Software-Defined Internet Architecture:
Decoupling Architecture from Infrastructure

Barath Raghavan
ICSI

Teemu Koponen
Nicira

Ali Ghodsi
UC Berkeley

Martín Casado
Nicira

Sylvia Ratnasamy
UC Berkeley

Scott Shenker
ICSI / UC Berkeley

ABSTRACT

In current networks, a domain can effectively run a network architecture only if it is explicitly supported by the network infrastructure. This coupling between architecture and infrastructure means that any significant architectural change involves sizable costs for vendors (for development) and network operators (for deployment), creating a significant barrier to architectural evolution.

In this paper we advocate decoupling architecture from infrastructure by leveraging the recent advances in SDN, the re-emergence of software forwarding, and MPLS's distinction between network's core and edge. We sketch our design, called Software-Defined Internet Architecture (SDIA), and show how it would ease the adoption of various new Internet architectures and blur the distinction between architectures and services.

1 Introduction

The goal of this paper is simple: to change architectural evolution from a hardware problem into a software one. And our solution is rather standard, borrowing heavily from long-standing (*e.g.,* MPLS) and emerging (*e.g.,* SDN) deployment practices. However, to provide the necessary context, we start by asking two questions: what is the problem and why are current efforts insufficient?

What is the problem? Despite the Internet's unparalleled success, it has long been known that the Internet architecture has significant deficiencies. Attempts to address these architectural issues with incrementally deployable modifications have had limited success, so the research community has turned to exploring "clean-slate" designs to provide a deeper intellectual understanding of the problem. Unsurprisingly, none of these clean-slate architectures have been deployed.

To bridge this gap between what is architecturally desirable and what is feasibly deployable, there is a small but growing literature on *architectural evolvability* [3, 8, 11, 14]. The resulting proposals include, among other ideas, measures to: (i) automatically cope with partial deployment (such as the fallback behaviors in XIA [11]) and (ii) structure the network stack with far more modularity (as discussed in [8, 14]) so that aspects of a particular architecture are not embedded within applications. These would greatly increase our ability to adopt major architectural improvements through incrementally deployable design changes.[1]

Why are these measures insufficient? Because even after these advances, the *architecture* remains coupled to the *infrastructure*. To make this clear, we first define these terms more precisely:

- **Architecture:** This refers to the current IP protocol or, more generally, any globally agreed upon convention that dictates how packets are handled.
- **Infrastructure:** This refers to the physical equipment used to build networks (routers, switches, fiber, cables, etc.). Of particular note are the forwarding ASICs, which take years to develop and are tailored to the specific forwarding semantics of the IP architecture.

When we say that the architecture is coupled to the infrastructure, we mean that any significant change to IP (or its successors) would require replacing (or overhauling) the routers used to build the network, because the forwarding ASICs only have limited flexibility.[2] While the proposals for architectural evolution do away with many other barriers to deployment (*e.g.,* barriers such as having IP addresses

[1]The adoption of one of the clean-slate "evolvable" architectures would enable subsequent incremental deployment of architectural changes. However, the initial deployment of one of these architectures would not be incremental.

[2]There are clearly some changes that can be implemented by changing router software, but new generations of chips were required to support IPv6 at linespeed and we expect the same would be required for many of the new clean-slate architectures in the literature, such as those being explored in NSF's FIA.

Hotnets '12, October 29–30, 2012, Seattle, WA, USA.

embedded in both applications and interdomain routing), they do not alter the basic fact that significantly changing how packets are handled will require a new infrastructure that has the appropriate hardware and software.

OpenFlow improves the situation, but only to a limited degree. To support a wide range of architectures, forwarding elements (routers, switches, middleboxes, etc.) must be prepared to support fully general packet matching and forwarding actions, and there is little chance that OpenFlow will ever support such generality because it would entail substantially higher costs, particularly in terms of the header size over which arbitrary matching could be done. It is therefore unlikely that such generality would be the standardized norm for all deployed routers.[3]

Thus, we must find other ways of decoupling the architecture from the infrastructure. This is important because it would greatly reduce the costs and inconvenience of deploying a new architecture and thereby enable architectural evolution to happen far more freely.[4] We believe that the tools for such a decoupling are already largely in hand, they merely need to be applied more systematically, and the rest of this paper is devoted to articulating that case.

