THE OPTICAL NETWORK CONTROL PLANE

Abdella Battou[*] Ghassen Ben Brahim[†] Bilal Khan[‡]

Abstract

Just a few years ago, the phrase "optical internet" meant little more than a few WDM fibers whose wavelengths were serving as a link layer to interconnect routers via add/drop multiplexors. By comparison, today multiwavelength reconfigurable optical switches are readily available, enabling for the first time, the development of large-scale high speed optical networks operating at rates in excess of 10 Gbps. In such networks, IP routers and ATM switches that are connected to an optical core can respond to changing bandwidth needs of applications by dynamically establishing lightpaths on-demand. Modern optical switches fall into one of two broad classes (i) those with opto-electronic fabrics and (ii) those which are all-optical. Opto-electronic switches have the advantage of being easier to manage, and offer attractive features such as sub-rate grooming and in-band signaling. In contrast, all-optical switches require more sophisticated monitoring and adaptive control mechanisms, but yield bit-rate independence and transparent access to the full raw capacity of the optical channel. The future success of both types of optical switches depends largely on their performance and ability to be scalably deployed. In this context, we describe the design of an optical network control plane, with particular emphasis on the difficult issues in optical network management, and lightpath signaling and routing.

Keywords: Optical networks, control plane, routing, signaling, management.

1 Network Management

Network management functions can be broadly partitioned into the two classes of monitoring and control.

1. *Network Monitoring.* Network monitoring is the term used to describe passive management functions which cause minimal interference in the network's state. The main goal of such is to collect data and analyze it to extract useful information about the status of network Typical data collected includes static information regarding

[*]Princeton Networks

[†]Princeton Networks

[‡]Advanced Engineering & Sciences ITT, Center for Computational Science at the Naval Research Laboratory, Washington D.C.

network configuration, as well as dynamic information about network events as they occur. An example of network monitoring management function would be the collection amortized performance statistics about a network element or subnetwork.

2. *Network Control.* Network control is the term used to describe active network functions which change the underlying state of the network. Some examples of these include the provisioning of connections, starting or shutting down individual network elements, components, etc.

A network management system is composed principally of two types of processes: (i) agents running on managed nodes, (ii) network managers collecting and processing information collected by agents. In large networks, management processes may be further stratified into a hierarchy. The managers use predefined management protocols such as SNMP to talk to the agents. The information that is exchanged is organized in a tree-like address space referred to as a Management Information Base (MIB).

Network management system architectures fall into two general categories:

1. *Centralized network management*—In this architecture, all manager processes live on a single machine, perhaps even in the same process. This type of management architecture has been implemented and used extensively, predominantly because of its simplicity.

2. *Distributed network management*—Managers reside on several machines distributed throughout the network, perhaps replicated and/or mobile. Although distributed management architectures have not been used extensively in practice because of the necessary complexity of their software implementations, they offer superior features like scalability, fault tolerance, and availability. We discuss these advantages briefly in what follows.

    - *Scalability.* Many NMS functions require considerable processing power. An example of sych a function is the aggregation required during upward propagation of information through hierarchies of network managers—this aggregation requires computation of compact representations of the composite state of lower-level managers. In addition, distributing network management across multiple machines permits event filtering and processing to be moved closer to the agents generating these events. This not only reduce latency of the management system, but also reduces the communication costs attributable to control traffic. A centralized NMS approach implicitly forces aggregation schemes and even reporting mechanisms to be simplisitic in order to avoid choking the system.
    - *Fault-Tolerance.* An NMS must be to be tolerant to failures, both of network components that it is managing *and* of individual managers/agents. Distributing the NMS across many machines minimizes the impact of network hardware failures and software instabilities in any particular set of agents/managers, by isolating the side-effects of such failures to some region of the distributed management system. A sophisticated distributed NMS must be able to survive

network partitions: it must permit independent control of the pieces of a partitioned network, while supporting re-integration of NMS functions once the partition heals.

- *Availability.* Distributing the NMS increases its availability, thereby permitting its services to be efficiently utilized by administrators and users located at any network node. At the same time, the latency of issuing such requests is reduced, while the load of handling the requests is favorably distributed across many machines.

Currently, networks maintain their management information in distributed in a management information base (MIB) maintained by each of the agents. Typically these MIBs are passive collections of data without smarts, and possess very unfriendly interfaces. A better design for an information base would use an object-oriented model where object attributes are recursively defined as the result of effective computations of several simple or composite attributes. Such an architecture facilitates network managers to dynamically define new composite templates customized to their individual needs, which can then be "filled in" and maintained by the NMS. Such composite templates might include aggregations which summarize detailed low-level attributes, or refinements of such data obtained by detailed computation, analysis and annotation. Composed objects should

1. be dynamically specifiable and be defined both spatially or temporally. By spatial attributes we mean quantities derived from collections of managed network objects, e.g. from all the nodes in a subnetwork, or all the subnetworks in a network. The NMS should permit these spatial groupings to be made dynamically. In contrast, temporal attributes refer to quantities derived from collections of values of low-level attributes over a specified time interval.
2. be persistent to avoid recomputation of their state,
3. provide secure access to their internal attributes,
4. provide an event filter installation interface (e.g. via Java MBean registration),
5. be able to handle event subscription and notification,
6. provide visualization through HTML and Java interfaces, and lastly, they should
7. use a standard naming convention to allow efficient distribution and discovery of management information using distributed directory or Jini services.

1.1 Distributed Network Management Architecture

Recently, distributed architectures for network management have left the realm of research laboratories and made their debut in commercial environments. This debut was largely facilitated by the impressive progress in the Java development community. Exemplary of this progress are the Entreprise JavaBeans (EJBs) which make it possible to use Java for building distributed 3-tier entreprise applications. Other examples include the Java Management Extensions (JMX) and JIRO, which are new Java technologies for network and service management [?], [?]. Here we shall describe a Distributed Network Management Architecture designed based on these technologies; such an architecture can be implemented easily in the Java language.

The agents are developed using the Java Management Extensions (JMX). An agent's intelligence is implemented in MBeans (Managed Beans) that are registered dynamically in an MBean server. These MBeans can generate alarms, notifications, as well as take actions for active management. An MBean represents one or more composed objects and may implement some preprocessing/filtering that normally is done by a manager–performing this preprocessing/filtering nearer at the source reduces the amount of management traffic. To be accessible, these MBeans have to be registered with the MBean server. This registration happens dynamically and can be done by the agent itself or by a manager. The MBean server and the MBeans constitute the agent. MBeans can be registered as either volatile or persistent within the MBean server.

To interface to these MBeans remotely, clients need a proxy object. These proxies or client Beans are generated using a special compiler `mogen`. Any operation performed locally on a proxy is propagated to the remote MBean. MBeans resolve concurrent access by multiple clients based on their own concurrency model; the simplest being strong sequential consistency via mutual exclusion of client operations.

Adapters are used by proxies when talking to MBeans. Adapters implement protocols to talk to the agent, and they are designed as MBeans so they can be dynamically downloaded. in this way, new protocols can be instantiated dynamically and registered on the agent on demand. Currently, the following adapters are available - RMI, HTTP, HTTP over SSL, IIOP, SNMP ,and HTML. For example, once the HTML adapter is installed on the agent, any application can point its web browser to the agent to view all MBeans on that agent. Here the HTML adapter generates automatically the HTML pages which are in turn rendered on the client's browser. A collection of adapters together with the proxies using them, together form a manager. A proxy may use multiple adapters simultaneously.

Interestingly, an agent can a register client bean and become a manager to another agent that is implementing the corresponding remote MBean from which the client bean is generated. This lifts the classical strict separation between agents and managers and gives us a more symmetric, aesthetically pleasing model. The benefit of such an architecture is that there is no need to change the agent to allow a new application to talk to it. All that is needed is to register a new adapter for that application, the remaining agent code stays unchanged.

Serveral services are supported to simplify agent and manager development. Below is a partial list of the services we provide, and a brief description of each:

1. **Repository service**:
   Allows MBeans to be registered in an MBean server, and permits them to be specified as either volatile or persistent.

2. **MetaData service**:
   Allows determination of MBean properties and methods, by using the Java Reflection API.

3. **Filtering service**:
   Allows attaching filters to specific subset of MBeans in the MBean server. The sets of MBeans could be all registered MBeans for example, or only those that satisfy a certain rules specified in terms of methods, attributes, and object names.

4. **Discovery service**:
   Allows discovery of other MBean servers. This is achieved by ensuring that all active MBean servers that have registered the **Discovery Responder** will acknowledge an IP broadcast of a special "Who is out there?" message.

5. **Management Applet (MLet) service**:
   This is the service that downloads and configures MBeans dynamically. The Applet loads an HTML page that specifies all the necessary information about the MBean. It then downloads the implementation of the MBean (Java classes), and finally registers the MBean in the MBean server.

1.2 Directory service

The directory service is a lightweight database to facilitate the management and configuration of agents and managers. This directory could be designed as a branch in the organization LDAP server or built dynamically as a distributed directory service by agents and managers.

1.2.1 LDAP server approach

This approach is standard and very easy to implement. It requires laying the configuration data using the LDAP Data Interchange Format (LDIF) in the LDAP directory server. LDIF is a text-based format to display entry information using mnemonic named attributes and text values. A text-based format makes writing tools to process or create entries straight forward. Commercial, as well as public domain (openLDAP) LDAP servers are available [**?**].

1.2.2 Distributed approach

The LDAP approach has the advantage of being simple but has the drawback of being a single point of failure. A distributed approach where local directories are dynamically combined to form a global directory is a more robust alternative.

1. **Local directory**: Each Agent and Manager maintains a local directory tree which is part of the distributed directory. It contains the following information:

- Agent/Manager configuration and the location of the current distributed directory root node.
- Capabilities of the Agent/Manager
- MBeans installed at this Agent/Manager and their capabilities

The local directory can be implemented either as an MBean or as a Jini service. We describe the case where it is implemented as an MBean.

Local directories combine to form the distributed directory. The manager holding the root node of the tree is called the Root Manager. The tree is created by an algorithm that is capable of handling network partitioning and failures. An election algorithm based on priorities is started to elect the root node.

Each manager is initialy configured with a list of managers that can become Root Managers, each with a priority. On start-up a root node is created in the local manager and the local directory is registered with it. This way, MBeans are instantaneously provided with a working distributed directory service allowing locally scoped MBeans to start executing successfully.

After this minimal setup, then Manager A will poll its list of root configured managers until a running and reachable manager B is reached. If manager B is not a Root, then A is redirected to B's Root. If B claims to be the Root and its priority is higher or equal to A's then B becomes the Root Manager for A. Otherwise A becomes the Root manager for B.

To attach to a new root, a manager registers its local directory with the root. From then on, the root node created locally acts as a proxy to the actual root node. This process is repeated and terminates when the root node created is the one with the highest priority.

As each manager polls its root manager at regular intervals to check for its availability, any changes to the root of the distributed directory are then passed down to the managers. If the root directory fails or a network failure causes the network to partition the root election algorithm is run again. Each manager that lost contact with the root elects itself as root and later merges with the other managers creating a distributed directory in the partition. During uptime each manager builds a list of discovered managers in addition to its configured list of root nodes. This list is used when the list of root nodes is unreachable.

1.3 Managers

Managers form the middle-tier of the distributed network management architecture, agents being the lower and client applications the upper. Managers present two facades - the functional and the management. These are sometimes called the data path and the control path or service functionality and service administration. The manager architecture should offer basic services in both facades. These basic services facilitate and speed the development of new managers. Synchronization and transaction support, security, and persistence should be part of the basic services, as well as deployment tools. JIRO for example, provides additional basic services such as a distributed publish/subscribe event service, a logging service, and a task scheduling service. The hierarchy of managers is dynamically built from configuration information in the LDAP server. We describe 2 types of managers - Topology and Connection in details and only mention the Fault Manager.

1.3.1 Topology Manager

The Topology Manager defines a set of topology objects and the operations clients use to build network topologies. It also defines some semantic rules for understanding network objects behaviors such as containment rules. The model used should be simple enough to allow easy integration with the other managers and generic enough to allow cross-domain integration. Examples of such models are ITU-T G.805 and ITU-T M.3100. Here we will borrow from the ITU-T G.805 and build a simpler model. The model uses four types of objects to represent the physical and logical structures in a network and build a hierarchical view of a network (hypergraph).

