A Purpose-built Global Network: Google's Move to SDN

case study

A DISCUSSION WITH AMIN VAHDAT, DAVID CLARK, AND JENNIFER REXFORD

Everything about Google is at scale, of course—a market cap of legendary proportions, an unrivaled talent pool, enough intellectual property to keep armies of attorneys in Guccis for life, and, oh yeah, a private WAN (wide area network) bigger than you can possibly imagine that also happens to be growing substantially faster than the Internet as a whole.

Unfortunately, bigger isn't always better, at least not where networks are concerned, since along with massive size come massive costs, bigger management challenges, and the knowledge that traditional solutions probably aren't going to cut it. And then there's this: specialized network gear doesn't come cheap.

Adding it all up, Google found itself on a cost curve it considered unsustainable. Perhaps even worse, it saw itself at the mercy of a small number of network equipment vendors that have proved to be slow in terms of delivering the capabilities requested by the company. Which is why Google ultimately came to decide it should take more control of its own networking destiny. That's when being really, really big proved to be a nice asset after all, since being at Google scale means you can disrupt markets all on your own.

So this is the story of what Google ended up doing to get out of the box it found itself in with its backbone network. Spoiler alert: SDNs (software-defined networks) have played a major role. Amin Vahdat, Google's tech lead for networking,

helps us tell the story of how that's all played out. He's both a Distinguished Engineer and a Google Fellow. Somehow he also manages to find time to teach at UC San Diego and Duke.

Jennifer Rexford, a computer science professor at Princeton renowned for her expertise in SDN, also contributes to the discussion, drawing on her early work designing SDN-like architectures deployed in AT&T's backbone network, as well as her recent research on novel programming abstractions for SDN controller platforms.

Finally, most of the questions that drive this discussion come courtesy of David Clark, the Internet pioneer who served as chief protocol architect of the network throughout the 1980s. He is also a senior research scientist at the MIT Computer Science and Artificial Intelligence Laboratory, where he has been working for nearly 45 years.

DAVID CLARK I wonder if people have a full appreciation for the scale of your private wide area network.

AMIN VAHDAT Probably not. I think our private-facing WAN is among the biggest in the world, with growth characteristics that actually outstrip the Internet. Some recent external measurements indicate that our backbone carries the equivalent of 10 percent of all the traffic on the global Internet. The rate at which that volume is growing is faster than for the Internet as a whole.

This means the traditional ways of building, scaling, and managing wide area networks weren't exactly optimized or targeted for Google's use case. Because of that, the amount of money we had been allocating to our private WAN, both

in terms of capital expenditures and operating expenses, had started to look unsustainable—meaning we really needed to get onto a different curve. So we started looking for a different architecture that would offer us different properties.

We actually had a number of unique characteristics to take into account there. For one thing, we essentially run two separate networks—a public-facing one and a private-facing one that connects our data centers worldwide. The growth rate on the private-facing network has exceeded that on the public one; yet the availability requirements weren't as strict, and the number of interconnected sites to support was actually relatively modest.

In terms of coming up with a new architecture, from a traffic-engineering perspective, we quickly concluded that a centralized view of global demand would allow us to make better decisions more rapidly than would be possible with a fully decentralized protocol. In other words, given that we control all the elements in this particular network, it would clearly be more difficult to reconstruct a view of the system from the perspective of individual routing and switching elements than to look at them from a central perspective. Moreover, a centralized view could potentially be run on dedicated servers—perhaps on a number of dedicated servers, each possessing more processing power and considerably more memory than you would find with the embedded processors that run in traditional switches. So the ability to take advantage of general-purpose hardware became something of a priority for us as well. Those

considerations, among many others, ultimately led us to an SDN architecture.

JENNIFER REXFORD I would add that SDN offers network-wide visibility, network-wide control, and direct control over traffic in the network. That represents a significant departure from the way existing distributed control planes work, which is to force network administrators to coax the network into doing their bidding. Basically, what I think Google and some other companies find attractive about SDN is the ability to affect policy more directly from a single location with one view of the network as a whole.

DC When did you first start looking at this?

AV We started thinking about it in 2008, and the first implementation efforts probably kicked off in 2009, with the initial deployment coming in 2010.

