IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received February 12, 2020, accepted March 12, 2020, date of publication March 23, 2020, date of current version April 6, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.2982500

Efficient Integration of Logical Topology Design and Survivable Traffic Grooming for Throughput Enhancement in WDM Optical Networks

ASIMA BHATTACHARYA[1], MALABIKA SARDER[2], MONISH CHATTERJEE[3], (Member, IEEE), AND DIGANTA SAHA[4]

[1]TEOCO Software Pvt., Ltd., Kolkata 700091, India
[2]Samsung India Electronics Pvt., Ltd., Noida 201301, India
[3]Department of Computer Science and Engineering, Asansol Engineering College, Asansol 713305, India
[4]Department of Computer Science and Engineering, Jadavpur University, Kolkata 700032, India

Corresponding author: Monish Chatterjee (monish_chatterjee@yahoo.com)

ABSTRACT Optical networks that employ traffic grooming, merge several low-speed traffic streams onto a high-speed lightpath. A fiber cut (or a link failure) in such networks can disrupt a large number of lightpaths and can cause a huge amount of data loss. Designing survivability schemes to make the network withstand a link failure is thus vital. In this paper we address the problem of survivable traffic grooming in a static scenario. In a static scenario the primary objective is to maximize throughput with the available network resources. Since the problem is NP-Complete, we propose an efficient heuristic TATG (Throughput Aware Traffic Grooming) for the problem. TATG employs an efficient approach of integrating logical topology design and survivable traffic grooming with heuristics at sub-problem level. Our approach ensures that the logical topology carrying the traffic always stay connected in the event of a link failure. Time complexity analysis shows that our heuristic runs in polynomial time. We study two separate forms of survivability in heuristics TATG-SC (TATG -Survivability at Connection) and TATG-SL (TATG-Survivability at Lightpath) with three categories of traffic demand. Performance comparison with a well known heuristic shows that the proposed approach provides much better throughput performance when resources are scarce. The average percentage decrease in request blocking over all experiments obtained using TATG-SC and TATG-SL is 54.87% and 59.57% respectively.

INDEX TERMS Logical topology, survivability, survivable routing, traffic grooming, throughput aware, WDM networks.

I. INTRODUCTION

Optical Wavelength Division Multiplexed (WDM) networks with their huge transmission bandwidth forms the backbone network of the Internet [1]. Such networks use optical communication channels called *lightpaths* [2] to carry data in the form of encoded optical signal at a rate of 10 or even 40 Gbps. Compared to the huge bandwidth of a lightpath OC-768 (10 Gbps to 40 Gbps) in modern networks, individual requests for connections are typically for data streams at a much lower data communication rate, of the order of megabits per second (Mbps) such as OC-3 and OC-12. This huge mismatch between the capacity of individual lightpaths and the bandwidth requirements of individual traffic demands has led to development of *traffic grooming* techniques.

Traffic grooming techniques in WDM networks can be defined as a group of techniques for combining a number of low-speed data streams from users, to use the high capacity of lightpaths as efficiently as possible [3]. The sub-wavelength connections with their corresponding traffic demands may be either known in advance or may randomly arrive/depart and accordingly, traffic grooming is classified as *static* [26] or *dynamic* [10] and there has been considerable interest of the research community in both of these areas. The lightpaths in a WDM network can be considered as the edges of a *directed* graph $G(V, E)$, where the nodes in V are the end-nodes of the network. Such a graph is called the *logical topology* (also called the virtual topology) of an optical network and the edges of such a graph are called *logical edges* [2].

The associate editor coordinating the review of this manuscript and approving it for publication was Cristina Rottondi.

Establishment of a lightpath involves considerable cost. This is because not only a transmitter at the source and a receiver at the destination are required, but a wavelength or WDM channel is consumed in every fiber link in the lightpath route. But since a lightpath offers huge bandwidth, it is important to utilize the bandwidth offered by the existing logical topology $G(V, E)$ as efficiently as possible. Thus for a new request for connection with traffic demand t from source s to destination d, a search may be initiated to find a sequence of logical edges (lightpaths having sufficient capacity to accommodate demand t), connecting s and d. If such a directed path is found then the demand t can be routed over it. Such a directed path will be called a *logical path* from s to d in G. It may be noted that logical paths can be single-hop or multi-hop.