- network
- subnetwork
- link
- link termination point

Subnetworks are recursive and termination points are not shared between networks. Each subnetwork/network has two view : an internal view (shown as dotted oval in Figure 1) and an external view. The external view publishes the subnetwork access points to its super network. Figure 1 show network1 with six access points for setting up end-to-end optical trails. Internally, Network1 contains two linked subnetworks - Network2 and Network3. So the external terminations of Network1 must be directly mapped to the external terminations of Network2 and Network3. Both Network2 and Network3 have four external terminations advertized to Network1. One of them is used as a link termination between them by Network1. The network is built recursively and stored in the configuration file at the root manager in the distributed directory. Using a discovery protocol, managers register for topology events which are generated by agents. Managers can aggregate these events into a subnet graph which they send to their parents. Finally at the root manager we get the full view of the network. To get detailed views of the network, a client can navigate the directory service or can request a flat view of a subnetwork or the full network. Topology managers at different levels are either elected or configured through the LDAP server. Having them all run on the same switch makes the data exchange fast, but leads to a load balancing problem. The usefulness of this approach is its ability to model any technology as a subnetwork whose services are made avalaible through its termination points. Mapping the subnetwork termination points at the network level provides subnetwork/technology abstractions and a software approach to integrate different technologies and offer services seamlessly on top of all them.

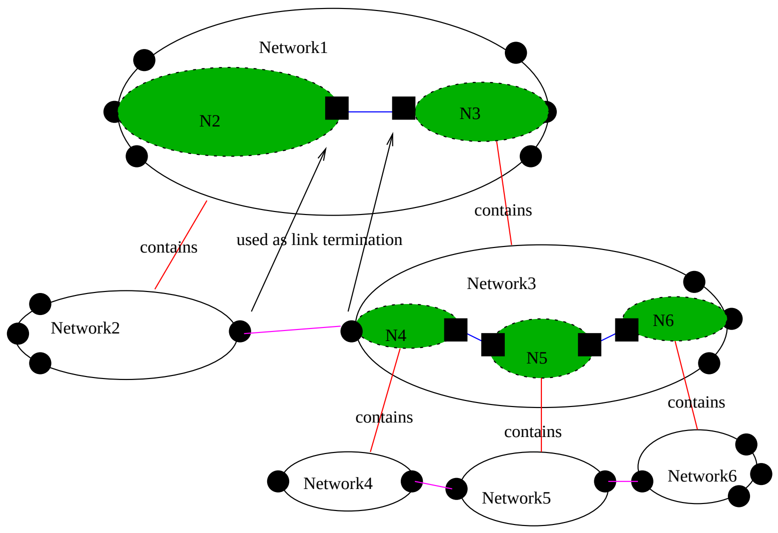


Figure 1: Network Hierarchy

1.3.2 Connection Manager

The goal of the connection manager is to manage connections across multiple domains. As in the topology manager, the connection manager defines objects to model connection services and semantic rules for the behavior of these objects. Some of these rules include

- **partition**: as in the network layer is partitioned into a set of subnetworks.
- **contain** : as in a network contains many trails or a link termination point contains many network connection termination points (NCTP) one per wavelength for a DWDM network.
- **delimit** : as in a subnetwork is delimited by a set of network connection termination points.
- **bind**: as in a link connection is bound to 2 network connection termination points (NCTP).

The goal, same as in the topology manager, is for these objects to be generic enough to offer integration across multiple technologies. Connections have to have service information such as protocol (WDM optical path, ATM SVC/PVC, Frame Relay, IP, etc), quality of service (QoS), and bandwidth. The connection model is built on top of the topology model see Figure 2 for some of the relationships between topology manager objects and connection manager objects.

The key design criteria here is to ascertain that both,the connection model, as well as the topology model are generic enough to support the service management functions.

So here the concept of connection is abstract and supports the end-to-end view of the network toplogy irrespective of the individual network technologies. This concept is valid in ATM networks, WDM networks, and Ethernet networks. The major benefit from such a design is that it provides a simple view of the network functions to integrate services in a global and scalable network. Services do not require detailed knowledge of the underlying technology they ride. In addition, it provides the integration point for multiple technologies.

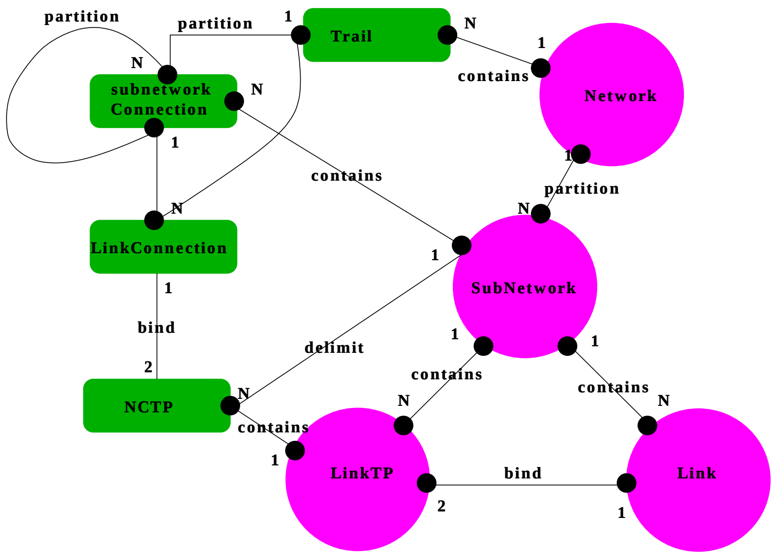


Figure 2: Connection manager objects and relationships with topology manager

The connection manager defines the following objects

- Link Connection (LC): resource that transfers data across a link between two subnets - a wavelength channel in WDM and an SVC in ATM.
- Connection Termination Point (CTP): end point of a subnet connection or link connection.
- Subnetwork Connection(SC): resource that transfers data between two CTP located at the edge of a subnet. The degenerate case is a cross-connect in the switch fabric.
- Trail Termination Point (TTP): access point for clients.
- Trail: resource that transfers client information between two TTPs.

to store the state of the network from connections point of view. This state is shared by all network management components icluding applications and must be kept consistent. The connection manager objects are dynamic, they are created and deleted by users dynamically. Figure 3 shows a point-to-point connection and a point-to-multipoint connection at the network level, while Figure 4 shows an example of an end-to-end trail across two subnets and showing all the objects involved.

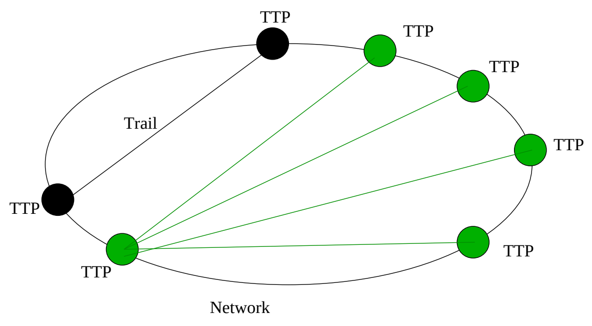


Figure 3: Network, Trails, and Trail termination points

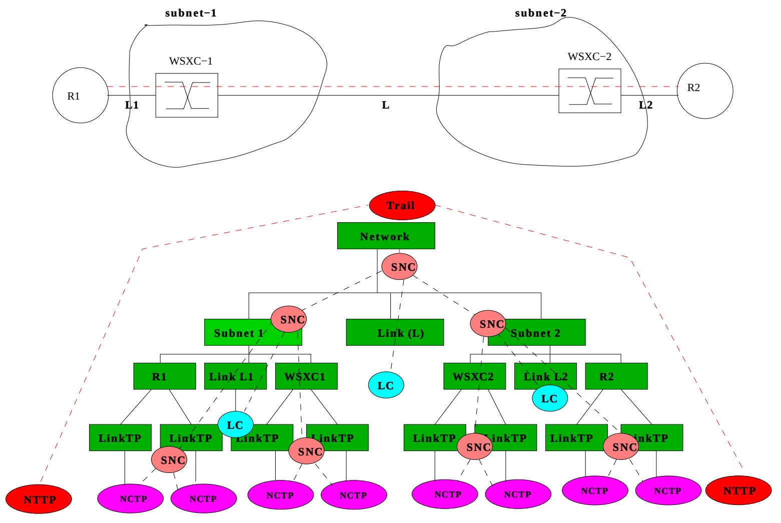


Figure 4: Trail setup between routers across 2 WSXCs

A connection/trail object represents the service provided by the connection manager. A network management station is used to add, delete, find, and navigate through these trails. They are two ways to setup a trail:

- smart setup - uses the signaling protocol:
  Here the connection manager uses the signaling API to setup a trail between source A and destination B.

- manual setup - uses the network management:
  The connection manager either interfaces with the routing protocol to get a route from source A to destination B, or uses the topology manager to get the route. Once

a route is found it recursively sends setups through the subnets until it reaches the network elements. The connection is setup as a transaction.

Routing protocols used to select routes are discussed below.

1.3.3 Fault Manager

The goals of the fault manager is to precisely identify the fault, and automatically restore services. Faults range from fiber cuts, to hardware failures, software bugs, procedural errors, and security breaks. Fault management is distributed across the network to bring its intelligence closer to the network element to get real-time restoration. A subnetwork fault manager can collect all alarms, do some processing/filtering and only report higher level errors to network managers. One of the hardest parts is the expert engine/tool for alarm correlation and fault identification. This engine takes the state of the network as input and it diagnoses problems and provides corrective actions. Its actions come from inference rules on the knowledge base composed of human expertise and domain knowledge based largely on previous experiences. Real-time expert systems are still an active area of reseach in Distributed Artificial Intelligence field. Their goal is to provide a framework for autonomous expert systems to cooperatively work to allow expertise sharing and load balancing. This framework includes a language for communication, a negotiation strategy, and efficient ways to represent and exchange domain expertise. Such a framework should guarantee the convergence of the negotiations algorithm with the ultimate goal of designing autonomous networks. The fault manager has to work in concert with the other managers. Topology related failures are forwarded to the topology managers which translate them into graphical events in the network maps at different levels. The difficulty is to correlate same layer and inter-layer faults. In addition, one needs to translate topology, connection, and performance faults into service faults that are meaningful to the user.

2 Signaling in an optical network

The goal of a signaling protocol is to dynamically setup unidirectional or bidirectional optical paths with minimum latency to transport data across an all-optical network. These paths remain active for an arbitrary amount of time, and are not re-established after a network failure. On the other side, the permanent optical paths, which are provisioned paths are destined to remain established for long periods of time. In addition, provisioned paths are persistent, so they are re-established after network failures. This section, describes dynamic optical path setup, while the network management section, describes the provisioning of permanent optical paths.

It is assumed that the optical path characterized by the dedicated wavelength 1510, is used for all signaling. It is available on every fiber, and is extracted on the input port at every switch and converted to electronics so that the switch controller can act on the signaling messages, and re-inserted back on the output port and converted to optical. it also assumes that this protocol is symmetric with respect to user to network interface (UNI) and network to network interface (NNI). On the user/client side we expect a router or an ATM switch, and on the network side or server side we expect an optical switch. First, a session is established between the user and the network to ascertain reliable and in order delivery of management and signaling messages. TCP port 65000 from the experimental port number pool 49152-65535 is used (see http://www.iana.org/). Session messages are used to register and deregister client addresses. A helloTimer is kept on both side to detect failures and session resets. We define four states - sessionReleased, awaitingEstablish, sessionEstablished, and awaitingReleased. The following session messages are exchanged between client and server : sessionSetup, sessionSetupConfirm, sessionRelease, sessionReleaseConfirm. SDL diagrams of the session FSM are shown in Figures 5, 6,7,and 8.

