Towards a Richer Set of Services in Software-Defined Networks

Roberto Bifulco
NEC Laboratories Europe
roberto.bifulco@neclab.eu

Ghassan O. Karame
NEC Laboratories Europe
ghassan.karame@neclab.eu

***Abstract*—Software-Defined Networking (SDN) has drawn increasing attention from both industry and academia, owing to its premise to simplify the management and control over large networks. While the SDN technology was initially deployed within datacenters, there are currently early deployments of SDN in Wide-Area Networks; SDN is further envisioned to be deployed in the near future within fixed and mobile networks.**

In this paper, we show that the envisioned deployment of SDN within operator networks opens the doors to novel network security services that the operators can efficiently offer. More specifically, we propose in this work two exemplary operator services; our first solution enables the construction of secure location proofs for registered network users. Our second solution offers users the possibility to request the setup of network paths that are tailored to their specific security constraints, e.g., that only cross domains regulated by appropriate legislations which match their security policies. We show that this can be securely and efficiently attained by leveraging basic functionality from the OpenFlow protocol. In this respect, we evaluate the feasibility of our proposals by means of implementation within a realistic testbed composed of hardware OpenFlow-enabled switches; our findings suggest that our proposals can be deployed with minimal overhead within existing SDN-enabled networks.

I. Introduction

The emergence of Software-Defined Networking (SDN) has lead researchers and practitioners into the design of a number of innovative network functions and applications in an easier and flexible way. This is due to the main premise behind SDN, which lies in the separation of the "control plane" and the "data plane", thus greatly facilitating network management.

A successful instantiation of the SDN principles is OpenFlow [1], which defines a communication protocol and a network switch interface programming model specification. OpenFlow enables a *controller* to perform flow-based network forwarding via a number of *OpenFlow-operated switches*. Hence, the controller installs rules at the switches using the control interface; the switches then implement the requested rules on the data plane. OpenFlow is already experiencing real-world deployments within campus networks and datacenters [2]. The premises of SDN have also lead to early deployments in Wide-Area Networks [3]; the deployment of OpenFlow-enabled technology is envisioned in the near future within fixed and mobile network operators [4]–[7].

Besides the ease of management, we argue that OpenFlow simplifies the integration of security functions and applications within the network. For instance, recent work has demonstrated that OpenFlow can help deploying security policies in the network [8], performing dynamic load-balancing [9], [10], implementing network intrusion detection/prevention [11], performing secure end-to-end network measurements [12], etc.

In this work, we show that SDN (and OpenFlow) empowers network operators to efficiently offer a richer set of security services tailored to satisfy the performance and security requirements of their individual network users. Here, we consider a network comprising of the federation of one or more OpenFlow domains and we propose two solutions leveraging the OpenFlow protocol that aim at offering users the means to request additional services from the network. Our first solution, dubbed NPoL, enables operators to securely issue network location proofs for their users; these proofs can be submitted to location-based services such as YouTube, e-voting, e-banking services, etc. Our second solution, dubbed UdP, builds upon NPoL and offers users the possibility to request the setup of user-defined paths that are tailored to their specific Quality-of-Service (QoS) constraints. For instance, communicating entities might want to ensure that their traffic only crosses network domains that are regulated by appropriate legislations which matches their security policies—without compromising the desired QoS requirements (e.g., bandwidth, delay, jitter). UdP could also provide assurance for companies and governments that their Internet traffic cannot be hijacked by untrusted network operators for surveillance purposes [13], [14]. Besides proposing UdP, we present an efficient instantiation of our scheme within a realistic testbed composed of OpenFlow-enabled switches.

The main benefits of our proposals are: *(i)* given the surge of Internet surveillance/hijacking events [13]–[18], our proposals enable users to benefit from additional security assurances from the network itself, *(ii)* our proposals are likely to boost the revenues of network operators since these operators can offer their customers *differentiated* services without incurring additional costs in terms of network infrastructure. Finally, *(iii)* we show that this vision can be efficiently attained within ex-

SENT '14, 23 February 2014, San Diego, CA, USA

http://dx.doi.org/10.14722/sent.2014.23006

isting OpenFlow-enabled networks and conveniently scales to accommodate potential requests originating from the multitude of available network users. Note that a thorough analysis of the economic viability of our scheme is out of the scope of the paper; instead we focus in this work on demonstrating the feasibility of securely deploying such services within existing OpenFlow networks.

Our contributions can be summarized as follows:

- We introduce NPoL, a novel OpenFlow application which can be used by registered users to acquire secure location proofs from the network operator. We analyze the security of our solution, and we describe an instantiation of NPoL within existing SDN-enabled networks.
- We show that OpenFlow also enables the provisioning of user-defined constrained paths. We present a solution, dubbed UdP, that can be efficiently deployed within existing OpenFlow networks and we analyze its performance and security provisions.
- We implement and evaluate a prototype instantiation of UdP within the Beacon OpenFlow controller [19] and using a realistic testbed comprising of NEC OpenFlow-enabled switches.

The remainder of the paper is organized as follows. In Section II, we overview the OpenFlow protocol and we present our model. In Section III, we introduce and analyze our solution, NPoL, that leverages OpenFlow in order to construct secure location proofs for network users. In Section IV, we introduce and implement our second solution, UdP, which enables the efficient provisioning of user-defined paths in the network. In Section V, we overview related work in the area and we conclude the paper in Section VI.

II. Background & Model

In what follows, we briefly overview the main features of OpenFlow and we state our system and attacker model.

A. OpenFlow

OpenFlow [20] provides an architecture that separates the data plane of the network, i.e., the switching hardware, from its control plane which implements the forwarding decision logic. More specifically, the control plane usually comprises a logically centralized entity—the controller—which is typically implemented in software (using a high level programming language). The controller uses the interface and the programming model defined by the OpenFlow specification in order to configure/program the switches in the network. The communication between the controller and switches is established using a control channel, which is usually implemented as an out-of-band dedicated TCP/IP network. Here, TLS can be used to secure the communication between the controller and the switches.

The function of a controller is to provide a set of rules to be installed at the different switches by means of the OpenFlow protocol. In the OpenFlow terminology, the rules used to program a switch are called *flow table entries* (FTE). An FTE is defined by *(i)* the *match set* that defines the network flows to which the entry is applied, *(ii)* the *action set* which defines the processing and the forwarding decision that must be applied to the matched flows, *(iii)* a *priority* field of the entries installed in a switch, and *(iv)* an *expiration time* specified as a timeout. For instance, a typical FTE provides semantics like "forward network flows with network destination address 1.1.1.1 to port 3". Notice that the rules space is a limited resource in an OpenFlow switch; hence, the control logic design must take into account the total number of rules that are required to be installed at switches in order for any SDN solution to be deployed in practice.

Moreover, we point out that an OpenFlow switch can be instructed to forward packets to a "controller port", thus allowing the controller to inspect packet headers. In this case, the switch sends a *packet_in* message to the controller, which contains the original packet's headers and additional information—such as the switch and the port IDs onto which the packet was received.

B. System and Attacker Model

Throughout this paper, we assume the following network model. We assume a network comprising of one or more OpenFlow-supported domains. In the sequel, we denote by $\mathcal{D}_i$ the i-th domain, and by $\mathcal{C}_i$, the (logically-centralized) controller that governs domain $\mathcal{D}_i$. We assume that each controller $\mathcal{C}_i$ is equipped with a public/private key pair and we assume that the various controllers know each others' public keys.

