3

The Network Management Problem

Having looked at some of the nuts and bolts of network management technology, we now consider some of the problems of managing large networks. In many respects the large enterprise networks of today are reminiscent of the islands of automation that were common in manufacturing during the 1980s and 1990s. The challenge facing manufacturers was in linking together the islands of microprocessor-based controllers, PCs, minicomputers, and other components to allow end-to-end actions such as aggregated order entries leading to automated production runs. The hope was that the islands of automation could be joined so that the previously isolated intelligence could be leveraged to manufacture better products. Similar problems beset network operators at the beginning of the 21st century as traffic types and volumes continue to grow. In parallel with this, the range of deployed NMS is also growing. Multiple NMS adds to operational expense.

There is a strong need to reduce the cost of ownership and improve the return on investment (ROI) for network equipment. This is true not just during periods of economic downturn, but has become the norm as SLAs are applied to both enterprise and SP networks. NMS technology provides the network operator with some increasingly useful capabilities. One of these is a move away from tedious, error-prone, manually intensive operations to software-assisted, automated end-to-end operations.

Network operators must be able to execute automated end-to-end management operations on their networks [Telcordia]. An example of this is VLAN management in which an NMS GUI provides a visual picture—such as a cloud—of VLAN members (ports, MAC addresses, VLAN IDs). The NMS

can also provide the ability to easily add, delete, and modify VLAN members as well as indicate any faults (e.g., link failures, warm starts) as and when they occur. Another example is enterprise WAN management in which ATM or FR virtual circuits are used to carry the traffic from branch offices into central sites. In this case, the enterprise network manager wants to be able to easily create, delete, modify, and view any faults on the virtual circuits (and the underlying nodes, links, and interfaces) to the remote sites. Other examples include storage (including SANs) management and video/audio conferencing equipment management. As we saw in Chapter 1, "Large Enterprise Networks," the range of enterprise network services is growing all the time and so also is the associated management overhead.

The benefit of this type of end-to-end capability is a large reduction in the cost of managing enterprise networks by SLA fulfillment, less need for arcane NE know-how, smooth enterprise business processes, and happy end users. Open, vendor-independent NMS are needed for this, and later we look at ways in which software layering helps in designing and building such systems. Simple ideas such as always using default MIB values (seen in Chapter 1), pragmatic database design (matching default database and MIB values) and technology-sensitive menus also play an important part in providing NMS vendor-independence. The issue of presenting menu options appropriate to a given selected NE provides abstraction; for example, if the user wants to add a given NE interface to an IEEE 802.1Q VLAN, then (in order for the operation to be meaningful) that device must support this frame-tagging technology. The NMS should be able to figure this out and present the option *only* if the underlying hardware supports it. By presenting only appropriate options (rather than all possible options), the NMS reduces the amount of data the user must sift through to actually execute network management actions.

Automated, flow-through actions are required for as many network management operations as possible, including the following FCAPS areas:

- Provisioning
- Detecting faults
- Checking (and verifying) performance
- Billing/accounting
- Initiating repairs or network upgrades
- Maintaining the network inventory

Provisioning is a general term that relates to configuring network-resident objects, such as VLANs, VPNs, and virtual connections. It resolves down to the

act of modifying agent MIB object instances, that is, SNMP `setRequests`. Provisioning usually involves both `sets` and `gets`. Later in this chapter we see this when we want to add a new entry to the MPLS tunnel table. We must read the instance value of the object `mplsTunnelIndexNext` before sending a `setRequest` to actually create the tunnel. Many NMS do not permit provisioning for a variety of reasons:

- Provisioning code is hard to implement because of the issue of timeouts (i.e., when many `set` messages are sent, one or more may time out).
- NE security settings are required to prevent unauthorized actions.
- There is a lack of support for transactions that span multiple SNMP sets (i.e., SNMP does not provide rollback, a mechanism for use when failure occurs in one of a related sequence of SNMP sets. The burden of providing lengthy transactions and/or rollback is on the NMS).
- Provisioning actions can alter network dynamics (i.e., pushing a lot of sets into the network adds traffic and may also affect the performance of the local agents).

If the NMS does not allow provisioning, then some other means must be found; usually, this is the EMS/CLI. SNMPv3 provides adequate security for NMS provisioning operations.

Fault detection is a crucial element of network management. NMS fault detection is most effective when it provides an end-to-end view; for example, if a VLAN link to the backbone network is broken (as in VLAN 2 in Chapter 1, Figure 1–4), then that VLAN GUI element (e.g., a network cloud) should change color instantly. The NMS user should then be able to drill down via the GUI to determine the exact nature of the problem. The NMS should give an indication of the problem as well as a possible resolution (as we've seen, this is often called root-cause analysis). The NMS should also cater to the case where the user is not looking at the NMS topology and should provide some other means of announcing the problem, for instance, by email, mobile phone short text message, or pager.

Performance management is increasingly important to enterprises that use service level agreements (SLAs). These are contractual specifications between IT and the enterprise users for service uptime, downtime, bandwidth/system/network availability, and so on.

Billing is important for those services that directly cost the enterprise money, such as the PSTN. It is important for appropriate billing to be generated for such services. Billing may even be applied to incoming calls because they con-

sume enterprise network resources. Other elements of billing include departmental charges for remote logins to the network (external SP connections may be needed, for example, for remote-access VPN service) and other uses of the network, such as conference bridges. An important element of billing is verifying that network resources, such as congested PSTN/WAN trunks, are dimensioned correctly. In Chapter 1, we mentioned that branch offices are sometimes charged a flat rate for centralized corporate services (e.g., voice, LAN/WAN support). This is accounting rather than billing. In billing, money tends to be paid to some external organization, whereas in accounting, money may be merely transferred from one part of an organization to another. Many service providers offer services that are billed using a flat-rate model—for example, *x* dollars per month for an ATM link with bandwidth of y Mbps. Usage-based billing is increasingly attractive to customers because it allows for a pay-for-use or pay-as-you-grow model. It is likely that usage-based billing/accounting will increasingly be needed in enterprise NMS applications. This is particularly true as SLAs are adopted in enterprises.

Networks are dynamic entities, and repairs and upgrades are a constant concern for most enterprises. Any NE can become faulty, and switch/router interfaces can become congested. Repairs and upgrades need to be carried out and recorded, and the NMS is an effective means of achieving this.

All of the FCAPS applications combine to preserve and maintain the network inventory. An important aspect of any NMS is that the FCAPS applications are often inextricably interwoven; for example, a fault may be due to a specific link becoming congested, and this in turn may affect the performance of part of the network. We look at the important area of mediation in Chapter 6, "Network Management Software Components."

It is usually difficult to efficiently create NMS FCAPS applications without a base of high-quality EMS facilities. This base takes the form of a well-implemented SNMP agent software with the standard MIB and (if necessary) well-designed private MIB extensions. Private MIB extensions are needed for cases where vendors have added additional features that differentiate their NEs from the competition.

All these sophisticated NMS features come at a price: NMS software is expensive and is often priced on a per-node basis, increasing the network cost base. Clearly, the bigger the network, the bigger the NMS price tag (however, the ratio of cost/bit may go down).

This chapter focuses on the following major issues and their proposed solutions:

- Bringing the managed data to the code
- Scalability
- The shortage of development skills for creating management systems
- The shortage of operational skills for running networks

Bringing the Managed Data to the Code

Bringing data and code together is a fundamental computing concept. It is central to the area of network management, and current trends in NE development bring it to center stage. Loading a locally hosted text file into an editor like Microsoft Notepad is a simple example: The editor is the code and the text file is the data. In this case, the code and data reside on the same machine, and bringing them together is a trivial task. Getting SNMP agent data to the manager code is not a trivial task in the distributed data model of network management because:

- Managed objects reside on many SNMP agent hosts.
- Copies of managed objects reside on SNMP management systems.
- Changes in agent data may have to be regularly reconciled with the management system copy.

Agent-hosted managed objects change in tandem with the dynamics of the host machine and the underlying network—for example, the `ipInReceives` object from Chapter 1, which changes value every time an IP packet is received. This and many other managed objects change value constantly, providing a means for modeling the underlying system and its place in the network. The same is true of all managed NEs. MIBs provide a foundation for the management data model. The management system must keep track of relevant object value changes and apply new changes as and when they are required. As mentioned in Chapter 1, the management system keeps track of the NEs by a combination of polling, issuing `set` messages, and listening for notifications. This is a classic problem of storing the same data in two different places and is illustrated in Figure 3–1, where a management system tracks the objects in a managed network using the SNMP messages we saw in Chapter 2, "SNMPv3 and Network Management."

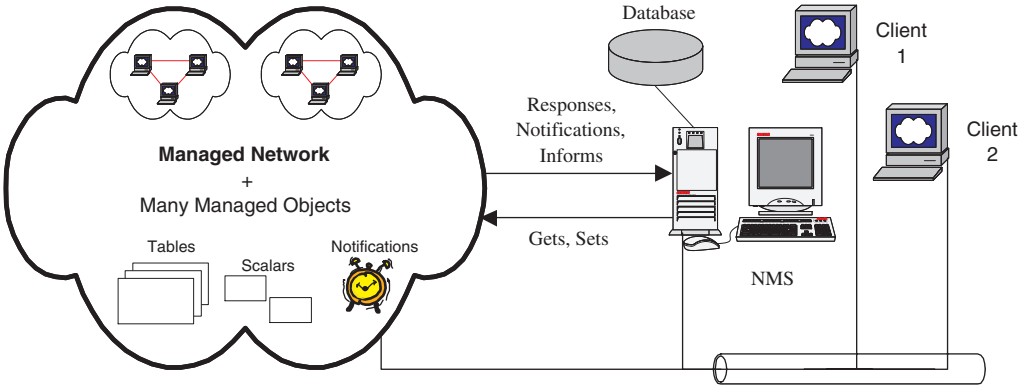


Figure 3–1 Components of an NMS.

Figure 3–1 illustrates a managed network, a central NMS server, a relational database, and several client users. The clients access the FCAPS services exported by the NMS, for example, viewing faults, provisioning, and security configuration. The NMS strives to keep up with changes in the NEs and to reflect these in the clients.

