Figure 7: An example of a loop if the switches are set by RESERVE messages

Figure 8: A spanning tree of spare links.


handling link failures and wavelength failures on a link.

References


the failed fiber to the working one upon detection of the failure. We propose here a more economical way. Consider the graph defined by the network. On this graph we build a spanning tree of spare links as shown in Fig. 8. Each node consists of a switching node that performs the wavelength routing function as before, as well as a bypass switch connected as shown in the figure. At node A for example, under normal conditions the bypass switch would connect Link BA to BA-s, Link DA to DA-s and Link EA to EA-s respectively, thus being transparent. If a link, say DA, fails, the bypass switch is set so that DA-s is now connected to the spare link BA-spare. In general the controllers at the nodes next to the failure determine the path on the spanning tree that connects the two nodes, send messages to the controllers on that path to set up the bypass switches at the intermediate nodes accordingly so that the signal on the failed link is rerouted via the spanning tree to its destination. For example, in Fig. 8, if link DA fails the signals on link DA are rerouted via the spanning-tree links BA-spare, BC-spare and CD-spare. At the same time, the controllers next to the failure inform all other controllers about usage of the portion of the tree that was taken.

8 Summary

This paper proposed distributed protocols for setting up, taking down and maintaining the state of connections in a wavelength-routed optical network. These protocols are general in that they can be used in networks with (a) different kinds of node architectures (electrical/optical switching, with or without wavelength conversion, etc.), (b) different route-computation algorithms that reflect the underlying constraints imposed by the nodes and the physical layer, and (c) any kind of underlying mechanism using which the network controllers can communicate with each other.

We showed that in the absence of failures and contentions, the setup protocol always correctly sets up a connection. The probability of contentions is also minimized by the protocol. The update protocol ensures that the nodes have a consistent view of all the connections. The update and takedown protocols together ensure that resources are released under most circumstances when they are no longer required. Fast protocols were also proposed for
as opposed to the regular setup procedure, here each controller on the path of the substitute connection sets the switch and the status bit to 0 at the time when it receives RESERVE, i.e., when it enters state reserved. The endpoints of the connection start sending data on the backup connection as soon as they receive the RESERVE message. Controllers send RESERVE-ACK messages back to the local controller and start RESERVE-TIMER.

- when all RESERVE-ACK messages are received by the local controller, the latter sends SETUP messages in parallel.

- when each controller on the path of the substitute connection receives the SETUP message, it enters state up and cancels the RESERVE-TIMER.

The procedure above speeds up the setup of the substitute connection, but has the drawback that it allows loops in up with status bit = 0, as illustrated in the following example (see Fig. 7): A wavelength on link EC fails and E attempts to set up a substitute connection on the route E,C,A,B,D,F. Also, a wavelength on link HA fails and H attempts to set up a substitute connection on route on H,A,C,D,B,G. The RESERVE of the former reaches A and B, the switches are set and then the controllers at A and B fail. The RESERVE of the latter reaches C and D and then those controllers fail. They all recover in state up with status bits = 0 and a loop A,B,D,C. Obviously, all controllers along the loop have to fail in order for the loop with status bits = 0 to be formed. Otherwise, the RESERVE-TIMER will expire at the controller that doesn’t fail, preventing the loop with status bits = 0 to be formed.

There should be a topological update for the spare wavelength, so that all controllers will know that if there is another failure, they will not attempt to use routes that overlap with the used ones. Connections that use the spare wavelength should be rerouted as soon as possible.

7.2 Link Failure

The obvious way to backup link failures is to install two fibers on every link on different physical paths, and use optical switches at the ends of the link to transfer the signal from
The originator controller senses that the connection is not up via the IDLE-TIMER and at that time it sends TAKEDOWN messages. The messages will arrive at the controllers that are connected to the source controller, thus $d)$. The originator and destination controllers sense that the connection is not up via the IDLE-TIMER and at that time delete the corresponding entry from their CST. Part $e)$ follows now from Lemma 3b).

When a link fails, the controller next to the link deletes the corresponding entry in $<E2>$ and then $f)$ follows from Lemma 3b).

\section{Backup}

In this section we describe procedures for fast recovery of connections after the failure of a particular wavelength on a link or of an entire link.

\subsection{Wavelength Failure on a Link}

For backup purposes, we keep one of the wavelengths as spare on each link in the entire network. The recovery procedure is coordinated by the controller at one of the nodes where the failure occurs, referred to as the \textit{local controller}. We use some convention to decide which of the two controllers next to the failure is in charge, for example the controller that is first in the path from originator to destination. When the local controller detects the failure of a wavelength on a link it deducts from its local CST the endpoints of the affected connection, finds (or has stored) some substitute path between these two endpoints, and establishes a substitute connection between the two by sending messages to all controllers on the substitute path. The setup protocol proposed in Sec. 3 requires several exchanges of messages and may be too lengthy for a substitute connection. To this end we propose a faster procedure for establishing substitute connections:

- The local controller sends RESERVE messages in parallel to all the controllers along the substitute path.
b) If all controllers corresponding to a loop connection section are working, they will eventually delete the corresponding entries.

