CHAPTER

# SDN FUTURES 15

An accurate forecast of the future of any nascent technology calls for a better crystal ball than what is usually available. To attempt such a look into the future requires that we first lift ourselves up out of the details of SDN on which we have focused in the latter half of this book and view the current state of SDN affairs with a broader perspective. We need to consider what lies in store for SDN as a whole as well as individually for each of the SDN alternatives described in Chapter 6. We are optimistic about the future of SDN. Our optimism for the future of SDN is partly founded on the belief that there are many potential areas of application for the technology that lie outside those we explored in Chapters 8–10. For that reason, we dedicate a significant part of this chapter to revealing just a few of the novel use cases and research areas for Open SDN that are beginning to attract attention. Before that, however, we should acknowledge that not all contemporary press releases about SDN are filled with unbridled optimism. In the next section, we look at the turbulence and some of the pessimism surrounding SDN.

## 15.1 CURRENT STATE OF AFFAIRS

The turbulence that has surrounded the SDN movement during these early years has closely tracked the well-known *Gartner Hype Cycle* [1]. We depict a generic version of this hype cycle in Fig. 15.1. For SDN, the *technology trigger* was the birth of OpenFlow and the coining of the term Software Defined Networks in 2009. [1] The MAC table and VLAN ID exhaustion we have discussed in earlier chapters were manifesting themselves in the data center in the years leading up to 2012. This provided hard evidence that networking was at the breaking point in the data center. The fact that SDN provides a solution to these problems added fuel to the SDN fire. The *peak of inflated expectations* was reflected in the flurry of acquisitions in 2012 and the sizable VC investments preceding and coinciding with that. The *trough of disillusionment*, in our opinion, occurred in early 2013.

Big Switch's fate leading up to 2013 and then in the first half of that year mirrored these trends. Their withdrawal first from board status of the OpenDaylight consortium and subsequent total exit reflected their feeling that forces other than a commitment to Open SDN were influencing that organization's choice of controller technology. That was followed by their *big pivot* which was an admission that their early strategy of open platforms needed to be retooled to be plug-and-play solutions for a broader

[1]When we refer to the first use of the term Software Defined Networks we mean in the specific context in which it is used in this book and in conjunction with the work surrounding OpenFlow at Stanford University. The term itself was used previously in patents in the 1980s.

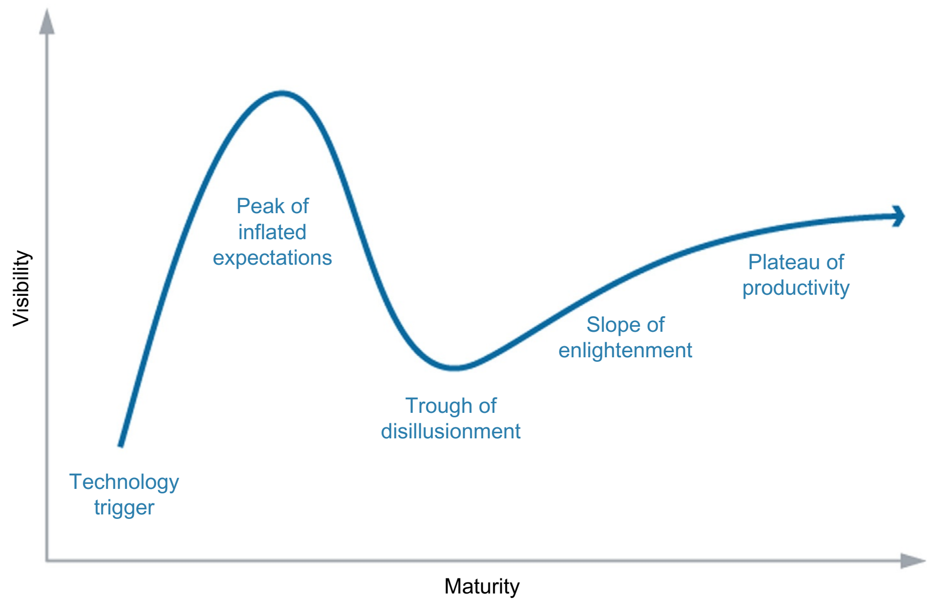

**FIG. 15.1**

Gartner Hype Cycle.

*(Source: Gartner, Inc. Research methodology. Retrieved from: http://www.gartner.com/it/products/research/methodologies/research_hype.jsp.)*

customer base (see Section 14.9.1). Big Switch's moves during 2013 were reflective of a more general fragmentation of the SDN movement.

Many feel that the degree of SDN washing that now occurs means that SDN can mean anything to anyone. Indeed, one of the founding fathers of SDN listed in Section 11.1.1, Martin Casado, was quoted in April 2013 as saying "I actually don't know what SDN means anymore, to be honest" [2]. If this is the feeling of one of the core group that coined the very term SDN then one might reasonably fear that SDN has lost much of its original focus.

The Gartner Hype Cycle does not end with the trough of disillusionment, however, nor does Open SDN's story begin to fall apart in 2013. Just as the Hype Cycle predicts that disillusionment is followed by the *slope of enlightenment* and then by the *plateau of productivity*, so do we now begin to see realistic assessments for where SDN will be applied. We believe that this slope of enlightenment in SDN's future reflects a middle ground between the extremes of belief that it will totally unseat the incumbent NEMs and traditional switching technology, and believing that it is largely an academic exercise with applicability restricted to the 1% of customers sophisticated enough to grapple on their own with the technological change. As an example, Big Switch's pivot has moved the company toward a strategy based on OpenFlow-enabled white-box switches. Thus, while their strategy is more tactical than it had been, they are still betting on the long-term future of OpenFlow. Furthermore, SDN has already established a beachhead in the data center that demonstrates that it is there to stay. SDN experts from HP, Dell, and NEC debating the future of SDN in [3] caution against impatience with the rate of penetration of SDN. They point out that server virtualization in the data center has existed for over a decade and only now is reaching 50% market penetration.

**DISCUSSION QUESTION**

In which of the five phases depicted in Fig. 15.1 do you believe SDN found itself in 2015?

We believe that for SDN to transition from the slope of enlightenment to the plateau of productivity we must first see evidence of an entirely new generation of switching ASICs that have been purposely built with SDN and OpenFlow in mind. Switch ASICs may take 2 years or more to mature and reach production. Perhaps the amount of flow table space required will mandate new silicon geometries and the time to availability may be even longer.

The integration of SDN into the work being performed by the IEEE and other important forums is further evidence of its growing acceptance. Nonetheless, the exact forms of SDN that are here to stay are still debated. In this chapter we will discuss some new forms of SDN that have emerged between 2012 and 2015 as well as areas of research and experimental use cases that provide evidence of the totally unique ways that SDN can address networking problems. The slope of enlightenment embodies a recognition that SDN solves certain networking problems better than any other known technology and that it will indeed be disruptive there. This is a strong vindication of SDN but falls short of being the death knell for traditional networking, as some have argued. It is likely that the reduction in hype will reduce the appeal of applying the SDN moniker to every new networking variant, and the different versions of SDN that gain market traction, while not all OpenFlow-based by any means, will tend increasingly to those characteristics that we used to define Open SDN in Section 4.1.

Among those basic characteristics, one wildcard is that of *openness*. Openness means different things to different members of the networking ecosystem. For an academic researcher, it means open source available to all. On the other extreme, openness to an NEM may mean exposing certain APIs to your product in order to allow user-written applications to fine-tune external control knobs that the NEM exposes. (These two extremes reflect our earlier definitions of Open SDN vs SDN via APIs.) A special report [4] on the history and demise of the *Open Systems Interconnection* (OSI) standards concluded with the following remark: “Perhaps the most important lesson is that ‘openness’ is full of contradictions. OSI brought to light the deep incompatibility between idealistic visions of openness and the political and economic realities of the international networking industry. And OSI eventually collapsed because it could not reconcile the divergent desires of all the interested parties.” This lesson learned the hard way for OSI should not be lost on Open SDN evangelists. Too strict of an interpretation of openness in the definition of SDN could result in a similar fragmentation of interests between the open source community and industry. If major parts of industry cannot align behind the Open SDN movement, then the movement will morph into something that is synergistic with industry’s goals or it will be relegated to the halls of academia. This is the *realpolitik* of the Internet.

