Driving Software Defined Networks with XSP

Ezra Kissel
Department of Computer & Information Sciences
University of Delaware, Newark, DE 19716
Email: kissel@cis.udel.edu

Guilherme Fernandes, Matthew Jaffee,
Martin Swany, Miao Zhang
Indiana University, Bloomington, IN 47405
Email: {fernandes,mjaffee,swany,miaozhan}@indiana.edu

Abstract—**This paper presents the eXtensible Session Protocol (XSP), which provides a control plane for driving Software Defined Networks (SDNs). The XSP model supports proactive, application-driven configuration of dynamic network resources with support for authentication and authorization, within an extensible protocol framework. We describe XSP application use cases in SDNs using OpenFlow enabled network devices as well as dynamic forwarding rule management that can be implemented on existing router platforms.**

I. Introduction

Software Defined Networks (SDNs) have the potential to revolutionize how networks are built, controlled, and managed. By allowing the forwarding (or flow) hardware in network devices to be controlled by external software, SDNs can enable an unprecedented amount of flexibility in network operation. OpenFlow [1] provides a standard mechanism to realize SDNs, but is not the only possibility. A broad spectrum of possible network control can be realized through SDNs, from centralized computation of traditional routes, to explicit virtual machine routing, to per-flow control.

This new paradigm requires new approaches that will enable users and applications to take a more active role, along with the development of protocols and architectures that will drive these dynamic network environments. By abstracting out common functionality inherent in the operation of a network, SDNs let us consider networking as a service that can be manipulated using standard interfaces.

Common networking functionality such as topology discovery and MAC address learning in Layer 2 environments are simplified in SDNs by locating topology information in a centralized controller. In many cases, the topology of the network is well known and the repeated discovery overhead is unnecessary. In addition, this offloading of topology information reduces the memory footprint on the physical switch and provides a flexible software interface for fine grained control.

Another common SDN model is installation of rules on demand. However, this type of flow micromanagement raises a number of scalability issues, and in many cases, such as those involved with setting up traffic engineered paths for bulk-data flows, this reactive behavior can be viewed as an unnecessary complication. In this paper, we present a model where rules or "flow entries" can be installed proactively, based on the requirements of the application. We contend that this approach reduces overhead for devices in the data center as well as the management and configuration burden on sites across administrative domains.

To that end, we present the eXtensible Session Protocol (XSP) for use in Software Defined Networks. The goal of XSP is to provide a general and extensible protocol and associated framework for managing the interaction between applications and network-based services, and between the devices that provide those services. Based on the OSI session model [2], an XSP protocol exchange can include a "call setup" phase, potentially with a network element other than the transport layer destination. This allows applications that implement the protocol to explicitly, and securely, signal network services to configure the network on their behalf. Our premise is that if applications, particularly network-intensive applications, signal the network before connecting, we can add a new dimension of control to software defined networks. The control channel established transparently between applications and network services through XSP permits the exchange of far more application-specific context. Applications may indicate connectivity requirements that cannot be derived by the network from packet headers alone. XSP represents an architecture for applications to interact with SDNs, regardless of how they are implemented.

We make the following contributions in this paper: XSP is presented as a novel and extensible framework for driving software defined networks; We analyze the overhead of our session-based approach and contrast it to current OpenFlow networks; finally, we present evidence in favor of the proactive, session-based approach in two practical implementations.

II. Overview

A key insight into the design of XSP is that communication at session initiation can provide a number of benefits for managing application connections, including the on-demand setup and configuration of network services. XSP provides a collection of modules, or building blocks, that can be used by applications to interact with advanced network technologies while providing a framework for adding additional functionality as necessary. The essence of this approach is that the XSP session layer lives above the transport layer, and provides protocol encapsulation for both control and data protocol units between peer session entities. An XSP API provides a standard interface for applications and services to interact with the underlying session layer protocol implementation.

