From Ethane to SDN and Beyond

Martín Casado
Andreessen Horowitz
martin@a16z.com

Nick McKeown
Stanford University
nickm@stanford.edu

Scott Shenker
University of California, Berkeley
shenker@icsi.berkeley.edu

This article is an editorial note submitted to CCR. It has NOT been peer reviewed.
The authors take full responsibility for this article's technical content. Comments can be posted through CCR Online.

ABSTRACT

We briefly describe the history behind the Ethane paper and its ultimate evolution into SDN and beyond.

CCS CONCEPTS

• **Networks** → **Network architectures**; **Network types**;

KEYWORDS

Software Defined Networks (SDN)

1 THE ETHANE STORY

SDN is often described as a revolutionary architectural approach to building networks; after all, it was named and first discussed in the context of research. So you might be surprised to learn that we do not think SDN is defined by a particular architecture or technical approach at all. Rather, we think the revolutionary aspect of SDN was about who is in charge. Who gets to decide what functions, protocols, and features are supported by a network: Is it the equipment vendors (e.g., router manufacturers), the technology providers (e.g., chip and optics manufacturers), or those who own and operate networks? The big change that happened in networking since the Ethane paper was published in 2007, and since the term SDN was first used, is that those who own and operate networks are starting to take charge of how their networks operate. This, we believe, is the real story of SDN. In this paper we review how the revolution started, the intellectual and technological underpinnings that gave rise to it, and how it has continued well beyond SDN.

Before 2000, the Internet grew incredibly quickly because it was easy to plug interoperable pieces together, with no need to ask a central controlling authority; it helped that the network infrastructure was deliberately and carefully designed to be simple and streamlined. It was taken for granted that networks were built using commercially available switches, routers, base-stations, and access points sold by traditional networking equipment vendors, then strung together and configured to implement the behavior the network operator desired. Over time, thousands of new IETF RFCs and IEEE standards were written and router manufacturers – who needed to serve many customers with one product – added more and more features to their routers. By the mid 2000s, the routers used by ISPs were so complicated that they were based on more than 100 million lines of source code – ironically, more than 10-times the complexity of the largest telephone exchange ever built – and they supported hundreds of protocols. The Internet was paying the price for this complexity: routers were bloated, power hungry, and expensive, and internally they were based on old engineering practices and poorly defined APIs.

Yet, most customers only used a handful of these features. One approach would have been to completely redesign the router hardware and software around cleaner APIs, better abstractions, and modern software practices. But time-to-market pressures meant they couldn't start over with a simpler design. And while in other industries startups can enter the market with more efficient approaches, the barrier to entry in the router business had been made so tall that there was little chance for significant innovation. Instead, the router vendors continued to fight problems of reliability and security brought on by overwhelming complexity. The research community, sensing the frustration and struggling to get new ideas adopted, labeled the Internet as "ossified" and unable to change. In response, they started research programs like NewArch [7], GENI [4], FIND [9], 100x100, and the Clean Slate Program to investigate how the Internet might move past this stagnation.

Arising out of this intellectual ferment, our Ethane paper [18] described a different way of building and managing a network. We had noticed that on Stanford campus there were about 2,000 Ethernet switches and routers in wiring closets, each one contained one or more control processors running millions of lines of code. The essential networking task was quite simple – authenticate users and route packets over Ethernet and IPv4 between local and remote end-points – yet the university employed over 200 people to keep the network going.

It all seemed unnecessarily complicated and expensive, and almost impossible to manage as no one person could have a clear view of how the whole network operated. So we decided to design and prototype a network that used operating system design principles so that the entire network could be managed via a centrally-declared, high-level policy. Our goal was both to reduce operational overhead and complexity, and to show that it was possible to build a network architecture that could be globally programmed from a centralized (or tightly clustered) control plane thus vastly reducing both the programming and operating models.

For Ethane, we built and deployed access points and (NetFPGA-based) switches that were externally controlled from central servers. The prototype was an extreme design, more as a thought experiment, to test the limits of scale: Every flow was intercepted (later called *reactive control*) and sent to a central controller which authenticated the user and then decided whether or not the flow was allowed by consulting the 137 policies we identified in our campus network (e.g., laptops cannot accept incoming connections; VoIP phones must not be moved so they could be found in an E911 emergency; Windows machines should be confined to their own VLAN).

