Bandwidth, Bandwidth, Bandwidth!

Lori MacVittie, 2013-19-09

#SDN To really provision bandwidth efficiently you have to get inside the application.

One of the most commonly cited use cases for SDN (the classical, architectural definition) centers on ensuring quality of service for applications, usually by adjusting bandwidth constraints and prioritization, sometimes dynamically based on the operating conditions present on the network.

In such a scenario the application magically informs the SDN controller of its bandwidth and service-level requirements and the controller adjusts the network and distributes the appropriate flow tables to the network fabric to support the application.

This is a great vision, but it is not without challenges.

The most significant obstacle is actually **not** getting the application to talk to the SDN controller. Northbound APIs could be used for this purpose, or some other API-based mechanism that is used to instruct the controller on application specific requirements. Let's not rat hole on that and assume that this is easily enough accomplished.

At this point the SDN controller has some requirements dictated by an application. Given the way in which an SDN controller distributes forwarding information to the network fabric, one has to ask how the SDN controller will represent the requirements of the application and, more importantly, how it will distribute those requirements.

Assuming a classical SDN architecture and the use of OpenFlow or a protocol similar in capability, the flow table in the network fabric will only be able to distinguish packets on a per-IP / port combination. Let's assume that's an accurate representation of the overall topology; that is, every application has a distinct IP / port combination. That means the SDN controller can, in fact, push flow table rules that are able to provision the appropriate bandwidth for those application flows as well enforce prioritization (if that's needed, too).

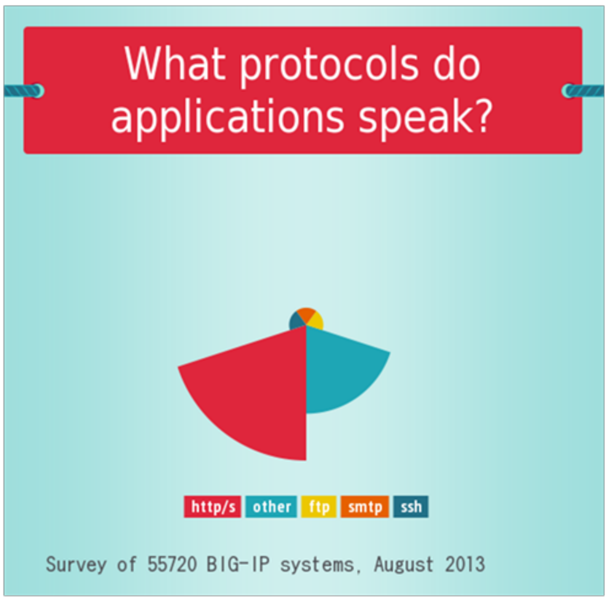
So far so good. You're thinking I'm barking up a pedantic tree or something, aren't here? Nope, here comes a significant problem starting with the question: How does the application define its need for bandwidth?

"Applications" today are comprised of a variety of functions and capabilities ranging from the delivery of simple text to dozens of images to embedded multi-media to video (and probably a few others I'm missing). The bandwidth needs of video is different from text is different from images is different for real-time messaging applications. Sensitivity to latency, throughput, bandwidth - these characteristics are peculiar to **content-types**, not the application itself (capabilities of the client-side network and device not withstanding, either). Given an application will varying - sometimes wildly - content types and requirements, should it simply request from the network the highest throughput and lowest latency required of all content being delivered? That's terribly inefficient.

HTTP is the new TCP

At the root of the problem is the reality that HTTP is the new TCP, with a significant percentage (62% in our research) of applications all using HTTP. A smaller percentage of those applications use port 8080 and port 443, but are still HTTP. In an increasingly API-enabled application world, the best chance we have to profile bandwidth needs for an "application" is at the URI level.

HTTP is the new TCP



The number of applications that use HTTP is 63% of in surveyed systems.

63%

All the interesting application-layer stuff is going on ***above*** layer 7 (HTTP) or more precisely ***within*** layer 7, in the payload (and across multiple packets and flows, but that's a different discussion). To really define the specific bandwidth needs of an application you have to look at the content being delivered. In many cases that content-type can be deduced from clues in the URI (file extensions like JPG, PNG, CSS, etc...) or extracted from the HTTP header **Content-Type**, which spells it out. In either case, you must be able to inspect and evaluate data in the HTTP payload, not merely IP and TCP parameters.

The biggest problem is that the current SDN architectural model, which focuses heavily on packet and flow-based processing, does not have the depth of visibility necessary to properly distinguish ***content type*** within an application and thus apply routing and forwarding policies based on each content type's unique requirements. An application delivering both video (a plurality of video is delivered via HTTP today, and it's increasing rapidly) and text will either need to be optimized for one or the other, but not both. The same is true for images, and even for different delivery models (push, pull, real-time, static) of text-based information.

To do that you need visibility ***into*** the application, down to the payload in some cases. That's just not a capability that the classical SDN architecture today is able to provide, for a variety of reasons. Current SDN architectures assume visibility and action on L2-4 only. Unfortunately the data necessary is at and above L7.

Ultimately the answer to this conundrum is to include L7 capable data path elements in the SDN architecture. The standard L2-3 SDN fabric can then optimally route packets through the network based on general, application-oriented network requirements while allowing the L7 aware data path elements the ability to do what they do best: inspect, analyze, evaluate and even modify (optimize) application messages in order to optimally deliver data to the end-user.

Application awareness, as it's often referred to, is not enough. To really ensure the network - and thus SDN - is able to offer application-specific services in the network requires application fluency. And application fluency isn't something you find by peeking at packets from layer 2-4. You've got to go deeper - to layer 7 and beyond.

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

F5 Networks, Inc.	F5 Networks	F5 Networks Ltd.	F5 Networks
Corporate Headquarters	Asia-Pacific	Europe/Middle-East/Africa	Japan K.K.
info@f5.com	apacinfo@f5.com	emeainfo@f5.com	f5j-info@f5.com