The traffic-grooming problem has the following four sub-problems, which are not necessarily independent [4]. Any solution to the problem must be able to efficiently address all of these.

a) *Topology Design sub-problem:* Determine the logical topology consisting of lightpaths for the given set of traffic demands,

b) *Routing sub-problem:* Determine the route of each lightpath by routing over the physical topology,

c) *Wavelength Assignment sub-problem:* Assign a channel (wavelength) in each link traversed by the lightpath route in the physical topology and

d) *Traffic Routing sub-problem:* Route the traffic over the logical topology i.e. make a choice of the logical path to use for each user traffic stream in a way that the total load of each edge in the logical path does not exceed the capacity of a lightpath.

Since it is extremely important to efficiently utilize the bandwidth offered by a lightpath, the problem of traffic grooming [3]–[5], [10]–[15], [17]–[19] and [21]–[28] has attracted a lot of research interest. Since a lightpath in a traffic-groomed optical network carries huge amount of data and 100 or more lightpaths may traverse a fiber, the disruption of this traffic due to a fiber cut (link failure), even for a short period of time is a serious event. Consequently some of the works have extended the problem of traffic grooming, by addressing the joint problem of survivable network design and traffic grooming [6], [8], [16], [20] and [24] known as the *survivable traffic grooming* (STG) problem. The traffic grooming problem is known to be NP-complete [5] and consequently the problem of STG remains NP-complete [6]. Thus researchers over the years have proposed heuristic solutions to the problem of traffic grooming with or without survivability.

Since networks resources are scarce, it is necessary that any heuristic solution to the traffic grooming problem with or without survivability must minimize the use of these resources. Consequently the objective of [10] and [17] is to minimize the number of transceivers (alternatively the number of lightpaths), the objective of [19] and [22] is to minimize the number of transceivers and the number of wavelengths, the objective of [5], [11] and [14] is to minimize electronic switching cost and the objective of [15] is to minimize the use of Add Drop Multiplexers (ADMs). The work [4] addresses minimizing of lightpaths, minimizing of wavelengths used and minimizing the number of traffic hops (delay), each through a separate heuristic. Given a specified number of resources, the works [6], [8], [12], [13] and [24] studies the problem of maximizing the number of accommodated traffic demands or throughput maximization.

Traffic grooming may be carried out using the *bifurcated* model adopted in [27] or the *non-bifurcated* model adopted in [4], [8] and [24]. In the non-bifurcated model each user traffic stream is routed over a single logical path. Thus in the non-bifurcated model the traffic demand corresponding to a user request becomes part of the total load of each lightpath in the selected logical path from its source to its destination. In the bifurcated model, the traffic stream corresponding to any request is divided into a number of data streams, each having lower data rates than that of the original traffic stream, and each data stream is carried by a distinct logical path from source to destination.

The bifurcated model is more resource efficient, but the non-bifurcated model provides many technical advantages [3], [8] and [24]. In our work, we do not consider splitting of traffic demands as most of the works on STG use the non-bifurcated model. Also a recent study [28] finds that if a lightpath carries a large amount of fine-granularity demands each with a very small wavelength fraction, the number of (de)-multiplexing ports increases considerably. We use the *non-bifurcated* model of traffic grooming in *static* traffic environment.

Due to negligibly small probability of the presence of multiple failures, most work on faults assumes that there can be only a single fault in the network [2], [8] and [24]. In this paper we focus on *single link fault*, the most common type of fault. The standard way of ensuring survivability from single link failure is to employ *dedicated path protection* (DPP) or *shared path protection* (SPP) [2]. If DPP is employed then for every working path, network resources are reserved in advance for a link disjoint backup path and no resource sharing is allowed among the idle backup paths. In SPP however, two or more backup paths are allowed to share network resources if and only if the corresponding working paths are mutually link disjoint, making SPP more resource efficient than DPP. Since survivable network design involves idle reservation of resources, it is customary for any scheme of survivability to minimize the use of network resources to the extent possible.

