Unifying Packet and Circuit Switched Networks with OpenFlow

Saurav Das, Guru Parulkar, Nick McKeown

Stanford University

OPENFLOW-TR-2009-4

Abstract:

There have been many attempts to unify the control and management of circuit and packet switched networks, but none have taken hold. In this paper we propose a simple way to unify both types of network using OpenFlow. The basic idea is that a simple flow abstraction fits well with both types of network, provides a common paradigm for control, and makes it easy to insert new functionality into the network. OpenFlow provides a common API to the underlying hardware, and allows all of the routing, control and management to be defined in software outside the datapath. We discuss our motivation, ideas and goals and report on a proof-of-concept OpenFlow enabled packet and circuit network recently demonstrated at SuperComputing 09.

December 7, 2009

Available at `http://OpenFlowSwitch.org/downloads/technicalreports/openflow-tr-2009-4-unification.pdf`

Unifying Packet and Circuit Switched Networks

Saurav Das, Guru Parulkar, Nick McKeown
Department of Electrical Engineering, Stanford University

*Abstract***— There have been many attempts to unify the control and management of circuit and packet switched networks, but none have taken hold. In this paper we propose a simple way to unify both types of network using OpenFlow. The basic idea is that a simple flow abstraction fits well with both types of network, provides a common paradigm for control, and makes it easy to insert new functionality into the network. OpenFlow provides a common API to the underlying hardware, and allows all of the routing, control and management to be defined in software outside the datapath.**

Keywords- Computer networks; Internetworking; Optical communication equipment; Wide area networks;

I. INTRODUCTION

Big networks are expensive to run and service providers are always looking for ways to reduce their capital and operational costs. One approach is to combine different specialized networks, reducing the number of technologies the operator needs expertise in, and reducing the number of boundaries between different types of networks. For example, many network operators have combined their separate voice and data networks to great effect. We call this “horizontal convergence” in which (typically) IP data networks replace specialized voice, video and control networks.

This paper is about “vertical convergence” in which networks running at two layers are converged. Over the years, there has been much talk about how transport networks (built from optical and electronic circuit switches) could be subsumed by the packet-switched services that run over them. There are several ways to do it. One way is to use only packet switching, and emulate circuits [1] where fixed rate services are needed. Another way is to connect packet-switch routers with direct point-to-point optical WDM links, and remove transport layer switching altogether [2].

We don’t believe optical circuit switching will (or should) be eliminated; on the contrary, we believe it offers significant advantages in the core of the network. First, optical switching is much more scalable; an optical circuit switch can switch much higher data rates, and consume much less power than an electronic packet switch. As a consequence, they are simpler, lower cost and more space efficient than an electronic packet switch. A useful rule of thumb is that an optical circuit switch consumes about 1/10th of the volume, 1/10th of the power and costs about 1/10th the price as an electronic packet switch with the same capacity. On the other hand, a circuit switch doesn’t have the statistical multiplexing benefits of a packet switch. This matters little at the core of the network where flows destined to the same next hop are naturally bundled, and their aggregate is relatively smooth. On the other hand, closer to the edge of the network packet switching offers big benefits due to statistical multiplexing and more fine-grain control.

We therefore seek a way to reap the benefits of both circuit switching and packet switching, by allowing a network operator to decide the correct mix of technologies. We reason that if both types of switch are controlled and used the same way, then it gives the operator maximum flexibility to design their own network. In particular, in this paper, we propose how circuit and packet switched networks can be controlled via the OpenFlow protocol.

IP and transport networks today are separate. In a typical service provider’s organization, two networks are operated and managed by separate groups. For example, operators such as AT&T and Verizon run separate IP and SONET/WDM networks leading to lots of duplication. Fault tolerance is a prime example: The underlying transport network often operates with 1:1 protection, while the IP network running on top operates at less than 30% link utilization in preparation for unexpected traffic surges and link failures.

IP and transport networks do not interact. IP routers are typically connected via wide-area pseudo-static circuits. IP networks (L3) cannot benefit from dynamic switching in L1/L0 networks, and instead regard the links as dumb pipes. If an operator could dynamically create and destroy light paths, their networks could be more cost efficient, and use less energy.

We are not the first to suggest a unified way to control packet and circuit switches. Most notably GMPLS [3] and the OIF [4] have experimented with alternative approaches. In Section VI we explain why we think these approaches are too complex and have not taken off. Sections II and III provide background on OpenFlow architecture [5], describing the separation of data and control planes. In Section IV we explore how the flow abstraction can unify packet and circuit networks. Section V describes prototype unified OpenFlow networks we are building in collaboration with circuit switch vendors.

II. OPENFLOW ARCHITECTURE

In today’s packet networks, a router/switch is both the control element which makes control decisions on traffic routing, as well as the forwarding element responsible for

This work was supported by the Stanford University Clean Slate Research Program

traffic forwarding, and both these functionalities are tightly linked (Fig. 1a). Housing control and data functions in the same box makes routers complex and fragile, quite unlike the streamlined routers envisaged by the Internet pioneers [6]. Today, a backbone router runs millions of lines of source code, and a plethora of features in software and hardware.

Transport networks are similar. While traditionally they have had a separation between a circuit switched data plane and a packet switched control plane, this control could reside within the box (Fig. 1b) or outside the box with proprietary interfaces (Fig. 1c). Additionally, out-of-box-control may not even be a distributed control plane, but more likely an Element Management System (EMS) / Network Management System (NMS) hierarchy. Those that desire the former are headed towards the same problems seen in packet switched networks today.

OpenFlow advocates a clean separation between the data plane and the control plane in packet *or* circuit networks (Fig. 2). Because the data plane is typically implemented in hardware, OpenFlow provides the control plane with a common hardware abstraction. A network (for example an autonomous system) is managed by a network-wide operating system (e.g. NOX [7]), running on multiple software controllers (Fig. 3), that controls the data plane using the OpenFlow protocol.

