Source-Directed Routing

Another of IPv6's extension headers is the routing header. In the absence of this header, routing for IPv6 differs very little from that of IPv4 under CIDR. The routing header contains a list of IPv6 addresses that represent nodes or topological areas that the packet should visit en route to its destination. A topological area may be, for example, a backbone provider's network. Specifying that packets must visit this network would be a way of implementing provider selection on a packet-by-packet basis. Thus, a host could say that it wants some packets to go through a provider that is cheap, others through a provider that provides high reliability, and still others through a provider that the host trusts to provide security.

To provide the ability to specify topological entities rather than individual nodes, IPv6 defines an *anycast* address. An anycast address is assigned to a set of interfaces, and packets sent to that address will go to the "nearest" of those interfaces, with nearest being determined by the routing protocols. For example, all the routers of a backbone provider could be assigned a single anycast address, which would be used in the routing header.

4.3 Multicast

Multi-access networks like Ethernet implement multicast in hardware. There are, however, applications that need a broader multicasting capability that is effective at the scale of the Internet. For example, when a radio station is broadcast over the Internet, the same data must be sent to all the hosts where a user has tuned in to that station. In that example, the communication is one-to-many. Other examples of one-to-many applications include transmitting the same news, current stock prices, software updates, or TV channels to multiple hosts. The latter example is commonly called IPTV.

There are also applications whose communication is many-to-many, such as multimedia teleconferencing, online multiplayer gaming, or distributed simulations. In such cases, members of a group receive data from multiple senders, typically each other. From any particular sender, they all receive the same data.

Normal IP communication, in which each packet must be addressed and sent to a single host, is not well suited to such applications. If an application has data to send to a group, it would have to send a separate packet with the identical data to each member of the group. This redundancy consumes more bandwidth than necessary. Furthermore, the redundant traffic is not distributed evenly but rather is focused around the sending host, and may easily exceed the capacity of the sending host and the nearby networks and routers.

To better support many-to-many and one-to-many communication, IP provides an IP-level multicast analogous to the link-level multicast provided by multi-access networks like Ethernet. Now that we are introducing the concept of multicast for IP, we also need a term for the traditional one-to-one service of IP that has been described so far: That service is referred to as *unicast*.

The basic IP multicast model is a many-to-many model based on multicast *groups*, where each group has its own IP *multicast address*. The hosts that are members of a group receive copies of any packets sent to that group's multicast address. A host can be in multiple groups, and it can join and leave groups freely by telling its local router using a protocol that we will discuss shortly. Thus, while we think of unicast addresses as being associated with a node or an interface, multicast addresses are associated with an abstract group, the membership of which changes dynamically over time. Further, the original IP multicast service model allows *any* host to send multicast traffic to a group; it doesn't have to be a member of the group, and there may be any number of such senders to a given group.

Using IP multicast to send the identical packet to each member of the group, a host sends a single copy of the packet addressed to the group's multicast address. The sending host doesn't need to know the individual unicast IP address of each member of the group because, as we will see, that knowledge is distributed among the routers in the internetwork. Similarly, the sending host doesn't need to send multiple copies of the packet because the routers will make copies whenever they have to forward the packet over more than one link. Compared to using unicast IP to deliver the same packets to many receivers, IP multicast is more scalable because it eliminates the redundant traffic (packets) that would have been sent many times over the same links, especially those near to the sending host.

IP's original many-to-many multicast has been supplemented with support for a form of one-to-many multicast. In this model of one-to-many multicast, called *Source-Specific Multicast* (SSM), a receiving host specifies both a multicast group and a specific sending host. The receiving host would then receive multicasts addressed to the specified group, but only if they are from the specified sender. Many Internet multicast applications (e.g., radio broadcasts) fit the SSM model. To contrast it with SSM, IP's original many-to-many model is sometimes referred to as *Any Source Multicast* (ASM).

A host signals its desire to join or leave a multicast group by communicating with its local router using a special protocol for just that purpose. In IPv4, that protocol is the *Internet Group Management Protocol* (IGMP); in IPv6, it is *Multicast Listener Discovery* (MLD). The router then has the responsibility for making multicast behave correctly with regard to that host. Because a host may fail to leave a multicast group when it should (after a crash or other failure, for example), the router periodically polls the network to determine which groups are still of interest to the attached hosts.

4.3.1 Multicast Addresses

IP has a subrange of its address space reserved for multicast addresses. In IPv4, these addresses are assigned in the class D address space, and IPv6 also has a portion of its address space reserved for multicast group addresses. Some subranges of the multicast ranges are reserved for intradomain multicast, so they can be reused independently by different domains.

There are thus 28 bits of possible multicast address in IPv4 when we ignore the prefix shared by all multicast addresses. This presents a problem when attempting to take advantage of hardware multicasting on a local area network (LAN). Let's take the case of Ethernet. Ethernet multicast addresses have only 23 bits when we ignore their shared prefix. In other words, to take advantage of Ethernet multicasting, IP has to map 28-bit IP multicast addresses into 23-bit Ethernet multicast addresses. This is implemented by taking the low-order 23 bits of any IP multicast address to use as its Ethernet multicast address and ignoring the high-order 5 bits. Thus, 32 (2^5) IP addresses map into each one of the Ethernet addresses.

> In this section we use Ethernet as a canonical example of a networking technology that supports multicast in hardware, but the same is also true of PON (Passive Optical Networks), which is the access network technology often used to deliver fiber-to-the-home. In fact, IP Multicast over PON is now a common way to deliver IPTV to homes.

When a host on an Ethernet joins an IP multicast group, it configures its Ethernet interface to receive any packets with the corresponding Ethernet multicast address. Unfortunately, this causes the receiving host to receive not only the multicast traffic it desired but also traffic sent to any of the other 31 IP multicast groups that map to the same Ethernet address, if they are routed to that Ethernet. Therefore, IP at the receiving host must examine the IP header of any multicast packet to determine whether the packet really belongs to the desired group. In summary, the mismatch of multicast address sizes means that multicast traffic may

place a burden on hosts that are not even interested in the group to which the traffic was sent. Fortunately, in some switched networks (such as switched Ethernet) this problem can be mitigated by schemes wherein the switches recognize unwanted packets and discard them.

One perplexing question is how senders and receivers learn which multicast addresses to use in the first place. This is normally handled by out-of-band means, and there are some quite sophisticated tools to enable group addresses to be advertised on the Internet.

4.3.2 Multicast Routing (DVMRP, PIM, MSDP)

A router's unicast forwarding tables indicate, for any IP address, which link to use to forward the unicast packet. To support multicast, a router must additionally have multicast forwarding tables that indicate, based on multicast address, which links—possibly more than one—to use to forward the multicast packet (the router duplicates the packet if it is to be forwarded over multiple links). Thus, where unicast forwarding tables collectively specify a set of paths, multicast forwarding tables collectively specify a set of trees: *multicast distribution trees*. Furthermore, to support Source-Specific Multicast (and, it turns out, for some types of Any Source Multicast), the multicast forwarding tables must indicate which links to use based on the combination of multicast address and the (unicast) IP address of the source, again specifying a set of trees.

Multicast routing is the process by which the multicast distribution trees are determined or, more concretely, the process by which the multicast forwarding tables are built. As with unicast routing, it is not enough that a multicast routing protocol "work"; it must also scale reasonably well as the network grows, and it must accommodate the autonomy of different routing domains.

DVMRP

Distance-vector routing used in unicast can be extended to support multicast. The resulting protocol is called *Distance Vector Multicast Routing Protocol*, or DVMRP. DVMRP was the first multicast routing protocol to see widespread use.

Recall that, in the distance-vector algorithm, each router maintains a table of `Destination, Cost, NextHop` tuples, and exchanges a list of `(Destination, Cost)` pairs with its directly connected neighbors. Extending this algorithm to support multicast is a two-stage process. First, we create a broadcast mechanism that allows a packet to be forwarded to all the networks on the internet. Second, we need to refine this mechanism so that it prunes back networks that do not have hosts that belong to the multicast group. Consequently, DVMRP is one of several multicast routing protocols described as *flood-and-prune* protocols.

Given a unicast routing table, each router knows that the current shortest path to a given `destination` goes through `NextHop`. Thus, whenever it receives a multicast packet from source S, the router forwards the packet on all outgoing links (except the one on which the packet arrived) if and only if the packet arrived over the link that is on the shortest path to S (i.e., the packet came *from* the `NextHop` associated with S in the routing table). This strategy effectively floods packets outward from S but does not loop packets back toward S.

There are two major shortcomings to this approach. The first is that it truly floods the network; it has no provision for avoiding LANs that have no members in the multicast group. We address this problem below. The second limitation is that a given packet will be forwarded over a LAN by each of the routers connected

to that LAN. This is due to the forwarding strategy of flooding packets on all links other than the one on which the packet arrived, without regard to whether or not those links are part of the shortest-path tree rooted at the source.

The solution to this second limitation is to eliminate the duplicate broadcast packets that are generated when more than one router is connected to a given LAN. One way to do this is to designate one router as the *parent* router for each link, relative to the source, where only the parent router is allowed to forward multicast packets from that source over the LAN. The router that has the shortest path to source S is selected as the parent; a tie between two routers would be broken according to which router has the smallest address. A given router can learn if it is the parent for the LAN (again relative to each possible source) based upon the distance-vector messages it exchanges with its neighbors.