The control plane, running remotely on a regular Linux server, chose the route and told the switches how to forward packets.

2 LESSONS FROM ETHANE

The primary takeaway was not that this approach was too hard, but that it was so easy. To our surprise, we concluded that the 2,000 CPUs in the switches on campus could be replaced by a single CPU, with easily enough capacity to make decisions for all the flows on campus. We also learned some more general architectural lessons:

- **The network control plane lacked good abstractions.** The data plane, through encapsulation and layering, already had well-defined protocol abstractions; but at the time, the control plane had almost none. The control plane implementations lived within individual network elements, and had a proprietary interface with the forwarding ASIC.
- **There was a natural initial abstraction.** Many network control mechanisms (e.g., routing protocols, firewall functions, management) can be broken into two parts: (1) A distributed protocol that builds a consistent global view of the state of the network (e.g., the link-state protocol in OSPF that builds a view of the current topology, or a database of authenticated users in 802.1x), and (2) An algorithm that acts upon the global state (e.g. Dijkstra's algorithm, or user-specific access control). We learned that we could separate the two by providing the global view of network state as an abstraction upon which many use-specific algorithms could be built.
- **Innovation required two open interfaces.** While many different algorithms can be applied against the global view, rapid innovation can only happen if the operators – rather than the equipment vendor – determine which algorithms are deployed. This requires that there be an open interface to this control abstraction, so operators can deploy code (from third-party vendors, or open-source, or created in-house) that uses this interface. In addition, these algorithms must also be able to access an open interface that allows them to configure the forwarding plane in each network element. After our experience with Ethane, we grew to see these two interfaces as the crucial enablers of innovation.

We firmly believed that networks built this way would be simpler to deploy, upgrade and manage. Eager to share our conclusions with others, we went to talk to our colleagues at a few networking equipment vendors. The reaction was very interesting: At software companies, the reaction was almost a shrugging of shoulders - of course: Software is eating the world because of the ease of development, testing and deployment of new ideas. If you put software in the hands of developers, they will tailor it to meet their needs.

At networking equipment companies, the reception was quite the opposite: They were threatened by a model that handed over control to the network owner. On one particular occasion, we presented our ideas to Cisco; they went red in the face with anger and told us it could never work. It was in the parking lot after our meeting that we resolved to try and make it happen.

We embarked on a deliberate two pronged approach: The first was to realize our newfound architectural lessons in a concrete design. To do that, we created an open API (OpenFlow) that allowed the forwarding plane in each router to be externally configured, and a general SDN controller (NOX) that would expose the global network view and use OpenFlow to control the forwarding plane. The goal was to offer platforms and standards that could be used by academics, researchers, hobbyists, and industry for whatever purposes suited them. It was the start of the community movement behind SDN, which then lead to NOX [22], POX [12], FlowVisor [28] and so on.

The second prong was commercial. Our discussions with networking equipment vendors had convinced us that they would be very resistant to our approach, so we would need to bring products to market ourselves. We started Nicira in 2007 to develop products around SDN and provide the necessary support needed for production deployment. In the early days Nicira worked with Google to develop ONIX, the distributed SDN control plane that Google deployed in its networks. Google first published their deployment of ONIX/OpenFlow in their private backbone network in 2013 [23] and then in their data center in 2016 [29].

But soon Nicira realized that switch vendors were not eager to add a usable OpenFlow interface to control their equipment remotely from ONIX. They felt too threatened by the SDN model to engage, and instead continued their proprietary ways. Nicira was then at a crossroads, in that our progress was being stymied by the very stagnation we were trying to overcome. Fortunately, around this time virtualized datacenters were becoming more popular, and they provided two things necessary for the successful commercialization of SDN: a killer app, and a viable deployment path.

The "killer app" was network virtualization. Network virtualization allows each tenant to run a virtual network in the cloud that has its own address space and is configured according to the tenant's requirements, thus giving each tenant the illusion of having their own private network completely under their control. This was something customers wanted, but could not be done at scale with traditional network management.

