Chapter 6

Conclusions

This thesis described a new approach towards common control, design and operation of converged packet and circuit networks. In WANs today, packet-switched IP networks and circuit-switched Transport networks are separate networks; typically planned, designed and operated by separate divisions, even within the same organization.

Such structure has many shortcomings. First and foremost, it serves to increase the Total Cost of Operations for a carrier – operating two networks separately results in functionality and resource duplication across layers; increased management overhead; and time/labor intensive manual coordination between different teams; all of which contribute towards higher Capex and Opex. Second, such structure means that IP networks today are completely based on packet switching. Which in turn results in a dependence on expensive, power-hungry and sometimes fragile backbone routers; together with massively over-provisioned links; neither of which appear to be scalable in the long run. The Internet core today simply cannot benefit from more scalable circuit *switches*, nor take advantage of dynamic circuit *switching*[†]. Finally, lack of interaction with the IP network, means that the transport network has no visibility into IP traffic patterns and application requirements. Without interaction with a higher layer, there is often *no need* to support dynamic services, and therefore little use for an automated control plane. As a result the Transport network provider today is essentially a seller of

† As we show in Appendix A, circuit-switches cost less than a 1/10th the price, and can consume less than 1/10h the power and volume of an equivalent packet-switch of the same capacity. Furthermore, there are certain functions *dynamic* circuits perform exceedingly well in the core – recovery, BoD, guarantees – that IP networks cannot take advantage of today.

dumb-pipes that remain largely static and under the provider's manual control, where bringing up a new circuit to support a service can take weeks or months.

Our proposal addresses these issues by offering a new unified-control-architecture for carrier networks. Our solution was designed around two control-abstractions based on an emerging idea called Software Defined Networking (SDN). The first abstraction we proposed is a common-flow abstraction that fits well with both types of networks and provides a common paradigm for control. It is based on a data-abstraction of switch flow-tables, manipulated by a common switch-API. The flow-tables take the form of lookup-tables in packet switches and cross-connect tables in circuit switches. Together with the switch-API, it abstracts away layer and vendor specific hardware and interfaces, while providing a flexible forwarding plane for manipulation by a common control plane. The second abstraction we proposed is a common-map abstraction, based on a data-abstraction of a centralized network-wide common-map, manipulated by a network-API. The common-map has full visibility into both packet and circuit switches networks, allowing creation of network applications that work across packets and circuits. Full visibility allows applications to jointly and globally optimize network-functions and services across multiple layers. And implementing network-functions as centralized-applications is simple and extensible, as the common-map abstraction hides the details of state-distribution from the applications.

We implemented these guiding principles in three network-prototypes we built to help us progressively verify our architectural constructs and ultimately validate our architectural claims. Two early prototypes focused on the common-flow abstraction for packet-switches working in concert with different kinds of circuit switches. The third, more complete prototype helped us understand the intricacies of building a converged network with a common-map and network-API. With this prototype we built a testbed that emulated WAN structure; and then implemented a new and fairly involved network capability which benefits from both packet and circuit switching.

With this new network capability implemented on our prototype, we validated our simplicity and extensibility claims, by comparing our implementation to our-best-guess on what it would take to implement the same functions using industry-standard solutions for packet-circuit control. We found our solution to be two-orders-of-magnitude simpler (in terms of line-of-code) compared to the industry solution, thereby validating our simplicity argument. More importantly, we presented qualitative architectural insights into why our solution is more extensible than current industry-solutions. We believe there are two main reasons why current packet-circuit control solutions are hard to use and extend: a) the use of the UNI interface which results in loss of visibility at the intersection of packet and circuit networks; and b) the implementation of services/network-functions as distributed systems which are tied to the state-distribution mechanisms. We argued that since our solution does not suffer from these limitations, it makes it easy to introduce new functionality into the network, or change existing functionality just as easily; thereby verifying our extensibility claim.

This latter fact bodes well for service providers who find it hard to differentiate their service-offerings from other carriers. Their networks today are built using closed-systems (routers and switches) from the same set of vendors with the same set of features. But with the use of our control-architecture, carriers can innovate in their networks across packets and circuits, and find ways to provide new revenue generating services without having to implement the features and services in the routers themselves. Put simply, our control architecture can lead to a faster pace of innovation and service differentiation.