Consider a given linear connection section and suppose that the corresponding connection is not up. Then:

c) The corresponding entry is deleted at all controllers that do not receive the SETUP message and at all controllers that are strongly connected to one of the controllers that does not receive SETUP.

d) If the originator controller is up, it has or will send TAKEDOWN messages. The corresponding entry will be deleted at all controllers that are connected to the originator from the time the originator sends the TAKEDOWN messages and until the TAKEDOWN messages arrive.

e) If the originator or destination controller is up, the corresponding entry will be deleted at all controllers that are strongly connected to it.

f) If a link is down, the corresponding entry will be deleted at all controllers that are strongly connected to the controllers that are adjacent to the failed link.

Proof: By Lemma 1, we do not have to worry about a controller reentering state reserve or up after the corresponding entry is deleted. Part a) follows from Lemma 2b). To prove b), it is sufficient by Lemma 3 to show that at least one controller deletes the corresponding entry, since all controllers around the loop are strongly connected. If all controllers around the loop have entries that correspond to the connection section, some of the entries must be in state reserved. An entry cannot stay in reserved forever, according to Lemma 3 and not all entries can go to up, because that would close a loop with entries in state up, thus at least one entry will be deleted. Hence b) follows. Controllers that do not receive the RESERVE message or that respond with a RESERVE-NACK message do not have a corresponding entry. Those that respond with RESERVE-ACK enter state reserved and if they do not receive the SETUP message, will delete the entry according to Lemma 3a). The controllers that are strongly connected to such a controller and do receive the SETUP message, move to state up and will delete the corresponding entry according to Lemma 3b). Thus c) follows.
a) A controller cannot stay indefinitely in state reserved.

b) Let \( A_m \) and \( A_p \) be two controllers that are strongly connected and remain so for sufficiently long time. If \( A_m \) has the entry corresponding to the considered section in state up and \( A_p \) does not have a corresponding entry, then \( A_m \) will also delete the entry.

**Proof:** While in state reserved, the RESERVE-TIMER runs and if a SETUP(id) or a TAKEDOWN(id) doesn’t arrive, the RESERVE-TIMER will expire and take the entry out of reserved state. Thus a). Controller \( A_m \) has an entry corresponding to the considered connection in state up and \( A_p \) does not. Therefore there is a last controller between \( A_m \) and \( A_p \) that has an entry in up and, without loss of generality, assume that \( A_m \) is this last controller. Denote by \( A_{m+1} \) the controller next to \( A_m \) towards \( A_p \) and let \( \lambda_m \) be the wavelength of the considered connection on the link \((A_m, A_{m+1})\). At node \( A_{m+1} \) there is no entry in state up with port \( A_m, \lambda_m \). We have to consider only the following two cases:

- \( A_{m+1} \) does not have a CST entry with port \( A_m, \lambda_m \). When the CST update sent by \( A_{m+1} \) reaches \( A_m \), the latter will delete its entry in \( <C4> \).

- \( A_{m+1} \) has an CST entry with port \( A_m, \lambda_m \) in state reserved. This can occur due to an attempt to reserve and set up a new connection. But the reservation attempt will not be successful because \( \lambda_m \) is not free at \( A_m \) on link \((A_m, A_{m+1})\) and therefore either the RESERVE-TIMER will expire or a TAKEDOWN message will arrive at \( A_{m+1} \). At that time \( A_{m+1} \) will delete the entry. Since we assume that the CST updates are frequent enough, controller \( A_{m+1} \) will send a CST update before it has time to get another reservation attempt, if any. When \( A_m \) receives that CST update, it will delete the entry.

\( \square \)

**Theorem 3**  \((Release of resources)\) Suppose that the network reaches a state when there are no more failures/recoveries of links, switches or controllers.

a) There are no loop connection sections with all corresponding controllers working and corresponding entries in state up.
demonstrated in Sec. 7.1.

**Definition 4** A connection is said to be up if it has been set up successfully and its endpoints are using it.

**Theorem 2** If a connection is up, the corresponding entries at all controllers that are alive are in state up.

*Proof:* The entry at each controller enters state up when the SETUP message arrives at the controller and the corresponding status bits are set to 0. If a controller fails, when it comes back up, the entry is entered into the CST in state up. Since the connection is up, no TAKEDOWN messages are sent and no IDLE-TIMER expires. Thus the only way to change the state of the entry is to delete it in <C4>. Suppose, by contradiction, that there is a controller that deletes the entry and let A be the first one. This means that the CST it receives from adjacent controller B say on the connection does not contain a related entry. Controller B did have the entry, in reserved or up, because the connection can be up only after all controllers had an entry in reserved for the connection. In addition, no RESERVE-TIMER has expired. Thus B has deleted the entry in <C4> before sending the CST update, contradicting the fact that A is the first one to do so.

**Definition 5**

a) Two controllers on a connection section are said to be connected.

b) Two connected controllers are said to be strongly connected if the controllers between them on the connection are up.