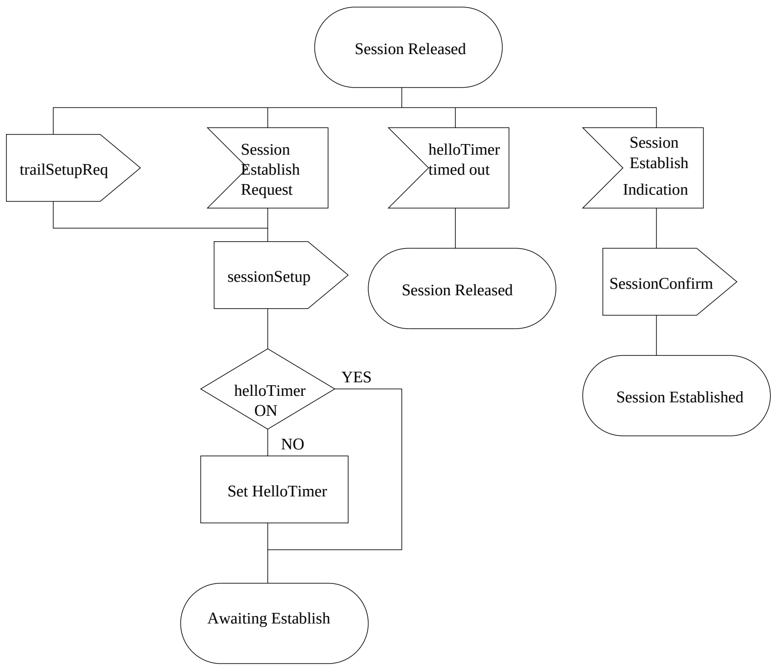


Figure 5: starting in session release state

2.1 The Simple Signaling Protocol (SSP)

In this section, we describe a simple signaling protocol for optical networks. SSP is inspired from the ATM Forum UNI signaling protocol and it defines five types of messages:

- **TRAIL SETUP**: Used to request a trail establishment

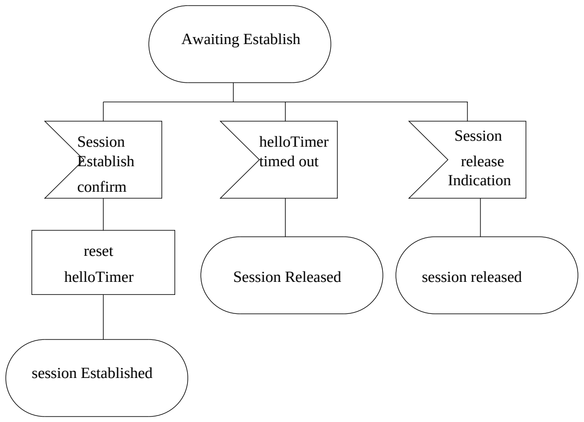


Figure 6: awaiting establish state

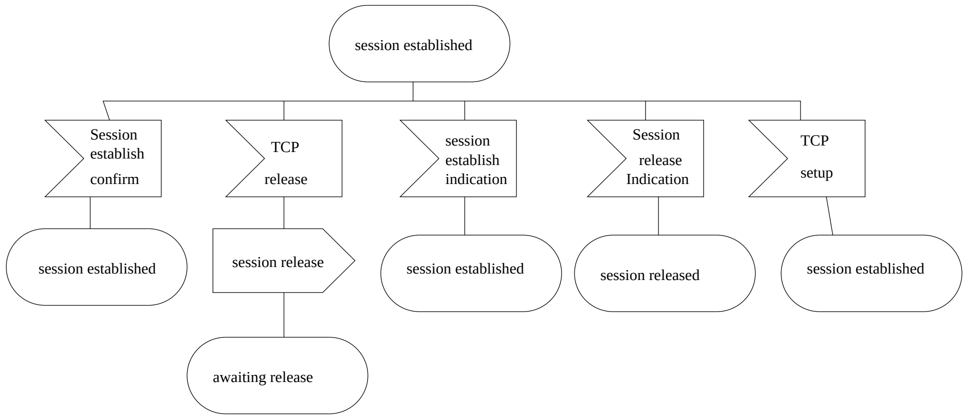


Figure 7: established state

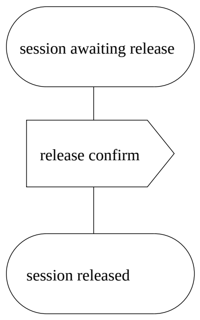


Figure 8: awaiting release state

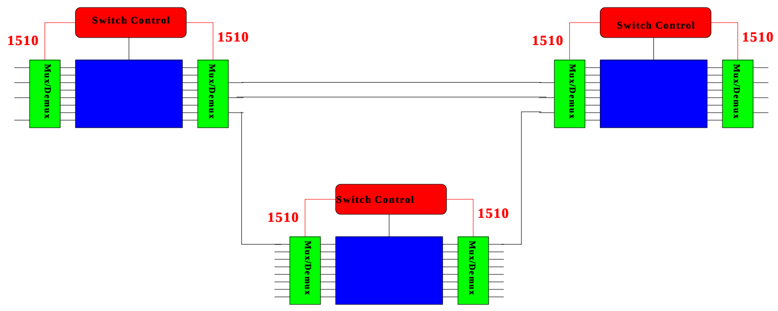


Figure 9:

- **TRAIL PROCEEDING**: Used by the Network to acknowledge the receipt of a SETUP message from the client.
- **TRAIL READY**: Generated by the receiver when it accepts a trail and it is forwarded back to the originator of the SETUP message by every switch on the way to it.
- **TRAIL READY ACK**: Generated by the network to signal that the trail is ready.
- **TRAIL RELEASE**: Generated by either the source, the destination or any intermediate switch in the trail path when it decides to teardown the trail
- **TRAIL RELEASE COMPLETE**: Generated to acknowledge the receipt of a TRAIL RELEASE message.

Every message is composed of a header and a body. The header (10) is with fixed length and includes:

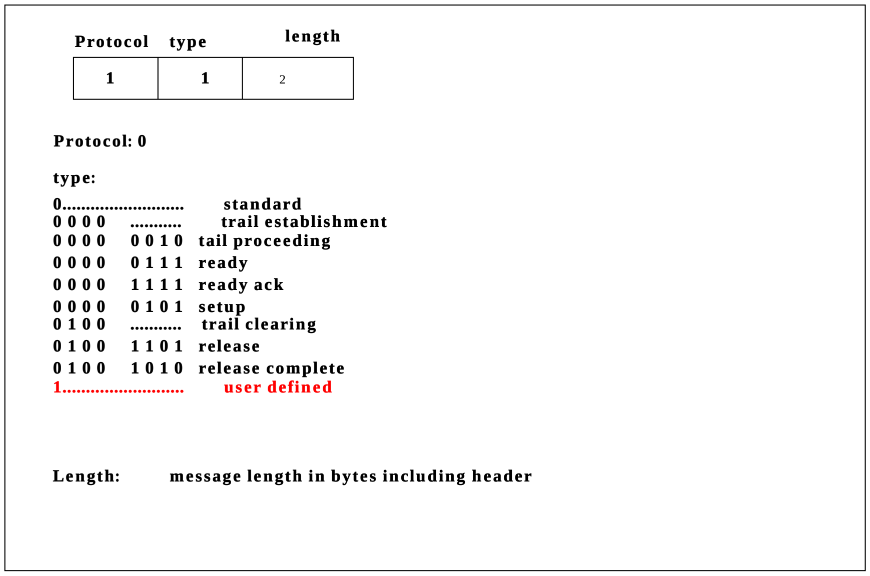
- protocol version number.
- message type.
- message length in bytes of the whole message including the header.

The body of each message has a variable length and is composed of information elements. An information element has a type, a length in bytes, and a value.

Information elements can be one of two types: either required information elements or user-defined information elements, which are silently ignored by any implementation that does not understand them.

SSP defines five information elements:

- End Point IE.



Protocol	type	length
1	1	2

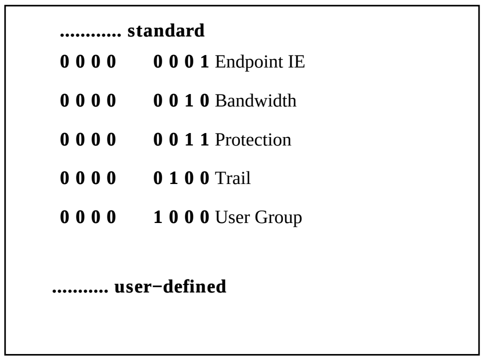
Protocol: 0

type:

0.........................		**standard**
0 0 0 0	**..........**	**trail establishment**
0 0 0 0	**0 0 1 0**	**tail proceeding**
0 0 0 0	**0 1 1 1**	**ready**
0 0 0 0	**1 1 1 1**	**ready ack**
0 0 0 0	**0 1 0 1**	**setup**
0 1 0 0	**..........**	**trail clearing**
0 1 0 0	**1 1 0 1**	**release**
0 1 0 0	**1 0 1 0**	**release complete**
1.........................		**user defined**

Length: **message length in bytes including header**

Figure 10: Message Header Format

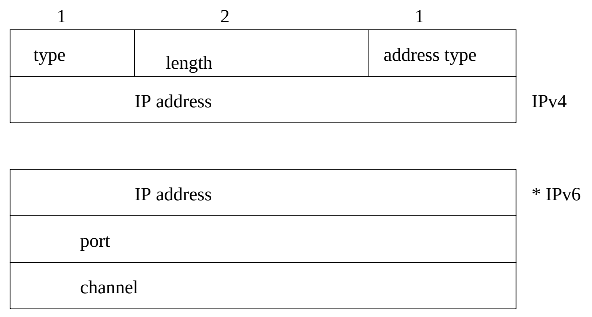


.... standard

0 0 0 0	**0 0 0 1** Endpoint IE
0 0 0 0	**0 0 1 0** Bandwidth
0 0 0 0	**0 0 1 1** Protection
0 0 0 0	**0 1 0 0** Trail
0 0 0 0	**1 0 0 0** User Group

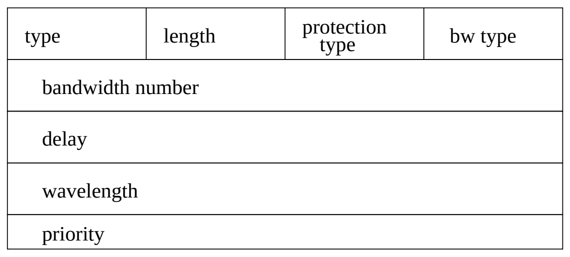
.... user−defined

Figure 11: Information element format



address type: 0 for IPv4 and 1 for IPv6

Figure 12: End point Information element



protection type:

0x01	unprotected
0x02	1+ 1 protection
0x03	auto−reroute

bandwidth type:

0x0001	transparent
0x0002	OC768
0x0003	OC192
0x0004	OC48
0x0005	OC12
0x0006	OC3
0x0007	STS−1
0x0008	Etrhernet

Figure 13: QoS Information element

- QoS IE.
- Trail IE

type	length	unused	address type
IPv4			

IPv6
id
wavelength

Figure 14: End point Information element

- User Group IE.

type	length	unused
VPN OUI		
VPN INDEX		

Figure 15: User group Information element

- Cause IE.

- **TRAIL SETUP**: The trail setup message includes the following information elements:
  1. caller endpoint.
  2. callee endpoint.
  3. QoS.
- **TRAIL PROCEEDING**:
  1. Trail identification.
- **TRAIL READY**:
  1. Trail identification.
- **TRAIL READY ACK**
- **TRAIL RELEASE**:
  1. Trail identification.
  2. Cause.
- **TRAIL RELEASE COMPLETE**:
  1. Cause.

A source client uses a timer after sending its SETUP message. This timer is armed to wait for the TRAIL PROCEEDING message or the TRAIL READY message. If either timer expires before the receipt of the expected message, a TRAIL RELEASE message is sent.

SSP assumes that the routing protocol is able to return a path to the destination, as well as an available wavelength on that path if none is specified. If a wavelength is specified when a route is requested, the returned wavelength must match the requested one. If the network cannot allocate a path with the requested wavelength it returns a TRAIL RELEASE COMPLETE back to the client. SSP defines the following states:

- *NULL*: no path exists.
- *READY*: path is ready.
- *SETUP_SENT*: this state exists when a **TRAIL SETUP** message has left the user.
- *SETUP_RCVD*: this state exists when the user receives a **TRAIL SETUP** message.
- *TRAILPROC_RCVD*: this state exists when the user has received a **TRAIL PROCEEDING** message after sending a **TRAIL SETUP**.

- *TRAILPROC_SENT*: this state exists when the user has acknowledged the receipt of a **SETUP** message by sending a **CALL PROCEEDING** message.
- *READY_SENT*: this state exists for an incoming path when the user has sent the **TRAIL READY** message but has not received the **TRAIL READY ACK** message from the network.
- *READY_RCVD*: This state exists when a **TRAIL READY** message is received but a **TRAIL READY ACK** has not been sent yet.
- *RELEASE_SENT*: This state exists after a **TRAIL RELEASE** message is sent and before **TRAIL RELEASE COMPLETE** is received.
- *RELEASE_RCVD*: This state exists after a **TRAIL RELEASE** message is received and before **TRAIL RELEASE COMPLETE** is sent.