At the same time carriers that own both kinds of networks, can benefit from reduced TCO by operating one converged network instead of two separate ones. Our control solution already helps converge the operation of the different switching technologies, while allowing the network-operator the flexibility to choose the right-mix of technologies for the services they provide. It also paves the path for developing common management tools, planning and designing network upgrades by a single team, and reducing functionality and resource duplication across layers.

As an example of the potential savings possible in a converged packet-circuit network with our unified control architecture, we performed a Capex and Opex analysis on a nationwide US backbone IP network. We considered industry standard practices for IP network design as reference design (IP-over-WDM). We then proposed a converged network we call IP-and-DCS, based of the following: a) replacing all backbone routers in PoPs with packet-optical switches; b) using a full-mesh of variable bandwidth circuits between PoPs; and c) adopting our unified control architecture for common control of packet and circuit switching. The latter feature is especially important in realizing the full-mesh topology, and for enabling close interaction between packets, dynamic circuits, and application/service needs. Interestingly we found that such construction can save us as much as 60% in Capex costs and nearly 40% in Opex costs when compared to the reference design. Importantly, such savings are insensitive to varying traffic matrices; and scale better (at a lower $/Tbps slope) as the network is upgraded to accommodate 5X or greater traffic-growth. We expect even greater Opex savings, as the use of a unified control plane eliminates time and labor intensive manual coordination and provisioning between IP and transport teams.

Even if carriers prefer a less integrated approach than the one described above, IP networks can still benefit from dynamic circuit-switching in transport networks, by leveraging a key aspect of our unified-control architecture: *slicing*. A network-slice according to our definition is a combination of bandwidth resources and control over switching resources. With slicing, a transport service provider can carve up slices of their networks and sell them as a new service to ISP. It gives transport-SPs incentive to share information with ISPs which is critical towards the creation of the common-map. ISPs can then create common-maps of their packet network and their slice of the transport network, to jointly optimize services across them and enable network-applications such as those we have described: dynamic-links, variable-bandwidth links, application-aware traffic-engineering, unified-recovery and others. Furthermore offering slices presents a new revenue opportunity for transport service providers, making them more than dumbpipe sellers – a crucial fact given that all services are moving to IP. Importantly slicing also helps the gradual adoption of our unified control architecture into today's networks. Without gradual adoption the successful implementation of any new-control technology would only be limited to green-field deployments, which are a rarity in core networks.

We have also applied the SDN based control architecture to MPLS networks. While MPLS has delivered on services (TE, VPNs), operators feel that it has failed to deliver on the promise of cheaper, inexpensive switching; due in large part to the burden of bearing a tightly coupled, distributed IP/MPLS control plane of ever-increasing complexity. We showed that by retaining the MPLS *data*-plane (flow-abstraction) and introducing a simpler and extensible control-plane (map-abstraction), we can *provide the services* that MPLS provides, *without the complexity* of the IP/MPLS control plane. The implications are many-fold – simpler switching hardware based solely on MPLS (and supporting just OpenFlow) can be realized; all existing MPLS services can be supported, made simpler, more-dynamic and optimized; or new services/functionality can be created on MPLS rapidly without requiring extensions to existing protocols. To demonstrate our approach, we showed nearly every major feature of MPLS-TE that we implemented in just a few thousand lines of code; compared to nearly two orders-of-magnitude greater code-lines required in traditional distributed approaches.

Our work is in the early stages. Many challenges remain, the most important of which is the development of the common-map-abstraction by a network-operating system that is designed to operate on multiple-servers in multiple geographic locations, to ease scalability and performance concerns. Work is underway in industry and academia to realize such a control plane and meet the related challenges.

We are convinced that (with further development) if the ideas in this thesis are adopted by service providers, the greatest impact would be that they can remain profitable as the Internet grows. As a result, they would have greater incentive to invest in their networks, which in-turn would benefit society immensely.

6.1 Related Work

Our unified control architecture for packet and circuit networks is based on Software Defined Networking (SDN) principles [17]. There is a rich history of related ideas both in academia and industry. The 4D project [138], RCP [139] and the SANE and ETHANE [18] works are part of a related line of research that advocate the separation of control and data planes in packet networks.

In the industry, Ipsilon's General Switch Management Protocol (GSMP [140]) also allowed an external controller to control one or more ATM switches. GSMP was later extended to control other kinds of label-switches such as MPLS and Frame-Relay switches [141]. GSMP was not used widely; mainly because ATM and Frame Relay networks fell out of favor, and dominant vendors chose to implement the IP/MPLS control plane for packet-networks.

