Extending SDN to Large-Scale Networks

James McCauley[♮‡], Aurojit Panda[♮], Martin Casado[◇], Teemu Koponen[◇], Scott Shenker[♮‡]
[♮]UC Berkeley, [‡] ICSI, [◇]VMWare Inc.

1 Introduction

When networks were first developed, the control plane merely needed to provide end-to-end connectivity, which is something that existing distributed routing protocols do reasonably well. However, current networks are expected to provide far more than just simple connectivity — the control plane must meet an increasing set of new demands including traffic engineering, access control, isolation, and virtualization. One could build new control plane mechanisms for each of these new requirements, but *Software-Defined Networking (SDN)* provides the opportunity to extend the capabilities of the control plane in a more principled and manageable manner. There has been substantial research on how to build SDN control planes (see, *e.g.*, [1, 3–9]), but these efforts have primarily focused on networks of limited scope such as datacenters and enterprises. What has not been widely discussed in the academic literature is how to design the SDN control plane for a large and globally distributed network.

Here we present a vision for such a control plane. We incorporate hierarchy into SDN by introducing a recursive building block — the *Logical xBar*. Our design combines the ability to scale to global networks with the locality of control required to allow for portions of the network to continue functioning even if remote sites (or links to them) have failed. Further, it provides reasonable convergence behavior and transactional semantics for configuration changes, and is sufficiently general to provide a range of functionality, including routing, access control, and traffic engineering. Our design provides a clean separation of concerns through which the nature of the hierarchy, the replication of computation, the physical topology, and the implementation of forwarding behaviors are largely orthogonal. We have simulated our hierarchical design on synthetic AS and internetwork topologies of up to 10000 switches, and have demonstrated sub-second computation times.

2 Building Blocks

It is well known that the key to scaling network systems is aggregating smaller units for forwarding into larger ones. For example, while we typically think of physical switches or routers as the smallest unit of network functionality, they are actually a seamless composition of individual linecards. Similarly, network fabrics [2] seamlessly join individual switches into a single, large unit of forwarding. OSPF/IS-IS areas, ATM's PNNI, and the Internet itself all similarly make use of aggregation for scalability and manageability. In our architecture, we reuse this same successful methodology within the context of SDN. Specifically, we introduce the following two building blocks:

Logical xBar. The *Logical xBar* is a programmable entity that can switch packets between ports. Thus, a "first order" xBar could merely be an OpenFlow switch. Beyond this, however, xBars can be composed into larger xBars: with suitable configuration of their individual forwarding tables, multiple smaller xBars can be used to implement a single larger xBar with a single larger forwarding table. This process is explicitly recursive, allowing for xBars to be stacked to any desired depth.

Logical Server. In a physical router or switch, forwarding table management and control plane computations are carried out in a management CPU. For logical xBars, these computations are handled by *Logical Servers*. Architecturally, logical servers appear as a single machine (that is, they are logically centralized), but in practice they can be replicated for fault tolerance.

2.1 Hierarchy

The recursive aggregation of logical xBars and their logical servers naturally imposes a hierarchy on the network. However, we do not assume that the network topology itself is explicitly hierarchical; rather, we assume that the network displays the kind of *locality* found in almost all engineered networks, in which links are more likely to exist between nearby network nodes than remote ones and administrative control applies to a localized set of links. Indeed, rather than arbitrary clusterings of switches, we expect the edges of most logical xBars to fall on natural physical or administrative boundaries, *e.g.*, PoP-based xBars aggregated into regional network xBars aggregated into continental network xBars, or individual LAN xBars aggregated into building xBars aggregated into campus xBars.

2.2 Computation

Each xBar holds two types of information. The first of these is *state* information about the conditions of the underlying network; this can be thought of as a generalized form of topology. State information originates at the bottom of the hierarchy and is passed upwards from child to parent, possibly being summarized or filtered along the way. The second type of information is

configuration information that defines how an xBar should behave; think of this as a generalized form of a forwarding table. Each logical server, given a configuration and state for its associated logical xBar, must compute the configurations for each of the associated child xBars. This computation is simply determining how to implement the given forwarding table on top of the given topology. It is this *table on topology* (ToT) computation that recurses at all levels of the hierarchy. The upwards propagation of summarized state and the downwards propagation of configuration information (after the ToT computation), taken together, implement the control plane.