Even though SNMP agents form a major part of the management system infrastructure, they are physically remote from the management system. Agent data is created and maintained in a computational execution space removed from that of the management system. For example, the `ipInReceives` object is mapped into the tables maintained by the host TCP/IP protocol suite, and from there it gets its value.[1] Therefore, `get` or `set` messages sent from a manager to an agent result in computation on the agent host. The manager merely collects the results of the agent response. The manager-agent interaction can be seen as a loose type of message-based remote procedure call (RPC). The merit of not using a true RPC mechanism is the lack of associated overhead.

This is at once the strength and the weakness of SNMP. The important point is that the problem of getting the agent data to the manager is always present, particularly as networks grow in size and complexity. (This problem is not restricted to SNMP. Web site authors have a similar problem when they want to embed Java or JavaScript in their pages. The Java code must be downloaded

1. This is true for hosts; for routers, the ipInReceives object is part of the interface statistics and not part of the IP stack.

along with the HTML in an effort to marry the browser with the Web site code and data. Interestingly, in network management the process is reversed: The data is brought to the code.) So, should the management system simply request all of the agent data? This is possibly acceptable on small networks but not on heavily loaded, mission-critical enterprise and SP networks. For this reason, the management system struggles to maintain an accurate picture of the ever-changing network. This is a key network management concept.

If an ATM network operator prefers not to use signaled virtual circuits, then an extra monitoring burden is placed on the NMS. This is so because unsignaled connections do not recover from intermediate link or node failures. Such failures give rise to a race between the operator fixing the problem and the user noticing a service loss. These considerations lead us to an important principle concerning NMS technology: **The quality of an NMS is inversely proportional to the gap between its picture of the network and the actual state of the underlying network—the smaller the gap, the better the NMS**. An ideal NMS responds to network changes instantaneously. Real systems will always experience delays in updating themselves, and it is the goal of the designers and developers to minimize them.

As managed NEs become more complex, an extra burden is placed on the management system. The scale of this burden is explored in the next section.

Scalability: Today's Network Is Tomorrow's NE

Scalability is one of the biggest problems facing modern networking (security is another). It affects devices [RouterScale2002], interfaces, links, internal soft objects (such as virtual circuits), and management systems. **A scalability problem occurs when an increase in the number of instances of a given managed object in the network necessitates a compensating, proportional resource increase inside the management system.**

Layer 2 VPN Scalability

Scalability problems tend to arise in situations of proportional growth. Figure 3–2 illustrates a layer 2 VPN, which provides a good example of scalability. This scheme is often referred to as an overlay network because the IP network is overlaid on the underlying ATM infrastructure.

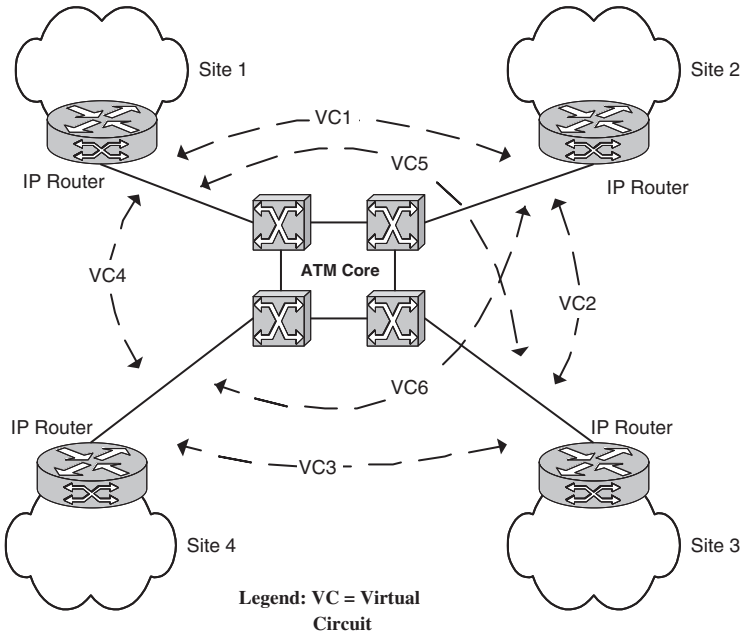


Figure 3–2 Layer 2 VPN scalability: the N^2 problem.

Sites 1 to 4 are all part of the one enterprise. This makes Figure 3–2 what is often called an intranet VPN. If one or more of the other sites is part of another organization, such as a customer or supplier, then we have an extranet VPN. Yet another VPN variant is the access VPN, which allows remote users to connect to it over some type of access technology, such as dialup.

In Figure 3–2, four sites are contained in the VPN, with one IP router in each site cloud. In order to achieve full layer 3 connectivity, each site must have a virtual circuit connection to every other site. These connections are created through the ATM core. So, the number of ATM virtual circuits required is six; that is, $N * (N - 1)/2$, where N is the number of sites. The full mesh of six bidirectional virtual circuits is shown in Figure 3–2 as VC1-VC6. A full mesh provides the necessary connectivity for the VPN [PrinRussell]. This is generally referred to as the N^2 problem because the number of layer 2 virtual circuits required is proportional to the square of the number of sites. Anything in networking that grows at the rate of N^2 tends to give rise to a problem of scale. The reason for calling this the N^2 problem is because as the number of sites gets bigger, the N^2 term is more significant than the other terms.

The problem gets worse if the ATM virtual circuits in the core are unidirectional (some vendors support only unidirectional permanent virtual circuits, or PVCs) in nature because then the number must be doubled in order for IP traffic to flow in both directions. Adding a new site to the VPN requires the creation of new virtual circuits from that site to all other sites. When the number of sites and subscribers is very large, the number of virtual circuits required tends to become unmanageable. Another less obvious problem with this is that each virtual circuit consumes switch capacity in terms of memory and control processor resources. Added to this is link bandwidth and fabric switching capacity if the virtual circuits reserve QoS resources.

As if that wasn't enough, a further problem with layer 2 VPNs is that topology changes in the core can result in routing information exchanges of the order of N^4 [DavieRehkter2000].

In contrast, layer 3 VPNs provide a much more scalable solution because the number of connections required is proportional to number of sites, not the square of the number of sites. Layer 3 VPNs (such as RFC 2547) avoid the need for a full mesh between all of the customer edge routers by providing features such as:

- A layer 3 core
- Overlapping IP address ranges across the connected sites (if separate organizations use the same VPN service)
- Multiple routing table instances in the provider edge routers

Not surprisingly, layer 3 VPN technology is an area of great interest to both enterprise network managers and service providers. For enterprises, layer 3 VPNs provide advanced, potentially low-cost networking features while allowing the service to be provided and managed by a service provider. For SP networks, layer 3 VPNs provide a scalable solution as well as an opportunity to extend services all the way to the customer premises.

Virtual Circuit Status Monitoring

Another example of scalability concerns the way in which an NMS manages status changes in ATM virtual circuits. If it is required for the NMS to read and interpret a MIB object table row for every ATM circuit, and this is done in an unrestricted fashion (where the NMS attempts to read all table entries at the same time), then there is scope for scalability problems. They arise when the number of MIB table entries becomes very large. In attempting to keep up with

large tables on possibly hundreds of agents throughout a big network, the NMS can run low on resources such as processing power, memory, and disk, and can effectively grind to a halt (unrestricted SNMP `gets` also use up network bandwidth). The large number of managed objects gives rise to NMS resource issues—that is, scalability. Really large SNMP tables can possibly also tax the resources of the host agent, resulting in timeouts and lost messages. The discussion here assumes this not to be the case.

Some method has to be devised for reading only those table entries that have changed rather than all entries. Processing large (ever-increasing) tables is not scalable. Agents may be able to assist in this by using some form of compressed data in `GetResponse` and `Trap` messages. This would require:

- A new type of MIB object.
- Compression software facilities in the agents and managers. To a degree, this could be considered to run counter to the philosophy of simplicity associated with SNMP.

A similar problem can occur in reverse when a manager wants to add entries to a large MIB table. In many cases (for example, when creating MPLS LSPs), it is necessary for the SNMP manager to specify an integer index for the new row. The index is a unique (in the sense of a relational database key) column in the table, and the next free value is used for the new LSP. So, if there are 10,000 LSPs already stored sequentially starting at 1, then the next available index is 10,001. The agent knows what the next free value is because it maintains the table. However, the manager may not necessarily know the extent of the table and often has to resort to an expensive MIB walk to determine the next free index. This is because agent data can be changed without the management system knowing about it; for example, using a command-line (or craft) interface, a user can independently add, delete, or modify NE data. This is mapped into the agent MIB, and if these operations do not generate traps, then the management system is oblivious to the data changes (unless it has an automatic discovery capability that reads the affected tables). There is a better way of navigating tables to cater to this and other situations. This brings us to our first MIB note.

MIB Note: Scalability

A very useful object type for large table management (as described above) is a counter conceptually called `nextObjectIndex`. This object provides the index number of the next available slot in a table. The agent maintains the value of this object so that when the manager has to add a new row to the MPLS tunnel table, it need only get the value of the associated `nextObjectIndex`. This avoids the overhead of MIB walks to actually count the entries and figure out the next free value. Once a new entry is added to the table, the agent increments the value of its `nextObjectIndex`. It is encouraging to see this type of MIB object being used in the IETF draft-standard MPLS MIBS, (e.g., `mplsTunnelIndexNext` [IETF-TE-MPLS]). This takes cognizance of scalability issues in the standard MIB document and avoids the need for proprietary solutions. It also provides a good example for implementers. Scalability issues like this can be difficult (or impossible) to resolve without the support of special MIB objects. This type of scalability issue will become more pressing as networks and the complexity of the constituent managed objects continue to grow.

Network operators and their users increasingly demand more bandwidth, faster networks, and bigger devices. Scalability concerns are growing because routers and switches are routinely expected to support the creation of millions of virtual circuits [ATM&IP2001]. Not only can devices support this number of objects, they can also create these circuits at an increasingly fast pace—tens of thousands per second. To illustrate the scale of this, let's assume in Figure 3–3 that there are hundreds of thousands of nodes (we show just a few).

Client 2 now executes a bulk provisioning operation. This results in the NMS server requesting that MPLS router LER A is to create two blocks of 10,000 signaled LSPs originating at A. The first 10,000 LSPs follow the path LER A-LSR A-LSR B-LSR C-LER B, while the second set follows the path LER A-LSR A-Cloud-LSR C-LER B. (The cloud in the latter case could be another network.) Further, let's assume that LER A can create LSPs at a rate of 10,000 per second. This means that once the intermediate node MIBs have been populated and the LSPs become operational, the network will emit a tunnel-up trap for every LSP. So, the management system has to be able to handle 20,000 traps coming in very fast from the network. There could be scope here for aggregating traps in compressed form, as mentioned earlier.

