follows. In Section 2, we describe the D-SDN framework. In Sections 3 and 4, we elaborate on the implementation and on the evaluation of our SDN framework, respectively. Section 4 also includes the use cases adopted as proof of concept. We review prior works in Section 5. Finally, in Section 6, we present the final discussions.

2. D-SDN OVERVIEW

In this section, we first present an overview of the Decentralize-SDN (D-SDN) framework. Then, we describe the protocols that support communication between the D-SDN controllers as well as D-SDN's fault tolerance and security mechanisms.

2.1 Overview

D-SDN defines two types of controllers: *Main Controllers* (MCs) and *Secondary Controllers* (SCs). The main difference between them is that SCs require that MCs authorize and delegate control to them before SCs are able to act as SDN controllers. In addition, we envi-

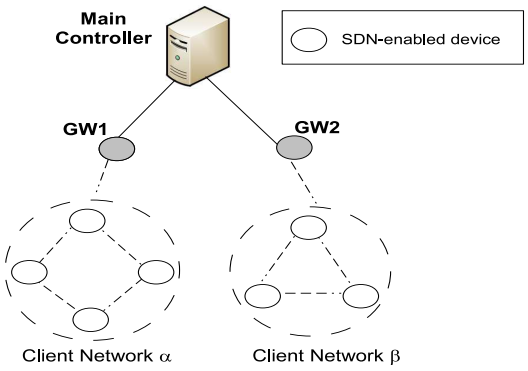

Figure 2: Distribution of control from MC to SCs to control mobile ad-hoc networks.

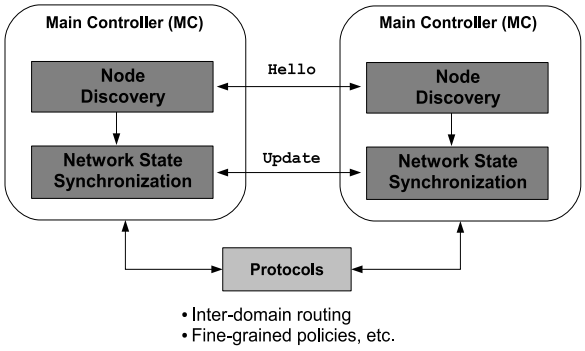

Figure 3: Communication between main controllers.

sion that SCs will typically be responsible for managing SDN switches in a sub-domain within the MC's domain.

Figure 2 shows an example scenario where an MC controls two ad-hoc networks (MANETs) that connect through. Mobile devices in the MANETs are SDN-enabled and rely on the MC's forwarding decisions. Thus, a new flow in the MANET generates a request from the corresponding mobile node to the MC, which responds with the appropriate flow modification messages to the mobile device. In the example of Figure 2, using D-SDN, gateways can act as SCs upon MC authorization and delegation. As a result, flow modification packets would not need to reach the MC and could be controlled directly by the corresponding SC. The ability to control MANETs locally is important especially in scenarios where they are not always connected to the infrastructure.

Hierarchy of Controllers

In D-SDN, control distribution is achieved using a hierarchy of MCs and SCs which can also be used to improve control plane availability and fault tolerance. Following the hierarchy, MCs can *delegate* control of certain devices to a particular SC. For example, an SC would not be allowed to write new flow entries to a device's flow table without a delegation from the corresponding MC. Note that SCs must have been previously authenticated by the MC or some other trusted third-party authority before being able to participate in the network control plane.

Delegation Process

An MC can delegate the control to an SC with respect to a set of SDN-enabled devices. Delegation can be initiated by an MC or can occur upon a request from the SC. A delegation request can be triggered by different kinds of events. For example, when a new SC is deployed geographically closer to a set of devices, it could request delegation from the MC to control these devices. Another example is a scenario in which mobile devices in a MANET need connection to the Internet through a gateway node. The gateway can then request authorization from the MC for playing the role of an SC to the new devices that joined the network.

2.2 Controller-to-Controller Communication

Our framework does not assume that MCs are configured in advance with information about other controllers. Instead, controller discovery is performed in a dynamic fashion which allows mobile devices to act as secondary controllers.

MC-MC Communication

Communication among main controllers is mainly based on `Hello`, `Update` and `Error` messages. A `Hello` message includes the *controller identity*. The content of an `Update` message depends on the service being supported. Section 2.3 provides a concrete example of an `Update` message. An `Error` message includes data regarding events such as an error on the application module or the network operating system.

Figure 3 illustrates an sample message exchange between MCs.

1. Controllers exchange `Hello` messages to discover neighbors;
2. Reachable neighbors exchange `Update` messages between themselves;
3. `Error` messages are sent in case of an event is detected.

Thus, `Update` messages allow network state to be exchanged between MCs. Note that in our framework, each controller can have its own database. Thus, controllers are not necessarily sharing a network file system as in some logically centralized SDN control implementation.

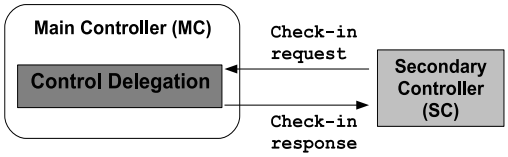

Figure 4: Delegation of control from main- to secondary controller.

SC-SC communication follows a similar procedure. Section 2.3 elaborates on particular contents for the `Update` message in order to provide fault tolerance among SCs.

MC-SC Communication

As previously pointed out, a network device is only able to act as an SC upon the authorization of the corresponding MC through a `Control Delegation` message. In addition, MC-SC communication usually happens within the same administrative domain. Control delegation is illustrated in Figure 4 and proceeds as follows:

- `Check-in Request`: an SC requests authorization for managing a specific SDN-enabled device, which can be attached to an identity for accounting purposes. The MC, upon accessing its database, authorizes or denies access;

- `Check-in Response`: a positive feedback grants the SC the ability to respond to further requests of the new network device. Non granted access is also acknowledged to an SC, which, in this case, will not be allowed to control the SDN-enabled device that required services.

If the MC needs to revoke control from an SC, then it sends a `Check-out` message to the SC. In addition, the MC should maintain a *delegation table* in order to manage delegation requests from other SCs.