The IETF ForCES framework [142] defines a set of standard mechanisms and guidelines for control and forwarding separation, which is similar in spirit to OpenFlow; but it does not define a specific protocol like OpenFlow does to control a forwarding plane abstraction. The ForCES framework does not advocate the SDN map-abstraction or the flow-abstraction. It only seeks to define a framework for standardizing the exchange of information between third-party router-control plane software and off-the-shelf forwarding hardware.

We are the first to propose the use of SDN ideas to develop a unified control architecture for packets *and* circuits. None of the works mentioned above are related to circuit networks. The only prior work for unified packet and circuit control is MPLS/GMPLS. And so we take a closer look at MPLS/GMPLS in the Section 6.1.1 and contrast it with our SDN based ideas for packet-circuit control. We also take a closer look at a related data-plane technology called MPLS-TP that is being introduced in transport networks. In Sec. 6.1.2 we discuss IP and MPLS-TP networks in keeping with the original premise of this thesis – finding a way to run *one* network instead of two.

6.1.1 MPLS/GMPLS and ASON

Generalized Multi-Protocol Label Switching (GMPLS) was designed as a superset of MPLS, and intended to offer an intelligent and automated *unified control plane* (UCP) for a variety of networking technologies – both packet and circuit (see Fig. 1 from [24]).

Conceptually, Generalized Multi-protocol Label Switching (GMPLS) in the *data-plane*, uses the core idea of treating packets as flows/LSPs, and extends it to the circuit-domain where circuits were recognized as LSPs as well[†] [58]. Thus in the data-plane both packets and circuits could be treated as LSPs (similar to our common-flow abstraction). But in the *control-plane*, GMPLS ended up adopting the MPLS control plane protocols (OSPF-TE, RSVP-TE) as well. Since MPLS already had a well developed control plane (derived from a well-developed IP control plane), GMPLS simply extended the same distributed routing and signaling protocols (OSPF-TE, RSVP-TE) to control circuit switches [20, 23-25]. The reasoning given for adopting the MPLS control-protocols was 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 automated, unified control plane for a variety of technologies – packet, time-slots, wavelengths, and fibers.

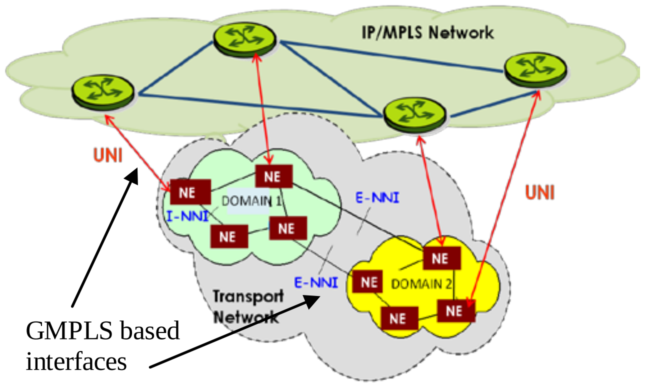
But at the end of the day, GMPLS was still just a collection of control protocols. There was a need to fit GMPLS to an architectural-model (better understood as a usage model) for the interaction between IP and Transport networks. The IETF defined three such models – peer, overlay and augmented. They differed by the amount of state that was shared between the IP and transport network: all, none and some, respectively [19].

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, as proposed by the IETF *peer* model [19]. But the peer model has never found favor for mostly two reasons: it ignores organizational boundaries where information sharing is prohibited; and it increases load on (fragile)

[†] with 'implicit' labels – like the color of a wavelength

routing protocols, which now have to distribute not just packet-switch and link information, but transport-switch and link information as well.

What did find favor (at least in other standards bodies) is the IETF *overlay* model [19], which was formalized by the ITU into its own architectural description for Automatically Switched Optical Networks (ASON) [21]. It took into account existing transport architecture and organizational boundaries. The overlay model separates the control of IP and transport networks, by treating the IP network as an overlay network, which shares no information with the transport network and crucially, 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. Such an interface between the IP and transport network is known as a User-Network-Interface (UNI) [†].