Historically, the role, architecture, and necessity of the session layer has been questioned. In the prevalent TCP/IP, or "Internet" model [3] in use today, the upper layer protocol considerations outlined within the OSI model have received little attention in most network architecture designs. Although the design and operation of the Internet today is no doubt an impressive feat, it could also be described as a victim of its very own success. Investigating new protocols and developing novel network architectures have become significant challenges as any functional changes within the current model have serious deployment consequences. The irony is that as networks become more heterogeneous, smarter, and increasingly dynamic, many of the session layer dialog and control primitives have once again become desirable and are being explored as a way to mitigate the barriers in effecting significant and real change within the TCP/IP model.

Recently, functionality consistent with a session layer has experienced renewed interest, particularly in the realm of mobile and delay-tolerant networks. The session layer can provide a number of capabilities for managing transport re-binding, device naming, and connection recovery and checkpointing [4], [5], as well as providing an interface with which to manage in-the-network services. Various "future Internet" projects include similar functionality which may indeed become essential as networks become increasingly dynamic and network topologies, connectivity models, and usage patterns become more varied. Traditional end-to-end arguments are also being re-evaluated as these advances drive new modes of operation [6].

With the advent of software defined networking, architectural considerations are now at the forefront as new methods are needed that will provide secure access and scalable management for these dynamic environments. Our work with XSP begins to address many of these concerns while providing a framework for incremental deployment with existing implementations.

III. The XSP Session Layer

The XSP Session Layer is designed as a collection of modular service handlers, or *SH*, that are accessed through a common API. Providing a standard interface to the network, each SH uses the underlying XSP protocol to communicate between session-enabled endpoints. The XSP network architecture is illustrated in Figure 1, with the SH layer indicated by dotted boxes. Details of the session protocol implementation can be found in [7].

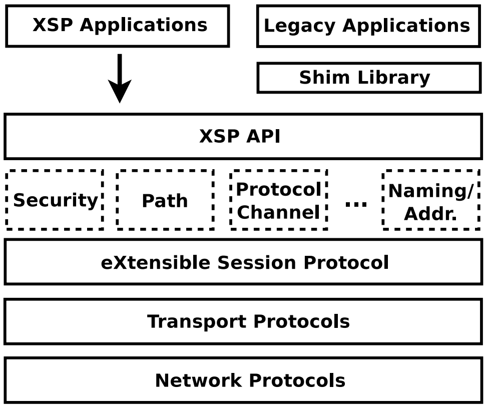


Fig. 1: XSP stack architecture

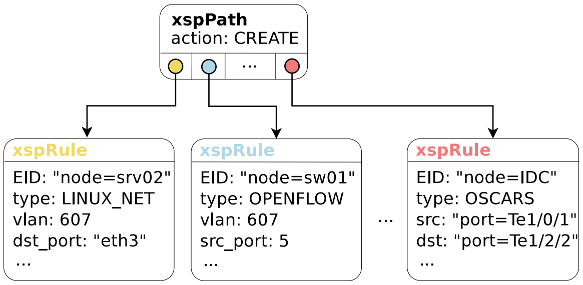
Each SH provides a framework for implementing a specific instance of that service for the application. For example, the *Security* SH makes available a number of different authentication methods that can be requested. The SH layer is implemented as a set of modular libraries that can be dynamically loaded when requested by the application. Additional modules can be added by simply implementing another handler within the desired XSP SH framework. We show a subset of the available framework components in the SH layer.

Of particular importance in the context of SDNs is the *Path* SH, which provides a number of modules for configuring network devices. We describe this feature in further detail below. Additionally, XSP supports the use of various control and data channels with the *Protocol Channel* SH, and the *Naming/Addressing* SH allows for the integration of external endpoint location and identification systems. One of the challenges for the SH abstraction is defining the core set of features that the session layer should provide. While the current SH implementation enables applications to use what we consider a common set of built-in features (i.e. the current state of the art), the true advantage is realized with the extensibility of the framework in supporting additional capabilities, all within a consistent interface, as future network architectures evolve and take shape.