2 Our Approach

We base our approach on current practice, so we start by noting that modern network infrastructure no longer resembles the "classic" Internet design. Among the deviations from the classic architecture are the following:

- **MPLS** is widely used within domains for VPNs and traffic engineering. Many routers never look at an IP address, and instead rely solely on MPLS labels for forwarding. For our purposes, the key attribute of MPLS is that it distinguishes between the network edge and the network core, and decouples the core from whatever global architecture is deployed.
- **Software-Defined Networking (SDN)** is an emerging technology that is used in several datacenters and at least one WAN [12]. SDN effectively separates the control plane from the data plane, much like earlier efforts such as RCP [4] and 4D [10], and provides a programmatic interface so that new control plane functionality requires only writing code for the network "controllers" (the servers that run the control plane for the network).
- **Middleboxes**: Many, if not most, packets are processed by one or more middleboxes that perform tasks beyond mere IP forwarding. Middleboxes are roughly as numerous as routers in most enterprise networks [17], and represent the way most new network functionality is deployed today (*e.g.,* firewalls, WAN optimizers).
- **Software Forwarding**, by which we mean packet forwarding using commodity processors such as x86s and GPUs, is increasingly common. To be sure, forwarding ASICs still have the overall price/performance edge at the high end, but current deployment practices — where software forwarding is the norm in hypervisors and in most middleboxes — are tangible proof that software forwarding is a viable choice in many settings.[5]

While these advances are well known, what has not been adequately recognized is that when they are applied systematically and in concert they decouple architecture from infrastructure. That is, these developments lead us to a design where the data plane consists of a network *core*, which uses its own internal addressing scheme (much like L2 networks do today), and a network *edge* that uses software-forwarding. Moreover, all architectural dependencies reside at the edge, which can be easily modified because packet forwarding is done in software there. The control plane of this architecture uses SDN to control the edge routers (to specify their forwarding actions), and leaves the design of the core control plane up to each domain. Note, however, in our version of SDN we do not restrict ourselves to the OpenFlow model of control, but instead allow the controller to install arbitrary forwarding code (either through a VM or x86 code) on software routers.

We call this design the *Software-Defined Internet Architecture* (SDIA), which we describe more fully below.

3 Software-Defined Internet Architecture

To motivate our design, we first provide a top-down perspective on this problem.

3.1 Top-Down Perspective

Networks are currently specified in what might be called a "bottom-up" fashion, with an emphasis on standardizing the behavior of each individual router. In the forwarding plane, low-level standards dictate the packet formats and forwarding behavior in excruciating detail. This bottom-up approach was necessary to ensure that any two routers that happen to be placed in the same network could interoperate. However, when the behavior of these individual routers is managed externally through SDN, then one need not specify the behavior at such a fine granularity. One does need OpenFlow (or an equivalent protocol) to specify how the controller manages router behavior, and this should be

[3] Long before OpenFlow, Active Networking [20] proposed fully general forwarding behavior, but has never been demonstrated as feasible at today's forwarding speeds.

[4] We assume that the other impediments to architectural evolution, such as architectures embedded in applications, have been taken care of by proposals such as in [8]. And, as noted there, those changes are also not altogether different from what is deployed today in various places, but merely applied consistently and comprehensively.

[5] In this paper we only consider the two extremes, fully custom ASICs and fully commodity processors. Network processors occupy the spectrum in between these two extreme points. However, we believe network processors are neither ascendant (they are widely seen as difficult to program) nor antithetical to our story. As long as network processors can support general capabilities, then they qualify as software forwarding.

standardized across all vendors, but one need not specify the forwarding behavior of each box beforehand because as long as two routers talk to the same controller they can be made to interoperate by the controller.

