Logical Topology Design for WDM Networks Using Survivable Routing

A. Jackel, S. Bandyopadhyay and Y. Aneja

University of Windsor
Windsor, Canada N9B 3P4
{arunita, subir, aneja}@uwindsor.ca

Abstract—Survivable routing of a logical topology ensures that the lightpaths are routed in such a way that a single link failure does not disconnect the network. However, even if the network remains connected after a failure, there is no guarantee that the resulting logical topology will be able to support the required traffic. In this paper, we introduce a new approach that integrates the logical topology design and survivable routing problems. When a topology is generated using our approach, it is guaranteed to have a survivable routing. We further ensure that the topology is able to handle the entire traffic demand, for any single link failure. We have formulated an ILP that optimally designs a survivable logical topology, and also proposed a fast heuristic which can be used for large networks.

Keywords- WDM networks; survivable routing; logical topology design

I. INTRODUCTION

In IP over WDM networks, a logical topology is mapped over an underlying wavelength routed WDM network. In such networks, a single physical link can support many logical links or lightpaths [1], and failure of even a single fiber may cause the logical topology to become disconnected. Therefore, it is vital that the lightpaths be routed in such a way that a single link failure does not disconnect the network. Such a routing is called survivable routing [2]. There has been considerable interest in this area, in recent years. In [2], authors determine the necessary and sufficient conditions for survivable routing and show that it is a NP-complete problem. Then, they present an integer linear program (ILP) to obtain a survivable routing for a given logical topology. In [3] an improved ILP formulation for computing the survivable routing is presented, as well as a tabu-search based heuristic. In [4], the authors present a fast local search algorithm for survivable routing and in [5] a heuristic, based on piecewise survivability, is presented. In [6], a survivable routing is found by partitioning the logical topology into several disjoint sub-cycles. [7] looks at the routing problem for WDM mesh networks with partial wavelength conversion and [8] shows that the problem is NP-complete, even when the logical topology is restricted to rings.

All the different approaches for survivable routing that are presented in the literature assume that a logical topology and the underlying physical topology are given. Therefore, the logical topology design problem is decoupled from the routing and wavelength assignment (RWA) problem. In this context, survivable routing can be viewed as a static RWA problem [9], where the set of lightpaths is known in advance. The logical topology design step determines the logical topology (i.e. set of lightpaths) that is capable of supporting a specified traffic demand. If this is completely decoupled from the RWA, there may not be any survivable routing for the given logical topology. Furthermore, survivable routing only ensures that a given logical topology remains connected, in the case of a single link failure. However, even if the network remains connected after a failure, there is no guarantee that the resulting logical topology will be able to support the required traffic.

In this paper, we introduce a new approach that integrates the logical topology design and survivable RWA problems. We assume that there are no wavelength converters available. Therefore, a lightpath must be assigned the same channel on each fiber it traverses, i.e. we follow the wavelength continuity constraint. Given an underlying physical network and traffic demand, our approach designs a logical topology that will always have a survivable routing over the physical topology. Furthermore, we not only ensure that the logical topology remains connected, but also that it is capable of supporting the entire traffic demand, for any single link failure. Our main contributions are as follows. i) first we develop an integer linear program (ILP) that determines a suitable logical topology and a survivable routing for that topology, given the traffic demand and the physical topology. The ILP also determines a feasible traffic routing over the logical topology ii) next, we develop a fast and efficient heuristic for survivable logical topology design. The heuristic can be used to quickly design a logical topology and a survivable routing for that topology, for large networks.

It is important to note that this scheme is not the same as the protection scheme since there are no backup lightpaths to replace primary lightpaths. This may be viewed as a scheme for guaranteed restoration where it is known that, in the case of a failure of any edge \( e \), the surviving logical topology will be able to support the traffic demand. When an edge \( e \) fails, we have to change the routing scheme so that the lightpaths that used edge \( e \)
are not used to carry any traffic. Our ILP and our heuristic determine the routing information for each edge failure which may be saved in a database for quick retrieval in the case of a failure.

The remainder of the paper is organized as follows. In Section II, we define our problem and present the ILP formulation. In Section III, we describe the heuristic approach. We discuss our experimental results in Section IV and conclude with a critical summary in Section V.