We also assume that each network user $u_j \in \mathcal{D}_i$ is equipped with a public and private key pair, denoted by (pk_j, sk_j) respectively. Our analysis covers the typical case where users are connected to the network using a Customer Premises Equipment (CPE, also commonly referred to as "home gateway"); here, we assume that the CPE is under the governance of $\mathcal{C}_i$. Figure 1 summarizes our network model.

In our analysis, we assume a secure channel between the user machines and the network routers/switches on the one hand and between switches and the corresponding controllers on the other hand.

We consider one or more colluding malicious network users who are motivated to increase their benefit in the network, e.g., by claiming a different network location, etc. We assume that these users have knowledge of the measures used by the controllers/switches in order to deter misbehavior in the network. We, however, assume that these users are computationally bounded, and as such cannot forge signatures, break encryption schemes, etc. Finally, we assume that the controllers and network components are trusted and cannot be compromised.

III. NPoL: Location Proofs using OpenFlow

In this section, we introduce NPoL, a novel OpenFlow application which can be used by network users to automatically acquire location proofs from the network operator.

A. Location Proofs

Location-based services (e.g., Foursquare [21] and Yelp [22]) are gaining increasing importance recently. Several applications enable users (e.g., using mobile devices) to

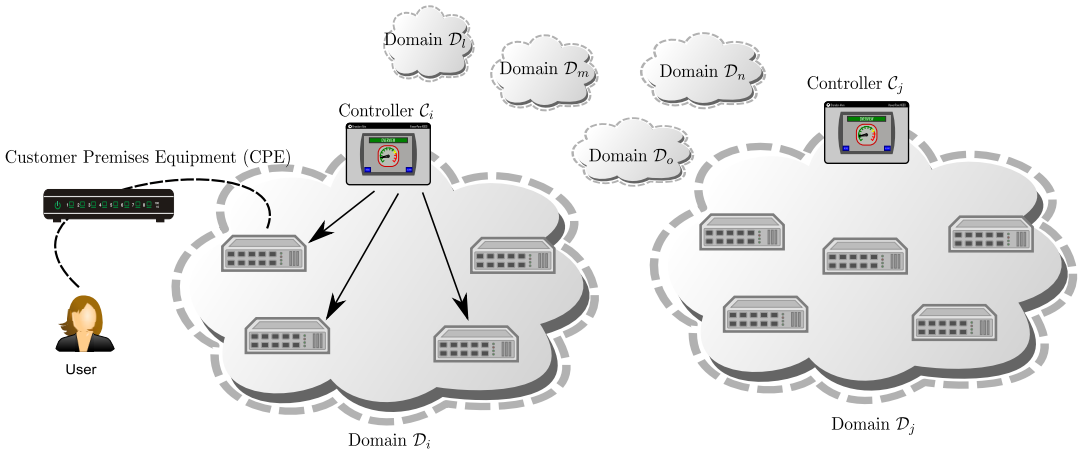

Fig. 1. System model. We assume a network comprising of one or more OpenFlow-supported domains. Each domain $\mathcal{D}_i$ is governed by a (logically centralized) controller $\mathcal{C}_i$. Users are connected to the network by means of CPEs.

discover and communicate their locations to a server in the cloud; in turn, the server uses this information to return data relevant to the users' locations. For instance, a number of existing services can only be acquired by users who are located within a specific geographical area; this includes banking services, Youtube and content delivery services, among many others. Location information also proves to be useful for a number of security-critical services such as police investigations, e-voting, etc. However, while many devices nowadays are capable of discovering and reporting their locations, they lack a mechanism to prove their locations to third-party applications and services. Indeed, as the utility of relying on location information increases, so do the incentives for users to misreport their locations. Recent studies [23]–[25] suggest that current mechanisms such as IP-based Geolocation, and network coordinate systems cannot be used, solely, to securely estimate the location of devices since these techniques are vulnerable to a wide range of location manipulation attacks.

"Location proofs" consist of a certificate that certifies the presence of a given entity at a certain location at some point in time. The literature comprises a number of proposals for location proof architectures (e.g., [26]–[29]). Most of these solutions require additional functionality from network components (e.g., such as performing distance bounding measurements, generating signatures) in order to issue such proofs. Other solutions (e.g., [24]) require the reliance on trusted landmark nodes within the network which can securely determine the location of users. The unavailability of such functionality within existing networks has made location proofs absent from the current landscape of services offered by network operators.

In what follows, we introduce our solution, NPoL, which leverages OpenFlow and enables operators to issue location proofs without incurring modifications to the network.

B. NPoL: Network Proofs of Location

NPoL leverages OpenFlow to *(i)* acquire the network location of users and to *(ii)* enforce anti-spoofing of IPs in the network. By doing so, NPoL can ensure that a given IP is present at a given location.

More specifically, in OpenFlow, the controller typically configures the switches on the edge of the network, which are connected directly to the CPEs to generate a *packet_in* message upon reception of a network packet—when such a packet is not matched by any forwarding rules installed within the switch. The *packet_in* message contains the switch ID and the port ID from where the packet has been received; it also includes the packets' headers, so that the controller can inspect those headers and extract relevant information.

Using this information, the controller builds a dynamic location table in NPoL which contains the locations of current IP addresses. Here, "location" connotes the presence of the user's device at a given switch's port. This network location can be easily paired with a geographical location, e.g., retrieved from the network operatorâĂŹs inventory database[1]. Note that in case the IP address of the user changes, then the controller can immediately detect the change and issue new proofs of location for the newly obtained IP address. Moreover, our solution also ensures that an IP address is correctly assigned to its current location in case of address rewriting, e.g., for load balancing purposes. For instance, consider the case where the controller needs to issue a location proof for a node whose address has been rewritten by a network switch. Since we assume that the switch/CPE is under the governance of the controller, the controller is also aware of the translation table, and of the mapping to the corresponding translated location.

Moreover, to prevent IP spoofing in NPoL, the controller installs a new forwarding rule on the outermost switch from which it received the *packet_in* message to ensure that only packets with the specified source IP addresses are forwarded from that port. In the OpenFlow protocol, this rule looks like:

Match set: All wildcards but (NW_SRC that has value "IP_j")
Action set: OUTPUT (port: a)

In this example, packets from IP_j are forwarded onto port a. Using this approach, the controller ensures that IP addresses

[1]This geo-location information is not likely to go stale since it is required in order to e.g., physically maintain the switches.

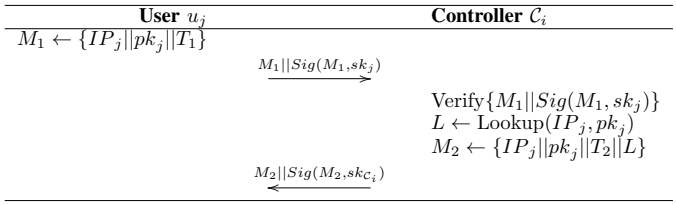

Fig. 2. Sketch of the messages exchanged between users and the controller in NPoL.

in domain $\mathcal{D}_i$ cannot be spoofed[2]. Notice that this property—which is straightforward to guarantee in OpenFlow—is relatively challenging to enforce otherwise. Indeed, network operators usually only enforce anti-spoofing rules at the data link layer of the network stack[3].

