

Chapter 5

Introducing SDN Control in MPLS Networks

MPLS networks have evolved over the last 10-15 years and have become critically important for ISPs. MPLS is primarily used in two ways: to perform traffic engineering in IP networks, providing greater determinism and more efficient usage of network resources; and for enabling MPLS based L2 or L3 enterprise Virtual Private Network (VPN) services, which continues to be one of more profitable services offered by ISPs. MPLS is the preferred solution for such services, mainly because plain-vanilla IP networks are incapable of providing the same services; and older ways of providing these services using ATM or Frame-Relay networks are no longer used.

As carriers deploy MPLS they find that (a) even though the MPLS data plane was meant to be simple, vendors support MPLS as an additional *feature* on complex, energy hogging, expensive core routers;[†] and (b) the IP/MPLS control plane has become exceedingly complex with a wide variety of protocols tightly intertwined with the associated data-plane mechanisms.

We make the observation that in any MPLS network, there are simple data-plane mechanisms of pushing-on, swapping, and popping-off MPLS labels in a label-switched-path. These mechanisms are controlled by a number of control-plane protocols that help

[†] eg. Cisco's CRS-1 and Juniper's T-640

provide the features and services (Fig. 5.1a). However, as we have shown in Chapter 3, any change to these services or the creation of a new service, in-most-cases, involves changes to these protocols or the creation of an entirely new protocol, which are lengthy, time-consuming processes. Yet, the data plane mechanisms remain the same simple push, swap, and pop operations.

And so in this work, we take a different approach to MPLS networks (Fig. 5.1b). We use the standard MPLS data-plane together with a simple and extensible control-plane based on OpenFlow and SDN. We find that the MPLS data-plane has similarities to the flow-abstraction. *But the MPLS control-plane does not make use of the map-abstraction.* Thus we retain the standard MPLS data-plane, and introduce the map-abstraction for the control plane.

There are significant advantages to doing so. The control-plane is greatly simplified and is de-coupled from a simple data-plane. We can still provide all the services that MPLS networks provide today; but more importantly, we can go beyond what MPLS provides today. We can globally optimize the services; make them more dynamic; or create new services by simply programming networking applications on top of the map abstraction. New capabilities are no longer tied to layers of protocols (which are eliminated). And the switch-API (OpenFlow) doesn't need to change either for all it gives is control over the simple push/swap/pop data-plane operations, which remain the same.

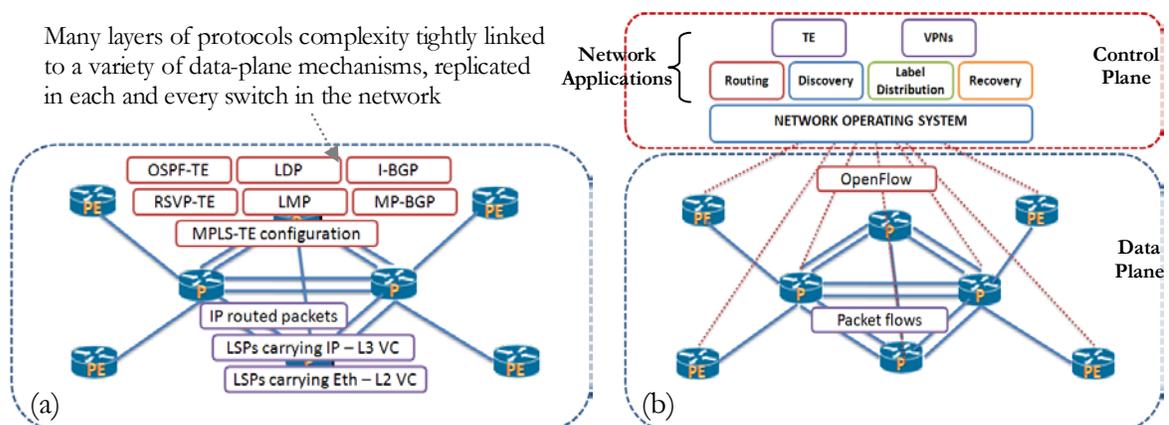


Fig. 5.1 (a) Today's IP/MPLS networks (b) MPLS with the Map-abstraction

In this chapter, we first discuss MPLS in relation to the flow and map abstractions. We show how flows in MPLS are close to the description of the flow-abstraction in the SDN context. And we also show how maps in MPLS networks are not quite the same as maps in SDN. But more importantly, we discuss how the map-abstraction is missing in MPLS. Accordingly all the benefits of the map-abstraction in terms of simplicity, extensibility and global-optimization are also missing in MPLS networks. We give examples of the benefits of introducing the map abstraction in MPLS networks.

To demonstrate that our approach can replicate services provided by MPLS today, we discuss a prototype implementation of MPLS based traffic engineering (MPLS-TE). There are two goals of this prototype: First, we show the simplicity of our approach compared to the existing MPLS control plane, by implementing nearly all the features that are part of MPLS-TE in just 4500 lines of code, compared with an estimated 100,000+ lines of code from a more traditional approach; and second, we show how the implementation of features can be made different from the traditional implementation in ways that either, greatly simplifies the feature, or provides a feature that MPLS-TE cannot provide.

Finally, we discuss how introducing the map-abstraction in MPLS networks fits well with our unified-control architecture for packet and circuit networks; a fact that makes our control architecture ideal for multi-layer networks.

5.1 MPLS Usage Models

There are two basic usage-models for MPLS, which correspond to the two services that MPLS provides – VPNs and TE. VPNs use what can best be described as a datagram model, similar to IP datagram forwarding. On the other hand, TE uses a flow-based model. Both models can and do exist simultaneously in today's networks. In this section, we briefly discuss these models. Note that the terminology used for naming the models is

not industry-standard. In fact the industry does not have terminology for the usage models. We introduce the terminology to best reflect the mechanisms involved.

5.1.1 MPLS Datagram Model

The datagram-like usage of MPLS is very similar to plain-vanilla IP routing. In this case:

- Label distribution happens in an *unsolicited* way [106], where every router sends out label-bindings for IGP-learned prefixes to each of its neighbors without being asked for it. For example, in Fig. 5.2, once the IGP (like OSPF or IS-IS) converges, every router learns of the router-addresses for all other routers in the IGP's domain. Each router creates label-bindings for these addresses, and then distributes these bindings to their neighbors (actually to their LDP neighbors). For example, in the figure, R6 receives label-bindings for R3's router-address from each of its neighbors R2, R4 and R5. Each binding implies that if R6 wants to reach R3 via a neighbor, it should use the associated binding advertized by the neighbor. R6 needs to use label 67 to go through R2; label 10024 through R4, and so on.

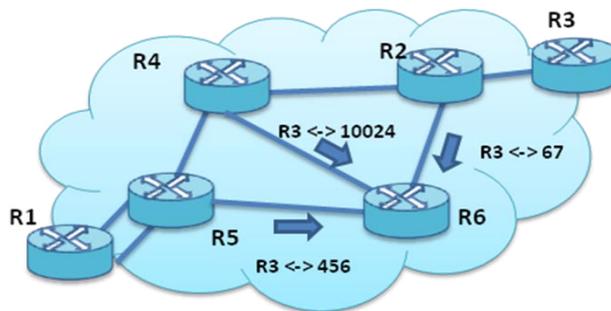


Fig. 5.2 Unsolicited Label Distribution

- The packets that match an FEC[†] at a router are then forwarded to the next-hop by inserting the label advertized by the next-hop. But the *selection of the next-hop* is still based on an SPF calculation from IP routing at each router. In other words paths for LSPs through the network are chosen by IP routing and thus are the same as IP shortest paths.

[†] FEC stands for Forwarding Equivalence Class – basically all packets that match an FEC (rule) are forwarded the same way (equivalently). The most common FEC is the IP-destination address.

And so, just like IP routing, each router makes an *independent decision* on what to do with incoming packets. In Fig. 5.3, R1 chooses to have two different flow definitions or FECs – one considers both IP src and dst addresses; the other defines the FEC to be all web traffic (TCP port 80) destined to an IP dst address range. Let's assume that both flows need to reach R3 to get forwarded to their ultimate destination. In this type of MPLS network, R1 *cannot define different paths (or LSPs)* for the two FECs from R1 to R3. The path that gets selected for the LSP is the IP-shortest path (say R1-R5-R4-R2-R3).

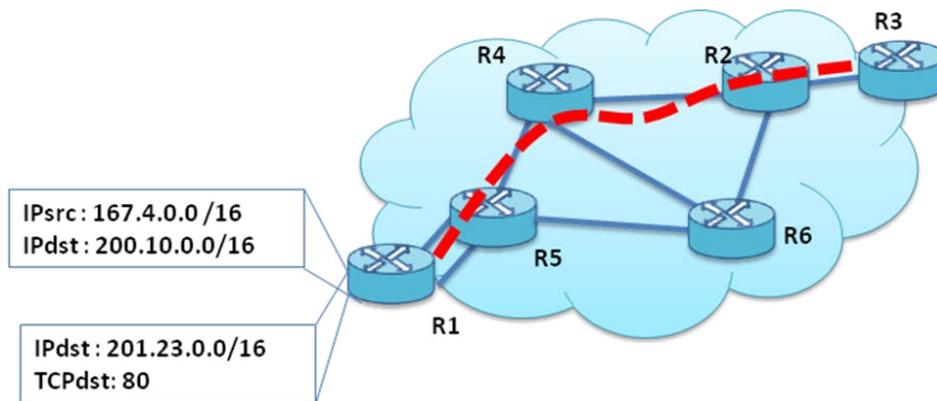


Fig. 5.3 MPLS Datagram Model

This is because the FEC definitions are not propagated to every router in the network. So when packets for the two different flows arrive at R5, the latter is unaware of how R1 defined the FECs. It simply considers *the only FEC that every router considers* for every packet – IP destination prefixes. R5 finds that both 200.10.0.0/16 and 201.23.0.0/16 need to go through R3; and that R4 is the next-hop (instead of R6) on the shortest IGP path to R3; finds the label-binding that R4 advertised for R3, swaps it on to the packet and sends it to R4. This procedure is repeated at R4 and R2. In other words it makes no sense to have different flow definitions in R1[†], because the flows differentiation is lost to the rest of the network, which treats them as the same. In other words, in this usage-model, an LSP is not actually *set-up*. The label-switched path develops ‘on-its-own’, and no router in the network has any control over the path the LSP takes through the network.

[†] Unless it is used for access-control into the network (accept, deny, rate-control). Again this is only at the edge of the network.

In essence, this usage model is simply IP-routing-with-labels, and is a holdover from the days when it was faster to do a lookup on a fixed 20-bit label than a variable length IP addresses. Today, it has found use in MPLS-VPNs, where the MPLS label-stack is used to deliver full mesh connectivity with address-space isolation. The outer label identifies the destination-router (PE) and the inner label identifies the VPN VRF instance. The path to the PE router is determined via IP routing, and forwarding is performed using the outer-label. We will discuss this MPLS use-case in the SDN context in Sec. 5.4.

5.1.2 MPLS Flow Model

In the flow-based usage-model, LSPs are actually set-up in the network by head-end routers (also called Label Edge Routers). Packets are classified into FECs. All packets in the same FEC are forwarded equivalently via an LSP whose path can be determined independent from regular IP routing (Fig. 5.4). Path calculation is done dynamically by the head-end router or offline by a management system. And it is typically signaled explicitly from the head-end to all other routers (LSRs) along the path using a signaling protocol like RSVP. Label distribution is therefore typically downstream-on-demand and ordered [106]. Importantly in this usage model LSRs do not make independent routing/forwarding decisions on packets. Once an LSP is set-up and the LER has mapped an FEC to an LSP, packets are forwarded as part of *flows* (FEC+LSP).

