SFP: Toward a Scalable, Efficient, Stable Protocol for Federation of Software Defined Networks

Franck Le[*] Christopher Leet[‡] Christian Makaya[*] Miguel Rio[†] Xin Wang[+‡] Y. Richard Yang[+‡]
IBM[*] Tongji[+] UCL[†] Yale[‡]

***Abstract*—As more networks deploy software defined networking (SDN) to take advantage of its benefits, including logically centralized, more flexible network-control programming, the inter-connection of such SDN-based networks has become a major remaining challenge. Traditional interconnection approaches such as BGP are designed for less flexible networks and hence can have serious efficiency, scalablity and stability issues such as the compact policy program instantiation (CPPI) issue. In this paper, we define the requirements of the inter-connection of SDN networks and propose a novel protocol, called the SDN Federation Protocol (SFP), to achieve scalable, efficient, and stable interconnection of SDN networks. Instead of being a traditional push protocol such as BGP, SFP adopts a novel pub-sub model to substantially increase flexibility, efficiency and scalability. Going beyond only packet handling, SFP introduces flexible network information spaces, such as the packet space and the flowset space, to achieve more efficient, autonomous federation of network resources spanning across multiple networks.**

Keywords: SDN, interdomain, BGP, SFP

I. Introduction

A major recent development in computer networking is the introduction of software defined networking (SDN), which allows a network to define its behaviors through centralized policies at a conceptually centralized network controller. With the emergence of key components including (1) south-bound datapath models (*e.g.*, Openflow [1], P4 [2]), (2) high-level programming models (*e.g.*, Frenetic [3], Maple [4]), (3) north-bound application interfaces (*e.g.*, IETF supa), and (4) emerging killer SDN applications (*e.g.*, B4 [5]), SDN has made significant progress in realizing the vision that a network can be effectively controlled using a simple, centralized control-plane program with a global view.

As individual networks start to deploy SDN, it is a natural next step to interconnect such SDN networks. Although there are many settings where fully localized, transparent deployment of SDN is enough to reap the benefits of SDN (*e.g.*, edge domains [6]), there are also many settings where the interconnection of individually administrated SDN networks can be beneficial, for example, in expanded reachability settings, where one network needs to reach entities hosted in another network, and in pooled resources settings, where multiple networks benefit from resources made available from each other, as long as the sharing conforms to their local policy programs. In the rest of this paper, when we refer to an SDN network, we mean an individually administrated network. We refer to the interconnection of individually administrated SDN networks as realizing the federation of involved SDN networks or SDN federation for short.

Achieving SDN federation can be challenging, and hence existing work (*e.g.*, SDX [7], SD-WAN) focuses on limited settings. Since some might think that one can still use traditional interdomain protocols to achieve SDN federation, consider the issues when applying BGP, the well-developed, *de facto* interdomain protocol of the Internet, to achieve SDN federation. Unfortunately, applying BGP to SDN federation can have at least two basic mismatches, and both can result in substantial efficiency and scalability issues. First, designed in a traditional networking setting using only destination-IP based routing, BGP uses a single-dimension (*i.e.*, the destination-IP dimension) routing information model. An SDN network, however, can conduct highly flexible, multi-dimension routing, based on a large number of decision dimensions, including not only destination IP, but also source IP, protocol, and port numbers. Naive conversion of multi-dimension SDN decisions to the BGP single-dimension information model can lead to dimension cross-product explosions, resulting in serious scalability issues. Second, designed in a traditional networking setting with limited programmability, BGP is fundamentally a *full instantiation* information-exchange protocol, in that the program decisions at each network need to be fully instantiated as data (*i.e.*, BGP routing information base) and then exchanged among networks. The extremely large decision space of SDN, and in particular, the two-layer SDN decision model, where the routing information base layer is only a cache of the SDN program layer, can make full instantiation unfeasible.

The objective of this paper is to conduct a systematic investigation and design of SDN federation. Instead of adopting an incremental approach of adapting an existing design such as BGP, we conduct a clean design of SDN federation, to address fundamental challenges introduced by SDN, including multi-dimension decision, two-layer SDN decision.

It turns out that the aforementioned challenges can be addressed using simple, extensible techniques, which we introduce as key components of a novel protocol called the SDN federation protocol (SFP). Specifically, to avoid full instantiation, SFP uses a simple, novel pub-sub model to constrain the space where program behaviors are converted into data to be exchanged; to avoid dimension cross-product explosion, SFP introduces novel, information instantiation graph; going beyond only packet handling, SFP introduces flexible network information spaces, such as the flowset space, to achieve more efficient, autonomous federation of network

resources spanning across multiple networks.