In the following we assume that the UPDATE-TIMER is such that the frequency of CST updates is high compared to connection setup attempts. As seen in the proof of Lemma 3b), if this is not the case, there may be race conditions between the two that will prevent release of resources.

**Lemma 3**
Lemma 2 (Loop Freedom)

a) There are no loop connection sections with all corresponding status bits $= 0$.

b) There are no loop connection sections with all corresponding controllers working and corresponding CST entries in state up.

Proof: A switch setting can be changed either when a switch recovers after failure (since the setting upon recovery of the switch is independent of its setting just before the failure) or when a controller receives a SETUP(id) message, in $<J4>$. A switch recovers with status bit $= 1$ ( $<S1>$ ) and therefore it cannot close a loop with status bits $= 0$. Now suppose a change in a switch setting at node $A_p$, caused by the receipt of a SETUP(id) message by the controller at $A_p$, closes a loop with status bits $= 0$. Note that if all controllers on the path from the originator to node $A_p$ had previously received the SETUP(id) message, then all switches on that path would have been set according to the selected path for that connection, which doesn’t contain loops, and thus the switch setting at $A_p$ could not have closed a loop. This means that at least one controller on the path from the originator to node $A_p$ did not yet receive the SETUP(id) message. Out of the nodes corresponding to these controllers, let $A_m$ be the closest to $A_p$ and let $A_{m+1}$ be the node next to $A_m$ on the path from $A_m$ to $A_p$. The controller at $A_m$ has reserved the connection and was up at that time (otherwise the SETUP(id) would not be sent), thus at that time the corresponding status bit was 1. Since $A_p$ closes a loop with status bits $= 0$, the status bit corresponding to the port at $A_m$ pointing to $A_{m+1}$ is now 0 and thus has changed in the interim from 1 to 0. This can happen only when the controller receives SETUP(id), but since $A_m$ did not receive the SETUP(id) of the connection under consideration, the SETUP(id) that changed the status bit of the port at $A_m$ pointing to $A_{m+1}$ must have been part of a different connection. However this cannot happen, because this would have required a reservation of that connection of the corresponding port at $A_{m+1}$, while in fact $A_{m+1}$ has reserved and setup the connection that goes through node $A_p$ on this port. Therefore the switch setting at $A_p$ cannot close a loop with status bits $= 0$. This proves $a$). Since from Lemma 1e), an entry in up necessarily has corresponding status bit $= 0$, part $b$) also follows.

The loop freedom property is important since loops with all status bits $= 0$ tie up resources that are never released. Slight variations in the setup protocol can cause loops, as
b) A loop connection section is a path of ports at nodes $A_1, A_2, \ldots, A_{n-1}, A_1$ corresponding to wavelengths $\lambda_1, \lambda_2, \ldots, \lambda_{n-1}$ such that (see Fig. 6):

- at node $A_i$, the $A_{i-1}, \lambda_{i-1}$ link port is connected to the $A_{i+1}, \lambda_i$ link port, for $i = 2, \ldots, (n-1)$
- at node $A_1$, the $A_2, \lambda_1$ link port is connected to the $A_{n-1}, \lambda_{n-1}$ link port

Figure 5: A linear connection section

Figure 6: A loop connection section
previously reserved by any other originator and there are no link/wavelength failures, the controller sends back RESERVE-ACK and sets state(id) = reserved, so \( a \) holds.

Consider the time after all controllers have put their corresponding entry into state reserved. Since the IDLE-TIMER’s do not expire, no TAKEDOWN is sent by the originator. In addition, no RESERVE-TIMER expires and therefore no controller will delete the entry while in reserved state. Thus all CST messages received in up state will contain related entries and thus no controller will delete the entry while in state up. Thus all entries stay in reserved or up states.

When all RESERVE-ACK’s arrive at the originator, the latter is still by assumption in state = reserved, since the RESERVE-TIMER did not expire and CST updates do not affect nodes with state(id) = reserved (\(<C3>\)). Thus it sends SETUP to all controllers on the path \(<H3>\). Each controller receives SETUP in state reserved and sets state(id) = up. Thus \( b \) holds.

When the SETUP arrives at each controller, the latter sends SETUP-ACK. When the originator receives the SETUP-ACK’s, it is still in state up, since the IDLE-TIMER does not expire, and therefore the originator starts using the connection and sends SETUP-CONFIRM to the destination. When the latter receives the SETUP-CONFIRM, it starts using the connection.

\[\square\]

Definition 3

\( a \) A linear connection section is a path of ports at nodes \( A_1, A_2, \ldots, A_{n-1}, A_n \) corresponding to wavelengths \( \lambda_1, \lambda_2, \ldots, \lambda_{n-1} \) such that (see Fig. 5):

- at node \( A_i \), the \( A_{i-1}, \lambda_{i-1} \) link port is connected to the \( A_{i+1}, \lambda_i \) link port, for \( i = 2, \ldots, (n-1) \)
- at node \( A_1 \), the \( A_2, \lambda_1 \) link port is either connected to the local I/O port or is disconnected
- at node \( A_n \), the \( A_{n-1}, \lambda_{n-1} \) link port is either connected to the local I/O port or is disconnected
Proof: The given id is assigned not more than once, and thus the originator sends no
more than one RESERVE(id) message to each controller, in <F4>. A controller enters state
reserved with the given id when it receives a RESERVE(id) message and thus can enter this
state at most once. This proves a), b).