DC What were the features of SDN your engineers found most appealing as they were first trying to solve these problems back in 2008?

AV Starting with the caveat that everything is bound to look a lot wiser in retrospect, I think the best way to answer that would be to talk about why we weren't satisfied with the prevailing architectures at the time. Our biggest frustration was that hardware and software were typically bundled together into a single platform, which basically left you at the mercy of certain vendors to come up with any of the new features you needed to meet requirements already confronting you. So if we bought a piece of hardware from a vendor to handle our switching and routing, we would then also be dependent on that vendor to come up with any new

protocols or software capabilities we might need later.

That was a huge issue for us since we already were playing in a high-end, specialized environment that required specialized platforms—meaning exorbitantly expensive platforms since the big vendors would quite naturally want to recoup their substantial engineering investments over the relatively small number of units they would have any hope of selling.

What's more, buying a bundled solution from a vendor meant buying all the capabilities any customer of that vendor might want, with respect to both hardware and software. In many cases, this was overkill for our use cases. I should probably add we initially were looking only to provide for high-volume but relatively low-value traffic. This probably helps explain why we didn't want to invest in totally bulletproof, ironclad systems that offered state-of-the-art fault tolerance, the most elaborate routing protocols, and all the other bells and whistles. Over time, this evolved as we started moving higher-value traffic to the network. Still, our underlying philosophy remains: add support as necessary in the simplest way possible, both from a features and a management perspective.

Another big issue for us was that we realized decentralized protocols wouldn't necessarily give us predictability and control over our network, which at the time was already giving us fits in that convergence of the network to some state depended on ordering events that had already occurred across the network and from one link to another—meaning we had little to no control over

the final state the system would wind up in. Certainly, we couldn't get to a "global optimum," and beyond that, we couldn't even predict which of the many local optimums the system might converge to. This made network planning substantially harder. It also forced us to overprovision much more than we wanted. Now, mind you, I don't think any of these considerations are unique to Google.

And here's another familiar pain point that really bothered us—one I'm sure you'll have plenty of perspective on, Dave—and that is, we were tired of being at the mercy of the IETF (Internet Engineering Task Force) standardization process in terms of getting new functionality into our infrastructure. What we really wanted was to get to where we could write our own software so we would be able to get the functionality we needed whenever we needed it.

JR The high-end equipment for transit providers not only has reliability mechanisms that might be more extensive than what was warranted for this particular network, but also offers support for a wide range of link technologies to account for all the different customers, peers, or providers a transit network might ever end up linking to. That's why you'll find everything in there, from serial links to Packet over SONET. Google's private WAN, on the other hand, is far more homogeneous, meaning there's really no need to support such a wide range of line-card technologies. Moreover, since there's no need for a private WAN to communicate with the global Internet, support for large routing tables was also clearly unnecessary. So, for any number of reasons, the sorts of boxes the big carriers might

be looking to purchase clearly would have been a poor fit for Google from the perspective of both line-card diversity and routing scalability.

DC I can certainly see it wouldn't be cost effective to buy commercial high-end routers.

AV What's interesting is that even Cisco and Juniper are now increasingly starting to leverage commodity silicon, at least for their lower-end data-center products.

DC Aren't you building your own routers?

AV Well, they're routers in the sense they provide for external BGP (Border Gateway Protocol) peering, but they would never be mistaken for Cisco routers. Yet we've found we can achieve considerable cost savings by building just for what we need without taking on support for every single protocol ever invented.

DC Also, there's that matter of centralized control. It's my sense that some people have an overly simplistic view of what SDN offers there, in that they imagine you have a bunch of routers and a centralized controller, but what you actually have is more sophisticated than that. In fact, as I understand it, there's a hierarchy of control in this network, with one controller for each site.

AV That's correct.

DC And that's not running a peer-to-peer distributed algorithm either. You get conceptually centralized control, but it's realized in a fairly sophisticated way. So that raises two questions. First, is that mental model generally in keeping with the level of complexity SDN is going to involve in practice? Also, just how much of that did you end

up building yourself? My impression is you had to code a considerable amount of the controller, which seems like quite a price to pay to avoid getting trapped by standards.