Another way of addressing survivability can be accomplished through the notion of *survivable routing of logical topologies* introduced in [7]. In this scheme, *routing and wavelength assignment* (RWA) of lightpaths are performed in a manner that the logical topology stays connected after any arbitrary single link failure. However, the work [7] does not consider routing of sub-wavelength traffic and so even if a survivable routing of the logical topology exists, there

is no assurance that existing logical topology will be able to support the required traffic. The authors in [8] and [24] enhanced this concept by introducing the traffic routing phase and propose the complete and combined problem of STG and *logical topology design* (LTD). We shall refer to the scheme in [24] which is an extension of the work [8] as STGLTD. Results in [24] show that STGLTD requires much lesser number of lightpaths and wavelengths than schemes such as [6], which use DPP and SPP for solving the said problem, suggesting STGLTD to be more resource efficient.

We use STGLTD to present a scheme of survivable network design that does not employ a backup path for every working path as in DPP or SPP. Our scheme is designed to achieve significant resource saving. This is because the logical topology is augmented with an additional lightpath if and only if the existing capacity of logical topology is unable to support the traffic stream of a connection, disrupted due to a link failure. Minimization of use of WDM channels or wavelengths can significantly improve the performance of any traffic grooming scheme [4], [19] and [22]. Our heuristic not only minimizes the number of wavelengths used per link to the extent possible, it also attempts to minimize the number of wavelengths in lightpath routes while restricting the number of traffic hops (delay) to the extent possible. To the best of our knowledge few heuristics for STG available in the literature have taken into consideration all the above mentioned parameters to address the problem.

We use the *non-bifurcated* model of traffic grooming in *static* traffic environment. We address STG of unicast traffic demands [6], [8], [16], [20] and [24] and not many-to-many traffic grooming [17], [19] and [22]. We propose both forms of survivability for our heuristic i.e. survivability at connection level and survivability at lightpath level. To the best of our knowledge few heuristics for STG available in the literature have taken into consideration all the above mentioned parameters to address the problem.

The rest of the paper is organized as follows. In sec II we provide a broad review of the previous works, in sec III we illustrate why a decision between single-hop and multi-hop grooming is important for STG, in sec IV we include our contributions, in sec V we provide the heuristic solution to the STG problem, in sec VI we provide the two forms of survivability studied in this work, in sec VII we analyze the time complexity of the proposed heuristic algorithm, in sec VIII we provide the performance comparisons and analyze our results and finally we present our conclusions in sec IX.

II. REVIEW WORK

There has been a great deal of research interest to study the problem of traffic grooming in optical WDM networks with or without survivability lately. An auxiliary graph model to address traffic grooming in heterogeneous networks (networks with nodes having variable number of transceivers, variable wavelength conversion capability and grooming capabilities and variable number of wavelengths in links) is proposed in [4] and the work also proposes three grooming policies for the network operator. The first one tries to minimize the number of lightpaths, the second tries to minimize the number of wavelength-links and the third tries to minimize the number of traffic hops. The authors [5] prove that the problem of minimizing switching cost in path networks during traffic grooming is NP-complete. The authors [6] propose heuristics for maximizing throughput during STG employing dedicated path protection and shared path protection with survivability at connection level in WDM mesh networks.

The problem of STG and the logical topology design problem are combined in [8] and [24] to design a scheme for grooming survivable traffic using much lesser number of backup paths than schemes that employ path protection. The authors further propose a heuristics for maximizing the traffic carried by the surviving logical topology. Static and dynamic versions of the problem, various grooming models and policies, grooming node architectures and design of grooming network is presented by authors in [9]. The work [10] propose a transceiver saving auxiliary graph model for dynamic traffic grooming in WDM mesh networks that can take care of two constraint: the wavelength constraint and the transceiver constraint. The traffic-grooming problem in WDM path, star and tree topologies with the objective of minimizing the network-wide electronic switching are considered [11].

The authors in [12] consider an alternate approach to dynamic traffic grooming. They consider designing a static logical topology in advance and then route randomly arriving user connections over it for avoiding frequent lightpath setup and teardown. Two problems are considered: minimizing resource usage constrained by traffic blocking requirements and maximize performance constrained by given resources. The problem of traffic grooming with the objective of maximizing the number of accommodated traffic demands in SONET/WDM networks with a given number of ADMs is addressed in [13]. The authors first prove that the problem is NP-hard and then propose an approximation algorithm for the problem. A transparent Wavelength Routing Switch (WRS) architecture named G+ is proposed in [14] that allow aggregation of traffic en-route over existing optical routes without undergoing expensive optical-electrical-optical (OEO) conversions. The authors propose optical routes called ''lightours'' such that traffic demands can partially use optical routes to reach their destination.