2.3 Fault Tolerance

In the case of failures, controllers can use the appropriate controller-to-controller protocols to elect a new controller. In particular, in the PSN case study presented in Section 4.3, we show how SCs use D-SDN's SC-SC protocol to implement fault tolerance in case of failure of the current SC.

Inspired by OpenFlow (OF) 1.3 [17], we differentiate *active* controllers from *read-only* controllers. OF specifies such roles as *master* and *slave*, respectively. OF includes the *equal* role, which is similar to the master with the exception that there can be at most one master per switch. Slaves do not receive messages from a switch. For example, they neither receive *packet-in* messages nor acknowledge of flow modification. However, they can become active by sending a *role request* message.

The concept of active and read-only controllers is important to allow synchronization for fault tolerance. The event of a failed *master* controller can be detected by other nodes if the master stops sending `Update` messages. Such messages, thus, are not only used for synchronization, but also as a keep alive flag.

The content of the `Update` message for fault tolerance is defined as follows:

- **list of applications**: application modules currently running on the controller;

- **controller role and respective switches**: a list of the switches under the controllers' administration for the master, slave and equal roles.

Even though the aforementioned controller roles are only specified in the Openflow standard, other SDN implementations would adopt a similar method.

Triggering Controller Replacement

In order to initiate fault recovery, a *timeout* value is defined for receiving `Update` messages from the "master" SC controller. If a failed master SC stops sending messages, after the *timeout* interval, the participating SCs will start an election protocol.

In the case that only a fraction of the controllers do not receive `Update` messages, another master will be elected only if there is enough quorum. Controllers that still receive `Update` messages will not participate in the election protocol. If a new master is elected, it will inform the corresponding devices that will be under its control. These devices will then remove the old master from the master role (whether it actually failed or not.)

2.4 Security

In order to provide security services such as confidentiality, integrity and mutual authentication, many cryptographic schemes could be used. We next discuss some trade-offs when choosing a particular cryptographic method and show for every design decision made how the method adopted increases the resilience of our framework to attacks such as impersonation, man-in-the-middle and replay-attacks. Further, in Section 3, we elaborate on the implementation adopted in our framework.

Secure Controller-Controller Communication

Secure communication among MCs, whether it is inter– or intra-domain, can be implemented by means of public key cryptography. In particular, an MC should possess each recipient's (other MCs' and SCs') public key in order to encrypt a message. Integrity and authentication can be provided through digital signatures, i.e., using

the origin's private key and, thus, requiring the recipient to have the public key of the signer.

However, some practical challenges raised by the technique described above need to be addressed. Public key distribution schemes will need to be implemented. An alternative would be storing the necessary keys before deployment; however, newly deployed controllers will require changes in already deployed devices. Additionally, public key validation will be needed to ensure that a public key received from the network is indeed the correct one. This is usually implemented using a Public Key Infrastructure (PKI), which then requires a trusted entity, known as Certification Authority (CA), to be online to respond to certification requests (i.e., a certificate for the requested public key and identity).

PKI is not well suited to MC-SC communication because SCs can also be instantiated on end user devices. Generation of certificates for each SC would not be practical due to maintenance costs. Moreover, it may not be possible for an SC to access a CA, since mobile nodes may be subject to intermittent connectivity (recall that SCs might be nodes in a MANET).

Security Inside SDN Domains

An SDN domain is composed of MCs, SCs, SDN-enabled devices and general end hosts. A security scheme should take all those entities into account. We argue that Identity Based Cryptography (IBC) [24, 1] is suited for providing security services in an administrative domain.

IBC allows a user to calculate a public key from an arbitrary string. Choosing the users' identity as a public key has advantages such as: (1) there is no need to verify the public key using an online CA; and (2) a user only needs the recipients' identities in order to calculate public keys (i.e., there is no need to ask for public keys). In addition, cryptographic protocols are simple and efficient under the paradigm of IBC.

Given that a user's public key is tied to an unique identity, the issue becomes how to obtain the corresponding secret key. A Trusted Third Party (TTP) is responsible for secret key generation, which is performed by using the TTP's secret key, also known as *master secret key*, and the public key of the target user.

Note that all secret keys can be computed by the TTP. Fortunately, in the scenario explored here, there is a synergy present between controllers and TTPs. Controllers can be considered as trusted entities, since they provide interfaces to applications that perform management tasks. Thus, in the context of IBC, a controller could be responsible for generating (and possibly distributing) private keys to users in its domain.

In Section 3, we elaborate on the implementation of IBC integrated with our framework. We also provide details on the method used for key agreement, which allows the use of efficient cryptographic schemes.

3. IMPLEMENTATION

The concept of SDN is centered around controlling forwarding policies and has been thus far put into practice in infrastructure-based networks. However, in heterogeneous and possibly self-organizing environments, the devices involved in data forwarding are also end devices themselves. Consequently, end devices should be able to perform the same functions as SDN switches, e.g., communicate with controllers and understand how to handle forwarding roles. Since end devices in infrastructure-less networks are typically portable and do not have access to continuous power sources, the deployment need to be lightweight in terms of code, storage, communication, and power consumption footprint.

In this work we consider that mobile nodes can instantiate a software switch, which we refer in the paper as *Virtual Switch (VS)*. From an implementation point of view, a VS can be an OpenFlow software switch; thus, a mobile node can be an *SDN-enabled device* that is responsible for forwarding incoming traffic, maintaining flow tables, and communicating with the controller when needed.

Our framework is comprised of a *server-side* and a *client-side*. The server-side exposes an interface to a hierarchy of controllers. The client-side provides accounting data to the servers as well as management of cryptographic material that is used for providing security services such as data confidentiality and authentication.

Applications along with the client-side of our framework can run in a virtual machine attached to the VS. Devices that run the server-side provided by our framework are denoted as *HSDN-enabled*. Figure 5 illustrates the device categories defined by our framework. Note that a mobile node can be connected via wire or wireless to an SDN switch and run the client side of our framework.

HSDN-enabled devices are mainly controllers and special purpose nodes such as gateways for interconnecting infrastructure-based and infrastructure-less networks [16]. Such nodes can play an important role in the network, since they are responsible for extending the network connectivity and relay upstream and downstream traffic.