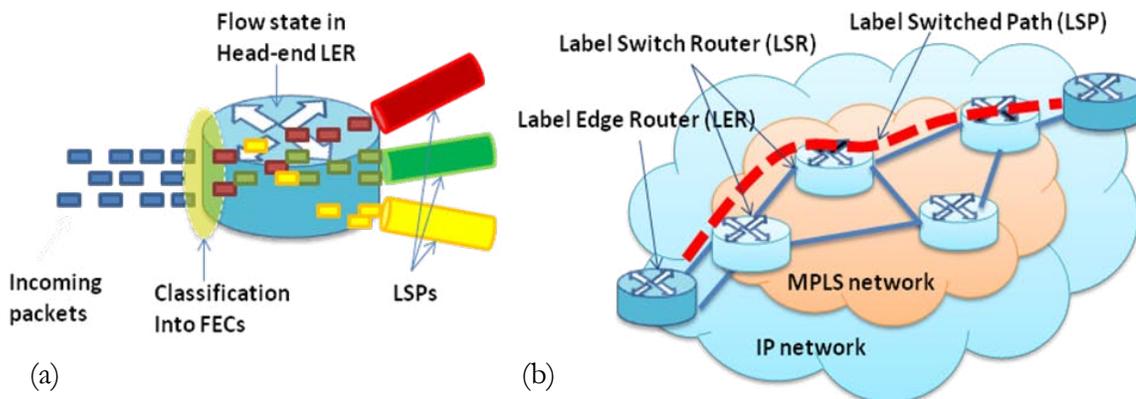


Fig. 5.4 MPLS Flow Model

The MPLS flow-model is used to provide MPLS based Traffic Engineering. Carriers widely use MPLS-TE today to achieve a) more deterministic behavior in IP networks; and b) greater efficiency in the utilization of network resources. In MPLS-TE the LSPs are more commonly known as TE-LSPs or tunnels. Traffic engineering is accomplished by routing the tunnels over under-utilized links in the network, and then routing IP traffic through those tunnels. MPLS-TE provides admission-control for TE-LSPs (tunnels) via bandwidth-reservation and constrained SPF routing. Additionally, there are a number of ‘features’ that MPLS-TE provides to ease the management, operation, and utility of tunnels. Such features include: Auto-Route, Auto-Bandwidth, tunnel-priorities, DS-TE, load-balancing, explicit-routes, re-optimization timers and so on. We will discuss this use-case in the SDN context in more detail in this chapter.

5.2 MPLS and Abstractions

The primary objective of this section is to show how MPLS compares to our control architecture. We show that while the MPLS definition of flows is not exactly the same as our definition, it is close enough to be applicable. But while MPLS retains and makes use of *a map*, it completely misses out on the *map-abstraction*. We discuss a few examples of what the map-abstraction brings to MPLS networks.

5.2.1 MPLS and the Flow-Abstraction

We mentioned before that MPLS has the concept of flows. In the following discussion we compare MPLS based-flows to the SDN based flow-abstraction from Sec 2.1.1. We conclude that MPLS based flows (FEC+LSP) are not as generic and flexible as the SDN flow-abstraction in terms of the ‘match’-definitions and forwarding-actions. Neither do they support vendor-neutral switch-APIs.[†] But nonetheless, flows exist in MPLS based WANs.

[†] For example, the API into the FEC table is called Policy Based Routing (PBR) in Cisco routers, and Filter Based Forwarding in Juniper routers. Naturally the CLI commands are different.

- Logical Association:
 - SDN: A packet-flow is a logical association between packets that are part of the same communication and are given the same treatment in the network (within a switch and from switch-to-switch)
 - MPLS: MPLS has the concept of FECs (Forwarding Equivalence Classes) whereby packets are classified into FECs and forwarded equivalently in the switch and in the network via Label Switched Paths (LSPs). FECs therefore embody the logical association between packets in the same flow.
- Data Abstraction:
 - SDN: The data-abstraction is the representation of packet-switches as flow-tables, ports and queues. The flow is defined in tables which have the ability to identify the flow generically from multiple parts of the packet header in *each* switch.
 - MPLS: The data-abstraction differs from switch to switch in MPLS. In a Label Edge Router (LER) where the FEC is established, packets can be generically classified from multiple parts of the packet header. Such classification can be performed by techniques such as Policy Based Routing (PBR) working on a data-abstraction of a routing/forwarding table (RIB/FIB).[†] Once packets have been classified, a label is pushed on to the packet. Thereafter in all other switches along the flow's path, the Label Switch Routers (LSRs) have a data-abstraction of a label-table (LIB/LFIB) i.e. they only match on the MPLS label to identify the flow.

This distinction between the data-abstractions in LERs and LSRs can be useful; Using simpler exact-match forwarding on a label takes less space in forwarding tables than matching on packets generically. But it is also more restrictive – only MPLS labels can be matched with this data-abstraction. If however, a generic table abstraction is used in *every* switch along the flow's path, a) nothing prevents us from dividing things up like MPLS does by inserting an MPLS label; but b) if we wish we could do the same with other labels like VLAN tags; or c) in other cases, continue identifying

[†] Actually, PBR does not alter the routing tables. ASICs are implemented such that PBR policies supersede the forwarding decision in the routing table. Nevertheless, for the sake of comparison, we assume that the data-abstraction of the routing table in LERs is altered by an API such as PBR.

flows generically from multiple parts of the packet header, while *changing* the definition of the flow as we go along a path [54]. Thus our definition of the switch's data-abstraction is more flexible, which when coupled with the map-abstraction, makes networks more programmable.

- Treatment within a switch:
 - SDN: Packets that match a flow are treated the same way within a switch, where the flow-table applies the same set of actions on the packets.
 - MPLS: All packets that are part of the same FEC are certainly treated the same way within a switch. But again the set of actions that can be applied are more restrictive – pushing, swapping and popping off an MPLS label.
- Treatment from switch to switch:
 - SDN: Each switch through which the flow traverses does not make an independent isolated decision on the treatment to the same flow.
 - MPLS: This holds true when LSPs are established by head-end LERs.
- Accounting & Resource Management:
 - SDN: The flow-definition serves as a common-handle with which accounting and resource management can be performed at the flow-level in the network.
 - MPLS: This holds true for MPLS when using LSPs for traffic engineering.
- Layer-independent Switch-API:
 - SDN: The data-abstractions in all switches are manipulated by a layer-independent switch-API (like OpenFlow).
 - MPLS: Manipulation of the FEC in LERs may be exposed through the CLI by features like Policy Based Routing (PBR). Or it may be restrictive and hidden by features like MPLS AutoRoute [92]. In LSRs, the API is completely hidden and only operated on by label-distribution protocols like LDP and RSVP.

It is also worth pointing out that we compared the SDN flow-abstraction to the flow-based usage of MPLS (Sec. 5.1.2) where: multiple FEC definitions are possible; head-end

routers make decisions on LSP paths; label-distribution is downstream on-demand and ordered; and resource management can be performed on the basis of LSPs [92]. MPLS based flows are more restrictive than the SDN flow-abstraction; but it is fair to say that flow-abstraction exists in some form in MPLS networks.

We *do not compare* the SDN flow-abstraction to the MPLS datagram-based usage-model (Sec. 5.1.1) for the following reasons: In the datagram-model, the only logical association (or FEC or flow-definition) that matters network-wide is the IP-destination address; forwarding decisions are still made independently router to router; and one cannot perform resource management at the flow level. However, even though the SDN flow-abstraction cannot (and should not) be compared to the MPLS datagram usage-model; the former can still provide the service the latter enables – VPNs. We discuss this point in more detail in Sec. 5.4.

5.2.2 MPLS and the Map-Abstraction

When comparing MPLS network control with our control-architecture, it is important to distinguish between the *map* and the *map-abstraction*. We discuss them individually.

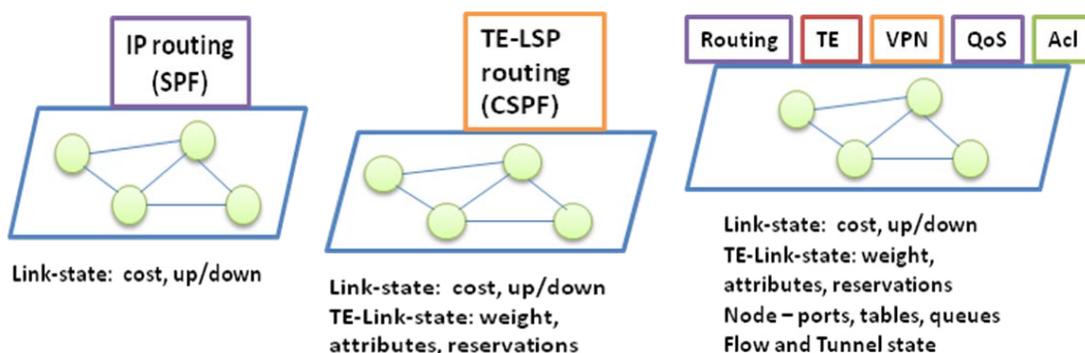


Fig. 5.5 Network Maps in a) IP networks b) MPLS-TE networks c) SDN

Maps: A map by our definition is an annotated graph of the network topology. First let's see how the maps themselves differ in different networks. In plain-vanilla IP networks, distributed routing protocols like OSPF and IS-IS distribute link state to each

router in the network. This allows each router in the network to build a map (Fig. 5.5a) of their own that is consistent with maps built by other routers. The map includes topology information like nodes and links, as well as the ‘cost’ of each link for use in an SPF calculation. The only other link-‘state’ that is distributed by the routing protocol is whether the link is up-or-down. This is a very limited map, which allows the basic network-functions of SPF routing and re-routing around failures, and not much more.

In MPLS-TE networks, the same distributed routing protocols are used, but this time with TE extensions that enable it to carry more link-state. Such state includes maximum reservable bandwidth on a link, un-reserved bandwidth per priority level, link attributes and administrative weights [92]. And so, the map that gets constructed in each router (Fig. 5.5b) has node and link information as before, but now with additional link-state. This allows a router to perform more sophisticated path calculations that take into account bandwidth and other constraints on links when calculating TE-LSP paths.

An SDN map has all the information about the topology and link-state discussed so far. But in addition, it has a lot more information about the network-node internals like switch flow-tables, queues, ports and their state and statistics. It also has global information on not just link-state but also switch-state, and if needed TE-tunnel and packet-flow state (Fig. 5.5c, Fig. 1.10 & Sec. 2.2.1). The net result is that the map can be used to support not just traffic-engineering based features, but a host of other network-applications. Such applications may include access-control, mobility management, VPNs, bandwidth-on-demand, QoS, datacenter backup/migration, and more. In short, the map in SDNs, together with the flexible flow abstraction, makes networks more programmable.

Map-Abstraction: So far we have compared and discussed the differences in the maps themselves. Now let’s consider the map *abstraction*. The map-abstraction helps centralize decision-making. It allows control-programs to be written in a centralized way with full, global visibility into the network, without worrying about how that global visibility (the map) is created, supported (perhaps with multiple physical controllers) and kept up to date. MPLS networks lack such a map-*abstraction*. We have discussed the

benefits of the map-abstraction before (Sec. 1.4.2). Here we focus on the benefits in the context of MPLS networks.

Simplicity: The map abstraction helps simplify the implementation of control programs for the simple reason that they are no longer implemented as distributed applications. When control programs are centralized and decoupled from the process of state collection and dissemination, they do not have to contend with subtle interactions with the state distribution mechanisms; which can be complex and is often the source of many errors.

Take the simple case of a network-function like IP based traffic-engineering on top of plain-vanilla IP routing. Each router is a decision-maker and shortest-path decisions are made on the basis of costs assigned to links. In most cases routing-protocols assign link costs taking the bandwidth of a link into account. But network operators have the ability to configure and change the cost of any link, and often do so, to avail of some rudimentary IP traffic-engineering benefits (without invoking MPLS-TE and its complex control plane).

While changing link costs is simple, the effect it can have is far from trivial. Changing the cost of a link in one part of the network, potentially changes a lot of shortest-paths that can in-turn affect traffic in a completely different part of the network. In principle it can be disruptive to many (or all) traffic flows. Worse, while the routing-protocol converges, loops may get formed and packets may be lost. Thus changing link-costs is considered just as disruptive as link-failures. But when decision making is centralized and decoupled from the state-distribution mechanism, such a network-function can be a lot simpler. When link-costs change, the controller can simply be programmed to not re-route existing flows in the network, and use new shortest-paths only for new flows. Or it could selectively re-route existing flows. The choice lies with the application-programmer (network-operator).

As a more concrete example, in Sec. 5.3, we will show that centralization of control-functions results in simpler code. We have implemented nearly all the features of MPLS

traffic engineering in fewer than 5000 lines of code [107], which is at least a couple of orders of magnitude smaller than distributed implementations of the same.