Figure 16 represents the path establishments state diagram scenario. Figure 17 represents the path release state diagram scenario.

2.1.1 Trail Establishment Request

A new trail establishment is initiated by transferring a **TRAIL SETUP** message on wavelength 1510 across the signaling interface. The client starts the **setupTimer** and moves to *SETUP_SENT* state. If the client does not get a response before this timer expires for the first time, the **TRAIL SETUP** message is retransmitted and timer is restarted. If the client does not receive a response to the **TRAIL SETUP** after the **setupTimer** timer has expired for the second time , the client clears the trail establishment internally.

2.1.2 Trail Proceeding

If the network can satisfy the request for a route to a specific destination with requested wavelength, or for a route on any of the available wavelength, then the switch sends a **TRAIL PROCEEDING** message to the client to acknowledge the receipt of the **TRAIL SETUP** message and moves to *TRAILPROC_SENT* state and forwards the **TRAIL SETUP** message to the next hop towards the destination. When it gets a **TRAIL READY** message from this next hop it will forward it to the client and move to *READY* state. When the client gets this **TRAIL PROCEEDING** message, it stops the **setupTimer** and starts the **readyTimer** and moves to *TRAILPROC_RCVD*. If the **readyTimer** expires before receiving the **TRAIL READY** message, the client initiates a trail clearing procedure towards the network.

2.1.3 Trail Acceptance

When the network gets the **TRAIL READY** message, it moves to *READY* state, then forwards this message to the client. When the client gets the **TRAIL READY** message, it stops either **setupTimer** or **readyTimer** in case they are still running. Then, it sends a **TRAIL READY ACK** message to the network, and moves to *READY* state. At this stage, an end-to-end trail is established and data can be transfered.

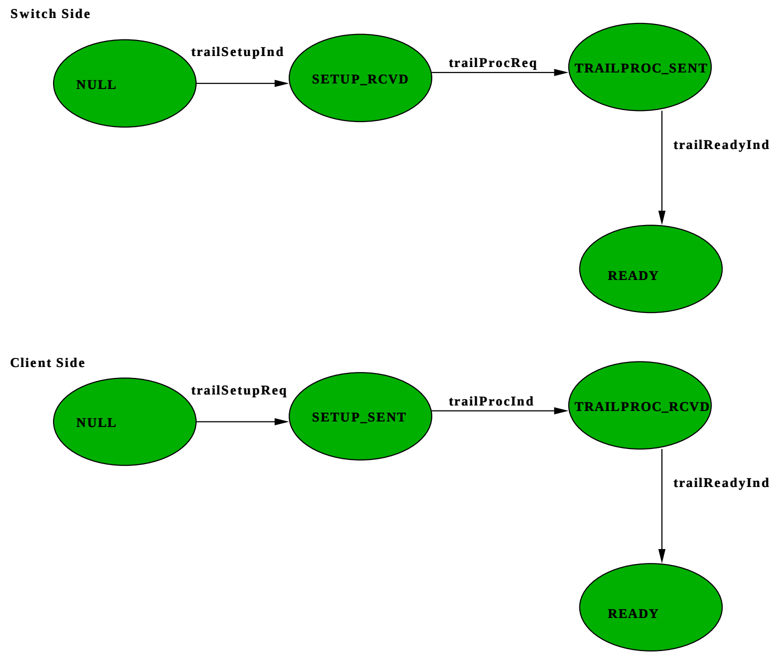


Figure 16: Path Establishment State Diagram

2.1.4 Trail Rejection

There are two places when a trail rejection can occur - in the network or at the destination client. The rejection occurs in the network if a switch can not proceed the trail because either, there is no route to destination, or there is not enough resources (wavelength). However, if there is no application to accept the trail or again there is a lack of resources, the rejection occurs at the destination. In both cases the network initiates a trail clearing towards the client originator.

2.1.5 Trail Clearing

The **TRAIL RELEASE** message is sent by the client to request the optical network to clear an end-to-end trail. It is also sent by the network to notify a client to start

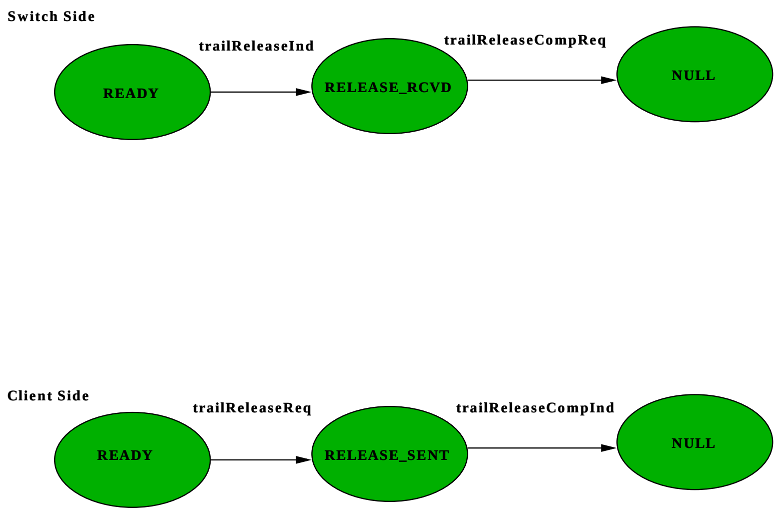


Figure 17: Path Release State Diagram

clearing a trail. The client can release it trail control block and trail identification after send a **TRAIL RELEASE COMPLETE** message. The **TRAIL RELEASE** message is global - it travels end-to-end. So the **TRAIL RELEASE COMPLETE** is sent by either end to notify the other that it has released all resources regarding this trail and so they can be reused. The **TRAIL RELEASE COMPLETE** message has only local significance.

Normally, a trail is cleared when the client or network sends a **TRAIL RELEASE** message. The only exception to this rule is when the client or the network rejects the trail for a lack of resources or incompatibilities and responds with **TRAIL RELEASE COMPLETE** message to a **TRAIL SETUP** message. In normal cases, the client arms the **releaseTimer** after sending the **TRAIL RELEASE** message and moves to *RELEASE_SENT* state. The network moves to *RELEASE_RCVD* state once it sees the **TRAIL RELEASE** message, starts the clearing procedure towards the destination by forwarding **TRAIL RELEASE** to the next hop towards the destination, then it frees the trail control block and responds with a **TRAIL RELEASE COMPLETE** message to the client. Upon receiving the **TRAIL RELEASE COMPLETE** message the client stops the **releaseTimer** and moves to the *NULL* state. If the **releaseTimer** expires before hearing from the network, for the first time, the client resends **TRAIL RELEASE** message, re-arms the **releaseTimer** and stays in the *RELEASE_SENT* state. If however, the **releaseTimer** expires before hearing from the network for the second time, then the client frees the trail control block and goes to the *NULL* state.

A collision occurs when both sides, client and network transfer a **TRAIL RELEASE** message at the same time. If the client receives a **TRAIL RELEASE** message when in *RELEASE_SENT* state, it stops **releaseTimer**, frees the trail control block and moves to *NULL* state.

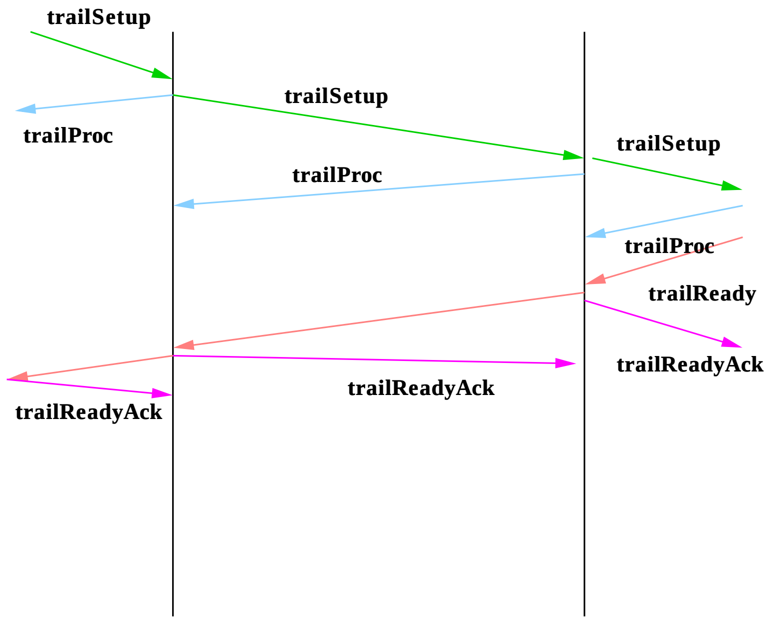


Figure 18: Trail Establishment Timing Diagram

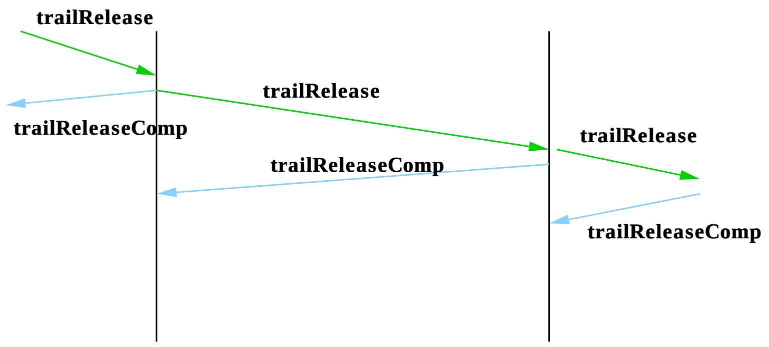


Figure 19: Trail Release Timing Diagram

Figure 18 represents the trail establishment timimg diagram scenario. Figure 19 represents the trail release timing diagram scenario.

3 Routing in Optical Networks

The central algorithmic problem in WDM routing is: Given a WDM network's physical topology (including the characteristics of its links) and a set of source-destination pairs, (i) *route assignment:* compute a route for a lightpath between each source-destination pair, and (ii) *wavelength assignment:* for each link traversed by this lightpath, determine the wavelength to be allocated for the lightpath on the given link. Together, these two assignment problems are often referred to as the *Routing and Wavelength Assignment (RWA)* problem. Typically, the RWA problem is solved by considering each of its two constituent subproblems in turn [22].

The RWA problem is complicated by the fact that an OXC switch may be (optionally) equipped with *wavelength conversion* hardware which permits lightpaths transient through the switch to enter and leave the switch on different wavelengths. Consider figure 20, which illustrates a WDM network consisting of five OXC switches interconnected

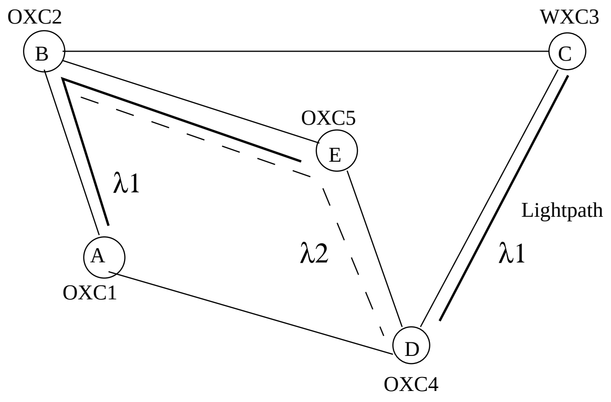


Figure 20: A WDM Routing Network with 5 OXCs

by optical fiber links. If no switches are equipped with wavelength converters, then a lightpath in this network must always occupy the same wavelength on every fiber link it traverses. This restriction is commonly known as the *wavelength continuity constraint* for non-wavelength converting switches. An OXC switch that is equipped with wavelength conversion hardware is exempt from this constraint. The reader is referred to the book by Ramaswami and Sivarajan [20] for a comprehensive introduction to WDM network technologies.

A solution to an instance of the RWA problem must necessarily [1], [2] respect the following constraints: First, two lightpaths traveling on a given link must be assigned different wavelengths. Second, any lightpath that transits through an OXC switch that is not wavelength conversion capable, must use the same ingress and egress wavelengths. The central responsibility of the WDM routing protocol is to ensure that the information required for route assignment and wavelength assignment is maintained by the network in a scalable manner.

1. **Routing Assignment**

   There are three broad classes of strategies being used presently to address the routing assignment problem [22]. These strategies are referred to as *fixed routing, fixed alternate routing, and adaptive routing.*

   Fixed routing is a simple technique which involves maintaining a fixed routing table at each candidate source node. The routing table consists of one entry for each candidate destination node, where the entry specifies the path from the source to the destination. Fixed routing is simple to implement, but is subject to unacceptably high blocking probabilities as wavelength availability on links becomes scarce, and when link failures occur.