The originator can send to each controller at most one SETUP(id) message, in <H3>. Thus c).

The entry is set to state up at the time when the switch is set for the corresponding
connection. The other time when an entry is set to up is if the controller recovers after a
failure, in which case each switch setting with status bit = 0 is translated into an entry in
state up (<A4>). This proves d). When the switch comes up, if the status bit = 1, the
entry, if it exists, is deleted (<S2>). When the state is set to up, the status bit of the
corresponding switch ports is set or has value = 0 (<A4>,<H4>,<J3>). This proves the
first part of e). If the controller comes up, the only entries that appear in the CST are those
in state up, corresponding to ports with status bit = 0 (<A4>). When an entry is deleted,
the corresponding status bit is changed to 1. Thus e).

Theorem 1 (Successful setup) Suppose that a controller receives a connection setup request
from the higher layer. Suppose that a route is found and no failures of links, wavelengths or
controllers occur along the route for sufficiently long time. Suppose that in addition no other
originator reserves the same wavelength on one of the links of the route before the considered
controller does and no RESERVE-TIMER or IDLE-TIMER expires. Then:

a) the wavelength is successfully reserved at all controllers along the connection (state(id) =
reserved).

b) the connection is set up at all controllers along the connection (state(id) = up).

c) both the originator and the destination will start using the connection.

Proof: By Lemma 1, we do not have to worry about a controller receiving twice the
RESERVE or SETUP messages. Since there are no controller failures, the RESERVE mes-
gages sent by the originator in <F4> arrive at each controller. Since the wavelength is not
6 Properties

In this section, we prove the properties of the protocol as stated in Sec. 1.

Recall that we have assumed in Sec. 2.1 that the connection-id is unique and no two connections carry the same id.

Definition 2

a) A switch setting is said to belong to the connection for which the setting was set.

b) An entry in state up is said to correspond to the switch setting that has been set when the entry entered up (\(<H5>,<J4>\) ) or to the switch setting for which the entry was entered in the CST when the controller came up (\(<A3>\) ).

Lemma 1 (Preliminary Properties) Consider a given connection with a given id and a given selected route. For the given connection

a) at most one RESERVE(id) message arrives at each controller.

b) each controller can be in reserved state at most once.

c) at most one SETUP(id) message arrives at each controller.

d) an entry in state up always contains the correct corresponding switch setting.

e) if an entry is in state up , the status bit at the corresponding switch ports is 0; if a status bit is 0, then if the controller is working, it has a corresponding entry in state up.
Actions related to a given connection id

F1  event: connection setup request from higher layer (originator)
F2    determine route and wavelengths
F3    assign entry and id
F4    send RESERVE(id) to each controller on the path
F5    state(id) = reserved
F6    start RESERVE-TIMER(id)
G1  event: RESERVE(id) received
G2    if wavelength is available on corresponding link (no entry in CST)
G3    send RESERVE-ACK(id)
G4    start RESERVE-TIMER(id)
G5    state(id) = reserved
G6    else
G7    send RESERVE-NACK(id)
H1  event: RESERVE-ACK(id) received (originator)
H2    if all RESERVE-ACK(id)'s received and state(id) = reserved
H3    send SETUP(id) to each controller on the path
H4    state(id) = up
H5    set switches
H6    set status bits corresponding to connected switch ports = 0
H7    start IDLE-TIMER(id)
H8    cancel RESERVE-TIMER(id)
I1  event: RESERVE-NACK(id) received
I2    send TAKEDOWN(id) messages to each controller on the path
I3    delete entry(id); corresponding status bit = 1
J1  event: SETUP(id) received
J2    if state(id) = reserved
J3    state(id) = up
J4    set switch
J5    corresponding status bits = 0
J6    send SETUP-ACK(id) to originator
J7    if destination
J8    set switch
J9    start IDLE-TIMER(id)
J10   cancel RESERVE-TIMER(id)
K1  event: SETUP-ACK(id) received (originator)
K2    if all SETUP-ACK(id)'s received and state = up
K3    start using connection
K4    send SETUP-CONFIRM(id) to destination
L1  event: SETUP-CONFIRM(id) received (destination)
L2    start using connection
M1  event: message received on connection(id)
M2    restart IDLE-TIMER(id)
N1  event: RESERVE-TIMER(id) expires
N2    delete entry(id); corresponding status bit = 1
N3    originator: send TAKEDOWN(id) messages to each controller on the path
O1  event: IDLE-TIMER(id) expires (originator or destination)
O2    delete entry(id); corresponding status bit = 1
O3    originator: send TAKEDOWN(id) messages to each controller on the path
Algorithm for each controller