But how could this be deployed without the switches supporting OpenFlow? In a virtualized datacenter, every server runs a software switch, called a vswitch, that provides connectivity between the VMs residing on the server. We recognized that supporting an application like network virtualization did not require controlling the forwarding plane in every switch, but only at the vswitches on the servers. Nicira then developed Open vSwitch (OVS[26]) to provide an open-source vswitch that could be controlled remotely by ONIX or any SDN controller. OVS was quickly adopted, and it then became possible to deploy edge-based SDN systems in virtualized datacenters without touching any of the networking hardware.

This is how network virtualization became the first widely-deployed SDN application: an urgent customer need, and a viable deployment path. The descendant of Nicira's original implementation of network virtualization, NSX[24], is now a multi-billion dollar business for VMware, and many others have built upon the original ideas.

3 THE FOUNDATIONS OF SDN

SDN as a term has become so broad it now extends far beyond the original work to encompass an industry movement that was

starting to happen anyway, as well as many projects and research efforts that far predate our work. So with that in mind, we reflect a bit about the work that we drew inspiration from when building Ethane, then OpenFlow, NOX[22], ONIX[25], and NSX.

The original problem statement, of a centrally managed network with an OS-like unified policy language, came from Martín's experience working on secure government networks, where it was clear that traditional network designs simply were not suited for implementing a unified network security policy. We drew inspiration from 4D [21] in the original solution. While we ultimately choose a different point in the design space based on flows for control and generality, we strongly agreed with the notion of decoupling the control plane, and relegating the dataplane to simple forwarding elements that were remotely populated.

Over the years of pursuing the more general agenda of giving operators control over their networks, we have drawn from a tremendous amount of previous work, both from the academic literature and internal commercial efforts. We would like to stress that much of that work also lay the foundation for efforts done under the SDN umbrella, so SDN's roots are far broader than Ethane. While there is too much history to adequately cover here, Feamster et. al. have provided a thorough discussion [20].

In addition to these intellectual roots, there were two powerful technological forces that gave rise to SDN. One was the rise of modern hyperscale datacenters. They created a new market segment which (i) demanded more flexibility than traditional networks, (ii) had extensive green-field deployments, so they were not chained to traditional approaches, and (iii) were run by network operators who had the expertise to devise their own networking solutions if given the appropriate hardware. The other trend was the rise of merchant switching silicon – from providers such as Broadcom, Fulcrum, and Marvell – that enabled the development of white-box networking gear. As a result, the operators of hyperscale datacenters could insist on being in control of their network: SDN was perhaps the means by which they did so, but their economic and intellectual power, coupled with the availability of white-box switches, is what really changed who was in control.

4 THE TREND CONTINUES

SDN is only the first step in giving operators control over their networks. While SDN focused mostly on the control plane, two more recent developments deal with the dataplane. While most discussion of networks tend to focus on routers, there are a wide variety of network "middleboxes" deployed in most networks, such as firewalls, WAN optimizers, and the like. In fact, according to a study [27], roughly one-third of the network elements in most networks are these more sophisticated network functions. Just like routers, these network functions were typically provided by vendors who packaged proprietary software (and sometimes hardware) in a closed box. In 2012, a white paper on Network Function Virtualization (NFV) [8] was written by network operators that called for these network functions to be deployed as VMs on racks of commodity servers. This would allow network operators to install and remove these network functions without any change in hardware, giving them much more control over their network. While NFV is not yet widely adopted, there is widespread agreement on its inevitability. NFV is not directly connected to, or enabled by, SDN, but it shares the mission of changing the locus of control.

However, neither SDN nor NFV wrested control of the actual packet processing functions in routers away from the traditional vendors, and if a network owner is not in charge of how packets are processed, then they are not really in charge of their network. In recent years there has been an important trend towards making the forwarding plane programmable, as exemplified by the P4 programming language [17] and various programmable forwarding chips.

Giving operators control over their networks has played out largely in the commercial arena, changing how we build networks for datacenters, carriers, and enterprises. Open-source projects have, for the first time, made significant inroads into production networks, in addition to software switches such as OVS, BESS [1], and VPP [16], this includes switch operating systems (e.g. SONiC [14], FBOSS [19], Stratum [15], DANOS [3]), control planes (e.g. ONOS [11], ODL [13]) and network-wide management systems (e.g. CORD [2], ONAP [10]).