II. PROBLEM DEFINITION

A. Network Parameters

We are given the following information about the current state of the network:

- A physical fiber network G[N, E] with |N| = n, and |E| = m.
- A set of channels K that each fiber can accommodate.
- A set of R edge-disjoint paths, over the physical topology, between each source-destination pair. In this paper, we have used R = 3. The same assumption was made in [10].
- A set of potential logical edges (lightpaths) P. We assume that there is one potential (directed) logical edge from each source s to each destination d, so |P| = n(n-1).
- \( d_p^e = 1 \), if and only if the \( r \)th physical route for the \( p \)th potential lightpath uses link \( e \).
- \( s_q(d_q) \) = the source (destination) node for commodity \( q \).
- A set of commodities \( Q \). If there is a non-zero traffic requirement from a source node \( s_q \) to a destination node \( d_q \), then this source-destination pair corresponds to a commodity \( q \in Q \).
- A traffic demand matrix \( T = (t_q) \), where \( t_q \) is the amount of traffic corresponding to commodity \( q \in Q \).

We assume that \( t_q \) is expressed as a fraction of the capacity of a lightpath.

B. Integer Variables

The decision variables described in this section are binary integer variables. For each potential logical edge, we define three types of binary variables: the lightpath selection variable \( b_p \); the route assignment variable \( x_{pr} \); and the channel assignment variables \( w_{kp} \). The lightpath assignment variable determines whether or not a particular lightpath (logical edge) is included in the logical topology. The route assignment variable determines how a lightpath, if selected, is routed over the physical topology. The channel assignment variable assigns a single wavelength channel to each selected lightpath. These three types of variables are defined below.

- \( b_p = 1 \), if and only if the \( p \)th lightpath is included in the logical topology.
- \( x_{pr} = 1 \), if and only if the \( p \)th lightpath is included in the logical topology and the \( r \)th pre-computed route is used to route the lightpath.
- \( w_{kp} = 1 \), if and only if the \( p \)th lightpath is included in the logical topology and is assigned channel \( k \).

C. Continuous Variables

We define two types of continuous variables \( \delta_{kp}^e \), used for survivable RWA of the selected lightpaths, and \( f_{pq}^e \), used to route traffic over the logical topology. These continuous variables are defined below.

- \( \delta_{kp}^e = 1 \), if and only if the \( p \)th lightpath is routed over physical link \( e \), and assigned channel \( k \).
- \( f_{pq}^e \) = the amount of flow for the \( q \)th commodity, over the \( p \)th lightpath, if physical link \( e \) fails.

D. ILP Formulation

In our formulation, the objective is to minimize the number of lightpaths in the logical topology, as shown in (1).

\[
\text{Min } \sum_{p} b_p \tag{1}
\]

Since each lightpath requires one transmitter and one receiver, minimizing the number of lightpaths minimizes the cost of transmitters and receivers. Another possible objective may be to minimize the total number of wavelength-links. This can be easily done by changing the objective function to \( \text{Min } \sum_{e} \sum_{k} \sum_{p} \delta_{kp}^e \).

Subject to:

\[
\sum_{r} x_{rp} = b_p, \forall p \in P \tag{2}
\]

Constraint (2) ensures that if the \( p \)th lightpath is included in the logical topology (i.e., \( b_p = 1 \)), then it is allocated exactly one route over the physical topology. If \( b_p = 0 \), then the \( p \)th lightpath is not in the logical topology, and there is no need to allocate a route.

\[
\sum_{k \in K} w_{kp} = b_p, \forall p \in P \tag{3}
\]
Constraint (3) enforces the wavelength continuity constraint for a selected lightpath and ensures that exactly one wavelength is assigned to each selected lightpath.

\[ \sum_{q \in Q} f_{pq}^e \leq b_p, \forall p \in P, \forall e \in E \]  

(4)

Constraint (4) ensures that i) there can be a non-zero flow on the \( p^e \) lightpath only if that lightpath is selected i.e. \( b_p = 1 \), and ii) the total flow on a selected lightpath does not exceed the capacity of a lightpath.

\[ \sum_{q \in Q} f_{pq}^e \leq \sum_{r} x_{rp}, \forall p \in P, \forall e \in E \]  

(5)