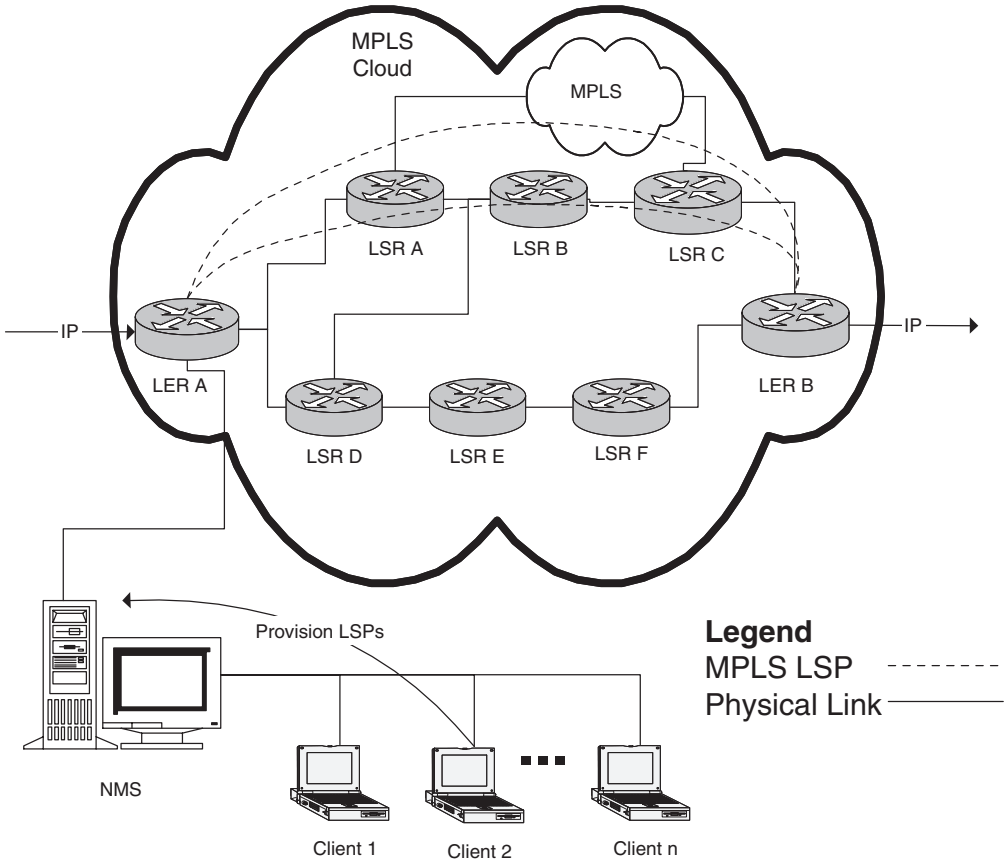


Figure 3–3 Creating LSPs in an MPLS network.

Since the LSPs are now operational, this must be reflected in the management system database and the active client/user interfaces (Clients 1 to *n* in Figure 3–3). The clients could be viewing (or provisioning, like Client 2) LSPs in the network, and any required changes to their views should be made as quickly as possible.

The problems don't stop there, because the LSPs must then be managed for further changes, such as:

- Status (e.g., becoming congested or going out of service)
- Faults such as an intermediate node/link failure or receipt of an invalid MPLS label
- Deletion by a user via a CLI (i.e., outside the management system)

- Modification by a user (changing the administrative status from up to down)

The result of any or all of these is some change in the LSP managed object attributes. The NMS picture of the network state is then at variance with the actual picture. All such changes must be reflected in the NMS as quickly as possible. The detailed functions of a typical NMS are discussed in Chapter 5, "A Real NMS."

The above discussion is a little simplistic; that is, it is likely that many of the above LSPs might be aggregated into one LSP with more reserved bandwidth. However, we illustrate the case merely to point out that if the emerging NEs are capable of generating large numbers of virtual circuits quickly, then the NMS must be able to support that in all of the affected FCAPS areas.

A noteworthy point in Figure 3–3 is the direction of the IP service; this is indicated as being from left to right. This reflects the fact that MPLS is a forwarding technology. If it is required to move IP traffic from LER B towards LER A, then LSPs have to be created specifically for this purpose, originating at LER B and terminating at LER A.

Other Enterprise Network Scalability Issues

The discussion in the previous section applies mostly to SP networks. Scalability concerns are also profoundly affecting enterprise networks in the following areas:

- Storage solutions, such as adding, deleting, modifying, and monitoring SANs
- Administration of firewalls, such as rules for permitting or blocking packet transit
- Routers, such as access control lists and static routes
- Security management, such as encryption keys, biometrics facilities, and password control
- Application management

SANs are becoming a vital storage service. Storage needs are steadily increasing as the number and complexity of applications in use grows. The administration burden associated with firewalls, routers, security, and applications deployment is growing all the time as user populations expand and work practices become more automated.

Light Reading Trials

Internet core routers from Cisco, Juniper, Charlotte's Networks, and Foundry Networks were stress-tested during 2001 [LightReading]. A range of tests were carried out, including:

- MPLS throughput
- Latency
- IP throughput at OC-48
- IP throughput at OC-192

During the MPLS throughput testing, the Juniper router supported the creation and use of up to 10,000 LSPs, while the Cisco router topped out at 5,000. Service providers expect to be able to build networks using devices capable of creating millions of LSPs. Clearly, at the time these tests were run, the equipment vendors involved had a long way to go.

In late November 2002, Light Reading executed trials against multiservice switches. One of the tests concerned a sequential (i.e., one after the other) LSP (specifically, tunnel) setup in which the vendor switch (from Alcatel) acted as an LSR and later as an egress LER. The switch was subjected to a steady stream of LSP-creation request messages using RSVP-TE. The test stopped as soon as the switch rejected a setup message. The test imposed 8,000 simultaneous RSVP-TE LSPs for core switches (i.e., LSRs) and 15,000 simultaneous RSVP-TE tunnels for edge switches (i.e., LERs). The Alcatel switch passed the test.

Large NEs

The trend is towards the deployment of much bigger devices, and as with any engineering proposition, this has advantages and disadvantages. The advantages of bigger (denser) routers and switches are:

- They reduce the number of devices required, saving central office (CO) space and reducing cooling and power requirements.
- They may help to reduce cabling by aggregating links.
- They offer a richer feature set.

Reduced inventory is important because less maintenance is needed, and a richer feature set provides for greater control and functional diversity. (Minimizing inventory also reduces operational and capital expenditure and helps retain cash. It also reduces the risk of holding obsolescent stock.) The disadvantages of such devices are:

- They are harder to manage.
- They potentially generate vast amounts of management data.
- They are a possible single point of failure if not backed up.

Compressing so much functionality into devices makes them harder to manage. Correlating faults with services and their users becomes problematic because of the sheer weight of connections. Also, the management system must support more interaction in all of the FCAPS functional areas. This adds up to more I/O and computation.

A related problem is that of SNMP agents timing out during periods of heavy traffic. The SNMP software process on a given NE may have been given a lower priority than the other service implementation modules, with the result that management operations may not be processed quickly enough. In other words, heavily loaded devices may temporarily starve the onboard management software. It is precisely at the time of heavy loading that the management function is most needed in order to control the network, and this may not be possible because of:

- Process priority clashes
- SNMP message queue sizes that are too small
- Excessive I/O interrupts

Management system availability is increasingly important on large networks, and it is also more at risk precisely because of the growing dimensions of the network. Management system capabilities have to be extended beyond what is currently available. The skills required to create these systems seem to be in short supply in the industry. This shortage of skills is examined in the next section.

Expensive (and Scarce) Development Skill Sets

Building management systems for the devices of today and tomorrow is increasingly difficult. (The same is true for device development where new technologies such as MPLS and Gigabit Ethernet are being added to new and legacy layer 2 NEs.) Some vendor organizations have completely separate groups devoted to NE and management system development. This introduces a need for clear communication between the groups. Aside from this, the skill set required of NMS software developers is growing and includes, in some cases, what have traditionally been separate disciplines:

- Object-oriented development and modeling using Unified Modeling Language (UML) for capturing requirements, defining actors (system users)

and use cases (the principal transactions and features), and mapping them into software classes
- Java/C++
- GUI, often packaged as part of a browser and providing access to network diagrams, provisioning facilities, faults, accounting, and so on
- Server software for long-running, multiclient FCAPS processes
- Specific support for mature/developing features, such as ATM/MPLS
- CORBA for multiple programming languages and remote object support across heterogeneous environments
- Database design/upgrade—matching MIB to database schema across numerous NMS/NE software releases[2]
- Deployment and installation issues—performance is always an important deployment issue, as is ease of installation
- IP routing
- MPLS
- Layer 2 technologies such as ATM, FR, and Gigabit Ethernet
- Legacy technologies such as voice-over-TDM and X.25
- Ability to develop generic software components and models—the management system can hide much of the complex underlying detail of running the network
- Client/server design
- Managed object design, part of the modeling phase for the management system
- MIB design—often there is a need for new objects in the managed devices to support the management system

This is an impressive set of skills for even the most experienced engineer. An excellent overall knowledge of these areas is needed along with an ability to focus on any one of them. The general migration to a layer 3 infrastructure is another reason for the widening gap between available development skills and required product features. Natural attrition, promotions, and new entrants to the industry ensure that there is a steady supply of engineers who are fairly *unlikely* to have all the required skills. Added to this is the need for customers to see rapid ROI for all infrastructural purchases. It seems a different type of

2. A schema is the set of data definitions for a given repository; for example, table 1 contains an integer key field, table 2 contains text fields, and so on. In other words, the schema describes the database structure similarly to the way a MIB describes management data.

approach is needed for developing management systems, one that involves adoption of:

- A solution mindset
- Distributed, creative problem solving
- Taking ownership
- Acquiring domain expertise
- Embracing short development cycles
- Minimizing code changes
- Strong testing capability

Acquiring skills like these would positively enhance the development process. We examine strategies for developing these capabilities in the next chapter. For now, the elements of them and their advantages are described.