The rest of this paper is organized as follows. Section II gives the requirements of SFP. Section III evaluates related work and lists the issues of existing work. Sections IV is the main technical section and gives the SFP design. Section V gives future research directions to conclude the paper.

II. Design Requirements

We list the following requirements for a SDN federation protocol:

- *Efficiency of resource integration across networks*: The protocol should be low overhead, and scalable. The scalability requirement applies to memory, network, and compute resources. For memory resources, for example, under the BGP paradigm, an AS advertises all of its routes to its neighbors, even if a neighbor may have no interest nor traffic to specific destinations. In contrast, a different communication model may have domains subscribe to specific header spaces they are interested in, and only receive information (including updates) associated with those header spaces. For network resources, the protocol should make efficient usage of the bandwidth across domains. Finally, for compute resources, a domain must be able to reply to a neighbor's query within a reasonable time period.

- *Autonomy*: There is a trade-off between autonomy and stability: When every node implements the same algorithm, stability can more easily be guaranteed (*e.g.*, shortest path routing). However, this model is rigid, and does not allow each domain to implement its own policy. Conversely, in a system where each domain can implement its own policy, policies from different domains may conflict and result in instabilities [8]. The protocol should offer a large degree of autonomy to its domains while still guaranteeing the stability of the system.

- *Privacy*: Although a first domain may send multiple queries to a second domain, the first domain should not be able to learn information that the second domain considers sensitive and chooses not to reveal to the first domain.

III. Related Work

With the preceding requirements, we now evaluate related work. Since SDN was initially developed for single administrative domains (*e.g.*, enterprise, datacenter) [6], *i.e.*, intra-domain management and control, only recently are there a number of proposals to extend the benefits of SDN across domains (*e.g.*, [7], [9]–[13]). Many of these proposals adopt an incremental approach and suggest extending, or defining a new protocol similar to, BGP – the current de-facto inter-domain routing protocol – to carry additional information [9], [11], [12]. However, extending BGP to support SDN does not satisfy all of the requirements in Section II. In particular, using BGP presents the following issues:

Cross-product explosion: One of the main benefits of SDN is the support for highly flexible, multi-dimension routing, based on a large number of decision dimensions, including IP addresses, protocol, and port numbers. In contrast, BGP uses only the destination IP as the routing decision. Simply extending the single dimension (*i.e.*, destination IP) of BGP to multiple dimensions can lead to dimension cross-product explosions, and scalability issues. For example, routing for N destinations taking into account up the source IP addresses and destination ports may require up to N x M x P entries, with M representing the number of source IP addresses, and P the number of destination ports.

Full instantiation: BGP is a full instantiation information-exchange protocol. It requires every Autonomous System to compute and advertise the best route for every destination *a priori*. SDN has a much larger decision space, and adopting this model where every network computes and exchanges its decisions for every set of flows *a priori* may be infeasible.

Domain backup: Networks may require reachability despite link failures, router failures, and network partitions, as long as a physical path to the destination exists. However, BGP does not support domain backup and prevents paths having traversed a given Autonomous System from being re-injected into that same Autonomous System. Despite it being a design decision, opertional networks may require domain bcakup [14], [15].

Multi-flows handling: Policies in SDN networks not only control the reachability of flows but also manage resource allocations (*e.g.*, bandwidth) of flows which are not independent from each other. Hence, multi-flow resource querying is more complicated than a single flow due to sharing links, concurrency of flows, and internal sensitive information preservation which is not considered in BGP model.

Although protocols beyond BGP have been proposed, they cannot satisfy the requirements in Section II.

Specifically, a main interdomain protocol for SDN is SDX [7], which enables more expressive policies than conventional hop-by-hop, destination-based forwarding without requiring extensions to BGP. However, SDX relies on a Route Server deployed at Internet Exchange Point (IXP). Autonomous Systems write their policies to the Route Server, and only Autonomous Systems participating at the IXP can therefore benefit from it. In contrast, our requirements include that it is not restricted to Autonomous Systems present at specific IXPs. In addition, our requirements include resource integration to provide resource information (*e.g.*, available bandwidth) across networks. IXP cannot satisfy this requirement.

