acmqueue

OpenFlow: A Radical New Idea in Networking

An open standard that enables software-defined networking

Thomas A. Limoncelli

Computer networks have historically evolved box by box, with individual network elements occupying specific ecological niches as routers, switches, load balancers, NATs (network address translations), or firewalls. Software-defined networking proposes to overturn that ecology, turning the network as a whole into a platform and the individual network elements into programmable entities. The apps running on the network platform can optimize traffic flows to take the shortest path, just as the current distributed protocols do, but they can also optimize the network to maximize link utilization, create different reachability domains for different users, or make device mobility seamless.

OpenFlow, an open standard that enables software-defined networking in IP networks, is a new network technology that will enable many new applications and new ways of managing networks. Here are three real, though somewhat fictionalized, applications:

EXAMPLE 1: BANDWIDTH MANAGEMENT. A typical wide area network has 30 percent utilization; it must "reserve" bandwidth for "burst" times. Using OpenFlow, however, a system was developed in which internal application systems (consumers) that need bulk data transfer could use the spare bandwidth. Typical uses include daily replication of datasets, database backups, and the bulk transmission of logs. Consumers register the source, destination, and quantity of data to be transferred with a central service. The service does various calculations and sends the results to the routers so they know how to forward this bulk data when links are otherwise unused. Communication between the applications and the central service is bidirectional: applications inform the service of their needs, the service replies when the bandwidth is available, and the application informs the service when it is done. Meanwhile, the routers feed realtime usage information to the central service.

As a result, the network has 90-95 percent utilization. This is unheard of in the industry. The CIO is excited, but the CFO is the one who is really impressed. The competition is paying three times as much CAPEX (capital expenditure) and OPEX (operational expenditure) for the same network capacity.

This example is based on what Google is doing today, as described by Urs Hoelzle, a Google Fellow and senior vice president of technical infrastructure, in his keynote address at the 2012 Open Networking Summit.[1]

EXAMPLE 2: TENANTIZED NETWORKING. A large virtual-machine hosting company has a network management problem. Its service is "tenantized," meaning that each customer is isolated from the others like tenants in a building. As a virtual machine migrates from one physical host to another, the VLAN (virtual LAN) segments must be reconfigured. Inside the physical machine is a software-emulated network that connects the virtual machines. There must be careful coordination between the physical VLANs and the software-emulated world. The problem is that connections can

be misconfigured by human error, and the boundaries between tenants are... tenuous.

OpenFlow allows new features to be added to the VM management system so that it communicates with the network infrastructure as virtual and physical machines are added and changed. This application ensures that each tenant is isolated from the others no matter how the VMs migrate or where the physical machines are located. This separation is programmed end-to-end and verifiable.

This example is based on presentations such as the one delivered at the 2012 Open Networking Summit by Geng Lin, CTO, Networking Business, Dell Inc.[3]

EXAMPLE 3: GAME SERVER. Some graduate students set up a Quake server running on a spare laptop as part of a closed competition. Latency to this server is important not only to gameplay, but also to fairness. Because these are poor students, the server runs on a spare laptop. During the competition someone picks up the laptop and unplugs it from the wired Ethernet. It joins the Wi-Fi as expected. The person then walks with the laptop all the way to the cafeteria and returns an hour later. During this path the laptop changes IP address four times, and bandwidth varies from 100 Mbps on wired Ethernet, to 11 Mbps on an 802.11b connection in the computer science building, to 54 Mbps on the 802.11g connection in the cafeteria. Nevertheless, the game competition continues without interruption.

Using OpenFlow, the graduate students wrote an application so that no matter what subnetwork the laptop is moved to, its IP address will not change. Also, in the event of network congestion, traffic to and from the game server will receive higher priority: the routers will try to drop other traffic before the game-related traffic, thus ensuring smooth gameplay for everyone. The software on the laptop is unmodified; all the "smarts" are in the network. The competition is a big success.

This example is loosely based on the recipients of the Best Demo award at SIGCOMM (Special Interest Group on Data Communications) 2008.[4] More-serious examples of wireless uses of OpenFlow were detailed by Sachin Katti, assistant professor of electrical engineering and computer science at Stanford University, in a presentation at the 2012 Open Networking Summit.[2]

A NETWORK ROUTING SYSTEM THAT LETS YOU WRITE APPS

OpenFlow is either the most innovative new idea in networking, or it is a radical step backward—a devolution. It is a new idea because it changes the fundamental way we think about network-routing architecture and paves the way for new opportunities and applications. It is a devolution because it is a radical simplification—the kind that results in previously unheard of new possibilities.