   In contrast, *fixed alternate routing* attempts to address the shortcomings of fixed routing by augmenting each entry in the routing table to be a prioritized set of paths from source to destination, rather than just a single one. By doing so, fixed alternate routing is less sensitive to link failures and wavelength and thus offers lower connection blocking probabilities.

   Finally, *adaptive routing* schemes attempt to select a path between a source and destination pair based on dynamically collected information concerning the network's state. This technique is much more resilient to link failures and less sensitive to wavelength scarcity, thus offers lower connection blocking probability than fixed adaptive routing approaches.

2. **Wavelength Assignment**

   The second half of the RWA problem, wavelength assignment, is addressed algorithmically using strategies based either on heuristics, or on *graph coloring* algorithms. Presently considered heuristics include Random Wavelength Assignment, First-Fit, Least-Used (SPREAD), Most-Used (PACK). Min-product, Least Loaded, MAX-SUM, Relative Capacity Loss, Distributed Relative Capacity Loss, Wavelength Reservation, and Protecting Threshold. For a detailed comparison between the performance of these heuristics, the reader is referred to the review by Zang et al. in [22].

3. **RWA and Traffic Engineering**

   One popular extension of adaptive routing schemes are the so-called *least congested path* (LCP) schemes, which attempt to choose the path between source-destination pairs based on current traffic patterns in the network. By considering each wavelength on a link to be a fraction of the link's available bandwidth, LCP schemes are easily adapted to operate in a WDM environment. Unfortunately, LCP schemes introduce significant complexity, both in terms of the protocols that must operate to maintain information about network state, and the algorithms responsible for carrying out the route computation. Together, these concerns and their resolution are referred to as *Traffic Engineering (TE).*

TE is responsible for mapping traffic flows to the managed physical network resources. Specifically, TE provides the ability to move traffic flows away from the shortest path and into less congested paths in order to satisfy the *Quality of Service (QoS)* requirements of higher layers. The main purpose of TE is to load balance the network traffic across various links, routers, and switches, thereby minimizing congestion and ensuring that the network resources are not over or under-utilized. Regrettably, many existing routing protocols do provide QoS and TE adequately to model WDM networks, and so require augmenting the protocol and packet specifications.

Major classes of TE attributes are described by Awduche et al. in [6]. These include: (1) *traffic parameter attributes* used to capture the characteristics of the traffic streams, (2) *generic path selection and maintenance attributes* specifying rules for selecting the route taken by an incoming traffic as well as the rules for maintenance of paths that have already been established, (3) *priority attributes* specifying relative importance of different traffic flows, (4) *preemption attributes* stating rules for preemption of resources, (5) *resilience attributes* specifying the behavior of certain traffic under fault conditions, (6) *policing attributes* defining the actions that should be taken by the underlying protocols when incoming traffic becomes non-compliant. In [11], Fedyk and Ghanwani present additional metrics and resource information for links tailored specifically for traffic engineering, including hop count metrics, bandwidth based metrics, delay based metrics, economic cost or expense based metrics, and administrative weight.

To summarize, the responsibility of an optical routing protocol is to facilitate lightpath establishment between OXC switches, i.e. it must provide access to sufficient information to solve the problems of Routing Assignment and Wavelength Assignment, while permitting TE information concerning the current network traffic and the QoS requirements of higher layers to be taken into consideration.

3.1 Architectural models for Optical Routing Information Exchange

In this section, we examine how to carry IP over optical networks. We assume that IP routers are attached to an optical core network and are connected to other IP routers over dynamically established optical paths. In order to establish these optical paths, routers and optical switches need to exchanges routing information. Two architectural models are currently under consideration - Overlay model and Peer model:

1. **Overlay Model**

   In this model the optical network is considered quasi-static and optical paths are viewed as fibers between routers. This approach hides all of the optical network complexities from routers. The optical network runs a separate routing entity and one ends up with two routing systems. This is the model chosen for IP over ATM where PNNI is the routing system in the ATM network. Standard interfaces are defined between the routers and the optical switches - *Optical User Network Interface (OUNI)* and between the optical switches - *Optical Network to Network Interface(ONNI)*. Both the *Optical Domain Service Interconnect (ODSI)* and the *Optical Internetworking Forum (OIF)* have chosen this model and are in the early stage of standard development.

  * *Manual configuration:* Each router attached to the optical network is manually configured with optical endpoint addresses corresponding to IP destinations behind it.
  * *Reachability protocol across OUNI:* Each router attached to the optical network runs a reachability protocol across the OUNI with its corresponding optical switch to obtain IP addresses of all routers attached to the optical network. This information is then used to build the initial set of IP routing adjacencies which run an IP routing protocol to acquire all IP destination addresses reachable over the optical network.

2. **Peer Model**

   In this model the IP router is entirely aware of the details of the underlying optical network and can compute end-to-end paths across the optical network. The optical switch to which the IP router is attached to acts just like any other router thus the usage of the term "peer". Optical switch controllers will run an updated version of OSPF which is the standard routing protocol in the Internet and will exchange reachability information with routers. IETF is presently considering this model, because of the issues encountered in the IP over ATM experience.

  * *Routing protocol across OUNI:* Each router and its associated optical switch run an IP routing protocol allowing them to exchange full reachability information across the OUNI.

Many routing protocols already in existence may be adapted to provide adaptive routing in WDM. In the next section we evaluate some of these protocols.

3.2 Candidate Adaptive Routing Protocols for WDM Networks

Routing protocols can be divided into two broad classes, *Interior Gateway Protocols (IGPs)*, and *Exterior Gateway Protocols (EGPs)*. IGPs are intended for use within an *Autonomous System* , i.e. a set of networks that are under a single or closely coordinated management organizations. Routing between different autonomous systems is maintained by the EGPs, of which the *Border Gateway protocols (BGPs)* are one example. While IGPs are designed to dynamically maintain information about network topology, produce optimal routes, and respond to changes quickly, EGPs emphasize stability and administrative control above route optimality. As the networking community moves towards further decentralization of control, EGPs are viewed as serving to protect one system of networks against errors or intentional misrepresentation by other systems.

Along a different axis, routing protocols can also be classified into: *distance vector routing* protocols which implement distributed algorithm to compute all-pairs of shortest paths, and *link state routing* protocols which propagate information about network topology to all routers.

In distance vector routing protocols, such as the *Routing Information Protocol (RIP)*, each router maintains the distance from itself to each possible destination, and the distance vector routing protocols relaxes these values down to their shortest value. In doing so, RIP is essentially a distributed implementation of the Bellman-Ford all-pairs shortest path algorithm.

In link state routing protocols, such as the *Open Shortest Path First (OSPF)* protocol, every router maintains a representation of the network's topology, and this representation is updated regularly on the basis of the information being exchanged between the network routers. In particular, each router is responsible for discovering its neighboring routers, and then advertising the identities of its adjacent neighbors and the "cost" to reach each. This information is advertised between routers by periodic exchange of link state packets. A link state protocol arms each router with a dynamic map of the network's topology; using this, the router may compute paths to any destination.

In the subsections that follow, we evaluate several candidate routing protocols and assess their suitability for operation in the WDM environment.

3.2.1 Routing Information Protocol (RIP)

The Routing Information Protocol (see [13, 14, 19]) is a distance vector routing protocol for small networks. Each RIP-enabled router is classified as either active or passive: active routers advertise their routes to other routers; passive routers listen and update their routes based on the advertisements they receive but do not advertise any information themselves. Typically, routers run RIP in active mode, while hosts use passive mode. RIP is attractive because of its simplicity and elegance but suffers from several undesirable features, which we list below:

- *Network size limitations:* RIP was designed as a routing protocol for small networks of diameter $\leqslant 15$. The convergence time of RIP limits its applicability for larger networks.

- *Slow convergence:* The essential reasons behind the slow convergence are (1) the information that RIP needs to propagate is itself routed using the same routing tables that RIP strives to build, and (2) the accuracy of these routing tables increases only gradually because of the nature of distributed Bellman-Ford.

- *Count to infinity:* The failure of a link or node failure may result in two routers continually exchanging update vectors because each is claiming to have the right information. Many solutions (e.g. “Split Horizon”, “Flush Updates”, “Poison Reverse Update”, etc.) have been proposed to address this issue. But count to infinity may still occur even when using these techniques.

* *Looping:* Incorrect routing information available in some nodes may result in forwarding loops within the network.
* *Restricted view of the network:* A RIP router does not have a view of the topology of the entire network, and so cannot detect problems such as looping or count to infinity.
* *Limitations on the metrics that can be used for shortest path computation:* RIP attempts to determine the shortest path by relaxing the count of intermediate hops between each source-destination pair. It is unclear how to extend RIP to support computation of the “best path” using multiple interdependent (particularly non-additive) link metrics, such as those required by TE.
* *Inability to support load balancing:* Occasionally the need arises to split traffic load between two parallel paths between two nodes, or to cache alternate paths for use in case of failures. RIP presently supports only the computation of the unique shortest path between nodes.

The undesirable characteristics described above make RIP a poor candidate to serve as the foundation for an optical routing protocol.

The next 3 protocols we consider are all link state routing protocols. By sequencing successive updates of link state information, these protocols sidestep the count-to-infinity problem exhibited by RIP. In addition, by enabling each switch to acquire a complete view of the network topology, they implicitly avoid RIP’s routing loop problem.

3.2.2 Intermediate System To Intermediate System (IS-IS)

In ISO terminology, a router is referred to as an *Intermediate System* (IS). The *IS-IS* protocol (see [8, 15, 19]) is a link state IGP that was originally developed for routing ISO/CLNP[1] packets. IS-IS has many desirable features, including:

* *Scalability to large networks:* IS-IS intra-domain routing is organized into a 2-levels of hierarchy, allowing large domains to be administratively subdivided.
* *Multi-path forwarding:* IS-IS uses a “shortest path first” algorithm to determine routes, and the new IS-IS protocol supports multi-path forwarding.
* *Genericity:* IS-IS can support multiple communication protocols and is therefore generic.

[1]International Organization for Standardization/Connectionless Network Protocol

IS-IS does not flood any routing information across level 1 areas. Thus, internal IS-IS routers always send traffic destined to other level 1 areas to the nearest level 2 router Area Border Router (ABR). If there is more than one ABR, IS-IS may not pick the optimal route. In this manner, IS-IS trades off route optimality in exchange for lower control traffic, this allows IS-IS to scale to larger networks. Unfortunately, IS-IS is not an IETF standard and has only one RFC, which is not up to date with commercial implementations. As a consequence, IS-IS is no longer an open IGP routing protocol, and so is a risky choice for extension to the optical domain.

3.2.3 Private Network to Network Interface (PNNI)

PNNI is a link state routing protocol standardized by the ATM Forum [5]. Intended for use in ATM networks, PNNI offers many features that would make it a good starting point for developing a WDM routing protocol. These include:

* *Scalability to large networks:* PNNI implements hierarchic partitioning of networks into Peer Groups (up to 104 levels), and supports aggregation of information from within each Peer Group. Together, hierarchy and aggregation provide a flexible tradeoff curve between route optimality and protocol traffic. The PNNI hierarchy is configured automatically based on switch addresses using dynamic election protocols.
* *Support for traffic engineering and QoS:* PNNI distributes link state information in PNNI Topology State Elements (PTSEs); these come in many flavors, including Horizontal Link InfoGroup (HLinkIG) packets for advertising link characteristics, and Nodal InfoGroup (NodalIG) packets for advertising switch state information. HLinkIG packets include resource availabilities for a link on a per service-class basis; it would be quite easy to extend this description to include wavelength information. Similarly, it would be fairly straightforward to extend the NodalIG to include information about the wavelength conversion capabilities of a switch.
* *Standardization:* PNNI was approved by the ATM Forum and has since been adopted by most ATM vendors and shown to work in the field.

In terms of the technical advantages outlined above, PNNI is a strong candidate to serve as the foundation for an optical routing protocol. PNNI supports policy-based QoS aware routing by by permitting the source switch (and each border switch) to compute a partial route through its peer-group towards the destination. This makes it difficult for PNNI to support multi-path forwarding. PNNI is, relatively a complex protocol, and perhaps as a consequence, the optical switch industry has turned away from considering PNNI-based WDM routing solutions. Interoperability with existing optical switch technology makes PNNI a problematic choice.