3.1 SDN-enabled Devices

SDN-enabled devices can be instantiated by using OpenFlow-enabled Network Interface Card (NIC)[2]. Thus, mobile nodes can deploy an embedded virtual switch in low-latency NICs. On the other hand, one could use software-based solutions to instantiate SDNs. With

[2]http://www.projectfloodlight.org/blog/2013/03/21/demo-openflow-enabled-network-card/

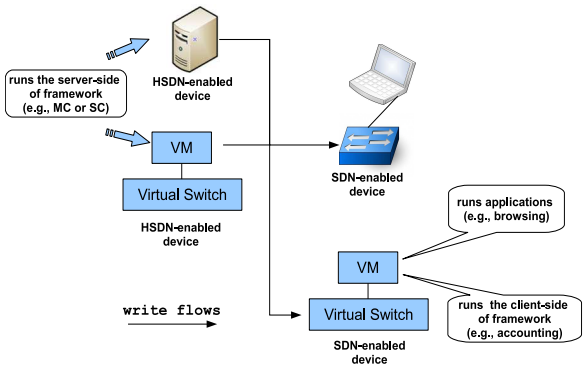

Figure 5: Node category defined by the framework: SDN and HSDN-enabled devices.

some overhead, it allows flexible and simple deployment in production environment (e.g., customers are not provided with hardware and only install software).

Given the aforementioned trade-offs, we adopted OpenVSwitch (OVS) for providing SDN functionalities to mobile nodes in our implementation. OVS can be managed by OpenFlow through TCP connections. A concern is whether a Virtual Machine (VM) over OVS compromises the controller performance. In Section 4.1, we argue that it is not generally true by showing some experimentation results. This setting is most promising for SCs because they can be mobile devices. In addition, SCs are under a limited scope, since they only participate in intra-domain communication and manage certain devices upon MC authorization.

In the following Section 2.4, we discuss some of the trade-offs when implementing security.

3.2 Security

Here, we provide further details on the particular security schemes adopted. In our proposal, MC-MC communication can be implemented using Transport Layer Security (TLS) [8] over a Public Key Infrastructure (PKI). We do not provide specific implementation for such scenario, since TLS is a well known standard. On the other hand, we elaborate on the MC-SC and SC-SC communication as well as on secure communication between the client- and server-side or our framework (e.g., end-host authentication). Notice that the OpenFlow standard defines TLS for securing the communication between an SDN-enabled device and a controller; thus, this particular secure channel is not investigated here.

Applying IBC to SDNs

As discussed in Section 2.4, there is a synergy between controllers and TTPs. In particular, MCs can play the role of a TTP. As a result, our IBC implementation inside an administrative domain defines the following cryptographic protocols:

ID_X	identity of node X
s	master secret key (controller's secret key)
S_X	private key of node X
P_X	public key of node X (derived from ID_X)
$K_{X,Y}$	key established between nodes X and Y
ctr	counter
$\texttt{authenc}(msg, k)$	authenticated encryption of msg using key k
$\texttt{enc}(msg, k)$	encryption of msg using key k
$\texttt{dec}(msg, k)$	decryption of msg using key k
mac	authentication tag
$e(\cdot, \cdot)$	pairing function

Table 1: Notation.

- **Setup:** defines how security is bootstrapped in a procedure that is performed once;
- **Key agreement**: uses pairing functions [1, 21], identities and cryptographic material to agree on pairwise-keys;
- **Handshaking**: the process of authenticating a requesting device (e.g., end-host device or SC). The authenticator can be an MC or an SC.

Next, we elaborate on the aforementioned protocols using the notation presented in Table 1.

Setup

As public keys are derived from identities, the TTP (i.e., the controller) maps the node identity, ID_X, to a point in the elliptic curve, P_X. This mapping is a public parameter, since a node is allowed to generate any device's public key. The TTP generates a master secret key s and calculates each node's private key as $S_X = sP_X$. This value should be either sent privately by the TTP or pre-deployed on the device (i.e., SC or end-host device).

Authenticated Key Agreement

Pairings[1, 21] provide practical implementation for authenticated key agreement (AKA) over IBC, which is an elegant alternative to non-authenticated schemes such as the Diffie-Hellman interactive key exchange. We informally define pairings as follows.

Let q be a prime number and $\mathbb{Z}_q^*$ be a cyclic group of the integers modulo q. Let $\mathbb{G}$ be a cyclic additive group and $\mathbb{G}_T$ be a cyclic multiplicative group of the same order q. Then, $e : \mathbb{G} \times \mathbb{G} \rightarrow \mathbb{G}_T$ is a map that satisfies (1) bilinear: $e(aP, bQ) = e(P, Q)^{ab}$ $\forall P, Q \in \mathbb{G}$ and $\forall a, b \in \mathbb{Z}_q^*$; (2) non-degenerate: $e(P, P) \neq 1$; and (3) computable: there exists an efficient algorithm to compute $e(P, Q)$ $\forall P, Q \in \mathbb{G}$. The group $\mathbb{G}$ can be implemented using a group of points on an elliptic curve and the group $\mathbb{G}_T$ using a subgroup of a finite field. In particular, we use pairings referred to as *Type 1* [21], with the property that $e(P, Q) = e(Q, P)$.

The AKA procedure considered here has the main goal of avoiding public key encryption. It means that,

once a key is agreed between two nodes using public key cryptography (e.g., IBC), they can use the shared key for confidentiality and data authentication. This method is very efficient because it uses symmetric cryptography for the most part of the overall message exchange. Symmetric algorithms for authenticated encryption, encryption and decryption are denoted in the paper as `authenc`$(\cdot)$, `enc`$(\cdot)$ and `dec`$(\cdot)$, respectively. We refer to our prior work [18] for further detail and examples of algorithms.