OpenFlow abstracts each data plane switch as a flow-table. The control plane makes decisions as to how each flow is forwarded (reactively as new flows start, or proactively in advance), then caches its decision in the data plane's flow tables. For example, the control plane might decide to route all of the http traffic destined to Europe along the same path, and so would add a flow-entry in the flow-table of each switch along the path. OpenFlow allows various types of actions on flows (e.g. forward, multicast, drop, tunnel), as outlined in the current specification [8]. The network-wide operating system decides how every flow is routed, which ones are admitted, where they are replicated, and (optionally) the data-rate they receive. And so now the control plane determines access control, routing, multicast, load-balancing and so on. Moving the decision making out of the data plane means the data plane is oblivious to how, say, routing decisions are made, and a new routing protocol can be added in software. We say that the network is now "software-defined".

The consequences are quite far-reaching. By providing a standardized open interface to the data plane, innovation can take place at a much faster pace than today. Network owners and operators (as well as vendors, researchers and 3rd party developers) can all add new functionality and services to the network. New functionality and services are added by creating network services on the network operating system or the controller using another standardized API (Fig 2). This helps the network to *evolve* more rapidly (e.g. to try out new access control methods, or to provide alternative mobility managers). And it potentially paves the way to greater *diversity* of solutions because of a much larger pool of developers.

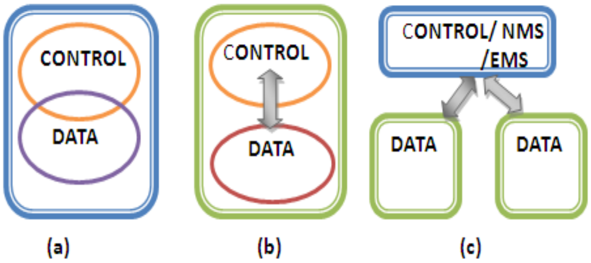


Fig. 1. Different architectures in today's packet and circuit networks

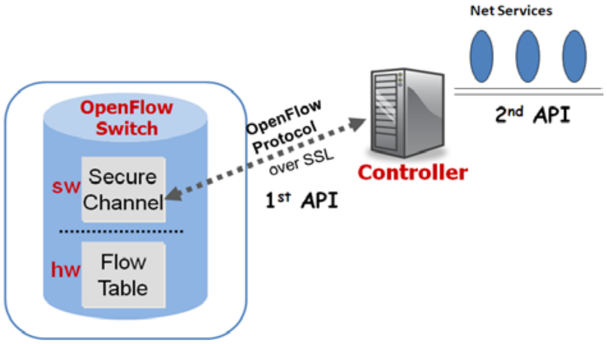


Fig. 2. OpenFlow Network Architecture.

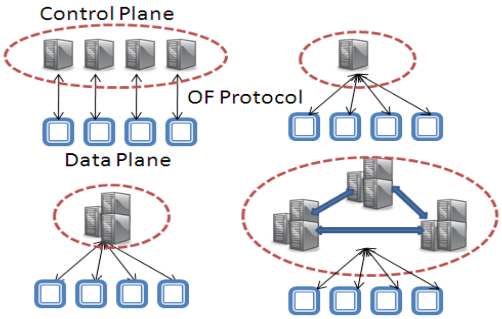


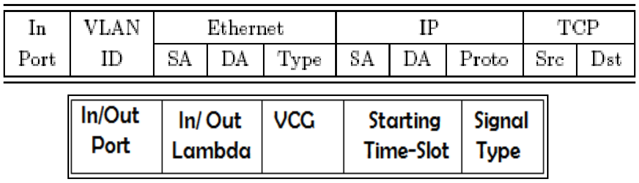
Fig. 3. Control Plane Architectures

An OpenFlow network is easy to virtualize. In [9] we describe how an OpenFlow network can be "sliced" into several independent and isolated networks, each with its own controller. The slices can be used to try new versions of features (*evolution*) or to try radically new architectures, including new routing protocols, address formats and so on (*diversity*).

A number of vendors have created prototype implementations of OpenFlow on Ethernet switches, IP routers, WiFi access points and a WiMAX basestation. Stanford is deploying OpenFlow enabled networks in its CS and EE buildings that support both production and experimental traffic, and we are working with seven other campuses to help deploy OpenFlow in their network too.

III. THE FLOW ABSTRACTION

OpenFlow abstracts the data plane as a flow-table. A flow can be defined as any combination of L2, L3 and L4 packet headers as well as L1/L0 circuit flows (Fig. 4).



In Port	VLAN ID	Ethernet			IP			TCP	
		SA	DA	Type	SA	DA	Proto	Src	Dst

In/Out Port	In/ Out Lambda	VCG	Starting Time-Slot	Signal Type

Fig. 4. OpenFlow Switch Flow Table Entry

Incoming packets are matched against the flow definitions; if there is a match, a set of actions are performed, and statistics kept. Packets that don't match any flow entry are (typically) encapsulated and sent to the controller. The controller can decide how to process the packet, and then (optionally) cache its decision in the data plane so future packets in the flow are processed the same way. Thus while each packet is switched individually, the *flow* is the basic unit of manipulation within the switch. A management application or a provider could also proactively create a set of flow table entries, in anticipation of some traffic flows, in order to avoid setup delays. OpenFlow is backward compatible with legacy networks; an OpenFlow packet switch could behave as an Ethernet switch, IP router or standalone L4 firewall by defining flows with their respective headers. OpenFlow can be deployed in existing networks, allowing service providers to gradually gain confidence in it. In fact on the same OpenFlow network, a network operator can support standard Ethernet and IP production traffic as well as experimental flows that are defined as combinations of packet headers at different layers.