Constraint (5) states that the total flow on the \( p^e \) lightpath, when link \( e \) fails is 0 (i.e., \( \sum_{q \in Q} f_{pq}^e = 0 \)), if the physical route assigned to the \( p^e \) lightpath includes link \( e \). In other words, if link \( e \) fails, then all traffic must be routed using only those lightpaths whose physical routes do not include link \( e \) (i.e., \( \sum_{r} x_{rp} \cdot d_{pe}^r = 0 \)).

\[ f_{pq}^e - \sum_{p: \text{from}(p) = i} f_{pq}^e = \begin{cases} t_q, & \text{if } i = s_q \\ -t_q, & \text{if } i = d_q \\ 0, & \text{otherwise} \end{cases} \]  

(6)

Constraint (6) gives the standard flow conservation constraint for each possible failure scenario. In (6) we consider each link \( e \) failures and ensure that the entire traffic demand can be routed over the surviving logical topology, even if link \( e \) fails.

\[ \sum_{r} x_{rp} \cdot d_{pe}^e + w_{kp} - \delta_{kp}^e \leq 1, \forall k \in K, \forall e \in E, \forall p \in P \]  

(7a)

\[ \sum_{r} x_{rp} \cdot d_{pe}^e - \delta_{kp}^e \geq 0, \forall k \in K, \forall e \in E, \forall p \in P \]  

(7b)

\[ w_{kp} - \delta_{kp}^e \geq 0, \forall k \in K, \forall e \in E, \forall p \in P \]  

(7c)

Constraints (7a) - (7c) are used to define the variable \( \delta_{kp}^e \). Constraint 7a states that if the \( p^e \) lightpath is routed over link \( e \) (i.e. \( \sum_{r} x_{rp} \cdot d_{pe}^e = 1 \)) and is assigned channel \( k \) (i.e. \( w_{kp} = 1 \)) then \( \delta_{kp}^e = 1 \). Constraint 7b and 7c state that if either \( \sum_{r} x_{rp} \cdot d_{pe}^e = 0 \), or \( w_{kp} = 0 \), then \( \delta_{kp}^e = 0 \). Together, (7a) – (7c) define the variable \( \delta_{kp}^e \), such that \( \delta_{kp}^e = 1 \) if and only if the \( p^e \) lightpath is selected and is routed over link \( e \) and assigned channel \( k \).

\[ \sum_{p \in P} \delta_{kp}^e \leq 1, \forall k \in K, \forall e \in E \]  

(8)

Finally, (8) takes care of the RWA for the selected lightpaths. Constraint (8) states that a channel \( k \) on a link \( e \) can be assigned to at most one lightpath.

III. HEURISTIC

In this section, we outline our heuristic for creating a logical topology, which can support the entire traffic demand for any single link failure. The first step in our design process is to create an initial logical topology, which is capable of supporting the required traffic under fault-free conditions. Then this initial topology is augmented by considering each potential single link failure scenario, and adding new lightpaths, if needed, to handle any disrupted traffic. An overview of our algorithm is given in Figure 1.

Figure 1. Overview of survivable topology design

1. success = CreateTopology(\( N, E, T, [\ ], LT_0 \))
2. If(success == 0) STOP and report failure
3. Repeat steps 4 – 11 for each \( e \in E \)
4. \( E = E - \{e\} \)
5. \( LP_{f} = \text{set of lightpaths in } LT_{0} \text{ which traverse link } e \)
6. \( LT_{\text{init}} = LT_{0} \cdot LP_{f} \)
7. \( C_{f} = \text{set of chains that includes a lightpath in } LP_{f} \)
8. \( T = \{ t_{q} \mid t_{q} \text{ is the part of traffic of commodity } q \in Q \text{ routed on a chain in } C_{f} \} \)
9. Repeat for each chain \( c \in C_{f} \) \{ \begin{align*}
q &= \text{commodity corresponding to chain } c \\
f_{q} &= \text{flow of commodity } q \text{ on chain } c \\
\text{reduce traffic on surviving edges of } c \text{ by } f_{q}
\end{align*} \}
10. success = CreateTopology(\( N, E', T, LT_{\text{init}}, LT_{\text{new}} \))
11. If (success == 0) STOP and report failure.