In NPoL, a user asks for his location proof by using an interface exposed by the controller, e.g., using a REST-based interface [31]. The protocol interactions in NPoL are depicted in Figure 2. Here, the user creates (and signs) a request location proof message M_1 which contains the user's public key (pk_j), the current IP address (IP_j) and a timestamp (T_1). After receiving M_1, the controller checks *(i)* that the signature is valid and that it matches pk_j and *(ii)* that the IP address for which the location proof is required is correct, by comparing it against the source IP address contained in the message's packet header. If these verifications pass, the controller performs a *lookup* for the user's device location in the dynamic location table using the IP address as search key in order to find the current geographical location of such device. Finally, the location is used to build the signed location proof which is delivered to the requesting user in M_2.

Security Analysis: NPoL provides a location proof that a given IP address, associated with the public key of the user is present at location L, at timestamp T_2. By binding both the IP address and the public key of the user to a specific location, NPoL ties the IP address of users to their identity (i.e., public keys) and inherently alleviates Sybil attacks [32] where users create several fake identities in several locations within the network[4]. Note that any third-party service which acquires the location proof M_2 can tunnel all communication with u_j using a secure channel established using its public key[5] pk_j. Recall that the controller prevents spoofing in its domain by installing the forwarding rules at the switches (see above).

We point out, however, that similar to existing solutions, NPoL cannot prevent attacks where a malicious user compromises several devices in the network and replicates her credentials across these devices [33]. Instead, NPoL ensures that the user has at least one device equipped with IP_j and pk_j at location L at time T_2. Clearly, if a malicious user uses the same credentials across several network locations within a small period of time, then the controllers can immediately detect such misbehavior by periodically exchanging location information of their users[6].

We also note that NPoL does not come at odds with user privacy. Indeed, a location proof request can only be issued by the user himself, since the request needs to be authenticated using the private key of the user (which should match its registered public key). We further note that in existing networks the location privacy of users is exposed to network operators who can keep track, at all times, of the network locations of their users.

Remark 1: We point out that NPoL can be easily extended to include the attestation of the software installed at the controller $\mathcal{C}_i$. Recall that attestation protocols serve to prove to any third party that $\mathcal{C}_i$ is executing genuine software. In this case, the user can include a random nonce within M_1, which can be used as an attestation challenge. $\mathcal{C}_i$ can then piggyback the attestation response within M_2. Further details about attestation protocols can be found in [34], [35].

IV. On-Demand Provisioning of User-Defined Paths

In this section, we show that OpenFlow also efficiently enables the on-demand provisioning of user-defined paths. We first start by introducing the concept of user-defined paths.

A. User-Defined Paths

The growth and complexity of conventional networks prevents network operators from offering fine-grained subscriptions to their customers; instead these networks can only, at best, offer their clients a handful of possible choices for their network subscriptions (e.g., in terms of connection bandwidth). SDN suggests a slight departure from this model, since it allows the operators to dynamically configure network switches and therefore to install fine-grained rules within the network, specifically tailored to the needs and requirements of individual network customers.

In this respect, we envision the deployment of a network service that enables users to request from the network operator the provisioning of a dedicated communication path between two IP addresses; the requested path can be constrained to some user requirements. For instance, one possible use-case would be to provision a user-defined path which only crosses network domains whose regulation abides by the users' security policies. Indeed, while there are clear legislations which protect net-neutrality and the privacy of online users, some countries clearly lack the existence of any legislation that prevents operators and entities from undergoing Internet surveillance [36]–[38]. Even worse, recent events reveal that some operators are purposefully hijacking Internet traffic [13], [14]. For instance, a hijack of major Internet domains in 2010 [13] resulted in the routing of a large proportion of Internet traffic to a single network operator. Note that such threats are typically hard to deter; while end-to-end encryption

[2]Note that this can be achieved in spite of user mobility. When users move, they will acquire new IPs. The controllers then install anti-spoofing rules for the newly obtained IPs.

[3]In Ethernet based Broadband Access networks [30], a common procedure to ensure that the source MAC address is correct is to install static anti-spoofing rules directly in access nodes, such as DSLAMs, as a one time action executed after the service provisioning contract subscription.

[4]Here, we assume that each network user has registered its public key within the controller at an earlier setup phase.

[5]As a by-product, NPoL can be used to securely distribute the public key of u_j to third-party services that know the public keys of the controllers.

[6]Here, the same user would appear to have moved at a very large speed across various network locations.

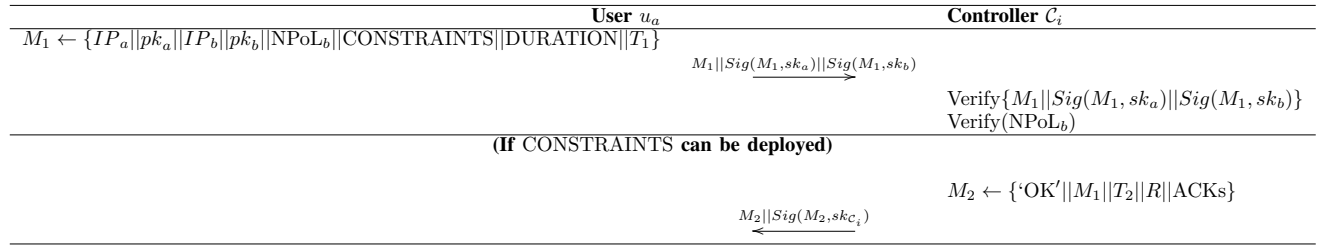

Fig. 3. Sketch of the messages exchanged in UdP. Here, we assume a user-defined path with requirements CONSTRAINTS. Recall that all communication between the users and the controller is performed over a secure channel.

can hide the contents of the packets, recent results show that end-to-end encryption does not necessarily hinder surveillance [15]–[18]. Furthermore, anonymizing systems, such as Tor [39], typically come at high performance costs, e.g., for VoIP communication.

Ideally, companies and governments would want to make sure that—whether purposefully or not—their Internet traffic does not cross untrusted network operators (e.g., in authoritarian countries). In the best case, these entities want to make sure that their traffic only crosses network domains that are regulated by appropriate legislations which matches their security policies, without compromising the desired QoS requirements (e.g., bandwidth, delay, jitter)—hence the motivation for the following Example 1.

Example 1: A company requests a given QoS level for all traffic crossing between two of its IP addresses (e.g., corresponding to two or more geo-located datacenters) while ensuring that exchanged flows between those IP addresses never cross domain A.

Challenges: A number of challenges need to be addressed to enable Example 1 in practice.

a) Scalability: If network operators would allow such level of fine-grained rule creation across the entire (federated) network, then this means that the number of flow table entries maintained per network router/switch will increase linearly with the number of user-defined paths. Here, we point out that existing routers can maintain, at best, flow entry tables with few thousands of entries. For instance, HP switches are reported to hold 1,500 flow entries [40], [41], while NEC reports that its switches can store more than 64,000 flows [42].

b) Security: Such a service implicitly requires that the user-defined path is solely used by legitimate clients (e.g., who purchased the service). As such, the network operator needs to prevent a malicious user from benefiting from another user's subscription. Needless to mention, this entire process should not incur considerable overhead on the various controllers.

B. UdP: Efficient Provisioning of User-Defined Paths

We start by presenting our solution, UdP, which builds upon NPoL in order to enable the efficient and secure provisioning of user-defined paths.