On the standardization side there has been some limited efforts to standardize interfaces between SDN controllers in different domains. SDNi [9] defines 3 elements: an SDN aggregator to aggregate intra-domain information, a RestAPI

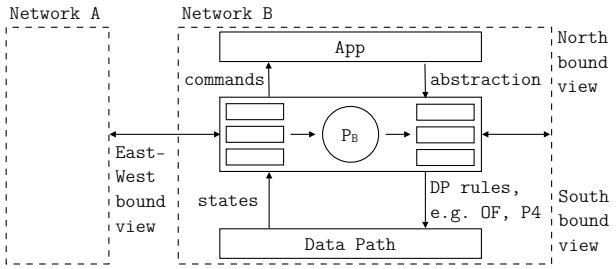

Fig. 1. The unifying model-views framework.

to obtain information from other domains and SDNi wrapper to share information from federated controllers.

Bueno et al. propose the Network Control Layer (NCL) [16], a software framework solution based on SDN, OpenFlow and Network as a Service (NaaS) paradigms. Petropoulos et al. [17] propose a way of coordinating Quality of Service (QoS) across different domains. They do not address all of the requirements in Section II.

IV. SFP Design

We now present the design of SFP. We start with an overview of the basic structure of SFP in Section IV.A. We then present in Section IV.B a key component of SFP, the SFP packet space view.

A. Overview

The model-views context: Considering SDN as a core concept, we design SFP in the context of a unified model-views framework, where the logically centralized control program P_A of network A defines the core model. To be concrete, we show an example P_A:

```
f(Packet pkt):
   if (pkt.tcpSrcPort != 80)
      return drop
   else
      srcClass = map_policy(pkt.Ipv4Src)
      return shortestPath(srcClass, pkt.Ipv4Dst)
```

It is from this model that multiple views are derived, where the views include the north bound (NB), which is the interface of the network to applications using the network; the south bound (SB), which is the realization from the control plane to the data plane; and the east-west bound (EWB), which is the interface between peering networks and hence is the focus of this paper. See Fig. 1 for illustration of this unifying framework. Adopting a unified model-views framework leads to multiple benefits, including automation (as views should be derivable from the model), sharing of common functions when computing multiple views, and consistency (among views, as implied from their consistency to the model).

Each view is different because it has different targets; for example SB is for switches and EWB is for peering networks. Although the focus of this paper is SFP, which provides the EWB view, since a reader may be more familiar with the SB view, we compare the EWB view with the SB view in Table I, to allow the reader better appreciate the SFP design.

Southbound	**East-west Bound**
Switch places physical constraints on SB, as SB must conform to switch computing models such as OpenFlow, and no loop in routing pipeline	No such constraints, but preferable to be similar to existing concepts such as BGP
No privacy concerns	Privacy concerns
Must compile into distributed devices	No such constraint
Controller can process packets (only partial compilation required)	Program must be fully compiled and transmitted

TABLE I
EWB vs SB pipeline comparison.

EWB spaces: In a high level, what a network B provides to another network A is how P_B will handle each packet that A is interested in. In the basic level, P_B processes each packet independently. Hence, the basic information that B needs is how each packet in a *packet space* is handled. As networks provide more services such as QoS and security services, P_B classifies packets into flows. Since the processing of flows can be correlated, in particular, in resource sharing case, we say that A may request information for a set of flows. We refer to this as the *flowset space*, where each point in the space is a set of flows.

In the general case, to better organize information, the EWB view consists of multiple spaces, with two predefined spaces: the packet space and the flowset space.

Basic SFP message flow: With the basic concepts of EWB spaces, we specify the basic SFP message flow, as illustrated in Fig. 2. It is a sub/pub protocol, where a network A sends a sequence of subscription interests to network B. Each subscription is for a subspace of an EWB space that B supports.

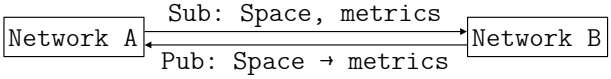

Fig. 2. Basic SFP message flow.

Network A can make multiple subscriptions, for example, with the first for the packet space to obtain reachability for a subset of IP destination addresses and then a set of flows that A plans to schedule. A then can send a sequence of flowset space subscriptions to understand the resource constraints for each flowset.

Example 1: A first sends a packet space subscription to B, and then B replies all packets that it can route:

```
A->B subscription:
  packet space: all packets
  metric: reachability
B-> reply:
  reachable subsets for each individual packet
```

As simple as Example 1 is, it shows that SFP can take BGP as a special case.