Developer Note: A Solution Mindset

Adopting a solution mindset is an important first step in effective NMS development. It reflects the move away from the purely technological aspect of products to embrace the way enterprises and service providers now look at overall solutions to business problems. The days of box shifting (selling NEs with little or no management facilities) are probably gone forever. This is no bad thing because the real cost of adding NEs to networks is the time and money required to debug and integrate them into the fabric of the network. This is why network operators require ease of management for new NEs. An extra problem for vendors is that hardware is increasingly sold at heavily discounted prices. The revenue from hardware sales may no longer match the effort required for development, though upgrades and enhancements may help offset this. Customers will pay for overall solutions that make it easier to manage and operate their networks. An example is consolidation of incompatible NMS into a single NMS, as we saw in Chapter 1. Solutions have a number of characteristics:

- Clear economic value
- Fulfillment of important requirements
- Resolution of one or more end-user problems

An issue facing many network operators is what to do with legacy layer 2 equipment. Should the operator simply throw away its existing hardware? This is a difficult question, and providing a migration path for such users is a good example of a solution. Existing deployed device software should also be main-

tained by the vendor for as long as possible in order to protect the network operator investment. (This is often easier said than done, as devices such as PABXs are increasingly discontinued because they have reached the end of their lifecycle.) Large networks don't change overnight, so management systems should be written to accommodate both legacy and new equipment. The MPLS ships-in-the-night (SIN) option that we discussed in Chapter 2 is an example of such an approach. SIN is a special mode of operation on MPLS nodes. It allows ATM users to upgrade their firmware (some devices may also need hardware upgrades) to MPLS and then simultaneously use both ATM and MPLS control planes on the same switch. The two technologies do not interact, but pass each other like ships in the night (hence the name). The logical progression of this is to try to allow any layer 2 service to cross an MPLS cloud. This is a good example of solutions thinking because it saves money, protects existing investments, and addresses important user problems.

Well-engineered management solutions are also of benefit to vendors when they are built from components. The elements of such solutions can be re-used in other areas and products. The vendor can leverage the solution for many purposes. Examples of management systems solutions include the following:

- Providing minimal management support for third-party devices. Many NMS are proprietary, supporting only the equipment vendor's hardware. Networks may contain multivendor NEs, so separate NMS are often required to support what are often very similar devices from different vendors. It is better for end users if the incumbent NMS provides limited (rather than no) support for third-party NEs. NMS vendors should be prepared to offer this support even if it means just device discovery and notification/trap handling.
- Creating generic management system components that can be used across numerous different products and technologies, such as ATM and MPLS connections. An ATM virtual circuit is not the same thing as an MPLS LSP, but the management software can still provide a technology-independent GUI abstraction for both. The user is then freed from the complexity of the underlying technologies and can perform similar management operations for both. This also reduces training time.
- Aiming for technology-independent software infrastructure using standard middleware, such as CORBA-based products, rather than custom-built facilities.

As far as possible, the management system should also provide code encapsulation for functions such as SNMP access, network message transport, and network protocols. This is illustrated in Figure 3–4, where the FCAPS areas are shielded from the complexities of the underlying SNMP, messaging services, and network technologies.

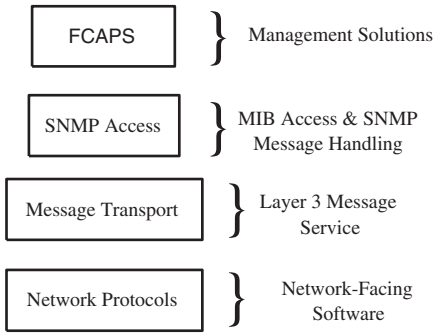


Figure 3–4 FCAPS software layers.

While this seems an obvious point relating to good software development practice, it's surprising how often low-level code (such as SNMP API calls) is called directly from the FCAPS layer. In many cases, this is just poor coding practice caused by inexperience or excessive haste. Then, the smallest change in the low-level code requires a full FCAPS rebuild. It is important that changes to MIBs or underlying protocols should *not* necessitate a full rebuild of the management system. This loose coupling (via APIs and layering) between components makes it easier for developers to take ownership of substantial product areas. In turn, this can help in avoiding situations in which a change to one component breaks the code in another component.

Developer Note: Distributed, Creative Problem Solving

Once management systems have been built and integrated into a vendor test network, they often present complex problems. The distributed nature of managed networks (NEs with local agents reporting to a centralized NMS) and the broad functional composition of NMS present difficult logistical development problems. Solving such problems is where the expanded skill set comes into its own. Typical problems are:

- Software bugs
- NE bugs (can be very hard to identify)

- Performance bottlenecks in any of the FCAPS applications due to congestion in the network, DBMS, agent, manager, and so on
- Database problems such as deadlocks, client disconnections, log files filling up, and so on
- Client applications crashing intermittently
- MIB table corruption, such as a number of `set` operations that only partially succeed—for example, three `setRequests` (against a MIB table) are sent but one message results in an agent timeout and the other two are successful, which could leave the table in an inconsistent state
- SNMP agent exceptions

Solving these and other problems requires a wide-ranging view of the system components and an excellent understanding of technologies like SNMP, databases, and networking. It also requires a creative approach to the use of debuggers, MIB browsers, trace files, and so on. Clearly, part of creative problem solving is a requirement for developers to have a high aptitude for testing. Such developers leverage their product and software knowledge to comprehensively test the system *prior* to delivery to QA. This helps in providing solid software builds to QA. Organizations play a role in facilitating this by the provision of many of the excellent tools available, including:

- UML support packages
- Java/C++/SDL products
- Version control
- Debuggers

The ultimate goal is zero-defect software. The complexity of NMS code often means that bug fixes can take a long time, often a day or more. Taking the time to do this is nearly always a good investment, provided any changes are properly tested.

Developer Note: Taking Ownership

Taking ownership is another important part of a solution mindset. In this, engineers strive to produce a complete feature without the need for handing off part of the development to others. A broad task can be ring-fenced by a small group of developers who take responsibility for design, development, and delivery. Given the skill set required for management system development, this is a difficult undertaking. It means that traditional development boundaries are

removed: no more pure GUI, backend, or database developers. All NMS software developers should strive to extend their portfolio of skills to achieve this.

Another aspect of taking ownership is being prepared to fix bugs in old code produced in earlier projects. This can be achieved in conjunction with maintenance and support developers. The important point is that ownership is maintained even as new projects are undertaken. This has the additional merit of extending institutional memory and minimizing the incidence of coding errors during support bug fixes. Institutional memory relates to individual developers with key knowledge of product infrastructure. It equips the organization to smoothly migrate products over numerous release cycles and is an essential skill for long-term development. The end result is more robust management software in customer networks.

Developer Note: Acquiring Domain Expertise and Linked Overviews

Many service providers employ domain experts for producing documents such as bid requests and requests for proposal. These are highly detailed documents that are sent to vendors. Service provider domain experts may be permanent staff or external consultants. Vendors tend to employ sales and marketing executives as inhouse domain experts. The interplay between these two groups ultimately drives much of the vendor's engineering effort. Both groups of domain experts tend to have impressive expertise. It is important that these skills are also available in engineering, because domain experts [JavaDev] tend to be in great demand. In other words, engineers need to become domain experts as well.

Domain expertise represents a range of detailed knowledge, such as IP/MPLS, that can be readily applied to the needs of an organization. For service providers, the knowledge of their domain experts is leveraged for structuring bid and proposal documents and generally formulating short-, medium-, and long-term strategies. Such knowledge might include areas such as:

- Layer 2 and layer 3 traffic engineering
- Layer 2 and layer 3 QoS
- Network management
- Convergence of legacy technologies into IP. Many service providers have built large IP networks in anticipation of forecasted massive demand. These IP networks are, in many cases, not profitable, so service providers

are keen to push existing revenue-generating services (such as layer 2) over them.

- Backward and forward compatibility of new technologies, such as MPLS. An example is that of a service provider with existing, revenue-generating services such as ATM, FR, TDM, and Ethernet. The service provider wants to retain customers but migrate the numerous incoming services into a common MPLS core.

The choice of technology, systems, and devices in each of the above areas is critical and is an opportunity for one or more domain experts.

Domain expertise is needed by engineers, for example, when adding technologies such as Gigabit Ethernet or MPLS to a suite of IP routers or ATM switches. The acquisition of domain expertise is an essential component of solutions engineering. This is easier said than done, because the number of technologies is increasing at the same time as layers 2 and 3 converge. Interestingly, the boundaries of modern networks are also shifting: Devices that were in the core a few years ago are now moving out to the edge. Also, devices that were in the access layer can be enhanced and moved into the distribution layer. In many cases, the different network layers may have individual management systems. The movement of devices across the layers means that support for a given NE may have to be removed from one management system and added to another. This adds to the knowledge burden of developers. Acquiring domain expertise is necessary for hard-pressed developers.

A key to becoming a domain expert lies in what we call *linked overviews,* described in the next section.

Linked Overviews

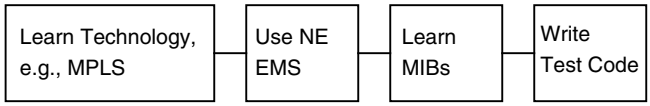
An increasingly common problem faced by NMS software developers is implementing network management support for new NE features, such as MPLS. The following four steps provide a linked overview of the management aspects of an NE feature:

1. Acquire a basic understanding of the managed technology—what it does, how it operates, and so on. For MPLS, the basic purpose is to carry traffic across an MPLS core, so connections (LSPs and tunnels) are important. These tunnels can be traffic-engineered and can also support

QoS, so path and resource blocks (more on these later in the chapter) are needed.

2. Use the EMS to get an understanding of how the NE provides the feature; for example, for MPLS, the user can separately create objects like paths and resource blocks. Then, these can be combined to create MPLS LSPs or tunnels. User manuals are often very useful during this process.
3. View the relevant MIBs using a MIB browser.
4. Write simple test code (e.g., in Java or C++) to view and modify the MIB objects.