3.2.4 Open Shortest Path First (OSPF)

Open Shortest Path First Routing Protocol [14, 15, 18] is a link-state IGP that distributes routing information between routers within a single autonomous system. OSPF has many desirable features, including:

* *Easily extended for TE and wavelength information.* OSPF distributes link state information in Link State Advertisement (LSA) packets. In the RFC [10], Coltun proposes Opaque LSAs to extended the OSPF protocol. Opaque LSAs have a standard LSA header followed by an application specific information, which may be used directly by OSPF or by other applications. Interoperability was achieved by introducing 3 new Link State (LS) types, each with fixed flooding regimes: (i) LS type 9 packets are not flooded beyond the local subnetwork, (ii) LS type 10 packets are not flooded beyond the borders of their associated area, and (iii) LS type 11 packets are flooded throughout the Autonomous System. Routers that are not opaque capable may receive Opaque LSAs because they cohabit broadcast networks with opaque capable routers; in this case the LSAs are discarded, because they have unknown LS types. Opaque LSAs make it possible to extend OSPF to incorporate new domain-specific TE information, as needed; a lot of effort has been expended to specify the exact format of these packets for TE in the IP domain.
* *Scalability:* OSPF supports 2-levels of hierarchy and groups routers into areas. Because OSPF advertises a summary of the information describing the areas inside the Autonomous System, the protocol is more likely (compared to IS-IS) to compute an optimal path when routing between different areas. OSPF converges reasonably quickly even for large networks.
* *Simplicity and Acceptance:* OSPF is a relatively simple protocol (compared to PNNI), and enjoys widespread commercial adoption.
* *Standardization:* OSPF is a full IETF Standard (unlike IS-IS).

In the next section, we present some of the OSPF-based initiatives for WDM routing.

3.3 OSPF-based Initiatives for WDM Routing

Currently, most of the routing protocols being proposed for WDM are based on OSPF. We consider two of the most promising proposed extensions now.

3.3.1 The QoS Extension to OSPF

In the RFC [4], Apostolopoulos et al. propose an extension to OSPF to support QoS. In the same reference, the authors propose that the Type of Service (TOS) field within the protocol packet options field should be used to specify whether the originating router is

QoS capable. The bandwidth and delay of each link will be encoded into the metric field of the LSA packet by QoS capable routers. By using three bits for the exponent part and the remaining thirteen bits for the mantissa, the scheme can be used to represent bandwidths ranging into Gbits/second.

The main advantage of this proposal is that it does not require the addition of any additional LSAs. This makes it simple to integrate QoS-capable routers running the extended protocol with routers running older versions of OSPF (i.e. non QoS-capable routers). However, the proposed scheme suffers from the drawback of not being able to advertise additional information such as a complete list of available wavelengths for routing inside WDM networks.

3.3.2 OSPF Extension Using Opaque LSAs

Here we give a brief outline of ongoing efforts of proposals using Opaque LSAs to extend OSPF for routing in the optical domain.

Recent Internet drafts by Chaudhuri et al. [9] and Basak et al. [7] present requirements for optical bandwidth management and restoration in a dynamically reconfigurable network. Both papers expand on the list of optical network characteristics that must be maintained in the OXC routing database. The information is classified into two categories: First there is the information that must be advertised using OSPF, e.g. the total number of active channels, the number of allocated channels, the number of preemptable channels, the risk groups throughout the network, etc. The second category consists of information that is kept locally and is not advertised for protocol scalability reasons. Examples of such information are the available capacity, the preemptable capacity, the association between fibers and wavelengths, etc. The proposals choose to put information about the wavelength availabilities in the second category, arguing that available wavelength changes occur frequently, so advertising the changes in the available wavelength does not yield a performance increase proportionate to the costs in terms of control traffic.

In the internet draft [16], Kompella et al. present an extension to OSPF for optical routing. In particular, they specify some of the parameters needed to be advertised by the Opaque OSPF LSAs. Like the author of [7, 9], Kompella et al propose that OSPF should not advertise any information pertaining to wavelength availability, and postpone wavelength assignment to the time when lightpaths are signaled.

In the internet draft [21], Wang et al. propose to use the OSPF protocols to advertise both the number of available wavelength per fiber and the available bandwidth. To address the objection that the advertisement of the available bandwidth is impractical because of rapid changes in network link usage, the authors also introduce the notion of “significant change”. In their extension, OSPF readvertises only if the available bandwidth changes by a factor which is higher than a certain tunable threshold.

Some of the TE and QoS parameters change very frequently, raising the issue of when to advertise the changes of the network characteristics throughout the whole network. The original OSPF standard mandates a variety of tunable parameters controlling the flooding of LSAs, including the "MinLSInterval" which specifies the time between any two consecutive LSA originations, and the "MinLSArrival" which limits the frequency of accepting a newer instances of LSAs.

In [3], Apostolopoulos et al. present other policies dealing with the issue of when a router should flood a new LSA to advertise changes in its link metric. Some of the proposed policies include:

- *Threshold based policies* that trigger updates when the difference between the previously flooded and the current value of available link bandwidth is larger than a configurable threshold.
- Class based policies which partition the capacity of a link into a number of classes and re-advertise when a class boundary is crossed.
- *Timer based policies* which generates update at fixed, or enforce a minimum spacing between two consecutive updates. time intervals.

In the next section, we present a design of a routing protocol in optical network (Toolkit for Routing in Optical Networks (TRON)). TRON combines both models for routing information exchange: Peer model in the sense that TRON makes the router aware of the optical network details, and overlay model in the sense that TRON uses the collected information to compute source routes.

3.4 Toolkit for Routing in Optical Networks (TRON)

The Toolkit for Routing in Optical Networks (TRON) is a simulation/emulation library developed to facilitate experiments on OSPF-based routing protocols for optical networks. Currently, TRON implements the *LightWave-OSPF* routing protocol, which is an adaptation of the optical extensions to OSPF proposed in [16] and [21]. TRON may be used both to simulate *LightWave-OSPF* routing in large optical networks, as well as to emulate routing on a live optical switch.

In the next section, we present the LightWave-OSPF.

3.4.1 LightWave-OSPF

LightWave-OSPF follows the internet draft [16] of Kompella et al which presents an extension to OSPF for routing in optical networks. The proposal makes use of the Opaque OSPF LSAs which were first introduced by Coltun [10] as a way to extend the OSPF protocol. While *LightWave-OSPF* largely adopts the proposal of [16], there is one crucial exception which we now describe.

Like the earlier drafts of Chaudhuri et al.[9] and Basak et al.[7], Kompella et al. take the stand that an optical adaptation of the OSPF protocol should not advertise any information pertaining to wavelength availability in link state advertisements. They argue that the set of available wavelengths changes frequently on each link, so advertising these changes would not yield a performance increase proportionate to the communication cost of increased control traffic. Instead, they postpone the problem of wavelength assignment to a later stage when the lightpaths are being signaled/provisioned. Here, the authors break from the proposal of Kompella et al.

Instead, *LightWave-OSPF* takes the approach of Wang et al. as presented in Internet draft [21]. There, the authors propose that an optical adaptation of the OSPF protocol should advertise both the number of available wavelengths per fiber and the total available bandwidth. To address the objection that advertisement of the available bandwidth is impractical because of rapid changes in network link usage, the authors of the draft incorporate a tunable threshold to determine when a change is "significant" enough to mandate readvertisement.

The rationale for selecting the approach of Wang et al is the following: In an heterogeneous optical network where a significant fraction of the switches are not wavelength-conversion capable, the absence of wavelength information in the description of network link will cause the route computation algorithm to be operating in the dark. In particular, as wavelength utilization of network links increases, the probability of selecting an effective source route that can actually be provisioned decreases dramatically, because while many routes exist from source to destination, wavelength continuity constraints at the transit non-wavelength-conversion-capable switches render most of these routes infeasible. In the proposals [9, 7, 16], the infeasibility of the computed routes would not be detected until signaling was actually attempted and failed. Each lightpath setup experience a large number of "crankbacks" or reattempts, with the cumulative effect being that as the load on the network increases, the lightpath setup latency would increase prohibitively (even in the absence of contention between concurrent setup requests!). We remark that the above problem gets worse as the fraction of non-wavelength-conversion-capable switches in the network increases.

LightWave-OSPF also incorporates the recommendations presented by Apostolopoulos et al. in [3] by implementing a range of policies to determine when a router should flood a new LSA advertising a change in its link metric. The policies presently implemented include: (1) *Threshold based policies* that trigger updates when the difference between the previously flooded and the current value of available link bandwidth is larger than a configurable threshold, and (2) *Timer based policies* which generates update at fixed, or enforce a minimum spacing between two consecutive updates. time intervals.

In the next section, we present the TRON architecture.

3.4.2 TRON Architecture

The TRON architecture can be decomposed functionally into a set of cooperating modules, as shown in figure 21. The modular design facilitates future modifications and extensions to the TRON software. Because each OXC switch has several interfaces, certain

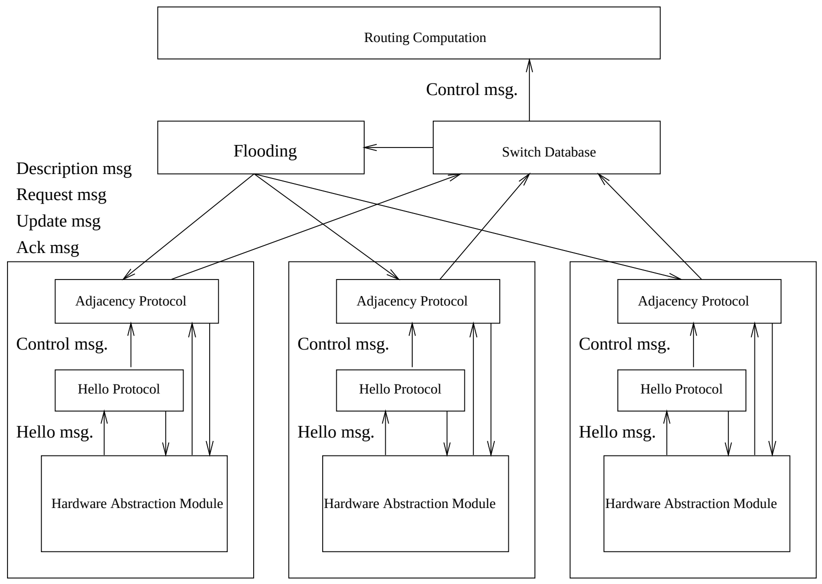


Figure 21: TRON Architecture

modules such as the "Adjacency Protocol", the "Hello Protocol", the "Network Interface", and the "Hardware Abstraction" modules have to be reproduced for each interface. In contrast, the "Routing", "Flooding" and "Switch Database" modules are instantiated once for each OXC switch. We will briefly description each of these components in the next section.

1. **Network Interface** and **Hardware Abstraction** modules

   The TRON stack is built over these two modules. Together, they are responsible for reporting any failure or change in local switch resources, by generating events and forwarding them upwards for the rest of the TRON stack to respond to.

   The Network Interface module represents the connection between the OXC router and the network. This module informs the router whether the interface to the physical network is up or down, and to maintain information about the type of network to which the interface attaches, the interface's IP address, the area ID, a list of neighboring routers, the router priority, the IP address of the Designated Router (DR) and Backup Designated Router (BDR).

   The Hardware Abstraction module monitors the OXC for alarms and alerts triggered by the failures or changes to the OXC's internal components. The The Hardware Abstraction module maps these physical hardware dependent signals to a set of generic messages that are processed by the Hello and Flooding module. An example of generic messages are the `OXCPortUp` and the `OXCPortDown` messages which indicate a port's operation and failure, respectively. Extending TRON to work with other optical switches then requires only modification to the Hardware Abstraction module.

2. **Hello** module

   The Hello module implements the Hello protocol, which is responsible for maintaining neighbor relationships between OXCs. Specifically, the Hello protocol serves to certify that the links connecting neighboring routers correctly support bidirectional communication. The protocol specifies that every `HelloInterval` seconds, each router sends a Hello packet containing the identities of the neighbors that it knows about. `TwoWay` connectivity is attained when the local router's ID is listed in the Hello packet received from a remote router; otherwise, the connectivity is declared to be only `OneWay`. The Hello protocol is also responsible for electing a Designated and Backup Designated Routers, by carrying election priority information in the Hello packets.

   The Hello module is located directly above the Network Interface and Hardware Abstraction modules, and the Hello protocol is sensitive to messages generated by these modules.