Reducing Protocol Load: The current IP/MPLS control plane uses a number of distributed-protocols that result in considerable computational load in router CPUs. Routers need to have all the intelligence necessary for aggregating and disseminating routing information, using distributed IP routing protocols like OSPF or IS-IS. In MPLS-TE networks, these routing protocols are extended to advertize extra information about links, and TE gives more reasons to generate control traffic as link-state changes more often[†]. Furthermore, another layer of complexity is added with the need for distributed signaling /label distribution mechanisms like RSVP-TE and LDP. And within a domain, an IP/MPLS network may additionally support a host of other protocols such as I-BGP, RIP, SNMP and MP-BGP, together with many more protocols for multicast and IPv6. All of these protocols contribute to control plane load, increased fragility and increased cost.

Some of these protocols are exceedingly complex due to multiple rounds of extensions over the years. RSVP is a good example. It was originally intended as an IntServ mechanism for hosts to request QoS from the network [112]. Intserv and RSVP were never used. But RSVP got extended to RSVP-TE to serve as a LSP signaling protocol. It is a poor choice for TE signaling; classic RSVP has many features (baggage) not intended for TE. For example, soft-state and frequent state-refresh messages were mechanisms for multicast-group support in classic-RSVP. But TE-LSPs don't involve hosts and are not one-to-many in practice. Further, RSVP runs directly over IP (instead of TCP); and so it needs its own reliable delivery mechanism. And it maintains multiple sessions per LSP. All of these 'features' result in lots of control plane messages and corresponding CPU load. Additionally, RSVP has been overburdened with many more features such as hierarchical LSP setup, point-to-multipoint LSP setup, LSP stitching, FRR setup, and GMPLS extensions, making an already complex protocol more bloated.

State-of-the-art routers come equipped with high-performance CPUs that can handle the computational load created by all these protocols. However carrier networks are not

[†] In our talks with Orange Telecom, we found that concern for “too many control messages” led them to turn off TE extensions on protocols like OSPF and IS-IS. They still use MPLS-TE but in a very static way.

all upgraded at the same time. Most often there is a mix of older and newer equipment. And from our talks with carriers[†], it is a real concern that the older ones cannot keep up. Consider this cycle –the carrier would have a need for new network functionality; they would approach router vendors to come-up with a solution; router-vendors would upgrade router-software (and sometimes change the protocols); upgraded software would in-turn increase CPU load; older routers/CPU cards would find it difficult to keep up; ultimately requiring an upgrade in router hardware.

SDN reduces switch CPU load by eliminating the need for many of these protocols. All current-distributed-protocol functionality can be replaced by a combination of network-applications, global-view and switch-API (OpenFlow). Furthermore, a Controller built on multiple-commodity servers can have much more CPU computational power than any one router. And in an SDN based network, when a carrier has a need for new network functionality, it need not result in upgrading switch-software or changing the switch-API. Our implementation of traffic-engineering functionality (in Sec. 5.3) is the perfect example of introducing new network functionality, simply by changing/creating network-applications on top of the map-abstraction.

Extensibility: The map-abstraction not only makes writing control-applications simpler, it also makes it easier to introduce new functionality into the network. Such new functionality could mean new applications or changes to existing applications. Importantly, it does *not* involve changes to the state-distribution mechanisms. The latter is solved once, and abstracted away from the control programs.

This is obviously not the case in today's IP/MPLS networks, where network functions are implemented as distributed applications, and each-function ends up using its own state-distribution mechanism (Fig. 5.6). Consider the path for introducing new functionality in the network today: First the carrier would have a need; then it would have to ask vendors to implement a solution; vendors would take considerable time to come up with pre-standard solutions that don't interoperate; then they would debate in the standards bodies to come up with an interoperable solution; the vendors then may or may

[†] Google and Global-Crossing.

not implement the entire standard; then there are carrier lab trials, limited field trials and finally, if all goes well, deployment. This is a 5-10 year process. At any time during this innovation cycle, the original needs of the carrier may change, and thus extensions are born and outdated baggage is carried cycle after cycle.

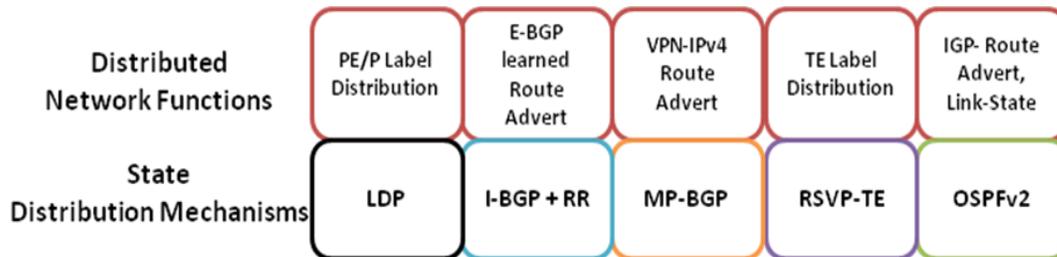


Fig. 5.6 Network Functions in Today's IP/MPLS Networks

SDN helps shorten the innovation cycle by making the network more extensible. It does so by un-chaining network functions/features/services from a) the creation of a new protocol; or b) the extension of an existing one; and c) requiring its implementation and deployment in each switch in the network

Global-View: One of the greatest benefits of the map-abstraction is the global-view it affords of all network-state. Such views allow the implementation of network-functions that globally-optimize network performance according to design or service needs. Without global-view, MPLS routers today can at best perform local-optimizations; which when coupled with the distribution-mechanisms, can lead to undesirable churn in the network. Consider an example from MPLS-TE.

Better utilization of network resources is one of the primary goals of traffic-engineering. MPLS performs traffic-engineering by forcing traffic through tunnels which are themselves routed over under-utilized parts of the network. The route that a tunnel takes depends on the bandwidth-reservation of the tunnel and the available un-reserved bandwidth on network links. As more tunnels get routed over a link (and therefore reserve bandwidth on it), the un-reserved bandwidth on the link diminishes, forcing newer tunnels to find routes over other links.

However the tunnel's reserved bandwidth is usually an *estimate* of the potential usage of the tunnel, made by the network-operator. The estimate is based on the source and destination routers and the average traffic between them. But traffic-matrices vary over time in un-predictable ways. And so a given tunnel's reservation could be very different from its actual usage at a given time, leading to an un-optimized network. Router vendors get around this problem by performing a local-optimization called Auto-Bandwidth.

With Auto-Bandwidth, a router at the head-end of a tunnel periodically checks the bandwidth usage of the tunnel, and alters that bandwidth-reservation to match the usage. But changes in reservation result in changes of link-state, as the links over which the tunnel is routed see a change in un-reserved bandwidth. Naturally such link-state has to be propagated to all routers in the network (depending on flooding-thresholds). But to further complicate matters, un-reserved bandwidth can be accounted for at *eight* priority levels; and tunnels can be assigned priorities whereby a higher-priority tunnel can preempt a lower-priority one by forcing the latter to re-route over a different path. This means that, as Auto-Bandwidth changes a tunnel's bandwidth reservation, the tunnel may force lower-priority tunnels to re-route, or the tunnel itself may get re-routed. In turn, re-routed tunnels further change link-state; causing greater flooding of link-state-update messages; and possibly even-more preemption of lower-priority tunnels. This cumulative effect is known as network-churn, as router CPUs have to deal with a lot of change in network state, especially in large carrier networks with tens of thousands of LSPs[†]. It is easy to see that churn is disruptive and undesirable from a network operator's view-point.

But churn is a direct result of local optimizations (like Auto-Bandwidth) performed by multiple decision makers (tunnel head-ends). Each router is only aware of the tunnels that it originates, and to some extent, the tunnels that pass-through it. For all other tunnels the router is only aware of the aggregate bandwidth reserved by these tunnels on links. In other words, even though the router builds a map giving global TE-link state, it only has a local-view of tunnel-state (or TE-LSP state). In SDN, the map-abstraction gives global-view of network-state, including all tunnel-routes and their current-reservations and

[†] Talks with Tier-1 ISPs suggest the use of 50k to 100k LSPs .

usages. And so an application can perform global-optimization of the tunnels while minimizing (even eliminating) network-churn.

Thus we find that IP/MPLS networks can benefit from the introduction of the map-abstraction in its control-plane. Existing control-functions and services can be simpler to implement and globally-optimize, while new functionality can be added more easily to the network. In the next section we show how traffic-engineering functionality can be added to a network supporting the standard MPLS data-plane and an SDN based control plane, *without using any* of the existing IP/MPLS control plane protocols.

5.3 MPLS-TE Features & Prototype

In this section we show how we can reproduce services existing in IP/MPLS networks today, and improve upon them in meaningful ways. We use the example of MPLS Traffic Engineering (MPLS-TE) and the wide-variety of TE ‘features’ that come with it.

For each feature that we have implemented in our prototype, we describe the way it is implemented in today’s networks; how that differs from the way we have implemented it in our SDN-based network; and finally how our approach enables some capabilities (within TE) that are not possible with the traditional approach. Importantly, our prototype serves as a validation of our architectural claims of simplicity and extensibility.

It also exemplifies the benefits of implementing networking-applications in a centralized way with a global-view, which comes from the introduction of the map-abstraction in MPLS networks.

5.3.1 MPLS-TE Features

We describe the following features: LSP Admission-Control; Auto-Route; Load-Balancing; DiffServ-Aware-TE; Auto-Bandwidth and LSP Priorities;

Admission-Control: The MPLS control-plane performs traffic-engineering primarily via an admission-control mechanism for LSPs. It is in principle similar to call-admission-control in telephony networks. The basic process is as follows:

- In the control plane, all IP links are assigned a maximum reservable bandwidth. TE-LSPs (tunnels) reserve part of the bandwidth on the links over which they are routed.
- The route (or path) that an LSP takes can be determined dynamically by the head-end router by running a Constrained-SPF algorithm. For example, given the source and destination routers for a tunnel (TE-LSP), together with its bandwidth-reservation (and priority, affinity etc.), CSPF can determine a path through the network that satisfies all the constraints – for example, a path over links that can meet the desired bandwidth reservation. If multiple such paths exist, the shortest one is selected.
- If a path that meets all the constraints cannot be found, the tunnel is rejected (admission-control). More importantly, tunnels tend to get routed (engineered) over links that are less used. If few pre-existing tunnels are routed over a link; the greater the chance that the link can satisfy bandwidth constraints for a new tunnel. And so network-resources are better utilized, and traffic in the tunnels may encounter less congested links (both TE goals).
- Once a path has been determined by a head-end router, it is signaled to all LSRs along the chosen route. Because there are multiple decision-makers (multiple head-ends) *contention* for resources may happen. For example, a head-end could setup a tunnel, but information on the resources used up by that tunnel may not propagate in-time to all other head-ends in the network (due to flooding-thresholds). In the mean time, another head-end tries to reserve a tunnel that contends for the same resources along links that overlap with the first tunnel. Because the path-decision for the second tunnel may have been made with stale information, every router along the way checks to see if the bandwidth-reservation is possible on the adjacent downstream link. If it is not, then the second LSP is rejected (again admission-control), and the (second) head-

end router is forced to re-route the tunnel, hopefully by this time with updated link-state (disseminated by the routing protocol). Ultimately it is a tradeoff – contention for resources can be reduced, if TE link information is always rapidly disseminated and head-ends calculate paths with up-to-date state; but it also ends up causing a lot more control-plane messages and CPU load.

- It is important to note that bandwidth reservations and LSP admission-control are purely control plane concepts. MPLS-TE *does not require* enforcement of tunnel bandwidths in the data plane [92], because the primary objective is to steer traffic, not to police or shape it in the data-plane. The latter techniques can be used in addition to MPLS-TE if desired.

SDN based Admission Control: Our SDN based implementation takes advantage of the fact that admission-control of LSPs are handled purely in the control plane. The TE application written above the map-abstraction: allocates reservable-bandwidth to each IP link; performs CSPF routing of tunnels; sets-up the tunnels in the data-plane by using OpenFlow to distribute labels to switches along the LSP path; and updates the map with LSP-path information and un-reserved bandwidth information for each link along the path. Importantly, since the Controller (more correctly the TE application) is the *only decision-maker*, contention for resources does not happen in this scenario. The Controller maintains the network-map and keeps it up-to-date with all tunnels and link reservations. So when a new tunnel is requested by the network-operator or by another application, the Controller is completely aware of all network-state, and can perform admission-control and LSP-setup without any contention. The data-plane switches are blissfully unaware of any bandwidth reservations for the LSPs. As a result, the data-plane switches themselves do not perform any admission-checks.