Step 4 essentially automates the actions in steps 2 and 3. The software produced in step 4 can then form the basis of Java classes that can eventually be imported into the finished NMS. The final stage in development is then adding code to the NMS to implement the overall MPLS management feature (i.e., FCAPS). An important observation about the above is that it depicts NMS development as a type of reverse engineering. If network management is provided at the end of NE development, then it has a reverse engineering flavor. If the two occur in parallel, then no real reverse engineering effort is required. We therefore view a linked overview as the resulting knowledge emanating from following the above four steps.



Some vendors provide simultaneous releases of both NE firmware and NMS software. In other words, NE and NMS development are inextricably interwoven.

Step 1 can be assisted using the RFCs on the IETF Web site [IETFWeb]. The other steps are carried out in conjunction with the NEs themselves. Some examples of linked overviews now follow.

Developer Note: An ATM Linked Overview

Many technologies can seem extremely complex at first, and then, as the learning curve progresses, the essential patterns begin to emerge. ATM [Tanenbaum1996], [ATMObjects] is a good example. Stripped down to its bare (linked overview) essentials:

- ATM is a layer 2 protocol suitable for deployment in a range of operational environments (in VLANs and ELANs, in the WAN, and also in SP networks).
- ATM offers a number of different categories and classes of service. The required service level is enforced by switches using policing (traffic cop function), shaping (modifying the traffic interarrival time), marking (for subsequent processing), and dropping.
- Traffic is presented to an ATM switch and converted into a stream of 53-byte ATM cells.
- The stream of cells is transmitted through an ATM cloud.
- A preconfigured virtual circuit dictates the route taken by the cell stream. Virtual circuits can be created either manually or using a signaling protocol. If no virtual circuit is present then PNNI can signal switched virtual circuits (SVCs).
- The virtual circuit route passes through intermediate node interfaces and uses a label-based addressing scheme.
- Bandwidth can be reserved along the path of this virtual circuit in what is called a contract.
- Various traffic engineering capabilities are available, such as dictating the route for a virtual circuit.

This list provides an overview of ATM technology. It joins (or links) the principal components needed for managing ATM. From this list, the essential ATM managed objects can be derived:

- ATM nodes
- A virtual circuit (switched, permanent, or soft) spanning one or more nodes
- A set of interfaces and links
- A set of locally significant labels used for addressing
- An optional route or designated transit list
- A bandwidth contract
- Traffic engineering settings
- QoS details

This is a good start on the road to defining managed objects for the support of ATM. It points to the merit of studying documents from the ATM Forum and ITU-T Broadband ISDN. The next stage (step 2) would be to experiment with the EMS of an ATM switch and see the above objects in practice, e.g., cre-

ating PVCs and SPVCs. Next, we would examine the MIB objects (step 3) [ATMObjects] involved in step 2, and then produce software (step 4) to read and write instances of those objects.

Developer Note: An IP Linked Overview

The convergence of layers 2 and 3 (e.g., connection-oriented operation of layer 3 networks) has forced the need for knowledge about IP onto practically everyone's agenda. IP is often used as an abbreviated form of TCP/IP, and this is the way it is used in this book. Like UNIX and Ethernet, IP is one of the great software engineering feats of the 20th century. Both are in widespread use (although UNIX, unlike DOS or Windows, has no single standard implementation) and have withstood the test of time. IP has a steep learning curve, but it can be summarized into a linked overview as follows:

- IP is packet-based—IP nodes make forwarding decisions with *every* packet.
- IP is *not* connection-oriented.
- IP provides a single class of service: best effort.
- IP does not provide traffic engineering capabilities.
- IP packets have two main sections: header and data.
- IP header lookups are required at each hop (with the present line-rate technology, lookups are no longer such a big issue. Routing protocol convergence may cause more problems).
- IP devices are either hosts or routers (often called gateways).
- Hosts do not forward IP packets—routers do.
- IP devices have routing tables.
- IP operates in conjunction with other protocols, such as OSPF, IS-IS, Border Gateway Protocol 4 (BGP4), and Internet Control Message Protocol (ICMP).
- Large IP networks can be structured as autonomous systems made up of smaller interior areas or levels.

So, the essential managed objects of IP are:

- IP nodes (routers, hosts, clients, servers)
- IP interfaces
- IP subnets

- IP protocols (routed protocols such as TCP/IP and routing protocols such as OSPF and IS-IS)
- Interior Gateway Protocol (IGP) areas (OSPF) or levels (IS-IS)
- Exterior Gateway Protocol (EGP) autonomous systems

The next stage (step 2) would be to experiment with the EMS of an IP router (or an IP/MPLS switch) and see the above objects in practice, for example, creating IP interfaces and subnets, and configuring routing protocols. Next, we would examine the MIB objects (step 3) involved in step 2 and then produce software (step 4) to read and write instances of those objects.

Embracing Short Development Cycles

The need for immediate ROI is prompting a demand for innovative management system solutions. Enterprises must be able to leverage their investments for both productivity (easier operation of networks) and financial gains (smoother business processes). This can result in shorter NMS software development cycles, which in turn requires a modified approach to producing NMS:

- Reduced feature sets in more frequent releases
- Foundation releases
- Good upgrade paths
- Getting good operational feedback from end users

If a management system release occurs every four or five months, then it is no longer necessary for every single requirement to be fulfilled. Instead, requirements can and probably should be prioritized over a range of releases. When a new technology is adopted and implemented on a range of NEs (such as VoIP or FR interworking), then only the mandatory management facilities should be implemented first. The rest can follow later. The first release becomes a foundation for the later ones. In time, the initial reduced feature set grows steadily to become part of a sophisticated end-user solution. Not all users would necessarily upgrade—just the ones who need the new foundation release features. The developers would carry out the bulk of the testing. As the releases occur, the user's data has to be carefully protected and upgraded as necessary. So, upgrade issues (such as scalability-related database schema changes) increasingly have to become part of the bread-and-butter of development.

Implicit in all this is the end user becoming a development partner providing valuable operational feedback. The user receives regular, reliable releases, and

the vendor sees fast return on development investment. Another benefit is that developers gain experience and expertise with each of the minor releases.

Minimizing Code Changes

Perhaps one of the most difficult software development skills to acquire is the ability to resist changing code. This applies to good and bad code, old and new. A crucial skill for developing NMS software is the ability to make small, focused fixes for specific problems without introducing new bugs. It can be extremely difficult to resist making simultaneous changes to neighboring code, but it is a vital discipline. Unnecessary code changes introduce bugs and increase the need for testing. Every code change should be fully tested as part of a mandatory change control process.

Elements of NMS Development

Development of NMS software is interesting and challenging. This section introduces some of the typical development areas and some important issues that often arise.

NMS Development

A typical scenario for a management system developer is the following:

- Using a browser-based GUI, the developer has provisioned onto the network a managed object such as an ATM virtual circuit or an MPLS LSP.
- The developer wants to check that the software executed the correct actions.
- During provisioning, the developer verifies that the correct Java code executed using a Java console and trace files (similar actions can be done for C/C++ systems).
- The database is updated by the management system code, and this can be checked by running an appropriate SQL script.
- The next step is verifying that the correct set of managed objects was written to the NE. To do this, the developer uses a MIB browser to check that the row object has been written to the associated agent MIB.

Clearly, this type of development requires a broad range of skills. Other skills not mentioned include:

- Data analysis—matching NE data to the NMS database schema
- Data analysis—defining NMS-resident objects that exist in complex component form in the network (an example is a VPN, as discussed earlier in this chapter)
- Upgrade considerations for when MIBs change (as they regularly do)
- UML, Java, and object-oriented development
- Class design for major NMS features, like MPLS provisioning
- GUI development
- Middleware using CORBA-based products
- Insulating applications from low-level code

These are now described in more detail.

Data Analysis

MIBs are crucial components in a management system. Many vendors provide the MIB modules for their NEs in the form of a number of text files. These files can then be incorporated into an NMS and also used in conjunction with MIB browsers. MIBs contain the managed object definitions of interest and are used to derive part of the database schema for the NMS. This latter is the structure and definition of the data used as the basis of the NMS. We assume the use of relational database technology in this book, and the model for this consists of tables. Typically, the NMS database schema contains a great many tables; for instance, there might be a table for storing the details of LSPs, another for PVCs, and so on. The schema represents an overall data definition for the NMS, and the managed object data is also defined here. This latter point means that there is a degree of duplication in that the NMS has a schema and the MIB objects of interest are contained in it as well as in the agent MIBs. This is what was meant earlier in the chapter when we mentioned storing the same data in two different places. The NMS tracks and modifies the values of NE-managed object instances and stores these in its own database.

While the MIBs can be used to form the basis of the management system information model, there are additional elements needed in the NMS database schema. These are default values, or the values used when the database is first built. Examples are −1 for integers and NULL for booleans (i.e., neither true nor false). Later, we will see the need for sensible default values, particularly for MIB objects that can be modified by the NMS.

The database product can be any of the excellent, general-purpose engines available, such as Informix and Oracle.

When MIBs Change: Upgrade Considerations

Supporting different MIB versions is a recurring network management problem. Let's assume that a given network has 20 of the same ATM/MPLS switch model all running firmware revision 3.2. Now, the network operator decides to upgrade five of the switches to firmware revision 4.0 (in order to gain access to some new feature). Upgrading software like this can be expensive if it results in any downtime due to software bugs. The cost can also be increased if extra hardware is needed, such as a processor upgrade, more RAM or flash memory, or extra line cards.

The NMS can provide assistance during upgrades by downloading the new image to the selected switches (and backing up the old image). However, the new features added to the switches means that they now support extended and possibly also new MIB modules. The NMS must continue to be able to interact with the devices not upgraded, so it must be able to recognize both the new and the old MIBs. Following are guidelines for providing an upgrade path following MIB changes:

- Deprecate old objects no longer in use—don't delete them from the MIB if at all possible.
- Keep the MIB object identifiers sequential; add new OIDs as necessary. It is not uncommon for new columns to be added to MIB tables as NEs are enhanced. The old objects should not be modified during any such enhancements in order to maintain backward compatibility.
- Don't change any existing OIDs in MIBs that are currently in use by the NMS. RFC 2578 provides guidelines for this.
- Ensure that MIB files do not have to be changed in order to work with management systems. Sometimes MIBs that successfully compile into agents cannot be parsed into management systems. This can be caused by limitations on the part of the management system or the agent parser. Whatever the reason, it is important that no manual changes are needed in order to incorporate MIBs into an NMS. This underlines the crucial role played by MIBs.