3. **Adjacency** module

   The Adjacency module lies above the Hello module within the TRON stack, and implements the Adjacency/Flooding (A/F) protocol. The A/F FSM (see figure 3) is quite similar to the adjacency FSM of the original OSPF. The A/F FSM is triggered by the Hello protocol: Upon reaching the `TwoWay` state the Hello protocol sends a `Begin` control message to the A/F FSM triggering it to become operational by entering the `Extstart` state.

   The Adjacency protocol is responsible for the synchronization of Switch Databases of neighboring routers. The exchange protocol is asymmetric: the first step is to determine a "master" and a "slave" relationship. After the master-slave relationship negotiation, the new state is `Exchanging`. The two routers then exchange Description packets containing a summary of their respective databases which lists the advertisements within it. Based on this, the routers will request and exchange the information as necessary to synchronize their databases. This is achieved using Request packets, Update packets, and Acknowledgment packets. The routers use checksums to ensure that all protocol packets have valid contents. Once all necessary exchanges have been completed, the protocol enters th `Full` state and the Flooding module is notified. Subsequently, the Adjacency module is subservient to the Flooding module which may periodically give it LSAs to flood. The state transition diagram of the A/F FSM is depicted in figure 22.

   The Adjacency protocol is terminated if the Hello module sends a message up withdrawing certification of `TwoWay` connectivity with the peer. Such failures are detected by the Hardware Abstraction and Network Interface modules, which inform the Hello protocol which in turn notifies the A.F FSM by sending a `Halt` control message. The latter message causes the interruption of the A/F FSM and its transition to the initial state `WaitMessageFromHello`.

   In *LightWave-OSPF*, the Update packets carry a list of Link State Advertisements (LSAs). These LSAs are encapsulated into Opaque LSAs. The Opaque LSAs

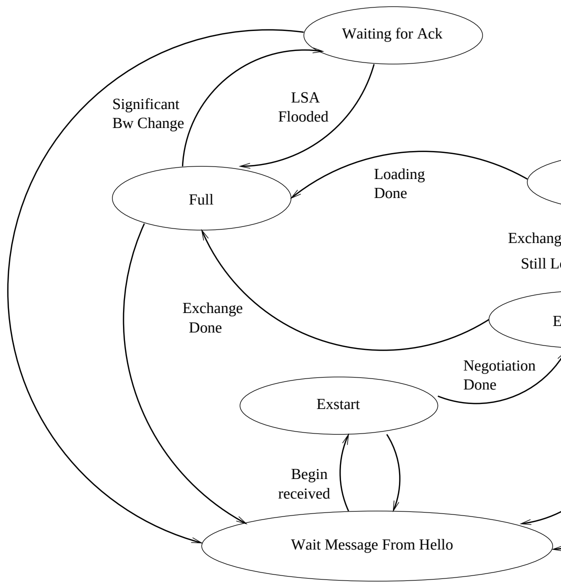


Figure 22: A/F FSM for *LightWave-OSPF*

carry different types of Type/Length/Value (TLVs) and SubTLVs packets. The TLVs and SubTLVs include the following: Traffic Engineering Metric, Maximum Bandwidth, Unreserved Bandwidth, Maximum Reservable Bandwidth, Resource Class/Color, Link Media Type, Shared Risk Link Group, Available Wavelength, Number of Active as well as Preemptable Channels.

4. **Flooding** module

   The Flooding module may interact with all instances of the Adjacency modules. This interaction begins only after the Adjacency module has fully synchronized with the remote peer. The Flooding module is invoked whenever one of the following three events occurs:

   (a) When there is a failure in one of the incident links as detected by the Hello module. In this scenario, the Flooding module must flush the LSA for the failed link from the network. It does this by inserting a new LSA for the link into its Switch Database (with a higher sequence number and age set to `MaxAge`) and forwarding it out on all interfaces, via their respective Adjacency modules.

   (b) In case of a significant change in the available bandwidth of an incident link. In this scenario, the Flooding module inserts a new, more accurate LSA describing the link into its Switch Database—with a higher sequence number—and forwarding the new LSA out on all interfaces, via their respective Adjacency modules.

(c) When new LSA arrives with more up-to-date information about the network's topology/attributes. In this scenario, the Flooding module inserts the newer LSA into the Switch Database, acknowledges the receipt of the update, and then forwards the new LSA on all interfaces via their Adjacency modules *with the exclusion of the Adjacency module that reported the update to the Flooding module.*

5. **Switch Database** module

   This module maintains a collection of LSAs which have been collected by the Adjacency protocol and subsequent operation of the Flooding module. Upon the receipt of a new LSA, the SwitchDatabase determines whether the LSA is a newer version of an LSAs that it already owns. If so, the newer LSA replaces the older one; otherwise the LSA is discarded. The SwitchDatabase is also responsible for continuously aging the LSAs and removing those whose age reaches `MaxAge`. The Switch Database also periodically reoriginates LSAs (with higher sequence number and age set to zero) that have been previously generated by the router, thereby ensuring that they do not disappear from the Databases of other routers in the network.

6. **Routing** module

   The Routing module has the main task of building a graph representation of the network based on the contents of the Switch Database module, and responds to requests for the computation of routes to destination routers. This route computation may require consideration of Traffic Engineering and QoS requirements for the connection, and reconciling them with the corresponding information within the LSAs. Traditionally, the next-hop along these routes is cached in a routing table for hop-by-hop routing. In the case of *LightWave-OSPF*, however, the entire route is used as a source-route; the lightpath provisioning protocol attempts to provision the path along the specified route.

3.4.3 Integration of Provisioning/Signaling with TRON

Presently, when running in emulation mode, TRON sends all its control messages out-of-band, on an auxiliary control network operating IP over ATM. Each optical switch running a TRON stack must therefore be configured with information about the identities of its neighboring optical switches. Specifically, it must be told the location and listening port of the TRON stacks of each of its neighbors, so that it can establish out-of-band connectivity to these peers prior to booting. In future releases of TRON, we hope to consider in-band transport of routing control messages.

In simulation mode, TRON operates multiple stacks within the same process, so all control messages are exchanged within the simulation process and no out-of-band (or in-band) connectivity is required.

The TRON software can be considered as the routing computation component of the provisioning protocols, which are in turn viewed as clients of TRON. These clients use the information available computed by TRON to physically setup lightpaths. In our present

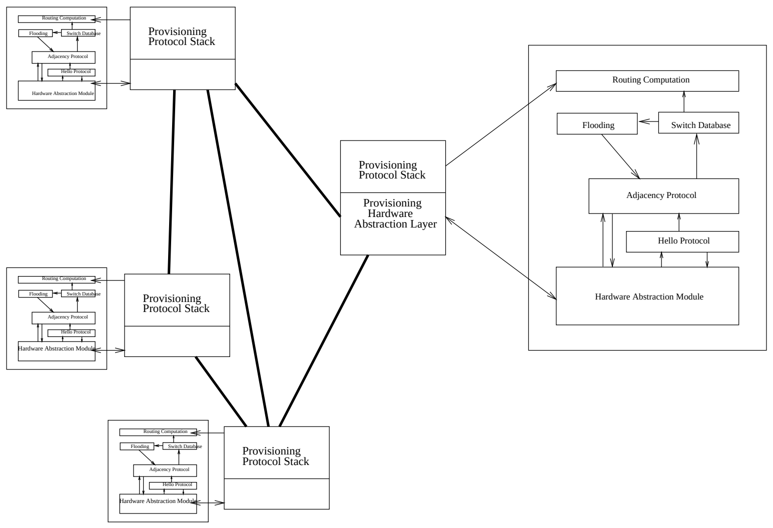

Routing Computation
Flooding
Switch Database
Adjacency Protocol
Hello Protocol
Hardware Abstraction Module
Provisioning Protocol Stack
Provisioning Hardware Abstraction Layer

Figure 23: Interaction of Optical OSPF and Provisioning protocols

scheme, the lightpath is determined by source routing establishment, unlike hop-by-hop schemes such LSP establishment in MPLS.

Once the provisioning protocol gets the necessary information from the Routing module, it starts physically setting up the path by establishing bindings at each of the transit switches. If it succeeds, the provisioning protocol informs the Hardware Abstraction modules of every switch along the route about the change in bandwidth availability on affected links.

The Hardware Abstraction modules report this change, and if the change is determined to be significant, a new link state advertisement of the affected link is reoriginated and flooded to all neighbor routers. If the change is not significant, then the notification is ignored. Figure 23 represents the interaction between the TRON and the provisioning protocol.

3.4.4 The CASiNO Framework

The TRON software was implemented in C++ using the Component Architecture for Simulating Network Objects (CASiNO) [17]. In this section, we present a brief overview of the CASiNO framework.

The Component Architecture for Simulating Network Objects (CASiNO) is useful for the implementation of communication protocol stacks and network simulators. CASiNO framework implements a rich, modular coarse-grained dataflow architecture, with an interface to a reactor kernel that manages the application's handlers for asynchronous I/O, real timers, and custom interrupts. In the subsequent sections, we introduce the principal components of CASiNO.

The **Reactor Kernel** is the name given to the collection of classes that provide applications with access to asynchronous events, such as the setting and expiration of timers, notification of data arrival, and delivery and receipt of intra-application interrupts. CASiNO can be used to implement both live network protocols as well as their simulations. Both cases require precise coordination of program execution relative to some notion of time. In the case of network simulation, however, the notion of time is quite different because the time within the simulation need not remain in lockstep with the real wall clock time. The Reactor Kernel can switch between these paradigms transparently, and user programs are not affected in any way (in fact are oblivious to) whether they are operating in simulated or "live" mode.

The **Data-Flow Architecture** is the name given to the collection of classes that define the behavior of the modules within a protocol stack. CASiNO has three tiers in its design: (i) The Conduit level of abstraction represents building blocks of the protocol stack; the function of the Conduit class is to provide a uniform interface for joining these blocks together and injecting Visitors into the resulting networks. (ii) The Behavior level of abstraction represents broad classes, each capturing different semantics regarding the Conduit's purpose and activity. Finally, (iii) at the Actor level of abstraction, the user defines concrete implementations. There are five subclasses of the Actor class: Terminal, Accessor, State, Creator, and Expander. These subclasses of the Actor correspond respectively to the Behavior's subclasses, Adapter, Mux, Protocol, Factory, and Cluster. We now describe each of these Behaviors and their associated Actor.

1. **Protocol Behavior - State Actor**

   A communication protocol is often specified by a finite state machine. A Protocol behavior implements a finite state machine using a specialized State Actor. When a Visitor arrives at a Protocol, it is handed to the State actor. The State then queries the Visitor to dynamically determine what type of Visitor it is. Once the State determines the type of the Visitor, the State transforms the Visitor into the appropriate type casting and operates on the Visitor accordingly. The State Actor is implemented using the state pattern defined in [12].

2. **Adapter Behavior - Terminal Actor**

   The Adapter Behavior as its name indicates, converts or adapts the outside world to the Conduits and Visitors comprising the CASiNO application. It is used typically for testing, where a Protocol stack is capped at both ends with two Adapters - one to inject Visitors and the other to sink them. The *Terminal* actor is used to configure the Adapter with a simple interface to allow Visitors to be injected and absorbed.

3. **Mux Behavior - Accessor Actor**

   The Conduit with a Mux Behavior allows developers to implement multiplexing and demultiplexing. The demultiplexing occurs when a Visitor arriving and needs to be routed to one of the several other Conduits on the basis of some aspect of the

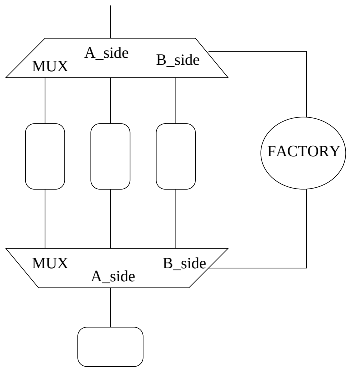


Figure 24: Common Usage of two Muxes and a Factory

Visitor's contents, which we refer to as the Visitor's key or address. The Accessor Actor is responsible for examining the incoming Visitor's contents and deciding which Conduit the Visitor should be sent to.