Auto-Route: Once tunnels are up in the data-plane, traffic needs to be redirected from IP links into the tunnels by LERs. Auto-Route is an effective way to do this without incurring the scaling problems of exposing tunnels to distributed-routing-protocols.

This latter point can best be explained with Fig. 5.7. Assume packet-flows from R1 to R3 are routed via $R5 \rightarrow R6 \rightarrow R2$. When the TE tunnel is created between R5 and R2, the flows need to be re-directed through the tunnel, as the tunnel is now a single-hop path between R5 and R2 (as opposed to a minimum of 2 hops for every other path[†]). Note that Fig. 5.7a shows the tunnel ‘conceptually’ as a single-hop between R5 and R2. But physically the tunnel is routed over the IP links in the network. For example, Fig. 5.9b shows one possible path the LSP takes in the IP network. Despite the number of hops in the physical tunnel path, from the perspective of routing packets in the IP network, the tunnel is still a single hop.

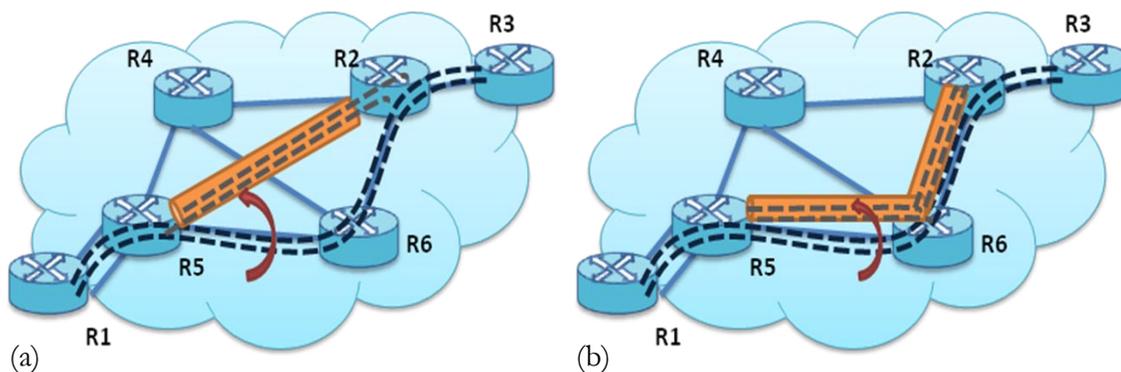


Fig. 5.7 Auto-Route: re-routing packets into a TE-LSP (tunnel). Tunnel is shown: (a) conceptually (b) physically routed over the IP links.

However this perspective does not materialize directly in the map over which packets are routed. The tunnels are *not* exposed to the distributed routing protocol i.e. routing adjacencies are not created across the tunnels. And so the map that the routing protocol creates in each router does *not* include the tunnels. The primary reason for not exposing tunnels to OSPF or IS-IS is the same $O(N^2)$ scaling issue discussed in the IP-over-ATM context in the previous chapter*. If the routing-protocol were exposed to the tunnels, then they would have to treat them as links and carry state for them, which is a scaling problem – as the number of links increases, more state needs to be disseminated; and when failures happen lots of control messages are generated, which in-turn increases router-CPU processing loads and could cause routing protocol instability.

[†] Assuming all links have the same IGP cost and the tunnel cost is that of a single-hop.

* See Sec. 4.2.1 under SDN based Unified Control Plane

But if tunnels are not allowed to show up in the IP-routing map, then how can packets be routed into them? This is where mechanisms like static-routes, PBR/FBF and Auto-Route play a role. Consider Fig. 5.8(a). MPLS-TE networks maintain *two* maps; one is used for regular SPF routing of IP packets; the other for CSPF routing of TE tunnels. All the mechanisms mentioned above can *change* the routing-decision made by SPF.

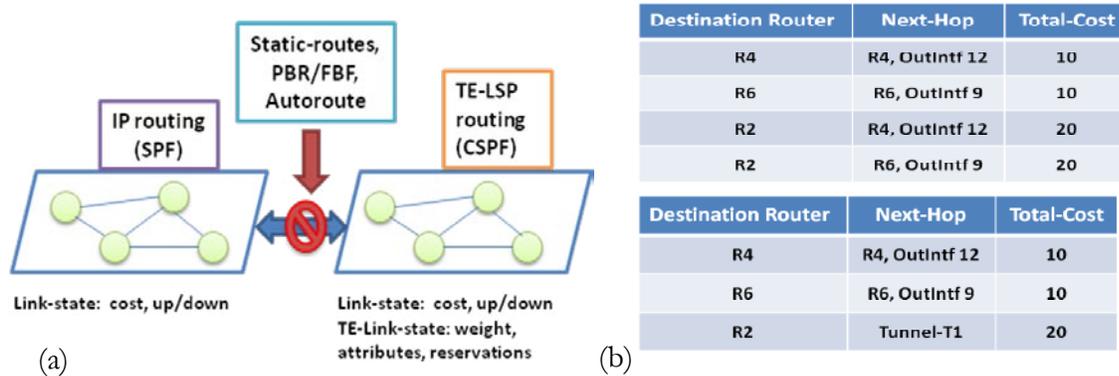


Fig. 5.8 (a) Need for Auto-Route (b) Effect of Auto-Route on Routing tables

Auto-Route, as the name suggests, re-routes packets into tunnels automatically [92]. It works by replacing routing decisions during SPF calculations. Consider the *partial* routing table shown in Fig. 5.10b calculated by R5 for the network shown in the upper part of Fig. 5.8b. It shows two equal-cost routes to R2 that have different next hops, R4 and R6; which R5 reaches via out-going interfaces 12 and 9 respectively. But invisible to SPF, R5 also has a tunnel to R2. Here is where Auto-Route kicks in (shown in the lower table in Fig. 5.8b) to reflect the effect of the tunnel. Both entries in the upper table are replaced with an entry where the next hop to R2 is the *tunnel* T1. The tunnel-interface is a logical (virtual) interface which is manifested behind-the-scenes by the physical interface (and label) on which the tunnel starts.

While Auto-Route is automated, it is also inflexible. It works very well for SPF routing based just on the destination-IP prefix. But it makes it hard to take a different (automated) approach to routing. For example, routing different types of traffic or

different customer's traffic differently is not possible with Auto-Route. If a different approach is required, then the network-operator has to fall back on static-routes and PBR, to supersede the decisions taken by SPF and Auto-Route. But while such mechanisms are flexible, they are not automated. They require human-intervention typically through the CLI in multiple routers along a path for each packet-flow that the operator may wish to route differently.

SDN approach to Auto-Route: SDN allows *both* flexibility and automation in its approach to routing. Together these attributes present a programmatic solution.

In contrast to Fig. 5.8a, with SDN we take the approach shown in Fig. 5.9. The network-map allows full visibility to node and link state. To this state (which comes from the switches), we add the link-state attributes we need for traffic engineering such as max-reservable-bandwidth, unreserved-bandwidth, priorities and other operator-defined attributes. TE routing of LSPs (tunnels) can be performed by an application on this map, and the resulting tunnels *are introduced as unidirectional links* in another map.

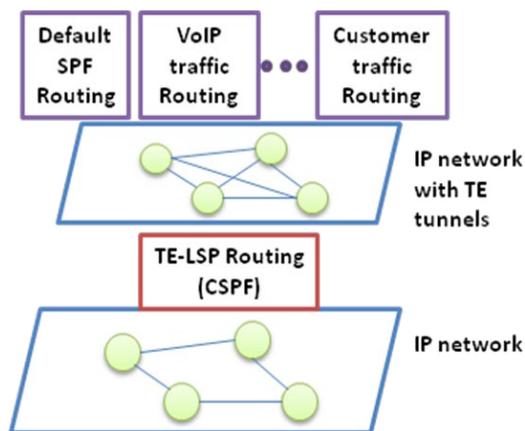


Fig. 5.9 Auto-Route in SDN

This latter map is used for routing packet-flows. We can simultaneously route different kinds of packet-flows differently – for example, traffic can be routed on the basis of the type of end-user application it supports (voip, web, video etc.); or it can be routed differently for different customers or services; all other traffic can use default IP

SPF routing. And the usage for each routing mechanism is programmatic, automated and dynamic.

We can take this approach because SDN *does not use* distributed link-state routing protocols to disseminate state; and so it is not limited by scalability issues of the same. In our implementation of Auto-Route (presented in the next section) the maps are updated by the Network-OS and switch-API (OpenFlow) which replace all the functionality of distributed routing protocols.

Load-Sharing: One of the direct consequences of Auto-Route is that it becomes harder to perform load-balancing in IP/MPLS networks. Especially for flows destined to the tail-end of a tunnel. Consider Fig. 5.7a. There are essentially three equal-cost routes from R5 to R2 - two that go via IP links and transit through R4 and R6, and the third is the tunnel from R5 to R2. Even assuming that the tunnel is routed on one of those paths (say $R5 \rightarrow R6 \rightarrow R2$), it still leaves us with two paths for load-balancing – the tunnel and the other path ($R5 \rightarrow R4 \rightarrow R2$). But due to Auto-Route (see lower table in 5.8b), only one of the paths – the tunnel – is used. In fact, due to the nature of Auto-Route, it is simply not possible to load-share traffic to a TE tunnel-tail between an IGP path (from SPF) and a TE tunnel-path [92]. The general solution to this problem is to create *multiple* tunnels to the same tunnel-tail and then load-share traffic between them. But the obvious drawback is that there are more TE-tunnels to create and manage.

SDN based Load-Sharing: Since we have more control over routing, and we represent tunnels as unidirectional links in the topology map, our approach to load-balancing is simple. In the above example, we simply route (and therefore load-share) traffic-flows over two paths – one that goes over the virtual-link that represents the tunnel, and the other over the IP links $R5 \rightarrow R4 \rightarrow R2$. In our implementation we perform load-balancing on a flow-level (for example by matching on the TCP/IP 5-tuple) in the control plane. But this can easily be performed in the data-plane with the ‘select’ group-construct, which is part of version 1.1 of the OpenFlow specification. Here the controller can add the physical port on R5 that leads to R4, as well as the physical port and label that represents

the start of the tunnel, to the same select-group to ensure load-sharing. Importantly we do not need two tunnels for load-sharing, and it does not matter if the two paths have the same cost or not.

DiffServ-Aware Traffic Engineering (DS-TE): DS-TE is a complicated marriage of two mechanisms: a) QoS based on the Differentiated Services architecture [115] that controls the per-hop behavior (PHB) of various traffic classes defined by the DiffServ Code Points (DSCP) bits; and b) MPLS-TE that controls the hop-to-hop (or path-level) behavior for IP traffic.

It is worth noting that on-their-own neither can *guarantee* QoS [92] on an end-to-end basis (within a network domain). For example, while DiffServ mechanisms can prioritize, shape and police traffic in class-based queues *within a switch*; it has no control over the path the traffic takes in the network domain. And so, congestion in a downstream node can cause traffic in some classes to suffer even with PHB control, as the congestion information is localized and not propagated back to upstream nodes. In this case, TE can help by trying to avoid congestion a-priori by steering traffic away via tunnels that reserve link-bandwidth and are admission-controlled.

However tunnel admission control is agnostic to the type (or class) of traffic that is being carried in the tunnel. And so traffic for a certain *class* of traffic cannot be steered away with regular TE. Therefore the purpose of DS-TE is to *enable tunnel admission-control on a per-QoS class basis*.

Consider Fig. 5.10a. It shows the queues that feed an outgoing port on which two (regular) MPLS-TE tunnels are routed. The TE tunnels reserve bandwidth on the link without taking into consideration the data-rates supported by the queues. Since the tunnels are unaware of the queues, they carry traffic for all class-types. Note that the highest-priority traffic in both tunnels comes out of the same queue. During times of congestion, it is possible that the min-rate guaranteed by the high-priority queue may not be enough to sustain all the high-priority traffic arriving at the queue.