AV Yes, we built a huge amount of the infrastructure and wrote all the software. We also collaborated with some people externally. But I'd say we managed to do that with a moderate-size team—not small, but certainly nothing like a software team at a major vendor. Again, that's because we purpose-built our infrastructure.

Still, I would agree it's not a simple system. Some of the complexity involved in maintaining hierarchical, multilevel control is inherent, given the need to isolate failure domains. I won't say SDN is necessarily simpler than the existing architectures, but I do think it offers some distinct advantages in terms of enabling rapid evolution, greater specialization, and increased efficiency.

JR For all the talk about where this might lead, I notice that in the SIGCOMM paper where you describe this network [B4: Experience with a Globally Deployed Software-Defined WAN, 2013], you also talk about all the effort made to incorporate IS-IS (Intermediate System to Intermediate System) and BGP as part of the solution. That struck me as strange, given that each of the endpoints within the B4 network or connected to it is under Google's control—meaning you clearly could have chosen not to use any legacy protocols whatsoever. What value did you see in holding onto them?

AV That actually was a critically important decision, so I'm glad you brought it up. We decided on an incremental

deployment strategy after much consideration, and that's something we wanted to emphasize for the benefit of ISPs when we were writing that SIGCOMM paper.

The question was: did we want to have a flag day where we flipped all our data centers over to SDN in one fell swoop? Or did we want to do it one data center at a time while making it look like everything was just the same as ever to all the other sites? So you could say we ended up making huge investments just to re-create what we already had, only with a less mature system. That took quite a while.

Also, there was a fair amount of time where we had only baseline SDN—without any traffic engineering—deployed. Basically, that was the case throughout the whole period we were bringing up SDN one data center at a time. I still think that was the right approach since it gave us an opportunity to gain some much-needed experience with SDN.

So, while I agree that BGP and IS-IS are not where we want to be long term, they certainly have provided us with a critical evolution path to move from a non-SDN network to an SDN one.

DC You're making some really important points here. For a large ISP like Comcast, for example, the equivalent of doing one data center at a time might be focusing on just one metropolitan area at a time. But even just transitioning a single metropolitan area would be complicated enough, so it would be good to think in terms of approaching that incrementally such that they could always fall back to stuff known to work, like shortest-path routing.

AV Oh, yes, I view that as critically important. You really need

to have that sort of hybrid deployment model. In fact, even now we continue to have a big red button that lets us fall back to shortest-path routing should we ever feel the need to do so. And that's not even taking long-term considerations like backward compatibility into account. It's just that whenever you're deploying for any system as large and complex as the Internet—or our private WAN for that matter—it's really, really important to take a hybrid approach.

As with any pioneering effort, Google's push into software-defined networking has come with a number of risks—most notably the potential for breaking the Internet's time-honored fate-sharing principle, along with its established mechanism for distributed consensus. For any network engineer schooled over the past few decades, this ought to be more than enough to set off alarms, since it has been long accepted that any scenario that could potentially lead to independent failures of the brain and body might easily result in bizarre failure patterns from which recovery could prove tremendously difficult. But now, after a careful reexamination of the current Internet landscape, it appears these are risks that could actually be mitigated through the combination of centralized control and a bit of clever traffic engineering.

DC In rolling out the network, what was the biggest risk you faced?

AV Probably what most concerned me was that we were breaking the fate-sharing principle—which is to say we were

putting ourselves in a situation where either the controller could fail without the switch failing, or the switch could fail without the controller failing. That generally leads to big problems in distributed computing, as many people learned the hard way once remote procedure calls became a dominant paradigm.

DC I find your comments about fate sharing somewhat amusing since back when we started doing the Internet, we were quite critical of the telephone company because it didn't really have a good system for dynamic routing. So it would gold-plate all its technology and then run with these stupid, feeble routers that crashed all the time, since that basically was all that was available back then. Dynamic routing was supposed to give us the network resilience we would need to get away with running those crappy routers. But I think what we've learned is that dynamic routing might have been a good idea had the protocols actually proved responsive enough to let people make timely compensating engineering decisions.

