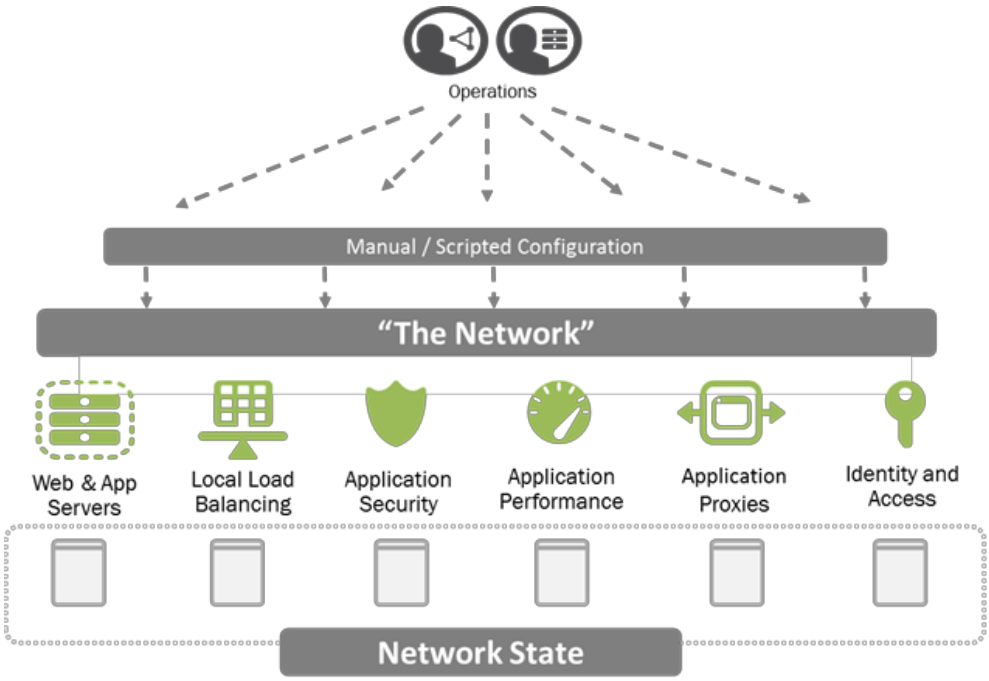
SDN's Eventually Consistent Network Problem

Lori MacVittie, 2014-07-07

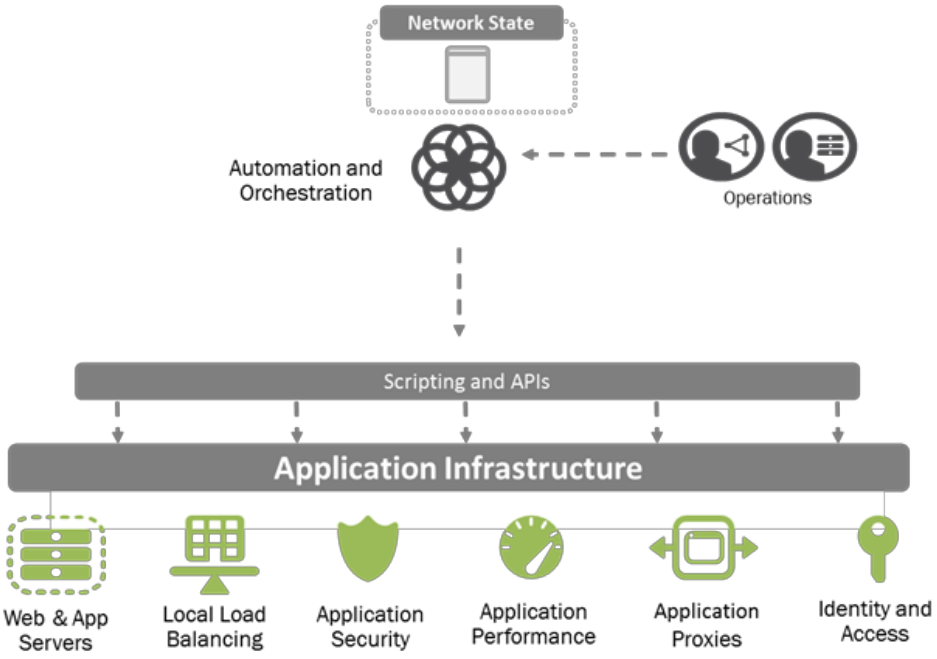
#SDN #OpenDaylight #SDAS Clustering controllers to address scalability concerns introduces a well-understood problem: consistency