Following these guidelines helps provide a seamless upgrade path for the addition of new MIB objects while at the same time maintaining support for

existing ones. MIB objects should only ever be removed with the utmost caution because there may be management system software that relies on their presence. Non-existent MIB objects that are accessed by an NMS will result in an SNMP exception propagating back to the NMS.

Adding new technologies to NEs is a major cause of significant MIB changes. This causes additional problems for the management system because (as we've seen) it derives its managed object model from the MIBs. New MIB objects that are needed in the database require corresponding schema changes. These can be effected using either SQL scripts or special-purpose code. Changing the management system schema is not without risk. Existing application code is affected, and it can easily introduce bugs. The skills required to match MIB changes to schema updates are very important.

UML, Java, and Object-Oriented Development

The use of standards is a recurring theme throughout the networking industry. It is essential for management system developers to adopt a standards-based approach in their work. To this end, we can use linked overviews in conjunction with documents like IETF RFCs and ETSI/ANSI standards. UML [UML-Rumbaugh] provides a standard technique for system development. It provides some very useful tools for both specification and development.

UML allows for the development process to be opened up to a degree that is difficult to match with older methods such as the waterfall model. It allows for the separation of requirements from specification and design decisions by the provision of different views including:

- Structured classification (use cases, classes, components, and nodes)
- Dynamic behavior (describes system changes over time)
- Model management (organization of the model itself)

In effect, UML provides a model-based scheme of development. The model and its associated views then become part of the finished software.

Domain experts and other stakeholders can assist the development by defining the principal actors (either internal system users or end users) in the management system. Use cases provide a means for defining the ways in which the actors interact with the system. Sequence diagrams describe message exchanges that go together to make up transactions. The different views fit well into a solution-engineering context because they naturally allow for several different perspectives. These and the other features of UML can greatly assist the

construction of robust management systems. Products like Rational Rose are excellent for this purpose [RationalRose].

Class Design for Major NMS Features

Class design can be initiated from UML use cases. Using tools like Rational Rose helps to facilitate automation of this process. Normally, generated classes are merely skeletons with no code included. However, even this is useful, because once the main classes are defined, the programming task is bounded. The great merit of UML is that the stakeholders can influence the class structure. This departs from the older approach in which stakeholders just specify requirements.

GUI Development

An important aspect of management system development is the GUI. This is particularly so when the client is thin. A well-designed GUI reduces the need for training and provides an effective tool for managing networks. It provides the user interface and should be as generic as possible. To this end, visual controls should be as technology-independent as possible, for example, using the words like *connection* instead of PVC or LSP/tunnel, or *routes* instead of ATM designated transit list or MPLS Explicit Route Object (we describe these last two objects later in the chapter—for the moment, let's take them as simple paths through a network). The visual controls should also hide as much complexity as possible and provide sensible default values that the user can override if required.

There are many excellent tools, such as Borland JBuilder, available for crafting GUIs. Often, the GUI is the last item of a management system to be fully tested. So, the individual GUI components should be fully tested using tools like JBuilder well in advance of full integration. Developers' problem-solving skills should be focused on fully exercising as much GUI code as possible to reduce delays in delivery.

Middleware Using CORBA-Based Products

CORBA is a suite of specifications issued by the Object Management Group [OMGWeb]. Using the Internet Inter-Orb Protocol (IIOP), a CORBA-based program from any vendor can interoperate with another vendor's CORBA-based program. This works for almost any combination of platforms, program-

ming languages, and networks. CORBA applications are made up of objects that present an interface. This interface is part of the contract that a server object offers to the clients that invoke it. The interface is defined in the OMG Interface Definition Language (IDL) and is completely separate from the underlying implementation. So, application A written on PC X can call the exported methods of classes in application B on UNIX workstation Y located in a remote network. CORBA provides the glue needed to join many different code components together into what looks like a single system.

The distributed nature of network management provides a good setting for the use of CORBA-based software. One example is SNMP notification/trap management. When the management system receives a trap from the network, it can store the details in a database and then notify another application such as a GUI client. This notification can take the form of invoking an object in a CORBA application implemented on the client machine. This can also be achieved using technologies such as Java Remote Method Invocation (RMI), RPC, or COM, but CORBA provides what is almost complete independence from the underlying systems and networks. This is an extremely powerful capability.

Insulating Applications from Low-Level Code

Insulating applications from low-level code was briefly described earlier and illustrated in Figure 3–4. It is very important that the various layers of management system software be as technology-independent as possible. This is similar in concept to the way in which network architectures are layered. Each layer is implemented independently. Only the services offered to upper layers are exposed [Tanenbaum1996]. The implementation is encapsulated inside the layer. In a similar fashion, low-level code that provides access to technology such as SNMP, MIBs, IP, and User Datagram Protocol (UDP) should be partitioned as much as possible. Only a simple interface should be exposed to the layer above. This also aids comprehension by other developers. We will see this technique in the MPLS case study in Chapter 8.

Expensive (and Scarce) Operational Skill Sets

Skills shortages are not restricted to the supply side of the industry. The growing complexity of networks is pointing to increasingly scarce operational skills. After all the investment has been made and the training courses run, it is up to the network operator to deploy skilled personnel to actually run the network. The

skill set required for running networks is increasingly broad and includes knowledge of a wide range of different technologies. These technologies are not restricted to a single layer of the OSI model; they include devices that support the following:

- ATM
- FR
- Gigabit Ethernet
- Optical technologies such as SONET/SDH, DWDM, cross-connects, and multiplexers
- Access
- Transport
- MPLS
- IP
- VPN
- VoIP
- SAN and NAS
- Firewalls, load balancing, servers

Added to this is possibly more than one management system, particularly for complex networks. So, the operator needs to have a reasonable understanding of the various management systems. The types of skills needed by network operators are:

- Deploying and configuring devices
- Setting up and enabling routing and signaling protocols
- Partitioning layer 2 and layer 3 networks
- Billing and accounting
- Planning, including capacity planning
- Performance
- Provisioning of LSPs, PVCs, and so on
- Traffic management
- Backup and restore of NE firmware and configuration databases
- MIB browsing
- Trap management
- Security
- Modeling what-if scenarios
- Understanding application trace files and debug facilities

Just as for developers, this is a complex skill set. Vendors can greatly assist network operators by providing high-quality solutions in both the NEs and NMS. Network operators should also try to keep up with new technologies by studying the relevant standards documents.

Multiservice Switches

We have made much mention of the migration towards a packet-based infrastructure and its relevance to enterprise network operators. We have also noted that enterprise networks usually contain much legacy equipment. Enterprise network operators typically want to:

- Reduce the payback period for new purchases
- Maintain and expand existing network services
- Reduce operational costs associated with multiple networks, such as telephony and LAN

MPLS provides a way of filling these needs in conjunction with multiservice switches. These switches allow specified levels of QoS and traffic engineering for the following technologies:

- ATM
- FR
- TDM
- IP

It is anticipated [MultiserviceSwitch] that these technologies will be deployed on multiservice switches once the relevant standards have been finalized.

MPLS: Second Chunk

Chapter 2 gave a brief introduction to MPLS and described how it enables IP networks to be operated in a fashion similar to layer 2 networks. Currently, ATM is the only layer 2 technology that provides traffic engineering and guaranteed QoS for a range of network service classes—voice, video, and data. A specified class of service can be allocated to new services without affecting the existing ones (provided the network has the appropriate bandwidth resources). MPLS promises to bring similar flexibility to layer 3 networks.[3]

This second chunk of MPLS digs a little more deeply into MPLS and more fully describes the principal components of the technology. This is another

example of a linked overview, which strips the technology down to its essential components. These can be considered to be the managed objects of MPLS:

- Explicit Route Objects (ERO), strict and loose
- Resource blocks
- Tunnels and LSPs
- In-segments
- Out-segments
- Cross-connects
- Routing protocols
- Signaling protocols
- Label operations: lookup, push, swap, and pop
- Traffic engineering
- QoS

As we'll see, the hardest way to manage MPLS networks is to not use signaling to set up LSPs. Why would a user not want to use signaling? The NEs might not support signaling. Another reason is control: The operator alone might want to dictate all the objects that get provisioned in an unsignaled network. Some large service providers actually operate their ATM networks in this way and might decide to do the same with MPLS. With no signaling support, it is up to the operator to create all of the above objects—we will do just this in Chapter 8 when we manually create an LSP. Another reason for not wanting signaling protocols running in the network is the extra resources they consume. All networks are only as strong as their weakest link, and with complex signaling protocols running in the background, bottlenecks may suddenly appear unexpectedly. We don't necessarily agree with running or not running any particular signaling protocols—we just mention the above as possible explanations. One important point is that signaling is probably required as networks become very large (tens to hundreds of thousands of nodes).

We describe LSP setup using the MPLS MIBs in some detail in Chapter 8.

3. We should point out that a lot of work is also underway to use MPLS for the transport of legacy traffic below layer 3, such as Ethernet, ATM AAL5, and voice. MPLS offers similar capabilities to the operators of such networks in the form of LSPs and so on. So, for our discussions we assume that the traffic type being transported is not so important from the management perspective.

Explicit Route Objects

An ERO is a list of layer 3 address hops inside an MPLS cloud. Similar to an ATM designated transit list (DTL), it describes a list of MPLS nodes through which a tunnel passes. The path taken by the tunnels in Figure 3–3 is an example of an ERO. The purpose of an ERO is to allow the user to specify the route that a tunnel will take. In other words, it allows the user to constrain the route. EROs can be either strict or loose. A strict ERO specifies *all* the hops in the path. A loose ERO allows for intermediate networks in the path, such as another cloud (e.g., an SP network). EROs are stored in a MIB table on the originating node and can be used by more than one tunnel originating on that MPLS node. EROs are *not* used in the manual creation of LSPs.

EROs are used by signaling protocols (such as RSVP-TE) to create tunnels. The path specified in the ERO must be realizable (i.e., links must exist between the designated nodes) and any required bandwidth resources (described in the next section) must be available.