To understand this paradox, some history is needed. Before the iPhone, there was no app store where nearly anyone could publish an app. If you had a phone that could run applications at all, each app was individually negotiated with each carrier. Vetting was intense. What if an app were to send the wrong bit out the antenna and take down the entire phone system? (Why do we have a phone system that one wrong bit could take down?) The process was expensive, slow, and highly political.

Some fictional (but not *that* fictional) examples:

- "We're sorry but your tic-tac-toe game doesn't fit the '*c'est la mode*' that we feel our company represents."
- "Yes, we would love to have your calorie tracker on our phone, but we require a few critical changes. Most importantly the colors must match our corporate logo."

- "We regret to inform you that we are not interested in having your game run on our smartphone. First, we can't imagine why people would want to play a game on their phones. Our phones are for serious people. Second, it would drain the battery if people used their phones for anything other than phone calls. Third, we find the idea of tossing birds at pigs to be rather distasteful. Our customers would never be interested."

No apps exist for networks today. A niche idea is a distraction from the vendor's roadmap. It could ruin the product's *'c'est la mode'*. Routers are too critical. Router protocols are difficult to design, implementing them requires highly specialized programming skills, and verification is expensive.

BUT WHAT IF IT BECAME EASY?

To understand why writing routing protocols is so difficult requires knowledge of how routing works in today's networks. Networks are made of endpoints (your PC and the server it is talking to) and the intermediate devices that connect them. Between your PC and www.google.com there may be 20 to 100 routers (technically, *routers and switches,* but for the sake of simplicity all packet-passing devices are called *routers* in this article). Each router has many network-interface jacks, or *ports*. A packet arrives at one port, the router examines it to determine where it is trying to go, and sends it out the port that will bring it one *hop* closer to its destination.

When your PC sends a packet, it does not know how to get to the destination. It simply marks the packet with the desired destination, gives it to the next hop, and trusts that this router, and all the following routers, will eventually get the packet to the destination. The fact that this happens trillions of times a second around the world and nearly every packet reaches its destination is awe-inspiring.

No "Internet map" exists for planning the route a packet must take to reach its destination. Your PC does not know ahead of time the entire path to the destination. Neither does the first router, nor the next. Every hop trusts that the next hop will be one step closer and eventually the packet will reach the destination.

How does each router know what the next hop should be? Every router works independently to figure it out for itself. The routers cooperate in an algorithm that works like this: each one periodically tells its neighbors which networks it connects to, and each neighbor accumulates this information and uses it to deduce the design of the entire network. If a router could think out loud, this is what you might hear: "My neighbor on port 37 told me she is 10 hops away from network 172.11.11.0, but the neighbor on port 20 told me he is four hops away. Aha! I'll make a note that if I receive a packet for network 172.11.11.0, I should send it out port 20. Four hops is better than 10!"

Although they share topological information, routers do the route calculations independently. Even if the network topology means two nearby routers will calculate similar results, they do not share the results of the overlapping calculations. Since every CPU cycle uses a certain amount of power, this duplication of effort is not energy efficient.

Fortunately, routers need to be concerned with only their particular organization's network, not the entire Internet. Generally, a router stores the complete *route table* only for its part of the Internet—the company, university, or ISP it is part of (its autonomous domain). Packets destined for outside that domain are sent to *gateway routers*, which interconnect with other organizations. Another algorithm determines how the first set of routers knows where the gateway routers are. The combination of these two algorithms requires less RAM and CPU horsepower than if every router

had to know the entire Internet. If not for this optimization, every router would need enough RAM and CPU horsepower to store an inventory of the entire Internet. The cost would be staggering.

The routing algorithms are complicated by the fact that routers are given clues and must infer what they need to know. For example, the conclusion that "port 20 is the best place to send packets destined for 172.11.11.0" is inferred from clues given by other routers. A router has no way of sending a question to another router. Imagine if automobiles had no windows but you could talk to any driver within 25 feet. Rather than knowing what is ahead, you would have to infer a worldview based on what everyone else was saying. If everyone were using the same vocabulary and the same system of cooperative logical reasoning, then every car could get to where it was going without a collision. That's the Internet: no maps; close your eyes and drive.