This allows us to take a top-down perspective, by which we mean that we focus not on what each box does individually but instead first look at how to decompose Internet service into well-defined tasks, and then consider how to implement those tasks in a modular fashion. For instance, consider the case where host X in domain A is trying to reach host Y in domain B. Then, the job of providing connectivity between X and Y can be supported by the following four tasks:

- **Interdomain task**: The highest-level task is for the domains to carry the packet from domain A to domain B, which may require traversing one or more intermediate domains.
- **Intradomain transit task**: To support the Interdomain task, domains must carry packets from the ingress peering domain to the egress peering domain. Thus, domains must be able to transit packets from one edge to another.
- **Intradomain delivery tasks**: To provide full end-to-end delivery of packets, domain A must carry the packet from host X to the edge of domain A (where the interdomain task takes over), and domain B must carry the packet from the edge of B to host Y. In addition, a domain must be capable of delivering packets between any two hosts in its domain.

In the rest of the paper we discuss how to implement these tasks in a modular fashion, so that the implementation of one task is not dependent on how another is implemented.

3.2 Interdomain Task

The interdomain task is to compute interdomain routes and then provide forwarding instructions to each domain in the form of domain-level forwarding entries such as:

Packets from peer domain A destined for domain B
$\downarrow$
Forward to peer domain C

To implement this in a modular way, we first borrow from the literature where various proposals (such as [2]) suggest a strict separation between intradomain and interdomain addressing.[6] We propose that the interdomain addressing uses some form of domain identifiers, and then the entire interdomain task is implemented with respect to these identifiers without reference to any intradomain addresses (as we mention below, each domain can choose its own internal addressing scheme).

[6]This is the only "clean-slate" aspect of our design, but there are several ways one could accomplish the same separation in a "dirty-slate" manner (such as using the IPv6 flow ID as the interdomain identifier).

We then propose that each domain be represented by a single logical server in the algorithm used to compute interdomain routes (*i.e.,* the server itself might be replicated for availability, but there is only one logical entity representing the domain when executing the interdomain routing algorithm); this is merely the extreme logical extension of route reflectors. This route computation could be as simple as running BGP among these logical servers, or it could involve an entirely new route computation algorithm.[7]

3.3 Intradomain Tasks

There are three intradomain tasks: edge-to-edge transit, edge-to-host delivery, and host-to-host delivery. Modularity requires that these are implemented in a way that is independent of how the interdomain task is implemented, but also that different domains can adopt different implementations of these intradomain tasks. To this end, we propose a "fabric-like" design as advocated in [5], which in turn was inspired by MPLS. The network is separated into an edge and a core. The core can use any internal forwarding and control plane mechanisms it chooses, ranging from SDN to today's intradomain routing protocols, as long as they support edge-to-edge, edge-to-host, and host-to-host delivery; in particular, each domain's core can use their own internal addressing scheme. The edge uses software forwarding with commodity processors managed by an SDN controller (the edge-controller), which understands the core management well enough to insert the appropriate packet headers to achieve internal or edge delivery.

The resulting design is highly modular. Only the edge routers need to understand interdomain addressing (by understand we mean have flow entries written in terms of interdomain addresses), and only core routers need to understand intradomain addressing (or they could merely understand MPLS-like labels, and the control plane would translate intradomain addresses to labels at the edge of the network). Moreover, the core routers only need to understand the intradomain addressing scheme in their domain, so domains can independently change their internal addressing scheme. Similarly, only the edge-controller needs to participate in the interdomain route computation, and only the core control plane needs to determine the internal routes (both routes between edges, and routes to and between internal hosts).

The key aspect of this design is that the only components needing to forward packets based on interdomain addresses are edge routers, which use software forwarding. As we explain later, this gives the design great architectural freedom, but here we first address a key point about whether it is reasonable to assume software forwarding at the edge.