## 15.2 SD-WAN

SDN for the WAN, or SD-WAN as it is commonly known, is one of the most promising future areas for the application of the SDN principles presented throughout this book. In Section 14.9.5 we discussed several of the startups that target this potent strategy of using an overlay of virtual connections, often via some tunneling mechanism across a set of WAN connections, including MPLS, Internet, and other WAN transport links [5]. The overlay is achieved by placing specialized edge devices at branch locations. These *Customer Premises Equipment* (CPE) devices are managed by a centralized controller. The key function of these edge devices is the intelligent mapping of different user traffic flows over the most appropriate available tunnel. To a large degree, this is focused on cost savings achieved by placing only the traffic that is highly sensitive to loss, delay, or jitter on the more expensive links offering those guarantees. Normal data traffic such as email and web browsing may be mapped to best-effort, less expensive broadband Internet connections. Some classes of traffic may be routed over less expensive wireless backup links if that traffic can sustain periodic downtime or lower bandwidth.

It is easy to envision such an SD-WAN as a natural extension of the SDN-via-Overlays introduced in Section 4.6.3 and discussed at length in Chapters 6 and 8. SD-WAN is indeed related to our earlier discussions on SDN-via-Overlays in that in the data center the virtual switches in the overlay solution decide over which tunnel to inject traffic over the data center network. The CPE gateways in SD-WAN solutions perform a similar function of intelligently mapping application traffic to a particular tunnel. The criteria by which these mapping decisions are made are much more complex in SD-WAN than in the data center, however. For instance, in the data center overlay solution, there is typically just one tunnel between two hosts, whereas a significant feature of the SD-WAN solution is intelligent selection between tunnel alternatives to optimize cost and performance.

We saw earlier that this overlay approach can be successfully used to address data center problems of large layer 2 domains where the size of the MAC address tables can exceed the capacity of the switches. This technology was also used to limit the propagation of broadcast traffic that is an inherent part of the layer 2 networking so prevalent in the data center. While such issues might not be relevant in binding together a geographically dispersed enterprise over the WAN, the application of overlays in SD-WAN is more motivated by cost savings. While the variations in cost and QoS characteristics of different transport links within the data center are negligible, this is not the case in the WAN. The cost of an MPLS link offering QoS guarantees is significantly higher than a best-effort Internet connection. The provisioning time for such an MPLS connection is far greater than for the Internet connection. The difference is even greater with other wireless WAN link alternatives such as LTE or satellite [6]. Another major difference between the data center overlay paradigm and SD-WAN lies in the fact that SD-WAN CPE devices exercise little control over the actual paths used by the virtual link. Whereas in theory an OpenFlow controller could determine the exact hop-by-hop routing of a virtual link in a data center, within SD-WAN the edge devices view the tunnels they have at their disposal as coarse entities with certain cost, loss, speed, and QoS characteristics. They may choose to map a particular application flow over a given link according to those characteristics, but they are unlikely to be able to dynamically change the characteristics of any one of those links on the fly, including the actual path over which that tunnel, for example, is routed.

The interest in applying SDN to the WAN in some ways is an extension of earlier efforts at WAN optimization, such as data compression, web caching, and *Dynamic Multipoint Virtual Private Networks* (DMVPN) [7]. Indeed, some of the companies offering SD-WAN products today have their roots in WAN optimization. In addition to generating local acknowledgments and caching data, WAN optimization is also related to shaping and steering traffic to where it is best suited. For example, QoS-sensitive traffic like VoIP could be directed to MPLS paths that could offer latency and jitter guarantees while other traffic could be shunted to less expensive Internet links. There were certainly pre-SDN attempts of moving control away from the data plane such as the use of *Route Reflectors* for computing the best paths for MPLS links [8]. This is fully generalized within SD-WAN via the use of a centralized controller in the SDN paradigm. Of the five fundamental traits of SDN that have been a recurring theme throughout this work, SD-WAN solutions exhibit *centralized control*, *plane separation*, and *network automation and virtualization*. To date, there is little evidence in SD-WAN solutions of *a simplified device* or *openness*.

Some facets of the SD-WAN solutions converge on standards-based solutions [9]. In particular, the encryption tunnels and key distribution converge on SSL VPNs and IPSec. Conversely, compression and optimization methods, along with the path selection algorithms, tend to be proprietary and where the unique value-add of the vendors is concentrated.

In general, SD-WAN refers to any WAN managed by software control. The *Open Networking User Group* (ONUG) defines two broad types of SD-WANs [10]:

- Intradomain SD-WANs, where a single administrative domain uses SDN-controlled switches to accomplish various network management tasks, such as provisioning of secure tunnels between multiple geographically distributed portions of a network that are under the control of a single administrative domain.
- Interdomain SD-WANs, where multiple independently operated domains connect to one another via a shared layer-2 switch to accomplish various network management tasks, including inbound traffic engineering and denial-of-service (DoS) attack prevention.

As with any purported technological panacea, there are many challenges confronted in the actual implementation of the SD-WAN paradigm. We discuss these in the next section.

**DISCUSSION QUESTION**

Which of the five basic SDN traits defined in Section 4.1 do SD-WAN solutions generally exhibit? Give an example of each.

### *The devil is in the details*

Due to the vibrant demand for commercial SD-WAN solutions, the marketplace abounds with contending claims and technologies. A very useful guide for evaluating and comparing SD-WAN solutions is found in [11]. Another is [12], where the author poses 13 questions to ask about how a given SD-WAN solution will address a specific challenge. These questions arose during the *Networking Field Day Event 9* [13]. Two vendors published their responses in [14,15], respectively. We leave it to interested reader to see all 13 questions and the various responses via the provided references. Here we look at three of the questions, one related to the concept of *host routing*, another dealing with *asymmetric routing*, and a third about the issue of *double encryption*.

Host routing question:

> How does the SD-WAN solution get traffic into the system? As in, routers attract traffic by being default gateways or being in the best path for a remote destination. SD-WAN tunnel endpoints need to attract traffic somehow, just like a WAN optimizer would. How is it done? WCCP? PBR? Static routing? (All 3 of those are mostly awful if you think about them for about 2.5 seconds.) Or do the SD-WAN endpoints interact with the underlay routing system with BGP or OSPF and advertise low cost routes across tunnels? Or are they placed inline? Or is some other method used? [12]

The answer depends on whether the CPE equipment is attempting to be standards-based or proprietary. If proprietary, some of the proposed solutions described *application traffic sniffing* whereby the CPE box analyzes the application traffic to determine the nature of the application and thus the appropriate transport link over which it should be mapped. Such a proprietary approach can be stymied if the application traffic is already encrypted by the application. Since most SD-WAN solutions suggest that the tunnels forming the virtual links should themselves be encrypted, this highlights the issue cited in question 8 in [12], which is how to deal with unnecessary double encryption. An alternative to application traffic sniffing in the CPE gateway is to use *host routing*. The key concept here is that the host itself understands that one gateway exiting the local site is preferred over another for that traffic type. A layer 2 alternative to this is for hosts to belong to different VLANs depending on the traffic type. A simple example of this is a data VLAN and a voice VLAN. The voice VLAN's traffic is mapped over the WAN virtual link that offers QoS guarantees and the data VLAN is mapped over the lower cost link that does not offer such guarantees.

Asymmetric routing question:

Is path symmetry important when traversing an SD-WAN infrastructure? Why or why not? Depending on how the controller handles flow state and reflects it to various endpoints in the tunnel overlay fabric, this could be an interesting answer. [12]

When routing decisions between two endpoints were based on simple criteria such as the least cost path, traffic flowing between those two endpoints would generally follow the same path regardless of which direction it flowed. With more complex routing criteria, the possibility of asymmetric routing becomes reality. Asymmetric routing means that the packets flowing from application A to application B may take a different path than packets flowing from B to A. This may or may not be a problem. It certainly is a documented problem for certain older classes of firewalls. Thus knowing whether or not a given SD-WAN solution may engender asymmetric routing may be an important consideration.

Double encryption question:

Double-encryption is often a bad thing for application performance. Can certain traffic flows be exempted from encryption? As in, encrypted application traffic is tunneled across the overlay fabric, but not encrypted a second time by the tunnel? [12]

The possibility of double encryption is both a performance consideration as well as a design consideration. If the SD-WAN device purports to determine application flow routing based on the aforementioned sniffing, this will not work in the event that the application has already encrypted the traffic. Possible workarounds here include using unencrypted DNS lookups to *guess* the nature of the application and thus the appropriate mapping of its traffic to a given WAN virtual link [16]. The second consideration of performance is simply the inefficiency of encrypting and decrypting the traffic a second time. Some SD-WAN solutions could offer unencrypted tunnels for WAN virtual links specifically for already-encrypted traffic. Another possible alternative would be to terminate the application encryption at the originating CPE gateway, perform the sniffing, and then reencrypt.

