Full-stack SDN: The Next Big Challenge?

Gianni Antichi, Gábor Rétvári
Disclaimer

Call for Papers - SOSR 2020

New This Year: Experience and Challenge Submissions

In addition to traditional research papers, this year the SOSR Organizing Committee is excited to announce the solicitation of Experience and Challenge papers. Given SDN has been successfully deployed in a variety of production environments, “Experience” papers detailing, evaluating, or providing a retrospective of the practical impacts of SDN deployments will be of great use to the scientific community. And while SDN technologies are currently deployed in production environments, the promise of software-driven networking remains large. As such, “Challenge” papers outlining the next steps, challenges, or radical ideas for software-driven networking and systems will also be requested. Both long and short Challenge and Experience papers are invited.

- This is a "Challenge" paper
- Don’t expect answers, only some (hopefully) interesting questions
TLDR;

Ethernet bridges handle packets at L2, IP routers at L3, and middleboxes add L4 processing capability
TLDR;

Ethernet bridges handle packets at L2, IP routers at L3, and middleboxes add L4 processing capability.

Ethernet bridges handle packets at L2, IP routers at L3, and middleboxes add L4 processing capability.


We argue it is time to extend SDN up into the Application layer (L7).
TLDR;

Questions? :D
Cloud 1.0: Monolithic apps deployed into VMs

Full app instances deployed into VMs

Exchange traffic over L2 and L3 protocols


Virtual Switch ➔ Virtual Machine ➔ Virtual Machine ➔ Virtual Machine ➔ Virtual Machine ➔ Virtual Switch

Fabric
Cloud 2.0: Microservices

Fine-grained decomposition of business logic into loosely coupled microservices

Lightweight isolation in Linux containers

Expose/consume services over application-layer (L7) protocols
Takeaway 1

With the transition to the microservice architecture, the main network communication pattern becomes application-layer (L7) protocols
Looking glass on microservices

Microservice communication relies on critical L7 network functions that are hardcoded into applications.

Examples: Load-balancing, L7 ACLs, circuit breaking, L7 health-checking, encryption, policing, observability, authentication and authorization.

<table>
<thead>
<tr>
<th>Network stack</th>
<th>App logic</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>L7</td>
</tr>
<tr>
<td></td>
<td>L4</td>
</tr>
<tr>
<td></td>
<td>L3</td>
</tr>
<tr>
<td></td>
<td>L2</td>
</tr>
</tbody>
</table>

Cannot impose L7 network policies centrally.
Example 1: Filter HTTP REST API calls

Microservices typically expose/consume services over RESTful HTTP APIs

These look the same for a conventional L2-L4 SDN switch (TCP, port=80/443)

The network **SHOULD** be able to filter connections based on HTTP header fields

The control plane **SHOULD** be able to set L7-ACLs in switches
Example 2: Differentiate/route based on VXLAN ID

If a new service version is deployed alongside production code..

VXLAN tunnels look the same for an L2-L4 SDN switch (UDP port is 4789)

The network **SHOULD** be able to handle traffic at the granularity of VXLAN Network Identifier!

The control plane **SHOULD** be able to install VXLAN routing rules in the dataplane.
Example 3: Police RTP streams by user ID

RTP streams look the same for an L2-L4 SDN switch

The network **SHOULD** be able to rate-limit RTP streams based on user ID (SSRC)

The control plane **SHOULD** be able to set/query counters at the granularity of individual RTP streams
Takeaway 2

Application-layer network functions SHOULD be moved out from applications into the dataplane to allow the enforcement of L7 network policies centrally.
State-of-the-art: The service mesh

The **service mesh** is an L7-SDN to manage HTTP-based microservice communication. Achieved by injecting an HTTP service proxy to each microservice.
State-of-the-art: The sidecar proxy model

The proxy runs side-by-side with the app and intercepts all ingress/egress traffic.
State-of-the-art: The sidecar proxy model

Even a local packet exchange requires stitching 3 connections one after the other. This is 6 kernel-space--user-space context switches (remote calls are even worse).
State-of-the-art: The sidecar proxy model

Check the paper for some numbers on how this architecture might affect network function performance!
Takeaway 3

The state-of-the-art L7 SDN is restricted to HTTP and runs on top of the inefficient sidecar-proxy model.
The challenge: Full-stack SDN

Microservice Container → Application → Business Logic → Service Proxy (L4-L7 policies) → Microservice Container

Virtual Switch (L2-L3 policies)
A local packet exchange would require now only 1 simple connection
This is only 2 kernel-space--user-space context switches!!!!
Full-stack SDN: How?

Process traffic at any layer in the protocol stack (UDP, TCP, RTP, WebSocket, Ethernet, IP, etc..)

Key components:

○ Full-stack SDN switch

○ Full-stack SDN control plane

See a couple of initial ideas in the paper
Conclusions

**Takeaway 1**
With the transition to the microservice architecture, the main network communication pattern becomes application-layer (L7) protocols.

**Takeaway 2**
Application-layer network functions SHOULD be moved out from applications into the dataplane to allow the enforcement of L7 network policies centrally.

**Takeaway 3**
The state-of-the-art L7 SDN is restricted to HTTP and runs on top of the inefficient sidecar-proxy model.

**Challenge:** Full-stack SDN
Thanks!