In UdP, users install a daemon application which serves to provide a user-interface to request a user-defined path. A sketch of the interactions required for setting up a user-defined path between IP_a and IP_b unfolds in Figure 3. Similar to NPoL, UdP binds the IP addresses of u_a and u_b to their public keys pk_a and pk_b, respectively. By doing so, UdP enables u_a and u_b to use the same path even if their IP addresses have changed (e.g., if they have relocated). In UdP, users can request the setup of paths that satisfy requirements 'CONSTRAINTS' by sending message $M_1 \leftarrow \{IP_a||pk_a||IP_b||pk_b||\text{NPoL}_b||\text{CONSTRAINTS}||\text{DURATION}||T_1\}$. Here, CONSTRAINTS could refer to QoS requirements (e.g., bandwidth), but could also specify the set of allowed domains $\mathcal{S}$ that the path can traverse. For example, u_a could require that $\mathcal{S} \in \{\mathcal{D} - \mathcal{D}_i\}$ (i.e., the path does not traverse domain $\mathcal{D}_i$}). Moreover, NPoL_b denotes a location proof binding IP_b and pk_b. Note that since the path is bi-directional, such a request needs to be authorized by both u_a and u_b. Here, we assume that u_a contacts u_b (using their respective daemons) in order to acquire NPoL_b and a signature over M_1. The UdP daemon of u_a then sends M_1 (along with u_a and u_b's signatures on M_1) to $\mathcal{C}_i$.

Note that in the (typical) case where IP_b is located outside $\mathcal{D}_i$, then $\mathcal{C}_i$ uses the location information contained in NPoL_b in order to contact the controller of the neighboring domain $\mathcal{D}_j$ on the path to IP_b and checks for the feasibility of installing a network path that satisfies CONSTRAINTS. This process recursively re-iterates until $\mathcal{C}_i$ acquire signed acknowledgement messages from all controllers on the path between IP_a and IP_b. For transparency, $\mathcal{C}_i$ includes these signed acknowledgements in the ACKs field in the receipt M_2 (cf. Figure 3).

If CONSTRAINTS could be deployed within the network, then $\mathcal{C}_i$ sends a receipt $M_2 \leftarrow \{\text{'OK'}||M_1||T_2||R||\text{ACKs}\}$ to u_a, u_b, and to the corresponding controllers involved in the provisioning of the path (here, R is a random nonce); these latter, along with $\mathcal{C}_i$ then install the appropriate rules on the switches for the required duration DURATION. Notice here that the rules installed at the first switch on the path are based on exact matching of the IP addresses of u_a and u_b and the matching of a packet header field with the cryptographic hash of the receipt M_2, denoted in the sequel by $h(M_2)$. In OpenFlow, this can be specified as follows:

Match set: (Not specified fields are wildcard) $\text{DL\_SRC} = \text{h}(\text{M}_2)$, $\text{NW\_SRC} = \text{IP}_\text{a}$, $\text{NW\_DST} = \text{IP}_\text{b}$
Hard Timeout: DURATION
Action set: OUTPUT (port: a)

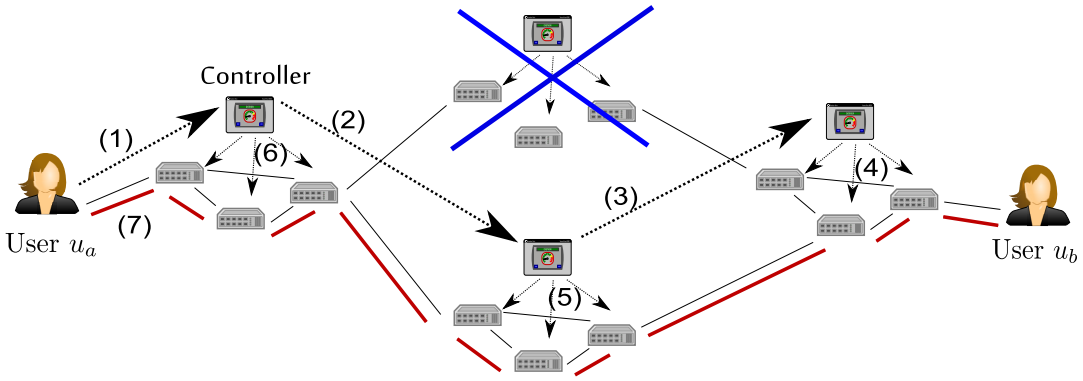

Fig. 4. Example of setting up a user-defined path in UdP. Here, the user-defined path between u_a and u_b should not traverse the crossed domain. The controller of the domain hosting u_a contacts the appropriate controllers and checks the feasibility of setting up a user-defined path according to the user requirements. If such a path can be established, the controllers install the corresponding flow rules in their domains. The resulting user-defined path is shown in solid red line. The dotted arrows refer to communication on the control plane.

Once the user-constrained path is provided, the user's UdP daemon rewrites the packets' source MAC address with the value of $h(M_2)$. Upon reception of the packets, the network switches check the presence of $h(M_2)$ in order to forward the packets accordingly to the required user-defined path. The process of setting up a user-defined path in UdP is summarized in Figure 4. By including $h(M_2)$ in the flow matching criteria, UdP does not require the controller to keep any intermediate state information in order to verify the authenticity of established subscriptions in the system.

Security Analysis: By tainting the flows with the hash of M_2 as a flow matching criteria, UdP associates each flow rule with a given subscription. This, in turn, enables the controller to verify, without the need to store metadata information about any subscription, whether a given user (i.e., public key) is entitled to make use of the user-defined path. Note that, after installation of the rules, only users with IP_a and IP_b can taint their flows using $h(M_2)$ to make use of the user-defined path. We point out, here, that spoofing attacks of IP_a are prevented by $\mathcal{C}_i$ (cf. Section III-B). Furthermore, the location proof NPoL_b provides a guarantee that the controller governing IP_b ensures that IP_b cannot be spoofed in its domain. Note that, alternatively, this guarantee could be explicitly requested by $\mathcal{C}_i$ upon path establishment. However, by requiring that NPoL_b is included within M_1, $\mathcal{C}_i$ offloads the work to the users, and ensures that they have committed enough of their resources before it proceeds to establish the path. Such an approach thwarts DoS attacks on the controller.

If, on the other hand, IP_a and/or IP_b refer to the addresses of CPEs or of NATs, then all users located behind the NATs can benefit from the installed user-defined path (e.g., in case of a company or a grouping of users). Note that in case u_a changes his IP (e.g., u_a moves to another network), then there is no forwarding rule for the tainted packet that arrives at the first operator's switch. The latter then encapsulates u_a's packets in a *packet_in* message which is sent to the controller. The controller subsequently requests the receipt of the subscription M_2; this message could be sent by leveraging existing functionality from OpenFlow, such as the ability to inject arbitrary packets at a switch's port using a *packet_out* message. After receiving this message, the UdP daemon replies back with the certificate M_2. After verifying the signature of M_2, the controller installs the appropriate rules on the switches.

In addition to minimizing storage within controllers, this approach throttles Denial-of-Service attacks on the controller. Here, malicious users have to present authentic receipts to the controller which match their flow taints in order to solicit the controller to install any rule. Notice here that the random number R within M_2 prevents tainting collisions for similar subscriptions, and ensures that the taint is hard to predict; recall that an external attacker cannot eavesdrop on the taints of existing flows since we assume a secure channel between the user and the switches. We note that even if an attacker can acquire a correct taint, she cannot make use of the user-defined path provisioned for u_a and u_b since she needs to spoof IP_a or IP_b in domains which strictly enforce anti-spoofing.