UNI: User-Network Interface between client and transport network

E-NNI: Exterior-Network Network Interface between vendor-islands

I-NNI: Interior-Network Network Interface within

Figure 6.1: ITU ASON model [21], similar to GMPLS overlay-model [19]

But the ITU went further. It not only required the UNI, but also defined other interfaces that maintained existing transport network structure – ie. vendor-islands and proprietary interfaces (Fig. 6.1). The transport switches in the vendor-islands would continue to be controlled by vendor-proprietary solutions, but those solutions would be made to inter-work by *layering GMPLS protocols on top* of those solutions, via interfaces knows an E-NNI. The IP network can make a request for services from the transport

[†] The IP network is the User and the transport network is the Network. The OIF UNI is an example of this interface [22]. Not surprisingly IETF has proposed its own UNI as well (RFC 4208).

network using what the UNI provides. Such a request is then relayed (and satisfied) across vendor domains in the transport network via the E-NNI interface.

We make a few observations here on MPLS and GMPLS. While MPLS has been very successful in wide-area IP networks, GMPLS has been a failure in terms of actual deployment. It has undergone a lengthy standardization process at the ITU, IETF and OIF; all the transport equipment vendors have implemented it in their network elements; there have been many interoperability demonstrations by vendors; and yet after a decade, there hasn't been even *one* significant commercial deployment of GMPLS as a *unified* control plane.

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 the IP network; 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 just for their switching nodes (in other words as a vendor-island I-NNI).

We are *not interested* in creating a control plane for Transport networks. On their own, transport networks *do not need* dynamic automated control, as they do not provide services that require dynamic circuit switching. We have discussed this in the Introduction, but it worth repeating – 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 architecture across *both* packets and circuits, where the benefits of both switching technologies can be taken advantage of from a common network-application standpoint.

It is in this context that GMPLS is an utter failure. We do believe that the initial choice to use the concept of a flow in the data plane as the common abstraction was the right one. However, the subsequent choices made or overlooked have contributed to its failure. We offer our perspective on why GMPLS fails as a UCP:

- Control Plane Complexity: Using the MPLS control plane as the starting point, results in protocols overly complex and fragile to make sense as a UCP. For example:
    - Dynamic Link-State Routing Protocols: 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. We discussed this in more detail in Sec. 3.4 (Challenge #3).
    - Protocol Re-Use: Furthermore, the reasoning given for starting with the MPLS suite of protocols to was to “avoid re-inventing the wheel”, by using existing protocols and merely extending them. However, with the amount of extension that has gone into the protocols, this is simply not true anymore. 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; and modified a third time to support the UNI interface. Every extension carried baggage from the previous extension increasing code-bloat and complexity.
    - Inflexible network architecture: To further increase the complexity of the control plane, GMPLS in the overlay model (Fig. 6.2) 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 our unified control architecture, we eliminate protocols like OSPF and RSVP, interfaces like the UNI, vendor-islands and vendor-proprietary interfaces. We replace them with a common switch-API (OpenFlow), a slicing-plane, a common-map and a network-API. In the process we drastically reduce the control-plane complexity, as evidenced by our analysis in Sec. 3.2.

- Lack of the common map-abstraction: In Chapter 3, we pointed out that the common-map abstraction and network-API (supported by the slicing plane) gives full visibility across packets and circuits, and isolates the implementation of network-functions from the state distribution mechanisms (Sec. 3.3.1). We argued 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 results in functions that can be written in a centralized way, making the control plane simple and extensible. GMPLS lacks the map-abstraction – it limits the services available to the IP network to the exact service-level definitions defined (and pre-baked into the infrastructure) by the UNI interface; and distributed implementation of applications across packets and circuits require lots of glue-code, patchwork to existing protocols, or new protocols. In short both features hamper simplicity, extensibility and kill programmability; ultimately limiting its use.
- 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 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 (see Sec. 3.4, Challenge #2).

Thus we find that GMPLS is completely un-usable when it comes to easing the inter-working of packet and circuit switching. It is too complex; too buttoned-down, too inflexible to be of any use from a common network-function or service standpoint across packets and circuits. IP networks will not touch it. Transport networks find little use for it. And so, it comes as 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 (as we have shown in Chs. 3 and 4); and should not be shrouded by the deficiencies of the control-mechanism. We hope that the adoption of our unified-control-architecture can help service providers benefit from the advantages of both switching technologies.