In the early days of routing, however, we didn't know how to do any of that. We went with the distributed protocols for the simple reason that they were the only ones we knew how to build. What I mean is that this idea of breaking fate sharing was absolutely terrifying to us since we knew a partition in the network might separate the controller from the switches that needed to be managed.

Basically, if the controller were to lose its view of the network, then there would be no way to reach into the network and put it back together again. Back in those days

we just didn't have any idea how to deal with that. That got us started down the road to the original Internet religious holding that we don't have to make the switches expensively robust if we have a strategy for rebuilding the network once something breaks, assuming that can be done fast enough and effectively enough to let us restore the necessary services.

JR We should also note that in addition to fate sharing, SDN is criticized for breaking distributed consensus, which is where the routers talk amongst themselves to reach agreement on a common view of network state. Anyway, the perception is that distributed consensus might end up getting broken since one or more controllers could get in the way.

But I would just like to say I think both of those battles have already been lost anyway—even before SDN became particularly prominent. That is, I think if you look closely at a current high-end router from Cisco or Juniper, you'll find they also employ distributed-system architectures, where the control plane might be running in a separate blade from the one where the data plane is running. That means those systems, too, are subject to these same problems where the brain and the body might fail independently.

DC Another concern from the old days is that whenever you have to rely on distributed protocols essentially to rebuild the network from the bottom up, you have to realize you might end up with a network that's not exactly the way you would want it once you've taken into account anything other than just connectivity. Basically, that's because we've never been very good at building distributed protocols capable of

doing anything more than simply restoring shortest-path connectivity.

There was always this concern that knowledge of a failure absolutely had to be propagated to the controller so the controller could then respond to it. Mind you, this concern had nothing to do with unplanned transient failures, which I think just goes to show how little we anticipated the problems network managers would actually face down the road. But when you think about it, knowledge of unplanned transient failures really does need to be propagated. Part of what worried us was that, depending on the order in which things failed in the network, the controller might end up not being able to see all that had failed until it actually started repairing things.

That, of course, could lead to some strange failure patterns, caused perhaps by multiple simultaneous failures or possibly just by the loss of a component responsible for controlling several other logical components—leaving you with a Baltimore tunnel fire or something along those lines, where the controller has to construct the net over and over and over again to obtain the topological information required to fix the network and restore it to its previous state. Is that an issue you still face with the system you now have running?

AV Failure patterns like these were exactly what we were trying to take on. As you were saying, the original Internet protocols were focused entirely on connectivity, and the traditional rule of thumb said you needed to overprovision all your links by a factor of three to meet the requirements

of a highly available "network fabric." But at the scale of this particular network, multiplying all the provisioning by three simply was not a sustainable model. We had to find a way out of that box.

DC That gets us back to the need to achieve higher network utilization. One of the things I find really interesting and distinctive is how you've managed to exploit traffic engineering to achieve some very high link loadings. I've always had it in the back of my mind that by identifying classes of traffic, some of which are more tolerant of being slowed down than others, and then employing a bit of traffic engineering and quality of service, you ought to be able to get some higher link loadings just by knowing which traffic can be slowed down. To what extent is SDN actually necessary to accomplish that? Prior to this, it seemed that Google was using DiffServ tags, so I just assumed DiffServ tags could be used to increase link loading by ensuring that latency-sensitive traffic didn't get disrupted. To what extent is traffic engineering dependent on moving to an SDN architecture or at least the SDN approach?

AV That isn't dependent on SDN. There's nothing in that respect that couldn't have been achieved by some other means. I think it really comes down to efficiency and iteration speed. I should add that you were absolutely right in your supposition: DiffServ can indeed be used to increase link loading. Our main concern, though, had to do with failures, and we had no way of predicting how the system would converge. So the overprovisioning was always to protect the latency-sensitive—or, if you will, revenue-generating—

traffic. Basically, for us to hit our SLAs (service-level agreements), that meant overprovisioning to cover worst-case convergence scenarios in a decentralized environment. Upon moving to a centralized environment, however, we found we could actually predict how things were going to converge under failure conditions, which meant we could get away with substantially less overprovisioning across our global network while still managing to hit our SLAs.

JR Plus, you could control exactly what intermediate stage the network goes through when it transitions from one configuration to another, whereas if you let the distributed protocols do it, then all bets are off as to which router ends up going first.