We employ the Sakai, Ohgishi and Kasahara (SOK) protocol [21] in AKA procedures. Thus, a private key is used together with identities for computing a shared key. As an example of the SOK protocol, assume users A and B want to agree on a shared key. Consider that they already received their private keys S_A and S_B, respectively, from a TTP. In addition, each user can obtain the public key of the other by using a public function, denoted as $\phi(\cdot)$. For example, user A obtains B's public key by computing $\phi(ID_B)$. To agree on a key, each user computes the pairing, which has the user's private key and the recipient's public key as inputs. For instance, A computes $K_{A,B} = e(S_A, P_B) = e(sP_A, P_B) = e(P_A, P_B)^s$ due to the bilinearity property. A similar procedure can be done by the user B, resulting in $K_{B,A} = e(P_B, P_A)^s = e(P_A, P_B)^s = K_{A,B}$.

The AKA is required by the handshaking procedure, which we describe next.

Handshaking

In the handshaking procedure, a new coming device, or requesting node (RN), is required to respond to a challenge, so that the authenticator is able to verify the devices's identity. It is worth noting that both parties should know each other's identity for exchanging authentication messages. This allows them to compute a shared key, which is used for authenticated encryption of the challenge. Thus, the authenticator can encrypt a message to the requesting node, which decrypts the ciphertext to obtain the challenge that can be encrypted again and sent back to the authenticator. Figure 6 shows in detail this process, in which an RN and an authenticator use a counter in order to protect messages from *replay attacks*. Note that the pair RN-authenticator can be any pairwise combination among MC, SC and end-host devices.

Figure 7 illustrates how the agreed keys can be used by means of symmetric cryptography. In particular, the figure shows how the messages are securely exchanged in order to provide the delegation of control (i.e., using the messages defined in Section 2).

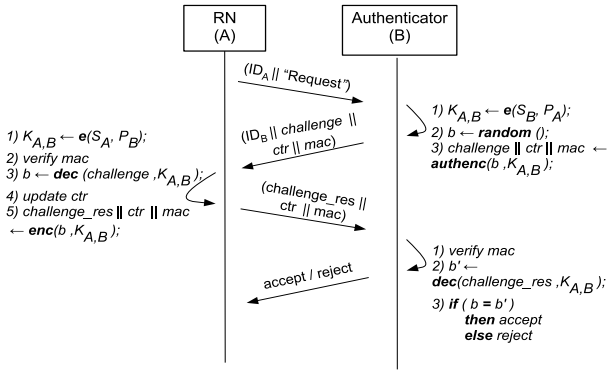

Figure 6: Detailed handshaking procedure.

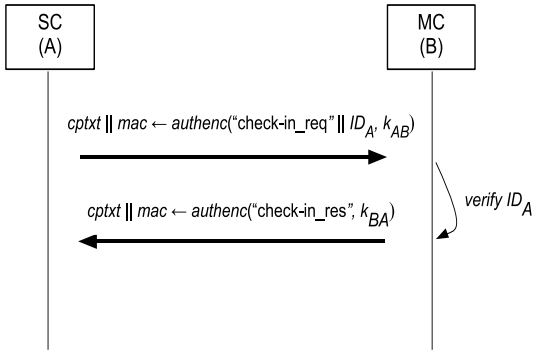

Figure 7: Authenticated key-agreement.

4. EVALUATION

In this section we evaluate our framework with experimentation on HSDN devices and on two use cases: *secure capacity sharing* [23] and *public safety networks (PSNs)* [22]. The main goal is to verify that the proposed framework is able to provide services over different implementations and application scenarios.

Regarding the HSDN device, which is the first experimentation described in this section, we investigated the performance of a controller running a VM over OVS. We compared this method to the setting of a standalone controller.

The first use case considers a node that can only connect to the Internet if a wireless gateway is available and willing to share its resources. Mobility can trigger a handover procedure, in which an node changes from one gateway to another. In addition, an MC delegates functions to SCs instantiated on network gateways, in which QoS policies can be applied. Mobile nodes, including gateways, are connected to an infrastructureless network.

In the second use case, we extend the capacity sharing scenario in order to study SDNs over a MANET, so that the control plane faces the challenge of dealing with connection disruptions. In particular, the scenario of a PSN is explored by providing resilience to failures in the control plane.

4.1 HSDN Device Performance

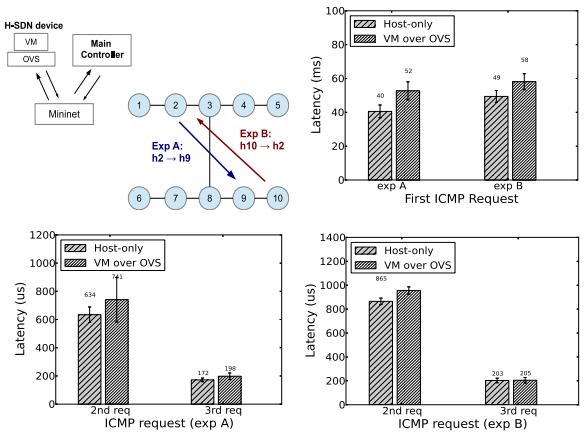

Figure 8: Impact of HSDN device.

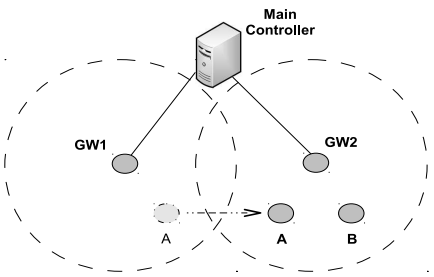

Figure 9: Upon authorization of a MC, GWs provide services to nodes that can experience handover.

We investigated the network performance of an HSDN device in two settings: a host-only machine and a VM over OVS. As controller, we used the Floodlight running a forwarding application. SDN-enabled devices were instantiated in an emulated network running in Mininet [13]. The experiments were carried out using separated machines for the Mininet network and the controller. Regarding the scenario, we created two interconnected networks and two ICMP flows, one for each individual experiment. We collected 10 samples and used a 95% confidence interval level when reporting our results.

Figure 8 shows the scenario and the results. The first ICMP request presents the most significant difference between the host-only and VM approach (about 10 milliseconds, as shown in the top right graph). However, it is only the first request, which is expected to experience high latency in SDNs due to switch-controller interaction. The other graphics (bottom right and left) show that the second and third requests experience a much lower difference between the two settings (i.e., host-only and VM), since they are on the order of microseconds. In particular, the difference of the third request was negligible for both experiments due to flow creation.