A. XSP Path Model

The XSP framework has been extended to provide applications with an interface for general network configuration. The *Path* SH enables the extensible and interchangeable use of a growing set of technologies. These include modules for configuring devices that speak a common protocol such as OpenFlow, network configuration of Linux-based hosts, and also services like OSCARS [8] that can dynamically provision network paths in "circuit networks". The *Path* framework provides a common abstraction that hides the implementation specifics of each handler, presenting a consistent interface to the user while still allowing access to technology-specific options (e.g. OpenFlow 15-tuple) if desired.

The *Path* interface exposed via the XSP API presents an **xspPath** structure to the application which can specify a set of dynamic path actions such as `CREATE`, `MODIFY`, `DELETE`, `QUERY`, etc. This structure can be manipulated to contain a number of **xspRule** entries that, taken together, define a complete end-to-end path, or it may simply contain a single rule to effect some particular configuration on a specific device. Figure 2 illustrates this structure.



action: CREATE

EID: "node=srv02"
type: LINUX_NET
vlan: 607
dst_port: "eth3"
...

EID: "node=sw01"
type: OPENFLOW
vlan: 607
src_port: 5
...

EID: "node=IDC"
type: OSCARS
src: "port=Te1/0/1"
dst: "port=Te1/2/2"
...

Fig. 2: An xspPath structure contains a list of "rules" to be configured by the indicated *Path* SH modules.

The xspPath structures are encoded by the XSP protocol implementation and communicated along the active session. An XSP-enabled service configured with the *Path* SH indicated within the xspRule entries loads the appropriate handler and applies the specified configuration. An XSP *Endpoint Identifier*, or EID, within the xspRule entries allows the handlers to identify and locate the particular network device or service to configure. The underlying handler interacts with the active session to provide acknowledgements if the operation completed successfully, or descriptive NACKs if it was unable to apply some or all of the device configuration.

B. XSP Daemon for SDN

To support XSP deployments, we have developed a service known as the *XSP Daemon*, or XSPd, which implements the core XSP protocol functionality and a configurable SH layer. This service is known as XSPd-SDN in its role as a session-based controller for software defined networks. In particular, the *Path* SH module for OpenFlow integrates the Stanford reference controller to support the proactive installation of flow entries signaled by XSP-enabled applications. A typical deployment of XSPd-SDN involves one or more instances within administrative domains, and pushing a configuration into the network may involve an exchange between multiple XSP-enabled controllers along a given path.

C. Authentication and Authorization

The initial XSP exchange allows explicit Authentication (AuthN) and Authorization (AuthZ). The mechanism for AuthN can make use of X.509 and SSH credentials, in addition to a number of password-based or external mechanisms. While any AuthZ policy can be imagined and implemented via the *Security* SH, the current implementation uses very simple "flowspace" rules to express authorized use. Of course a null authentication with IP address-based identity can still be subject to AuthZ policy – much like in today's firewalls.

A common scenario is authenticating a valid user when an application seeks to dynamically configure the network via XSP. The XSP controller maintains one or more Certificate Authorities (CAs) which user or host certificates may be validated against. In the SSH case, user-based authentication can be more simply managed via keys that match against active system accounts and may be subject to authorized use via pre-defined user attributes.

D. Application Support

The core XSP library is known as *libxsp*. Building an XSP-enabled application involves linking against *libxsp* to access the main protocol functionality, SH framework, and configurable connection support for establishing new sessions. An *Application* SH provides hooks for supporting application messaging via external modules. These application-defined handlers implement the desired message format and serialization/deserialization methods needed to process the message before being encoded/decoded by the XSP protocol implementation. This approach allows XSP to communicate application-specific information in a general manner while simultaneously maintaining session state between connected session peers.