As each logical server computes only the portions of the total network configuration relevant to its direct children, the computation is inherently distributed. Furthermore, the code for this ToT computation can be reused at all levels of the hierarchy, allowing policies to be applied at different levels of granularity (*e.g.*, the same ACL code can implement ACLs on individual switches or on logical xBars corresponding to entire datacenters).

The separation of state and configuration and the unidirectional nature of information transmissions (state goes up, configuration goes down) makes it easier to reason about the computation. Moreover, it avoids the iterative algorithms used in some routing protocols that can lead to long convergence times; convergence time in our system is a simple function of the depth of the hierarchy.

2.3 Labeling

Our implementation of logical xBars uses packet labeling as the primary data plane mechanism. The benefits of packet labeling have already been demonstrated by MPLS, and prior work has proposed that SDN in general should adopt a similar strategy (see [2]). As in MPLS, our design uses labels to decouple external protocols from internal forwarding mechanism. Additionally, labeling allows our system to provide *consistency* — a packet traversing the network is always forwarded based on a consistent set of forwarding rules. This is achieved by associating each new set of forwarding rules with a new set of labels. Only after the new labels have been installed within an xBar do we begin applying these new labels to packets at the edge of the xBar. Resultingly, a packet in unambiguously processed using either the new set of forwarding rules or the old ones.

3 Functionality and Locality

A design for wide area SDN networks is only as useful as the functionality that design supports, and ours supports a range of interesting features: to date, we have explored routing, access control and traffic engineering. It is also notable that these individual functionalities are implemented almost entirely independently of each other. In part, this is because their concerns are largely orthogonal and easily layered — they need only agree on a common labeling scheme.

In addition, our approach enables locality of control because each functionality is executed at the appropriate level of the hierarchy. Access control within a PoP is implemented by the logical xBar at that level (rather than by the top level xBar), and responses to local network failures are handled at the lowest level possible. For instance, if a regional network can recover from a failure and still provide connectivity between its external ports, then the failure information need not travel higher up in the control hierarchy. This systematic locality of control enables this approach to scale to global networks, because only rarely does the top level xBar need to have its configuration or state updated.

We believe that our approach can build scalable, extensible, and locally-scoped control planes for large-scale networks, and that it is the the first approach described in the literature that can do so.

4 References

[1] Beacon: A java-based OpenFlow control platform. http://www.beaconcontroller.net.

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

[3] N. Foster, R. Harrison, M. J. Freedman, C. Monsanto, J. Rexford, A. Story, and D. Walker. Frenetic: a Network Programming Language. In *Proc. of SIGPLAN ICFP*, September 2011.

[4] A. K. Nayak, A. Reimers, N. Feamster, and R. J. Clark. Resonance: Dynamic Access Control for Enterprise Networks. In *Proc. of SIGCOMM WREN*, August 2009.

[5] M. Reitblatt, N. Foster, J. Rexford, and D. Walker. Consistent Updates for Software-Defined Networks: Change You Can Believe In! In *Proc. of HotNets*, November 2011.

[6] R. Sherwood, G. Gibb, K.-K. Yap, G. Appenzeller, M. Casado, N. McKeown, and G. Parulkar. Can the Production Network Be the Testbed? In *Proc. of OSDI*, October 2010.

[7] Trema: An Open Source modular framework for developing OpenFlow controllers in Ruby/C. https://github.com/trema/trema.

[8] A. Voellmy and P. Hudak. Nettle: Taking the Sting Out of Programming Network Routers. In *Proc. of PADL*, January 2011.

[9] M. Yu, J. Rexford, M. J. Freedman, and J. Wang. Scalable Flow-based Networking with DIFANE. In *Proc. of SIGCOMM*, August 2010.