Rule Scalability: UdP requires a potentially high number of forwarding rules that need to be installed, in particular at the network switches located in the core of the network. This clearly does not scale well as the number of users which benefit from this service increases. To overcome this limitation, UdP relies on an innovative OpenFlow-based solution adapted from [43]. The main intuition here is that the controllers only install forwarding rules at the outermost (and least loaded) switches of the network (i.e., those connected to the CPEs and users), while no rules are installed at the overloaded core routers/switches.

Packet forwarding for UdP is then achieved as follows. After acquiring the path information from the controller, the outermost switch rewrites the packet's headers to encode *(i)* the list of actions that each switch on the path has to execute on the received packet, and *(ii)* a counter that encodes the current action that each switch needs to apply upon reception of the packet. That is, each switch, when receiving this modified packet, checks the counter field to find the action that has to be executed, increments the counter and executes the required action. This means that core switches no longer need to store a flow table entry per path, but simply have to store the required actions only; these typically require at most few tens of entries to be stored per core switch [43]. For instance, a switch with 24 ports could have 24 possible forwarding actions (one per each port), hence, a small number of possible actions are required

Fig. 5. The testbed used in our prototype evaluation. Our testbed is composed by two servers HP DL380G7 equipped with two Intel L5640 (6-Core 2.26GHz) processors, 24GB of RDIMM PC3-10600R-9 memory and by three NEC ProgrammableFlow PF5240 OpenFlow-enabled switches.

at any given hop. This also makes it possible to encode the sequence of the actions in current packet's header space. Notice that the last switch on the path restores the packet's original header.

We point out that this solution can be easily ported to the OpenFlow switches' firmware with small modifications. Further details on this solution can be found in [43].

C. Implementation Setup

We implemented UdP atop of the Beacon OpenFlow Controller [19]. To support controller scalability, UdP relies on a hierarchical controller architecture similar to [44]. Here, the controller consists of several layers, where the lower layer is solely interfacing with the switches. Each layer interacts with its upper layer when a network event (such as a *packet_in*) cannot be handled with the knowledge of the local layer. This approach distributes network events opportunistically, based on the assumption that a large fraction of these events can be locally handled; indeed this is an expected behavior in Wide-Area Networks (WAN) where interactions within local networks do not require the knowledge from other networks to be handled. Further details on this approach can be found in [44], [45].

The controller exposes a REST interface to receive user-defined path requests. The requests follow the protocol described in Figure 3, with the CONSTRAINTS encoded either using JSON or XML, to include requirements such as path end-points' IP addresses, the required QoS, the set of domains where the path should (or not) traverse, etc. Conforming with the specification of UdP, when a request is received, the controller computes the optimal path according to the current network topology and load, creates the certificate M_2 and stores the computed solution in memory, using a hash table that uses $h(M_2)$ as an index field.

The installation of user-defined paths can follow two strategies: *early* path installation or *lazy* path installation. In the former approach, the controller installs the path rules once the user's request is received, while in the latter approach, the controller defers the installation of the forwarding rules until the first tainted packet is received. Clearly, this approach enables the dynamic handling of new path installations and path updates upon end-point relocation.

Our implementation for UdP's daemon[7] exposes a GUI for the user to request the user-defined paths and creates a

[7]Note that the same solution can be equally mounted at the CPEs, in which case, the user interface can be made accessible from the local network, e.g., a web interface could be provided

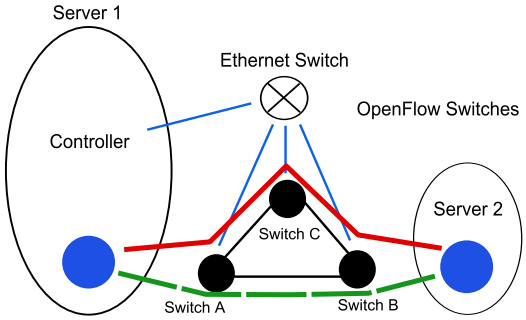

Fig. 6. Our evaluation setup. All OpenFlow switches are connected to each other on the data plane, and connect to the controller in a star topology via an Ethernet switch. We assume that user-defined path setup between "Server1" and "Server2".

software Ethernet bridge and a TAP virtual network interface. The bridge then acquires the original user's machine network setup in order to connect to the TAP interface and to the physical Ethernet interface of the user's machine. The TAP interface MAC address is set to the value of $h(M_2)$, so that any packet going out using this interface will be tainted by writing the value $h(M_2)$ in the source MAC address. Notice that this solution does not require any modification to the user's machine network stack. Finally, the UdP daemon adds an entry in the forwarding table of the machine's kernel in order to forward the packets belonging to the user-defined path using the TAP interface.

We conducted our experiments using the testbed shown in Figure 5. The testbed is composed by two HP DL380G7 servers equipped with two Intel L5640 (6-Core 2.26GHz) processors, 24GB of RDIMM PC3-10600R-9 memory and by three NEC ProgrammableFlow PF5240 OpenFlow-enabled switches [42]. Here, "Server1" has two network interfaces, one connected to a traditional Ethernet switch used for the OpenFlow control network, and the second one used to perform the tests using the OpenFlow switches data-path. In this way, "Server1" runs both the OpenFlow Controller and the UdP daemon.

D. Evaluation Results

First, we evaluate the performance of installing path forwarding rules for setting up a user-defined path between "Server1" and "Server2" (cf. Figure 6). Here, we relied on the *ping* tool to measure the delay experienced by network packets in our testbed. To accurately assess the performance of our scheme, there was no cross-traffic traversing the switches throughout the entire evaluation. Table I reports the various RTT propagation times witnessed by the first packet of a new network flow originating at "Server1" using the lazy and early rule installation strategies, respectively. Clearly, the early

Rule Installation strategy	First packet RTT (ms)
Early installation	0.159 (0.009664)
Lazy installation	18.270 (5.837111)

TABLE I. RTT PROPAGATION TIMES WITNESSED BY THE FIRST PACKET OF A NEW NETWORK FLOW ORIGINATING AT "SERVER1" USING THE LAZY AND EARLY RULE INSTALLATION STRATEGIES. EACH DATA POINT IS AVERAGED OVER 10 INDEPENDENT MEASUREMENTS. WE ALSO INCLUDE THE 95% CONFIDENCE INTERVALS IN BRACKETS.

installation causes the first packet to only perceive the usual network forwarding delay (which in our case was three hops), while the lazy installation introduces a considerable delay in the flow setup phase, since the packet has to wait until the packet is examined by the controller and the forwarding rules are installed at the various switches before being forwarded. In our evaluation, the lazy installation resulted in a total RTT of about 20ms. While this network delay can be acceptable in many scenarios, we acknowledge that it might result in a sensible performance degradation when dealing with real-time streaming. These results therefore motivate the need to extract the rule installation strategy from the user requirements. On one hand, lazy installation might result in a performance degradation in some use-cases, but on the other hand, this strategy is ideal for alleviating network resources usage, in particular during heavy workloads [8].

As explained before, both lazy and early installations only affect the outermost switches in UdP and are not required for inner core switches; in our testbed (cf. Figure 6), this translates to installing rules only at switches A and B. In this respect, we performed a preliminary evaluation of the number of required rules that need to be installed at switches A and B in order to support user-defined paths. For that purpose, we analyze a one day long traffic trace taken at a Digital Subscriber Line Access Multiplexer (DSLAM) of a European telecom operator in 2007, and we count the number of flows in the trace, using as flow definition the combination of source and destination IP addresses. In total, there were around 182,000 different combinations of addresses; only 10,000 of those were however active (i.e., where end-points are exchanging packets in a 10s time window) and as such should be installed at switches. Given that current OpenFlow switches can easily cope with this number of flows entries [42], this preliminary evaluation confirms that UdP can easily cope with current traffic dynamics.