In [10] we showed that cross-connect tables in transport NEs can also be regarded as OpenFlow flow tables. Flows can be defined as circuit flows using L1 time-slot switching based on SONET/SDH and Virtual Concatenation (VCGs) or L0 wavelength or fiber switching. Likewise, they could be defined across packet and circuit layers as well. Thus the OpenFlow architecture allows for 1) a flexible definition of what constitutes a flow, at what layer and switching granularity, and 2) the definition can be changed dynamically, over time or in different parts of the network; for example, to aggregate flows as they move from the edge to the core of the Internet.

IV. OpenFlow Unified Architecture

In this section we describe how the OpenFlow architecture can unify packet and circuit networks in different network planes.

A. Data Plane Unification

Once we consider the flow abstraction across various *underlying* switching technologies – both packet (L2/L3/L4) and circuit (L1/L0) – we effectively blur the distinction between packets and circuits and regard them both simply as flows of different granularity in a flow-switched network.

Fig. 5 shows two OpenFlow switches: an OpenFlow packet switch on the left, and on the right, a transport NE that supports both packet and circuit interfaces and switch fabrics. We do not show the controller here but assume that the switches speak the OpenFlow protocol with the controller. The switches also maintain flow tables in hardware – Rule, Action, Statistics (R, A, S) flow-table entries for the packet switching fabrics, and bidirectional cross-connect entries (IN-OUT) with associated actions for the circuit switching fabric. While this example shows a time-slot based TDM digital cross-connect, the concept applies equally well to wavelength based WDM optical cross-connects (ROADMs & OXCs).

The packet switch identifies two separate flows via a) a destination IP address (11.12.0.0) and b) http traffic (TCP port 80) destined for another IP address (11.13.0.0). The Action applied to packets belonging to the flows is to add separate VLAN tags to the two flows (ids 2 and 7) and forward the packets out of ports 1 and 2 respectively, both of which are connected to the packet interfaces of the NE. In the latter, the packets from the two flows match on flow entries defined on the VLAN tags. The Action applied here is to forward out of different *virtual* ports, VCGs 3 and 5, together with the designated encapsulation/adaptation specified in the flow action (not shown).

On the circuit side of the NE, VCG-3 has a collective bandwidth of 450 Mbps. This is represented by flow entries comprised of the virtual concatenation of 3 TDM signals (VC-4s) together with their physical ports and starting time-slots. On the other hand, VCG-5 cross-connects to a single 10 Gbps STS-192 signal out of port3 on the first time-slot. Thus, with the flow abstraction, the OpenFlow controller can flexibly define flows and assign different bandwidths and routes to those flows, simply by the addition of flow table entries in switches with different switching technologies and belonging to different switching layers, thereby leading to datapath unification. Furthermore, by not requiring the switches to host complex distributed control plane functionality, the switches benefit from increased robustness and decreased cost while being able to accommodate and benefit from different switching technologies, both packet and circuit, where appropriate**.**

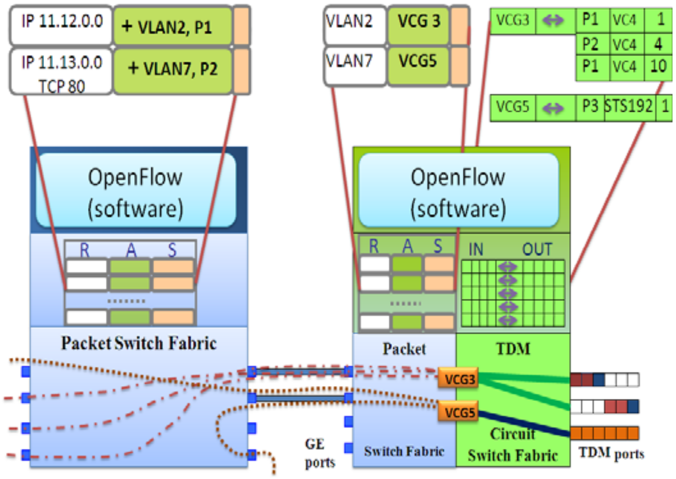


Fig. 5. OpenFlow Datapath Unification

B. *Control Plane Unification*

In Fig. 1, we had shown the significant disparity that exists in IP and Transport network control frameworks. Such disparity makes automated interworking between packet and circuit networks hard, if not impossible. The functions of routing, signaling and recovery across multiple layers and different architectures become significantly complex, so much so, that it is actually preferred to keep the networks separate and operated independently by different divisions, even within the same service provider organization. However, with OpenFlow, a single framework can be used to control any combination of OpenFlow enabled packet switches, circuit switches, as well as switches which have both packet and circuit interfaces and switch-fabrics (Fig. 6). Here's how –

1) By introducing the separation of data and control in the packet network together with the treatment of packets as flows, the single framework in Fig.6 becomes possible as opposed to the ones in Fig. 1, and 2) the OpenFlow protocol [8] has features for both circuit and packet switching hardware. It thereby allows the use of a single, standardized protocol for controlling the underlying heterogeneous hardware infrastructure, by the same controller. Another advantage of standardizing the interface is that the same controller can now be used to interface with many more switches, irrespective of the switching type and completely agnostic to the switch vendor. This has the direct effect of eliminating islands of vendor equipment (known as vendor domains in transport networks) that do not interoperate with other islands without manual control, and only speak to proprietary management systems.

It is also worth noting that such a unified control plane is greatly simplified compared to a fully distributed control plane such as in IP/MPLS and GMPLS networks. With a fully-distributed control plane in TE networks, link-state routing protocols disseminate link state information as well as resource availability information. This is required in such cases as each switch could make routing decisions and thus needs all the state information in the network. In multi-layer, multi-vendor-domain scenarios, distributed signaling becomes complex when going across packet and circuit networks, while the increased load on fragile link-state distributed routing protocols could result in increased network instability.