Notice that this refinement requires that each router keep, for each source, a bit for each of its incident links indicating whether or not it is the parent for that source/link pair. Keep in mind that in an internet setting, a source is a network, not a host, since an internet router is only interested in forwarding packets between networks. The resulting mechanism is sometimes called *Reverse Path Broadcast* (RPB) or *Reverse Path Forwarding* (RPF). The path is reverse because we are considering the shortest path toward the *source* when making our forwarding decisions, as compared to unicast routing, which looks for the shortest path to a given *destination*.

The RPB mechanism just described implements shortest-path broadcast. We now want to prune the set of networks that receives each packet addressed to group G to exclude those that have no hosts that are members of G. This can be accomplished in two stages. First, we need to recognize when a *leaf* network has no group members. Determining that a network is a leaf is easy—if the parent router as described above is the only router on the network, then the network is a leaf. Determining if any group members reside on the network is accomplished by having each host that is a member of group G periodically announce this fact over the network, as described in our earlier description of link-state multicast. The router then uses this information to decide whether or not to forward a multicast packet addressed to G over this LAN.

The second stage is to propagate this "no members of G here" information up the shortest-path tree. This is done by having the router augment the `(Destination, Cost)` pairs it sends to its neighbors with the set of groups for which the leaf network is interested in receiving multicast packets. This information can then be propagated from router to router, so that for each of its links a given router knows for what groups it should forward multicast packets.

Note that including all of this information in the routing update is a fairly expensive thing to do. In practice, therefore, this information is exchanged only when some source starts sending packets to that group. In other words, the strategy is to use RPB, which adds a small amount of overhead to the basic distance-vector algorithm, until a particular multicast address becomes active. At that time, routers that are not interested in receiving packets addressed to that group speak up, and that information is propagated to the other routers.

PIM-SM

Protocol Independent Multicast, or PIM, was developed in response to the scaling problems of earlier multicast routing protocols. In particular, it was recognized that the existing protocols did not scale well in environments where a relatively small proportion of routers want to receive traffic for a certain group. For example, broadcasting traffic to all routers until they explicitly ask to be removed from the distribution is not a good design choice if most routers don't want to receive the traffic in the first place. This situation is sufficiently common that PIM divides the problem space into *sparse mode* and *dense mode,* where sparse

and dense refer to the proportion of routers that will want the multicast. PIM dense mode (PIM-DM) uses a flood-and-prune algorithm like DVMRP and suffers from the same scalability problem. PIM sparse mode (PIM-SM) has become the dominant multicast routing protocol and is the focus of our discussion here. The "protocol independent" aspect of PIM, by the way, refers to the fact that, unlike earlier protocols such as DVMRP, PIM does not depend on any particular sort of unicast routing—it can be used with any unicast routing protocol, as we will see below.

In PIM-SM, routers explicitly join the multicast distribution tree using PIM protocol messages known as `Join` messages. Note the contrast to DVMRP's approach of creating a broadcast tree first and then pruning the uninterested routers. The question that arises is where to send those `Join` messages because, after all, any host (and any number of hosts) could send to the multicast group. To address this, PIM-SM assigns to each group a special router known as the *rendezvous point* (RP). In general, a number of routers in a domain are configured to be candidate RPs, and PIM-SM defines a set of procedures by which all the routers in a domain can agree on the router to use as the RP for a given group. These procedures are rather complex, as they must deal with a wide variety of scenarios, such as the failure of a candidate RP and the partitioning of a domain into two separate networks due to a number of link or node failures. For the rest of this discussion, we assume that all routers in a domain know the unicast IP address of the RP for a given group.

A multicast forwarding tree is built as a result of routers sending `Join` messages to the RP. PIM-SM allows two types of trees to be constructed: a *shared* tree, which may be used by all senders, and a *source-specific* tree, which may be used only by a specific sending host. The normal mode of operation creates the shared tree first, followed by one or more source-specific trees if there is enough traffic to warrant it. Because building trees installs state in the routers along the tree, it is important that the default is to have only one tree for a group, not one for every sender to a group.

When a router sends a `Join` message toward the RP for a group G, it is sent using normal IP unicast transmission. This is illustrated in Figure 4.14(a), in which router R4 is sending a `Join` to the rendezvous point for some group. The initial `Join` message is "wildcarded"; that is, it applies to all senders. A `Join` message clearly must pass through some sequence of routers before reaching the RP (e.g., R2). Each router along the path looks at the `Join` and creates a forwarding table entry for the shared tree, called a (*, G) entry (where * means "all senders"). To create the forwarding table entry, it looks at the interface on which the `Join` arrived and marks that interface as one on which it should forward data packets for this group. It then determines which interface it will use to forward the `Join` toward the RP. This will be the only acceptable interface for incoming packets sent to this group. It then forwards the `Join` toward the RP. Eventually, the message arrives at the RP, completing the construction of the tree branch. The shared tree thus constructed is shown as a solid line from the RP to R4 in Figure 4.14(a).

As more routers send `Join`s toward the RP, they cause new branches to be added to the tree, as illustrated in Figure 4.14(b). Note that, in this case, the `Join` only needs to travel to R2, which can add the new branch to the tree simply by adding a new outgoing interface to the forwarding table entry created for this group. R2 need not forward the `Join` on to the RP. Note also that the end result of this process is to build a tree whose root is the RP.

At this point, suppose a host wishes to send a message to the group. To do so, it constructs a packet with the appropriate multicast group address as its destination and sends it to a router on its local network known as the *designated router* (DR). Suppose the DR is R1 in Figure 4.14. There is no state for this multicast group between R1 and the RP at this point, so instead of simply forwarding the multicast packet, R1 *tunnels* it to the RP. That is, R1 encapsulates the multicast packet inside a PIM `Register` message that it sends to the unicast IP address of the RP. Just like an IP tunnel endpoint, the RP receives the packet addressed to it, looks at the payload of the `Register` message, and finds inside an IP packet addressed to the multicast address

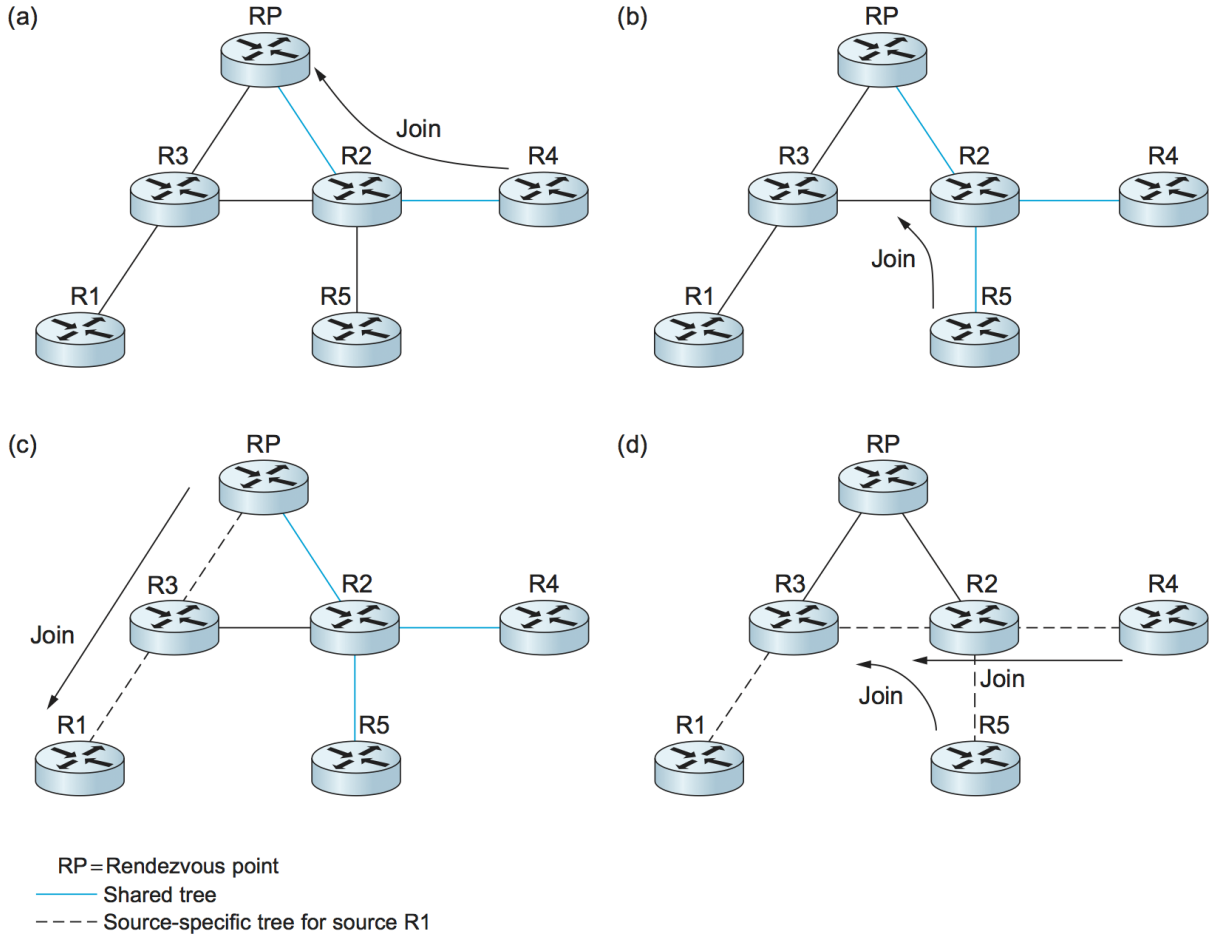

Figure 4.14.: PIM operation: (a) R4 sends a Join message to RP and joins shared tree; (b) R5 joins shared tree; (c) RP builds source-specific tree to R1 by sending a Join message to R1; (d) R4 and R5 build source-specific tree to R1 by sending Join messages to R1.

of this group. The RP, of course, does know what to do with such a packet—it sends it out onto the shared tree of which the RP is the root. In the example of Figure 4.14, this means that the RP sends the packet on to R2, which is able to forward it on to R4 and R5. The complete delivery of a packet from R1 to R4 and R5 is shown in Figure 4.15. We see the tunneled packet travel from R1 to the RP with an extra IP header containing the unicast address of RP, and then the multicast packet addressed to G making its way along the shared tree to R4 and R5.