4.2 Secure Capacity Sharing

In this section, we examine a use case where data sources are not directly accessible via the Internet, but through a capacity sharing application enabled by the proposed architecture. In this scenario nodes may choose to share part of their bandwidth to further broaden network coverage. For the following use case, we assume the network model illustrated in Figure 9, where a node "A", called here the "Requesting Node" (RN), wishes to connect to the Internet and accesses, for example, the World Wide Web. However, it is unable to connect to the existing network infrastructure (e.g., because A is out of range of the closest AP). Another node called gateway node 1 - "GW1", advertises its gateway services providing A the option to connect to the Internet through it. Note that A can connect to GW1 directly or through a wireless, multi-hop ad-hoc network (MANET) using some existing MANET routing protocol to route packets towards the GW1.

A gateway node (GW) is equipped with SDN switching capabilities as it plays the part of a software switch. It is responsible for receiving connection requests from RNs and authenticating them. Upon arrival of a RN (i.e., node A in our example), a GW also requests from the MC in its domain information about services to be provided, actions to be taken and policies to be implemented, such as checking availability of resources, implementing packet filtering rules, authentication the incoming RN, etc. There can be a number of GW nodes present and they should be able to perform other functions, such as handover of RNs and QoS policy enforcement.

Figure 9 also shows the scenario adopted for experimentation. As we elaborate next, we employ the controller hierarchy of the proposed framework by using the messages `Check-in Request` and `Check-in Response`.

4.2.1 *Steps for Secure Capacity Sharing*

We focus on the secure admission of the node A through the gateway as well as on secure handover and the enforcement of QoS policies. To control the access of network resources it is required not only to authenticate a new node A, but also to ascertain membership eligibility and bootstrap security services such as data confidentiality and authenticity.

The main steps for secure network access extension using SDN, illustrated in Figure 10, are as follows:

- **Gateway discovery:** GW nodes send periodic messages, announcing their gateway capabilities. The potential users, on the recipient of such messages will choose a GW, by sending a `Request` message to the most suitable candidate;
- **Handshaking**: a GW node responds to a user

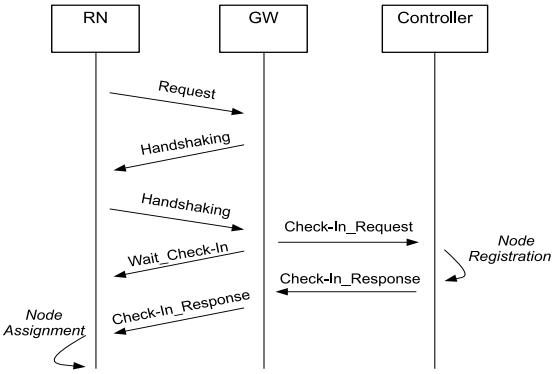

Figure 10: High level description for node authorization through main controller.

request and initiates a handshaking procedure for node authentication;

- **User check-in**: the GW requests authorization to the main controller, which queries its database in order to approve allocation of resources to the designated customer.

If a user is authorized, the main controller adds the new flow-table entries to the forwarding devices on user data path towards the Internet. The procedure of user check-in includes the *delegation of control* from the MC to the gateway with respect to user device administration.

4.2.2 Secure Handover

The scenario conceived here has an infrastructure-less network comprised by GW nodes and users. The former is able to run the server-side of our framework and thus, instantiate a "local" SC for the infrastructure-less network. As a first step to orchestrate ad hoc networks using decentralized SDNs, we explore secure handover. In particular, a user changes from one gateway to another.

A handover can be triggered by many events. Here, we adopt the scenario in which a user notices that a *more suitable*[3] GW becomes available. The user itself can send a request to the new candidate and perform a handshaking procedure. Then, the MC can orchestrate flow creation and removal in the new and old gateways, respectively. The algorithm of Figure 11 shows how secure handover is performed using our framework.

In order to demonstrate the handover, we generated a sequence of HTTP requests to an external web server (located outside the local network) and measured the throughput. We collected 10 samples for each element of the sequence and report a 95% confidence level in our

[3]The formal definition of "more suitable" is out of scope here. For the sake of completeness, we use *closest in range* as an example.

ALGORITHM: Secure Handover of node A from gateway GW_i to GW_j
ASSUMPTION: User A has already been granted access by MC through GW_i

1. GW_j announces its services
2. **if** GW_j is a better candidate than GW_i **then**
3. A sends a `Request` to GW_j
4. Handshaking is performed between A and GW_j
5. **if** A is authenticated by GW_j **then**
6. GW_j sends `Check-in Request` message to MC
7. MC queries node A in its database
8. **if** A is authorized by MC **then**
9. MC sends `Check-in Response` to GW_j
10. GW_j inserts new flow table entries for node A
11. GW_j acknowledges A
12. MC sends `Check-out` to GW_i
13. GW_i removes the flow table entries related to A

Figure 11: Algorithm for secure handover using the proposed framework

results. Figure 12 shows the results, in which *effective handover* points to the first HTTP request after the new gateway took over. It can be seen from the figure that the throughput fluctuates so that the handover cannot be observed among different HTTP requests. In this particular case, both gateways presented similar performance. We emphasize that our goal is not to increase performance among gateways, but to provide seamless handover.

4.2.3 QoS and Gateway Redundancy

Quality of service can be enforced by MCs or SCs by using ingress policy rates. In the same scenario of Figure 9, a gateway would prevent RNs from allocating more than a determined fraction of the total bandwidth provided by the ISP.

As in the handover scenario, we consider that an SC can be a GW capable of managing its SDN-enabled device (e.g., server-side of the framework running on a virtual machine atop the software switch). Thus, in this particular example, the ingress policy rate would be applied locally by limiting the incoming wireless traffic.