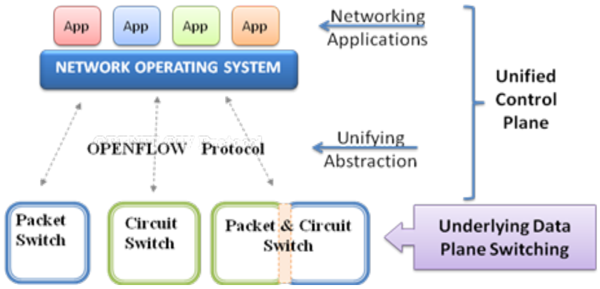


Fig. 6. OpenFlow Unified Architecture

But in OpenFlow, only the controller makes these decisions, the information for which is gleaned via the OpenFlow protocol directly from the switches. Thus by *eliminating* distributed signaling and distributed routing protocols within a controllers domain, the unified architecture benefits from a simple, robust, unified, automated control plane without layers and layers of complex fragile protocols.

Furthermore, when large packet and circuit network are planned, operated and managed independently, several management issues come up – careful planning and co-ordination has to take place between the groups that independently manage the networks, to ensure that something unexpected (like failures) or known activities (like maintenance) in one network does not effect the performance of the other. Expectedly, operating two networks and maintaining two groups that use completely different tools for managing the networks (SNMP for IP and TL-1 based proprietary NMS/OSS in Transport), result in considerable *opex* burden. We believe that this separation of management planes is a key hindrance to tighter integration of IP and Transport networks. However OpenFlow could help in this regard, as 1) the OpenFlow controller maintains a one-to-one relationship with each switch within its domain just like management systems in IP and Transport network do and 2) the OpenFlow protocol has features which allow the controller to perform management functions such as configure switches, query statistics, receive alarms, monitor performance etc.

C. *Virtualization Plane Unification*

A key component of the OpenFlow architecture is the *flow level* virtualization of the network and its resources. Virtualization has two key ingredients – programmability and isolation. The former is provided by the OpenFlow API itself, where clients can program the switches by flexibly defining flows according to their needs and inserting them into the flow tables. The latter is provided in the OpenFlow architecture by virtualizing the API itself with a thin layer of software which we call the FlowVisor.

The FlowVisor [9] is housed outside the switch leaving both the data plane as well as the controllers unmodified. The FlowVisor is transparent to the switches as well as the controllers and it enforces traffic isolation by monitoring and re-writing OpenFlow protocol messages. The switches think that they are talking to a single controller, while each controller thinks that it is controlling its own set of switches.

With the power of virtualization, OpenFlow can take into account *key* needs of the service provider – Transport network operators like to have precise manual control over the way traffic is routed over their network rather than give up that control to a software control plane irrespective of how intelligent that control plane may be. While they would like to respond faster and provide more dynamic services to meet their client needs, they feel that the resources they manage are too expensive, too valuable, and can cause far too much damage or loss of revenue if handled incorrectly.

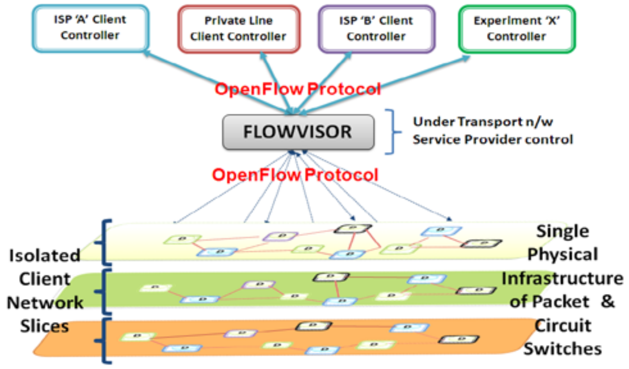

ISP 'A' Client Controller
Private Line Client Controller
ISP 'B' Client Controller
Experiment 'X' Controller
OpenFlow Protocol
FLOWVISOR
Under Transport n/w Service Provider control
OpenFlow Protocol
Isolated Client Network Slices
Single Physical Infrastructure of Packet & Circuit Switches

Fig. 7. OpenFlow Virtualization via FlowVisor

OpenFlow solves this problem by partitioning the network resources between multiple clients and isolating them, thereby ensuring that no client can intentionally or unintentionally disrupt service for any other client. The key is that in the unified architecture, OpenFlow enabled virtualization allows the transport service provider to *retain* control over the transport network, while allowing clients (such as an ISP) to use whatever automated intelligent control algorithms they may desire in their isolated slice of the network (Fig. 7).

We can further visualize that the client network (the ISP) can have its own virtualized network via a FlowVisor which is under the control of the ISP. The transport network resources provided to the ISP by the transport service provider can further be virtualized by the ISP for its own needs. Note that both FlowVisor's are capable of virtualizing either packet or circuit resources, or both. For example, under one construction, the transport service provider could virtualize just the circuit resources, while the ISP could virtualize both its own packet resources as well as the circuit ones it gets in its slice of the transport network– the OpenFlow architecture thereby allows for a hierarchy of FlowVisors.

V. Prototype Network and Demonstration

The key purpose of the OpenFlow architecture is to enable innovations and this paper shows how OpenFlow can enable innovations at the intersection of packet and circuit networks. We first describe an OpenFlow unified packet and circuit network testbed that we are building in our lab. We also present a couple of examples of new capabilities that are made much easier with OpenFlow.