6.1.2 MPLS-TP and Packet-Optical Transport Evolution

The last few years have seen an emerging trend in transport networks to move away from older TDM circuit-switched technologies like SONET/SDH; and include in its stead, packet-transport technology like MPLS-Transport Profile (MPLS-TP). MPLS-TP adds OAM functionality, which is missing in regular MPLS. It does so by defining a new reserved label-id, and new payloads such as the generic-associated-channel (G-ACh) which carry OAM information in an LSP [126]. The need to add OAM comes from the fact that while transport SPs wish to move away from SONET, they want to retain the rich OAM capabilities provided by SONET. Unfortunately, they also want to run MPLS-TP networks like they run SONET networks – manually, without a control plane [127].

At the same time some service-providers such as Verizon, believe that packet-transport like MPLS-TP is needed in *addition* to newer circuit technologies like OTN and multi-degree ROADMs [89, 90]. And so many vendors have planned product releases for such Packet-Optical Transport Switches [85-88].

While it is still murky as to which direction future transport systems will go, this much is clear – the idea to build packetized-transport is clearly a reflection of the extremely high costs involved in supporting all services on an IP core-network [128]. Since all services are moving to packet-networks; by building a packet-network *in parallel to the Internet core*, the hope is that some packet-services can be siphoned away from the costly IP network and provided by a packetized-transport network. In this way reduced load in the IP network could result in requiring lesser IP router-ports and thereby reducing Capex.

We believe that the idea of supporting services by introducing cheaper MPLS-TP ports is a sound one. But doing so in a network parallel to the IP network, neglects the fact that *two separate networks* will still need to be operated. Even if both are predominantly packet networks, their operational modes will be quite different, given that the MPLS-TP network is intended to be run with centralized control and supported/ supplemented with OTN and ROADMs. Instead, we have proposed an integrated IP/MPLS(-TP)/OTN network in Chapter 4, which not only achieves the desired Capex benefits, but also the Opex benefits of operating a single network with a multi-layer control plane.

6.2 Future Work

6.2.1 Challenges in the Control Plane

In this section, we discuss the challenges our architectural choices face, and highlight the work being done by others to overcome them.

Scale: A common perception of centralized decision making is that it cannot scale when compared to distributed solutions. However, with the common-map abstraction in the SDN approach, we note that only the control programs are implemented in a centralized way with a global view of the network. The common-map abstraction *itself* is

† At the time of this writing, MPLS-TP seems to be the technology of choice, winning out over older 'Carrier Ethernet' offerings like T-MPLS and PBB-TE.

created by a Network Operating-System, which is meant to be distributed and built to scale.

A question that is often asked regarding scale is: How many flows per second can the Controller handle? This is essentially a question regarding Controller throughput which can be a raw indicator of scale. [143] offers a side-by-side comparison of raw throughput for several open-source controllers. We note that a single server (4 cores, 8 threads) can process several millions of flows/sec from a group of 32 (emulated) switches, irrespective of the type of controller used. However this result needs to be put in perspective:

a) This result is essentially an indicator of the speed of response of the Network-OS software stack, which includes the TCP/IP and OpenFlow message-handling stack and a basic-network application like an Ethernet learning-switch. In general, when it comes to throughput, the controller's compute and memory resources are not limiting performance factors [130]. Trivially it is possible that a controller realized by a cluster of servers can replicate all the compute/memory resources found in routers in a fully distributed solution today. Such a cluster could run the same distributed algorithms as routers do today. But more importantly, such a compute cluster can far outperform a collection of routers in terms of compute and memory resources[†]. However in most cases the actual flows/sec handled will vary widely and depend on i) the network-application; ii) the state-distribution mechanism; and iii) the consistency overhead one must pay for maintaining (reading/writing) network state. However, the authors in [130] argue that such consistency need only be maintained on a per network event timescale (hundreds or thousands/sec), instead of a per-flow (millions/sec) or per-packet (billions/sec) timescale. And so maintaining (eventual) consistency for a (relatively) small number of events per sec is what allows the Controller to scale.
b) Another fact worth noting about the results in [143] is that they are produced from a Controller running on a single server. In our prototypes we used NOX which is also designed to run on a single server. However more advanced (commercial) network-

† For example, router/switch CPUs are typically several generations behind server CPUs.