Else \( LT_{0} = LT_{\text{new}} \)

In our description we will use the term chain, used in the operations research community [11], to denote a...
directed path, in the logical topology, from a source node to a destination node. In step 1, the function CreateTopology is used to design the initial topology $LT_0$. This initial topology is able to handle the traffic requirements under normal fault-free conditions. CreateTopology also finds a feasible routing and wavelength assignment for each lightpath in the topology. This function takes, as input, an underlying physical topology (consisting of the set of nodes $N$ and the set of physical links $E$), a set of traffic requirements $T$, and an initial logical topology $LT_{init}$ specified as a set of logical links. In step 1, when the design process first starts, we set $LT_{init} = \emptyset$, an empty list. CreateTopology generates a new logical topology ($LT_{new}$), which is capable of handling the traffic requirements in $T$ and has a feasible RWA over $G = [N, E]$. If the design process is successful, the function returns a value of 1, otherwise it returns 0, indicating failure. If the initial topology can be created successfully (i.e., success == 1, in step 2), the design process continues, otherwise the algorithm stops and reports failure.

In steps 4 – 11, we consider, a specific failure scenario where link $e$ becomes faulty, and augment, if needed, the current topology, with new lightpaths to handle the traffic disrupted because link $e$ is now faulty. In step 4, we update the physical topology by removing the faulty link $e$. In step 5, we calculate the set of lightpaths $LP_i$ that are affected by failure of link $e$. In other words, $LP_i$ includes those lightpaths that traverse link $e$. In step 6, we create a temporary logical topology ($LT_{temp}$) by removing the affected lightpaths from $LT_0$. This is given as the initial topology when the function CreateTopology is called (in step 10) to augment the current surviving logical topology. In step 7, we calculate the set of chains $C_i$ that are no longer usable. These chains are disrupted due to the failure of link $e$ since each chain contains at least one lightpath $l_p^i$ such that $l_p^i \in LP_j$. In step 8, we determine the amount of disrupted traffic $t_q^i$ ($0 \leq t_q^i \leq t_q$) for each commodity $q \in Q$ that uses a chain in $C_i$, and create a new traffic demand $T' = t_q^i$ for commodity $q$. The traffic $T'$ is not handled by the existing logical topology and the routing scheme designed in step 1 for the fault-free case. Step 9 is repeated for each chain $c \in C_i$. Since link $e$ is now faulty, the traffic on each of the disrupted chains is no longer handled by the logical topology. For each surviving logical edge in a disrupted chain $c$, we reduce the traffic on the logical edge by the amount of flow $f_q$, where $f_q$ is the flow on chain $c$, under fault-free conditions. This increases the spare capacity on the surviving lightpaths, which may be used to re-route some of the disrupted traffic. In step 10, we call the function CreateTopology with the modified physical and logical topologies, and the traffic requirements $T'$ representing the traffic originally carried on chains that use edge $e$. If all the disrupted traffic can be handled successfully, by utilizing the spare capacities in $LT_{init}$ or by adding new logical edges, we move on to the next failure scenario. Otherwise, the algorithm fails.

CreateTopology (Figure 2) is the main function that is responsible for creating or augmenting a logical topology to accommodate a given set of traffic requirements. It is a greedy heuristic which considers each commodity $q \in Q$, in turn, and tries to route the corresponding traffic $t_q$ from $s_q$ to $d_q$. We consider the commodities in decreasing order of the traffic requirements $t_q$, i.e. we try to route the traffic corresponding to the highest value of $t_q$ first. When routing the traffic for a given commodity $q$, the objective is to route the traffic $t_q$ from $s_q$ to $d_q$, in such a way that the cost (i.e., the number of new lightpaths created) is minimized.

In step 1 (Figure 2), we first initialize some parameters and choose the highest traffic entry $t_q \in T$, to be processed. Then, we create $R$ edge-disjoint shortest paths, using Dijkstra’s algorithm [12], for each node pair $(i, j)$. The reason for this step is that, in case we need to create a new lightpath from node $i$ to node $j$, we will use one of these $R$ paths to route the lightpath. It is much more efficient to check the $R$ pre-defined paths, than to search all possible paths in the physical network [10].

