Why OpenFlow/SDN Can Succeed Where GMPLS Failed

Saurav Das[(1)], Guru Parulkar[(1)], Nick McKeown[(1)]
[(1)] Stanford University, sd2@stanford.edu

Abstract *OpenFlow & Software Defined Networking (SDN) ideas offer drastically reduced complexity in the control plane, increased programmability and extensibility, and a gradual adoption path; all significant advantages over GMPLS for dynamic interaction between packet and circuit networks.*

Introduction

The operational benefits of unified control over packet and circuit networks have long been touted by industry and academia alike. In fact, GMPLS was designed as a superset of the MPLS control plane protocols to "avoid re-inventing the wheel"; by using existing protocols and merely extending them for the unique characteristics of the circuit-world. It was thought that the use of the same protocols could lead to an *intelligent, automated, unified control plane* (UCP) for a variety of technologies – packet, time-slots, wavelengths, and fibers.

But GMPLS has failed in terms of actual deployment in wide area networks. It has undergone a lengthy standardization process at the ITU, IETF and OIF; all transport equipment vendors have implemented it in their switches; there have been many interoperability demonstrations; and yet after a decade, there hasn't been even *one* significant commercial deployment of GMPLS as a *UCP*.

GMPLS may have found use in a limited way as a tool to help the NMS/EMS provision a circuit after being triggered manually by the management system. But we are not interested in such use of GMPLS. Such use has a) nothing to do with IP networks; and b) nothing to do even with the ASON view of using GMPLS as an interoperable *transport* control plane. Rather this use of GMPLS is merely a proprietary vendor implementation of a control plane (in ASON parlance - a vendor-island I-NNI).

We are *not interested* in creating a control plane just for Transport networks. On their own, they *do not need* intelligent automated control, as they do not provide services that require dynamic circuit switching. In any case all services are moving to IP; and it is only the IP network that supports services diverse enough to benefit from dynamic circuit switching.

And so, we are solely focused on creating a simple unified control plane *across both packets and circuits*, where the benefits of both switching technologies can be taken advantage of from a common service standpoint. We believe that only architectural change will enable true unified control. Accordingly, we propose a control architecture based on Software Defined Networking (SDN) ideas[1]. In this paper we offer our perspective on why GMPLS fails as a UCP; and compare it to SDN/OpenFlow to show why the latter has significant advantages in making packets and circuits work together.

GMPLS Architecture

GMPLS is ultimately just a collection of control protocols. The IETF also defined three architectural-model models for GMPLS usage– peer, overlay and augmented. They differed by the amount of state that was shared between the IP and transport network: all, none and some, respectively.

The peer model has never found favor. In principle, you *could* use a *single* instance of GMPLS protocols across packet and circuit domains, treating it as a unified-control-plane with full information sharing between routers and transport switches (peer model). But this ignores organizational boundaries where information sharing is prohibited; and it increases load on (fragile) routing protocols, which now have to distribute not just packet-switch/link information, but transport-switch/link information as well.

What did find favor (at least in other standards bodies) is the IETF *overlay* model, which was formalized by the ITU into its own architectural description for Automatically Switched Optical Networks (ASON). The overlay model separates the control of IP and transport networks, by treating the IP network as an

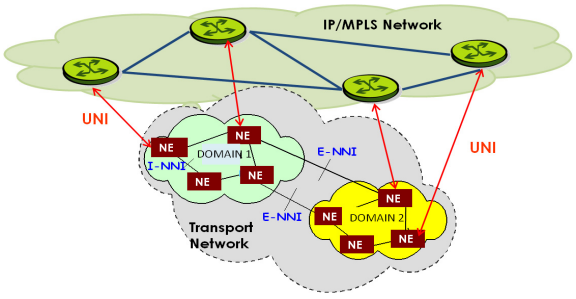

Fig. 1: MPLS/GMPLS/ASON

overlay network that runs a completely separate instance of the control plane – i.e. the IP/MPLS network runs the IP/MPLS control plane and the transport network runs the GMPLS control plane, and no information about the networks is shared across any boundary/interface between the networks in either direction. A *request for services* can be made across an interface (UNI) between the two networks (Fig.1). The transport switches remain in vendor-islands and continue to be controlled by vendor-proprietary solutions, but those solutions are made to inter-work by *layering GMPLS protocols on top* of those solutions, via interfaces knows an E-NNI. A request made across the UNI is then relayed (and satisfied) across vendor domains in the transport network via the E-NNI interfaces.