The problem of traffic grooming is addressed in [15] by considering a Min-Max optimization problem where the objective is to minimize the number of ADMs at the network node where the number of required SADMs is maximum over all nodes. Two dynamic restoration schemes for STG in WDM mesh networks that improve blocking probability and restoration performance are proposed in [16]. Given a set of many-to-many traffic requests, the work [17] addresses the problem of minimizing the cost of grooming the sessions by minimizing the number of transceivers used and addresses the problem for both splitting (nodes having optical splitting capability) and non-splitting networks. Given the blocking criteria in dynamic traffic grooming, the problem

of determining the minimum resources needed to meet this criteria is studied in [18]. The authors termed this problem as resource planning for the dynamic version of the problem.

The problem of designing and provisioning of WDM networks for many-to-many traffic grooming is addressed in [19] with the objective of minimizing the overall network cost (the number of transceivers and the number of wavelengths used). Based on different WDM node architectures, the authors propose four different WDM networks for many-to-many traffic grooming. A traffic grooming technique that use knowledge of connection holding times of traffic demands to utilize resources efficiently is proposed in [20]. The objective of the work is to design a stable logical topology that can accommodate a given set of traffic demands with specified setup and teardown times. A near optimal solution approach for multi-hop traffic grooming that takes less time and consumes lesser memory is proposed very recently [21] using branch-and-bound technique. The authors [22] study many-to-many traffic grooming with the objectives of minimizing the number of lightpaths and the number of wavelengths used and propose approximation algorithms as the problem is NP-hard.

Two dynamic lightpath rerouting algorithms namely least resources rerouting algorithm and load balanced rerouting algorithm that consider alternate routing and traffic grooming are proposed in [23] for all-optical networks. The algorithms initiate the rerouting procedure when a connection leaves and a lightpath is released and the lightpath to be rerouted is selected according to the respective objectives in the two algorithms. A new architecture is proposed in [25] for the optical nodes to support all-optical traffic grooming in Orthogonal Frequency Division Multiplexing (OFDM) based elastic optical networks. The work [26] proposes a static grooming policy that tries to reduce congestion on the logical topology in heterogeneous optical WDM networks. The work [27] proposes dynamic traffic grooming approaches which takes into account holding-times of connections and bandwidth availability. Authors consider splitting of connections into multiple sub-streams and then apply multipath routing and traffic grooming for better utilization of network resources.

In WDM networks, traffic demands containing a mixture of coarse and fine granularity can cause poor resource utilization of lightpaths and an excessive utilization of (de)-multiplexing ports in waveband switching. For example, if a coarse-granularity demand with a near-wavelength capacity is routed onto a lightpath, it will not be able to carry other demands on the other hand if a lightpath carries a large amount of fine-granularity demands each with a very small wavelength fraction, the number of (de)-multiplexing ports will increase considerably. The authors [28] call this gra-diversity problem. Furthermore, due to the large scale of the backbone, optical network has been divided into multiple domains to address scalability. The work [28] studies the gra-diversity problem and proposes a granularity-layered graph to facilitate traffic partition grooming in mixed granularity and multi-domain optical networks.

In [29] authors present heuristics for grooming node selection and dynamic traffic grooming in a scenario where nodes with grooming capability are sparse. The work [30] proposes a greedy iterative algorithm for aggregating scheduled traffic demands onto a lightpath in the electrical domain. A light trail is a unidirectional bus from a convener node to an end node. Authors [31] present a study on a number of traffic grooming strategies that use algorithms, architectures and schemes that requires a hierarchical structure on the physical topology. Excess capacity which usually exists in backbone networks to accommodate traffic fluctuations, can be efficiently managed through pre-provisioning when traffic is light, by re-provisioning of backup capacity and preventing the termination of pre-established resources [32].