We carried out experiments using one single gateway with the ingress policy set to 3 kBps. Then, another gateway with no restrictions becomes available as a redundant channel to the infrastructured network. We measured the throughput during sequences of HTTP requests to a web server located in the same prefix network. Figure 13 illustrates network performance when applying ingress policy rates to the capacity sharing scenario. Network performance is limited to the configured

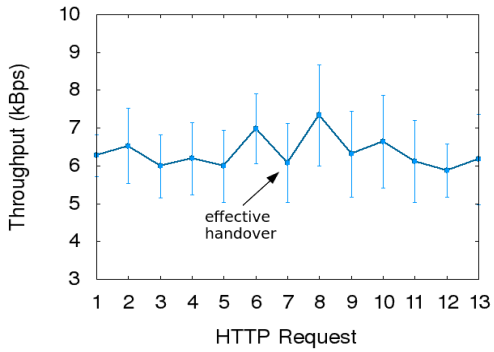

Figure 12: Throughput before and after handover.

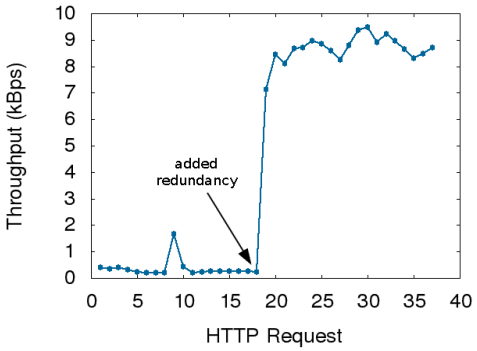

Figure 13: Throughput before and after the event of activating a redundant gateway with no QoS enforcement policy.

throughput as long as the redundant (non-restricted) GW is not activated.

QoS policies could also be used by service providers in order to offer differentiated services among users. Examples include scenarios in which customers that share their resources might get incentives to do so by means of higher ingress policies, while customers provided with temporary shared services are subjected to lower bandwidth.

Additional gateways can be used in other scenarios, regardless of QoS policies. For example, they can be useful in the case of a gateway with many users attached to it, which might generate a performance bottleneck, limiting the data throughput.

4.3 Public Safety Networks

We now showcase our proposal by applying it to Public Safety Networks (PSNs), a more challenging scenario where resilience to failures is an important requirement and the heterogeneity and mobility of the devices involved should be considered.

PSNs are built to detect and/or handle disaster events

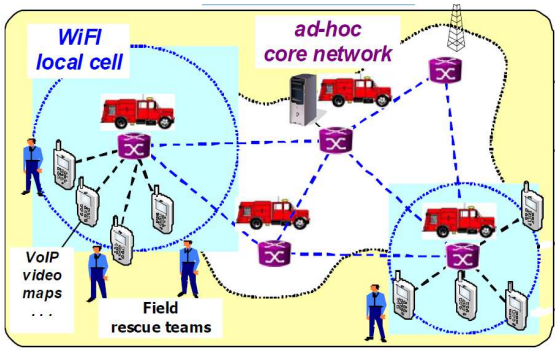

Figure 14: CHORIST broadband scenario for inter-agency communication. Source: [4]

[7, 22]. Such networks are set to provide communication and coordination for emergency responders and operations. Many of the challenges in the PSN field come from the variety of systems and agencies involved in the crisis response and from their mobility at the disaster site [7].

PSNs should experience rapid deployment and fault tolerance. Interoperability is also an important issue in this kind of networks. Different public safety authorities should be able to communicate for operations management.

By decentralizing the control plane, our proposed framework allows rapid deployment, reliability and interoperability. In addition, our testbed incorporates ad hoc networks extending the capacity sharing scenario of Section 4.2.

4.3.1 Decentralization and Fault Tolerance in PSNs

We envisage a scenario in which public safety authorities can organize themselves for exchanging valuable information regarding an emergency situation. In this context, for this use case, we built our proposed SDN framework over an emulated scenario that was based on the CHORIST architecture. CHORIST, which is a project funded by the European Commission, defines an architecture to address environment risk management and citizens' safety. It is considered a seminal work that proposes a network architecture that includes a rapidly deployable broadband scenario [4] at which authorities and emergency responders can exchange messages in the case of public safety event. We showcase our proposal over such a scenario, illustrated in Figure 14. In this figure vehicles are capable of serving as GWs to a network of different agency actors (e.g., firefighters and police officers).

As for SDNs, decentralization and fault tolerance are key features expected in such a scenario. The former is required because agencies could be subjected to different administrations and policies. Moreover, it would

not be efficient to run a centralized controller acting as an *operation center* in the ad hoc network due to delay and disruption connectivity. Regarding fault tolerance, failures would impact intra- and inter-agency communication. We next describe the adopted method for deploying such a scenario.

4.3.2 PSN Testbed

Our testbed instantiates SCs at the agencies' vehicles. A single agency can have many decentralized SCs that exchange messages with other agencies' SCs. Even though an MC can be optionally deployed in an operation center, we consider it would not be available anyway. Our goal is to demonstrate decentralization and fault tolerance in a single agency network[4]. We also consider that no infrastructure network is available. Thus, users (e.g., firefighters) have devices connected via a mobile ad hoc network managed by an agency's SCs.

Given that an MC cannot orchestrate the network, SCs should rely on our framework in order to continue operating correctly in the event of link failures. Figure 15 describes the proposed scenario for implementing tolerance to failures. For the purposes of this experiment, `Hello` and `Update` messages were used as a single message (here denoted as `Hello` only). Another method would be considering `Hello` for controller reachability and then, `Update` for exchanging of specific data, as our framework enables. However, this evaluation includes a heterogeneous scenario in which connection-oriented protocols are not desirable due to the fact that the network might suffer delays or disruptions.

Methodology and Results

The experiments were carried out using four controllers and one switch. A single node was set as master for the switch. All the nodes, including the switch, were configured in a wireless ad hoc network. We integrated the election protocol proposed in [12] with our framework. However, other similar protocols could be considered.

Before presenting the results, we elaborate on the main parameters of the system. Let t_h be the time between periodic `Hello` messages sent by the master controller. Let t_{out} be the timeout, or in other words, the time a non-master (i.e., role equal or slave as defined in Section 2.3) controller waits for receiving the next `Hello` message. Given that $0 < t_h \leq t_{out}$, the worst case scenario for controllers to detect a failure is when the master actually fails just after sending a `Hello` message. The best case scenario happens when the master fails immediately before sending the next `Hello` message, which results in a detection time of approximately $t_{out} - t_h$. Thus, the closer the values of t_{out} and t_h are, the faster it is to detect a failure.