Our own measurements of an unoptimized implementation using the Intel Data Plane Development Kit, show that longest-prefix match forwarding on minimum-sized packets,

[7]We are not, at this point, discussing scalability, but that would certainly be a consideration for any proposed algorithm.

including checksum verification and TTL adjustment, can be done at 6.7Gbps on a single 3.3Ghz core; this is a bit slower than reported in [16]. This is an optimistic estimate, since it does not include more complicated functions such as encapsulation or decapsulation, but it is also pessimistic by considering minimum-sized packets. In addition, the Routebricks work [7] suggests that one can gang many PCs together to form a forwarding engine with a high aggregate throughput with roughly linear scaling but at the cost of doubling the number of cores (due to Valiant load balancing). Is this enough to make software forwarding feasible at domain edges? The answer to this question depends, of course, on the amount of peering traffic a domain handles, and there is little data on this. Some informal estimates [1, 6, 15] suggest that 10Tbps is a reasonable order-of-magnitude estimate for peering bandwidth for a large domain; thus, an entire large domain's peering bandwidth could be handled by 1500 cores. Even if these estimates are off by a factor of two or more, they indicate that software forwarding at the edge should be easily within a domain's reach.

4 Interdomain Service Models

As mentioned previously, the only components involved in the interdomain task are the edge controllers (one per domain) and the edge routers (whose knowledge of the interdomain task comes only in the form of the forwarding actions received from the edge controller). The implications for architectural evolvability are profound.

- Changing how interdomain routing works, say changing from BGP to some novel interdomain routing scheme, would only involve changing software in the edge controllers to participate in this new distributed interdomain routing protocol. Of course, nothing in our design makes it easier to achieve agreement on a new routing scheme; only that once agreed to, deploying such a new design on an SDIA infrastructure is relatively simple.
- Changing how domains are addressed again only requires a change to the controller software. Because the edge routers use software forwarding, matching against different address structures is trivial; in fact, the software of the edge routers requires no change if the software was prepared to support general matching.
- Changing how hosts are addressed, say changing from IP to IPv6, is done on a domain-by-domain basis. A domain might have to buy new infrastructure to do so, but this is its own internal decision.

But changing to new routing schemes or addressing structures is only a small step forward, architecturally. Stealing a term from [8], one can consider different Interdomain Service Models (ISMs) which are delivery services agreed upon by all domains. As described in [8] it is easy to design host networking stacks that can support multiple of these ISMs in parallel, so the conceptual challenge is how to deploy new ISMs not how to use them. To define a new ISM in SDIA requires:

- A distributed interdomain algorithm among the edge-controllers that computes whatever state the controllers need to implement the service model; BGP is an example of such a distributed algorithm. Note that this distributed algorithm must take into account whatever policies these domains have about participating in this ISM (*e.g.,* who they are willing to peer with, how they are willing to interact, etc.).
- A set of forwarding actions that can be sent to these edge routers by the edge-controllers. Recall that we are assuming that these controllers can install arbitrary code in the software forwarding engine, either in the form of a VM or as x86 code.
- Measures to cope with partial deployment. This requires the existence of a basic unicast packet-delivery ISM (such as supplied by IP and BGP), so that non-peering domains can set up tunnels with each other. In addition, there must be some discovery mechanism so that domains participating in an ISM are aware of each other; this could fall out directly from the distributed algorithm (as it would in BGP), or be a separate mechanism.

Note that we did not include "supporting the ISM within the domain" on this list. This is certainly something that can be done, but need not happen before the ISM is initially deployed. One can think of the internal domain network much like an L2 network today; we propose deploying new features in IP or above (such as new congestion control algorithms, or CCN [13]) without proposing that all L2 networks change. Instead, we assume that L2 networks merely provide overprovisioned connections between the relevant L3 routers and are thus transparent. Similarly, we assume that initially new ISMs can be deployed with the edge-controller (and servers acting on its behalf) acting as the ISM nodes within the domain, and all other hops are merely providing overprovisioned paths to these servers.