Cisco's *Intelligent WAN* (IWAN) is a hybrid WAN offering and another example of SD-WAN. Cisco's venerable *Integrated Services Router* (ISR) plays the role of the CPE gateway in this solution.

You can automate IWAN feature configuration using the IWAN application that runs in Cisco's *Application Policy Infrastructure Controller—Enterprise Module* (APIC-EM). IWAN is built upon the DMVPN technology mentioned previously. DMVPN provides dynamic secure overlay networks using the established *Multipoint GRE* (mGRE) and *Next-Hop Resolution Protocol* (NHRP) technologies, all of which predate SD-WAN. DMVPN builds a network of tunnels across the Internet to form the SD-WAN virtual links.

Cisco IWAN builds an SD-WAN solution using largely existing Cisco technology as well as technology obtained via its acquisition of Meraki. Many of these are Cisco-proprietary protocols, but they are generally less opaque than the closed systems of the SD-WAN startups. The Cisco IWAN design guide [17] gives detailed instructions about configuration of the HSRP and *Enhanced Interior Gateway Routing Protocol* (EIGRP) routing protocols in order to get the right traffic routed over the right WAN link. The ability to form and send traffic over VPN tunnels over two interfaces provides administrators with a way to load-balance traffic across multiple links. *Policy-based Routing* (PBR) allows IT staff to configure paths for different application flows based on their source and destination IP addresses and ports [18]. IWAN also allows configuration of performance criteria for different classes of traffic. Path decisions are then made on a per-flow basis as a function of which of the available VPN tunnels meet these criteria, which is in turn determined by automatically gathered QoS metrics.

Cisco's IWAN differs from some of the startup alternatives in a few fundamental ways. The mapping of traffic is not based on an application sniffing approach. Instead, APIC-EM tells the border router which egress path to take based on the conditions of the path as well as routing policies, and then intelligently load-balances the application traffic across the available WAN virtual links. According to [17], the variety of the WAN links is limited to MPLS and the Internet, whereas some of the alternatives offer wireless WAN connections as well. The number of WAN links for a given site is limited to a primary and secondary link, whereas the choice between more than two WAN links is feasible with some of the other solutions. It is important to understand that the number of tunnel endpoints may be far greater than the number of offered WAN links, depending on the particular SD-WAN solution. Each tunnel endpoint may map to a different customer premise if the design is based on a mesh topology, or each tunnel could terminate in a centralized hub if the design is based on a *hub and spoke* topology. In addition, tunnels may offer different levels of encryption and potentially different QoS guarantees.

Many of the SD-WAN solutions purport to offer dynamic QoS assessment of links and, based on those assessments, to dynamically shift QoS traffic from one path to another. This assessment can be accomplished by *performance probes* periodically injected into the tunnels and then using them to empirically measure QoS levels. The assessments can then be used to adaptively direct traffic via the most appropriate tunnel. This is certainly feasible but the point is obscured when related marketing statements imply that this allows provision of QoS guarantees over the Internet. While it may be possible to detect current QoS characteristics over a given Internet connection and potentially react to it, this falls far short of the marketing hype that implies that the solution can actually enforce QoS levels.

While this general concept, called *dynamic path selection*, is touted by SD-WAN vendors as novel in SD-WAN, we remind the reader of the example in Section 9.6.1 of dynamically shunting traffic over *Optical Transport Networks* (OTN), which is based on similar principles. Hence, SDN has contemplated this before the advent of SD-WAN.

An interesting adjunct to SD-WAN is the possible application of the *Locator/ID Separation Protocol* (LISP) [19]. LISP uses *Endpoint Identifiers* (EIDs), which correspond to hosts, and *Routing Locators*

(RLOCs), which correspond to routers. An EID indicates who the host is and denotes a static mapping between a device and its owner. The RLOC, on the other hand, indicates *where* that device is at a given time. If one considers the mobility of users coming and going from branch offices of an enterprise, it is clear that integration of this kind of technology is very relevant to the problems purported to be addressed by SD-WAN solutions. Indeed, Cisco's APIC-EM controller integrates LISP with IWAN.

**DISCUSSION QUESTION**

Cisco's IWAN SD-WAN solution offers the alternatives of MPLS VPN and Internet-based WAN links. One possible configuration shown in [17] depicts a configuration with *no* MPLS links but only two Internet links, each from a different provider. Beyond additional bandwidth, what is one advantage proferred by such a configuration?

## 15.3 POTENTIAL NOVEL APPLICATIONS OF OPEN SDN

There is considerable active research related to SDN as well as many novel use cases being proposed that illustrate potential new areas where SDN can be applied or improved. In this section we will present a sampling of some of the areas that promise to increase the scope of applications of Open SDN and thus broaden the foundation of its future.

A full survey of SDN-related research is beyond the scope of this work. There are a number of conferences that have focused on SDN research including [20,21]. The proceedings of these conferences provide a good review of current research trends in this area. The survey of the past, present, and future of SDN found in [22] may also provide the reader with broader coverage of current research directions in this field.

We have selected below a number of areas of potential application of Open SDN with the intent of giving the reader a flavor of the areas of application that radically depart from those that we have described during the bulk of this work. It is our hope that this will underscore that this new network programming paradigm opens the doors to many possibilities, only a few of which have already been explored.

### 15.3.1 MANAGING NONTRADITIONAL PHYSICAL LAYER LINKS

There is growing interest in using OpenFlow to control devices that have flows but are not traditional packet switches. The two most promising areas involve flows over optical and wireless links. In Section 9.6.1 we presented a use case where SDN was used for offloading elephant flows onto optical devices. This represents one example of the general area of using OpenFlow to manage flows over physical layers that are outside the domain of classical LANs and WANs. In addition to the example in Section 9.6.1, we mentioned two startups in Section 14.9.7, Plexxi and Vello, that have brought products to market that marry optical switching with OpenFlow.

In [23], the researchers formalize the study of how to map big data applications to an OpenFlow network. An elephant flow is an example of a big data application. The authors propose mechanisms to build on the existing Hadoop[2] job scheduling method to optimize how and when to schedule jobs over the optical links. They point out that unlike the more classical SDN use cases of cloud network provisioning and WAN traffic engineering, such as those discussed in Chapters 8 and 9, such big data jobs require more rapid and frequent updates of the flow tables on the switches. They conclude that current OpenFlow switches are capable of handling their predicted number and frequency of flow table changes. They believe that there may be challenges in keeping the various flow tables across the network switches synchronized with this rate of flow table changes. This is important as one of the problems in traditional networks that we described in Section 1.5.3 was the relatively slow convergence time in the face of routing changes. At the fine granularity of flow table changes being made to route these big-data flows, slow convergence times may be an issue for optical offload.

Because of the ease of installing wireless backhaul links to bring mobile traffic back to the core from *Radio Access Network* (RAN) cells, wireless backhaul is becoming increasingly popular. One of the challenges in doing this is that the effective bandwidth of a wireless link is not constant as in its wired counterpart. In [24, Section 8.4], OpenFlow is proposed as a mechanism to segregate traffic from different providers and different types into separate flows which are then transmitted over a single shared wireless backhaul medium. In the example, the wireless backhaul technology is IEEE 802.16. While the segregation of different traffic types for differential QoS treatment is well established, in the example the wireless backhaul link is potentially shared by a number of different co-located RAN technologies (e.g., LTE, 3G, WiFi) that may be offered by several different operators. The wireless backhaul provider may have different SLA agreements with each of those operators, and for each operator different levels of service depending on the terminating RAN technology, traffic type, or business agreement. Since the wireless backhaul bandwidth capability itself may vary due to environmental conditions, the process of satisfying all of the various SLA commitments becomes far more complex and can benefit from OpenFlow's ability to segregate the traffic into different flows and route or police those flows dynamically with the changing wireless conditions.

### 15.3.2 APPLYING PROGRAMMING TECHNIQUES TO NETWORKS

One of the themes of this work has been that the greater network programmability inherent in OpenFlow provides benefits in the richness of policies and in the fine-grained control it offers the network programmer. To be fair, though, when a programming environment gives the programmer a lot of power, the probability of mis-programming the network is likely greater than the legacy situation where the network engineer was constrained by control knobs of limited functionality. Thus part of the ultimate success of OpenFlow is the development of tools that will aid in detecting such programming flaws.