At this point, we might be tempted to declare success, since all hosts can send to all receivers this way. However, there is some bandwidth inefficiency and processing cost in the encapsulation and decapsulation of packets on the way to the RP, so the RP forces knowledge about this group into the intervening routers so tunneling can be avoided. It sends a `Join` message toward the sending host (Figure 4.14(c)). As this `Join` travels toward the host, it causes the routers along the path (R3) to learn about the group, so that it will be possible for the DR to send the packet to the group as *native* (i.e., not tunneled) multicast packets.

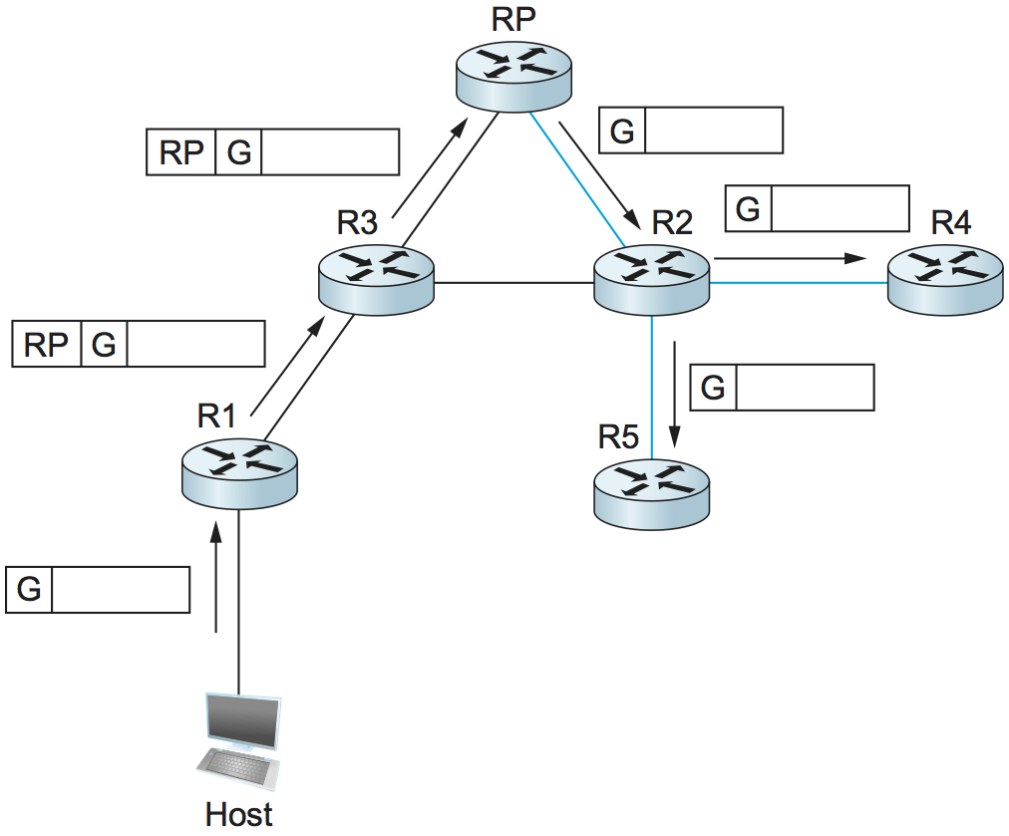

Figure 4.15.: Delivery of a packet along a shared tree. R1 tunnels the packet to the RP, which forwards it along the shared tree to R4 and R5.

An important detail to note at this stage is that the `Join` message sent by the RP to the sending host is specific to that sender, whereas the previous ones sent by R4 and R5 applied to all senders. Thus, the effect of the new `Join` is to create *sender-specific* state in the routers between the identified source and the RP. This is referred to as (S, G) state, since it applies to one sender to one group, and contrasts with the (*, G) state that was installed between the receivers and the RP that applies to all senders. Thus, in Figure 4.14(c), we see a source-specific route from R1 to the RP (indicated by the dashed line) and a tree that is valid for all senders from the RP to the receivers (indicated by the solid line).

The next possible optimization is to replace the entire shared tree with a source-specific tree. This is desirable because the path from sender to receiver via the RP might be significantly longer than the shortest possible path. This again is likely to be triggered by a high data rate being observed from some sender. In this case, the router at the downstream end of the tree—say, R4 in our example—sends a source-specific `Join` toward the source. As it follows the shortest path toward the source, the routers along the way create (S, G) state for this tree, and the result is a tree that has its root at the source, rather than the RP. Assuming both R4 and R5 made the switch to the source-specific tree, we would end up with the tree shown in Figure 4.14(d). Note that this tree no longer involves the RP at all. We have removed the shared tree from this picture to simplify the diagram, but in reality all routers with receivers for a group must stay on the shared tree in case new senders show up.

We can now see why PIM is protocol independent. All of its mechanisms for building and maintaining trees take advantage of unicast routing without depending on any particular unicast routing protocol. The formation of trees is entirely determined by the paths that `Join` messages follow, which is determined by the choice of shortest paths made by unicast routing. Thus, to be precise, PIM is "unicast routing protocol independent," as compared to DVMRP. Note that PIM is very much bound up with the Internet Protocol—it is not protocol independent in terms of network-layer protocols.

The design of PIM-SM again illustrates the challenges in building scalable networks and how scalability is sometimes pitted against some sort of optimality. The shared tree is certainly more scalable than a source-specific tree, in the sense that it reduces the total state in routers to be on the order of the number of groups rather than the number of senders times the number of groups. However, the source-specific tree is likely to be necessary to achieve efficient routing and effective use of link bandwidth.

Interdomain Multicast (MSDP)

PIM-SM has some significant shortcomings when it comes to interdomain multicast. In particular, the existence of a single RP for a group goes against the principle that domains are autonomous. For a given multicast group, all the participating domains would be dependent on the domain where the RP is located. Furthermore, if there is a particular multicast group for which a sender and some receivers shared a single domain, the multicast traffic would still have to be routed initially from the sender to those receivers via whatever domain has the RP for that multicast group. Consequently, the PIM-SM protocol is typically not used across domains, only within a domain.

To extend multicast across domains using PIM-SM, the Multicast Source Discovery Protocol (MSDP) was devised. MSDP is used to connect different domains—each running PIM-SM internally, with its own RPs—by connecting the RPs of the different domains. Each RP has one or more MSDP peer RPs in other domains. Each pair of MSDP peers is connected by a TCP connection over which the MSDP protocol runs. Together, all the MSDP peers for a given multicast group form a loose mesh that is used as a broadcast network. MSDP messages are broadcast through the mesh of peer RPs using the Reverse Path Broadcast algorithm that we discussed in the context of DVMRP.

What information does MSDP broadcast through the mesh of RPs? Not group membership information; when a host joins a group, the furthest that information will flow is its own domain's RP. Instead, it is source—multicast sender—information. Each RP knows the sources in its own domain because it receives a `Register` message whenever a new source arises. Each RP periodically uses MSDP to broadcast `Source Active` messages to its peers, giving the IP address of the source, the multicast group address, and the IP address of the originating RP.