Later, of course, it may behoove a domain to deploy core routers with support for a new ISM. For instance, supporting an Information-Centric Networking (ICN) design may require a "caching" action, and a match against the current cache table. But in the beginning ICN designs can be supported by having caching supported only at the edge-controller and servers acting on its behalf.

ISMs can range from low-level delivery models to high-level services. We list a few possibilities below:

- **Low-level ISMs:** best effort unicast delivery with different addressing or handling (such as in IPv6 or AIP [2]) or different interdomain routing (such as HLP [19] or Pathlets [9]), or interdomain multicast.
- **Higher-level ISMs:** HTTP-peering could be an ISM, as could various ICN designs (*e.g.,* CCN [13]). Going

further, one could imagine deploying social networking or recommender systems as ISMs.

The notion of an ISM blurs the distinction between low-level networking and high-level application services. An ISM is merely a service agreed-upon by the domains, and the notion of being able to define a new ISM with merely software deployed on each domain's edge-controller (which then installs software on the edge router) could make it easier for the domains to support new services.

When considering the limitations of the SDIA approach, there are two key constraints to consider. The first is the degree of processing each packet requires. Clearly designs that require extensive per-packet computation are infeasible within SDIA at high traffic volumes. The second is whether per-hop processing is required (such as in XCP or other router-assisted congestion control schemes). However, as noted previously, using SDIA to deploy such designs is similar in spirit to how we deploy IP over overprovisioned L2 networks.

5 Three Detailed Examples

We now provide a more detailed demonstration of the power of SDIA to deploy new services purely through software. We present three ISM designs, selected for their diversity. First we present a straightforward example: policy-compliant interdomain source routing for best-effort packet delivery. We then consider an ICN-style ISM that operates at the content level. We end with a third ISM that allows end users to invoke middlebox services; this ISM is not a packet delivery service, but instead coordinates existing packet-handling services.

There are some basic building blocks that are common to most ISM deployments. In each domain, an ISM implementation (which, recall, is merely code for the controller) has access to an up-to-date database of peering links and the neighboring domain for each such link. Similarly, each ISM implementation can interact with the mechanisms used in intradomain delivery, so the ISM knows which IP-like addresses (or MPLS-like labels) to use to reach either specific egresses or internal hosts. Also, recall that ISM implementations have two parts: the interdomain component which is a distributed algorithm among controllers, which we will call the Interdomain Distributed Algorithm (IDA), and the internal component which manages the edge switches, which we do not discuss below because in each case it is relatively straightforward.

5.1 Interdomain Packet Delivery via Pathlets

Here we show how a unicast best-effort packet service based upon pathlets [9] can be supported in SDIA. The pathlet IDA must distribute pathlet information to hosts. This can be done in many ways, but here we describe two. The simplest is merely for each domain to advertise the pathlets it supports to several large Internet services which host pathlet repositories (*e.g.,* Google, Yahoo, etc.) The IDA then is nothing more than agreeing on a few such repositories, and sending pathlet information there. Thus, implementing an ISM need not be done solely by the participating controllers, but can invoke external parties, or even be housed on centralized servers.

A more resilient form of IDA is for the controllers to gossip the pathlet information (as described in [9]). This guarantees pathlet information will eventually be delivered to a host as long as there is any path between the host and the domain supporting that pathlet, but involves a more complicated algorithm than relying on major Internet sites.

Once the pathlet information is available to hosts, they start sending packets whose headers contain a list of pathlets to use. When a transit packet arrives at a domain, the edge router must recognize the pathlet, figure out which egress port is appropriate for that pathlet, and then insert a packet header that allows the intradomain transit service to deliver the packet. This information can be supplied by the edge-controller, which in turn gets the information from the intradomain infrastructure. The same applies for when a packet arrives at the end of its path (and pathlet list), and must be delivered to a host.