Finally, we evaluate the performance of the system when subject to updates to a given user-defined path. Such updates could occur in case network failures occur or for optimization purposes, or to support user mobility. To evaluate UdP in this context, we dynamically replace the three-hop path (shown in solid red in Figure 6) with the two-hop path (not crossing switch C, shown in dotted green in Figure 6) and we measure the end-to-end packet delay during the path change. Figure 7 shows the resulting RTT between "Server1" and "Server2" when the aforementioned path update occurs[8]. Our results clearly show that no packets are lost and that the perceived RTT is not significantly affected when compared to the 'static' case featuring no path updates. Note that the variability in RTT witnessed immediately after the transmission of packet ID 25 is due to change in scheduling policies at end-hosts (to accommodate for the new shorter path).

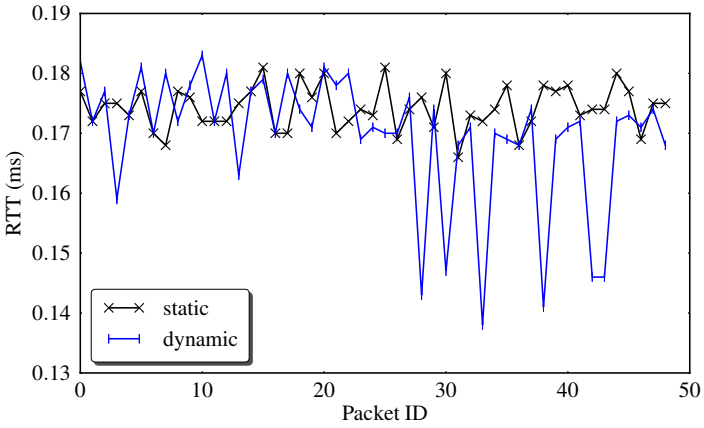

Fig. 7. Experienced per-packet delay when using UdP between "Server1" and "Server2" in the early and lazy installation strategies, respectively. The y-axis shows the experienced delay in milliseconds while the x-axis reports the packet sequence number. 'Static' refers to the case where no modification to the user-defined path occurs. 'Dynamic' refers to the scenario where the three-hop user-defined path in Figure 6 (in red) is replaced with the two-hop path (in red) before the transmission of packet ID 25.

V. Related Work

In this section, we overview related work in the area.

Location Proofs: The literature comprises a considerable number of proposals for location proof architectures. Most of these solutions require additional functionality from network components (e.g., such as generating signatures) in order to issue such proofs. Other solutions require the reliance on the presence of trusted landmark nodes within the network which can securely determine the location of users.

In [27], Luo *et al.* propose a privacy-friendly location proof architecture that relies on trust third party. In [28], [46], Capkun *et al.* rely on measuring the round trip time of signals to securely estimate the distance between two parties in wireless sensor networks. In [24], Kaafar *et al.* propose the reliance on trusted surveyor nodes in order to secure Internet coordinate systems.

In [47], Faria and Cheriton propose a location-based authentication architecture for WLANs, which relies on a trusted centralized entity that controls a group of access points and broadcasts random challenges to clients through these access points. In [48], Lenders *et al.* propose a geotagging service that allows a content creator to tag their content with a spatial timestamp. Other proposals (e.g., [49], [50]) rely on deploying dedicated hardware (e.g., Trusted Platform Modules) to ensure the correctness of the location reported by sensing devices.

OpenFlow/Next-Generation Internet Security: Security in OpenFlow has recently received considerable attention in the literature. In [9], Wang *et al.* propose algorithms for computing wildcard rules in order to adjust to changes in load-balancing policies without disrupting existing connections. In [10], Handigol and Seetharaman propose the reliance on customized flow routing in OpenFlow in order to dynamically perform load-balancing in the network.

In [11], Braga *el al.* propose a lightweight method for detecting DDoS attacks based on traffic flow features acquired using the OpenFlow-enabled NOX controller. In [12], [51], [52], Karame showed that the security of bottleneck bandwidth estimation and RTT latency measurements can be strengthened by acquiring functionality from OpenFlow-enabled networks.

[8]Notice that the path change happens in the proximity of the transmission of packet ID 25.

In [53], Shin *et al.* proposed a security application development framework designed to facilitate the design of modules for attack detection/mitigation. In [54], Porras *et al.* propose a software extension that enables the NOX OpenFlow controller to check flow rule contradictions in real time. In [55], Shin *et al.* propose an extension to the data plane in order to alleviate control plane saturation attacks aiming at disrupting network operations.

In [56], Andersen *et al.* propose AIP, a protocol which enables hosts and domains to prove that they have the address they claim to have without relying on any global trusted authority. In [57], Zhang *et al.* introduce a scheme which separates ASes into groups of independent routing sub-planes, which then interconnect to form complete routes. By doing so, this scheme promises route control, failure isolation, and explicit trust information for end-to-end communications.

VI. Conclusion

SDN has lead researchers and practitioners into thinking of innovative ways to integrate security applications within the network. In this paper, we showed that the inevitable deployment of SDN within operator networks opens the doors to novel security services that can be made available to users, without incurring additional costs in terms of network infrastructure.

In this respect, we illustrated and proposed two exemplary services that can be efficiently deployed within existing SDN networks. Our first solution, NPoL, enables the construction of secure location proofs in OpenFlow networks. Our second solution, UdP, leverages NPoL and enables users to request the setup of network paths that are tailored to their specific constraints, e.g., that only cross network domains regulated by appropriate legislation which matches their security policies. We evaluated the feasibility and practicality of deploying our solutions using a realistic testbed; our findings suggest that our proposals can be deployed with minimal overhead within existing OpenFlow-enabled networks.

Acknowledgment

The authors would like to thank Thomas Dietz for helpful feedback and for the support in setting our implementation testbed. The authors would also like to thank Francesco Gringoli and Maurizio Dusi for the help in analyzing DSLAM traffic traces.

References

[1] OpenFlow, Available from http://www.openflow.org/. [Online]. Available: http://www.openflow.org/

[2] NEC, "ONF Case study: Kanazawa University Hospital," https://www.opennetworking.org/images/stories/downloads/sdn-resources/customer-case-studies/cs-nec.pdf.

[3] S. Jain, A. Kumar, S. Mandal, J. Ong, L. Poutievski, A. Singh, S. Venkata, J. Wanderer, J. Zhou, M. Zhu, J. Zolla, U. Hölzle, S. Stuart, and A. Vahdat, "B4: Experience with a Globally-Deployed Software Defined WAN," in *Proceedings of ACM SIGCOMM*, ser. SIGCOMM '13. New York, NY, USA: ACM, 2013, pp. 3–14. [Online]. Available: http://doi.acm.org/10.1145/2486001.2486019

[4] T. Taleb and A. Ksentini, "Follow Me Cloud: Interworking Federated Clouds and Distributed Mobile Networks," *IEEE Network*, vol. 27, no. 5, pp. 12–19, 2013.

[5] "Software Defined Networking for Next Generation Internet Components," available from http://www.fp7-sparc.eu/goals/.