Example 2: A may be a organization with a set of flows that it can schedule. Then, we sends a flowset consisting of two flows, each defined by a standard OpenFlow 5-tuple:

```
A->B subscription:
  flowset space: A single flowset point consisting of two flows;
  metric: available bw
B->A reply: feasible region for the flowset
  (e.g., x1 <= 10; x2 <= 5; x1+x2 <= 7)
```

As simple as Example 2 is, it is provides resource constraints that are lacking in previous designs.

B. SFP Packet Space View

The preceding section defines the basic message flow. How the messages are computed and encoded are not specified. Although how messages are computed do not need to be standardized, message encoding need to be standardized for interoperability. In this section, we discuss both message computing and encoding for the packet space, which is the most fundamental space of the EWB.

Problem definition: We define the message computing and encoding problem of the packet space as the following compact policy program instantiation (CPPI) problem: given a program P which takes a point in packet space and returns a routing decision for that point's packets, and a queried subset of packet space, Q, what is the most compact instantiation of P's behavior over Q?

Such an instantiation can be used as a RIB to communicate an network's controller's program to other networks in response to queries. For this instantiation to be practical in production networks, however, it must also meet the following requirements.

- Correctness: P must be represented accurately.
- Updatability: Changes to P must be communicated without complete retransmission.
- Mergability: P's encoding must allow other networks to merge information into its RIB.
- Obfuscation: P's encoding should not reveal unqueried information about P, preserving P's privacy.

Solution architecture: The SFP packet space protocol attempts to solve the compact policy program instantiation problem while ensuring correctness, updatability, mergability and obfuscation by encoding P's behavior over subsets of packet space as pipeline table formatted RIBs. We present the protocol's architecture in Fig. 3.

Consider two networks, controlled by `Controller A` and `Controller B`, running SFP. Upon start up, SFP obtains its controller's routing programs P_A and P_B and compiles each program into a flow table pipeline, which is updated should its program or the data it was compiled with change.

Suppose that `A` sends `B` a packet space query. Upon receipt, `B` passes the query to its SFP app, which selects the subset of the query it is willing to respond to, selects the flow table rules in its pipeline queried by this subset, and formats these rules into a RIB. This RIB is then obfuscated to hide any information that it contains in excess of the query's scope, and passed back to `B`, which sends it out as a reply to `A`.

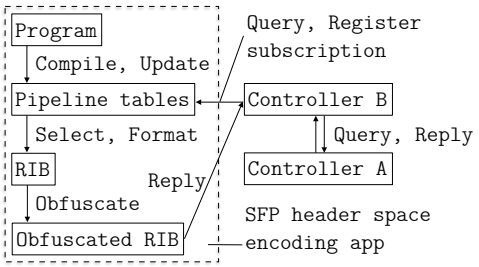

Fig. 3. packet space encoding architecture.

Having given an overview of SFP's packet space encoding architecture, we now examine each of the architecture's three stages in more detail.

Program compilation and update: Program compilation is the transformation of a generic program, expressed in a high level language, into a pipeline of flow tables. Programs are compiled into flow tables because flow tables allow efficient querying and require little additional processing to be transmitted efficiently. Compilation occurs "only-once", offline, avoiding the burden of per request program compilation and removing a potentially lengthy computation from the runtime environment.

We present the compiler's architecture in Fig. 4. First SFP compiles the program into a per instruction table (PIT) pipeline using two synergistic techniques: Symbolic map and Flow Explore. Subsequently, SFP compresses the PIT pipeline into a more compact representation via a third technique, Deep Expansion.

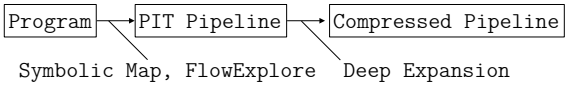

Fig. 4. Program to pipeline compiler architecture.

Per Instruction Table (PIT) pipelines: The key insight underlying the per instruction table (PIT) pipeline is that any program instruction I can be converted into a flow table which matches on I's argument's bindings and whose actions reflect I's execution on a given binding.