The SDN trend has also been important for extending networks into domains where establishing connectivity has been hard. In one such effort, a network virtualization approach similar to NSX was used to implement a distributed mobile packet core that works with low cost base stations, and whose management stack runs in the cloud [6]. True to the early promises of SDN, this results in the ability to dramatically reduce the cost of building out LTE networks, and to make them easier to set up and manage. This has been used to connect a number of rural communities, such as Havasupai [5], the most remote city in the lower 48, where traditional approaches have simply been cost or operationally prohibitive.

SDN came about because of a pressing need for those who own and control networks to decide how they work. With NFV and programmable switches now joining SDN in this movement, we now have the capability to install programmable interfaces at every level of the network, from the control plane to the forwarding plane to more sophisticated network functions. When these technologies become ubiquitously deployed, network operators will be in full control of their networks.

Besides enabling new functionality and the ability to customize the network to the operator's needs, this full set of programmable interfaces allows the networking community to start applying other tools – such as verification and declarative interfaces – that are widely used elsewhere in systems. If we continue to push forward with this trend, we soon will not be teaching students about protocols. Instead, we will teach them about a programmable substrate, called a network, where you describe a behavior you want in software, which is then compiled to make future versions of the Internet.

REFERENCES

[1] Berkeley Extensible Software Switch (BESS). https://github.com/NetSys/bess.
[2] CORD: Central Office Re-architected as a Datacenter. https://www.opennetworking.org/cord/.
[3] DANOS: Distributed Network Operating System. https://www.linuxfoundation.org/networking-orchestration/2018/03/the-linux-foundation-hosts-danos-project-a-unified-network-operating-system/.
[4] GENI Project. https://www.geni.net/.
[5] Havasupai secures license to retain, expand internet access. https://www.apnews.com/f3725f04d8c54c1fa6777f37406b858a.