More formally, in [25] the author suggests that Open SDN provides a proper abstraction of the network that allows us to address networking challenges more efficiently. The author draws an analogy to the difference between programming a CPU in today's high-level languages and application development environments versus programming in machine language. A corollary of this is that the network abstraction allows other advanced programming methodologies to be applied, including debuggers, analysis tools, network simulation, verification tools, and others. These tools have enabled

[2]Hadoop is open source software that enables the distributed processing of large data sets across clusters of commodity computers.

tremendous progress in the development of software because of formal analysis and the application of theory. No equivalents have been practical in the networking space because of the lack of the abstraction as described in [25]. We consider later some of the tools that are enabled by this new way of looking at networking.

### *Network debugger*

In Section 4.1.3 we drew an analogy between the control plane of networks and computer languages and CPUs. We explained that we no longer work at the level of assembly language because high-level languages provide us with efficient abstractions to program the CPUs effectively. We argued that the network needs a similar evolution toward a high-level abstraction of network programming, and suggested that OpenFlow is a good example of such an abstraction. Following this same analogy, if OpenFlow provides us with a way of programming the network as a single entity, can we now consider other advanced programming techniques and tools for application in the network? For example, if we view the network as a vast distributed computer, can we envision a debugger for that computer? This is precisely what the work performed in [26] attempts. The authors propose a *network debugger* (ndb) that will implement basic debugger functionality at the packet level. They aspire to implement a full set of debugger actions, including *breakpoint*, *watch*, *backtrace*, *single-step*, and *continue*. In [26] the authors describe an example where they have implemented both breakpoints and packet-backtraces. For instance, a breakpoint could be set in a network switch such that when a packet matches no flow entry in that switch, the breakpoint is hit. At that point the network programmer could request a packet backtrace showing which flow entries were matched by which switch that led to the packet arriving to this switch where no flow entry matched. The fact that such technology can even be contemplated underscores the more deterministic network behavior possible under the centralized programming model of Open SDN.

### *No bugs in controller execution*

It is also naive to believe that a controller programming a large set of switches that are in a constant state of forwarding packets is an *entirely* deterministic system. The architecture of OpenFlow does not tightly synchronize the programming of multiple switches when the controller must simultaneously program flow entries in several switches. Due to the entropy involved in this process, it may not be possible to ensure that all flow entries are in place before the packets belonging to that flow begin to arrive. Thus race conditions and even routing loops are possible for short periods of time. While it is likely impossible to completely eliminate this hysteresis during rapid network reprogramming, a tool that could model the effects of a particular controller program (i.e., application) could help minimize such occurrences and ensure their effects were short lived. In [27] the authors propose *No Bugs in Controller Execution* (NICE), a tool for modeling the behavior of OpenFlow applications to ensure their correctness, including detecting forwarding loops and black holes. Advanced modeling techniques are used to create input scenarios that test all possible execution paths within the application while taking account of variability of the state of the network switches and links. In [27] the authors modeled and debugged three sample applications: a MAC-learning switch, an in-network server load balancer, and energy-efficient traffic engineering. Additional recent work related to troubleshooting bugs in SDN networks can be found in [28].

### Veriflow

While there is great appeal to checking for OpenFlow application correctness off-line and before deployment, it may be necessary or desirable to perform real-time formal checking for correct network behavior. In [29] the authors describe Veriflow, a system that resides between the controller and the switches and verifies the correctness of each flow entry update before it is applied. One advantage of the approach described in [29] is that it does not require knowledge of all the network programs themselves as it verifies correctness based on observations of flow rules as they are sent from the controller to the switches. One of the biggest challenges faced in this study is to keep the latency of the correctness checks low enough to avoid becoming a bottleneck in the path between the controller and the switches. Such speed is not possible if one were to take a brute-force approach to reflecting the global network state which is very complex and changes rapidly. The authors claim to have developed algorithms that allow such checks to be performed at 100-μs granularity, which is sufficiently fast to avoid adversely affecting network performance. Their novel work is related to how to represent the portion of network state that could possibly be affected by a given rule change and to develop algorithms that reduce the search space such that the effect of a rule change on network state can be ascertained in real time.

### Proof-based verification of SDNs

Like NICE and Veriflow described previously, the authors of [30] propose the use of formal techniques to verify the correctness of SDN programming. The approach in [30] is to encode the controller program and switch functionality in a declarative programming language. The specific formal language used in this work is *NDlog*. They then apply formal proof techniques to confirm the correctness of the programming. Correctness of static configuration includes confirming needed reachability and the absence of loops. Correctness during updates proves that reachability goals and a loop-free topology are maintained across updates. Finally, correct implementation of the controller mandates that internal ports be programmed before ingress ports and that each flow is installed only once and in the correct order.

Verification of correctness of SDN is an important and active research area. Other relevant works in this area include [31,32].

### Yet another network controller

Throughout this book we have presented many different SDN controllers. Each new wave of controller innovation seems to carry with it a new API that is fine-tuned to expose the features of that controller. A downside of this paradigm is that application innovation is slowed by the constant retooling of applications to fit with the evolving controllers. Another drawback is that the rapid evolution of the controller platforms has stymied the development of a mature tool and language set that only evolves over a significant period of time. In [33] the authors formalize their premise of *applying operating system principles to SDN controller design* by providing the controller API and tools via a variant of the well-known Linux operating system. This variant is called *Yet Another Network Controller* (YANC). In this model, SDN applications are Linux processes. These processes communicate with the controller via the Linux file system. The internals of the YANC operating system are such that the file input and output correspond to the dialog typical between an SDN controller and its applications. The applications may be written in any language that supports that file system, providing a degree of freedom not typical of SDN controllers. Furthermore, YANC inherits the rich Linux tool set, making for a complete application development environment.

#### *Software defined networks as databases*

Just as YANC attempts to leverage learnings from the mature field of operating system design, the authors in [34] attempt to apply database techniques to SDN programming. The authors contend that while writing an SDN application may seem easy, applications exert very weak control over event ordering and the timing when changes are actually implemented. Inadequate control of the correctness of SDN programming is likely to be a significant growth inhibitor for SDN. The authors draw a parallel to databases where the programmer is able to ignore practical considerations related to concurrency and synchronization of distributed components. The work in [34] attempts to abstract the details related to the synchronization of multiple switches' flow tables using established models from the database world.

**DISCUSSION QUESTION**

In Section 7.6 we introduced a programming language named P4. How does P4 relate to the research discussed here in Section 15.3.2?

In this section we have presented a number of examples applying advanced programming methodologies to SDN networks. The fact that SDN inherently facilitates network abstraction is a key enabler to each example. Beyond the samples presented here, there are higher-level programming abstractions possible with SDN. We refer the interested reader to the work done in [35] using the *Procera* language, that using the *Pyretic* language in [36], and the *intents-based* approach discussed in Section 7.3.5.

### 15.3.3 SECURITY APPLICATIONS

SDN provides a fertile environment for novel work in the area of network resilience and security. A comprehensive survey of research done in this area can be found in [37]. In this section we delve into a selection of examples of new research in this area.

#### *Hiding IP addresses*

Many network attacks are based on identifying active IP addresses in a targeted domain. Protecting hosts by making it impossible for an attacker to identify a host's IP address would be an effective countermeasure. In [38] the authors propose that each protected host be assigned a virtual IP address which is the one exposed to the outside world by DNS lookups. In this method, the OpenFlow controller randomly and at high frequency assigns the virtual IP address to the protected hosts, maintaining the temporary mapping of virtual to physical IP addresses. Earlier systems based on DHCP and NAT exist to change the physical IP address of hosts but these mechanisms change the IP address too infrequently to be a strong defense. In [38] only authorized hosts are allowed to penetrate through to the physical IP address. The translation from the virtual IP address to the physical IP address happens at an OpenFlow switch immediately adjacent to the protected hosts. The unpredictability and speed of the virtual IP address mutation is key to thwarting the attacker's knowledge of the network and the planning of the attacks. While this approach could conceivably be implemented using specialized appliances in a conventional network, OpenFlow provides a flexible infrastructure that makes this approach tractable.