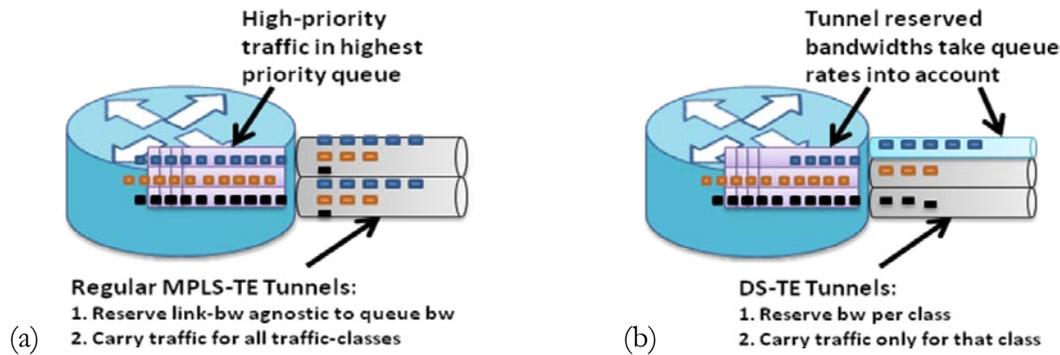


Fig. 5.10 (a) DiffServ with regular MPLS-TE (b) DS-TE

DS-TE tries to solve this problem by trying to steer away traffic for such a traffic-class (Fig. 5.10b). It does so by allowing tunnels to reserve bandwidth only for a certain class-of-traffic; out of a sub-pool of link-bandwidth that matches the queue-bandwidth for that class. For example, if the queue for the highest-priority traffic supports 100 Mbps, then tunnels for that class can only reserve link-bandwidth from the sub-pool of 100 Mbps, even if the max-reservable bandwidth may be 1 Gbps. This way if one tunnel for this class reserves 100 Mbps from the sub-pool, then other tunnels are not admitted on this link and thus forced to steer away from it.

SDN-based DS-TE: Diffserv-aware TE uses the same mechanisms as regular MPLS-TE; the subset of link-bandwidth advertized per-class, as well as per-class tunnel reservation and admission-control are all still purely control plane concepts. And so with SDN we can perform all of these tasks in the Controller. But the SDN based implementation has two distinct advantages:

- The process of forwarding class-specific traffic down a DS-TE tunnel is not trivial and hard to automate in networks that use DS-TE today. Auto-Route cannot help as the mechanism does not distinguish between traffic-classes. The only choice is the use of Policy Based Routing (PBR) [92], which requires configuration via manual-intervention and is not very programmatic. This is essentially the same issue as pointed out in the section on Auto-Route. With SDN and the use of OpenFlow,

forwarding traffic-type (or class) specific packet-flows down a tunnel is flexible, automated and therefore programmatic.

- DS-TE currently allows the advertisement of only one subset (called sub-pool) of link-bandwidth, and so can be used for only one traffic-class [116]. Herein lies the power of SDN. If operators tomorrow wanted the ability to create traffic-type specific tunnels for not one but *two or more* classes of traffic, it would *require a change to the distributed routing protocol* to carry the advertisement of more bandwidth sub-pools (plus the associated upgrades to router-software). In SDN, no protocols would change. Only the CSPF and related applications in the controller need to be changed.

Auto-Bandwidth: In the previous section we discussed the Auto-Bandwidth feature and associated problems with local-optimizations and network-churn. Here we would like to focus on its implementation. Auto-Bandwidth is implemented in the routers at the head-end of TE tunnels on a per-tunnel basis. Periodically the router collects information on the actual bandwidth-usage of the tunnel (eg. every 5 mins). At a different periodicity, it alters the bandwidth-reservation for the tunnel based on the observed usage. The change in the reservation for the tunnel is signaled to all other routers in the LSP; it also changes link-state which is then disseminated to all other routers in the network by the routing-protocol.

SDN-based Auto-Bandwidth: In our implementation, since all tunnel reservations, link-state and admission-control is maintained and performed in the Controller; Auto-Bandwidth is performed as an application in the Controller as well. The only thing we need from the switches is information on the usage of the tunnel. OpenFlow defines an LSP simply as another flow (instead of a virtual-port). Thus we can obtain flow-statistics (like transmitted-bytes) from the switch via OpenFlow, to gauge LSP usage. The application-programmer can set intervals for polling switches and adjusting the bandwidth-reservation. Changes in bandwidth-reservation are not signaled to the data-plane as the latter is unaware of any bandwidth reservations for the LSPs. Furthermore,

changes to link-state that come about from the change in the tunnel reservation, requires only updating the annotated topology-map and not the switches. The only time that Auto-Bandwidth induced change needs to be propagated down to the data plane is when it causes one or more tunnels to re-route. In such cases, the Controller processes the change by removing flow-state for the old tunnel route and replacing it with the new tunnel-route in the affected LSRs. As we have mentioned before, this can cause significant churn in a large network with many tens-of-thousands of LSPs. But with SDN, the effect-of-churn is minimized as routers no longer have to deal with RSVP or OSPF state-update messages. They only need to process/generate OpenFlow messages that change flow (LSP) routes. In Sec. 5.4, we'll discuss a way in which Auto-Bandwidth can be eliminated altogether with SDN.

LSP-Priorities: One final MPLS-TE feature we cover is LSP priorities. TE-LSPs can reserve bandwidth at eight different priority levels (with 0 being the highest priority and 7 the lowest). Accordingly links advertize un-reserved bandwidth at each of the priority levels, for both the global pool and the solitary DS-TE sub-pool. LSP priorities are used for preemption – a higher priority LSP can force a lower priority LSP to re-route if there isn't enough un-reserved bandwidth available for both of them on a link they both share. For example, if a link has maximum reservable bandwidth of 900 Mbps; of which 500 Mbps has been reserved by a Priority-0 tunnel; and 300 Mbps by a Priority-1 tunnel; then 400Mbps of unreserved bandwidth is available at Priority-0, and only 100Mbps is available at Priorities 1-7. But if a new Priority-0 tunnel comes along reserving 200Mbps; then all three tunnels cannot co-exist on the link as their cumulative reservations exceed the maximum reservable bandwidth on the link. The lower priority-1 tunnel is forced to re-route – the head-end for this tunnel periodically tries to find a new path which satisfies all its constraints.

SDN based LSP-Priorities: As with all other LSP attributes, the LSP-priority is just a number maintained in the annotated topology map within the Controller. The data-plane switches have no information on LSP priorities. We maintain tunnel priorities in the

controller, as well as link-state for reservable bandwidth at eight priority levels. CSPF is performed in the Controller; tunnel-priorities are taken into consideration; and any resultant pre-emption is achieved using OpenFlow messages to add, delete or modify flow-entries in LSRs corresponding to TE tunnels.

5.3.2 MPLS-TE Prototype

In this section we discuss a prototype implementation of MPLS Traffic-Engineering in an SDN based network (MPLS-TE). There are two goals of this implementation: First, we demonstrate that our approach is capable of reproducing existing services offered by MPLS network today, while validating the simplicity of our approach compared to the existing IP/MPLS control plane; and second, as described in the previous section, we show how the implementation of the features can be made different from the traditional implementation in ways that either, greatly simplifies the feature or provides a feature that MPLS-TE cannot provide.

Data-Plane: At the time this work was done (late 2010), handling of MPLS labels was just being introduced into the OpenFlow protocol (version 1.1[117]). As a result, there weren't any OpenFlow enabled switches (or controllers) that supported MPLS data-plane capabilities via OpenFlow.

We added MPLS data-plane capabilities to a software switch called Open vSwitch (OVS[118]) that fully supported version 1.0 of the OpenFlow protocol. We also had to change the OpenFlow protocol (v1.0) to add the capability of controlling an MPLS data-plane. To accomplish the latter, we borrowed the MPLS related features from v1.1 of the protocol and added it to v1.0[†].

[†] We chose not to implement the entire v1.1 spec as it includes a number of other features like groups and multiple-tables, which were not essential to this experiment.

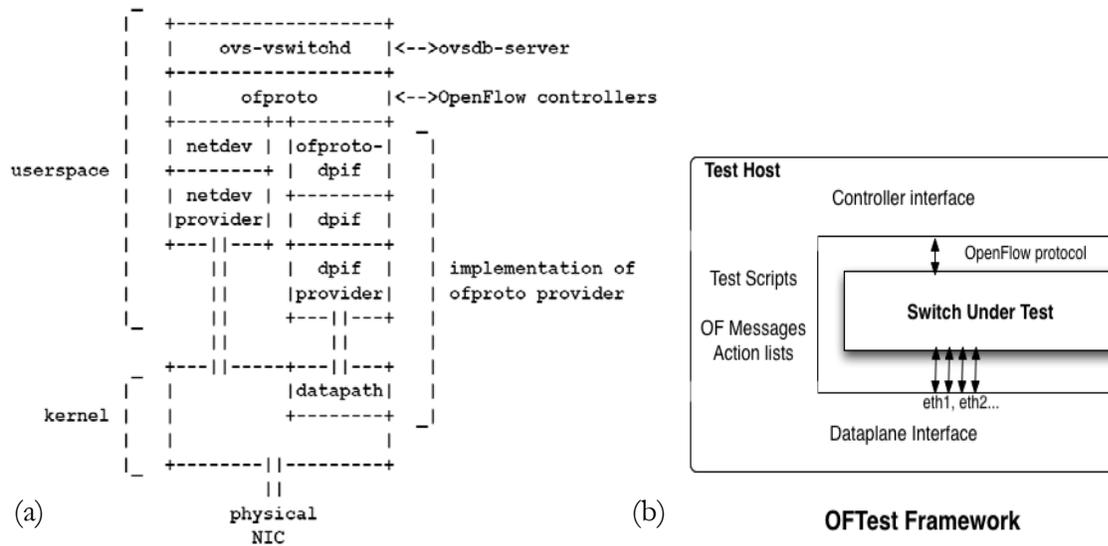


Fig. 5.11 (a) Open vSwitch Software Architecture [119] (b) OFTest Framework [120]

Fig. 5.11a shows the OVS software architecture. Details can be found in [119]. Here we focus on the parts we changed to add MPLS data-plane capabilities. OVS can be used purely as an OpenFlow switch, or it can be used with added functionality that is part of OVS. We chose to use it as an OpenFlow switch using the `ovs-openflowd` program instead of the `ovs-vswitchd` program. The former does not require communication with an `ovsdb-server`. Next in the stack shown in Fig. 5.11a, the `ofproto` module communicates with an external Controller. We changed this module to work with our additions to version 1.0 of the OpenFlow protocol borrowed from version 1.1 (for MPLS support). The `ofproto` module communicates with the switch using the `ofproto-dpif` interface.

The OVS switch datapath can be in kernel space (as shown in the figure) or in userspace. We chose the userspace datapath implemented in the `dpif-netdev` module, which also instantiates the `datapath-interface (dpif)` provider class. We added support for the MPLS data plane in the `dpif-netdev` module [120]. This included the ability to match on the outermost MPLS label (id and traffic-class bits) and perform the actions of pushing, changing and popping off multiple labels from packets together with TTL manipulations. Since our software switch was hosted in Linux, we used the built-in

netdev-linux module that abstracts interaction between ofproto and Linux network-devices (like Ethernet ports).

We tested our implementation of the MPLS data-plane using the OFTest framework [121,122] shown in Fig.5.11b. OFTest includes a data-plane interface that allows sending and receiving packets to a switch-under-test (OVS in this case). It also includes a control-plane interface that implements a light-weight OpenFlow controller that can be used to interact with the switch-under-test using the OpenFlow protocol. In our case this controller communicates with the ofproto module in OVS. We used OFTest to write test-scripts that verified the correct functionality of the MPLS data plane.

Control-Plane: The control-plane was hosted in a server running NOX [15]. NOX was modified to include our changes to version 1.0 of the OpenFlow protocol, thereby enabling control of an MPLS data-plane [123].