Every router is trying to infer an entire worldview based on what it hears. Driving these complicated algorithms requires a lot of CPU horsepower. Each router is an expensive device doing the same calculation as everyone else just to get the slightly different result it needs. Larger networks require more computation. As an enterprise's network grows, every router needs to be upgraded to handle the additional computation. The number and types of ports on the router haven't changed, but the engine no longer has enough capacity to run its algorithms. Sometimes this means adding RAM, but often it means swapping in a new and more expensive CPU. It's a good business model for network vendors: as you buy more routers, you need to buy upgrades for the routers you've already purchased. These aren't standard PCs that benefit from the constant Dell-vs.-HP-vs.-everyone-else price war; these are specialized, expensive CPUs that customers can't get anywhere else.

THE RADICAL SIMPLIFICATION OF OPENFLOW

The basic idea of OpenFlow is that things would be better if routers were dumb and downloaded their routing information from a big "route compiler in the sky," or RCITS. The CPU horsepower needed by a router would be a function of the speed and quantity of its ports. As the network grew, the RCITS would need more horsepower, but the routers would not. The RCITS would not have to be vendor specific, and vendors would compete to make the best RCITS software. You could change software vendors if a better one came along. The computer running the RCITS software could be off-the-shelf commodity hardware that takes advantage of Moore's law, price wars, and all that good stuff.

Obviously, it isn't that simple. You have to consider other issues such as redundancy, which requires multiple RCITS and a failover mechanism. An individual router needs to be smart enough to know how to route data between it and the RCITS, which may be several hops away. Also, the communications channel between entities needs to be secure. Lastly, we need a much better name than RCITS.

The OpenFlow standard addresses these issues. It uses the term *controller* or *controller platform* (yawn) rather than RCITS. The controller takes in configuration, network, and other information and outputs a different blob of data for each router, which interprets the blob as a decision table: packets are selected by port, MAC address, IP address, and other means. Once the packets are selected, the table indicates what to do with them: drop, forward, or mutate then forward. (This is a gross oversimplification; the full details are in the specification, technical white papers, and presentations at http://www.OpenFlow.org/wp/learnmore.)

Traditional networks can choose a route based on something other than the shortest path—

perhaps because it's cheaper, has better latency, or has spare capacity. Even with systems designed for such *traffic engineering*, such as MPLS TE (Multiprotocol Label Switching Traffic Engineering), it's difficult to manage effectively with any real granularity. OpenFlow, in contrast, enables you to program the network for fundamentally different optimizations on a per-flow basis. That means latency-sensitive traffic can take the fastest path, while bulk flows can take the cheapest. Rather than being based on particular endpoints, OpenFlow is granular down to the type of traffic coming from each endpoint.

OpenFlow doesn't stop at traffic engineering, however, because it is capable of doing more than populating a forwarding table. Because an OpenFlow-capable device can also rewrite the packets, it can act as a NAT or AFTR (address family transition router); and because it can drop packets, it can act as a firewall. It can also implement ECMP (equal-cost multi-path) or other load-balancing algorithms. Routers don't have to have the same role for every flow going through them; they can load-balance some, firewall others, and rewrite a few. Individual network elements are thus significantly more flexible in an OpenFlow environment, even as they shed some responsibilities to the centralized controller.

OpenFlow is designed to be used entirely within a single organization. All the routers in an OpenFlow domain act as one entity. The controller has godlike power over all the routers it controls. You wouldn't let someone outside your sphere of trust have such access. An ISP may use OpenFlow to control its routers, but it would not extend that control to the networks of customers. An enterprise might use OpenFlow to manage the network inside a large data center and have a different OpenFlow domain to manage its WAN. ISPs may use OpenFlow to run their patch of the Internet, but it is not intended to take over the Internet. It is not a replacement for inter-ISP systems such as BGP (Border Gateway Protocol).

The switch to OpenFlow is not expected to happen suddenly and, as with all new technologies, may not happen at all.