Prototype Network: The packet switches in our lab testbed are based on the NetFPGA platform [11], a programmable hardware platform that allows us to build new packet switching capabilities in hardware. These packet switches are interconnected via transport NEs and tens of kilometers of optical fiber, thereby emulating a wide-area network. The NEs comprise of line-terminating WDM equipment (transceivers and optical mux/demux) as well as switching elements (cross-connects) with different switching technologies - optical wavelength switches, and modular electronic switches with packet and TDM switching fabrics. All the switches (packet and circuit) have the OpenFlow feature built in and are under the control of a single controller running NOX [7]. Finally hosts are connected to the packet interfaces and traffic is transmitted end-to-end in the testbed under the control of network control and management applications running on the controller.

Example Application – Variable Bandwidth Packet Links: Today's IP networks have static link costs, where "costs" here refers to the weightage assigned to a link as part of a shortest-path computation. However IP traffic is quite dynamic with constantly changing demand that frequently results in congestion on links along certain routes in the network, while other routes remain underutilized. Congestion could be alleviated if routers could dynamically change link costs and trigger re-computation of Shortest Path Trees (SPT), with the net result that some flows take other routes to their destination and thereby relieve load on the congested link. However, re-computation of the SPT needs to happen in every single router within the routing domain, which is potentially disruptive to *all* the flows in the network. Furthermore, there exists the possibility of routing loops while the routers converge, and more importantly, route flaps, where the SPT re-calculation and subsequent re-routing causes congestion somewhere else in the network, which in-turn causes another SPT computation and traffic ends up oscillating between paths. Avoiding network oscillations is the fundamental reason why IGP link metrics have static costs today as oscillations are far more undesirable than poor traffic load efficiency.

However, in the OpenFlow unified architecture, long-lived network congestion can be alleviated by simply *increasing* the bandwidth along a packet link when needed, via dynamic circuit switching. In our testbed, we will create packet link congestion by dialing up the traffic between end-systems. When this traffic overwhelms the packet link bandwidth allocated by the underlying physical layer, the resultant congestion in the packet link may result in the packet switch output queues to overflow. When the queue length crosses a pre-determined threshold set beforehand by the controller, the switch asynchronously notifies the controller of the congestion. A simple congestion control software application running on top of the NOX controller (see Fig. 2) could then decide to temporarily turn 'on' spare interfaces on the packet switches, establish new circuit flows between them on the NEs, and re-direct some of the packet flows causing congestion (using L3 or L4 hashes) onto the spare interfaces, thereby relieving congestion. Later the circuit resources could be re-directed by the application elsewhere, allowing them to be shared amongst several packet switches. Developing such applications is made easy by NOX and the two APIs.

Importantly, in sharp contrast to today's IP networks, none of these changes are disruptive to existing packet flows elsewhere in the network. Since the switches don't run a distributed routing protocol, there is no need for convergence and no possibility for route flaps. The controller makes the decision of changing a link bandwidth, and it only affects the flows along the link and nowhere else. Variable bandwidth packet links could allow service providers to run their links at higher utilization and buttress bandwidth when needed, instead of over-provisioning the network (4X to 10X) in order to provide customer satisfaction in the face of uncertainty (traffic surges, link failures etc.).

Example Application –Dynamic Automated Optical Bypass: In some situations, the service provider could establish new links between packet switches, where one did not exist before, via the dynamic circuits in the underlying layer. Again this is highly undesirable (and not done) in today's IP networks because such links would have to show up in the IP topology, leading to the need for re-convergence. Dynamically setting up and tearing down new links in the IP topology could lead to the same problems as changing link costs. But as before, OpenFlow does not suffer from these drawbacks and could easily accomplish this task as a means of traffic engineering triggered by a TE application running on the controller, or be manually driven by the network operator. For example, if many of the flows through intermediate packet switches are transit flows, the TE application could recognize that, and create the new circuit dynamically between the end packet switches by bypassing the intermediate packet switches, thereby reducing their load as well as overall flow latency. We are prototyping these capabilities on our testbed and will demonstrate them over the next year.

VI. GMPLS FAILINGS

Generalized Multi-Protocol Label Switching (GMPLS) was designed as an extension to MPLS and was intended to offer an intelligent and automated unified control plane (UCP) for a variety of networking technologies – both packet and circuit. GMPLS has undergone a lengthy standardization process within the IETF (since 2000) and variations of the protocol suite have also gone through standardization at the ITU and the OIF. However, as of this writing, while GMPLS has existed in some form or another for the entire decade, it has yet to see a significant deployment in commercial networks.

GMPLS protocols could be used as a control plane for transport networks, but it seems overly complex and fragile to make sense as a UCP. First, it assumes an underlying, existing IP/MPLS network for control traffic (with a link-state routing). Second, and for us most importantly, GMPLS misses the opportunity to introduce a path for continued evolution of the UCP. Understandably, in its first incarnation, there are many issues GMPLS does not address well (for example, the conservative way in which network operators manage and partition their networks). By defining how "everything" works up-front in a homogeneous protocol suite, GMPLS leaves little room for innovation to take place in the field, by the network operators. Whether or not OpenFlow is the "right" answer, we do believe that it is important for a UCP to be easily virtualized, so that the network control layer can be sliced to allow continued evolution as experience is gained in the field.

VII. CONCLUSION

In this paper we describe how OpenFlow could be used to unify (and continually improve) the control of packet and circuit networks. We are prototyping OpenFlow on circuit and packet platforms in collaboration with vendors and plan to demonstrate new dynamic circuit switching capabilities that the packet networks can exploit for congestion avoidance and agile traffic engineering. But ours is a small effort, and will only scratch the surface with a handful of examples to show how a UCP can bring together both types of network. We believe there are many other improvements to be made by other researchers.

REFERENCES

[1] "Recommendation I.150, B-ISDN Asynchronous Transfer Mode Functional Characteristics", ITU.