Actions related to all connections
A1 event: controller comes up
A2 corresponding to each switch setting with status bit = 0
A3 enter entry in CST
A4 state(entry) = up : id(entry) = empty
A5 if switch setting points to local port: start IDLE-TIMER(entry)
B1 event: UPDATE-TIMER expires or change in CST
B2 send CST update
B3 start UPDATE-TIMER
B4 send topology update
C1 event: CST received from adjacent controller
C2 for each entry x in local CST that points to the link to the adjacent controller
C3 if state(x) = up and received CST does not contain related entry
C4 delete entry x; corresponding status bit = 1
D1 event: topology update received
D2 if timestamp higher than stored one
D3 update link utilization information
E1 event: adjacent link fails
E2 delete all entries in CST that use the link
E3 corresponding status bits = 1
Variables

- **id** - connection-id
- **CST** - connection switch table
- **state(entry)** - state of an entry
- **state(id)** - state of an entry with a given connection-id
- **entry(id)** - entry with a given id
- **status bit** - for each port in the router and local access switches

Initialization

- initialize CST to empty
- initialize topology database entries to list of adjacent links and no reserved ports.
- start UPDATE-TIMER

Figure 4: The finite state machine
with status bit $= 0$ correspond to working connections $<A3>, <A4>$. If the connection is in fact not up, the controller will learn about this via the CST update procedure.

In order to deal with the case when all controllers along a connection fail and then recover, the endpoints of a connection have a timer (IDLE-TIMER) that runs while receiving idle characters; if the timer expires, the entry corresponding to the connection is deleted $<O2>$, thereby triggering the deletion of entries at all other controllers via the CST update procedure. The IDLE-TIMER is started by the origin and the destination when each enters $up$ state and is restarted whenever a message is received on the connection.

While in $reserved$ state, the controller does not change the state of the entry in response to received CST updates $<C3>$. This prevents a controller from taking the reservation down in the case when it has reserved a wavelength while its neighbor is slow and has not done so yet. The RESERVE-TIMER runs in $reserved$ state to ensure that the reservation is released if the connection is not set up within reasonable time.

All CST entries corresponding to a link are deleted if the link fails $<E2>$. The entry corresponding to a port is deleted if the corresponding wavelength on that link fails. The CST update procedure ensures in both cases that the entries will be also deleted along the connection(s) that correspond to those entries.

## 5 The Code

The following code, reflecting the previous informal description of the protocol, is executed by each controller in the network. The finite state-machine of the protocol is given in Fig. 4.
and CST update. When there are no exceptions, either one will take the entire connection down. Each procedure handles a different scenario of exceptions.

The first procedure calls for the originator controller to send in parallel a TAKEDOWN message to each controller on the connection <P1>. Upon receiving the TAKEDOWN message, each controller deletes the entry corresponding to the connection in its CST <Q2>. The property of this procedure is that, even if some controllers on the connection are down, it takes the connection down at all working controllers. However, if the originator is down, the connection cannot be taken down by this procedure.

The second procedure is CST update, whereby periodically (when UPDATE-TIMER expires) and whenever there is a change in the CST <B1>, each controller sends separately to each neighboring controller a list of the wavelength, id and state of all connections that go over the link connecting them <B2>, as described in Sec. 2.2. When a controller receives the CST update message from a neighbor, it scans the entries in the local CST that point towards the link to that neighbor and looks for related entries in the received CST. If for a given entry in the local CST there is no related entry in the received one, the entry in the local CST is deleted <C4>.

The CST update procedure has the property that if any controller on the connection deletes the entry corresponding to a connection, the appropriate entries will be deleted at all controllers along the connection until the procedure meets a failed controller or the end of the connection. If a link fails, the controllers next to the failure will delete the entry corresponding to all connections that have traversed the failed link/wavelength on that link <E2>. Similarly, if a wavelength on a link fails, the local controller deletes the CST entry corresponding to the connection that traverses the failed wavelength <R2>. Then the CST update procedure will ensure that the entries corresponding to the connection will be deleted along the connection. Similarly, if the originator controller fails and the other end wants to take down the connection, it will simply delete the appropriate entry.

Note that since transmission of data on a connection does not employ the controller at intermediate nodes, failure of a controller after the connection is established does not require taking down the connection. The CST is lost upon the failure of the controller and at the time when the controller comes up again, it assumes that the switch settings connecting ports
the timer expires before it receives the SETUP message <N2>. The expiration time of the RESERVE-TIMER should be set such that it allows the reservation phase to finish at all nodes on one hand, while it allows cancellation if the setup process does not complete successfully within reasonable time. Note that it may be desirable to have redundant timers in case one timer fails.

If the reservation was successful at all nodes, then the setup procedure, described below, is started. The reservation may not be successful at all nodes even if the wavelengths were available on all links according to the originator’s topological database. This may happen if some of the controllers have in the interim received and accepted another reservation request for the same wavelength on one or more of the links of the path. In this case, the originator instructs all controllers, by TAKEDOWN messages sent over the controller network <I2>, to release the reservation.

If the reservation procedure was successful at all nodes, the originator sends SETUP messages to all controllers on the path <H3>. When a controller receives a SETUP message, if the CST contains an entry in state reserved with the same connection-id, it sets up the switches of the ports to be used by the connection, sets the status bits to 0, puts the entry into state up and sends a SETUP-ACK message to the originator controller <J2>-<J6>.