Resource Blocks

MPLS permits the reservation of resources in the network. This provides a means for network operators to deterministically carve up their bandwidth and allocate it to specific LSPs. Resource blocks provide a means for recording the bandwidth settings, and they can then be assigned to specific LSPs. The components of a resource block include:

- Maximum reserved bandwidth
- Maximum traffic burst size
- Packet length

A given LSP, such as one of those in Figure 3–3, could have an associated end-to-end bandwidth reservation of 2Mbps. This means that the LSP is engineered to carry 2Mbps of traffic along the specified route. Best-effort LSP operation is also possible, in which case the resource block is null.

Tunnels and LSPs

MPLS tunnels (as we saw in Chapter 2, Figure 2–8) represent a type of container for paths made up of nodes with configured in-segments, cross-connects, and out-segments. The tunnel is the on-ramp into the associated label switched path. What we didn't say in Figure 2–8 is that tunnel instances are also sup-

ported. A tunnel instance is a separate tunnel, which is in some sense owned by the first tunnel. In other words, the tunnel and its tunnel instances have a relationship with each other. This relationship consists of sharing the same endpoints and might be needed for backup (a disjoint path) or for load sharing. We will see this more clearly in Chapter 8.

MPLS-encapsulated packets enter the tunnel, pass across the appropriate path, and exhibit three important characteristics:

- Forwarding is based on MPLS label (rather than IP header) lookups (this is no longer an advantage enjoyed by MPLS nodes, as IP routers can now forward at line rate).
- Resource usage is fixed, based on those reserved at the time of connection creation.
- The path taken by the traffic is constrained by the path chosen in advance by the user.

Tunnels and LSPs provide reachability for traffic with specific destination IP addresses. Routing protocols direct packets onto specific tunnels and LSPs in order to reach the appropriate IP destination.

In-Segments and Out-Segments

In-segments on an MPLS node represent the point of ingress for traffic. Out-segments represent the point of egress for traffic. The two types of segment objects are logically combined using a cross-connect.

Cross-Connects

Cross-connects are objects that associate in- and out-segments together. The MPLS node uses the cross-connect settings to decide how to switch traffic between the segments. The cross-connect table supports the following connection types:

- Point-to-point
- Point-to-multipoint
- Multipoint-to-point

A cross-connect entry has both an administrative status and an operational status. The administrative status indicates the state required by the operator, whereas the operational status indicates the actual state of the cross-connect in the node. Operationally down cross-connects will not forward packets.

Routing Protocols

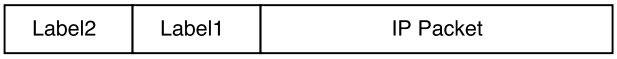
MPLS incorporates standard IP routing protocols such as OSPF, IS-IS and BGP4. This is done because these protocols have been used and proven over the course of many years. Incorporating them into the MPLS standards improved the chances of widespread deployment of MPLS. Traffic engineering extensions added to the routing protocols means that they can advertise and distribute both routing and resource (e.g., link bandwidth) details. This is crucial for facilitating the creation of route-constrained LSPs (i.e., tunnels). This ultimately allows the user requirements to influence the path taken by IP traffic through an MPLS cloud.

Signaling Protocols

As we've seen, the creation of LSPs and tunnels can be achieved either manually (similar to the way ATM PVCs are created) or via signaling. Signaled connections have resources reserved, labels distributed, and paths selected by protocols such as RSVP-TE or LDP.

Label Operations

As MPLS-labeled traffic traverses an LSP or tunnel, it is forwarded based on its encapsulated label value. The IP header is no longer consulted while the packet is inside the MPLS domain. MPLS labels can be stacked in a last-in-first-out fashion; that is, more than one label can be applied to a packet. Labels can be stacked (if the hardware supports it) up to the limit allowed by the layer 2 protocol. The label counts as part of the layer 2 message size. The outermost label (i.e., the last one pushed on the stack) is the one used for forwarding the packet. If more than one label is on the stack, then stripping off (or popping) the topmost label exposes the next label down. Forwarding is then carried out based on this label. This is illustrated below with an MPLS-encapsulated packet that has a stack containing two labels.



The current MPLS node uses Label2 when forwarding this packet. The operations that can be executed against labels are:

- Lookup: The node examines the value of the topmost label. This operation occurs at every node in an MPLS cloud. In our example, lookup would

occur using Label2. Typically, a label lookup results in the packet being relabeled and forwarded through a node interface indicated by the incoming label.

- Swap: This occurs when an MPLS node replaces the label with a new one.
- Pop: This occurs when the topmost label is removed from the stack. If the label stack has a depth of one, then the packet is no longer MPLS-encapsulated. In this case, an IP lookup can be performed using the IP header.
- Push: This occurs when a label is either pushed onto the label stack or attached to an unlabeled packet.

In Chapter 4, "Solving the Network Management Problem," we will see that the MPLS shim header contains a bit field called Stack. A value of 1 in the stack field indicates that this is the last remaining label in the stack; the value zero indicates that other labels are pushed beneath the current label. The value of the Stack field changes appropriately as labels are pushed and popped. An important point to note is that the MPLS labels have local significance only. The contents of MPLS labels can also assist in the QoS scheme (we will see this use of labels in more detail in Chapter 4).

MPLS Encapsulation

FR and ATM can accommodate MPLS labels in their layer 2 headers. Other technologies use a shim header for the label. This is a mini-header (more than one is allowed because stacking is supported) that sits beside the IP header. FR uses the data link connection identifier (DLCI) field, and ATM uses the VPI/VCI fields. ATM has another field called cell loss priority (CLP) that is used for QoS support. This field is used to mark cells for two service levels: Cells with a CLP of 1 are discarded (if incoming traffic levels are too high) prior to cells with a CLP of 0. Cells with a CLP of 0 are not guaranteed to be forwarded, but they will have precedence over cells with CLP of 1.

The MPLS encapsulation specifies four reserved label values:

- 0 – IPv4 explicit null that signals the receiving node to pop the label and execute an IP lookup
- 1 – Router alert that indicates to the receiving node to examine the packet more closely rather than simply forwarding it
- 2 – IPv6 explicit null
- 3 – Implicit null that signals the receiving node to pop the label and execute an IP lookup

When an MPLS node operates in SIN mode (ATM and MPLS running simultaneously), there may be additional constraints on the label range, but this is platform-specific.

QoS and Traffic Engineering

The Internet currently offers a single default service level: best effort. Many enterprises also offer just best-effort QoS for IP traffic but on an overengineered underlying network. Bandwidth in the LAN is relatively cheap and can be augmented as needed using switches. This means that the excess bandwidth helps avoid congestion, but this is a blunt instrument that doesn't scale well and is prone to congestion during very high bursts of traffic. In effect, overengineering passively avoids congestion but doesn't proactively prevent it. It's not unknown for the developers of NMS software to inadvertently flood their local LAN with SNMP traffic. Without proactive QoS and traffic engineering in the network, the enterprise network manager often has to resort to restricting access to the network to avoid such problems. A better solution would be to provide specific chunks of bandwidth (via MPLS LSPs) to the developers. WAN links normally have strictly limited bandwidth and provide an even stronger case for QoS management.

Traffic engineering is set to become a mandatory element of converged layer 3 enterprise networks [MPLS&Profits]. MPLS provides resource advertisements in its routing protocols (extended from the regular IP operation). Link bandwidth states are included as extensions of the standard IP routing protocols such as OSPF and IS-IS. These are traffic-engineering enhancements, and the modified protocols are referred to as OSPF-TE and IS-IS-TE. The purpose of the enhancements is to allow MPLS routers to construct a complete link-state database of the network, including available/allocated bandwidth.

QoS

The need for definable levels of QoS is due to the increasingly mission-critical, real-time applications being deployed on enterprise and SP networks. Overengineering of the core gets you only so far, but the WAN may then become a bottleneck (remember that a network is only ever as strong as its weakest link). Once the overengineered cores become depleted, there is a need for IP QoS. The issue of SP core depletion is interesting and slightly controversial because it is estimated [TimesMarch2002] that between 1998 and 2002,

service providers invested about $500 billion in fiber-optic networks and that only four percent of the new fiber is in use. The bulk of the fiber is idly sitting underground. Quite possibly, core depletion is therefore not an immediate issue for service providers, but this is not the case for enterprises.

An allied problem is due to the way in which enterprise and SP networks are inextricably interwoven. When an enterprise wants to send, say, email, Web, FTP, and VoIP traffic through an SP network, then a number of approaches can be taken by the enterprise:

1. It can rate all of its traffic as being equally important.
2. It can rate the VoIP traffic as being the most important.

Unfortunately, most enterprises choose the first option. This may be because:

- The enterprise equipment is not capable of marking traffic passed to the service provider.
- There is no incentive from the service provider.
- It is difficult to manage from within the enterprise network.

Whatever the reason, this causes a downstream problem for the service provider because there may be no easy way of differentiating the incoming traffic streams. In practice, it matters little if an email message arrives at its destination one minute or one second after sending it. The same would not be true of real-time, delay-sensitive traffic such as voice or video. As traffic levels increase, these issues become more pressing for service providers, prompting the need for QoS to be applied in both the enterprise and SP networks. Service providers often deploy (and manage) customer premises equipment (CPE) to execute such functions.

There are essentially three approaches that can be adopted for providing different levels of network service [CrollPackmanBW]:

- Best effort, as provided by the Internet
- Fine granularity QoS, such as Integrated Services (IntServ)
- Coarse granularity QoS, such as Differentiated Services (DiffServ)

We're all familiar with the best-effort model from surfing the Internet. The Internet does its best to deliver traffic between senders and receivers, but there is no guarantee. This works well for non-real-time traffic, such as email and Web site access. Internet telephony is also reasonable as long as the users aren't too fussy about quality.

Another approach to service provision is fine granularity QoS of which the IntServ model is an implementation. The IntServ model allows path (called

microflow) reservation across the network, and traffic is pushed into these paths in order to get to its destination. The paths have to be explicitly reserved, and they must also be periodically refreshed. IntServ implementations have two elements of overhead made up of path handling and refreshing.

The third approach is coarse granularity QoS, and it uses the technique of traffic marking. This is the model used by DiffServ. It is often considered to be more scalable than IntServ because no microflows are required. Packets are marked (at the point of origination or by a downstream NE such as a customer edge router) with a specific value. This value is then used as the packet is forwarded through the network. For DiffServ, the values are called DiffServ Code Points (DSCP), each of which corresponds to a given traffic-handling treatment. As the marked traffic makes its way through the network, it is processed at each node in accordance with its DSCP. The DSCP specifies what is called a behavior aggregate; that is, all packets with this DSCP fall into the same category. Each category represents a distinct class of service. When a router receives a packet marked for a given class of service, it imposes what is called a per-hop-behavior (PHB) on the packet. This is the way the packet is treated by the node.