[6] Magma web page. https://connectivity.fb.com/magma/.
[7] NewArch Project. https://www.isi.edu/newarch/.
[8] NFV White Paper. https://portal.etsi.org/NFV/NFV_White_Paper.pdf.
[9] NSF Future Interjet Architecture Project. http://www.nets-fia.net/.
[10] ONAP: Open Network Automation Platform. https://www.onap.org/.
[11] ONOS: Open Source Network Operating System. https://onosproject.org/.
[12] POX. https://github.com/noxrepo/pox.
[13] The OpenDaylight Project. https://www.opendaylight.org/.
[14] The SONiC Project. https://azure.github.io/SONiC/.
[15] The Stratum Project. https://www.opennetworking.org/stratum/.
[16] VPP Software Switch. https://wiki.fd.io/view/VPP.
[17] Pat Bosshart, Dan Daly, Glen Gibb, Martin Izzard, Nick McKeown, Jennifer Rexford, Cole Schlesinger, Dan Talayco, Amin Vahdat, George Varghese, et al. 2014. P4: Programming protocol-independent packet processors. *ACM SIGCOMM Computer Communication Review* 44 (2014), 87–95.
[18] Martin Casado, Michael J. Freedman, Justin Pettit, Jianying Luo, Nick McKeown, and Scott Shenker. 2007. Ethane: Taking Control of the Enterprise. *SIGCOMM Comput. Commun. Rev.* 37, 4 (Aug. 2007), 1–12. https://doi.org/10.1145/1282427.1282382
[19] Sean Choi, Boris Burkov, Alex Eckert, Tian Fang, Saman Kazemkhani, Rob Sherwood, Ying Zhang, and Hongyi Zeng. 2018. FBOSS: Building Switch Software at Scale. In *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication (SIGCOMM '18)*. ACM, New York, NY, USA, 342–356. https://doi.org/10.1145/3230543.3230546
[20] Nick Feamster, Jennifer Rexford, and Ellen Zegura. 2014. The Road to SDN: An Intellectual History of Programmable Networks. *SIGCOMM Comput. Commun. Rev.* 44, 2 (April 2014), 87–98. https://doi.org/10.1145/2602204.2602219
[21] Albert Greenberg, Gisli Hjalmtysson, David A. Maltz, Andy Myers, Jennifer Rexford, Geoffrey Xie, Hong Yan, Jibin Zhan, and Hui Zhang. 2005. A Clean Slate 4D Approach to Network Control and Management. *SIGCOMM Comput. Commun. Rev.* 35, 5 (Oct. 2005), 41–54. https://doi.org/10.1145/1096536.1096541
[22] Natasha Gude, Teemu Koponen, Justin Pettit, Ben Pfaff, Martín Casado, Nick McKeown, and Scott Shenker. 2008. NOX: Towards an Operating System for Networks. *SIGCOMM Comput. Commun. Rev.* 38, 3 (July 2008), 105–110. https://doi.org/10.1145/1384609.1384625
[23] Chi-Yao Hong, Subhasree Mandal, Mohammad Al-Fares, Min Zhu, Richard Alimi, Kondapa Naidu B., Chandan Bhagat, Sourabh Jain, Jay Kaimal, Shiyu Liang, Kirill Mendelev, Steve Padgett, Faro Rabe, Saikat Ray, Malveeka Tewari, Matt Tierney, Monika Zahn, Jonathan Zolla, Joon Ong, and Amin Vahdat. 2018. B4 and After: Managing Hierarchy, Partitioning, and Asymmetry for Availability and Scale in Google's Software-defined WAN. In *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication (SIGCOMM '18)*. ACM, New York, NY, USA, 74–87. https://doi.org/10.1145/3230543.3230545
[24] Teemu Koponen, Keith Amidon, Peter Balland, Martin Casado, Anupam Chanda, Bryan Fulton, Igor Ganichev, Jesse Gross, Paul Ingram, Ethan Jackson, Andrew Lambeth, Romain Lenglet, Shih-Hao Li, Amar Padmanabhan, Justin Pettit, Ben Pfaff, Rajiv Ramanathan, Scott Shenker, Alan Shieh, Jeremy Stribling, Pankaj Thakkar, Dan Wendlandt, Alexander Yip, and Ronghua Zhang. 2014. Network Virtualization in Multi-tenant Datacenters. In *11th USENIX Symposium on Networked Systems Design and Implementation (NSDI 14)*. USENIX Association, Seattle, WA, 203–216. https://www.usenix.org/conference/nsdi14/technical-sessions/presentation/koponen
[25] Teemu Koponen, Martin Casado, Natasha Gude, Jeremy Stribling, Leon Poutievski, Min Zhu, Rajiv Ramanathan, Yuichiro Iwata, Hiroaki Inoue, Takayuki Hama, and Scott Shenker. 2010. Onix: A Distributed Control Platform for Large-scale Production Networks. In *Proceedings of the 9th USENIX Conference on Operating Systems Design and Implementation (OSDI'10)*. USENIX Association, Berkeley, CA, USA, 351–364. http://dl.acm.org/citation.cfm?id=1924943.1924968
[26] Ben Pfaff, Justin Pettit, Teemu Koponen, Ethan Jackson, Andy Zhou, Jarno Rajahalme, Jesse Gross, Alex Wang, Joe Stringer, Pravin Shelar, Keith Amidon, and Martin Casado. 2015. The Design and Implementation of Open vSwitch. In *12th USENIX Symposium on Networked Systems Design and Implementation (NSDI 15)*. USENIX Association, Oakland, CA, 117–130. https://www.usenix.org/conference/nsdi15/technical-sessions/presentation/pfaff
[27] Justine Sherry, Shaddi Hasan, Colin Scott, Arvind Krishnamurthy, Sylvia Ratnasamy, and Vyas Sekar. 2012. Making Middleboxes Someone else's Problem: Network Processing As a Cloud Service. In *Proceedings of the ACM SIGCOMM 2012 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication (SIGCOMM '12)*. ACM, New York, NY, USA, 13–24. https://doi.org/10.1145/2342356.2342359
[28] Rob Sherwood, Glen Gibb, Kok kiong Yap, Martin Casado, Nick Mckeown, and Guru Parulkar. 2009. *FlowVisor: A Network Virtualization Layer*. Technical Report.
[29] Arjun Singh, Joon Ong, Amit Agarwal, Glen Anderson, Ashby Armistead, Roy Bannon, Seb Boving, Gaurav Desai, Bob Felderman, Paulie Germano, Anand Kanagala, Hong Liu, Jeff Provost, Jason Simmons, Eiichi Tanda, Jim Wanderer, Urs Hölzle, Stephen Stuart, and Amin Vahdat. 2016. Jupiter Rising: A Decade of Clos Topologies and Centralized Control in Google's Datacenter Network. *Commun. ACM* 59, 9 (Aug. 2016), 88–97. https://doi.org/10.1145/2975159