When the originator receives SETUP-ACK messages, it can start using the connection. It also sends a SETUP-CONFIRM message to the destination, informing the latter that it can start using the connection.

4 Takedown and Update

If a connection is not needed anymore or if a link or the wavelength on a link on the connection fails, the connection should be taken down and its resources released. From the control point of view, taking down a connection means updating the CST’s at all controllers along the connection by deleting the corresponding entry and setting the status bit to 1. The switch settings in the node remain the same until they are needed for new connections.

We employ two procedures for taking connections down, explicit TAKEDOWN messages
3 Setting Up Connections

The operations that need to be performed in order to establish a new connection between two nodes are the following:

- **Wavelength and Route Determination**: The originator node finds a sequence of ports with unused wavelengths that can be potentially connected to form a path to the destination. This can be deduced from the topology database and wavelength usage database, both of which are updated by the topology update protocol. The actual route computation algorithm used for this purpose is outside the scope of this paper; several possibilities exist [8, 9].

- **Reservation**: The originator node requests all controllers on the connection to reserve the selected wavelengths on all links of the path. This stage is necessary because without it, loops of committed resources may be formed, that are never released (see Lemma 2 and Sec. 7.1).

- **Connection setup/release**: If the reservation step is successful at all nodes in the path, switches are set by the corresponding controllers to accomplish the connection. Otherwise, the reservation is released at all nodes at which it was accomplished. Switch settings do not change upon reservation or release.

The reservation process is performed by the originator node sending messages in parallel to all other controllers on the path, requesting reservation of the corresponding wavelengths. The messages are sent over the communication network connecting the controllers. By sending messages in parallel, the originator minimizes the probability that this process will collide with another reservation process that may attempt to reserve the same wavelength on one or more links of the path. Each controller on the path sends a positive (RESERVE-ACK) or negative (RESERVE-NACK) response to the originator <G3>,<G7>. If the response is positive, the responding controller includes in the CST a corresponding entry in state reserved <G5>. No switch settings are changed at this point, since there is no point in setting the switches unless it is known that all reservations were successful. The controller uses the RESERVE-TIMER <G4> in state reserved and deletes the entry if
proach the controllers need to know the topology of the network and the usage of wavelengths on the network links. To this end, each controller keeps a map of the switch cross-connections in all nodes in the network. We employ a topology update protocol similar to the ones used in networks such as the ARPANET [22] or the plaNET high-speed network [18]. To achieve the topology update, we assume that there is a broadcast mechanism between the controllers over the communication network that connects them.

We will use a periodic topological update procedure [23] in which each node periodically broadcasts pertinent topology information to all the other nodes. In addition nodes can generate event-driven updates as well. The period between updates needs to be tuned depending on the the frequency of changes within the network. We shall assume that there is some mechanism, either sequence numbers or timestamps, that provides ordering information about update messages originating at a node. This is necessary to prevent looping of messages and allowing nodes to discard information from old (outdated) update messages [23].

The information in the topological update messages is generated from the node CST. A topological update message contains the list of up links adjacent to the node and their wavelength usage. The latter includes wavelengths used by connections that are in state up or reserved. It also includes wavelengths that may not be available due to failures. For example, node A with the CST of Fig. 2 generates update message \((B, \lambda_1, \lambda_2, \lambda_3), (C, \lambda_2, \lambda_3), (D, \lambda_1, \lambda_3)\).

Every node in the network maintains a topology database that reflects its knowledge of the nodes that are up, the highest timestamp of update messages received from each such node, the links that are up next to each node and their wavelength usage. Whenever a node receives an update message about another node with a later timestamp than the stored one, it updates the corresponding information. The topology information is used by nodes to decide the path of a new connection it intends to establish.

We do not discuss how node, link and wavelength failures are detected in the network. This is covered in [24]. However once a failure is detected, the nodes detecting the failure (which need not necessarily be adjacent to the failure), use the topology update protocol to distribute this information to the other nodes in the network.
<table>
<thead>
<tr>
<th>connection-id</th>
<th>port</th>
<th>port</th>
<th>state</th>
</tr>
</thead>
<tbody>
<tr>
<td>$B, \lambda_3$</td>
<td>$D, \lambda_1$</td>
<td>$I/O$</td>
<td>$up$</td>
</tr>
<tr>
<td>$D, \lambda_3$</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 3: The connection switch table at Node A after recovery

in the CST, with empty connection-id field $<A2>, <A3>, <A4>$. For example, if the controller at node A comes up and finds the switching node as in Fig. 1, with status bit = 1 on $(C, \lambda_3)$ and $(B, \lambda_1)$ and status bit = 0 on $(B, \lambda_3), (D, \lambda_1)$ and $(D, \lambda_3)$, it sets the CST as in Fig. 3. The CST will be updated using a CST update protocol as described in Sec. 2.2.