CreateTopology ($N, E, T, LT_{init}, LT_{new}$)

1. Initialize parameters

   $t_q = \max \{ t_j \mid t_j \in T \}$

   route-found = 1

2. Create $R$ edge-disjoint routes for each node pair.
3. While ($t_q > 0$) and (route-found == 1) {

   4. ($x$, paths) = maxflow ($s_q$, $d_q$, $t_q$, $LT_{init}$)

   5. If ($x > 0$) then {

      If ($x > t_q$) then $x = t_q$

      update-flows ($x$, $paths$, $LT_{init}$)

      $t_q = t_q - x$

   } else {

      route-found = findRoute ($N, E, t_q$, $LT_{init}, LT_{new}$)

      $t_q = 0$

      $LT_{init} = LT_{new}$

   } 7. $t_q = \max \{ t_j \mid t_j \in T \}$

} 8. Return route-found

Figure 2. Overview of the CreateTopology function

In step 3, we consider each commodity in turn, until either i) all traffic $t_q \in T$ has been successfully handled or ii) the topology design process has failed (route-found == 0). Steps 4-7 describe how we handle the traffic requirement $t_q$ for a single commodity $q$. In step 4, we try to route the current traffic $t_q$, using existing lightpaths only. We use a well-known single commodity maxflow
algorithm [11] to determine the maximum amount of flow \( x \), corresponding to commodity \( q \), that can be routed using the existing topology. Function \textit{maxflow} returns the maximum possible flow \( x \) and the \textit{paths} along which the flows can be sent. If \( x > t_p \), we only need to send \( t_p \) amount of flow, otherwise we send the maximum possible amount \( x \) (step 5). After sending the required flow, we update the spare capacity of the appropriate lightpaths and reduce the requirement for commodity \( q \) by \( x \) (step 5).

If it is not possible to send any flow for commodity \( q \), using existing lightpaths (i.e., \( x = 0 \)), then we try to establish new lightpaths to handle \( t_q \) (step 6). The function \textit{findRoute} takes the physical topology \([N, E]\), the current logical topology \(LT_{\text{curr}}\) and the traffic \( t_q \) and generates a new logical topology \(LT_{\text{new}}\) which can handle \( t_q \). In step 6, we make the simplifying assumption that the entire traffic \( t_q \) from the source \( s_q \) to the destination \( d_q \) is sent over one single route only, and is not split over multiple paths. If \( t_q \) could be handled successfully, then the function \textit{findRoute} returns a value of 1. Otherwise, \textit{findRoute} returns 0, indicating failure. The function \textit{findRoute} is a breadth-first search that starts at the source node \( s_q \). In each successive iteration, the function finds a set of nodes \( N(t) \) that can be reached by adding one extra lightpath. After each iteration, the function checks if \( d_q \in N(t) \). If \( d_q \in N(t) \) after \( r \) iterations, then it means that the traffic \( t_q \) can be routed by adding \( r \) new lightpaths. Details of the \textit{findRoute} function are straight-forward and are available in [13].

IV. RESULTS

A. Number of Lightpaths

In this section, we present and analyze our experimental results. All experiments were carried out on a 900MHz SUN platform. The ILP solutions were generated using CPLEX 9.0 [14]. As expected, the ILP solutions quickly become computationally intractable, as the size of the network increases. We have generated optimal solutions, based on our ILP formulation, for smaller networks. For each experiment, we randomly generated a number of physical topologies, and traffic matrices. Table I shows a comparison of the average number of lightpaths required for logical topologies generated using the ILP and our heuristic (H1).

<table>
<thead>
<tr>
<th>No. of channels per fiber</th>
<th>No. of nodes</th>
<th>Number of lightpaths required</th>
<th>High traffic</th>
<th>Medium traffic</th>
<th>Low traffic</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>ILP</td>
<td>H1</td>
<td>ILP</td>
<td>H1</td>
</tr>
<tr>
<td>4</td>
<td>8</td>
<td>10</td>
<td>12</td>
<td>9</td>
<td>12</td>
</tr>
<tr>
<td></td>
<td>16</td>
<td>10</td>
<td>12</td>
<td>9</td>
<td>12</td>
</tr>
<tr>
<td></td>
<td>8</td>
<td>14</td>
<td>21</td>
<td>8</td>
<td>17</td>
</tr>
<tr>
<td></td>
<td>16</td>
<td>14</td>
<td>21</td>
<td>8</td>
<td>17</td>
</tr>
</tbody>
</table>

From Table I, we see that the ILP can provide significant improvements, particularly under high traffic loads. However, this approach is not feasible for larger networks.