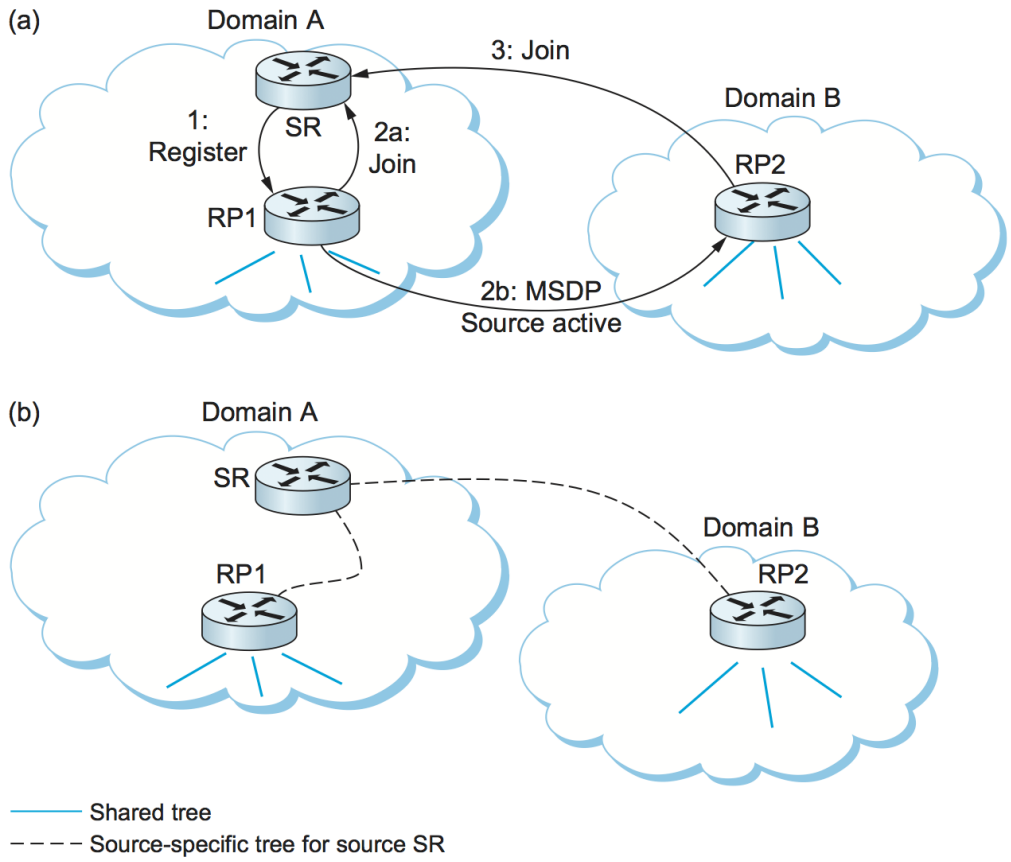

Figure 4.16.: MSDP operation: (a) The source SR sends a Register message to its domain's RP, RP1; then RP1 sends a source-specific Join message to SR and an MSDP Source Active message to its MSDP peer in Domain B, RP2; then RP2 sends a source-specific Join message to SR. (b) As a result, RP1 and RP2 are in the source-specific tree for source SR.

If an MSDP peer RP that receives one of these broadcasts has active receivers for that multicast group, it sends a source-specific `Join`, on that RP's own behalf, to the source host, as shown in Figure 4.16(a). The `Join` message builds a branch of the source-specific tree to this RP, as shown in Figure 4.16(b). The result is that every RP that is part of the MSDP network and has active receivers for a particular multicast group is added to the source-specific tree of the new source. When an RP receives a multicast from the source, the RP uses its shared tree to forward the multicast to the receivers in its domain.

Source-Specific Multicast (PIM-SSM)

The original service model of PIM was, like earlier multicast protocols, a many-to-many model. Receivers joined a group, and any host could send to the group. However, it was recognized in the late 1990s that it might be useful to add a one-to-many model. Lots of multicast applications, after all, have only one legitimate sender, such as the speaker at a conference being sent over the Internet. We already saw that PIM-SM can create source-specific shortest path trees as an optimization after using the shared tree initially. In the original PIM design, this optimization was invisible to hosts—only routers joined source-specific trees. However, once the need for a one-to-many service model was recognized, it was decided to make the source-specific routing capability of PIM-SM explicitly available to hosts. It turns out that this mainly required changes to IGMP and its IPv6 analog, MLD, rather than PIM itself. The newly exposed capability is now known as PIM-SSM (PIM Source-Specific Multicast).

PIM-SSM introduces a new concept, the *channel*, which is the combination of a source address S and a group address G. The group address G looks just like a normal IP multicast address, and both IPv4 and IPv6 have allocated subranges of the multicast address space for SSM. To use PIM-SSM, a host specifies both the group and the source in an IGMP Membership Report message to its local router. That router then sends a PIM-SM source-specific `Join` message toward the source, thereby adding a branch to itself in the source-specific tree, just as was described above for "normal" PIM-SM, but bypassing the whole shared-tree stage. Since the tree that results is source specific, only the designated source can send packets on that tree.

The introduction of PIM-SSM has provided some significant benefits, particularly since there is relatively high demand for one-to-many multicasting:

- Multicasts travel more directly to receivers.
- The address of a channel is effectively a multicast group address plus a source address. Therefore, given that a certain range of multicast group addresses will be used for SSM exclusively, multiple domains can use the same multicast group address independently and without conflict, as long as they use it only with sources in their own domains.
- Because only the specified source can send to an SSM group, there is less risk of attacks based on malicious hosts overwhelming the routers or receivers with bogus multicast traffic.
- PIM-SSM can be used across domains exactly as it is used within a domain, without reliance on anything like MSDP.

SSM, therefore, is quite a useful addition to the multicast service model.

Bidirectional Trees (BIDIR-PIM)

We round off our discussion of multicast with another enhancement to PIM known as *Bidirectional PIM*. BIDIR-PIM is a recent variant of PIM-SM that is well suited to many-to-many multicasting within a domain,

especially when senders and receivers to a group may be the same, as in a multiparty videoconference, for example. As in PIM-SM, would-be receivers join groups by sending IGMP Membership Report messages (which must not be source specific), and a shared tree rooted at an RP is used to forward multicast packets to receivers. Unlike PIM-SM, however, the shared tree also has branches to the *sources*. That wouldn't make any sense with PIM-SM's unidirectional tree, but BIDIR-PIM's trees are bidirectional—a router that receives a multicast packet from a downstream branch can forward it both up the tree and down other branches. The route followed to deliver a packet to any particular receiver goes only as far up the tree as necessary before going down the branch to that receiver. See the multicast route from R1 to R2 in Figure 4.17(b) for an example. R4 forwards a multicast packet downstream to R2 at the same time that it forwards a copy of the same packet upstream to R5.

A surprising aspect of BIDIR-PIM is that there need not actually be an RP. All that is needed is a routable address, which is known as an RP address even though it need not be the address of an RP or anything at all. How can this be? A `Join` from a receiver is forwarded toward the RP address until it reaches a router with an interface on the link where the RP address would reside, where the Join terminates. Figure 4.17(a) shows a `Join` from R2 terminating at R5, and a `Join` from R3 terminating at R6. The upstream forwarding of a multicast packet similarly flows toward the RP address until it reaches a router with an interface on the link where the RP address would reside, but then the router forwards the multicast packet onto that link as the final step of upstream forwarding, ensuring that all other routers on that link receive the packet. Figure 4.17(b) illustrates the flow of multicast traffic originating at R1.

BIDIR-PIM cannot thus far be used across domains. On the other hand, it has several advantages over PIM-SM for many-to-many multicast within a domain:

- There is no source registration process because the routers already know how to route a multicast packet toward the RP address.
- The routes are more direct than those that use PIM-SM's shared tree because they go only as far up the tree as necessary, not all the way to the RP.
- Bidirectional trees use much less state than the source-specific trees of PIM-SM because there is never any source-specific state. (On the other hand, the routes will be longer than those of source-specific trees.)
- The RP cannot be a bottleneck, and indeed no actual RP is needed.

One conclusion to draw from the fact that there are so many different approaches to multicast just within PIM is that multicast is a difficult problem space in which to find optimal solutions. You need to decide which criteria you want to optimize (bandwidth usage, router state, path length, etc.) and what sort of application you are trying to support (one-to-many, many-to-many, etc.) before you can make a choice of the "best" multicast mode for the task.

4.4 Multiprotocol Label Switching

We continue our discussion of enhancements to IP by describing an addition to the Internet architecture that is very widely used but largely hidden from end users. The enhancement, called *Multiprotocol Label Switching* (MPLS), combines some of the properties of virtual circuits with the flexibility and robustness of datagrams. On the one hand, MPLS is very much associated with the Internet Protocol's datagram-based architecture—it relies on IP addresses and IP routing protocols to do its job. On the other hand, MPLS-enabled routers also forward packets by examining relatively short, fixed-length labels, and these