[6] A. Gudipati, D. Perry, L. E. Li, and S. Katti, "SoftRAN: Software Defined Radio Access Network," in *Proceedings of the ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking*, ser. HotSDN '13. New York, NY, USA: ACM, 2013, pp. 25–30. [Online]. Available: http://doi.acm.org/10.1145/2491185.2491207

[7] L. E. Li, Z. M. Mao, and J. Rexford, "Toward Software-Defined Cellular Networks," in *Proceedings of the European Workshop on Software Defined Networking*, ser. EWSDN '12. Washington, DC, USA: IEEE Computer Society, 2012, pp. 7–12. [Online]. Available: http://dx.doi.org/10.1109/EWSDN.2012.28

[8] Z. A. Qazi, C.-C. Tu, L. Chiang, R. Miao, V. Sekar, and M. Yu, "SIMPLE-fying middlebox policy enforcement using SDN," in *Proceedings of the ACM SIGCOMM 2013 conference on SIGCOMM*, ser. SIGCOMM '13. New York, NY, USA: ACM, 2013, pp. 27–38. [Online]. Available: http://doi.acm.org/10.1145/2486001.2486022

[9] R. Wan, D. Butnariu, and J. Rexford, "OpenFlow-Based Server Load Balancing Gone Wild Into the Wild : Core Ideas," in *Proceedings of USENIX Hot-ICE*, 2011.

[10] N. Handigol and S. Seetharaman, "Plug-n-Serve: Load-balancing Web Traffic using OpenFlow," in *Computer Communication Review (ACM SIGCOMM)*, 2009.

[11] R. Braga, E. Mota, and A. Passito, "Lightweight DDoS flooding attack detection using NOX/OpenFlow," in *Proceedings of the 2010 IEEE 35th Conference on Local Computer Networks*, ser. LCN '10. Washington, DC, USA: IEEE Computer Society, 2010, pp. 408–415. [Online]. Available: http://dx.doi.org/10.1109/LCN.2010.5735752

[12] G. Karame, "Towards Trustworthy Network Measurements," in *TRUST*, 2013, pp. 83–91.

[13] "China's Internet hijack: Attack or accident?" available from http://www.infoworld.com/t/routers-and-switches/chinas-internet-hijack-attack-or-accident-461.

[14] "Iranian Internet Infrastructure and Policy Report," available from http://smallmedia.org.uk//sites/default/files/u8/iiipjune.pdf.

[15] C. V. Wright, L. Ballard, S. E. Coull, F. Monrose, and G. M. Masson, "Spot Me if You Can: Uncovering Spoken Phrases in Encrypted VoIP Conversations," in *Proceedings of the 2008 IEEE Symposium on Security and Privacy*, ser. SP '08. Washington, DC, USA: IEEE Computer Society, 2008, pp. 35–49. [Online]. Available: http://dx.doi.org/10.1109/SP.2008.21

[16] M. Backes, G. Doychev, M. Dürmuth, and B. Köpf, "Speaker Recognition in Encrypted Voice Streams," in *Proceedings of the European conference on Research in Computer Security*, ser. ESORICS'10. Berlin, Heidelberg: Springer-Verlag, 2010, pp. 508–523. [Online]. Available: http://dl.acm.org/citation.cfm?id=1888881.1888921

[17] "US and UK spy agencies have cracked most online encryption," available from http://www.thenational.ae/news/world/americas/us-and-uk-spy-agencies-have-cracked-most-online-encryption.

[18] "The NSA Has A Devastating Backdoor Around Lots Of Web Encryption," available from http://www.businessinsider.com/the-scariest-part-about-the-nsa-access-2013-9.

[19] D. Erickson, "The Beacon Openflow Controller," in *Proceedings of the ACM SIGCOMM Workshop on Hot topics in Software Defined Networking*, ser. HotSDN '13. New York, NY, USA: ACM, 2013, pp. 13–18. [Online]. Available: http://doi.acm.org/10.1145/2491185.2491189

[20] "Open Networking Foundation," https://www.opennetworking.org/. [Online]. Available: https://www.opennetworking.org/

[21] "Foursquare," available from http://foursquare.com/.

[22] "Yelp," available from http://www.yelp.com/.

[23] P. Gill, Y. Ganjali, B. Wong, and D. Lie, "Dude, where is that IP?: Circumventing Measurement-based IP Geolocation," in *Proceedings of the 19th USENIX conference on Security*, ser. USENIX Security'10. Berkeley, CA, USA: USENIX Association, 2010, pp. 16–16. [Online]. Available: http://dl.acm.org/citation.cfm?id=1929820.1929842

[24] M. A. Kâafar, L. Mathy, C. Barakat, K. Salamatian, T. Turletti, and W. Dabbous, "Securing Internet Coordinate Embedding Systems," in *SIGCOMM*, 2007, pp. 61–72.

[25] M. A. Kâafar, L. Mathy, T. Turletti, and W. Dabbous, "Virtual Networks under Attack: Disrupting Internet Coordinate Systems," in *CoNEXT*, 2006, p. 12.

[26] S. Saroiu and A. Wolman, "Enabling New Mobile Applications with Location Proofs," in *Proceedings of Workshop on Mobile Computing Systems and Applications*, ser. HotMobile '09. New York, NY, USA: ACM, 2009, pp. 3:1–3:6. [Online]. Available: http://doi.acm.org/10.1145/1514411.1514414

[27] W. Luo and U. Hengartner, "VeriPlace: a Privacy-Aware Location Proof Architecture," in *Proceedings of the International Conference on Advances in Geographic Information Systems*, ser. GIS '10. New York, NY, USA: ACM, 2010, pp. 23–32. [Online]. Available: http://doi.acm.org/10.1145/1869790.1869797

[28] S. Čapkun, L. Buttyán, and J.-P. Hubaux, "SECTOR: Secure Tracking of Node Encounters in Multi-hop Wireless Networks," in *Proceedings of the 1st ACM workshop on Security of Ad Hoc and Sensor Networks*, ser. SASN '03. New York, NY, USA: ACM, 2003, pp. 21–32. [Online]. Available: http://doi.acm.org/10.1145/986858.986862

[29] A. Sheth, S. Seshan, and D. Wetherall, "Geo-fencing: Confining Wi-Fi Coverage to Physical Boundaries," in *Proceedings of the 7th International Conference on Pervasive Computing*, ser. Pervasive '09. Berlin, Heidelberg: Springer-Verlag, 2009, pp. 274–290. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-01516-8_19

[30] Broadband Forum, "Migration to Ethernet-based DSL Aggregation - Issue 2," Tech. Rep. 101, 2011, http://www.broadband-forum.org/technical/download/TR-101_Issue-2.pdf.

[31] R. T. Fielding and R. N. Taylor, "Principled Design of the Modern Web Architecture," *ACM Trans. Internet Technol.*, vol. 2, no. 2, pp. 115–150, May 2002. [Online]. Available: http://doi.acm.org/10.1145/514183.514185

[32] J. R. Douceur, "The Sybil Attack," in *Revised Papers from the First International Workshop on Peer-to-Peer Systems*, ser. IPTPS '01. London, UK, UK: Springer-Verlag, 2002, pp. 251–260. [Online]. Available: http://dl.acm.org/citation.cfm?id=646334.687813

[33] Y.-C. Hu, A. Perrig, and D. B. Johnson, "Packet Leashes: a Defense against Wormhole Attacks in Wireless Networks," in *INFOCOM*, vol. 3. IEEE, Mar. 2003, pp. 1976–1986 vol.3. [Online]. Available: http://dx.doi.org/10.1109/infcom.2003.1209219

[34] J. M. McCune, B. Parno, A. Perrig, M. K. Reiter, and H. Isozaki, "Flicker: an Execution Infrastructure for TCB Minimization," in *EuroSys*, 2008, pp. 315–328.