[4]Inter-agency communication would be implemented by means of the capacity sharing use case, using vehicles as gateways.

GOAL: Evaluation of fault tolerance in a MANET, in which SC_i is the active controller. SC_j takes over after SC_i failure.

ASSUMPTIONS: (1) MC does not interfere with the process of fault tolerance (e.g., MANET is disconnected from broader network); (2) SCs are authorized (a priori, by the MC) to take any control plane activities inside the MANET; (3) A single SC, here SC_i, is set as *master* for SDN-enabled devices.

1. SCs exchange periodic `Hello` messages with their identities and roles for each SDN-enabled device, if any exists, under their scope;
2. SC_i fails (e.g., ran out of battery);
3. An election protocol is triggered among SCs due to a *timeout* for receiving `Hello` messages from the *master* controller;
4. The elected controller, say SC_j, requests the administration of the corresponding SDN-enabled devices and effectively replaces the failed controller;
5. `Role Reply` messages from devices confirm that SC_j took over.

Figure 15: Scenario for fault tolerance among the SCs inside a MANET.

Minimum	2.3
Maximum	6.7
Average	4.2 ± 0.5

Table 2: Time in seconds to recover from a failure ($t_{out} = 5$ and $t_h = 3$).

In our experiments, we used $t_{out} = 5$ seconds and $t_h = 3$ seconds. We avoided similar values for t_h and t_{out} because controllers would experience overhead when detecting failures that actually never happened.

We collected 20 samples and computed a 95% confidence interval. We used a random failure time at each sample. Table 2 shows the recovery time, which is not only the time to detect a failure, but also the time it takes for the new master to take control of the switch. The minimum time (i.e., 2.3 seconds) is close to the best case scenario of the failure detection mentioned earlier. This shows that the scenario proposed in Figure 15 can be efficient. On the other hand, the maximum time can be slightly greater than the timeout if our fault tolerance scheme experiences delays or losses. That might occur in a wireless environment such as the one implemented in our experiments.

5. RELATED WORK

An important challenge that arises when offloading control from the networking hardware is the controller placement problem [6], which attempts to determine

both the optimal number of controllers and their location within the network topology, often choosing between optimizing for average and worst case latency. The link latency used for communication between the controller and switches is of great importance when dimensioning a network or evaluating its performance [2]. The work in [19] presented evaluations to show how the controller and the network perform with latency issues on the control link. They concluded that the switch-to-control latency, has a major impact on the overall behavior of the network, as each switch cannot forward data for an interval of time, until receiving the message from the controller that inserts the appropriate rules in the flow table. This interval can grow with the link latency and impacts dramatically the performance of network applications.

Offloading back to the switches new functionalities in order to reduce the load in the control plane was a method proposed in a few previous works, such as DIFANE [28] and DevoFlow [2]. However, this approach might be conflicting with the principles of SDNs.

An alternative would be to move control decisions closer to the datapaths. Previous work such as [11, 26, 25] propose a logically centralized but physically distributed control plane by means of a distributed file system. These approaches consider that all the controllers run the same applications and are under a central administration, since they share a common synchronized filesystem. This is strong a assumption, that can lead to overhead and delay, thus limiting responsiveness which can lead to suboptimal decisions. Such trade-offs on distributing the control plane under a logically centralized scheme are investigated by Levin et al. [14].

Distributed control and the need for dynamic assignment of switches to controllers are addressed in [3], where a mechanism to dynamically handover switches from one controller to the other as needed is proposed. Kandoo [5], on the other hand, proposes a hierarchical distribution of the control plane, where a logically centralized controller maintains the network-wide state. DISCO [10] is a distributed multi-domain SDN control plane architecture that enables end-to-end delivery network services, including QoS. It separates intra-domain and inter-domain information, each performed by a specific part of the architecture.

However, these approaches have been proposed for datacenters and infrastructured networks where the different instances of controllers share huge amount of information to ensure consistency. These approaches fall short when distributing control over heterogeneous network environments. In the case of heterogeneous and possibly self-organizing environments, the devices involved in data forwarding are also end-devices themselves. Consequently, end-devices should be able to perform the functions such as, communicating with controllers and handling forwarding rules. Moreover, most of previous works have been focusing on increasing scalability, while our solution considers not only this aspect, but also fault tolerance on the control plane, security and the enabling of new services.

Fault tolerance itself is a topic that still lacks investigation in the context of the control plane. Issues on this topic have been addressed recently, however, the focus has been on the data plane. In [20] a new programing language was proposed to provide constructs for writing programs in terms of paths through the network and explicit fault-tolerance requirements, guaranteeing that traffic flows along the paths dictated by the program. Authors in [9] argue that centralizing fault tolerance management at the data plane in the controller would lead to unacceptable performance. Their approach, similar to MPLS global path protection, was implemented as a modification in the Openflow protocol to allow the partitioning of fault management functionality between controller and switches.

6. CONCLUSION

We proposed Decentralize-SDN, a general framework that can be used to enable a wide range of current- as well as future network services and applications. D-SDN supports control distribution by defining a hierarchy of controllers. Main controllers can delegate functions to secondary controllers, which can use D-SDN for fault tolerance. In addition, security services such as confidentiality and authentication are integrated with the proposed framework. We evaluated the D-SDN framework using two use cases: network capacity sharing and public safety networks. For future work we envisage new network services and applications integrated with D-SDN, such as inter-domain routing and load balancing.

7. AVAILABILITY

The proposed framework is available for download from

`http://inrg.cse.ucsc.edu/community`

8. REFERENCES

[1] Boneh, D., and Franklin, M. Identity-based encryption from the weil pairing. In *CRYPTO 2001*, J. Kilian, Ed., vol. 2139 of *LNCS*. Springer Berlin Heidelberg, 2001, pp. 213–229.

[2] Curtis, A. R., Mogul, J. C., Tourrilhes, J., Yalagandula, P., Sharma, P., and Banerjee, S. Devoflow: scaling flow management for high-performance networks. *SIGCOMM Comput. Commun. Rev. 41*, 4 (Aug. 2011), 254–265.