The current XSP implementation includes a Sockets API-compatible client library, *libxsp_client*, that is being retooled to expand feature support. This provides familiar semantics such as *open*, *close*, *connect*, *send*, *recv* and extends these with session-specific calls that interact with the SH layer. Thus, an application may establish a session for a reliable, stream-oriented connection, as would be provided by TCP, while being able to configure a desired authentication scheme, dynamic network path, separate data channels, etc., all from a common interface.

Our XSP implementation also provides a transparent wrapper, known as the *Shim Library* indicated in Figure 1. Using library interposition (e.g. via the Linux LD_PRELOAD mechanism), the wrapper allows existing applications to take advantage of XSP without requiring any source code modifications.

IV. XSP AS AN INTERFACE TO SDNS

One design goal of XSP is to provide applications and systems with a mechanism to directly interact with SDNs, with support for authentication and authorization. The essence of the "southbound" interface of an SDN (e.g. OpenFlow) is the installation of rules in the data plane. At least one of the "northbound" interfaces must be simply the offering of potential rules to a network element for installation, subject to authorization policy. While it is beyond the scope of this paper, it is interesting to note that a BGP session is essentially the same thing: one BGP peer starts a session with another and offers reachability information, which consists of implied forwarding rules.

One of our use cases involves creating "dedicated" paths for bulk data transfers. Another is having a network edge admit only those application flows that are authenticated and which match a particular policy – effectively adding firewall rules dynamically. In Figure 3, we show how XSP can be used to allow a remote endpoint to control or influence the network via an authenticated session.

SDNs allow us to treat networks as a service. Just like applications can affect the operating systems they run on, we imagine similar functionality for a software defined network. XSP is an ideal interface to this kind of network service. Techniques like virtualizing a network into "slices" provide separation of control over resources that XSP can leverage. GENI [9] is an example of how networks might look in the future, with dynamic on-demand provisioning of network resources for a particular user or enterprise. Policy is maintained by the resource operators, but configuration is exposed to applications via a standard interface.

The notion of a *session* in XSP, in the most natural sense of the word, provides an inherent scope for state, including forwarding rule lifetime. Applications can refer back to that state, make changes, and configure the network to suit their particular requirements.

An open question for SDNs is how to best manage configuration across administratively unique domains, each with potentially different managed devices and AuthN/AuthZ requirements. XSP is designed so that applications can make generic *Path* requests through the XSP API, which are then translated to technology-specific configurations at the XSPd-SDN within the given domain. Configuration state is forwarded between neighboring XSPd-SDN instances, and each XSPd-SDN applies the specified *Path* and AuthZ parameters that are received over the active XSP session. The session connections between XSPd-SDN instances can use various security schemes and credentials that may exist between sites, and negotiate for acceptable authorization.

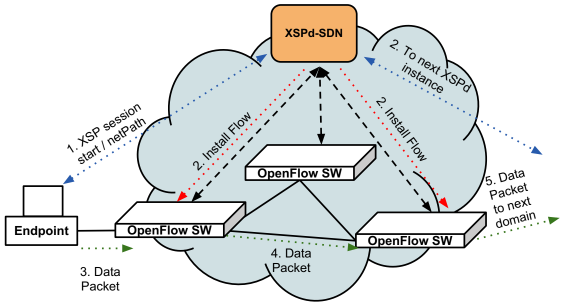


Fig. 3: An application-driven XSP session proactively configuring the network

Our model also makes no assumptions about topology-awareness from the application's perspective. A *Path* request may simply ask for a path from endpoint *A* to endpoint *B*, provide a complete specification of the flow entries along *A* to *B*, or some level of detail in between. The XSP implementation is designed to remain flexible enough to incorporate a number of solutions. A balanced approach could let XSPd-SDN instances manage their local domain topology, allowing path discovery and exchange through in-network services while the application can more appropriately specify path characteristics such as duration and quality of service.