The PHB consists of queuing strategy, policing (dropping the packet or remarking it), shaping, classifying, and so on. It has two main functions:

1. Scheduling the packet into the egress interface
2. Discarding the packet using a drop priority

Scheduling refers not to how packets are stored in the queues, but how they are pulled from the queues by the scheduler. The QoS experienced is the final result of all this activity as this packet (and associated packets) makes its way through the network.

In Chapter 4 we will see how these models are used in the context of MPLS. For the moment, we illustrate how IP packets are marked with DSCPs in the Differentiated Services (DS) field (formerly known as the Type of Service) in the IP header. This is illustrated in Figure 3–5, where the IP header DS field is located in the second octet position. The value assigned to this field can be used to provide specified forwarding treatment for the packet as it traverses the DiffServ domain.

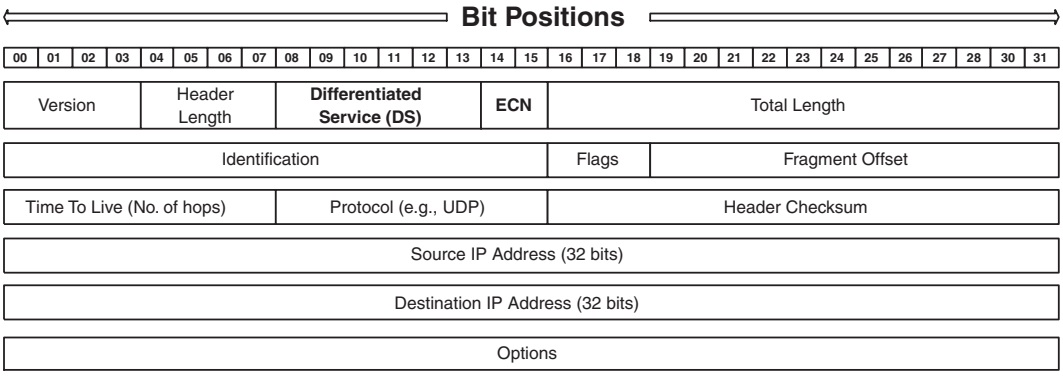


Figure 3–5 The IP header.

RFC 3260 provides an update, clarifications, and new elements of DiffServ. Among other things:

- It clarifies that the DSCP field occupies only the first six bits of the DS field.
- The other two bits will be used for Explicit Congestion Notification (ECN) [DavieRehkter2000], as can be seen in Figure 3–5.

Downstream switches and routers then take the DS value as the cue for special treatment—that is, differentiated services. Packet service metrics can include:

- Bandwidth allocation
- Packet loss rate
- Delay
- Jitter (variation in delay)

The value in the DS field can be used to dictate these metrics. Clearly, the nodes traversed by the marked packets must know the meaning of the DSCPs. This is an important part of DiffServ: the mapping between DSCP and the forwarding treatment. Ultimately, this is reflected in the end-user service level.

MPLS and Scalability

In the standard MPLS MIBs, the tunnels on a given NE reside in the `mplsTunnelTable`. Figure 3–6 illustrates an extract from the MPLS Traffic Engineering MIB [IETF-TE-MPLS].

```
MplsTunnelEntry ::= SEQUENCE {                          1  Sequence or Table
    mplsTunnelIndex             MplsTunnelIndex,
    mplsTunnelInstance          MplsTunnelInstanceIndex,
    mplsTunnelIngressLSRId      MplsLsrIdentifier,
    mplsTunnelEgressLSRId       MplsLsrIdentifier,
    mplsTunnelName              DisplayString,
    mplsTunnelDescr             DisplayString,
    mplsTunnelIsIf              TruthValue,
    mplsTunnelIfIndex           InterfaceIndexOrZero,
    mplsTunnelXCPointer         RowPointer,
    mplsTunnelSignallingProto   INTEGER,
    mplsTunnelSetupPrio         INTEGER,
    mplsTunnelHoldingPrio       INTEGER,
    mplsTunnelSessionAttributes BITS,
    mplsTunnelOwner             INTEGER,
    mplsTunnelLocalProtectInUse TruthValue,
    mplsTunnelResourcePointer   RowPointer,
    mplsTunnelInstancePriority  Unsigned32,
    mplsTunnelHopTableIndex     MplsPathIndexOrZero,
    mplsTunnelARHopTableIndex   MplsPathIndexOrZero,
    mplsTunnelCHopTableIndex    MplsPathIndexOrZero,
    mplsTunnelPrimaryInstance   MplsTunnelInstanceIndex,
    mplsTunnelPrimaryTimeUp     TimeTicks,
    mplsTunnelPathChanges       Counter32,
    mplsTunnelLastPathChange    TimeTicks,
    mplsTunnelCreationTime      TimeStamp,
    mplsTunnelStateTransitions  Counter32,
    mplsTunnelIncludeAnyAffinity MplsTunnelAffinity,
    mplsTunnelIncludeAllAffinity MplsTunnelAffinity,
    mplsTunnelExcludeAllAffinity MplsTunnelAffinity,
    mplsTunnelPathInUse         MplsPathIndexOrZero,
    mplsTunnelRole              INTEGER,
    mplsTunnelTotalUpTime       TimeTicks,
    mplsTunnelInstanceUpTime    TimeTicks,
    mplsTunnelAdminStatus       INTEGER,
    mplsTunnelOperStatus        INTEGER,
    mplsTunnelRowStatus         RowStatus,
    mplsTunnelStorageType       StorageType  }
```

Figure 3–6 The MPLS tunnel table.

Figure 3–6 shows the objects contained in the `mplsTunnelTable`. The `mplsTunnelTable` is made up of instances of `MplsTunnelEntry`, as seen by arrow 1 in Figure 3–6.

Each entry in this table should be seen as a column in a row; for example, `mplsTunnelIndex` can be considered a key value (in the relational database sense). This is depicted in Table 3–1, where some of the columns are arranged and assigned specific values. The exact meanings of the entries in Table 3–1 are explained in Chapter 8. For the moment, a short description is given.

Table 3–1 MPLS Tunnel Table Excerpt

MPLS TUNNEL INDEX	MPLS TUNNEL HOP TABLE INDEX	MPLS TUNNEL INGRESS LSR ID	MPLS TUNNEL NAME
1	1	LER A	TETunnel_1
2	1	LER A	TETunnel_2
3	1	LER A	TETunnel_3
5	1	LER A	TETunnel_5

The column `mplsTunnelIndex` provides a unique key value for each tunnel on the node in question, starting at 1 and increasing with each entry added to the table (tunnel instances are described in Chapter 8). The column `mplsTunnelHopTableIndex` provides an index into a hop table that describes the path taken through the MPLS cloud by the tunnel. The column `mplsTunnelIngressLSRId` is the designated ingress node for the tunnel and has the value LER A for all the tunnels listed in Table 3–1. This column would most likely be either an IP address or a router ID, but a name is chosen here for simplicity. The column `mplsTunnelName` is simply a text descriptor for the tunnel. One notable feature of Table 3–1 is that there is no entry for index 4. This can occur when the user deletes the fourth entry. The table entries are *not* then moved up to fill the gap.

This table can typically include millions of rows (as mentioned earlier in the Light Reading Trials). Let's assume that each row is roughly 300 bytes in size. That means the overall size of the `mplsTunnelTable` for an SNMP agent containing 3 million LSPs is 3,000,000 * 300, or just under 9MB. This would assume a network containing possibly tens or hundreds of thousands of MPLS nodes. It is not practical to try to read or write an object of this size using SNMP. Unfortunately, such an operation might be necessary if a network is being initially commissioned or rebalanced after adding new hardware. Also, many NMS provide a connection discovery feature that must retrieve all virtual circuits (ATM/MPLS) from the network and figure out details for each circuit, such as traffic resource allocations and links traversed. One way to assist in improving scalability is to indicate in the MIB which objects have changed. A scheme for this would be to provide a second table called a tunnel-change table linked to the tunnel table. The tunnel-change table could have summarized boolean entries for a block of tunnel table entries. Let's say we have 1,000,000

tunnels and we assign a block size of 10,000 to the tunnel-change table. This means that any changes in the first 10,000 tunnels are reflected in the first entry in the change table. Any change in the next 10,000 tunnels are reflected in the second change table entry. With a block size of 10,000 we would then have 100 entries in the change table, or 100 * 10,000 = 1,000,000 tunnels. The NMS could then consult the change table in order to see which blocks in the tunnel table have changed. This would help avoid the problem of reading back all the tunnel table entries.

Summary

There are some serious problems affecting network management. Bringing managed data and code together is one of the central foundations of computing and network management. Achieving this union of data and code in a scalable fashion is a problem that gets more difficult as networks grow. MIB tables expand as more network-resident managed objects such as virtual circuits are added. Our first MIB note (on scalability) records a useful object that can help in managing additions to large (integer-indexed) MIB tables. The increased size of networks is matched by ever more dense devices. The latter both help and hinder operators.

The designers of management systems need a rarified skill set that matches the range of technologies embedded in NEs and networks. More emphasis is needed on solutions than on technology, particularly as the components of the technology are combined in new and complex ways, for instance, in layer 2 and layer 3 VPNs. Solutions should try to hide as much of the underlying network complexity as possible. NMS technology can help in hiding unnecessary complexity.

The liberal use of standards documents and linked overviews are some important tools for tackling the complexity of system development, managed object derivation, and definition. Standards documents can be used in conjunction with UML to inform and open up the development process to stakeholders. Like management system developers, network operators also require an increasingly impressive range of skills. Just as for developers, this is both a challenge and an opportunity.

MPLS was described in greater detail, introducing some of its managed objects. Networks must increasingly support a growing range of traffic types. These traffic types require specific traffic engineering and QoS handling in layer 2 and layer 3 networks. The different types of QoS provide the means of imple-

menting the required QoS. While MPLS helps solve many problems, it also can suffer from scalability issues as the number of LSPs increases. Scalability can be improved by incorporating techniques such as change tables in the agent.