### *Flowguard*

In [39] authors propose *Flowguard*, an SDN firewall capable of filtering not only at the packet level but also at the flow level. An entire flow may violate policy and should thus be rejected. A key theme of the authors is that flow policy changes may be in conflict with firewall policy and resolving these conflicts is the responsibility of an SDN firewall. Thus the authors assert that an SDN firewall is both a packet filter and also a policy checker. In Flowguard, *flow packet violations* are distinguished from a broader class of *flow policy violations*. Flow policy violations may occur when a new flow is installed or when policy is changed. The simpler of the two cases is when a new flow is installed. This may violate existing policy and may thus be rejected. It is more complicated when flow policy is changed as this change can cause a violation of firewall policy. The authors illustrate this via a simple example of an OpenFlow-based load balancer which may alter packet header fields in order to change flow paths. When flows are modified by this load balancer, it is imperative that the packet header fields modified by the OpenFlow rules do not result in a firewall rule being circumvented. In order to prevent this, an SDN firewall must check that flow policy does not conflict with firewall policy. Flowguard decomposes this policy conflict problem into *entire* violations and *partial* violations. Outcomes may include rejection of the policy update, removal of affected flows or, in the case of a partial violation, specific packet blocking. The authors contrast Flowguard with Veriflow which, as described in Section 15.3.2 does not do its flow analysis in real time. Flowguard, however, does work in real time. The authors provide data that shows that Flowguard performs favorably as compared with Floodlight's built-in firewall.

### *Segregating IPSec traffic in mobile networks*

Wireless providers using LTE secure the user and control data between the base station (eNB) and their network core using IPSec tunnels. These tunnels terminate in the core at a *Serving Gateway* (S-GW). As currently specified, there is a single IPSec tunnel encrypting all services. IPSec tunneling carries significant overhead, and significant efficiency gains are possible if traffic that does not require the security afforded by IPSec can be sent in the clear. In two separate proposals [24, Sections 8.5 and 8.12] the authors suggest using OpenFlow to map between different traffic types and the appropriate level of IPSec security. In [24, Section 8.5] the focus is on making the mapping of individual flows to different IPSec tunnels. In [24, Section 8.12] the emphasis is on distinguishing those traffic types requiring security and to only tunnel those. The authors suggest that YouTube video, social media, and software updates are examples of traffic that does not require IPSec encryption and can thus be sent in the clear. While the authors do not describe how such traffic types would be detected, one possible means of doing so that avoids the Deep Packet Inspection problem is to base the decision on server addresses that belong to YouTube, Facebook, Microsoft software updates, and so on. An additional possible benefit of providing flow-specific treatment to these other traffic types is that it is not always necessary to send separate copies of this kind of traffic from the network core. By inserting a *deduplication* processor closer to the mobile users, this redundant traffic can be cached at that location and a twofold bandwidth savings is realized by eliminating the round trip to the S-GW. By providing the operators control over which flows are tunneled and which are not, this allows operators to monetize their investment by offering different plans with different levels of security.

## 15.3.4 ROAMING IN MOBILE NETWORKS

### *Mobile traffic offload*

In Chapter 9 we presented a number of SDN applications involving traffic steering, where flows are directed toward security systems or load balanced among a set of similar-function servers. In the multiradio environment common for today's mobile operators, a new SDN application is possible in the area of mobile traffic offload. Mobile offload means moving a client *mobile node* (MN) from one *Radio Access Network* (RAN) to another. This may make sense for a number of reasons, but the one most often offered is to shunt the traffic to a RAN where the spectrum is more available or less expensive than the one currently used by the MN. Such offloading has been contemplated for some time by mobile operators but existing approaches have not provided the flexible, fine-grained control offered by Open SDN. In [24, Section 8.3] a use case is described where OpenFlow switches are used in key gateways in the 3GPP architecture. Based on observing flow-related criteria and the location of the MN, an OpenFlow application can redirect the MN's access connection from 3G to a WiFi hotspot, for example. If the MN needs to roam from WiFi to a cellular radio technology the same mechanism may be used. This approach allows operators to flexibly and dynamically apply offloading policies rather than static policies that are not able to adapt to changing network conditions. For example, if a user is currently connected to a heavily loaded WiFi hotspot and located within a lightly loaded LTE cell, it may make sense to do reverse offload and move the user from WiFi back to LTE. Such a decision hinges on observing rapidly changing conditions and could not be put into effect based on static policies. Note that while the basic traffic steering inside the 3GPP packet network is native to OpenFlow here, the control signaling related to the RF aspects of the roam would have to be coordinated with the appropriate 3GPP signaling functions, as those are outside the current scope of OpenFlow.

### *Media-independent handovers*

IEEE 802.21 is an established protocol for media-independent handovers between 802-family *Points of Access* (PoAs). Examples of PoAs are 802.11 access points and 802.16 base stations. The 802.21 *Point of Service* (PoS) is responsible for the messaging to the PoAs to accomplish either make-before-break or break-before-make handovers (roams). A significant part of accomplishing such handovers is to redirect the traffic flow from an MN such that it enters and exits the network from the new PoA. This combined with the fact that OpenFlow is natively media-independent leads to a natural overlap between the role defined for the 802.21 PoS and an OpenFlow controller. This synergy is discussed in [24, Section 8.6] and a proof of concept proposed to design a system where an OpenFlow controller uses 802.21 messages to control roaming between 802.11 access points. The authors suggest that extensions to OpenFlow may be necessary, however.

### *Infrastructure-controlled roaming in 802.11 networks*

In [40] the authors present Odin, an SDN-based framework to facilitate AAA, policy, mobility, and interference management in 802.11 networks. The fact that in 802.11 the client makes the decision regarding with which AP to associate at any given time has presented a major challenge to giving the network explicit control over with which AP a client associates. Controlling the choice and timing of the client-AP association in 802.11 is fundamental to roaming, RF interference and load balancing solutions. The Odin master function, which is an SDN application, controls Odin agents in each of the APs. These agents implement light virtual access points (LVAPs) that appear as physical APs to the client. Each client is assigned a unique LVAP, giving each client a unique BSSID. Since LVAPs may be instantiated on any of the APs under the control of the controller, the client may physically roam to another physical AP by the controller moving its LVAP instantiation to that new physical AP. OpenFlow is used to move the user data flows in accordance with such roaming.

#### *BeHop: SDN for dense WiFi networks*

In BeHop, presented in [41], the authors describe a virtualized WiFi architecture intended to provide improved wireless performance in dense WiFi deployments. The authors contend that much of the chaos that results in dense WiFi environments stems from the fact that in 802.11 it is the client who selects an AP for association. These client decisions are often not globally optimal. BeHop incorporates the notion of a single, personal AP that follows the user wherever he goes. This personal AP is a virtual AP, of course, and in this sense reminiscent of the ideas presented previously in Odin. Since the user is always connected to his own personal SSID, he never formally *roams* in the 802.11 sense. BeHop is implemented on standard Netgear APs running OpenWRT and *Open vSwitch* (OVS). The SDN-based coordination this affords allows for intelligent channel assignments, load-balancing users across APs, and energy savings via the powering-off of redundant APs.

### 15.3.5 TRAFFIC ENGINEERING IN MOBILE NETWORKS

#### *Dynamic assignment of flows to fluctuating backhaul links*

In Section 9.2.2 we presented an example of Open SDN being used for traffic engineering in MPLS networks. In mobile networks, there is a novel opportunity to apply traffic engineering to wireless backhaul infrastructure links. Unlike the backhaul example in Section 15.3.1, the wireless links we refer to here connect switching elements that may carry the traffic of multiple base stations (eNBs). The appeal of wireless backhaul links is growing as the number of base stations grows, the locations of the switching elements become more dynamic, and new backhaul capacity is brought online to a more freely chosen set of locations. A downside of wireless backhaul is that the bandwidth of the wireless backhaul is both more limited and, more importantly, less stable than in its wired counterparts. Current resource management practices are static and do not redirect load dynamically based on short-term fluctuations in wireless capacity.

In [24, Section 8.2] the authors propose that an OpenFlow controller be enabled to be aware of the current available bandwidth on the set of wireless links it is managing. It may be managing a hybrid set of wired and wireless backhaul links. If OpenFlow is responsible for assigning user flows to that set of backhaul links, that assignment can be made as a function of the SLAs specific to each flow. High SLA (higher guarantees) traffic can be steered over wired links and low SLA traffic can be steered over a wireless link. If one wireless link is experiencing temporary spectrum perturbation, then OpenFlow can shunt the traffic over a currently stable wireless link. Note that this proposal requires a mechanism by which the SDN application on the OpenFlow controller be made aware of changes in the wireless bandwidth on each of the wireless links it is managing.

#### *Sharing wireless backhaul across multiple operators*

The IEEE has chartered a group to study how to enable an Open Mobile Network Interface for omni-Range Access Networks (OmniRAN). The omni-Range aspect implies a unified interface to the multiple radio access network types in the 802 family. These include 802.11 and 802.16, among others. The business model behind this is that multiple operators offering a range of RAN technologies in a well-defined geographic area could benefit from a common backhaul infrastructure shared by all the operators for all the different radio technologies. The cost savings of this approach compared to separate backhaul networks for each (operator, RAN) pair is significant. In [24, Section 8.16] the authors argue that as both OpenFlow and OmniRAN are media-independent that there is a natural synergy in applying OpenFlow as the protocol for controlling and configuring the various IEEE 802 nodes in this architecture, as well as using OpenFlow for its native benefits of fine-grained control of network flows and implementation of network policy. This effort would require extension to OpenFlow for controlling and configuring the IEEE 802 nodes.