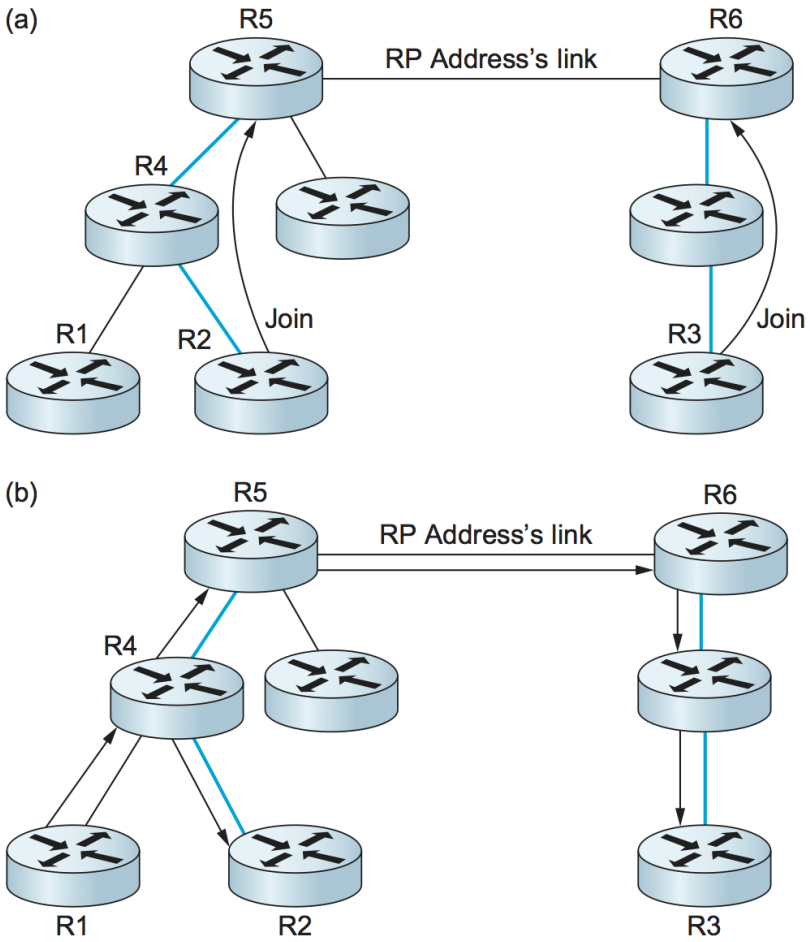

Figure 4.17.: BIDIR-PIM operation: (a) R2 and R3 send Join messages toward the RP address that terminate when they reach a router on the RP address's link. (b) A multicast packet from R1 is forwarded upstream to the RP address's link and downstream wherever it intersects a group member branch.

labels have local scope, just like in a virtual circuit network. It is perhaps this marriage of two seemingly opposed technologies that has caused MPLS to have a somewhat mixed reception in the Internet engineering community.

Before looking at how MPLS works, it is reasonable to ask "what is it good for?" Many claims have been made for MPLS, but there are three main things that it is used for today:

- To enable IP capabilities on devices that do not have the capability to forward IP datagrams in the normal manner
- To forward IP packets along explicit routes—precalculated routes that don't necessarily match those that normal IP routing protocols would select
- To support certain types of virtual private network services

It is worth noting that one of the original goals—improving performance—is not on the list. This has a lot to do with the advances that have been made in forwarding algorithms for IP routers in recent years and with the complex set of factors beyond header processing that determine performance.

The best way to understand how MPLS works is to look at some examples of its use. In the next three sections, we will look at examples to illustrate the three applications of MPLS mentioned above.

4.4.1 Destination-Based Forwarding

One of the earliest publications to introduce the idea of attaching labels to IP packets was a paper by Chandranmenon and Vargese that described an idea called *threaded indices*. A very similar idea is now implemented in MPLS-enabled routers. The following example shows how this idea works.

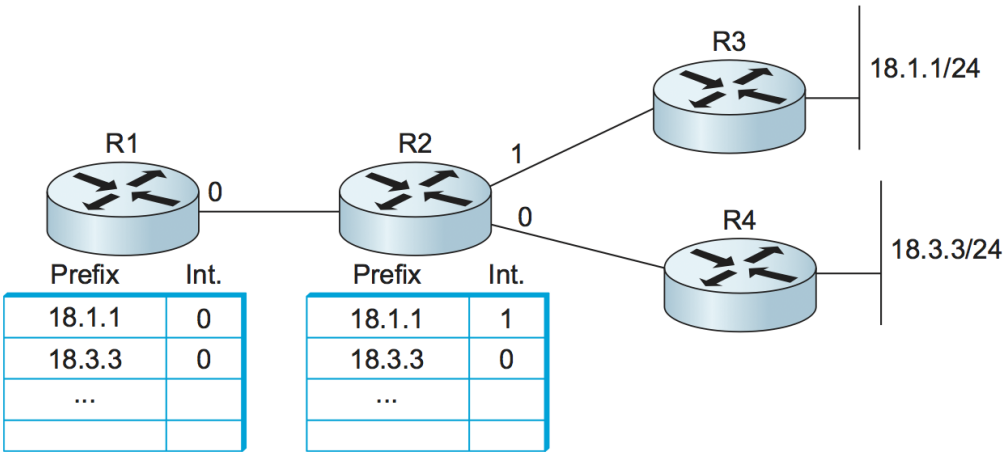

Figure 4.18.: Routing tables in example network.

Consider the network in Figure 4.18. Each of the two routers on the far right (R3 and R4) has one connected network, with prefixes `18.1.1/24` and `18.3.3/24`. The remaining routers (R1 and R2) have routing tables that indicate which outgoing interface each router would use when forwarding packets to one of those two networks.

When MPLS is enabled on a router, the router allocates a label for each prefix in its routing table and advertises both the label and the prefix that it represents to its neighboring routers. This advertisement is

carried in the Label Distribution Protocol. This is illustrated in Figure 4.19. Router R2 has allocated the label value 15 for the prefix 18.1.1 and the label value 16 for the prefix 18.3.3. These labels can be chosen at the convenience of the allocating router and can be thought of as indices into the routing table. After allocating the labels, R2 advertises the label bindings to its neighbors; in this case, we see R2 advertising a binding between the label 15 and the prefix 18.1.1 to R1. The meaning of such an advertisement is that R2 has said, in effect, "Please attach the label 15 to all packets sent to me that are destined to prefix 18.1.1." R1 stores the label in a table alongside the prefix that it represents as the remote or outgoing label for any packets that it sends to that prefix.

In Figure 4.19(c), we see another label advertisement from router R3 to R2 for the prefix 18.1.1, and R2 places the remote label that it learned from R3 in the appropriate place in its table.

At this point, we can look at what happens when a packet is forwarded in this network. Suppose a packet destined to the IP address 18.1.1.5 arrives from the left to router R1. R1 in this case is referred to as a *Label Edge Router* (LER); an LER performs a complete IP lookup on arriving IP packets and then applies labels to them as a result of the lookup. In this case, R1 would see that 18.1.1.5 matches the prefix 18.1.1 in its forwarding table and that this entry contains both an outgoing interface and a remote label value. R1 therefore attaches the remote label 15 to the packet before sending it.

When the packet arrives at R2, R2 looks only at the label in the packet, not the IP address. The forwarding table at R2 indicates that packets arriving with a label value of 15 should be sent out interface 1 and that they should carry the label value 24, as advertised by router R3. R2 therefore rewrites, or swaps, the label and forwards it on to R3.

What has been accomplished by all this application and swapping of labels? Observe that when R2 forwarded the packet in this example it never actually needed to examine the IP address. Instead, R2 looked only at the incoming label. Thus, we have replaced the normal IP destination address lookup with a label lookup. To understand why this is significant, it helps to recall that, although IP addresses are always the same length, IP prefixes are of variable length, and the IP destination address lookup algorithm needs to find the *longest match*—the longest prefix that matches the high order bits in the IP address of the packet being forwarded. By contrast, the label forwarding mechanism just described is an *exact match* algorithm. It is possible to implement a very simple exact match algorithm, for example, by using the label as an index into an array, where each element in the array is one line in the forwarding table.

Note that, while the forwarding algorithm has been changed from longest match to exact match, the routing algorithm can be any standard IP routing algorithm (e.g., OSPF). The path that a packet will follow in this environment is the exact same path that it would have followed if MPLS were not involved: the path chosen by the IP routing algorithms. All that has changed is the forwarding algorithm.

An important fundamental concept of MPLS is illustrated by this example. Every MPLS label is associated with a *forwarding equivalence class* (FEC)—a set of packets that are to receive the same forwarding treatment in a particular router. In this example, each prefix in the routing table is an FEC; that is, all packets that match the prefix 18.1.1—no matter what the low order bits of the IP address are—get forwarded along the same path. Thus, each router can allocate one label that maps to 18.1.1, and any packet that contains an IP address whose high order bits match that prefix can be forwarded using that label.

As we will see in the subsequent examples, FECs are a very powerful and flexible concept. FECs can be formed using almost any criteria; for example, all the packets corresponding to a particular customer could be considered to be in the same FEC.

Returning to the example at hand, we observe that changing the forwarding algorithm from normal IP forwarding to label swapping has an important consequence: Devices that previously didn't know how to