### 2.2 The Connection Update Procedure

The purpose of the CST update procedure is to update controllers as of the status of the connections that go through their switches. Periodically (when UPDATE-TIMER expires) and whenever there is a change in the CST $<B1>$, each controller sends separately to each neighboring controller a list of the wavelength, id and state of all connections that go over the link connecting them $<B2>$. For example, controller A with CST as in Fig. 2 sends to C the message $((\lambda_3, (B, C, 10), up), (\lambda_2, (E, C, 8), reserved))$. To neighboring controller D it sends the message $((\lambda_1, (E, D, 12), up), (\lambda_3, (C, A, 8), up))$. The actions taken by the neighboring controllers upon receiving the message will be described in Sec. 4.

### 2.3 The Topology Update Protocol

In order to establish a new connection, the originator controller must obtain a route for the connection. The route may be obtained from topology databases held at each controller, from a centralized database or from hierarchical ones. The present paper does not deal with the problem of deciding which approach is better for what type of networks.

For illustration purposes only, we briefly discuss the decentralized approach. In this ap-
fibers between nodes $A$ and $B$.

**Definition 1** Related entries in the CST’s of two neighboring controllers are entries of the same wavelength that point to the link connecting them. For example, for two neighboring controllers $A$ and $B$, an entry containing the port $B, \lambda_2$ that points to link $(A, B)$ for wavelength $\lambda_2$ in the CST of controller $A$ is related to an entry $A, \lambda_2$ in the CST at controller $B$.

Suppose a wavelength on one of the links fails. This could happen because the entire link fails, or because some component of the switch associated with this wavelength fails. When this wavelength is brought up again, its switch is set such that its link port is connected to the I/O. In order to indicate to the controller (that might be down too or go down) that the switch settings were not set by a previous connection setup procedure, we use status bits. To every port of each switch, we attach a status bit that can be altered by the controller. Whenever such a port recovers from a failure as described above, its status bit set to 1. When a port is connected to another port for the use of a connection during the connection setup procedure to be described in Sec. 3, the controller sets the status bits of those ports to 0. When a connection is taken down by the controller, the corresponding status bit is set to 1.

If a controller goes down, it normally loses its CST. When it comes back up, it initializes its table by reading the current switch settings in the switching node. Switch settings of ports with status bit = 1 are disregarded, while those with status bit = 0 generate an entry.
2 Maintaining State Information

Each controller must keep track of the connections flowing through its associated node. This is done via a connection switch table and connection update procedure, described below. In order to setup connections, the controllers must also keep track of the availability of wavelengths on each link in the network, which is done via a topology update protocol, also described below.

2.1 Connection Switch Table (CST)

Every connection is identified by a connection-id that consists of a triplet (originator, destination, sequence number). Two different connections may carry the same sequence number if they have different originators or destinations. The connection-id is assigned by the originator and we assume that the sequence number contains enough bits so that no wrap around occurs. Consequently, we can assume in our protocol that once a connection-id triplet is assigned, it will never be reassigned to another connection (in practice, it may be reassigned a very long time after the previous one was taken down, long enough to ensure that there is no trace of the old connection in the entire network). The connection-id is carried in all control messages related to the connection.

The controller keeps a connection switch table (CST) that indicates the settings of its switches, the connection-id of the connections that use them and their state. At a given node, a connection may be in state reserved or up. The controller sets the switches corresponding to a connection when its state changes from reserved to up. A sample table corresponding to the configuration of a particular node, say node A of Fig. 1 is given in Fig. 2. Note that for connection \((E, C, 8)\) that is in reserved state, the switch settings do not correspond to the path of the connection as reflected in the CST, but rather to some old connection. Port \(B, \lambda_2\) will be connected to port \(C, \lambda_2\) only when the connection goes to state up at node A.

Note that the CST is easily modified to accommodate networks with multiple fibers between nodes by assigning different labels the ports corresponding to the different fibers, e.g., \(B_1, \lambda_3\) and \(B_2, \lambda_3\) would represent connections using the same wavelength but on different
resulting message throughputs.

The main differences between our work and the related work relate specifically to the fact that the network is optical and transparent: each connection can carry data at a different bit rate and use a different protocol. This makes it difficult for nodes to monitor the state of a connection going through them since they cannot interpret the data flowing through. Secondly, the optical switches retain their state and continue to pass data through even if there is a failure that causes their control circuitry or the node controllers to fail. Thus when a controller recovers after a failure, it must determine whether the switch setting corresponds to a valid connection (one that was indeed set up earlier) or not. Moreover it cannot determine this by itself since it cannot monitor the data directly but must learn about this from other nodes.

The present work develops distributed protocols for setting up and taking down connections in a reliable and failsafe manner. The key features of the protocol as developed in the following sections and proved in Sec. 6 are the following:

a) The probability of reservation conflicts with other connection originators is reduced during the process of reserving resources by a connection originator. This enables connection originators to obtain resources with high probability.

b) Connections stay up even if controllers along their paths fail.

c) Controllers along the path of a connection have a consistent view of the connection state.

d) Connection resources are released at all nodes after takedown.