Centralizing route planning has many benefits:

- **TAKES ADVANTAGE OF MOORE'S LAW.** Not only are general-purpose computers cheaper and faster, but there is more variety. In the Google presentation at the 2012 Open Networking Summit, Urs Höelzle said that Google does its route computations using the Google compute infrastructure.
- **OFFERS DEEPER INTEGRATION.** End-to-end communication can occur directly from the applications all the way to the router. Imagine if every Web-based service in your enterprise could forward bandwidth requirements to the controller, which could then respond with whether or not the request could be satisfied. This would be a radical change over the "send and pray" architectures in use today.
- **TURNS NETWORK HARDWARE INTO A COMMODITY.** The CPU and RAM horsepower required by the device is a function of the speeds and number of ports on the device as shipped. Therefore, it can be calculated during design, eliminating the need to factor in slack. Also, designing and manufacturing a device with a fixed configuration (not upgradable) is less expensive.
- **MAKES ALGORITHMS SIMPLER.** Rather than making decisions based on inference and relying on cooperating algorithms, more-direct algorithms can be used. A dictatorship is the most efficient form of government. Suppose the VoIP phones of the campus EMS team should always get the bandwidth they need. It is easier to direct each router on campus to give EMS phones priority than it is to develop an algorithm whereby each router infers which devices are EMS, verifies a trust

model, confirms that trust, and allocates the bandwidth—and hopes that the other routers are doing the same thing.

- **ENABLES APPS.** The controller can have APIs that can be used by applications. This democratizes router features. Anyone (with proper authorization and access) can create network features. No open source ecosystem exists for applications that run within the Cisco IOS operating system. Creating one will be much easier in the OpenFlow world. Apps will not control individual routers (this is already possible) but the entire network as a single entity.
- **ALLOWS GLOBAL OPTIMIZATION AND PLANNING.** Current routing protocols require each router to optimize independently, often resulting in a routing plan that is optimal locally but not globally. The OpenFlow controller can plan based on a global, mostly omniscient, understanding of the network.
- **PROVIDES CENTRALIZED CONTROL.** Not that today's routers don't have any APIs, but the applications would need to communicate with the API of every router to get anything done, and I don't know any network engineer who is open to that. With OpenFlow, apps can talk to the controller where authentication, authorization, and vetting can happen in one place rather than on every router.

CONCLUSION

In the past, smartphone vendors carefully controlled which applications ran on their phones and had perfectly valid reasons for doing so: quality of apps, preventing instability in the phone network, and protection of their revenue streams. We can all see now that the paradigm shift of permitting the mass creation of apps did not result in a "wild west" of chaos and lost revenue. Instead, it inspired entirely new categories of applications and new revenue streams that exceed the value of the ones they replaced. Who would have imagined Angry Birds or apps that monitor our sleep patterns to calculate the optimal time to wake us up? Apps are crazy, weird, and disruptive—and I can't imagine a world without them.

OpenFlow has the potential to be similarly disruptive. The democratization of network-based apps is a crazy idea and will result in weird applications, but someday we will talk about the old days and it will be difficult to imagine what it was once like.

REFERENCES

1. Hoelzle, U. 2012. Keynote speech at the Open Networking Summit; http://www.youtube.com/watch?v=VLHJUfgxEO4.
2. Katti, S. 2012. OpenRadio: virtualizing cellular wireless infrastructure. Presented at the Open Networking Summit; http://opennetsummit.org/talks/ONS2012/katti-wed-openradio.pdf.
3. Lin, G. 2012. Industry perspectives of SDN: technical challenges and business use cases. Presentation at the Open Networking Summit; http://opennetsummit.org/talks/ONS2012/lin-tue-usecases.pdf (slides 6-7).
4. SIGCOMM Demo. 2008; http://www.openflow.org/wp/2008/10/video-of-sigcomm-demo/.

LOVE IT, HATE IT? LET US KNOW

feedback@queue.acm.org

THOMAS A. LIMONCELLI is an internationally recognized author, speaker, and system administrator. His best-known books include *Time Management for System Administrators* (O'Reilly, 2005) and *The Practice of System and Network Administration,* 2nd edition (Addison-Wesley, 2007). In 2005 he received the SAGE Outstanding Achievement Award. See his blog at http://EverythingSysadmin.com.