A. Evaluating overheads

When a packet arrives at an OpenFlow switch, the default behavior is for the switch to send a *Packet-IN* control message to its associated controller. We call this the *reactive* OpenFlow case. A typical controller operation involves modifying the forwarding table of the switch (i.e. installing a "flow entry") so the next matched packet in the flow is switched across the data plane. Resources like packet buffers, cross-sectional bandwidth between data plane and CPU, and the CPU in the network device itself are all potential bottlenecks. The ability for switch implementations to scale this approach as data center's traffic increases is a cause for concern, and has been the focus of a number of research efforts [10], [11]. In contrast, our *proactive* installation of flow entries with XSPd-SDN ahead of the application data keeps edge switches operating in the data plane, avoiding the requirement to handle new flows over the slow path altogether.

Although we remove the dependence on per-flow processing at the switch, the overhead involved in establishing an XSP session takes time, and authentication adds to this time. Table I shows the times to connect for various latencies between the client and a running XSPd instance. We can compare these with the time it takes to generate a *Packet-IN* in the purely reactive case. While all these numbers are based on current implementations, and thus subject to change, we observe about 3ms overhead for a *Packet-IN* on a current switch.

	Anonymous		**SSL/X.509**	
Latency	Mean (ms)	σ	Mean (ms)	σ
2ms	7	.013	17	.078
5ms	16	.007	32	.074
10ms	31	.015	57	.06

TABLE I: XSPd-SDN session establishment overhead

V. Practical Use Cases

A. Application-directed path selection

Giving applications the ability to influence the network opens a variety of possibilities; if certain applications could choose their own paths, their traffic could be re-routed and shaped without adding additional computational expense inside the network. We have demonstrated the potential for application-directed path selection at a recent conference (SC11), by successfully using XSP to allow a file transfer application to dynamically switch its traffic from a shared path to a dedicated path when poor network performance was detected. Figure 4 illustrates a representative topology, where domains A and B are connected through at least two or more distinct WAN paths. On each endhost, we ran GridFTP [12] servers that were extended with a loadable XSP driver implemented within the Globus XIO [13] framework on which GridFTP is built. The driver included instrumentation and a performance threshold parameter that allowed it to signal the active control session when the transfer rate fell below a given value. As a GridFTP transfer began, the XSP driver initialized a session with the running XSPd-SDN instances in both domains. These XSP-enabled controllers installed flow entries within the OpenFlow network allowing the file transfer to proceed along the default shared path. As increasing amounts of competing traffic was introduced along the path, the GridFTP transfer was severely impaired, triggering an XSP *Path* signal that allowed the matching GridFTP flows to be redirected over the dedicated path. The actual reconfiguration was performed by the XSPd-SDN instances in each domain, which updated the OpenFlow switch flow entries based on this application feedback. The observable transfer performance was instantly improved without any active intervention by users or network operators.

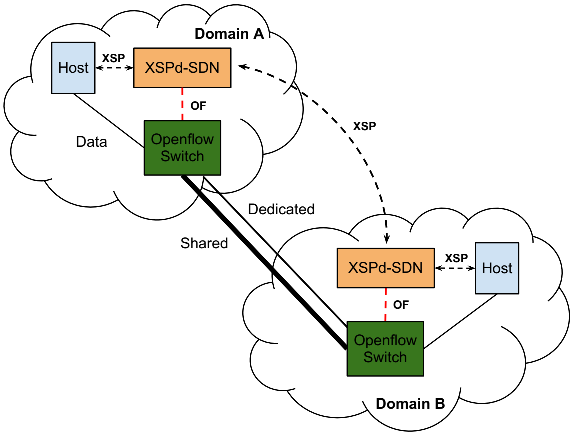


Fig. 4: XSP controlling path selection

B. A dynamic firewall

Traditional firewalls have a variety of issues - they tend to be statically configured and require administrator action to insert and remove rules. We propose the use of XSP as a framework for creating a dynamic firewall. We can imagine a scenario where a network has a “default drop” policy; when an application or service needs to `connect()` or `listen()`, it can actively notify the network using XSP, triggering the installation of appropriate rules. Using AuthN/AuthZ negotiated over a session, an XSPd-SDN instance enables secure access to firewall functionality, applied on a per-user basis. A timeout and renewal mechanism can be implemented to ensure that firewall rules do not become stale.