The UNI/E-NNI interfaces have not been used commercially and IP networks today do not request transport services dynamically.

OpenFlow and Software Defined Networking

SDN advocates the separation of data and control planes for packet and circuit networks. Packets and circuits can be commonly thought of as flows, and the data-plane switches are abstracted and presented to external software-controllers running a network operating-system (Fig. 2). All network control logic (such as for routing, TE, BoD etc.) is implemented as applications on top of the netOS. The distributed nature of the controllers is abstracted away by the network-OS, so that control applications are written with a centralized viewpoint[2].

The applications make control decisions that manipulate (via a network-API) an annotated map of the network presented to them and kept consistent by the netOS. In turn the netOS translates the map-manipulations into data-plane rules by *programming* the data-plane switch *flow*-tables via a switch-API (OpenFlow).

The common-map abstraction allows simpler

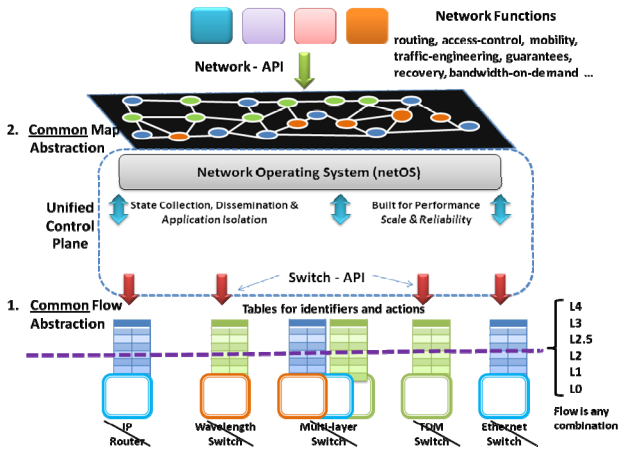

Fig. 2: SDN based Unified Control Architecture[1]

implementation of control-functions; makes it easier to insert new functionality into a network which is more-programmable; and with a global-view it affords joint-optimization of functions and services across packets and circuits[1].

GMPLS vs. OF/SDN

In this section, we discuss shortcomings of ASON/GMPLS, and offer our perspective on the advantages of the OF/SDN approach:

Control Plane Complexity: GMPLS use of the MPLS control plane results in protocols overly complex and fragile to make sense as a UCP.

Distributed link-state routing protocols like OSPF or IS-IS have convergence and stability issues if network state changes too fast or too often. This is the fundamental reason why IP networks today do not support dynamic links or dynamic link weights. To extend OSPF and use it in a dynamic circuit network with its effect being felt by the same or another instance of OSPF in the packet network is dangerous and unwarranted (and so, no one tries it).

Furthermore, the "avoid re-inventing the wheel" argument is simply not true anymore, given the amount of extensions that have gone into the protocols. Consider RSVP – it was originally intended for hosts to signal for resources from networks in the IntServ architecture; but then extended to serve as an LSP signaling mechanism in MPLS-TE; extended again to include signaling for transport network LSPs in GMPLS; and modified a third time to support the UNI interface. Every extension carried baggage from the previous extension increasing code-bloat and complexity.

Finally, further increasing the complexity, ASON/GMPLS insists on retaining transport network vendor-islands; layered architecture *within* the transport network; and new interfaces (UNI/E-NNI) to glue them all together. While retaining vendor islands and proprietary interfaces (and adding GMPLS on top) may help with backward compatibility to deployed hardware; in reality it makes the overall solution so complex that no one tries it in production.

In SDN, we eliminate protocols like OSPF and RSVP, interfaces like the UNI, and vendor-proprietary islands and interfaces. We replace them with a common switch-API (OpenFlow) and a distributed network-OS that make use of the distributed-systems techniques for sharing-state amongst multiple servers. In the process we drastically reduce control-plane complexity.