Table II shows the results of our experiments for medium to large networks. For such networks, it was not possible to obtain optimal solutions in a reasonable amount of time, using ILP. Therefore, we have only reported the results from our heuristic. We tested networks of different sizes, ranging from 6 to 40 nodes. For each network size, we randomly generated 10 different physical topologies. Each topology was then tested with at least 15 different traffic matrices. The traffic matrices were categorized into three groups: high, medium and low traffic. A definition of the categories, along with the individual traffic matrices and physical topologies are available in [13]. Table II shows the average values obtained from the different experiments. In Table II, \( K \) represents the number of available WDM channels per fiber. If all experiments for a particular network size and traffic load failed, we have put a ‘x’ in the corresponding entry. Otherwise, we have put the average values for all successful designs. It is clear from Table II that, for successful designs, the value of \( K \) has no significant effect on the number of lightpaths. This is not surprising, since if we can realize a topology with \( L_p \) lightpaths for \( K=16 \), it should always be possible to do so for \( K=32 \). The minor variations in the averages are due to the fact that sometimes a design fails for \( K=16 \), but succeeds for \( K=32 \). These designs typically require more lightpaths than can be accommodated with \( K=16 \), leading to a slight increase in the average number of lightpaths for \( K=32 \).

<table>
<thead>
<tr>
<th>No. of nodes</th>
<th>Number of Lightpaths Required</th>
<th>High Traffic</th>
<th>Medium Traffic</th>
<th>Low Traffic</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>K=16</td>
<td>K=32</td>
<td>K=16</td>
<td>K=32</td>
</tr>
<tr>
<td>6</td>
<td>29</td>
<td>29</td>
<td>25</td>
<td>25</td>
</tr>
<tr>
<td>8</td>
<td>53</td>
<td>53</td>
<td>41</td>
<td>41</td>
</tr>
<tr>
<td>10</td>
<td>79</td>
<td>79</td>
<td>62</td>
<td>62</td>
</tr>
<tr>
<td>14</td>
<td>166</td>
<td>166</td>
<td>120</td>
<td>120</td>
</tr>
<tr>
<td>20</td>
<td>323</td>
<td>323</td>
<td>247</td>
<td>247</td>
</tr>
<tr>
<td>25</td>
<td>502</td>
<td>505</td>
<td>390</td>
<td>390</td>
</tr>
<tr>
<td>30</td>
<td>716</td>
<td>719</td>
<td>552</td>
<td>552</td>
</tr>
<tr>
<td>40</td>
<td>x</td>
<td>x</td>
<td>986</td>
<td>x</td>
</tr>
</tbody>
</table>

Figure 3 shows how the number of lightpaths required to implement a feasible logical topology varies with the size of the network and with traffic load. We see that, as the size of the network increases, the number of lightpaths required to implement a survivable topology also increases steadily. Also, for a given network size, more lightpaths are needed to accommodate a higher traffic demand.
Figure 3. Number of lightpaths versus size of network

B. Solution Time

Figure 4 shows how the solution time increases with the size of the network. We have found that, for both the ILP and the heuristic, the traffic load does not significantly affect the solution time. The main factor is the size of the network. For the ILP, the solution time increases very rapidly and it takes several hours to obtain a solution even for small network, with only 5 or 6 nodes. For the heuristic, the increase is much slower, varying from less than 1ms for the small networks to several seconds for the larger ones.

Figure 4. Solution time versus size of network

V. CONCLUSIONS

In this paper, we have introduced a novel ILP for logical topology design in WDM mesh networks, using survivable routing. Our main contribution is that we integrate the logical topology design and survivable routing problems, so that when a topology is generated using our approach, it is guaranteed to have a survivable routing. The logical topology remains connected for all possible single link failures. Furthermore, we guarantee that the surviving topology is not only connected, but is capable of handling the entire traffic demand for all single link failure scenarios. The ILP can be used to generate optimal topologies for small networks. We have also presented a fast and efficient heuristic, which can be used to generate survivable topologies for much larger networks. Experimental results demonstrate that the heuristic can provide reasonable solutions, but is much faster and more scalable.

REFERENCES