For example, consider the instruction `I-example: if (pkt.dstPort < 1024)`. We can transparently replace `I-example` with the table `I-example-PIT`.

```
I-example-PIT:
  pri  dstPort  Action
  ---  -------  ---------------
  1    1        goto if block
  1    ...      ...
  1    2014     goto if block
  1    2015     goto else block
  1    ...      ...

I-symbolic-mapped:
  pri dstPort      Action
  --- -----------  ---------------
  1   10000000000  goto if block
  2   1xxxxxxxxxx  goto else block
  3   xxxxxxxxxxx  goto if block
```

Symbolic map: While we can encode any I as a PIT by enumerating all the values in that I's domain, network programs often reference variables whose domain is too large to practically enumerate - an IPv6 address, for example, can

take 2^{128} distinct values. Fortunately, certain common types of Is have succinct PIT encodings. Such Is, specifically comparisons, conditionals, lookup tables, memory reads and switch function invocations (*e.g.* checksum) are referred to as symbolic mappable, and their encoding as symbolic mapping. `I-example`, for instance, can be symbolically mapped to the table `I-symbolic-mapped`.

The first step to converting generic programs written in high level languages to PIT pipelines is thus passing over the program, identifying symbolic mappable Is and symbolic mapping them.

Flow-Explore: To complete the transformation of a program into a PIT pipeline we must encode the remaining non-symbolic mappable instructions. Such encoding is accomplished by exploring all possible execution paths through the program and adding the results of encoding some I during exploration to that I's flow table, a process referred to as Flow-Explore.

Deep Expansion: While a PIT pipeline is a valid flow table representation of a program, it is often possible to compress it by merging PITs, conserving transmission bandwidth. Our compression transformation Deep Expansion uses heuristics such as shared header fields to identify and combine mergable PITs.

Selecting, formatting, and subscribing queried rules: Upon receiving a query, SFP selects the subset of the query it is willing to respond to Q and then selects the rules in its program's pipeline tables queried by Q. Such selection is accomplished simply by recursively exploring each path through the pipeline, marking each rule for transmission whose match fields correspond either to values within Q or to intermediate values set earlier in the path currently being explored, and terminating the exploration of any path whose rules' match fields correspond to neither value.

After exploration, the rules marked for transmission are used to construct a reduced pipeline which only matches on values within Q. This pipeline constitutes SFP's pipeline formatted RIB, and is conceptually ready for transfer through SFP's controller's EWB interface.

As an example, consider formulating a RIB in response to the query {`dstAddr = (130.91.6.0/30, 18.0.0.0/8), dstPort = x`} from the single table pipeline `example-pipeline`. Rules with their `dstAddr` in the `130.91.6.0/30` block and the default drop rule fall within the queried packet space and are selected for transmission, generating the RIB `example-RIB`.

`example-pipeline:`

dstAddr	dstPort	action
130.91.6.0/32	< 1024	out(1)
130.91.6.1/32	< 1024	out(2)
130.91.6.2/31	< 1024	out(1)
130.91.6.2/31	>= 1024	out(3)
130.91.6.4/30	< 1024	out(4)
130.99.6.8/30	< 1024	out(1)
x.x.x.x	x	drop

`example-RIB:`

dstAddr	dstPort	action
130.91.6.0/32	< 1024	out(1)
130.91.6.1/32	< 1024	out(2)
130.91.6.2/31	< 1024	out(1)
130.91.6.2/31	>= 1024	out(3)
x.x.x.x	x	drop

Rules transmitted to a controller are also marked as subscribed to that controller. If a subscribed rule is affected or eclipsed by an update to the controller's program or to the data that the program depends on, the updated rule or rules are transmitted to the subscribed controller.

Obfuscation: After encoding P_B in an RIB, SFP obfuscates the RIB, limiting the information that the RIB shares and guaranteeing B's privacy. In particular, obfuscation provides two privacy guarantees to controllers:

- Flow privacy: Controllers' network flows are hidden.
- Program privacy: Controllers' programs are hidden.

A naïve RIB containing a subset of a program's pipeline tables may reveal information about the program's structure and the flows it prescribes, necessitating obfuscation.

SFP obfuscates RIBs by transforming them with an obfuscation operator. One potential obfuscation operator, for example, is to first map each terminal action in an RIB's pipeline to a single bit label - `1` if the action transmits packets it applies to and `0` if the action drops them, and then reduce the pipeline by merging rules that map to the same label where possible using TCAM range encoding.

Mapping terminal RIB pipeline actions to generic transmit/drop labels hides the flows prescribed by the pipeline, helping achieve flow privacy. Reducing the pipeline by merging rules that map to the same generic label blurs the pipeline's decisions, helping achieve program privacy. Further investigation into mechanisms to provide different levels of privacy is ongoing.