AV Exactly. So I think the total amount of improvement we realized through our centralized scheme relative to a decentralized scheme in steady state actually proved to be relatively modest—let's say a 10, maybe 15, percent improvement in the best case. What proved to be far more important was the predictability under failure, the improved ability to analyze failure conditions, and the means for transitioning the system from one state to another—again in a predictable manner that allowed for the protection of latency-sensitive traffic. That's what really made it possible for us to get away with less overprovisioning.

JR Also, if you're using legacy protocols, even to the extent you can predict what they're going to do, the network management tools you use to make that prediction need essentially to invert the control plane so you can model what it's likely to do once it's poked and prodded in various

ways. But with a centralized network, if you want to be able to predict what's going to happen when you perform planned maintenance or need to deal with some particular failure scenario, you can just run the exact same code the controller is going to run so you'll know exactly what's going to happen.

AV To put this in some perspective, what we've really managed to accomplish is to lay some important groundwork. That is, I think we still have a long road ahead of us, but the traffic-engineering aspect is an important early step on that journey. It's one that drives a lot of capital-expenditure savings, and it's now also an architecture on top of which we'll be able to deliver new functionality more rapidly and under software control—which is to say, we'll be able to deliver that functionality under *our* control. We'll no longer have to wait for someone else to deliver critical functionality to us. Working in small teams, we should be able to deliver substantial functionality in just a matter of months in a tested, reproducible environment and then roll out that functionality globally. Ultimately, I think that's going to be the biggest win of all—and the first demonstration of that is traffic engineering.

Increased autonomy isn't the only win, of course. Significantly improved link loadings and the ability to scale quickly in response to increased demand are two other obvious advantages Google has already managed to realize with its private backbone WAN. In fact, the experience so far with both SDN and centralized management

has been encouraging enough that efforts are under way to take much the same approach in retooling Google's public-facing network. The challenges that will be encountered there, however, promise to be much greater.

DC Getting right to the punch line, what do you see as the biggest improvements you've managed to achieve by going with SDN?

AV Well, as we were saying earlier, through a combination of centralized traffic engineering and quality-of-service differentiation, we've managed to distinguish high-value traffic from the bulk traffic that's not nearly as latency-sensitive. That has made it possible to run many of our links at near 100 percent utilization levels.

DC I think that comment is likely to draw some attention.

AV Of course, our experience with this private-facing WAN hasn't been uniformly positive. We've certainly had our hiccups and challenges along the way. But, overall, it has exceeded all of our initial expectations, and it's being used in ways we hadn't anticipated for much more critical traffic than we had initially considered. What's more, the growth rate has been substantial—larger than what we've experienced with our public-facing network, in fact.

Now, given that we have to support all the different protocol checkbox features and line cards on our public-facing network, our cost structures there are even worse, which is why we're working to push this same approach—not the exact same techniques, but the general approach—into our public-facing network as well. That work is already ongoing, but it will surely be a long effort.

DC What are some of the key differences between the public-facing net and the private net that you'll need to take into account?

AV For one thing, as you can imagine, we have many more peering points in our public-facing network. Our availability requirements are also much higher. The set of protocols we have to support is larger. The routing tables we have to carry are substantially larger—certainly more than a million Internet prefixes and millions of different advertisements from our peers, just for starters. So, basically, as we move from the private net to the public one, the overall number of sites, the size of the traffic exchanges, the robustness required to talk to external peers, and the sorts of interfaces we have to support will all change substantially. That means the public net is clearly a harder problem, but given the understanding we've gained from our experience with the private net, I'd say that undertaking now looks far less daunting than it did a few years ago.

DC Does this suggest any similar sort of transition for the big carriers?

AV What I personally find exciting in that respect are the possibilities for what I call *SDN peering*. BGP takes a distrustful view of the world, but what if individual ISPs—or peers, if you will—decide they want to at least selectively open up some additional information about their networks dynamically? Looking at it naively, I think if they were to share some information about downstream traffic patterns, they would be able to make end-to-end transit times a lot faster and basically improve the user experience

tremendously. By making it possible for the ISPs to use their more lightly loaded paths better, the carriers themselves would also benefit.