[2] http://www.cisco.com/en/US/solutions/ns341/ns525/ns537/ipodwdm_announcement.html

[3] E. Mannie, "Generalized Multi-Protocol Label Switching Architecture", RFC3945

[4] OIF-UNI-2.0, http://www.oiforum.com/public/impagreements.html

[5] N. McKeown, et. al., "OpenFlow: Enabling Innovation in Campus Networks", SIGCOMM CCR, Vol. 38, Issue 2, March 2008

[6] J.H. Saltzer, D.P. Reed, D.D. Clark, "End-to-end arguments in system design", Conference on Distributed Computing Systems, April 1981.

[7] N. Gude, et. al., " NOX: Towards an Operating System for Networks", SIGCOMM CCReview , Vol. 38, Issue 3, July 2008

[8] OpenFlow Switch Specification v0.9, http://www.openflowswitch.org/

[9] R. Sherwood, et. al., "Carving Research Slices Out of Your Production Networks with OpenFlow", SIGCOMM 2009 Demo, September 2009

[10] S. Das, G. Parulkar, N. McKeown, "Simple Unified Control for Packet and Circuit Networks", IEEE Photonics Society Summer Topical on Future Global Networks, July 2009.

[11] NetFPGA website, http://www.netfpga.org

APPENDIX A

As a proof of concept, we built a simple OpenFlow enabled packet and circuit network using carrier-class Ciena CoreDirector CI (CD) switches, and an application that sets up, modifies and tears-down L1/L2 flows on–demand and dynamically responds to network congestion. We demonstrated the network and application at SC09. Other network applications that can exploit the common API and OpenFlow enabled packet and circuit switches include integrated routing and traffic engineering, integrated network recovery, QoS, virtualization and more. In this section we report on our proof of concept network, application, and the SC09 demonstration.

The demonstration network was housed in three exhibit booths as shown in Fig. 8a. Each booth hosted a single CD supporting both layer 2 (GE) interfaces with a packet switching fabric, as well as layer 1 (SONET/SDH) interfaces with a TDM switching fabric. The CDs natively support the OpenFlow protocol for their packet and circuit switching fabrics, and are thereby controlled by an external controller, running NOX [7] as the network OS, over an out-of-band Ethernet network. Video clients and servers were connected to the GE client interfaces of the CDs, which were themselves connected via OC-48 SONET/SDH links. The three CDs together form a small demo network, emulating a real scenario in the Internet today – end user clients requesting services (web, e-mail, video etc.) from remote servers, with the requests going out as IP/Ethernet packets, encountering packet switches such as Ethernet switches and IP routers, before getting bundled into circuits for transport over the long-haul.

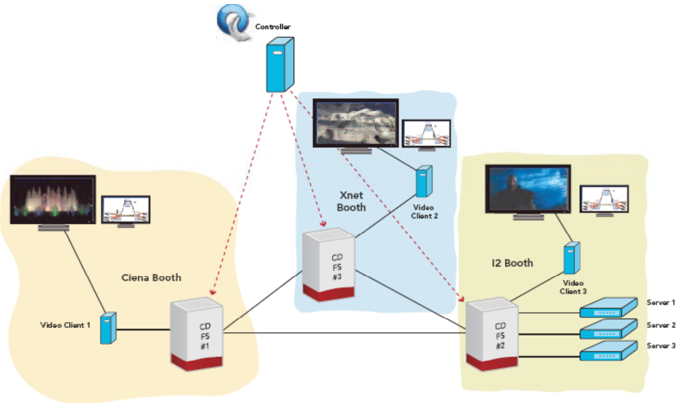
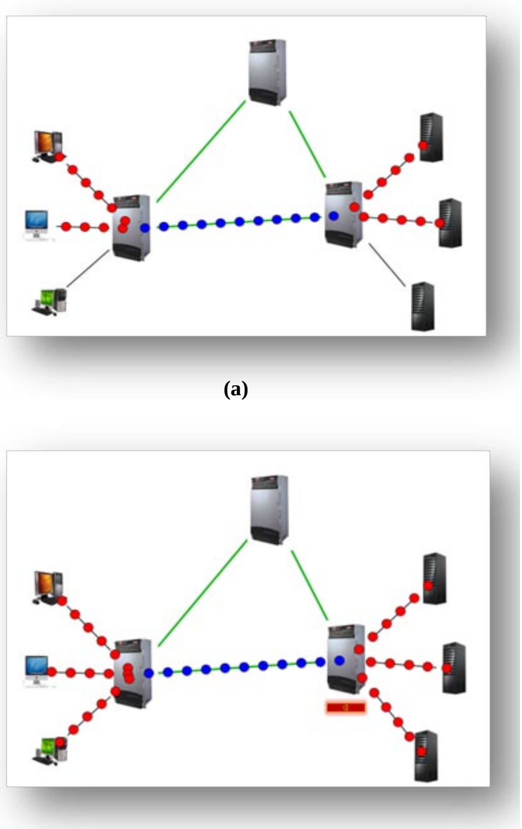