[35] J. M. McCune, Y. Li, N. Qu, Z. Zhou, A. Datta, V. D. Gligor, and A. Perrig, "TrustVisor: Efficient TCB Reduction and Attestation," in *IEEE Symposium on Security and Privacy*, 2010, pp. 143–158.

[36] "Government Internet Surveillance Starts With Eyes Built in the West," available from https://www.eff.org/deeplinks/2011/09/government-internet-surveillance-starts-eyes-built.

[37] "Snowden: US Spies On China's Universities and Mobile Firms," available from http://thediplomat.com/china-power/snowden-us-spies-on-chinas-universities-and-mobile-firms/.

[38] "China could be leveraging electronic exports to spy on the US," available from http://www.techspot.com/news/46494-china-could-be-leveraging-electronic-exports-to-spy-on-the-us.html.

[39] R. Dingledine, N. Mathewson, and P. Syverson, "Tor: the Second-Generation Onion Router," in *Proceedings of the 13th conference on USENIX Security Symposium - Volume 13*, ser. SSYM'04. Berkeley, CA, USA: USENIX Association, 2004, pp. 21–21. [Online]. Available: http://dl.acm.org/citation.cfm?id=1251375.1251396

[40] J. C. Mogul, J. Tourrilhes, P. Yalagandula, P. Sharma, A. R. Curtis, and S. Banerjee, "DevoFlow: Cost-Effective Flow Management for High Performance Enterprise Networks," in *Proceedings of the ACM SIGCOMM Workshop on Hot Topics in Networks*, ser. Hotnets-IX. New York, NY, USA: ACM, 2010, pp. 1:1–1:6. [Online]. Available: http://doi.acm.org/10.1145/1868447.1868448

[41] D. Y. Huang, K. Yocum, and A. C. Snoeren, "High-Fidelity Switch Models for Software-Defined Network Emulation," in *Proceedings of the ACM SIGCOMM Workshop on Hot topics in Software Defined Networking*, ser. HotSDN '13. New York, NY, USA: ACM, 2013, pp. 43–48. [Online]. Available: http://doi.acm.org/10.1145/2491185.2491188

[42] NEC, "NEC ProgrammableFlow UNIVERGE PF5240 Datasheet," http://www.necam.com/docs/?id=5ce9b8d9-e3f3-41de-a5c2-6bd7c9b37246.

[43] Y. Chiba, Y. Shinohara, and H. Shimonishi, "Source Flow: Handling Millions of Flows on Flow-based Nodes," in *Proceedings of ACM SIGCOMM*, ser. SIGCOMM '10. New York, NY, USA: ACM, 2010, pp. 465–466. [Online]. Available: http://doi.acm.org/10.1145/1851182.1851266

[44] R. Bifulco, R. Canonico, M. Brunner, P. Hasselmeyer, and F. Mir, "A Practical Experience in Designing an OpenFlow Controller," in *2012 European Workshop on Software Defined Networking (EWSDN)*, 2012, pp. 61–66.

[45] R. Bifulco, M. Brunner, R. Canonico, P. Hasselmeyer, and F. Mir, "Scalability of a Mobile Cloud Management System," in *Proceedings of the MCC workshop on Mobile cloud computing*, ser. MCC '12. New York, NY, USA: ACM, 2012, pp. 17–22. [Online]. Available: http://doi.acm.org/10.1145/2342509.2342514

[46] S. Capkun, M. Cagalj, G. Karame, and N. O. Tippenhauer, "Integrity Regions: Authentication through Presence in Wireless Networks," *IEEE Trans. Mob. Comput.*, vol. 9, no. 11, pp. 1608–1621, 2010.

[47] D. B. Faria and D. R. Cheriton, "No Long-term Secrets: Location-based Security in Over-provisioned Wireless LANs," in *In Proceedings of the ACM Workshop on Hot Topics in Networks (HotNets-III*, 2004.

[48] V. Lenders, E. Koukoumidis, P. Zhang, and M. Martonosi, "Location-based Trust for Mobile User-generated Content: Aplications, Challenges and Implementations," in *Proceedings of the Workshop on Mobile computing systems and applications*, ser. HotMobile '08. New York, NY, USA: ACM, 2008, pp. 60–64. [Online]. Available: http://doi.acm.org/10.1145/1411759.1411775

[49] S. Saroiu and A. Wolman, "I am a Sensor, and I Approve this Message," in *Proceedings of the Workshop on Mobile Computing Systems & Applications*, ser. HotMobile '10. New York, NY, USA: ACM, 2010, pp. 37–42. [Online]. Available: http://doi.acm.org/10.1145/1734583.1734593

[50] P. Gilbert, L. P. Cox, J. Jung, and D. Wetherall, "Toward Trustworthy Mobile Sensing," in *Proceedings of the Workshop on Mobile Computing Systems & Applications*, ser. HotMobile '10. New York, NY, USA: ACM, 2010, pp. 31–36. [Online]. Available: http://doi.acm.org/10.1145/1734583.1734592

[51] G. Karame, B. Danev, C. Bannwart, and S. Capkun, "On the Security of End-to-End Measurements Based on Packet-Pair Dispersions," *IEEE Transactions on Information Forensics and Security*, vol. 8, no. 1, pp. 149–162, 2013.

[52] G. Karame, D. Gubler, and S. Capkun, "On the Security of Bottleneck Bandwidth Estimation Techniques," in *SecureComm*, 2009, pp. 121–141.

[53] S. Shin, P. Porras, V. Yegneswaran, M. Fong, G. Gu, and M. Tyson, "FRESCO: Modular Composable Security Services for Software-Defined Networks," in *Proceedings of the ISOC Network and Distributed System Security Symposium*, 2013.

[54] P. Porras, S. Shin, S. Yegneswaran, M. T. M.W. Fong, and G. Gun, "A Security Enforcement Kernel for OpenFlow Networks," in *Proceedings of the ACM Sigcomm Workshop on Hot Topics in Software Defined Networking (HotSDN)*, 2012.

[55] S. Shin, V. Yegneswaran, P. Porras, and G. Gu, "Avant Guard: Scalable and Vigilant Switch Flow Management in Software-Defined Networks," in *Proceedings of ACM Computer and Communications Security (CCS)*, 2013.

[56] D. G. Andersen, H. Balakrishnan, N. Feamster, T. Koponen, D. Moon, and S. Shenker, "Accountable Internet Protocol (AIP)," *SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 4, pp. 339–350, Aug. 2008. [Online]. Available: http://doi.acm.org/10.1145/1402946.1402997

[57] X. Zhang, H.-C. Hsiao, G. Hasker, H. Chan, A. Perrig, and D. G. Andersen, "SCION: Scalability, Control, and Isolation on Next-Generation Networks," in *Proceedings of the 2011 IEEE Symposium on Security and Privacy*, ser. SP '11. Washington, DC, USA: IEEE Computer Society, 2011, pp. 212–227. [Online]. Available: http://dx.doi.org/10.1109/SP.2011.45