5.2 An Information-Centric ISM

There has been much recent interest in information-centric network architectures (known as ICN designs). The goal of an ICN design is to allow clients to ask for content by name, and have the infrastructure deliver a nearby copy if available. Thus, an ICN ISM essentially involves implementing content routing on an interdomain basis. The scalability of this approach depends in part on the naming used for objects (*i.e.,* whether or not the naming system enables a degree of aggregation), and there is some debate on this point within the ICN community. However, SDIA would be capable of supporting any such naming scheme, since the IDA is little more than running a name-based routing algorithm.

When an object request arrives at a domain, the edge router inspects the object being requested (which is part of the packet header) and looks into the name-based routing table that has been passed to it by the edge-controller. If the object is in that domain's cache, the edge router forwards the packet to the controller (or to a server acting on the controller's behalf). If the object is not in that domain's cache, then the edge router's routing table indicates the next-hop domain for the request.

This design initially treats the controller, and various servers it controls, as a domain-level cache. A domain can deploy additional servers, spreading them throughout the domain, so that the forwarding entry at an edge router points to the closest copy within a domain. Essentially the domain would be running its own internal centrally-controlled CDN. However, if a domain wanted to support ICN more natively, it could eventually deploy ICN-enabled routers.

The point is that the ICN-like service model would be available to hosts without additional internal support; they don't care how the service is implemented, only that it

delivers reasonably good service, and our experience with Akamai suggests that such a CDN-like approach would have satisfactory performance.

5.3 A Middlebox-Services ISM

As noted earlier, much new network functionality is deployed via middleboxes, and we do not expect this to change (even with SDIA deployed). Middlebox deployment and management is rather ad hoc, and certainly resides outside the architecture; in particular, hosts cannot request middlebox services from the network. There have been some recent designs, most notably [18], which provide an interface via which hosts can request various kinds of middlebox services. We now describe how this could be accomplished within SDIA, assuming that there is another ISM that supports basic packet delivery.

The IDA portion of the Middlebox-Services ISM merely involves the relaying of information about what middleboxes are available in each domain. The approach described in [18] can easily be supported through new controller software. The other portion of this ISM involves providing an interface for hosts, and this can be supported by the domain's edge controller. Once the IDA has determined which middlebox services will be provided, and where, it then has enough information to route the packets accordingly using the basic packet-delivery ISM.

6 Discussion

SDIA takes the ideas of SDN, MPLS, and software forwarding and applies them in a coherent and systematic manner. The result is a modular and flexible design where new interdomain service models can be implemented in software, but ASICs still do the bulk of the forwarding. These ISMs can range from low-level packet delivery (like IP) to higher-level services (more like HTTP). Initially these new ISMs require no special support within a domain; over time, as their usage grows, a domain may choose to deploy more specialized support.

This is radically different from the status quo. Today, router vendors must invest in ASIC support for new features *before* they can be widely used; in short, they take all the risk up-front. Moreover, this risk is high because domains face significant deployment costs in adopting a new design. In contrast, with SDIA, new infrastructure (to provide in-domain support) only occurs (if at all) *after* the ISM is widely used (so vendors can safely invest in development).

The design process can also be quite different from today. On purely technical grounds, software enables different tradeoffs. One could imagine very large interdomain packet headers, including self-certifying identifiers, whereas hardware typically favors smaller, fixed-size headers. Finally, since the core implementation is in software, standardization could proceed via open-source efforts rather than through large standards bodies. One could imagine networks being as nimble as Linux, rather than as lumbering as BGP.

7 References

[1] Akamai. Facts & Figures. `http://www.akamai.com/html/about/facts_figures.html`.

[2] D. G. Andersen, H. Balakrishnan, N. Feamster, T. Koponen, D. Moon, and S. Shenker. Accountable Internet Protocol (AIP). In *Proc. of SIGCOMM*, 2008.