### *SoftMoW*

SoftMoW [42] proposes a hierarchy of parent and child SDN controllers to more efficiently route cellular traffic. In particular this approach allows more direct paths between two local mobile stations rather than routing their traffic deep into a rigidly organized hierarchy as is done today. Currently the lack of a nearby *Packet Data Network Gateway* (PGW) often results in unnecessarily long and convoluted paths between the endpoints of mobile user traffic streams. The rise of machine-to-machine traffic in such fields as mobile health will only exacerbate this problem. SoftMoW replaces inflexible and expensive hardware devices, such as the PGWs, with SDN switches. The large cellular network is partitioned into independent and dynamic logical regions, where a child SDN controller manages the data plane of each region. Sophisticated, cross-region traffic engineering is accomplished via a label-swapping concept reminiscent of MPLS. Parent SDN controllers program rules that push global labels onto packets corresponding to traffic groups. When packets reach their local region, these global labels are popped and local labels pushed. Traffic remaining local to one region would never obtain global labels.

### *An OpenFlow switch on every smartphone!*

Today's smartphones generally have multiple radios. For example, it is common to see LTE, WiFi, and 3G radios on the same mobile phone. In the existing model the MN chooses which radio to use based on a static algorithm. This algorithm normally routes voice traffic over the best available cellular connection and data over a WiFi connection if it is available. If no WiFi connection is available, then data is sent via the cellular connection. This is a fairly inefficient and brute force use of the available radio spectrum. For example, if a functioning WiFi connection is poor, it may be much more efficient to route data over a functioning LTE connection. If multiple radios are available, it may be wise to use more than one simultaneously for different data flows. Expanding the horizon for OpenFlow applications to a bold new level, in [24, Section 8.8] there is a proposal to install an instance of OVS on every mobile device. This virtual switch would direct flows over the radio most appropriate for the type and volume of traffic as well as the current spectrum availability for the different radio types. The controller for this OVS instance resides at a gateway in the 3GPP architecture and uses its global knowledge of network state to steer flows in the mobile phone over the most appropriate radio according to the bandwidth and current loading of that particular radio spectrum in that time/location.

We have just described a number of examples of SDN-related research in the areas of roaming and traffic engineering in mobile networks. In general the field of mobile networking offers a ripe environment for SDN research and we do not attempt to provide a complete survey of the relevant work here. The interested reader will find other recent publications in this area in [3,24,43–46].

### 15.3.6 ENERGY SAVINGS

When asked about their vision for the future role of SDN, an expert panel [3] from HP, Dell, and NEC indicated that energy savings was an important area where SDN can play an increasing role. Data centers have enormous OPEX costs keeping their massive data warehouses cooled and fully and redundantly powered. Companies that produce compute and storage servers now tout their energy savings relative to their equivalents of only a few years ago. While there have been significant improvements in the energy efficiency of servers, the corresponding gains in networking equipment have been smaller.

Between the cooling and the energy consumed by the compute and storage servers, it is estimated [47] that about 70% of the power is allocated to those resources. Beyond that, however, it is estimated that approximately 10–20% of the total power is spent to power networking gear. Considering the vast sums spent on electric bills in the data center, making a dent in this 10–20% would represent significant savings.

#### *ElasticTree*

One approach to applying OpenFlow to energy savings in the data center, called *ElasticTree*, is described in [47]. The premise of that work is that data centers' networking infrastructure of links and switches is designed to handle an anticipated peak load and is consuming more power than necessary during normal operation. If a means could be found to only power the minimum necessary subset of switches at any moment, there is an opportunity for significant energy savings. The assertion in [47] is that during periods of less than peak load, OpenFlow can be used to shunt traffic around switches that are candidates for being powered off. Such a system assumes an out-of-band mechanism whereby switches may be powered on and off via the OpenFlow application. Such systems exist and are readily integrated with an OpenFlow application. The authors suggest that by varying the number of powered-on switches their system can provide the ability to fine-tune between energy efficiency, performance, and fault tolerance. The novel work in [47] is related to determining what subset of network switches and links need to be powered on to meet an established set of performance and fault tolerance criteria.

#### *Dynamic adjustment of wireless transmit power levels*

A related use case is proposed for wireless backhaul for mobile networks in [24, Section 8.8]. In this case, the mobile operator has the ability to vary the amount of power consumed by the wireless links themselves by varying the transmission power levels. For example, relatively lower traffic loads over a microwave link may require less bandwidth, which may be achieved with a lower transmission power level. If no traffic is flowing over a wireless link, the transmission power may be turned off entirely. As with ElasticTree, the proposal is to use OpenFlow to selectively direct traffic flows over the wireless links such that transmission power levels may be set to globally optimal settings.

#### *More energy efficient switching hardware*

A more direct approach to energy savings in an SDN environment is to directly design more energy efficient switches. Well-known methods of reducing power consumption used on servers are already being applied to switch design. At some point, use of electronic circuitry consumes energy, though. One of the most power-hungry components of a modern switch capable of flow-based policy enforcement is the TCAM. In [48] the authors propose prediction circuitry that allows flow identification of a high percentage of packets in order to avoid the power-hungry TCAM lookup. The prediction logic uses a memory cache that exploits *temporal locality* of packets. This temporal locality refers to the fact that numerous packets related to the same application flow often arrive at a switch ingress port close to one another. Each TCAM lookup that can be circumvented lowers the switch's cumulative power consumption. A different approach to such TCAM-based energy savings is proposed in [49] where the authors suggest organizing the TCAM into blocks such that the search space is segmented in accordance with those blocks. When a particular lookup is initiated by the switch, it is possible to identify that subset of internal blocks that may be relevant to the current search and only power on that subset of the TCAM during that search.

### 15.3.7 SDN-ENABLED SWITCHING CHIPS

In Section 14.9.3 we described two ongoing commercial efforts to build chips that are designed from the ground up to support advanced OpenFlow capability. We asserted in Section 15.1 that the *plateau of productivity* would likely remain elusive until such chips become commercially available. A survey of which vendor is offering what sort of commercial SDN-enhanced switching chip is available in [50]. In addition to these commercial efforts, this is also an active area of research. For example, in [51] the authors show a model of a 256-core programmable network processing unit. This chip is highly programmable and would support the nature and size of flow tables that will be required to exploit the features of OpenFlow 1.3. The main argument of [51] is that such a programmable network processor can be produced and still have the high throughput characteristics more commonly associated with purpose-built, nonprogrammable Ethernet chips. It is reasonable to believe that such a 256 core chip will be commercially available in the not-too-distant future. In [52] the authors describe an existing programmable networking processor with 256 core. While this is a general-purpose processing unit, this provides evidence of the scale that multicore SDN-specific ASICs are likely to achieve in the near future.

In Section 15.3.6 we described a modified ASIC design that would be more energy efficient than existing switching ASICs when performing the flow table lookups required in SDN flow processing. Another advancement is proposed in [53] where the combination of an SDN-enabled ASIC and local general-purpose CPU and memory could be programmed to handle current and yet-unanticipated uses of per-flow counters. This is an important issue, since maintaining and using counters on a per-flow basis is a key part of SDN programming, yet the inability to anticipate all future applications of these counters makes it difficult to build them all into ASIC designs. In [53] the authors describe a hybrid approach where the hardware counters on the ASIC are programmable and may be assigned to different purposes and to different flows. The counter information combines the matching rule number with the byte count of the matched packet. This information is periodically uploaded to the local CPU where the counters are mapped to general-purpose OpenFlow counters. Such a system can enable some of the high-level network programming we describe in Section 15.3.2 whereby complex software triggers based on a counter value being reached can be programmed directly into the general-purpose CPU.

In Chapter 10 we discussed the overlap between SDN and NFV and showed how many network functions can be virtualized in software. We explained how rudimentary *Intrusion Detection Systems* (IDS) can be built from SDN technology. At some point, though, the *Deep Packet Inspection* (DPI) required for some NFV tasks mandates specialized hardware. When this hardware becomes too specialized, the cost savings and simplicity expected of SDN and NFV are sacrificed. Thus it makes sense to enhance a commodity X86 platform such that it has DPI capabilities. In [54] the authors present a novel design for an energy efficient SDN/NFV accelerator chip compatible with low-cost X86 architectures. This is significant in that such an *Application Specific Instruction Set Processor* permits standard X86 software to control high-speed DPI functions.