Fig. 8(a) Demonstration Network at SC09

Fig. 8(b) Demo setup at Ciena booth

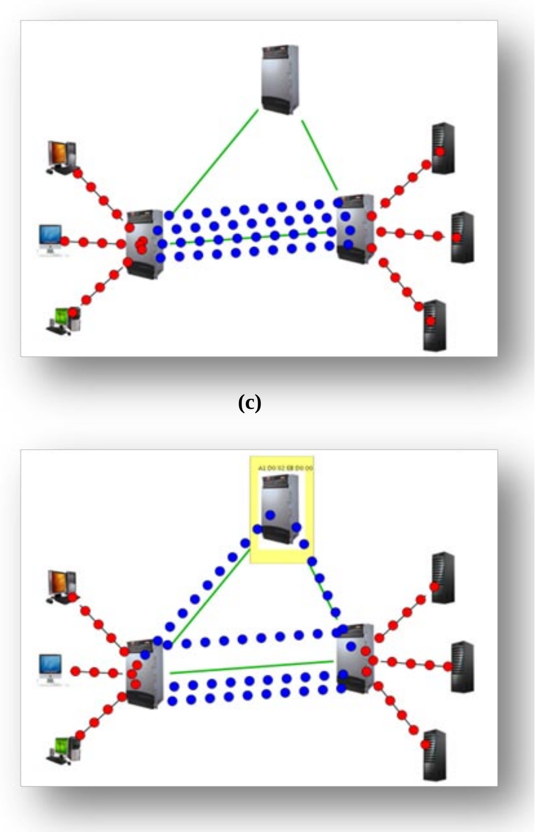
At the start of the demonstration, the CDs establish connectivity with the OpenFlow controller. The controller identifies the switches and their features through the OpenFlow interface and builds a switch/topology database. It then pre-provisions a SONET/SDH Virtual Concatenation Group (VCG) in the CDs which have GE interfaces. The VCGs serve as virtual ports interfacing Ethernet packet flows and SONET/SDH circuit flows. After this initial startup phase, one of the video clients makes a request for a video from a remote streaming video server. The request is initially redirected to the OpenFlow controller, which responds by directing CDs #1 and #2 to create an internal VLAN corresponding to the client port (in CD #1) and the video server port (in CD #2), and map the VLAN into the VCG virtual ports. The controller then provisions SONET signals and maps them into the same VCG, thereby enabling the packet flow to be transported over the circuit. All subsequent packets (in both directions) for this client-server pair match the existing flow definitions and get directly forwarded in hardware. As video data is received from the server, the packets are tagged with the internal VLAN id and mapped to the VCG. At the client side, the packets received from the VCG are switched to the client port based on the VLAN tag, which is then stripped off before the packets are forwarded to the client PCs, where the video is displayed on the screen. A GUI (shown in Fig. 9) was created that shows network state in real-time. Packet flows are shown in red and circuit flows in blue.

Initially, the cumulative data-rate of two video streams is less that the bandwidth of the STS-1(50Mbps) circuit flow they are multiplexed into (Fig. 9a), and the videos play smoothly on the client PC displays. However, when a third video stream is multiplexed into the same circuit, the bandwidth is exceeded, packets start getting dropped, congestion develops in the network and the video displays stall (Fig. 9b). However the congestion-control app running in the controller monitors network performance by acquiring switch port and flow statistics. It becomes aware of the packet drops, makes sure that the congestion is due to long-lived flows, and then responds by increasing the circuit bandwidth from 50 to 200 Mbps. It achieves this by adding more TDM signals to the VCG, thereby relieving congestion and restoring the video streams which start displaying smoothly again (Fig. 9c).



(a)

(b)



(c)

(d)

Fig. 9. OpenFlow GUI displaying real-time network state

Thus far, everything has happened with automated control without requiring any manual intervention. However at this point, if the network operator desires (for reasons such as load balancing, path diversity etc.), he can override the decision made by the app, by redirecting the individual circuit flows that make up the VCG along different paths in the network. This can be done by simply using a mouse to drag a circuit flow in the GUI from one path to another. Behind the scenes, the GUI interfaces with the controller and informs the app of the flow-drag. The latter assigns free time-slots along the new path and sends the appropriate commands to the switches, in essence performing a make-before-break operation (Fig. 4d) which is hitless to the video streams being transported, due to the use of SONET LCAS (Link Capacity Adjustment Scheme) technology. Finally, to end the demo, the app detects the end of the video streams and gracefully tears down the packet and circuit flows.

APPENDIX B

We have also demonstrated OpenFlow unified control in our lab testbed (Figs 10a & 10b) with the help of OpenFlow enabled packet switches based on the NetFPGA platform, and an OpenFlow enabled optical switch based on a Wavelength Selective Switch (WSS) granted by Fujitsu.

The NetFPGA is a programmable hardware platform that allows researchers to implement electronic packet switching functionality in hardware. It is a PCI card that consists of a Xilinx FPGA and 4 GE ports, and can be easily installed in any PCI slot in a host computer (Fig. 10c). The FPGA can be programmed to behave as a simple Ethernet switch or an IP router that switches packets between the 4 GE ports. But more importantly, it allows building *new* packet processing functionality in hardware, so networking researchers can try out their ideas and not be limited to functionality provided in commercial packet switching equipment. In our testbed, an OpenFlow packet switch is implemented in the NetFPGA, which switches flows defined on a subset of the 10-tuple shown in Fig. 4. The flow definition is inserted in the switch's hardware flow table, by the controller via the OpenFlow protocol [8].