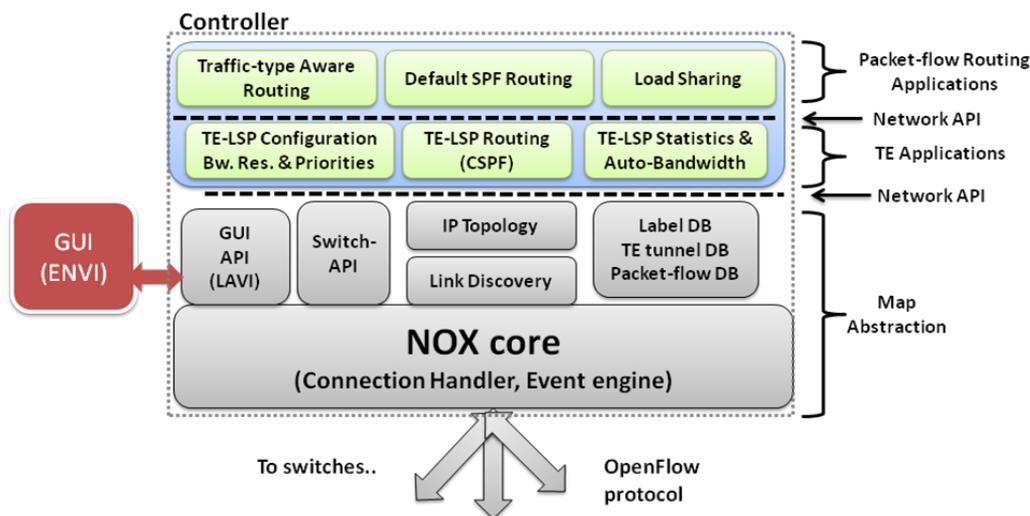


Fig. 5.12 MPLS-TE NOX Software Architecture

The software architecture is shown in Fig. 5.12. We use NOX's basic event engine and connection-handlers to the switches that are collectively referred to as nox-core. We also use NOX's link discovery module to discover IP links and construct the annotated topology-map. And we include several data-bases for packet-flows, tunnels and MPLS-

label information. Together with the switch and network-APIs, these modules present applications with a map-abstraction.

The first set of network-applications involves TE functionality. The TE-LSP configuration module is responsible for reading tunnel configuration files or alternately responding to config-messages from a GUI. Tunnels are configured by specifying the head-end and tail-end routers; the desired bandwidth and tunnel priority; the type of traffic the tunnel is designated for; and whether Auto-Bandwidth is enabled on the tunnel or not. With this information the config module uses the TE-LSP routing module to find a path between head-end and tail-end routers that satisfy all the constraints (bandwidth, priority, traffic-class). Once a path is found, the config module uses the network-API to establish the tunnel in the data-plane by inserting flow-entries for the LSP which includes label information. If Auto-Bandwidth has been configured on the tunnel, it also activates the polling process for LSP usage information in the TE-LSP statistics module. This information is used by the Auto-Bandwidth module to periodically change the tunnel reservation and update the TE-LSP database. But before such an update happens, the Auto-Bandwidth module checks the CSPF module to make sure that the new-bandwidth reservation is possible along the existing tunnel path. If it is possible but requires re-routing lower priority tunnels or re-routing the tunnel itself, it processes such change by invoking methods in the network-API.

Together the TE applications create, manage, re-route, and tear-down TE-LSPs (tunnels) in the data-plane. As explained in the previous section, our map-abstraction allows us to create another map above the TE applications (see Fig. 5.9). This latter map represents the TE tunnels as unidirectional links, on which packet-flow routing can be performed in an automated and programmatic way. We create three routing-modules for packet-flow routing. The default SPF routing-module uses NOX's built-in all pairs shortest-path routing module. The only difference is that instead of using this module directly on the IP topology, we use it on the topology that includes the IP links as well as the tunnels represented as unidirectional links. The default routing module is used for all

packet-flow routing. The other two modules are plugins into the default routing modules. The traffic-type aware routing module ensures that only those flows of a certain traffic-type (voice, video, web) are allowed into tunnels meant for the traffic-type (DS-TE). Essentially we use traffic-type to emulate traffic-classes (without actually setting or using the DSCP bits). And the load-balancing module ensures that traffic-flows can be load-shared between regular IP-links and links representing tunnels, to the tunnel-tail.

Experimental Setup: Our network of software switches is created in the Mininet environment [124]. Mininet is an open-source tool that creates a network of interconnected software switches on a single PC for emulating networks. It uses Linux process-based virtualization to run many hosts and switches on a single Linux kernel [125]. We use Mininet to emulate a wide-area IP/MPLS network. Our prototype system runs 15 instances of OVS (modified) to form the IP/MPLS network (Fig. 5.13).

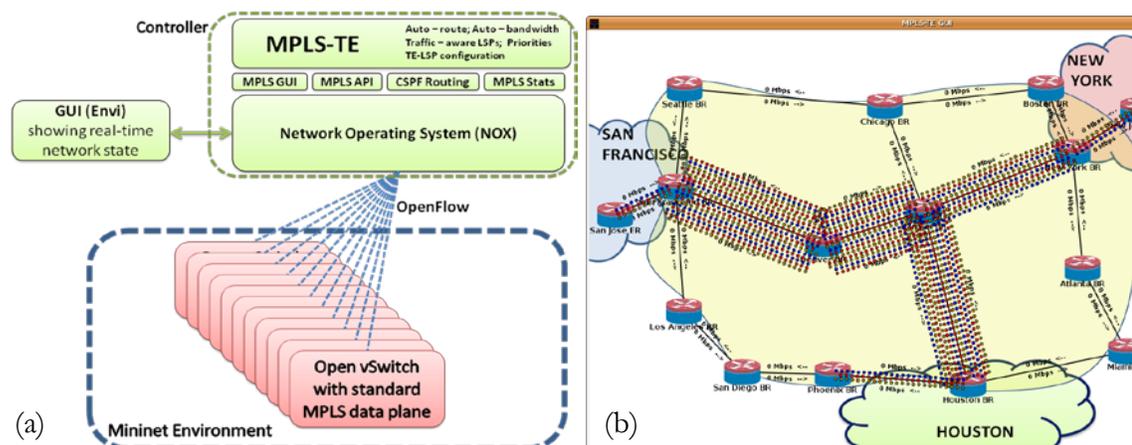


Fig. 5.13 MPLS-TE (a) Prototype (b) GUI view showing Emulated IP/MPLS WAN

Fig. 5.13b shows the GUI view for our emulated-WAN. The GUI shows network-state re-created in real-time via communications with the Controller. The emulated-network has nodes in 15 cities spanning the continental US. In the San Francisco (SFO) and New York (NYC) clouds, San Jose and New Jersey connect to the wide-area via the routers in SFO and NYC respectively. In Fig. 5.13b the GUI shows wide-area IP links between the cities. Unidirectional TE-tunnels are routed on these bi-directional IP links

by reserving bandwidth on the links in the direction of the tunnel (from head-end to tail-end). The GUI displays reserved-bandwidth information for each link in both directions. Each link is assumed to be a GE link, with a maximum reservable bandwidth of 90% of the link-bandwidth (i.e 900 Mbps). Finally, Fig. 5.13 also shows traffic-flows for three different traffic types (voice, video and web traffic) originating from traffic-generators (not shown) in San Jose and New Jersey.

Experiments: In the previous section we discussed several features related to MPLS-TE. The primary goal of this experiment is to achieve the basic functionality of traffic-engineering itself. Secondary goals include the ability to show TE related features such as Auto-Bandwidth and DS-TE.

Tunnel-Routing and Packet-Flow Routing: Traffic Engineering in MPLS is a two-step process: a) admission-control of bandwidth-reserved TE-LSPs (or tunnels) performed during a CSPF routing process for the TE-LSP; and b) (Re-) Routing packet-flows through a newly created TE-LSP. In Fig. 5.13b, we see the operation of a network without traffic-engineering. Traffic flows from the SFO and NY areas, destined to Kansas, Phoenix, Houston and NY, all follow the shortest path in the network; potentially congesting the $SFO \leftrightarrow DEN \leftrightarrow KAN \leftrightarrow NY$ links.

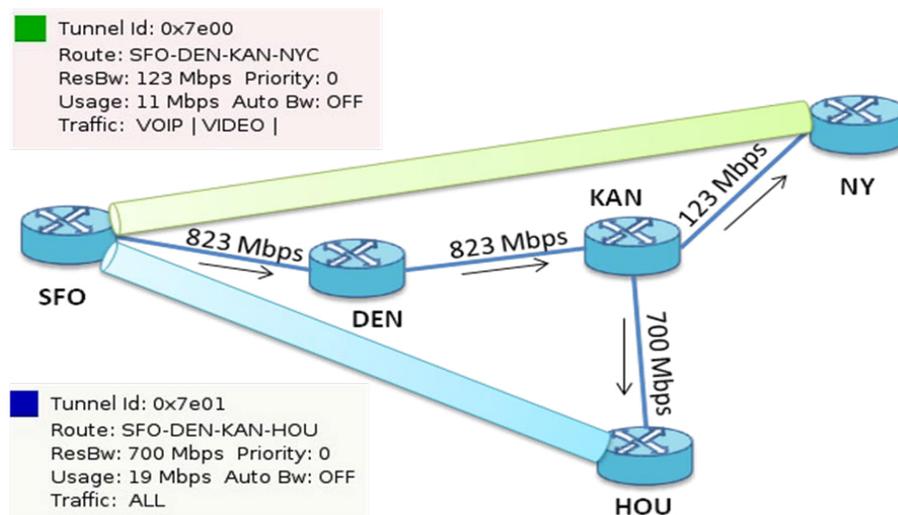


Fig. 5.14 Tunnel Routes, Bandwidth-Reservations & Auto-Route

In Fig. 5.14 we show two tunnels that were configured by the network-operator, routed by the CSPF module, and established in the data-path using OpenFlow, to enter flow-entries in the OVS flow-tables. The tunnels originate in SFO (head-end) and terminate in Houston and NYC (tail-ends).

The panels show information for each tunnel. It is worth noting here that Fig. 5.14 shows the tunnels from a conceptual standpoint (similar to Fig. 5.7a). But the tunnels are actually routed over the IP links. For example, the SFO→NYC tunnel is routed along the SFO→DEN→KAN→NYC links, reserving 123 Mbps of bandwidth on each link in the eastward direction. Similarly, the SFO→HOU tunnel is routed via SFO→DEN→KAN→HOU links reserving 700Mbps of bandwidth along the path. But another way to think about this is that this is also the annotated-topology-view presented to the packet-flow routing applications in Fig. 5.9 or Fig. 5.12. The tunnels are, from a packet-flow routing standpoint, uni-directional links which form the shortest (1-hop) path between routers. And so the routing-modules perform Auto-Route, as described in the previous section, by re-routing packet-flows into the tunnels.

For example, flows from SFO→NYC which previously traveled over the IP links now get re-routed (or Auto-Routed) via the 1-hop tunnel. It so happens that the tunnel is also routed along those same links. But this does not matter – the packets get imposed with labels in SFO and get label-switched in the DEN and KAN routers, which no longer have visibility into the IP packet. The flow-entries in DEN and KAN which previously matched on the IP-information in the packet-header, subsequently idle timeout, as they no longer match on the labeled-packets. We also use penultimate hop popping (PHP) of the label at the KAN LSR so that the NYC LSR receives the unlabeled IP packet; and so it only has to do a single lookup to forward the packet.

Another aspect of our design worth pointing out is that for the packet-routing applications to treat the tunnels as unidirectional links, they need to have the concept of a virtual-port. For example, the SFO→NYC tunnel is represented as a unidirectional link connecting a virtual-port of the SFO router to a virtual-port on the NYC router. But the

packet-flow routing modules are un-aware that the ports are virtual-ports. They simply decide to forward traffic out of these ports (eg. in SFO) and match on traffic incoming from these ports (eg. in NYC). But these virtual ports *do-not-exist* in the switches, as an OpenFlow switch treats LSPs like any other flow, instead of treating it as virtual ports (tunnel-interfaces). And so the TE- applications translate decisions made by the packet-routing applications from a virtual-port to the corresponding physical-port and label.

Admission-Control: We mentioned that tunnels SFO→NYC and SFO→HOU reserved 123 Mbps and 700 Mbps of bandwidth on all links that are part of their route. Note that in Fig. 5.14 we show these reservations circled in the eastward direction – for example, the KAN→NYC link shows a reservation of 123 Mbps (due to the SFO→NYC tunnel) and the DEN→KAN link shows a reservation of 823 Mbps (due to both tunnels being routed over the SFO→DEN→KAN links).