## 15.4 CONCLUSION

Ironically, one advantage of SDN washing is that it leads to an easy conclusion that SDN is on a path to a permanent and important part of networking technology for decades to come. This is true simply because the term SDN can be stretched to apply to ANY networking technology that involves a modicum of software control. Virtually all networking technologies involve software to some degree; hence, they are all SDN. This flippant analysis is actually not far removed from what appears in some NEM marketing materials. Thus, interpreted that way, SDN's solid hold on the future of networking is ascertained.

Taking a step in a more serious direction, SDN via Overlays has had such success in the data center that we conclude that this technology, in one form or another, is here for the long run. SDN via Opening up the Device is so nascent with so little track record that any prediction about its future is pure speculation. We feel that SDN via APIs is really a stopgap approach designed to prolong the life of existing equipment and, thus, will not be long-lived. To the extent that the APIs evolve and permit a remote and centralized controller to directly program the data plane of the switch, then SDN via APIs begins to approach Open SDN. It is significant that such API evolution is not merely a matter of developing new software interfaces. Direct programming of the data plane depends on an evolution of the data plane itself. The API evolution, then, is irrevocably caught up in the much longer delays customary with ASIC design cycles.

The founding precepts of the SDN movement are embodied in Open SDN. As we have admitted previously, we, the authors, are indeed Open SDN evangelists and we wish to conclude this work with a serious look at its future. We begin our answer by retreating to the analogy of Open SDN compared to the relationship of Linux with the entrenched world of the PC industry, dominated by the incumbent Microsoft and, to a much smaller degree, Apple. While Linux has actually enjoyed tremendous success, it has not come close to displacing the incumbents from their predominant market. Linux's most striking success has come instead in unexpected areas. The incumbents continue to dominate the PC operating system market, notwithstanding the few zealots who prefer the freedom and openness of Linux. (*How many people do you know who run Linux on their laptops?*) Linux has enjoyed great success, however, in the server and embedded OS markets. The amazing success of the Android operating system, is after all, based on Linux. Thus Linux's greatest success has come in areas related to but orthogonal to its original area of application.

In our opinion, the most likely scenario for Open SDN is to follow that Linux trajectory. For a variety of reasons, it will not soon displace the incumbent NEMS in traditional markets as was posited in 2009 through 2012. It will maintain and expand its beachhead in spaces where its technical superiority simply trumps all alternatives. Whether that beachhead remains restricted to those very large and very sophisticated customers for whom technical superiority of the Open SDN paradigm trumps the safety of the warm embrace of the established NEMs, or expands to a larger market because some NEMs manage to adopt Open SDN without sacrificing profits, is too early to tell. There will be applications and use cases such as those cited in Section 15.3 where Open SDN solves problems for which no good solution exists today. In those areas Open SDN will surely gain traction just as Linux has transformed the world of operating systems.

A Chinese proverb states “May you live in interesting times.” For better or worse, due to data centers, cloud, mobility, and the desire to simplify and reduce costs, networking today is in the midst of such “interesting times.” How it all plays out in the next few years is up for debate—but, whatever the outcome, *Software Defined Networking* is certain to be in the middle of it.

## REFERENCES

[1] Gartner. Research methodology. Retrieved from: http://www.gartner.com/it/products/research/methodologies/research_hype.jsp.

[2] Kerner S. OpenFlow inventor Martin Casado on SDN, VMware, and Software Defined Networking Hype [VIDEO]. Enterprise Networking Planet; 2013. Retrieved from: http://www.enterprisenetworkingplanet.com/netsp/openflow-inventor-martin-casado-sdn-vmware-software-defined-networking-video.html.

[3] Karimzadeh M, Sperotto A, Pras A. Software defined networking to improve mobility management performance. In: Monitoring and Securing Virtualized Networks and Services, Lecture Notes in Computer Science; vol. 8508. Berlin/Heidelberg: Springer; 2014. p. 118–22.

[4] Russell A. The Internet that wasn't. IEEE Spectr 2013;50(8):38–43.

[5] Khan F. The Top 4 SD-WAN Myths. Network Computing; 2016. Retrieved from: http://www.networkcomputing.com/networking/top-4-sd-wan-myths/1072744350?piddl_msgid=313766.

[6] Viprinet SD-WAN. Viprinet. Retrieved from: https://www.viprinet.com/en/why-viprinet/sd-wan .

[7] Conde D. SD-WAN: the Killer App for enterprise SDN? Network Computing; 2015. Retrieved from: http://www.networkcomputing.com/networking/sd-wan-killer-app-enterprise-sdn/1747434541.

[8] Conran M. Viptela—Software Defined WAN (SD-WAN). Viptela. Retrieved from: http://viptela.com/2015/05/viptela-software-defined-wan-sd-wan/.

[9] Ferro G. Concerns about SD-WAN Standards and Interoperability. Ethereal Mind—Infrastructure; 2015. Retrieved from: http://etherealmind.com/concerns-about-sd-wan-standards-and-interoperability/.

[10] Feamster N. Software defined wide-area networks. Open Networking User Group; 2015. Retrieved from: https://opennetworkingusergroup.com/software-defined-wide-area-networks/.

[11] Open Networking User Group. ONUG Software-Defined WAN use case. SD-WAN Working Group; 2014. https://opennetworkingusergroup.com/wp-content/uploads/2015/05/ONUG-SD-WAN-WG-Whitepaper_Final1.pdf .

[12] Banks E. Questions I'm asking myself about SD-WAN solutions. Ethan Banks on Technology; 2015. Retrieved from: http://ethancbanks.com/2015/02/13/questions-im-asking-myself-about-sd-wan-solutions/.

[13] Networking Field Day 9. Silicon Valley: 2015. Retrieved from: http://techfieldday.com/event/nfd9/.

[14] Cloudgenix. 13 Interesting questions about SD-WAN; 2015. Retrieved from: http://www.cloudgenix.com/13-interesting-questions-about-sd-wan/.

[15] Prabagaran R. Answering the profound SD-WAN questions that bother Ethan Banks. Viptela; 2015. Retrieved from: http://viptela.com/2015/03/answering-the-profound-sd-wan-questions-posed-by-ethan-banks/.

[16] Herbert J. Riverbed's take on SD-WAN. Network Computing; 2015. Retrieved from: http://www.networkcomputing.com/networking/riverbeds-take-sd-wan/1591748717.

[17] Cisco Systems. Intelligent WAN technology design guide; 2015. Retrieved from: http://www.cisco.com/c/dam/en/us/td/docs/solutions/CVD/Jan2015/CVD-IWANDesignGuide-JAN15.pdf.

[18] Cisco Meraki. IWAN Deployment Guide. Retrieved from: https://documentation.meraki.com/MX-Z/Deployment_Guides/IWAN_Deployment_Guide.

[19] Farinacci D, Fuller V, Meyer D, Lewis D. The Locator/ID Separation Protocol (LISP). RFC 6830. Internet Engineering Task Force; 2013.

[20] Hot topics in Software Defined Networking (HotSDN). In: SIGCOMM 2012. Helsinki; 2012. Retrieved from: http://conferences.sigcomm.org/sigcomm/2012/hotsdn.php.

[21] ACM SIGCOMM workshop on hot topics in Software Defined Networking (HotSDN). In: SIGCOMM 2013. Hong Kong; 2013. Retrieved from: http://conferences.sigcomm.org/sigcomm/2013/hotsdn.php.

[22] Nunes B, Mendonca M, Nguyen X, Obraczka K, Turletti T. A survey of software-defined networking: past, present and future of programmable networks. IEEE Commun Surv Tutorials 2014;16(3):1617–34.

[23] Wang G, Ng T, Shaikh A. Programming your network at run-time for Big Data applications. In: HotSDN. Helsinki, Finland; 2012. Retrieved from: http://conferences.sigcomm.org/sigcomm/2012/paper/hotsdn/p103.pdf.

[24] Schulz-Zander J, Sarrar N, Schmid S. Towards a scalable and near-sighted control plane architecture for WiFi SDNs. In: Proceedings of the Third Workshop on Hot Topics in Software Defined Networking (HotSDN '14). New York, NY: ACM; 2014. p. 217–8. doi:10.1145/2620728.2620772.

[25] Shenker S. The future of networking, and the past of protocols. Open Networking Summit. Stanford University; 2011.