We have prototyped such a system running on Juniper MX routers through the use of the JunOS SDK [14]. Our evaluation involved deploying a version of XSPd-SDN as a JunOS SDK application on two Juniper MX240 routers. Juniper MX routers contain a routing engine (RE), and a packet forwarding engine (PFE). The RE runs applications which control the packet forwarding behavior, thus allowing us to define the device configuration via software.

XSPd-SDN runs on the RE of each router and controls the PFE via the JunOS SDK API. This API includes a dynamic firewall library, which provides an interface to enable match-based filters and routing behavior that can discard, threshold, and redirect traffic on the PFE. An XSP-enabled RE effectively gives the router the capability to act as a flow manager. A deployment of such XSP-enabled routers as part of a broader SDN site installation gives applications the ability to manipulate network traffic in an arbitrary, yet policy-enforced, fashion without increasing operational overhead.

Similar to the case above, another useful scenario involves bypassing existing firewall devices altogether. The per-packet processing in firewalls can be a significant source of overhead, and thus they are often performance bottlenecks. In the same way that we can install forwarding rules *in* firewalls, XSPd-SDN allows us to redirect particular flows *around* stateful firewalls, based on an authenticated and authorized session. Application controlled SDNs allow us to remove sources of overhead without sacrificing security.

VI. Related Work

A number of SDN control plane approaches such as DIFANE [11], Hedera [15], HyperFlow [10], and Maestro [16] have proposed to manage scalability and control of SDNs, and OpenFlow networks in particular. These systems deal primarily with data center environments, and do not consider a framework for application-driven control of SDNs as in XSP. Other efforts efforts such as ForCES [17] and NSIS [18] propose frameworks for logically separating control and data planes within network processor devices. The scope of these approaches is tied to the interaction of closely coupled network elements whereas XSP aims to provide a mechanism to signal across applications and devices from an end-to-end perspective. We envision that XSP may interface with a ForCES control element for managing that particular class of network device. Our dynamic firewall work is related to systems for managing access [19] and delegating network security [20]. XSP addresses similar needs through the *Security* SH framework.

VII. Conclusion

We have presented the eXtensible Session Protocol (XSP) as a framework for applications to interface with SDNs in a secure and dynamic manner. We discuss the overhead of our approach and contrast that with the per-flow overhead of existing approaches. We have demonstrated a working model of XSP supporting application-directed path selection, and have prototyped a session-enabled dynamic software firewall. It is our belief that XSP provides a solid foundation for enabling new modes of network operation within future Internet architectures. As future work, we will investigate the bidirectional nature of XSP in order to provide network feedback for session-enabled applications. In this model, XSP provides information about network state that can be used to effectively "pre-tune" transport layer connections with the goal of improving the performance of the application.

References

[1] N. McKeown, T. Anderson, H. Balakrishnan, G. M. Parulkar, L. L. Peterson, J. Rexford, S. Shenker, and J. S. Turner, "Openflow: enabling innovation in campus networks," *Computer Communication Review*, vol. 38, no. 2, pp. 69–74, 2008.

[2] A. N. S. for Information Processing Systems, "Open systems interconnection – basic connection oriented session protocol specification," ANSI/ISO 8327-1987, 1992.

[3] "Requirements for internet hosts - communication layers," United States, 1989.

[4] M. Demmer and K. Fall, "The design and implementation of a session layer for delay-tolerant networks," *Comput. Commun.*, vol. 32, no. 16, pp. 1724–1730, 2009.