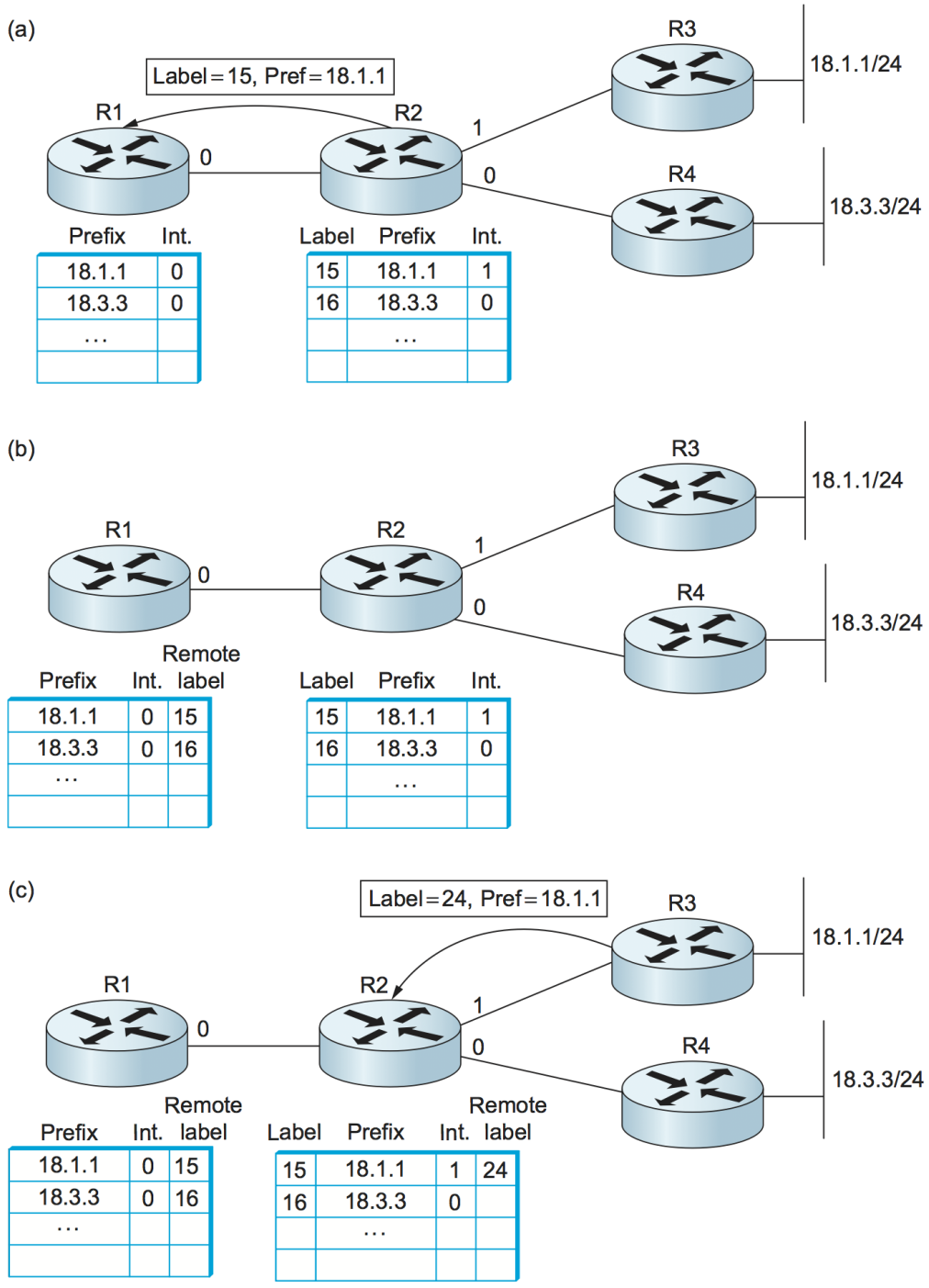

Figure 4.19.: (a) R2 allocates labels and advertises bindings to R1. (b) R1 stores the received labels in a table. (c) R3 advertises another binding, and R2 stores the received label in a table.

forward IP packets can be used to forward IP traffic in an MPLS network. The most notable early application of this result was to ATM switches, which can support MPLS without any changes to their forwarding hardware. ATM switches support the label-swapping forwarding algorithm just described, and by providing these switches with IP routing protocols and a method to distribute label bindings they could be turned into *Label Switching Routers* (LSRs)—devices that run IP control protocols but use the label switching forwarding algorithm. More recently, the same idea has been applied to optical switches.

Before we consider the purported benefits of turning an ATM switch into an LSR, we should tie up some loose ends. We have said that labels are "attached" to packets, but where exactly are they attached? The answer depends on the type of link on which packets are carried. Two common methods for carrying labels on packets are shown in Figure 4.20. When IP packets are carried as complete frames, as they are on most link types including Ethernet and PPP, the label is inserted as a "shim" between the layer 2 header and the IP (or other layer 3) header, as shown in the lower part of the figure. However, if an ATM switch is to function as an MPLS LSR, then the label needs to be in a place where the switch can use it, and that means it needs to be in the ATM cell header, exactly where one would normally find the virtual circuit identifier (VCI) and virtual path identifier (VPI) fields.

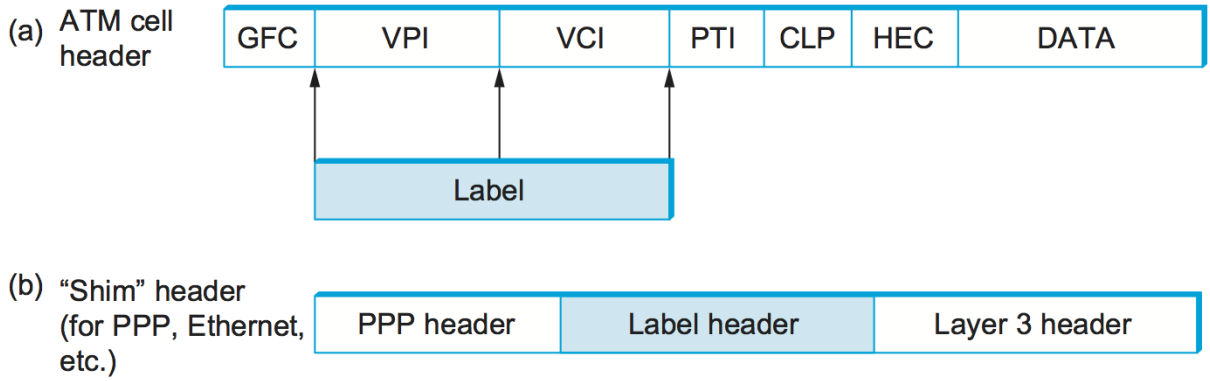

Figure 4.20.: (a) Label on an ATM-encapsulated packet; (b) label on a frame-encapsulated packet.

Having now devised a scheme by which an ATM switch can function as an LSR, what have we gained? One thing to note is that we could now build a network that uses a mixture of conventional IP routers, label edge routers, and ATM switches functioning as LSRs, and they would all use the same routing protocols. To understand the benefits of using the same protocols, consider the alternative. In Figure 4.21(a), we see a set of routers interconnected by virtual circuits over an ATM network, a configuration called an *overlay* network. At one point in time, networks of this type were often built because commercially available ATM switches supported higher total throughput than routers. Today, networks like this are less common because routers have caught up with and even surpassed ATM switches. However, these networks still exist because of the significant installed base of ATM switches in network backbones, which in turn is partly a result of ATM's ability to support a range of capabilities such as circuit emulation and virtual circuit services.

In an overlay network, each router would potentially be connected to each of the other routers by a virtual circuit, but in this case for clarity we have just shown the circuits from R1 to all of its peer routers. R1 has five routing neighbors and needs to exchange routing protocol messages with all of them—we say that R1 has five routing adjacencies. By contrast, in Figure 4.21(b), the ATM switches have been replaced with LSRs. There are no longer virtual circuits interconnecting the routers. Thus, R1 has only one adjacency, with LSR1. In large networks, running MPLS on the switches leads to a significant reduction in the number of adjacencies that each router must maintain and can greatly reduce the amount of work that the routers have to do to keep each other informed of topology changes.

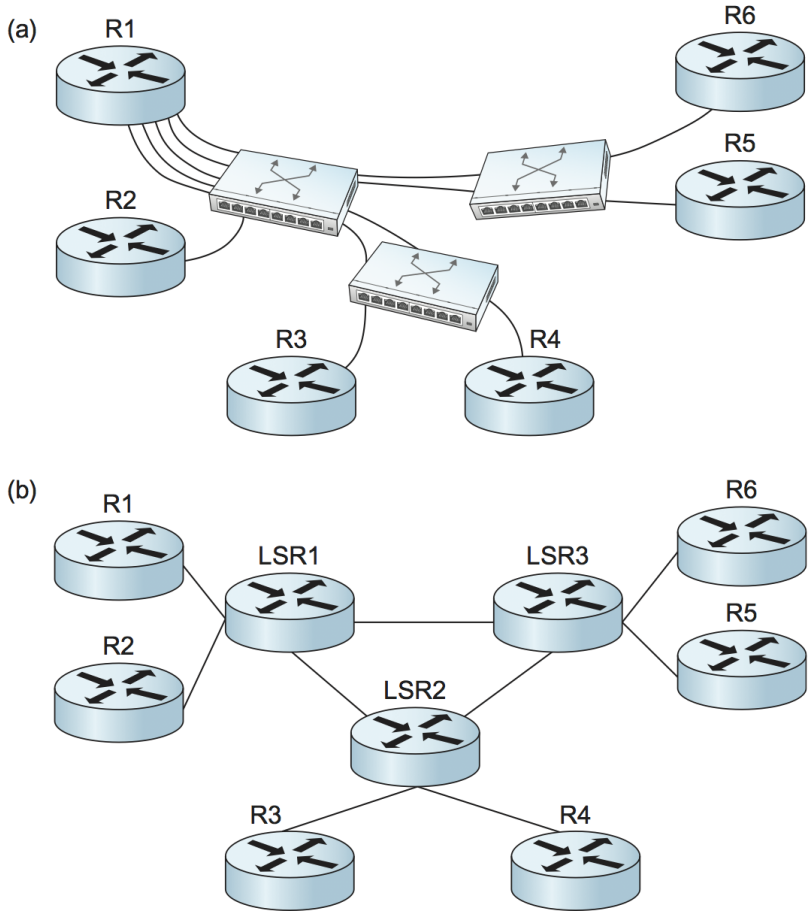

Figure 4.21.: (a) Routers connect to each other using an overlay of virtual circuits. (b) Routers peer directly with LSRs.