OSes [129] are designed to run on multiple servers to account for performance scale and resiliency, which are key requirements in production networks. In [129] the authors also argue for the use of well-known, general-purpose techniques* from the distributed-systems community, for the problem of state distribution and maintaining consistency and resiliency, in lieu of more specialized distributed routing protocols.

c) In most cases today we find that what limits throughput is not the Controller performance, but the switch CPU performance. The rate at which the switches generate events for the controller (like packet-ins), as well as the rate at which flow-messages from the Controller take effect in the data-plane, are both limiting factors. The primary cause is poor switch-ASIC to CPU bandwidth. We believe that as SDN popularity increases, systems will be built which will optimize for this mode of operation thereby bringing down the timescales involved.

d) The other issue in controller scaling is that of geographical scale. In wide-area networks, not only is it necessary to have the network-OS be supported over multiple servers at one geographic location, but it is likely that controllers will be located in multiple geographic locations, both within an AS and across multiple ASes. Naturally some Controller-to-Controller communication mechanism will be required. While still early days for SDN use in the wide-area, potential solutions have been discussed in [131]. We anticipate this to be a rich area of networking research in the near future.

Latency: Another significant objection to centralized decision making is the round-trip-time (RTT) it takes to go ‘outside-the-box’ to a controller for decisions. While it is true that there is a cost, the actual effect on network performance can vary depending on the way SDN is deployed.

If deployed in a way that every new flow must be routed ‘reactively’ by sending the first packet of the flow to the controller, then the RTT cost must be paid. The RTT is composed of the following times:

1. Time to generate an event (and its corresponding message) in a switch

* Examples are Zookeeper, Dynamo, Cassandra etc.

2. Time to propagate the event message to the controller
3. Time for the controller to act on the event and make a decision
4. Time for the decision to be propagated back to the switch
5. Time for the decision to actually take effect in the switch

Time intervals for #2, 3 and 4 collectively have been measured in the 100's of microseconds for local controllers that perform basic routing of flows [130]. This corresponds to controller throughputs of hundreds of thousands of flows per sec. In the wide-area the propagation times (#2 and #4) would increase, implying the need for placing controllers optimally. And as we have mentioned before, while today's switches exhibit longer times for #1 and #5, we expect that optimized switching systems will reduce this time component. Nevertheless an RTT cost must still be paid for the first packet of a new flow in this mode of operation.

But there is another mode of operation that can keep latencies low. For example, wide-area networks are often pre-planned (both MPLS and circuit) where primary paths for flows have backup paths and secondary backup paths that the switches are already aware of and can switch to for fast-failover. Such 'proactive' mechanisms are possible in SDN as well. The Controller can pre-compute and download primary and backup information in the switches beforehand, so that packets stay in the data plane when failures happen.

Another mechanism involves the use of default paths for all new flows. Such paths are already pre-installed (with lowest priority of match) in the switches chosen to be along the default path (possibly the shortest path). This way, data packets never leave the data-plane. Occasionally the controller samples the packets matching the default flow (by using something like sFlow [144]), and figures out what to do with them – again in this scheme new flows do not suffer the RTT cost [132].

Application Isolation: One of the defining characteristics of the common-flow abstraction is the ability to determine and resolve conflicts between decisions made by different network applications running on the same network OS. The basic idea is to help extensibility by not requiring new network applications to take into account other applications already in play.

Our implementation of the abstraction does not provide this isolation comprehensively. For example, prioritizing (how event-handlers in different applications are called for the same event) helps - one could write an application for a specific case of a more generic event, prioritize its event handler higher than the generic handler and then stop execution of the event after the special handler deals with it. But it does not provide isolation generically. Even with event prioritization, an application writer has to take into account all other applications that have subscribed to that event, which in-turn sometimes requires knowledge of how that application handles the event.

Some early and interesting work that takes a different approach to application-isolation can be found in [133].