[5] Y. Ismailov, K. Palmskog, M. Widell, P. Arvidsson, and Y. Wang, "Session layer resurgence: Towards mobile, disconnection- and delay-tolerant communication," in *ECUMN '07: Proceedings of the Fourth European Conference on Universal Multiservice Networks*. Washington, DC, USA: IEEE Computer Society, 2007, pp. 337–345.

[6] M. S. Blumenthal and D. D. Clark, "Rethinking the design of the internet: the end-to-end arguments vs. the brave new world," *ACM Trans. Internet Technol.*, vol. 1, no. 1, pp. 70–109, 2001.

[7] E. Kissel and M. Swany, "The extensible session protocol: A protocol for future internet architectures," Technical Report UDEL-2012/001, http://damsl.cis.udel.edu/projects/phoebus/kissel_xsp.pdf.

[8] "ESnet On-demand Secure Circuits and Advance Reservation System (OSCARS)," http://www.es.net/oscars/.

[9] "Geni: Global Environement for Network Innovations," http://www.geni.net/.

[10] A. Tootoonchian and Y. Ganjali, "Hyperflow: a distributed control plane for openflow," in *Proceedings of the 2010 internet network management conference on Research on enterprise networking*, ser. INM/WREN'10. Berkeley, CA, USA: USENIX Association, 2010, pp. 3–3. [Online]. Available: http://dl.acm.org/citation.cfm?id=1863133.1863136

[11] M. Yu, J. Rexford, M. J. Freedman, and J. Wang, "Scalable flow-based networking with difane," *SIGCOMM Comput. Commun. Rev.*, vol. 40, pp. 351–362, Aug. 2010. [Online]. Available: http://doi.acm.org/10.1145/1851275.1851224

[12] "GridFTP," http://www.globus.org/datagrid/gridftp.html.

[13] I. Foster and C. Kesselman, "Globus: A metacomputing infrastructure toolkit," *International Journal of Supercomputer Applications*, vol. 11, pp. 115–128, 1996.

[14] "Creating innovative embedded applications in the network with the junos sdk," White Paper, Juniper Networks, February 2011. [Online]. Available: http://www.juniper.net/us/en/local/pdf/whitepapers/2000378-en.pdf

[15] M. Al-fares, S. Radhakrishnan, B. Raghavan, N. Huang, and A. Vahdat, "Hedera: Dynamic flow scheduling for data center networks," in *In Proc. of Networked Systems Design and Implementation (NSDI) Symposium*, 2010.

[16] Z. C. Alan, L. Cox, and T. S. E. Ng, "Maestro: A system for scalable openflow control," Technical Report, Rice University, http://www.cs.rice.edu/~eugeneng/papers/TR10-11.pdf.

[17] A. Doria, R. Haas, J. Hadi Salim, H. Khosravi, and W. M. Wang, "ForCES Protocol Specification," Internet Draft draft-ietf-forces-protocol-22.txt, March 2009.

[18] "NSIS: Next steps in signaling," http://datatracker.ietf.org/wg/nsis/charter/.

[19] A. K. Nayak, A. Reimers, N. Feamster, and R. Clark, "Resonance: dynamic access control for enterprise networks," in *Proceedings of the 1st ACM workshop on Research on enterprise networking*, ser. WREN '09. New York, NY, USA: ACM, 2009, pp. 11–18. [Online]. Available: http://doi.acm.org/10.1145/1592681.1592684

[20] J. Naous, R. Stutsman, D. Mazires, N. McKeown, and N. Zeldovich, "Delegating network security with more information," in *Proceedings of the 1st ACM SIGCOMM 2009 Workshop on Research on Enterprise Networking, WREN 2009, Barcelona, Spain, August 21, 2009*, J. C. Mogul, Ed. ACM, 2009, pp. 19–26.