[3] T. Anderson, K. Birman, R. Broberg, M. Caesar, D. Comer, C. Cotton, M. Freedman, A. Haeberlen, Z. Ives, A. Krishnamurthy, W. Lehr, B. T. Loo, D. MaziÃĺres, A. Nicolosi, J. Smith, I. Stoica, R. van Renesse, M. Walfish, H. Weatherspoon, and C. Yoo. NEBULA - A Future Internet That Supports Trustworthy Cloud Computing. `http://nebula.cis.upenn.edu/NEBULA-WP.pdf`.

[4] M. Caesar, D. Caldwell, N. Feamster, J. Rexford, A. Shaikh, and K. van der Merwe. Design and Implementation of a Routing Control Platform. In *Proc. of NSDI*, 2005.

[5] M. Casado, T. Koponen, S. Shenker, and A. Tootoonchian. Fabric: A Retrospective on Evolving SDN. In *Proc. of HotSDN*, August 2012.

[6] 10 Facts About Peering, Comcast and Level 3, November 2010. `http://blog.comcast.com/`.

[7] M. Dobrescu, N. Egi, K. Argyraki, B.-G. Chun, K. Fall, G. Iannaccone, A. Knies, M. Manesh, and S. Ratnasamy. RouteBricks: Exploiting Parallelism to Scale Software Routers. In *Proceedings of SOSP*, 2009.

[8] A. Ghodsi, T. Koponen, B. Raghavan, S. Shenker, A. Singla, and J. Wilcox. Intelligent Design Enables Architectural Evolution. In *Proc. of Hotnets-X*, 2011.

[9] P. B. Godfrey, I. Ganichev, S. Shenker, and I. Stoica. Pathlet Routing. In *Proc. of SIGCOMM*, 2009.

[10] A. Greenberg, G. Hjalmtysson, D. A. Maltz, A. Myers, J. Rexford, G. Xie, H. Yan, J. Zhan, and H. Zhang. A Clean Slate 4D Approach to Network Control and Management. *SIGCOMM CCR*, 35(5), 2005.

[11] D. Han, A. Anand, F. Dogar, B. Li, H. Lim, M. Machado, A. Mukundan, W. Wu, A. Akella, D. G. Andersen, J. W. Byers, S. Seshan, and P. Steenkiste. XIA: Efficient Support for Evolvable Internetworking. In *Proc. of NSDI*, 2012.

[12] U. Hölzle. OpenFlow @ Google. *ONS*, 2012.

[13] V. Jacobson et al. Networking Named Content. In *Proc. of CoNEXT*, 2009.

[14] T. Koponen, S. Shenker, S. Shenker, H. Balakrishnan, N. Feamster, I. Ganichev, A. Ghodsi, P. B. Godfrey, N. McKeown, G. Parulkar, B. Raghavan, J. Rexford, S. Arianfar, and D. Kuptsov. Architecting for Innovation. *SIGCOMM CCR*, 41(3), 2011.

[15] NTT. About Our Network. `http://www.us.ntt.net/about/ipmap.cfm`.

[16] L. Rizzo. Netmap: a Novel Framework for Fast Packet I/O. In *Proc. of USENIX ATC*, 2012.

[17] J. Sherry, S. Hasan, C. Scott, A. Krishnamurthy, S. Ratnasamy, and V. Sekar. Making Middleboxes Someone Else's Problem: Network Processing as a Cloud Service. In *Proc. of SIGCOMM*, 2012.

[18] J. Sherry, D. C. Kim, S. S. Mahalingam, A. Tang, S. Wang, and S. Ratnasamy. Netcalls: End Host Function Calls to Network Traffic Processing Services. Technical Report EECS-2012-175, UCB, 2012.

[19] L. Subramanian, M. Caesar, C. T. Ee, M. Handley, M. Mao, S. Shenker, and I. Stoica. HLP: a Next Generation Inter-domain Routing Protocol. In *Proc. of SIGCOMM*, 2005.

[20] D. L. Tennenhouse and D. J. Wetherall. Towards an Active Network Architecture. In *Proceedings of DANCE*, 2002.