A second benefit of running the same routing protocols on edge routers and on the LSRs is that the edge routers now have a full view of the topology of the network. This means that if some link or node fails inside the network, the edge routers will have a better chance of picking a good new path than if the ATM switches rerouted the affected VCs without the knowledge of the edge routers.

Note that the step of "replacing" ATM switches with LSRs is actually achieved by changing the protocols running on the switches, but typically no change to the forwarding hardware is needed; that is, an ATM switch can often be converted to an MPLS LSR by upgrading only its software. Furthermore, an MPLS LSR might continue to support standard ATM capabilities at the same time as it runs the MPLS control protocols, in what is referred to as "ships in the night" mode.

The idea of running IP control protocols on devices that are unable to forward IP packets natively has been extended to Wavelength Division Multiplexing (WDM) and Time Division Multiplexing (TDM) networks (e.g., SONET). This is known as *Generalized MPLS* (GMPLS). Part of the motivation for GMPLS was to provide routers with topological knowledge of an optical network, just as in the ATM case. Even more important was the fact that there were no standard protocols for controlling optical devices, so MPLS proved to be a natural fit for that job.

4.4.2 Explicit Routing

IP has a source routing option, but it is not widely used for several reasons, including the fact that only a limited number of hops can be specified and because it is usual processed outside the "fast path" on most routers.

MPLS provides a convenient way to add capabilities similar to source-routing to IP networks, although the capability is more often referred to as *explicit routing* rather than *source routing*. One reason for the distinction is that it usually isn't the real source of the packet that picks the route. More often it is one of the routers inside a service provider's network. Figure 4.22 shows an example of how the explicit routing capability of MPLS might be applied. This sort of network is often called a *fish* network because of its shape (the routers R1 and R2 form the tail; R7 is at the head).

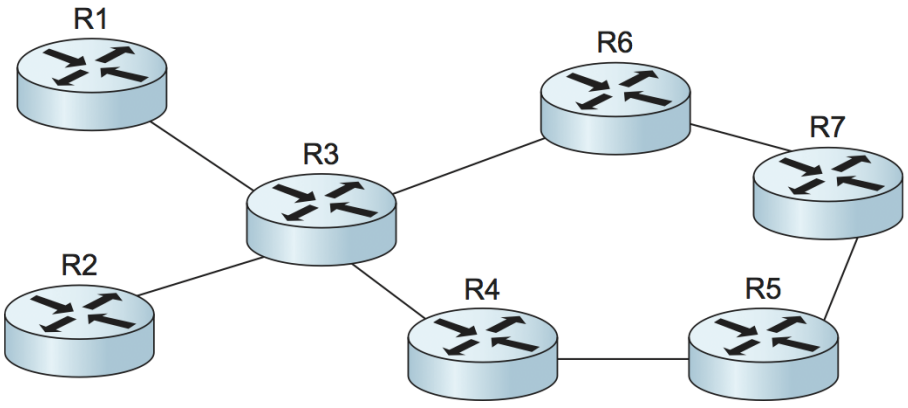

Figure 4.22.: A network requiring explicit routing.

Suppose that the operator of the network in Figure 4.22 has determined that any traffic flowing from R1 to R7 should follow the path R1-R3-R6-R7 and that any traffic going from R2 to R7 should follow the path R2-R3-R4-R5-R7. One reason for such a choice would be to make good use of the capacity available along the two distinct paths from R3 to R7. We can think of the R1-to-R7 traffic as constituting one forwarding

equivalence class, and the R2-to-R7 traffic constitutes a second FEC. Forwarding traffic in these two classes along different paths is difficult with normal IP routing, because R3 doesn't normally look at where traffic came from in making its forwarding decisions.

Because MPLS uses label swapping to forward packets, it is easy enough to achieve the desired routing if the routers are MPLS enabled. If R1 and R2 attach distinct labels to packets before sending them to R3—thus identifying them as being in different FECs—then R3 can forward packets from R1 and R2 along different paths. The question that then arises is how do all the routers in the network agree on what labels to use and how to forward packets with particular labels? Clearly, we can't use the same procedures as described in the preceding section to distribute labels, because those procedures establish labels that cause packets to follow the normal paths picked by IP routing, which is exactly what we are trying to avoid. Instead, a new mechanism is needed. It turns out that the protocol used for this task is the Resource Reservation Protocol (RSVP). For now it suffices to say that it is possible to send an RSVP message along an explicitly specified path (e.g., R1-R3-R6-R7) and use it to set up label forwarding table entries all along that path. This is very similar to the process of establishing a virtual circuit.

One of the applications of explicit routing is *traffic engineering*, which refers to the task of ensuring that sufficient resources are available in a network to meet the demands placed on it. Controlling exactly which paths the traffic flows on is an important part of traffic engineering. Explicit routing can also help to make networks more resilient in the face of failure, using a capability called *fast reroute*. For example, it is possible to precalculate a path from router A to router B that explicitly avoids a certain link L. In the event that link L fails, router A could send all traffic destined to B down the precalculated path. The combination of precalculation of the backup path and the explicit routing of packets along the path means that A doesn't need to wait for routing protocol packets to make their way across the network or for routing algorithms to be executed by various other nodes in the network. In certain circumstances, this can significantly reduce the time taken to reroute packets around a point of failure.

One final point to note about explicit routing is that explicit routes need not be calculated by a network operator as in the above example. Routers can use various algorithms to calculate explicit routes automatically. The most common of these is *constrained shortest path first* (CSPF), which is a link-state algorithm, but which also takes various *constraints* into account. For example, if it was required to find a path from R1 to R7 that could carry an offered load of 100 Mbps, we could say that the constraint is that each link must have at least 100 Mbps of available capacity. CSPF addresses this sort of problem.

4.4.3 Virtual Private Networks and Tunnels

One way to build virtual private networks (VPNs) is to use tunnels. It turns out that MPLS can be thought of as a way to build tunnels, and this makes it suitable for building VPNs of various types.

The simplest form of MPLS VPN to understand is a layer 2 VPN. In this type of VPN, MPLS is used to tunnel layer 2 data (such as Ethernet frames or ATM cells) across a network of MPLS-enabled routers. One reason for tunnels is to provide some sort of network service (such as multicast) that is not supported by some routers in the network. The same logic applies here: IP routers are not ATM switches, so you cannot provide an ATM virtual circuit service across a network of conventional routers. However, if you had a pair of routers interconnected by a tunnel, they could send ATM cells across the tunnel and emulate an ATM circuit. The term for this technique within the IETF is *pseudowire emulation*. Figure 4.23 illustrates the idea.

We have already seen how IP tunnels are built: The router at the entrance of the tunnel wraps the data to

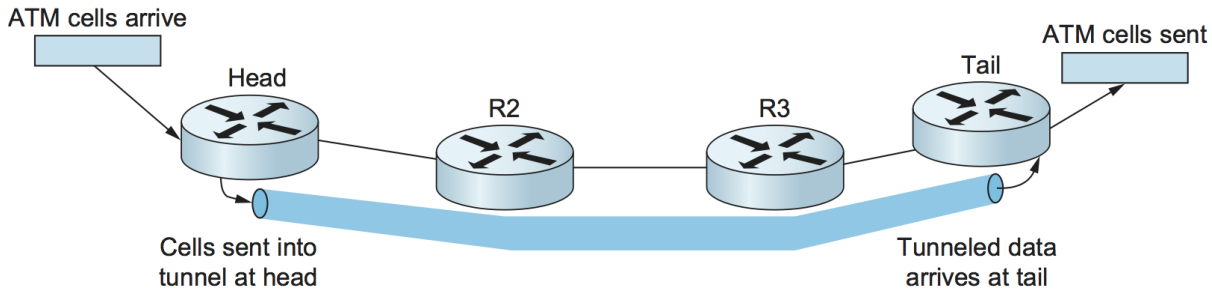

Figure 4.23.: An ATM circuit is emulated by a tunnel.

be tunneled in an IP header (the *tunnel header*), which represents the address of the router at the far end of the tunnel and sends the data like any other IP packet. The receiving router receives the packet with its own address in the header, strips the tunnel header, and finds the data that was tunneled, which it then processes. Exactly what it does with that data depends on what it is. For example, if it were another IP packet, it would then be forwarded on like a normal IP packet. However, it need not be an IP packet, as long as the receiving router knows what to do with non-IP packets. We'll return to the issue of how to handle non-IP data in a moment.

An MPLS tunnel is not too different from an IP tunnel, except that the tunnel header consists of an MPLS header rather than an IP header. Looking back to our first example, in Figure 4.19, we saw that router R1 attached a label (`15`) to every packet that it sent towards prefix 18.1.1. Such a packet would then follow the path R1-R2-R3, with each router in the path examining only the MPLS label. Thus, we observe that there was no requirement that R1 only send IP packets along this path—any data could be wrapped up in the MPLS header and it would follow the same path, because the intervening routers never look beyond the MPLS header. In this regard, an MPLS header is just like an IP tunnel header (except only 4 bytes long instead of 20 bytes). The only issue with sending non-IP traffic along a tunnel, MPLS or otherwise, is what to do with non-IP traffic when it reaches the end of the tunnel. The general solution is to carry some sort of demultiplexing identifier in the tunnel payload that tells the router at the end of the tunnel what to do. It turns out that an MPLS label is a perfect fit for such an identifier. An example will make this clear.