As an example, consider the obfuscation of `example-RIB`:

`after-map:`

dstAddr	dstPort	label
130.91.6.0/32	< 1024	1
130.91.6.1/32	< 1024	1
130.91.6.2/31	< 1024	1
130.91.6.2/31	>= 1024	1
x.x.x.x	x	0

`after-reduce:`

dstAddr	dstPort	label
130.91.6.0/31	< 1024	1
130.91.6.2/31	x	1
x.x.x.x	x	0

First, `example-RIB` is converted to `after-map` by mapping each of its terminal actions to the labels `0` and `1`. Next, the `after-map` table is reduced to the `after-reduce` table by merging `after-map`'s first two pairs of rules, exploiting their shared label `1` and mergable match fields.

V. Future Research Directions

Although SFP allows more flexible routing than BGP, and strives to address several of its issues, a number of challenges still need to be addressed. Below we list key research directions to fully realize the benefits of SFP.

SFP in multi-hop settings: This paper presented the communication paradigm between two adjacent domains. Extending the exchange and negotiation of network flows across multiple domains introduces new challenges. For example, how can a controller merge its local information with the information received from its peers? What operations can be executed? What output and format can be sent by the node to its neighbors?

SFP for multiflow queries: We illustrated how a domain can subscribe a flow to its neighbor. Ultimately, a domain

should be able to subscribe to multiple concurrent flows. This capability would allow the domain to learn of the impact when sending multiple flows simultaneously. For example, the flows may share a physical link. As a result, the flows would not get the total of the advertised bandwidths, but a lower throughput when sent concurrently. Querying for multiple flows may also allow a domain to infer the risk of flows experiencing disruption in the event of a shared node or link failure. However, can a domain reply to a multiflow query and still preserve its internal sensitive information?

SFP correctness and stability analysis: We will identify desired properties of correctness, and ensure that the protocol satisfies them. For example, in the absence of changes to the network topology, it is commonly desirable for a routing protocol to quickly converge to a stable state devoid of routing anomalies (*e.g.*, black holes, forwarding loops) [18]–[21]. Routing protocols should not be vulnerable to permanent route oscillations [8], [22], [23].

SFP for fully integrated interconnection: As a result of failures, a domain may be partitioned into multiple components which can no longer directly communicate with each other. There may exist physical paths between the components through external domains. However, BGP does not support domain backup and prevents those physical paths from being visible to the components. As a result, the components may become disconnected. To support domain backup, ISPs often have to resort to ad-hoc solutions [14], [15]. We will investigate and compare means to support domain backup as part of the proposed protocol.

ACKNOWLEDGEMENT

This research was supported in part by NSF grant #1440745, CC*IIE Integration: Dynamically Optimizing Research Data Workflow with a Software Defined Science Network; Google Research Award, SDN Programming Using Just Minimal Abstractions; NSFC #61672385, FAST Magellan. This research was also sponsored by the U.S. Army Research Laboratory and the U.K. Ministry of Defence under Agreement Number W911NF-16-3-0001. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the U.S. Army Research Laboratory, the U.S. Government, the U.K. Ministry of Defence or the U.K. Government. The U.S. and U.K. Governments are authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation hereon.

REFERENCES

[1] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "OpenFlow: Enabling innovation in campus networks," *SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 2, pp. 69–74, Mar. 2008.

[2] P. Bosshart, D. Daly, G. Gibb, M. Izzard, N. McKeown, J. Rexford, C. Schlesinger, D. Talayco, A. Vahdat, G. Varghese, and D. Walker, "P4: Programming protocol-independent packet processors," *SIGCOMM Comput. Commun. Rev.*, vol. 44, no. 3, pp. 87–95, Jul. 2014. [Online]. Available: http://doi.acm.org/10.1145/2656877.2656890

[3] N. Foster, R. Harrison, M. J. Freedman, C. Monsanto, J. Rexford, A. Story, and D. Walker, "Frenetic: A network programming language," in *Proceedings of the 16th ACM SIGPLAN International Conference on Functional Programming*, ser. ICFP '11. New York, NY, USA: ACM, 2011, pp. 279–291. [Online]. Available: http://doi.acm.org/10.1145/2034773.2034812