JR In general, current routing is strictly destination-based and doesn't consider the nature of applications. You can imagine, then, that SDN might be a great way to let the recipient of traffic reach upstream to say, "No, drop this traffic," or "Rate limit this traffic," or "Route this traffic differently because I can tell you something about the best paths to reach me that the upstream party doesn't know about." Similarly, I might say, "Hey, I want this video traffic to take this other path," or "I want it to pass through this other box," which again is something that's hard to accomplish with today's destination-based forwarding.

AV We've already talked to some customers who are interested in SDN-based peering, and I can tell you they're particularly interested in application-specific peering. They would like to be able to say, "Hey, I want my video traffic to go through this peer, while my non-video traffic goes through this other peer," either for performance or pricing reasons. And that's just awkward to do right now.

DC I think some evidence of how a different technology might enable a win-win here between what are otherwise adversarial interests would actually go a long way toward clarifying some of the business conversations currently going on.

JR One of the challenges for ISPs such as Comcast or AT&T, should they decide they want to move to SDN, is that they have a lot fewer end nodes than Google does. A transit

network really needs to carry full routes, if you will. Also, the big ISPs tend to have tremendous heterogeneity in their edge router equipment, and they don't upgrade everything at the same time either, so some of that equipment might be four or five years old, if not older.

Therefore, SDN deployments for the large carriers are going to be significantly more challenging than what Google has faced. I still think it's a promising direction they should pursue, but for various practical reasons it's just going to take longer for them to get there.

DC Earlier you alluded to some of the traffic-engineering advantages you believe SDN offers. Can you go into a bit of detail about some of the specific challenges you were looking to solve in order to build a more cost-effective network, given your particular set of problems?

AV As far as I can tell, the state of the art in network management still involves logging into individual network switches and managing them through a CLI (command-line interface). That just scales terribly in terms of people costs. It also scales horribly in terms of the myriad network interactions human beings need to keep track of inside their heads when it comes to how some action on one box might end up resulting in ripple effects across the whole network fabric.

DC For somebody who hasn't actually lived in the network operations world, it would be really hard to understand just how bad that can actually be. The idea that people are still programming routers using CLIs is a little mind-boggling. And the very idea that human beings are expected to figure

out the global consequences of what might happen if they should make one little fix here or another little fix there... it's like we never escaped the 1980s!

I'm sure there are some traditional network engineers who take great pride in their ability to keep all that junk in their heads. In fact, I imagine there has been some resistance to moving to higher-level management tools for the same reason some people back in the day refused to program in higher-level programming languages—namely, they were sure they would lose some efficiency by doing so. But when it comes to SDN, I hear you saying the exact opposite—that you can actually become far more efficient by moving to centralized control.

AV True, but change is always going to meet with a certain amount of resistance. One of the fundamental questions to be answered here has to do with whether truth about the network actually resides in individual boxes or in a centrally controlled infrastructure. You can well believe it's a radical shift for some network operators to come around to accepting that they shouldn't go looking for the truth in individual boxes anymore. But that hasn't been an issue for us since we've been fortunate enough to work with a talented—and tolerant—operations team at Google that's been more than willing to take on the challenges and pitfalls of SDN-based management.

Another interesting aspect of making the transition to SDN is that when things break, or at least don't work as you expect them to, unless you have a reasonable mental model of what the controller is trying to do, you might find it very

difficult to diagnose what's going on. In fact, I think one of the advantages network operations people have now when they're working with these protocols they know so well is that—while they may have only a very limited view and so have difficulties diagnosing everything—they at least have a familiar mental model they can work from when they're trying to debug and diagnose problems. Whereas with SDN, whenever things go bump in the night, someone who wasn't involved in writing the software in the first place is probably going to find it a lot more difficult to debug things.

DC This leads to a larger question I hear a lot of people asking now: Do network engineers need to be trained in computer science? Many aren't at this point. While it's one thing to go through the Cisco certification process, one might argue that in an SDN world people might need to pop up a level to master more general computer science concepts, particularly those having to do with distributed systems.

AV I think that's probably a fair comment. But I also think there are lots of very talented network engineers out there who are fully capable of adapting to new technologies.