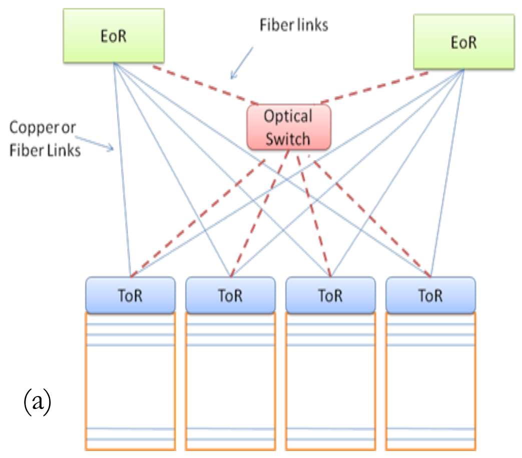
OpenFlow: The OpenFlow protocol serves as a switch-interface that instantiates of the common-flow abstraction. However the protocol has some limitations in terms of its structure (c-structs instead of TLVs or protobufs), and in the recent past it did not model real switch hardware well enough. The latest version of the protocol overcomes some of the limitations (multiple-tables in v1.1) and a new industry organization (the Open Networking Foundation) has been formed to further the development of the protocol [134]. The other factor involving hardware abstraction is that OpenFlow requires the ability to perform very flexible packet matching (combination of L2-L4), something that can be performed with TCAMs, but current hardware has small TCAM space. Instead they have large tables that can perform specific matches (just for L2 or just for L3). We expect that as more switches come to market optimized for OpenFlow this may change as well. Similarly our extensions to the protocol for circuit-switching are experimental and not comprehensive – for example, OTN switching is not supported and wavelength switching has only rudimentary support. But with the formation of the ONF, it is expected that the ability to control all kinds of circuit switches will eventually be included in the OpenFlow specification as well.

6.2.2 Applying pac.c to Other Network Domains

Recently there has been discussion of introducing optical switching in non-traditional networking domains such as datacenters [135, 136]. The advantages of optical circuit switching (guaranteed bandwidth, low-latency, creating new packet-links etc.) working in parallel with traditional packet switching has been recognized. Furthermore, in all proposals, OpenFlow and SDN ideas have been used or proposed in the control plane for control of packet and circuit switches [137].

We have similar views. While in this thesis, we have focused on traditional use of circuit switching in WANs, a lot of our ideas and conclusions can be transferred to other networking domains. The references mentioned above largely treat packet and circuit switching as *parallel* networks accessible to Top-of-Rack (ToR) switches. In addition, we believe that optical switching can be *integrated* into a packet-network as well in limited or extensive ways.

(a)



(b)

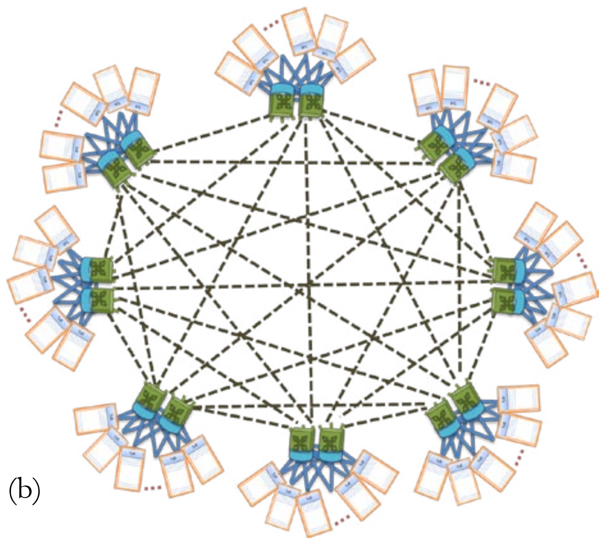


Fig. 6.2 Integrated Approaches for Packet and Circuit Switching in Datacenters

Under one construction (Fig. 6.2a), we can introduce small optical cross-connects *in-between* the levels of packet-switching hierarchy, with the specific purpose of creating *new links* between the switches they interconnect at different levels. These links are created when needed, to augment the bandwidth available between switches and routers. With this construction, we can temporarily change the over-subscription ratio by redirecting bandwidth when needed for as long as it is needed. For example, if the link between ToR1 and EoR1 is congested, we create a new link by cross-connecting their spare interfaces inside the optical fiber-switch. At a later time, if necessary, EoR1s spare interface can be connected to a different ToR switch.

In another construction (Fig. 6.2b), our IP-and-DCS design choices in Chapter 4 could translate to the datacenter as well. For example, a cluster of servers can be created with high bisection-bandwidth and low latency by using one-hop paths between all ToR switches. These ToR switches essentially take the place of the Access Routers in the IP-and-DCS WAN design. The one-hop paths between all ToR switches are supported by a full-mesh of dynamic-circuits between Packet-Optical switches. As before our unified control architecture is adopted to make the full mesh realizable; and to control the dynamic interaction between packets, circuits and services.

We expect that there will continue to be interest in applying packet and circuit switching together in datacenters and possibly even enterprise campus-networks. We believe that such constructions can avail of the benefits of both switching technologies only if used from a common application viewpoint that our control architecture provides.