One of the benefits of SDN is centralized control. That is, there is a single repository containing the known current state of the entire network. It is this centralization that enables intelligent application of new policies to govern and control the network - from new routes to user experience services like QoS. Because there is a single entity which has visibility into the state of the network as a whole, it can examine the topology at any given point and make determinations as to where this packet and that should be routed, how it is prioritized and even whether or not it is allowed to traverse the network.

It's a pretty powerful concept for networks, which traditionally distribute network state as individual configuration files across the data path.



Most of the focus of SDN is on the replacement of manual and scripted configuration methods with an API-driven mechanism. Whether that's OpenFlow or OpFlex or some other protocol is not really important as the benefit of operationalization is to provide a consistent interface from the perspective of the operator, not the device.



This is a real benefit; operationalization across operations and dev has proven to produce tangible benefits in the form of improved time to market and a reduction in errors. By centralizing network state in a controller, this model provides a comprehensive view of the network at any given moment. Because the controller is not just a repository but an active participant in the flow of data across the network, this visibility enables the controller to understand how to (ostensibly) non-disruptively change routes or apply new policies in real-time.

The benefit itself is not in question. What is in question is what happens when the controller of this new software-defined architecture becomes overwhelmed, and how to preserve that benefit when the centralized model must decentralize in order to scale.

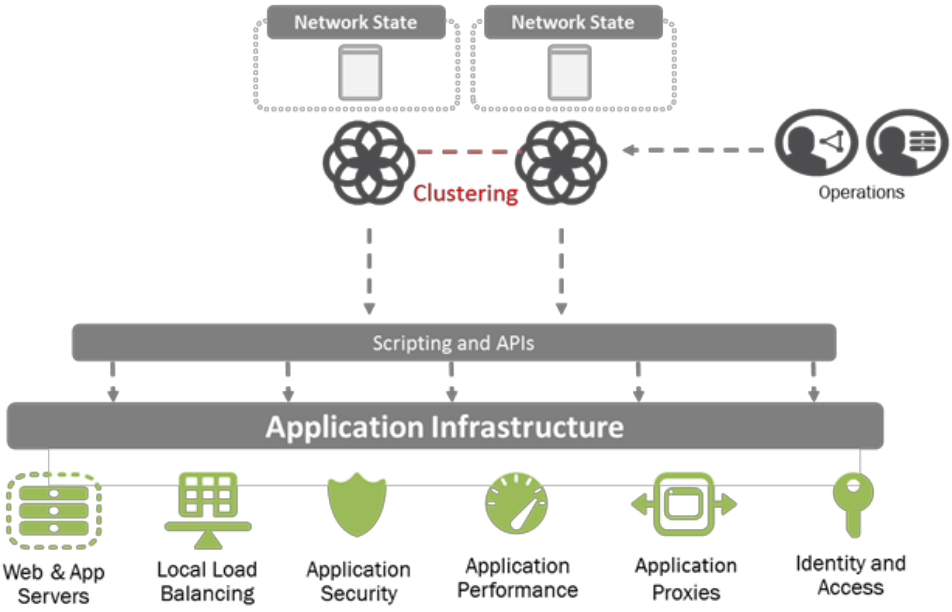
The Eventually Consistent Problem Comes to the Network

Eventual consistency is nothing new. It has always been an issue when scaling applications, particularly those that rely on shared data. Consider Amazon, if you will. If you and I are both shopping for the same thing, and I order before you, it may take seconds or more before the database is updated. If you were in the middle of ordering at the same time, you and I may be contending for the same item. Because my order takes a moment or two to propagate through the system, your view of the database (the availability of the item) is inconsistent with mine.