DC That being said, I think most of those network engineers probably don't currently do a lot of software development. More likely, they just assume they have more of a systems-integration role. It's possible that in the fullness of time, the advocates of SDN will try to supply enough components so that people with systems-integration skills, as opposed to coding skills, will find it easier to use SDN effectively. But I wonder whether, at that point, the complexity of

SDN will have started to resemble the complexity you've been trying to shed by stripping down your network. That is, I wonder whether the tradeoff between writing your own code or instead taking advantage of something that already offers you plenty of bells and whistles is somehow inherent—meaning you won't be able to entirely escape that by migrating to SDN.

AV I would argue that a lot of that has been driven by management requirements. I certainly agree that the Google model isn't going to work for everyone. One of the biggest reasons we've been able to succeed in this effort is because we have an operations team that's supportive of introducing risky new functionality.

With regard to your question about whether we'll truly be able to shed some of the complexity, I certainly hope so. By moving away from a box-centric view of network management to a fabric-centric view, we should be able to make things inherently simpler. Yet I think this also remains the biggest open question for SDN: Just how much progress will we actually realize in terms of simplifying operations management?

JR I think it's natural the two highest-profile early successes of SDN—namely, as a platform for network virtualization and the WAN deployment effort we're talking about here—are both instances where the controller platform, as well as the application that runs on top of the controller, have been highly integrated and developed by the same people. If SDN is going to prove successful in a much broader context—one where you don't have a huge software development team

at your disposal as well as a supportive organization behind you—it's going to be because there are reusable platforms available, along with the ability to build applications on top of those platforms.

Just as important, you would want to believe that many of those apps could come from parties other than those responsible for creating the platforms. We're actually starting to see a lot of innovation in this area right now, with work happening on lots of different controller platforms and people starting to consider abstractions that ought to make it possible to build applications on top.

But even before that happens, there are things SDN brings that perhaps were not critical for Google but likely will prove useful in other settings. One is that SDN could make it possible to scale back on much of the heterogeneity in device interfaces. Many of the companies that work on enterprise network management employ armies of developers just so they can build device drivers that speak at the CLI level with lots of different switches, routers, firewalls, and so on, meaning that a gradual move toward a more standard and open interface for talking to devices ought to go far to reduce some of the low-level complexity of automating management.

Beyond that, I think Google's design demonstrates that if you can separate the distributed management of state required for your network control logic from the network control logic itself, you can avoid reinventing the wheel of how to do reliable distributed state management while also separating that from every single protocol. Basically,

with each new protocol we design, we reinvent how to do distributed state management. But it turns out the distributed-systems community already has a number of really good reusable solutions for that.

DC Part of what I'm taking away here is that not everything Google did with its private WAN is going to be readily transferable into other operator contexts. There are some good reasons why this approach was an especially good fit for Google, both with regard to Google's specific requirements and the particular skills it has on hand in abundant supply. Also, as Amin noted, it helps that Google has a business culture that's more tolerant when it comes to following paths that initially put resilience and reliability at somewhat greater risk.

JR But I think some of the same cost arguments will ultimately apply to large carriers as well as to many large enterprises, so that might end up serving as an impetus for at least some of those organizations collectively subsidizing the R&D required to develop a suitable suite of SDN products they then could use. Otherwise, they might find themselves on an unsustainable cost curve when it comes to the purchase and operation of new network equipment.

For example, if you look at other domains, like the cellular core, you again find back offices full of exorbitantly expensive equipment that's typically quite brittle. I think you find much the same thing in enterprise. Changes are clearly going to proceed more slowly in those settings since they face much more difficult deployment challenges, far stricter reliability requirements, and maybe even some harder

scaling requirements. I mean, there's a good reason we're seeing SDN surface first in data centers and private WANs. Just the same, I think CAPEX (capital expenditure) and OPEX (operating expense) are ultimately going to prove to be compelling arguments in these other settings as well.

AV If you take it as inevitable, for example, that all video content is going to be distributed across the Internet at some point in the near future, then we're surely looking at some phenomenal network growth, which suggests the large carriers will at minimum soon become quite interested in seizing upon any CAPEX and OPEX savings they possibly can.

LOVE IT, HATE IT? LET US KNOW feedback@queue.acm.org