Let's assume we want to tunnel ATM cells from one router to another across a network of MPLS-enabled routers, as in Figure 4.23. Further, we assume that the goal is to emulate an ATM virtual circuit; that is, cells arrive at the entrance, or head, of the tunnel on a certain input port with a certain VCI and should leave the tail end of the tunnel on a certain output port and potentially different VCI. This can be accomplished by configuring the head and tail routers as follows:

- The head router needs to be configured with the incoming port, the incoming VCI, the demultiplexing label for this emulated circuit, and the address of the tunnel end router.
- The tail router needs to be configured with the outgoing port, the outgoing VCI, and the demultiplexing label.

Once the routers are provided with this information, we can see how an ATM cell would be forwarded. Figure 4.24 illustrates the steps.

1. An ATM cell arrives on the designated input port with the appropriate VCI value (101 in this example).
2. The head router attaches the demultiplexing label that identifies the emulated circuit.
3. The head router then attaches a second label, which is the tunnel label that will get the packet to the

tail router. This label is learned by mechanisms just like those described elsewhere in this section.

4. Routers between the head and tail forward the packet using only the tunnel label.
5. The tail router removes the tunnel label, finds the demultiplexing label, and recognizes the emulated circuit.
6. The tail router modifies the ATM VCI to the correct value (202 in this case) and sends it out the correct port.

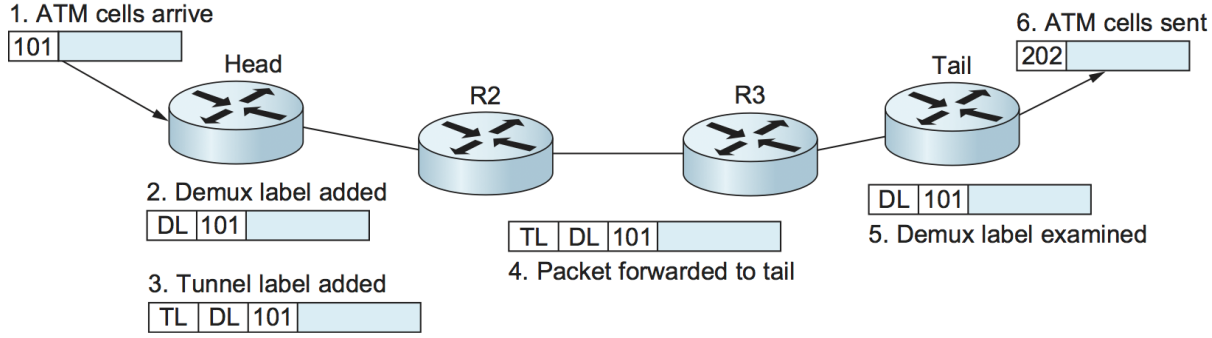

Figure 4.24.: Forward ATM cells along a tunnel.

One item in this example that might be surprising is that the packet has two labels attached to it. This is one of the interesting features of MPLS—labels may be stacked on a packet to any depth. This provides some useful scaling capabilities. In this example, it allows a single tunnel to carry a potentially large number of emulated circuits.

The same techniques described here can be applied to emulate many other layer 2 services, including Frame Relay and Ethernet. It is worth noting that virtually identical capabilities can be provided using IP tunnels; the main advantage of MPLS here is the shorter tunnel header.

Before MPLS was used to tunnel layer 2 services, it was also being used to support layer 3 VPNs. We won't go into the details of layer 3 VPNs, which are quite complex, but we will note that they represent one of the most popular uses of MPLS today. Layer 3 VPNs also use stacks of MPLS labels to tunnel packets across an IP network. However, the packets that are tunneled are themselves IP packets—hence, the name *layer 3* VPNs. In a layer 3 VPN, a single service provider operates a network of MPLS-enabled routers and provides a "virtually private" IP network service to any number of distinct customers. That is, each customer of the provider has some number of sites, and the service provider creates the illusion for each customer that there are no other customers on the network. The customer sees an IP network interconnecting his own sites and no other sites. This means that each customer is isolated from all other customers in terms of both routing and addressing. Customer A can't sent packets directly to customer B, and *vice versa*. Customer A can even use IP addresses that have also been used by customer B. The basic idea is illustrated in Figure 4.25. As in layer 2 VPNs, MPLS is used to tunnel packets from one site to another; however, the configuration of the tunnels is performed automatically by some fairly elaborate use of BGP, which is beyond the scope of this book.

Customer A in fact usually *can* send data to customer B in some restricted way. Most likely, both customer A and customer B have some connection to the global Internet, and thus it is probably possible for customer A to send email messages, for example, to the mail server inside customer B's network. The "privacy" offered by a VPN prevents customer A from having unrestricted access to all the machines and subnets inside customer B's network.

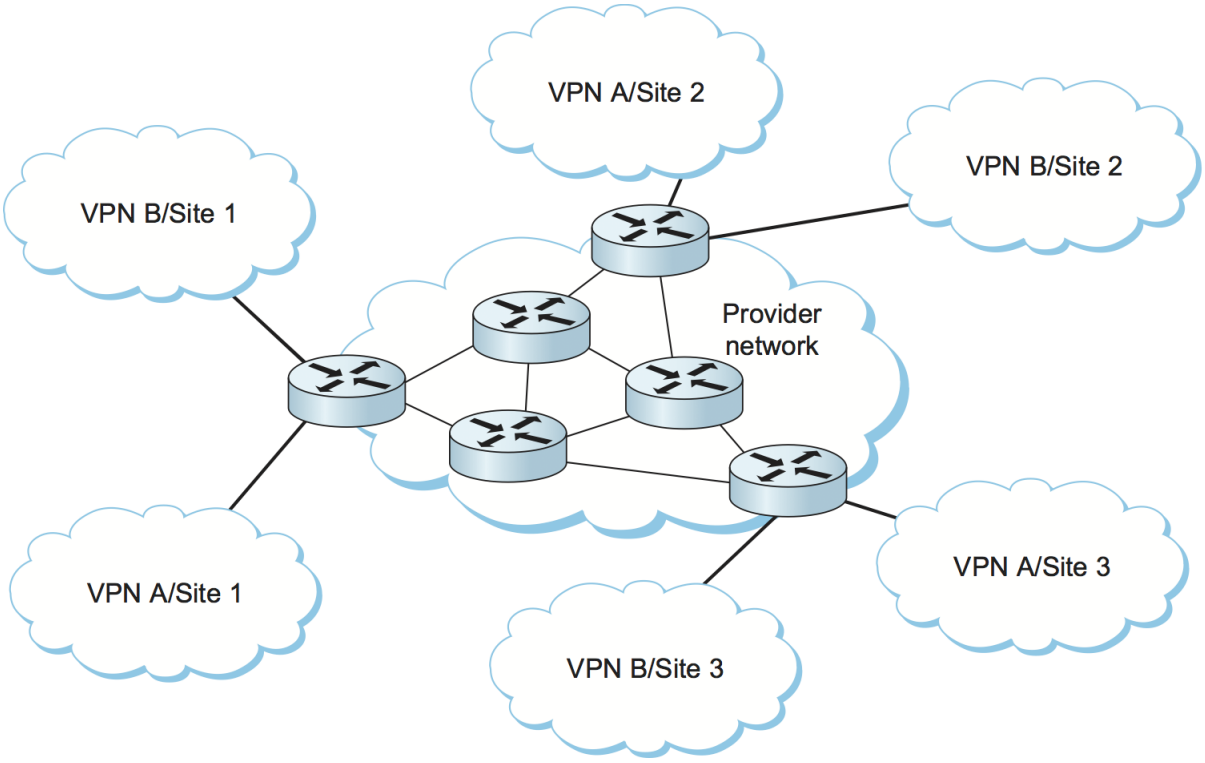

Figure 4.25.: Example of a layer 3 VPN. Customers A and B each obtain a virtually private IP service from a single provider.

In summary, MPLS is a rather versatile tool that has been applied to a wide range of different networking problems. It combines the label-swapping forwarding mechanism that is normally associated with virtual circuit networks with the routing and control protocols of IP datagram networks to produce a class of network that is somewhere between the two conventional extremes. This extends the capabilities of IP networks to enable, among other things, more precise control of routing and the support of a range of VPN services.

4.5 Routing Among Mobile Devices

It probably should not be a great surprise to learn that mobile devices present some challenges for the Internet architecture. The Internet was designed in an era when computers were large, immobile devices, and, while the Internet's designers probably had some notion that mobile devices might appear in the future, it's fair to assume it was not a top priority to accommodate them. Today, of course, mobile computers are everywhere, notably in the form of laptops and smartphones, and increasingly in other forms, such as drones. In this section, we will look at some of the challenges posed by the appearance of mobile devices and some of the current approaches to accommodating them.

4.5.1 Challenges for Mobile Networking

It is easy enough today to turn up in a wireless hotspot, connect to the Internet using 802.11 or some other wireless networking protocol, and obtain pretty good Internet service. One key enabling technology that