Lack of the common map-abstraction: ASON/GMPLS lack the map-abstraction. As a result – a) services available to the IP network are limited to the exact service-level definitions

defined (and pre-baked into the infrastructure) by the UNI interface; and b) the distributed implementation of network-functions across packets and circuits require lots of glue-code, and patchwork to existing protocols. Together these factors hamper simplicity, extensibility and kill programmability; ultimately limiting its use.

The common-map abstraction and network-API (supported by a slicing plane) gives full visibility across packets and circuits, and isolates the implementation of network-functions from the state-distribution mechanisms. We argue that this is the right abstraction: as full-visibility allows joint and global optimization of services and networking-functions across packets and circuits; and abstracting away the state-dissemination mechanisms result in networking-functions that can be implemented in a centralized way, making them simpler and more extensible.

We extended OpenFlow and a network-OS to build a prototype and UCP using our SDN based control architecture[3]. We demonstrated a fairly involved network-application on our prototype and showed that it takes two orders of magnitude lesser lines-of-code compared to existing solutions.

Lack of a gradual adoption path: Finally GMPLS provides no means for flexible and gradual adoption of a new control plane. Network operators are conservative. Transport network operators would like to respond faster and provide more dynamic services to meet their client needs. But at the same time, they loathe giving up precise manual control over the way traffic is routed over their network to a software control plane; irrespective of how intelligent that control plane may be. Additionally any new control technology has to compete with decades of established operational procedures. So the real key is to offer a way in which a network operator could gradually try out a new technology in a slice of the network to gain confidence in its abilities, while at the same time being able to flexibly choose the correct mix of technologies for a service.

In our control architecture we provide a gradual adoption path via incremental deployment using a slicing-plane. In slicing, a new control layer is placed between the underlying switches and the controller-layer. This new control layer (the slicing layer) partitions the underlying data-plane into multiple slices, where each slice can then be put under the control of different controllers. Circuits in a slice already provide data-plane isolation. The slicing plane crucially provides *control-plane isolation*. The transport network operator initially slices only a small part of its network and hands over control to an ISP's controller through its slicing plane. As long as the slicing plane guarantees isolation of the slices in both the data and control plane, the transport network operator *retains* control over the unsliced part of the transport network which can still be run manually, using established procedures, as it is today. Meanwhile the ISP is free to use whatever automated intelligent control algorithms it may desire in its isolated slice of the transport network. Over time as more confidence is gained in the slicing framework, more parts of the transport network could be sliced and offered as a service to other ISPs. It is worth noting that GMPLS provides no such means for flexible and gradual adoption.

Crucially, slicing also enables the ISP to run the common-map abstraction. The ISP's controller creates a common-map of the ISP's packet switches together with the circuit-switching-resources that are part of its own slice of the transport network. The (transport service provider's) slicing plane ensures that this view is restricted to *only* the slice that the ISP has purchased *and not* the entire transport network. Today, transport networks share no information and offer only static-bandwidth (dumb pipes). With slicing, transport networks can offer slices which include *bandwidth and control of virtualized transport-switches* to ISPs, thereby offering a new service.

Conclusions

We find that GMPLS is completely un-usable as a UCP. It is too complex; too buttoned-down, too inflexible to be of any use from a service standpoint across packets and circuits. IP networks will not touch it. Transport networks find little use for it. It's no surprise that it has never been used commercially across packets and circuits. This is unfortunate, as the benefits of dynamic-circuit-switching to packet networks are real and tangible and should not be shrouded by the deficiencies of the control-mechanism. We believe that only architectural change will enable true interoperation between packet and circuit networks. And so, we propose an SDN based architecture for a UCP as it is simple, extensible, and programmable and can be gradually adopted.

References

[1] S. Das, PhD thesis, Stanford University. http://www.openflow.org/wk/index.php/PAC.C
[2] S. Shenker et al., "The Future of Networking and the Past of Protocols", Open Networking Summit 2011.
[3] S. Das et al., Proc. OFC/NFOEC'11, NThD3.