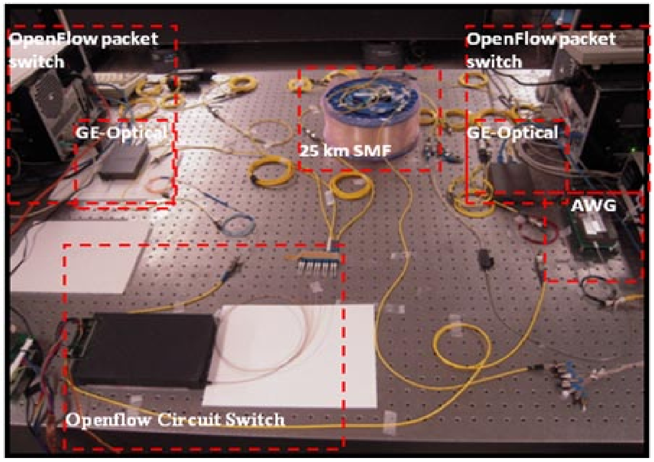


Fig**. 10(a) Testbed and components**

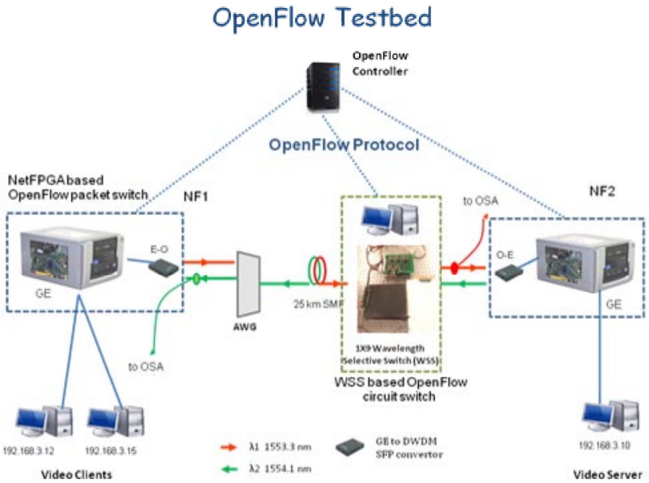


Fig. 10(b) Schematic of the testbed

The WSS is an all-optical lambda switch in a 1X9 configuration. It has the ability to independently switch any of 40 incoming wavelengths at the single input port, to any of the 9 output ports (Fig 10d). The incoming wavelengths (100 GHz spaced ITU DWDM lambdas) are demultiplexed and directed to their respective MEMS mirrors, which can be rotated to direct their wavelength to any of the 9 output ports, where they are multiplexed back into the outgoing fiber. In our setup we used the WSS bi-directionally, such that one of the 9 output ports actually served as an input port for a certain wavelength (λ2). The mirrors are controlled with a voltage driver which is sent commands over RS232 from a PC. In turn the PC interacts with the OpenFlow controller which directs the provisioning of cross-connections via the circuit switching features of the OpenFlow protocol [8]. Together the PC, driver and the WSS can be regarded as an OpenFlow enabled circuit switch.

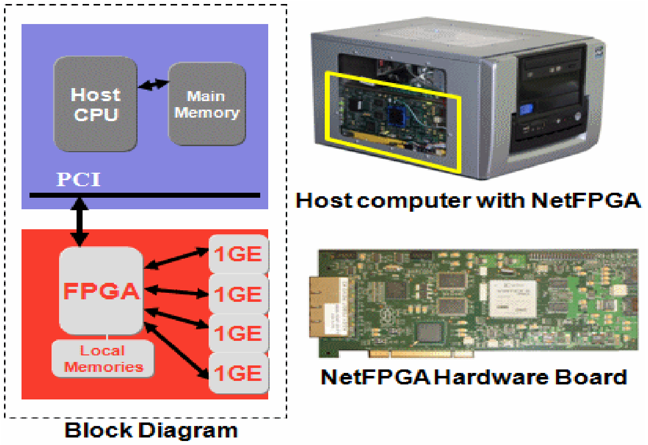


Fig. 10(c) NetFPGA platform

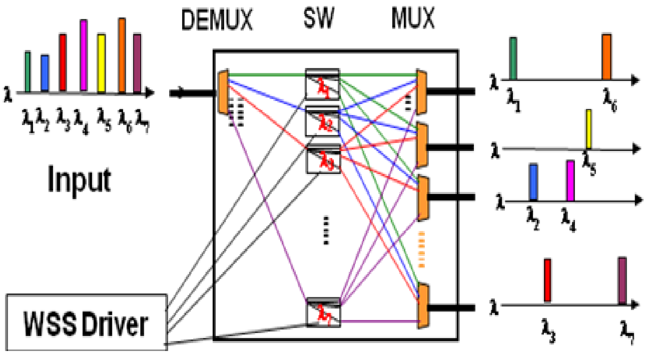


Fig. 10(d) Schematic of Wavelength Selective Switch

One of the four GE ports on each packet switch was connected to an electrical-to-optical converter from TrendNet (GE to SFP). We used DWDM SFP 2.5 Gbps transceiver modules in the converter which transmitted wavelengths of 1553.3 nm and 1554.1 nm respectively. These wavelengths traveling in opposite directions were multiplexed at the output/input of the first packet switch (NF1) by an AWG, transported through 25 kms of SM fiber, and then demultiplexed by the WSS before the input/output of the second packet switch (NF2). Together the OpenFlow packet and circuit switches form the underlying switching hardware that switch at different granularities (packet and lambda) and are controlled by an external unified control plane (compare Figs. 6 and 10b).

We connected client and video server PCs (end-hosts) to the packet switches via GE, and used Helix DNA as the video streaming server and Real Player as the client application. A video request is made by the client application to the client PC's TCP/IP stack (in the host OS), by providing the server's IP address, the L4 port it is listening on, as well as the video filename. The client OS initially does not have knowledge of the MAC address corresponding to the server's IP. It therefore generates an ARP request which is received by NF1 and is processed as the first packet of a new flow. Since the packet doesn't match any flow entries in NF1, it gets forwarded to the OpenFlow controller. The controller makes the decision to create the circuit by using the OpenFlow protocol to make the bidirectional wavelength cross-connection in the WSS, thereby bringing up the Ethernet link between NF1 and NF2. It then inserts flow table entries in the latter, to broadcast the ARP request to all interfaces other than the one which received the packet. This results in the ARP request reaching the server PC via the WSS and NF2. The server PC's OS sends the ARP reply, following which TCP handshaking takes place and a L4 connection is created, over which the video request is made. Finally, the server streams the video data packets over UDP, which are transported over the same bidirectional circuit created by the controller, and are received and displayed by the client.

We plan to continue development of the protocol, addition of OpenFlow to carrier class packet and circuit switches, and development of new network applications that can exploit the new capabilities.