[3] Dixit, A., Hao, F., Mukherjee, S., Lakshman, T., and Kompella, R. Towards an elastic distributed sdn controller. In *Proceedings of the second ACM SIGCOMM workshop on Hot topics in software*

defined networking (New York, NY, USA, 2013), HotSDN '13, ACM, pp. 7–12.

[4] EADS Defence and Security Systems. CHORIST final report SP0.R7, 2009. Available online at `http://www.chorist.eu/`.

[5] Hassas Yeganeh, S., and Ganjali, Y. Kandoo: A framework for efficient and scalable offloading of control applications. In *Proceedings of the First Workshop on Hot Topics in Software Defined Networks* (New York, NY, USA, 2012), HotSDN '12, ACM, pp. 19–24.

[6] Heller, B., Sherwood, R., and McKeown, N. The controller placement problem. In *Proceedings of the first workshop on Hot topics in software defined networks* (New York, NY, USA, 2012), HotSDN '12, ACM, pp. 7–12.

[7] Iapichino, Giuliana, Camara, D., Bonnet, C., and Filali, F. *Public Safety Networks*. IGI Global, 2011.

[8] IETF RFC5246. The transport layer security (tls) protocol version 1.2, 2008.

[9] Kempf, J., Bellagamba, E., Kern, A., Jocha, D., Takacs, A., and Skoldstrom, P. Scalable fault management for openflow. In *Communications (ICC), 2012 IEEE International Conference on* (2012), pp. 6606–6610.

[10] Kevin Phemius, Mathieu Bouet, J. L. DISCO: Distributed multi-domain sdn controllers. *CoRR abs/1308.6138* (2013).

[11] Koponen, T., Casado, M., Gude, N., Stribling, J., Poutievski, L., Zhu, M., Ramanathan, R., Iwata, Y., Inoue, H., Hama, T., et al. Onix: A distributed control platform for large-scale production networks. In *OSDI* (2010), vol. 10, pp. 1–6.

[12] Lamport, L. Paxos made simple. *ACM Sigact News 32*, 4 (2001), 18–25.

[13] Lantz, B., Heller, B., and McKeown, N. A network in a laptop: Rapid prototyping for software-defined networks. In *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks* (New York, NY, USA, 2010), Hotnets-IX, ACM, pp. 19:1–19:6.

[14] Levin, D., Wundsam, A., Heller, B., Handigol, N., and Feldmann, A. Logically centralized?: state distribution trade-offs in software defined networks. In *Proceedings of the first workshop on Hot topics in software defined networks* (New York, NY, USA, 2012), HotSDN '12, ACM, pp. 1–6.

[15] McKeown, N., Anderson, T., Balakrishnan, H., Parulkar, G., Peterson, L., Rexford, J., Shenker, S., and Turner, J. Openflow: enabling innovation in campus networks. *SIGCOMM Comput. Commun. Rev. 38*, 2 (Mar. 2008), 69–74.

[16] NordstrÃűm, E., Gunningberg, P., and Tschudin, C. Robust and flexible internet connectivity for mobile ad hoc networks. *Ad Hoc Networks 9*, 1 (2011), 1 – 15.

[17] Open Networking Foundation. Openflow switch specification (verson 1.3), 2012.

[18] Pereira, G. C., Santos, M. A., de Oliveira, B. T., Jr., M. A. S., Barreto, P. S., Margi, C. B., and Ruggiero, W. V. Smscrypto: A lightweight cryptographic framework for secure SMS transmission. *Journal of Systems and Software 86*, 3 (2013), 698 – 706.

[19] Phemius, K., and Bouet, M. Openflow: Why latency does matter. In *Integrated Network Management (IM 2013), 2013 IFIP/IEEE International Symposium on* (2013), pp. 680–683.

[20] Reitblatt, M., Canini, M., Guha, A., and Foster, N. Fattire: declarative fault tolerance for software-defined networks. In *Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking* (New York, NY, USA, 2013), HotSDN '13, ACM, pp. 109–114.

[21] Sakai, R., Ohgishi, K., and Kasahara, M. Cryptosystems based on pairing. In *Symposium on Cryptography and Information Security (SCISï£¡00)* (2000), pp. 26–28.

[22] Santos, M., and Margi, C. A secure multi-party protocol for sharing valuable information in public safety networks. In *IEEE International Conference on Mobile Adhoc and Sensor Systems Workshops* (2011), pp. 935–940.

[23] Santos, M. A. S., De Oliveira, B. T., Margi, C. B., Astuto, B. N., Turletti, T., Obraczka, K., et al. Software-defined networking based capacity sharing in hybrid networks. In *IEEE International Conference on Network Protocols Workshops* (2013).

[24] Shamir, A. Identity-based cryptosystems and signature schemes. In *CRYPTO*, G. Blakley and D. Chaum, Eds., vol. 196 of *LNCS*. Springer Berlin Heidelberg, 1985, pp. 47–53.

[25] Sherwood, R., Chan, M., Covington, A., Gibb, G., Flajslik, M., Handigol, N., Huang, T.-Y., Kazemian, P., Kobayashi, M., Naous, J., Seetharaman, S., Underhill, D., Yabe, T., Yap, K.-K., Yiakoumis, Y., Zeng, H., Appenzeller, G., Johari, R., McKeown, N., and Parulkar, G. Carving research slices out of your production networks with openflow. *SIGCOMM Comput. Commun. Rev. 40*, 1 (Jan. 2010), 129–130.

[26] Tootoonchian, A., and Ganjali, Y. Hyperflow: a distributed control plane for openflow. In *Proceedings of the 2010 internet network management conference on Research on enterprise networking* (Berkeley, CA, USA, 2010), INM/WREN'10, USENIX Association, pp. 3–3.

[27] Yeganeh, S., Tootoonchian, A., and Ganjali, Y. On scalability of software-defined networking. *Communications Magazine, IEEE 51*, 2 (2013), 136–141.

[28] Yu, M., Rexford, J., Freedman, M. J., and Wang, J. Scalable flow-based networking with difane. *SIGCOMM Comput. Commun. Rev. 41*, 4 (Aug. 2010), –.