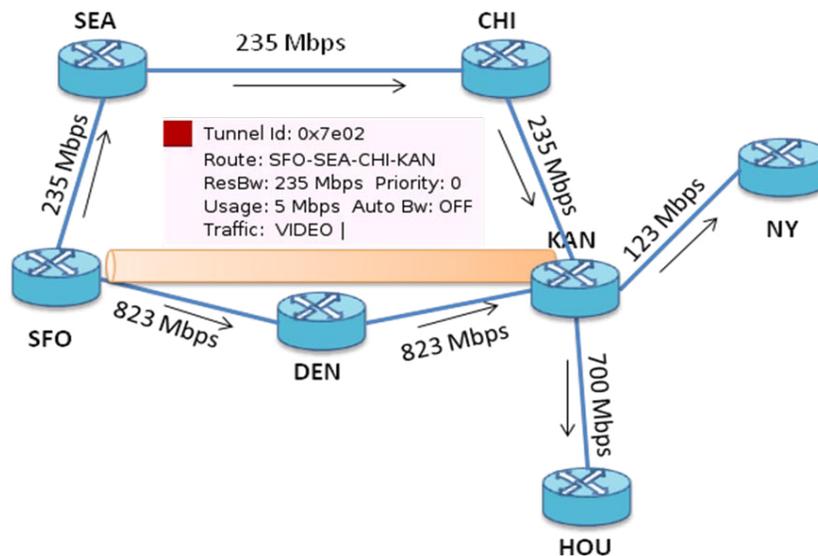


Fig. 5.15 MPLS-TE Admission-Control

Also note that the maximum reservable bandwidth on any link is 900 Mbps. Thus only 77 Mbps of bandwidth is un-reserved on the SFO→DEN→KAN links. Thus when the SFO→KAN *tunnel* highlighted in Fig. 5.15, requests 235 Mbps of bandwidth, it exceeds the unreserved bandwidth on the shortest path; and so the SFO→KAN tunnel is

forced to route along a less-utilized path of the network – the SFO→SEA→CHI→KAN links. This is an example of admission-control. Without it, the SFO→KAN tunnel would be routed along the same SFO→DEN→KAN links as the other two tunnels; and if the tunnels actually used the all the bandwidth they reserved ($123+700+235 = 1058$ Mbps) it would have congested the links.

Auto-Bandwidth and Priorities: Our next experiment involved the TE features of Auto-Bandwidth and tunnel-priorities and their interaction. Fig. 5.16 highlights two tunnels that originate from NYC: one with a tail-end in Houston (HOU) and the other with a tail-end in Phoenix (PHX).

Note from the panel for the NYC→HOU tunnel, Auto-Bw is turned on (unlike any of the other tunnels in Figs.5.14-15). Accordingly the bandwidth reservation for this tunnel tracks the actual usage of the tunnel. Although the tunnel was created (configured) with a bandwidth-reservation of 10 Mbps, the reservation increases as we channel more traffic through the tunnel (by turning on more traffic-generators); first up to 19 Mbps and then to 39 Mbps (the latter shown in Fig. 5.16).

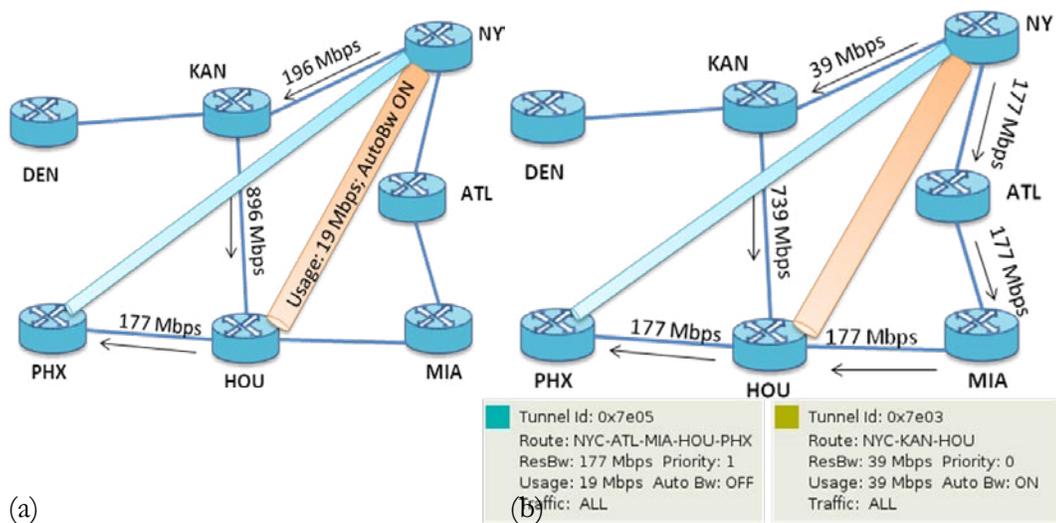


Fig. 5.16 Interaction of Auto-Bandwidth and Tunnel Priorities

But another effect happens along the way. Note that the route taken by the NYC→HOU tunnel includes the KAN→HOU link and reserves (first 10Mbps and then)

19 Mbps on it. Additionally the SFO→HOU tunnel also reserves 700 Mbps on the same link. The initial route taken by the NYC→PHX tunnel goes via the NYC→KAN→HOU→PHX links as it is the shortest path (in hop-counts) that satisfies the bandwidth required by the tunnel. Accordingly it too reserves 177 Mbps on the KAN→HOU link (as shown in Fig. 5.16a). This adds up to a total of 896 Mbps reserved on the KAN→HOU link which is very close to the maximum reservable amount of 900 Mbps. And so if we further increase the traffic going through the NYC→HOU tunnel to 39Mbps; then since Auto-Bandwidth is configured ON for this tunnel, it changes the reservation to 39 Mbps (as shown in Fig. 5.16b; which in-turn tries to increase the total reservation on the KAN→HOU link to 916 Mbps, which is not allowed. At this point some tunnel has to find another path.

Note that the NYC→HOU tunnel has priority-0 (highest priority) while the NYC→PHX tunnel has priority-1. Thus it is the lower priority tunnel that is forced to re-route and find another path that can meet its constraints. In Fig. 5.16b, we show that the tunnel has re-routed via the longer NYC→ATL→MIA→HOU→PHX path. This is an example of the dynamic interaction between Auto-Bandwidth and tunnel Priorities, which in a larger network with lots of tunnels, can cause significant network churn.

Class-Based Routing and Load-Balancing: Finally in Fig. 5.17 we show two examples of routing where we deviate from shortest-path routing. Note that Fig. 5.17 is a GUI snapshot that shows all five tunnels we have instantiated so far.

In the first example we implement routing based on traffic-classes. As discussed in the section on DS-TE, tunnels can be created for a traffic-class (identified by the DSCP bits in the IP header) by reserving bandwidth from a sub-pool of link-bandwidth resources. This way per-class admission control can be performed for tunnels designated for a class. The second part of DS-TE is to actually route traffic for that class into the specific tunnel, which today uses cumbersome, non-programmatic techniques. With OpenFlow, routing a traffic-class becomes easy and programmatic. In Fig. 5.17, we show the SFO→KAN tunnel designated to carry only video traffic, while the SFO→NYC

tunnel carries both video and VoIP traffic. In the SFO router, we simply use OpenFlow to enter rules that match on both destination IP address and the L4 transport-ports to indentify the tunnel and the traffic-type respectively; and then insert the appropriate label (as an ‘action’ on matching packets) for the tunnel the packets needs to enter. We can also use the IPv4 DSCP bits to match on a traffic-class (if for example the San Jose router marks the bits).

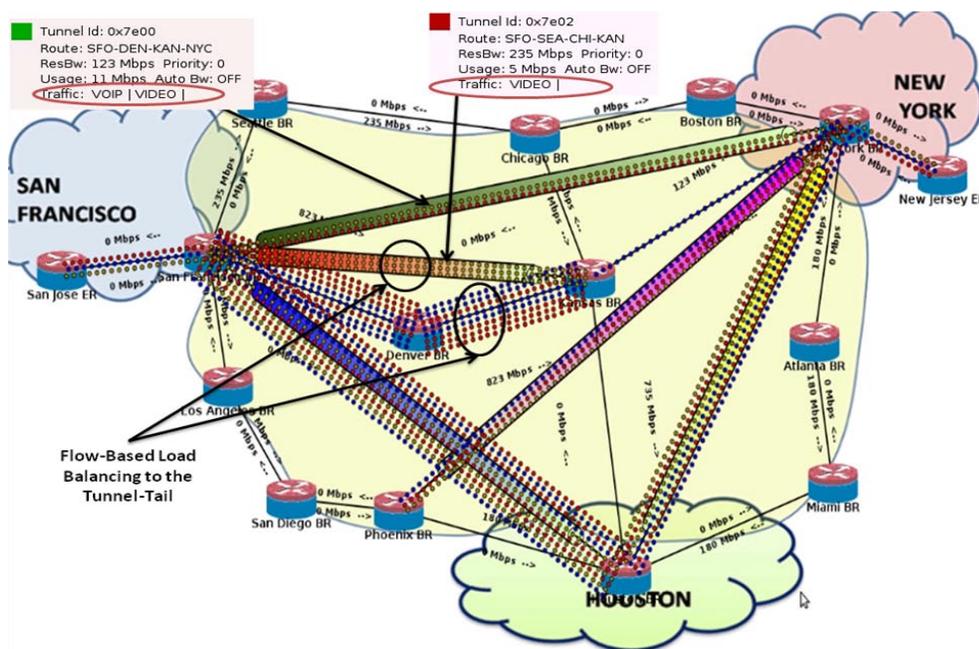


Fig. 5.17 GUI Snapshot: Routing of Traffic-Classes and Load Sharing

In this experiment, we do not perform per-class admission control as we did not reserve bandwidth from class specific sub-pools of link bandwidth. But this is a simple change to our CSPF algorithm (and nothing else). More importantly we can create multiple sub-pools each for a different traffic-class; to match with tunnels created for multiple traffic-classes (as shown); which is something that MPLS-TE cannot provide today without changing the routing protocol.

The second example of deviation from shortest-path routing shows load-balancing. In Fig. 5.17, traffic between SFO and KAN takes two routes: Video flows go through a

tunnel from SFO→KAN, which is actually routed via the Seattle and Chicago routers; and all other flows take the IP links between SFO→DEN→KAN. This is an example of load-sharing to a tunnel-tail, between a tunnel-path and an IP-link-path; a feature that MPLS-TE cannot provide today due to the nature of Auto-Route in MPLS-TE (as discussed in the previous section).

Conclusions: To summarize, we draw the following conclusions from our SDN based MPLS network prototype:

- We achieved the initial goal of verifying our architectural ideas of introducing the map-abstraction in MPLS networks. We showed that we can not only perform the basic operations of creating and maintaining LSPs in an OpenFlow enabled MPLS data-plane; but we can also provide higher-level services such as MPLS-TE by writing applications above a map-abstraction purely in the control-plane.
- We validated the simplicity of our approach when compared to more traditional approaches seen today. Our implementation of the traffic-engineering application together with all the features described in Sec. 5.2.1, took only 4500 lines-of-code. To accomplish the same in today's network would require implementations of OSPF-TE and RSVP-TE (together ~ 80k LOC) and implementations of all the features. At the time of this writing we could not find an open-source implementation of MPLS-TE features; and so we can only estimate, based on experience that each feature could take at a minimum 5-10k lines-of-code, pushing the total LOC well above 100k. It is easy to see that our SDN based implementation is much simpler given the two-orders of magnitude difference in the lines-of-code required. While our implementations is not production-ready, two orders of magnitude difference gives plenty of room to grow; giving us confidence that even production-ready code will be simpler to implement with the map-abstraction.
- Finally, we achieved our second goal of demonstrating that using SDN and the map-abstraction, we can implement the TE features in ways that a) either greatly simplify the feature; or b) provide a feature that MPLS-TE cannot provide. Examples of the

former include (i) our implementation of Auto-Route in a more flexible and programmatic way; and (ii) contention-free LSP establishment; while examples of the latter include (i) load-sharing traffic to a tunnel-tail, on a tunnel and an IGP path; and (ii) performing per-class admission control for more than one class of traffic.