It is assumed that eventually our views will be consistent, and that this age old unsolved problem of distributed computing simply must be accepted as unsolvable for now, Thus systems are designed with this principle in mind. Which means we end up back with Brewer's CAP Theorem staring us in the face and reminding us we can't be perfectly consistent in a distributed system, so we must deal with systems in such a way as to achieve eventually consistency.

At issue is the ability of a software controller to scale. The controller is, by design and necessity, part of the data path. That is both a blessing and a curse. It is from this fact that the real-time adaption of network behavior can be achieved, but it is also this fact which forces issues of scale and introduces the need for a distributed system from which the problem of eventual consistency derives. That's because more than one system will be the "master" repository for a given portion of network state. Even if one controller is designated as master of the network universe and thus maintains the "official" state of the network, there are those moments when the secondary (or tertiary) controller has modified the "official" state and introduces inconsistency. In the moments between when the two network states merge, there is the possibility that the first (master) controller will *also* try to make a decision based on information that relies on network state that is no longer valid. If Controller B, for example, removes a port from a VLAN, and before that state can propagate to the master, a packet arrives in the fabric, destined for that port, Controller A will have no way to know that it is no longer participating in the VLAN and will, as expected, tell the switch to route to that port.

The issue will be shortly resolved, assuming timely synchronization of network state across the cluster, but in the meantime performance (or availability) may be negatively impacted.



The problem with eventual consistency in the network is one of magnitude. Eventually consistent views of books in stock at Amazon has a very different impact than an eventually consistent view of the network underpinning today's applications and ultimately the business. We're not talking about losing out on a book, we're talking about potentially disrupting hundreds or thousands of applications that translates into hundreds of thousands or even millions of dollars. Ponemon's 2013 Cost of Data Center Outages proves this case out: "The average reported outage incident length was 86 minutes, resulting in average cost per incident of about $690,200."

Eventual consistency of the network may turn out to be quite costly.

Common Themes: Reliability and Control

This is not a new problem. This issue of stateful failover as applied to scalability of both infrastructure and applications is one that application delivery has been dealing with, well, for over a decade now. The issue when dealing with distributed state is always one of replication and synchronization between those devices providing for reliability. That doesn't change just because we move from one form factor to another, or from on-premise to cloud. The issue remains: how do we maintain an authoritative view of the state of an <application or network> while still enabling the scale necessary to meet demand?

While we (as in the industry "we") recognize that true stateful reliability - and thus perfect consistency - is currently unachievable due to the constraints of distributed system design, we also recognize that we can get pretty darn close. From an application perspective, the intelligence embedded in a service fabric is more than able to deal with the problem with minimal introduction of latency. That is, there will be a slight pause and some disruption when failure or disruption occurs in the network but if the service fabric is smart enough, the disruption is experienced by the end user as no more than a slight hiccup - likely unnoticeable.

But the further down the stack you go, toward core network function, the more disruptive such a hiccup is going to be.

That's one of the reasons a "centralized control, decentralized execution" architecture makes more sense from a network perspective. Such a model maintains authoritative control over the state of the network, but empowers individual components in the various fabrics (stateless L2-4 and stateful L4-7) that make up "the network" to maintain its own prescriptive configuration and take action when necessary based on the abstracted policies of the network as a whole.

Everyone likes to posit an answer to what will be the "killer app" for SDN. But before we can worry about that, we might want to consider what may be the "showstopper" obstacles for SDN. Eventual consistency when scaling controllers is one of those issues.

Because without a reliable and consistent network world, there is no application world. Or at least not one that users will be excited to rely on.

F5 Networks, Inc. | 401 Elliot Avenue West, Seattle, WA 98119 | 888-882-4447 | f5.com

F5 Networks, Inc.
Corporate Headquarters
info@f5.com

F5 Networks
Asia-Pacific
apacinfo@f5.com

F5 Networks Ltd.
Europe/Middle-East/Africa
emeainfo@f5.com

F5 Networks
Japan K.K.
f5j-info@f5.com