[4] A. Voellmy, J. Wang, Y. R. Yang, B. Ford, and P. Hudak, "Maple: Simplifying SDN programming using algorithmic policies," in *Proceedings of the ACM SIGCOMM 2013 Conference on SIGCOMM*, ser. SIGCOMM '13. ACM, 2013, pp. 87–98. [Online]. Available: http://doi.acm.org/10.1145/2486001.2486030

[5] S. Jain, A. Kumar, S. Mandal, J. Ong, L. Poutievski, A. Singh, S. Venkata, J. Wanderer, J. Zhou, M. Zhu, J. Zolla, U. Hlzle, S. Stuart, and A. Vahdat, "B4: Experience with a globally-deployed software defined WAN," in *Proceedings of the ACM SIGCOMM 2013 Conference on SIGCOMM*, ser. SIGCOMM '13, 2013.

[6] M. Casado, M. J. Freedman, J. Pettit, J. Luo, N. McKeown, and S. Shenker, "Ethane: Taking control of the enterprise," *SIGCOMM Comput. Commun. Rev.*, 2007.

[7] A. Gupta, L. Vanbever, M. Shahbaz, S. P. D. B. Schlinker, N. Feamster, J. Rexford, S. Shenker, R. Clark, and E. Katz-Bassett, "SDX: A Software Defined Internet Exchange," in *Proceedings og SIGCOMM 2014*. IEEE, August 2014, pp. 233–239.

[8] R. Mahajan, D. Wetherall, and T. Anderson, "Towards coordinated interdomain traffic engineering," in *Proceedings of Third Workshop on Hot Topics in Networks (HotNets-III)*, 2004.

[9] "Inter SDN Controller Communication (SDNi)," http://events.linuxfoundation.org/sites/events/files/slides/ODL-SDNi_0.pdf.

[10] K. Phemius, M. Bouet, and J. Leguay, "DISCO: Distributed multidomain SDN controllers," in *Proceedings of IEEE Network Operations and Management Symposium (NOMS)*, 2014.

[11] P. Lin, J. Bi, and Y. Wang, "East-West Bridge for SDN Network Peering," in *Proceedings of ICoC 2013*, 2013.

[12] "BGP based SDN http://network-insight.net/2015/09/bgp-based-sdn/," 2015.

[13] F. Benamrane, M. B. mamoun, and R. Benaini, "An East-West interface for distributed SDN control plane: Implementation and evaluation," *Computers & Electrical Engineering*, vol. 57, p. 162175, January 2017.

[14] F. Le, G. G. Xie, D. Pei, J. Wang, and H. Zhang, "Shedding light on the glue logic of the Internet routing architecture," in *ACM SIGCOMM Computer Communication Review*, 2008.

[15] Cisco, "OSPF Redistribution Among Different OSPF Processes," http://www.cisco.com/c/en/us/support/docs/ip/open-shortest-path-first-ospf/4170-ospfprocesses.html, 2016.

[16] I. Bueno, J. I. Aznar, E. Escalona, J. Ferrer, and J. A. Garcia-Espin, "An OpenNaaS Based SDN Framework for Dynamic QoS Control," in *Proceedings of IEEE Conference on SDN for Future Networks and Services (SDN4FNS)*, 2013.

[17] G. Petropoulos, F. Sardis, S. Spirou, and T. Mahmoodi, "Software-defined inter-networking: Enabling coordinated QoS control across the Internet," in *Proceedings of ICT 2016*, 2016.

[18] T. Griffin and G. Wilfong, "An analysis of BGP convergence properties," *ACM SIGCOMM Computer Communication Review*, 1999.

[19] J. L. Sobrinho, "An algebraic theory of dynamic network routing," *IEEE/ACM Transactions on Networking (TON)*, 2005.

[20] P. Francois, C. Filsfils, J. Evans, and O. Bonaventure, "Achieving subsecond IGP convergence in large IP networks," *SIGCOMM Comput. Commun. Rev.*, 2005.

[21] F. Le, G. Xie, and H. Zhang, "Theory and new primitives for safely connecting routing protocol instances," *ACM SIGCOMM Computer Communication Review*, 2010.

[22] T. Griffin, F. B. Shepherd, and G. Wilfong, "The stable paths problem and interdomain routing," *IEEE/ACM Transactions on Networking (ToN)*, 2002.

[23] L. Gao and J. Rexford, "Stable Internet routing without global coordination," *IEEE/ACM Transactions on Networking (TON)*, 2001.