5.4 Other Applications of SDN based MPLS

The map-abstraction brings the benefits of a simpler, more-extensible control plane to IP/MPLS networks, and provides network-applications with a global-view of network state. Together with the flow-abstraction in the MPLS data-plane and a switch-API like OpenFlow, MPLS networks based on SDN could potentially solve problems seen in today's networks. In our talks with Tier-1 ISPs, we are beginning to understand issues that carriers face in MPLS networks and how the SDN approach could help meet those challenges. Briefly, we cover a few of these issues in this section.

MPLS-VPNs: VPNs in MPLS come in two main flavors – L3VPNs and L2VPNs (there are variations of the latter). While both services use an MPLS backbone, LSPs in the former carry IP packets, and in the latter, Ethernet frames. Nevertheless, in either case, the primary objective of the MPLS backbone is to provide routing/switching information dissemination and isolation.

Building on our TE work, we recently demonstrated MPLS L3-VPNs [109]. We interactively created multiple isolated L3-VPNs, with overlapping client private-IP address spaces (customer-routes). The controller maintains VRFs for each VPN that a Provider-Edge router is part of. The VPNs also have customer specified topologies, and are supported by TE tunnels in the backbone.

We are beginning to understand the scaling implications of L3-VPNs. Today one of the main problems in L3-VPNs is that the service provider has to carry/propagate millions of customer IP addresses using protocols like MP-BGP. The SDN approach can

provide relief in this regard as it can reduce routing burden, by eliminating the need for protocols such as LDP, IS-IS and MP-BGP in the backbone.

Eliminating Auto-Bandwidth: In the previous section we explained how Auto-Bandwidth is a local-optimization technique [110], which together with LSP-priorities, can produce significant network churn. But operators have other options – notably the use of ‘offline’ (not on the routers) modeling tools, that have global-view of the network and can optimally route LSP paths. However the use of offline-tools introduces a different problem. Once the tools calculate LSP paths for tens-of-thousands of LSPs, the LSPs themselves have to be established or re-routed in a live-network. This process involves ‘locking’ head-end routers one-by-one and feeding it configuration files for LSPs that it needs to originate (via signaling between routers). The compilation of the configuration takes time, and the order in which tunnels are ‘brought-up’ has to be carefully coordinated. And all of this has to be done in a network carrying live-traffic.

But with SDN, operators have the ability to take an ‘offline’ optimization tool ‘online’. The map-abstraction offers the full-visibility required by optimization-tools (which can run as applications). But more importantly, the Controller can access the results of the optimization tool and use OpenFlow to directly manipulate the forwarding tables of all LSRs. In other words, OpenFlow acts as a switch-API that *dynamically* updates flow-table state, and therefore LSP state. And it can do so in parallel, for multiple LSPs, originating from multiple head-ends, while minimizing network-churn.

Improving Fast Re-Route Performance: From our talks with network-engineers at Tier-1 ISPs, we find the MPLS Fast Re-Route (FRRs) techniques perform poorly in operation. In FRR, backup LSPs are created to ‘locally-repair’ primary-LSPs [111]. The backup LSPs are pre-computed and signaled prior to any failure. Local-repair refers to re-directing traffic as close as possible to the failure. Two techniques exist for performing FRR: one-to-one backup and facility backup. But irrespective of the technique used, the problem with FRR is that, in a large ISP network, a) there are far too many nodes and links to backup; and b) because the number of backup tunnels is high,

these backup tunnels usually do *not* reserve-bandwidth on the links. And so it is an intractable problem to have any intelligent way for optimally routing backup-paths while backing up everything in the network. Although FRR is fast (< 50ms recovery), it is effectively a blunt protection mechanism. It is entirely possible that the result of an FRR re-route can cause congestion over a backup-path.

The key with SDN and OpenFlow is that it is dynamic - so when a failure happens, one can use FRR as it is today to do what it does today (as meta-stable state), just to give the Controller enough time to optimize around the specific failure. Note that this optimization is no longer an intractable problem, as the Controller knows exactly what failed. And so it does not have to plan for every possible failure, and then it can use OpenFlow to update the LSP-state dynamically once it has optimized around the failure. The net effect is that FRR can perform better by maintaining the speed that comes from local-protection, but availing of optimally routed backup paths that are dynamically updated in LSR flow-table state.

MPLS-TP/Multi-layer Control: Finally we consider applications of MPLS beyond the IP network. MPLS has been favored for establishing a *packet-based data-plane* technology for the transport network; Hence the name *MPLS-Transport Profile* (MPLS-TP) [113]. The basic idea is that there is a need for transport networks that are packet-switched[†], but such packet-networks should retain/reproduce the capabilities of traditional circuit-based transport networking, especially in providing equivalent OAM capabilities. OAM is provided using Generic Associated Channel packets in MPLS LSPs, or pseudowires [114]. Importantly, [113] points out that MPLS-TP should *not* depend on IP networks nor require dynamic control-planes for operation.

However, in our talks with industry, we find that a simple, dynamic control plane like an SDN based control-plane can offer value to an MPLS-TP network. Transport network operators, traditionally against the use of complex control-planes like those in IP/MPLS networks*, may find simpler OpenFlow based control more to their liking. And its value increases when considered in multi-layer scenarios. Transport network operators

[†] The Related Works section in Ch. 6 discusses MPLS-TP in more detail.

* See Challenge #2 in Sec. 3.4

frequently run multiple networks based on different technology layers simultaneously—ROADMs (Layer 0), SONET/SDH (Layer 1), OTN (Layer 1), to which they seek to add MPLS-TP (Layer2). And so instead of doing everything manually, or requiring separate control planes for each layer, a single OpenFlow/SDN based control plane for all layers can be an attractive option.

We believe that we have only started to scratch the surface for SDN applications in MPLS networks. In the next section, we show how we implement one such application.

5.5 Synergies with Packet-Circuit Control

In this section we take a brief look at the synergies between SDN based control of IP networks with MPLS in the data-plane and control of IP networks with Dynamic Circuit Switching. In Fig. 5.18 we show the use of the map-abstraction for control in both cases.

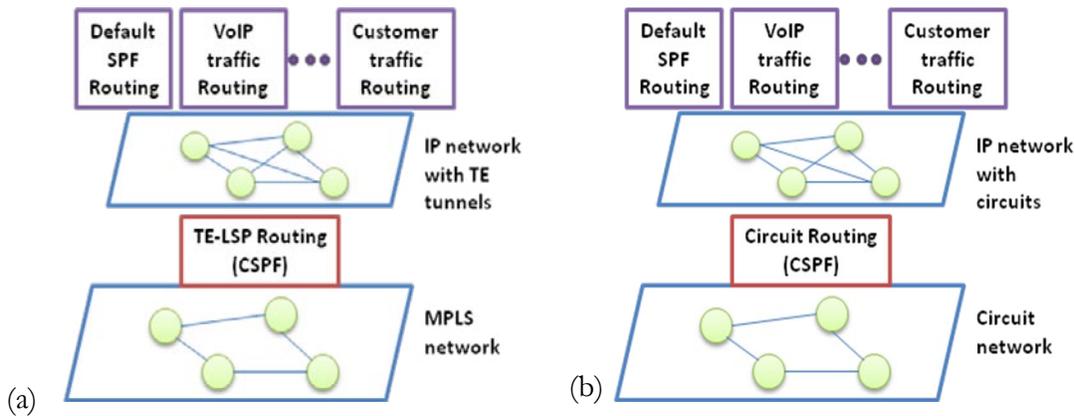


Fig. 5.18 Control of (a) IP and MPLS (b) IP and Dynamic Circuits

In an IP/MPLS network (Fig. 5.18a), the map-abstraction presents an annotated-topology map of the IP network, on top of which we perform TE-LSP routing based on TE-link attributes (like reservable-bandwidth, weight etc.) that we assign to the IP links. The routing uses a CSPF algorithm to determine tunnel routes. When IP is used with dynamic-circuits, in one construction (Fig. 5.18b), the map-abstraction presents an

annotated fiber topology, on which circuits are routed using CSPF and attributes we assign to the fiber/WDM-link. The net result of using CSPF in both cases is an MPLS-tunnel or a TDM/wavelength-circuit.

But in both cases they are represented as unidirectional or bidirectional links in the IP network, which is just another annotated-topology map in the SDN controller (the upper ones in Fig. 5.18). In other words circuits and TE-tunnels are represented in similar ways, and both can be dynamic with varying bandwidths and routes. And mapping packet-flows to circuit-flows is really the same as mapping packet-flows to TE tunnels, both of which we have demonstrated in similar ways with voice/video/web packet-flows in Chapters 3 and 5. In both cases, the mapping in the control-plane happens via a virtual-port abstraction[†]. The only difference is that the virtual-port is a purely control plane construct in the MPLS case; while in the circuit-case it represents the data-plane translation from packet-frames (like Ethernet) to framing used in circuit networks.

It is important to note that neither scenario shown in Fig. 5.18 is used today. TE tunnels are not represented as links in the IP network; the IP network does not treat packets as flows; and dynamic-circuits are not used in concert with IP network to dynamically create/modify/delete IP links. Yet the use of the map-abstraction together with the treatment of packets, tunnels and circuits as flows, enables a network-operator or service-provider to have a simple consistent view of their network.

5.6 Summary

The goal of this chapter was to introduce the map-abstraction in MPLS networks, argue for its benefits, and validate our architectural constructs and claims. Our motivation for doing so comes from discussions with service-providers who run large IP/MPLS backbones. In general they feel that while MPLS has delivered on new services like TE and VPNs, it has failed to live up to its promise of a simpler, cost-effective data-plane,

[†] See the sub-section titled ‘Tunnel Routing and Packet-Flow Routing’ in Section 5.2.2; and the sub-section titled ‘Mapping Packet-flows to Circuit-flows’ in Sec. 2.1.2.

due in large part to the burden of bearing a tightly-intertwined distributed control plane of ever-increasing complexity.

We showed that the MPLS data-plane is simple and useful in its label-stacking and push/swap/pop operations. And while it is less-flexible than the full flow-abstraction that is inherent in the definition of the OpenFlow protocol, the MPLS use of FECs+LSPs qualifies as ‘flows’. But it is the absence of the map-abstraction in the IP/MPLS control plane that makes it complex. So much so that in most cases the only flow-definition (or FEC) actually used is the ‘destination- IP address’.

We discussed the benefits of the map-abstraction in the context of MPLS networks. The benefits include simplifying the IP/MPLS control-plane which in-turn reduces protocol load in router CPUs (a real concern in carrier networks where not all hardware is upgraded at the same time). Another benefit of the map-abstraction is that it makes the control-plane more extensible, by not only providing the services that MPLS provides today (TE/VPNs); but also by making it easier to change or introduce new functionality into the network – a fact that can lead to its use as a control plane for MPLS-TP and multi-layer networks. And finally, a key-benefit of the map-abstraction is that it presents a global-view to network-applications which can then globally optimize the network-function or service (as opposed to the local-optimizations possible today). Such global optimizations can help reduce churn (due to Auto-bandwidth) or help MPLS based recovery (FRR) function more efficiently.

We built a network-prototype to verify our architectural constructs and validate our simplicity and extensibility claims. The data-plane switches used the standard MPLS data-plane mechanisms, and the control-plane was based on OpenFlow; on map-abstractions created by a network OS; and network-applications we implemented to replace the functionality of the distributed IP/MPLS control plane. We implemented nearly all the features of MPLS traffic-engineering including TE-LSP admission control, bandwidth reservations, tunnel-priorities, Auto-Route, Auto-Bandwidth, class-based TE, and load-sharing. And we did so in less than 5k lines-of-code, which is nearly two orders

of magnitude fewer than implementations using a traditional distributed approach (thereby validating our simplicity claim). Furthermore, we showed that implementing TE features with our design choices can either greatly simplify the feature or provide a capability that the current MPLS-TE control plane cannot provide (thereby validating our extensibility claim). Finally, we discussed how introducing the map-abstraction in MPLS networks fits well with the discussion in the previous chapters on packet-circuit unified network control, in that treating packets, tunnels and circuits as flows, enables a network-operator to benefit from a simple consistent view of their network.