[26] Handigol N, Heller B, Jeyakumar V, Mazieres D, McKeown N. Where is the Debugger for my Software-Defined Network? In: HotSDN. Helsinki, Finland; 2012. Retrieved from: http://conferences.sigcomm.org/sigcomm/2012/paper/hotsdn/p55.pdf.

[27] Canini M, Venzano D, Peresini P, Kostic D, Rexford J. A nice way to test openflow applications. In: 9th USENIX Symposium on Networked System Design and Implementation. San Jose, CA; 2012. Retrieved from: https://www.usenix.org/system/files/conference/nsdi12/nsdi12-final105.pdf.

[28] Scott C, Wundsam A, Raghavan B, Panda A, Or A, Lai J, et al. Troubleshooting blackbox SDN control software with minimal causal sequences. In: Proceedings of the 2014 ACM conference on SIGCOMM (SIGCOMM'14). New York, NY: ACM; 2014. p. 395–406. Retrieved from: http://people.eecs.berkeley.edu/∼rcs/research/sts.pdf.

[29] Khurshid A, Zhou W, Caesar M, Godfrey PB. VeriFlow: verifying network-wide invariants in real time. In: HotSDN. Helsinki, Finland; 2012. Retrieved from: http://conferences.sigcomm.org/sigcomm/2012/paper/hotsdn/p49.pdf.

[30] Chen C, Limin J, Wenchao Z, Thau LB. Proof-based verification of Software Defined Networks. Santa Clara, CA: Open Networking Summit; 2014. Retrieved from: http://www.archives.opennetsummit.org/pdf/2014/Research-Track/ONS2014_Chen_Chen_RESEARCH_TUE1.pdf.

[31] Kazemian P, Varghese G, McKeown N. Header space analysis: static checking for networks. In: Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation (NSDI'12). Berkeley, CA: USENIX Association; 2012. Retrieved from: https://www.usenix.org/system/files/conference/nsdi12/nsdi12-final8.pdf.

[32] Al-Shaer E, Al-Haj S. FlowChecker: configuration analysis and verification of federated openflow infrastructures. In: Proceedings of the 3rd ACM workshop on assurable and usable security configuration (SafeConfig '10). New York, NY: ACM; 2010. p. 37–44.

[33] Monaco M, Michel O, Keller E. Applying operating system principles to SDN controller design. In: Twelfth ACM Workshop on Hot Topics in Networks. SIGCOMM 2013. College Park, MD; 2013. Retrieved from: http://conferences.sigcomm.org/hotnets/2013/papers/hotnets-final97.pdf.

[34] Wang A, Zhou Y, Godfrey B, Caesar M. Software-defined networks as databases. Santa Clara, CA: Open Networking Summit; 2014. Retrieved from: http://www.archives.opennetsummit.org/pdf/2014/Research-Track/ONS2014_Anduo_Wang_RESEARCH_TUE1.pdf.

[35] Voellmy A, Kim H, Feamster N. Procera: a language for high-level reactive network control. In: Proceedings of the first workshop on hot topics in software defined networks (HotSDN '12). New York, NY: ACM; 2012.

[36] Monsanto C, Reich J, Foster N, Rexford J, Walker D. Composing software-defined networks. In: Proceedings of the 10th USENIX conference on Networked Systems Design and Implementation (NSDI'13). Berkeley, CA: USENIX Association; 2013.

[37] Silva AS, Smith P, Mauthe A, Schaeffer-Filho A. Resilience support in software-defined networking: a survey. Comput Netw 2015;92(1):189–207.

[38] Jafarian JH, Al-Shaer E, Duan Q. OpenFlow random host mutation: transparent moving target defense using Software Defined Networking. In: HotSDN. Helsinki, Finland; 2012. Retrieved from: http://conferences.sigcomm.org/sigcomm/2012/paper/hotsdn/p127.pdf.

[39] Hu H, Ahn G, Han W, Zhao Z. Towards a reliable SDN firewall. Santa Clara, CA: Open Networking Summit; 2014. Retrieved from: http://www.archives.opennetsummit.org/pdf/2014/Research-Track/ONS%202014_Hongxin_Hu_RESEARCH_MON.pdf.

[40] Suresh L, Schulz-Zander J, Merz R, Feldmann A, Vazao T. Towards programmable enterprise WLANs with Odin. In: HotSDN. Helsinki, Finland; 2012. Retrieved from: http://conferences.sigcomm.org/sigcomm/2012/paper/hotsdn/p115.pdf.

[41] Yiakoumis Y, Bansal M, Katti S, Van Reijendam J, McKeown N. BeHop: SDN for dense WiFi networks. Santa Clara, CA: Open Networking Summit; 2014. Retrieved from: http://www.archives.opennetsummit.org/pdf/2014/Research-Track/ONS2014_Yiannis_Yiakoumis_RESEARCH_TUE1.pdf.

[42] Moradi M, Li L, Mao Z. SoftMoW: a dynamic and scalable software defined architecture for cellular WANs. Santa Clara, CA: Open Networking Summit; 2014. Retrieved from: http://www.archives.opennetsummit.org/pdf/2014/Research-Track/ONS2014_Mehrdad_Moradi_RESEARCH_TUE2.pdf.

[43] Pupatwibul P, Banjar A, Sabbagh A, Braun R. Developing an application based on OpenFlow to enhance mobile IP networks. In: IEEE 38th Conference on Local Computer Networks Workshops (LCN Workshops); 2013. p. 936–40.

[44] Namal S, Ahmad I, Gurtov A, Ylianttila M. Enabling secure mobility with OpenFlow. In: IEEE SDN for Future Networks and Services (SDN4FNS); 2013. p. 1–5.

[45] Al-Sabbagh A, Pupatwibul P, Banjar A, Braun R. Optimization of the OpenFlow controller in wireless environments for enhancing mobility. In: IEEE 38th conference on Local Computer Networks workshops (LCN workshops); 2013. p. 930–5. doi:10.1109/LCNW.2013.6758534.

[46] Li Y, Wang H, Liu M, Zhang B, Mao H. Software defined networking for distributed mobility management. In: IEEE Globecom workshops (GC Wkshps); 2013. p. 885–9. doi:10.1109/GLOCOMW.2013.

[47] Heller B, Seetharaman S, Mahadevan P, Yiakoumis Y, Sharma P, Banerjee S, et al. ElasticTree: saving energy in data center networks. In: 7th USENIX Symposium on Networked System Design and Implementation. San Jose, CA; 2010. Retrieved from: https://www.usenix.org/legacy/event/nsdi10/tech/full_papers/heller.pdf.

[48] Congdon PT, Mohapatra P, Farrens M, Akella V. Simultaneously reducing latency and power consumption in openflow switches. IEEE/ACM Trans Netw 2014;22:1007–20.

[49] Ma Y, Banerjee S. A smart pre-classifier to reduce power consumption of TCAMs for multi-dimensional packet classification. In: SIGCOMM 2012. Helsinki; 2012. Retrieved from: http://conferences.sigcomm.org/sigcomm/2012/paper/sigcomm/p335.pdf.

[50] Johnson S. Primer: a new generation of programmable ASICs. Tech Target. Retrieved from: http://searchnetworking.techtarget.com/feature/Primer-A-new-generation-of-programmable-ASICs.

[51] Pongracz G, Molnar L, Kis ZL, Turanyi Z. Cheap silicon: a myth or reality? Picking the right data plane hardware for software defined networking. In: SIGCOMM 2013. Hong Kong: HotSDN; 2013. Retrieved from: http://conferences.sigcomm.org/sigcomm/2013/papers/hotsdn/p103.pdf.

[52] Dupont de Dinechin, B. KALRAY: high performance embedded computing on the MPPA single chip many-core processor. In: CERN Seminar. 2013. Retrieved from: http://indico.cern.ch/getFile.py/access?resId=0&materialId=slides&confId=272037.

[53] Mogul J, Congdon P. Hey, you darned counters! Get off my ASIC! In: SIGCOMM 2012. Helsinki: HotSDN; 2012. Retrieved from: http://conferences.sigcomm.org/sigcomm/2012/paper/hotsdn/p25.pdf.

[54] Yamazaki K, Osaka T, Yasuda S, Ohteru S, Miyazaki A. Accelerating SDN/NFV with transparent offloading architecture. Santa Clara, CA: Open Networking Summit; 2014. Retrieved from: http://www.archives.opennetsummit.org/pdf/2014/Research-Track/ONS2014_Koji_Yamazaki_RESEARCH_TUE2.pdf.