4. **Factory Behavior - Creator Actor**

   The Factory Conduit allows dynamic instantiation of new Conduits at Muxes. Thus, a Factory Behavior is always found with Muxes, see figure 24. A Conduit configured with Factory Behavior is connected to the B side of a Conduit configured with Mux Behavior. Upon receipt of a Visitor, the Factory may make a new Conduit. If it decides to do this, the Factory makes the newly instantiated Conduits known to the Accessor of the Mux. In this manner, Visitors can be sent to a Mux to install new Conduits dynamically, provided that a Factory is attached to the Mux. The Factory behavior is specified with a Creator Actor. After the installation of the newly created Conduit object, the incoming Visitor, which prompted the installation, is handed to the freshly instantiated Conduit by the Factory.

5. **Cluster Behavior - Expander Actor**

   In addition to the four primitive Behaviors of Protocol, Factory, Mux, and Adapter, CASiNO has generic extensibility via the Cluster Behavior. Just as the primitive behaviors mentioned above are configured with a particular Actor (i.e. a concrete State, Creator, Accessor, or Terminal object), the programmer can fully specify a Cluster's behavior by configuring it with a concrete Expander Actor. An Expander may be thought of as a black-box abstraction of a network of Conduits.

3.4.5 The TRON Architecture in terms of the CASiNO Framework

1. **Router** Expander

   The Router Expander is a collection of different types of Conduits: the SwitchDB Conduit, the SwitchProtocol Expander, and the HardwareAccessor Conduit. The SwitchProtocol Expander is attached to the B-side of the SwitchDB Conduit via its

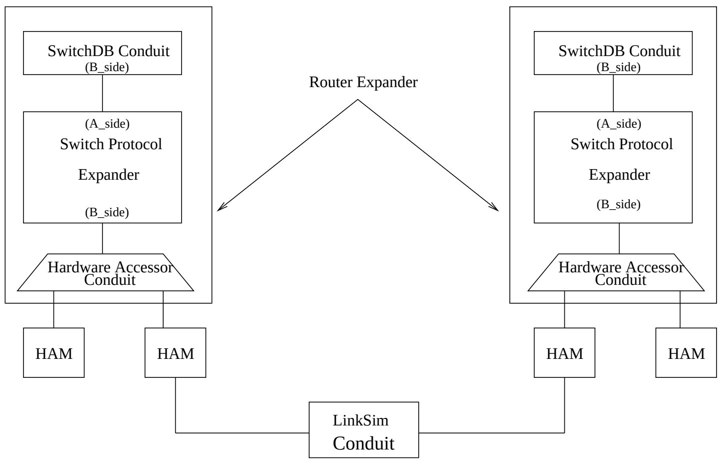


Figure 25: Router Expander Architecture

A-side, and to the A-side of the HardwareAccessor Conduit via its B-side. Figure 25 shows two Router Expanders attached to each other through a LinkSim Conduit, and two Hardware Abstraction Module (HAM). WE now describe each of these Conduits in further detail:

The **SwitchDB** Conduit maintains the switch network topology database. The operations it supports include: searching for LSAs, inserting LSAs, flooding LSAs, aging, flushing expired LSAs, etc. For example, when the Adjacency protocol running over the link reaches the `Full` state, the LSA corresponding to that link is instantiated and inserted into the SwitchDB. Once the LSA has been inserted, the SwitchDB calls the flooding protocol, which ages the LSA by adding the hop transmission delay time to the LSA age's field, then appends the LSA to an OSPF update Visitor. A copy of this Visitor is given to all Adjacency FSM running in the SwitchProtocol Expander.

The **SwitchProtocol** Expander represents the set of protocols running on the ports of the switch. More details about the internal architecture of the SwitchPort Expander will be provided in the next section.

The **HardwareAccessor** Conduit is responsible for routing the outgoing Visitors to their respective interfaces and HAMs.

The **HAM** is located between the Router Expander and the LinkSim Conduit. The main task of the HAM is to map the hardware signals and alarms returned by the OXC switch into a set of generic events which will be handled by the routing protocol. This module, insures that the routing protocol is hardware independent because the protocol relies only on the events returned by the HAM, and the HAM implements the adapter pattern [12]. In case of different types of hardware, only the HAM has to be updated so that the signals and alarms returned by the switch

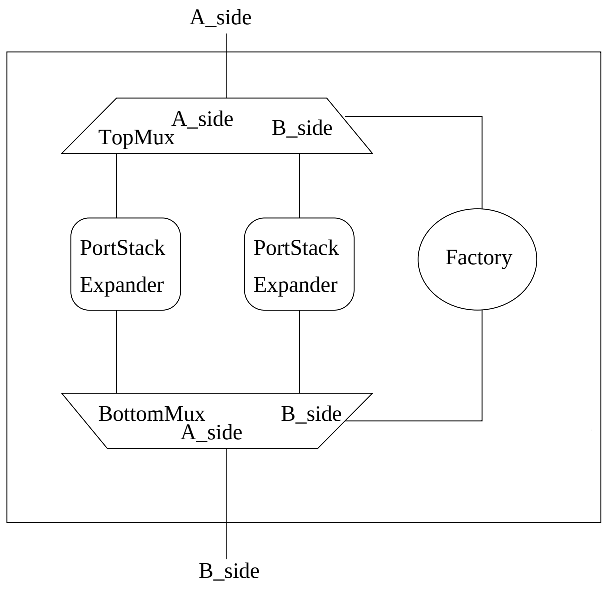


Figure 26: SwitchProtocol Expander

will be mapped to the same set of protocol events.

The **LinkSim** Conduit represents an optical link. It is is a means to simulate link failure, since it can be tuned so that it passes through a first set of Visitors, then suicides the next coming visitors (showing that the link is down), finally passes through the next incoming Visitors (indicating that the link has came up.)

We now examine the internal structure of the SwitchProtocol Expander.

2. **SwitchProtocol** Expander

   The SwitchProtocol Expander (figure 26) is a collection of several types of Conduits: a Factory Conduit, TopMux and BottomMux Conduit, and as many PortStack Conduits as needed.

   The TopMux and the BottomMux each maintains a dictionary which allows the incoming Visitors to be forwarded to their appropriate PortStack Expander. Upon the reception of a Visitor by either of these muxes, the Accessor searches in its dictionary for the Conduit PortStack which should receive the Visitor. In case this search fails, the Visitor is forwarded to the factory Conduit, causing the instantiation of a new PortStack Expander between the TopMux and the BottomMux. Hence the newly created PortStack Conduit becomes known to both muxes, and the Visitor is forwarded through the newly made Conduit PortStack.

   When this occurs, the above mechanism is used to dynamically create PortStack Expanders. As soon as two routers are joined by LinkSim Conduit, two instances of `InstallerVisitors` are made inside the LinkSim Conduit and one is sent to each of the attached routers. This special Visitor is always sent by the BottomMux to the factory, thus causing a new PortStack Conduit to be made.

We now describe the internal structure of the PortStack Expander.

3. **PortStack** Expander

   The PortStack architecture (figure 27) is a collection of two Conduits build on top of each other. The top Conduit contains a State Actor implementing the Adjacency (FSM). The bottom Conduit contains a State Actor is implementing the Hello FSM. The top Conduit is attached to the A-side of the bottom one through its B-side. Both Conduits are grouped together into a single Expander. The A-side of this Expander as a whole is the A-side of the Adjacency FSM Conduit, the B-side of the Expander is the B-side of the Hello FSM Conduit.

   All incoming Visitors entering the PortStack from the network are first seen by the Hello protocol Conduit. If the Visitor is not a `HelloVisitor`, then it just passed upwards to be handled by the Adjacency Protocol Conduit. The Hello FSM and the Adjacency FSM operate with the help of various types of timers. For example, in the Hello FSM, whenever a router sends a hello Packet to its peer, the FSM calls `Register()` the to start an `InactivityTimer`. This timer is aborted by calling `Cancel()` only if the Hello FSM receives hello packet in response to the sent message. On the other hand, if the `InactivityTimer` expires, the hello FSM concludes that the connection over this link is broken and therefore it has to notify the upper running protocol about this change. Likewise, during the database exchange, the Adjacency FSM uses timers to cope with packet loss. For example, a `DB_InactivityTimer` is always registered whenever a database description packet is sent. Expiring of the timer indicates a failure event in the Adjacency FSM, which triggers the necessary corrective actions.

   packet is sent, such as: `DB_InactivityTimer`, which is registered whenever a database description packet is sent, or `Req_InactivityTimer`, or `Upd_InactivityTimer`. Proceeding in this way, we assure that the database in all routers are most of the time synchronized.

References

[1] M. Ali, B. Ramamurthy, and J. Deogun. Routing and Wavelength Assignment (RWA) with power considerations in Optical Networks. IEEE Globecom '99 Symposium on High-Speed Networks, December, 1999.

[2] J. Anderson, J. Manchester, A. Rordiguez-Moral, and M. Veeraraghvan. Protocols and Architectures for IP Optical Networking. Bell Labs Technical Jounal, January, 1999.

[3] G. Apostolopoulos, R. Guerin, S. Kamat, and S. Tripathi. Quality of Service Based Routing: A performance Perspective. Proceedings of SIGCOMM, September, 1998.

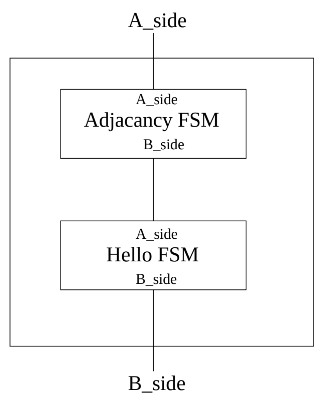


Figure 27: PortStack Expander

[4] G. Apostolopoulos, D. Williams, S. Kamat, A. O. R. Guerin, and T. Przygienda. QoS Routing Mechanisms and OSPF Extensions. Request for Comments No. 2676, August, 1999.

[5] ATM-Forum. *Private Network-Network Interface Specification, Version 1.0.* ATM-Forum, 1996.

[6] D. Awduche, J. Malcolm, J. Agogbua, and J. M. M. O'Dell. Requirements for Traffic Engineering over MPLS. Request for Comments No. 2702, September, 1999.

[7] D. Basak, D. Awduche, J. Drake, and Y. Rekhter. Multi-Protocol Lambda Switching: Issues in Combining MPLS Traffic Engineering Control with Optical Corssconnects. Internet Draft, January, 2000.

[8] R. Callon. Use of OSI IS-IS for routing in TCP/IP and Dual Environments. Request For Comments No. 1195, December, 1990.

[9] S. Chaudhuri, G. Hjalmtysson, and J. Yates. Control of Lightpaths in an Optical Network. Internet Draft, February, 2000.

[10] R. Coltun. The OSPF Opaque LSA Option. Request for Comments No. 2370, July, 1998.

[11] D. Fedyk and A. Ghanwani. Metrics and Resource Classes for Traffic Engineering. Internet Draft, October, 1999.

[12] J. V. Gamma, Helm. *Design Patterns: Elements of Reusable Object-Oriented Software.* Addison Wesleyprofessional computing series, April, 1997.

[13] C. Hedrick. Routing Information Protocol. Request for Comments No. 1058, June, 1988.

[14] C. Huitema. *Routing in The Internet.* Prentice Hall.

[15] IETF. The MPLS Operations Mailing List. IETF's MPLS Working Group.

[16] K. Kompella, Y. Rekhter, D. Awduche, J. L. G. Hjalmtysson, S. Okamoto, D. Basak, G. Bernstein, J. Drake, N. Margalit, and E. Stern. Extensions to IS-IS/OSPF and RSVP in support of MPL(ambda)S. Internet Draft, October, 1999.

[17] S. Mouncastle, D. Talmage, S. Marsh, B. Khan, A. Battou, and D. C. Lee. CASiNO: a Component Archtecture for Simulating Network Objects.

[18] J. Moy. OSPF Version 2. Request for Comments No. 2178, July, 1997.

[19] R. Perlman. *Interconnections Bridges and Routers.* Addison Wesley, June, 1993.

[20] R. Ramaswami and K. Sivarajan. *Optical Networks: A Practical Perspective.* Morgan Kaufmann Publishers, Inc, 1998.

[21] G. Wang, D. Fdyk, V. Sharma, K. Owens, G. Ash, M. Krishnaswamy, Y. Cao, M. Girish, H. Ruck, S. Bernstein, P. Nquyen, S. Ahluwalia, L. Wang, A. Doria, and H. Hummel. Extensions to OSPF/IS-IS for Optical Routing. Internet Draft, March, 2000.

[22] H. Zang, J. Jue, and B. Mukherjee. A Review of Routing and Wavelength Assignment Approaches for Wavelength-Routed Optical WDM Network.

Index