The rest of the paper is organized as follows. Sec. 2 describes the state information that is maintained in the network and how it is updated. Secs. 3 and 4 present the protocol for setup, takedown and update of connections. The pseudo code of the protocol is given in Sec. 5 and its properties are stated and proved in Sec. 6. Sec. 7 presents procedures for fast recovery of connections after the failure of a particular wavelength on a link, or of an entire link. Sec. 8 concludes the paper.
Wavelength-routed optical networks are being explored at the testbed level by several groups [1, 2, 6]. Lightpath routing algorithms have been explored in [7, 8, 9] and designing good virtual topologies has been studied in [10, 11, 12, 13, 14, 15].

Distributed control protocols have been developed for connection management of non-optical communication networks [16, 17, 18, 19]. While these protocols are not directly applicable to optical networks, some of their key ideas can be used to develop protocols for optical networks. There has been some recent work on distributed wavelength reservation schemes for optical networks [20, 21]. The focus in these papers is on proposing several different methods of reserving a wavelength to establish a connection and to study the
1 Introduction

A wide-area wavelength-routed optical network consists of switching nodes interconnected by wavelength-division-multiplexed (WDM) fiber-optic links, shown in Fig. 1. Each link carries traffic at $W$ wavelengths. At each node, a signal at a particular wavelength on an incoming link can either be received locally (if it is to be terminated at that node) or switched to another outgoing link and transmitted on a wavelength. The input/output wavelength mapping is dependent on the specific implementation of the node. If no wavelength conversion is provided inside the node, the outgoing wavelength must be the same as the incoming wavelength. With wavelength conversion inside the node, the outgoing wavelength can be chosen from a subset of the available wavelengths in general. Several implementations of these nodes have been proposed and demonstrated without wavelength conversion [1, 2, 3], and with wavelength conversion [4, 3].

The switches in a node are controlled by a controller. The controllers communicate with each other over some communication network, either out-of-band (for example, over an out-of-band TCP/IP network), or inband (for example, by dedicating a set of wavelengths within the network for this purpose), see [5]. We assume the existence of a reliable transport protocol in this network that ensures that messages between controllers are delivered reliably in sequence.

The network above allows us to set up connections between nodes in the network. A connection consists of a route in the network with a wavelength assigned on each link along the route. Two connections cannot be assigned the same wavelength on a given link. Each connection can carry data at Gb/s rates. Connections may be set up and taken down dynamically. Connections can be used to provide high-bandwidth pipes between nodes. Connections can also be used to realize a virtual topology, consisting of the nodes in the network with links in the virtual topology corresponding to connections in the actual network. Several connections are shown in Fig. 1. For example a connection between nodes $E$ and $B$ uses wavelength $\lambda_2$ on link $EB$, is switched to $\lambda_3$ at node $B$ and uses that wavelength on link $BA$ and uses $\lambda_1$ on link $AD$. Another connection between $A$ and $C$ must thus use a different wavelength $\lambda_3$ on link $AB$. 
Distributed Network Control for Optical Networks *

Rajiv Ramaswami and Adrian Segall

March 17, 1998

Abstract

This paper describes an architecture for controlling a wavelength-routed optical network. The optical network provides reconfigurable connections that can be used to carry different types of data, at possibly different bit rates. A connection consists of a path in the network and a wavelength on each link in that path. This work focuses on the mechanisms for controlling the optical connections. Distributed control protocols are provided for setting up and taking down connections reliably. These protocols allow connection originators to obtain resources with high probability by minimizing reservation conflicts, allow connections to stay up even if controllers along the path in the network fail, ensure that controllers in the network have a consistent view of the state of each connection, and ensure that all resources taken up by a connection are released once the connection is taken down. Fast protocols are also proposed for handling link failures and wavelength failures on a link. These protocols are general in that they can be used in networks with (a) different kinds of node architectures (electrical/optical switching, with or without wavelength conversion, etc.), (b) different route-computation algorithms that reflect the underlying constraints imposed by the nodes and the physical layer, and (c) any kind of underlying mechanism using which the network controllers can communicate with each other.

*This work was supported in part by grants MDA-972-92-C-0075 and MDA-972-95-C-0001 from ARPA. R. Ramaswami is with Tellabs Operations, Inc., Hawthorne, NY, rajiv@tellabs.com. A. Segall is with the Dept. of Computer Science, Technion, Haifa 32000, Israel, segall@cs.technion.ac.il. This work was done while the authors were with IBM T.J. Watson Research Center. A portion of this work appeared in Proc. Infocom'96.
Distributed Network Control for Optical Networks

by

R. Ramaswami and A. Segall

LCCN Report #9804
March 1998

Most of the technical reports of the Laboratory for Computer Communication Networking may be accessed on line through the Home Page of the CS Dept. The address is: http://www.cs.technion.ac.il/Reports/

For the elimination of any doubt, it is hereby stressed that the staff member and/or the Technion and/or the Technion Research and Development Foundation Ltd. will not be liable for any property damage and/or corporeal damage and/or expense and/or loss of any kind or sort that will be caused or may be caused to you or to anyone acting on your behalf, in consequence of this statement of opinion or this report, or in any connection to it.