A dynamic traffic grooming algorithm is proposed recently in [33] using an auxiliary graph model for fully utilizing the features of a light trail. The authors [34] recently propose two heuristics for dynamic traffic grooming of multicast sessions based on the bin packing method to reduce the number of wavelengths. In a recent work [35] the authors investigate the problem of traffic grooming using light trail in elastic optical and WiMAX networks.

III. DECISION FOR SINGLE-HOP AND MULTI-HOP GROOMING

The way logical topology is designed during traffic grooming can considerably affect the transmission delay and resource consumption. We illustrate this with a suitable example. Consider the 6-node physical topology shown in Fig.1a. Let each of the 7 links have a single wavelength channel λ_1. A connection request is of the form r_i: (s_i, di, t_i) where s_i is the source, d_i the destination and t_i the traffic demand in OC-n notation. Let $C = \{r_1$: (2, 1, OC-12), r_2: (1, 3, OC-12), r_3: (2, 3, OC-24), r_4: (2, 5, OC-12)$\}$ be the set of connection requests to be served and let the capacity of a lightpath be OC-48.

The request r_1: (2, 1, OC-12) is served by establishing a lightpath, routed along the shortest path $2\rightarrow1$ on the physical topology and using wavelength λ_1 in link (2, 1). This lightpath is represented by the directed edge $2\rightarrow1$ (logical edge) in the logical topology of Fig.1.b. After routing the traffic OC-12 of r_1, the capacity of the logical edge $2\rightarrow1$ is now OC-36. The request r_2: (1, 3, OC-12) is served in a similar way by establishing a lightpath, routed along the shortest path $1\rightarrow3$ on the physical topology and using wavelength λ_1 in link (1, 3). This lightpath is shown by the logical edge $1\rightarrow3$ in the logical topology of Fig.1.b. After routing the traffic OC-12, the capacity of the logical edge $1\rightarrow3$ is now OC-36.

For the request r_3: (2, 3, OC-24), no new lightpath is established as a logical path $2\rightarrow1\rightarrow3$ exits in Fig.1.b with each logical edge having a capacity of OC-36. So the traffic OC-24 of r_3 is routed along the logical path $2\rightarrow1\rightarrow3$ resulting in each logical edge $2\rightarrow1$ and $1\rightarrow3$ to have a residual capacity of OC-12. For the request r_4: (2, 5, OC-12), there is no logical path that connects nodes 2 and 5 and so a new lightpath has

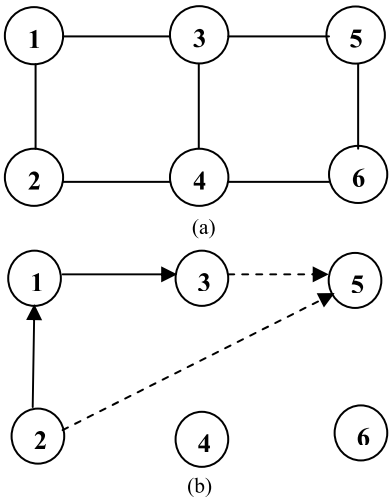

FIGURE 1. (a) Physical topology. (b) Logical topology.

to be established for serving r_4. There are two options to consider.

Option1: A lightpath may be directly established between 2 and 5, by routing along the shortest path 2→4→3→5 (or 2→4→6→5) in the physical topology requiring 3 units of λ_1 and then routing OC-12 of r_4 over it. The corresponding logical edge will then be the dashed 2→5 in Fig.1.b whose residual capacity will be OC-36. We shall call routes such as 2→4→3→5 (or 2→4→6→5) in the physical topology established directly between source and destination as *direct* routes.

Option2: A lightpath may be established between nodes 3 and 5, by routing along the shortest path 3→5 in the physical topology requiring only 1 unit of λ_1 and the corresponding logical edge will be then the dashed 3→5 in Fig.1b. The traffic amount OC-12 of r_4 can then be routed over the logical path 2→1→3→5. After routing the traffic, the logical edges 2→1 and 1→3 will exhaust their capacity while 3→5 will have a residual capacity of OC-36. We shall call routes such as 3→5 not directly established between the source and destination as *indirect* routes.

If option1 is used then the delay is just of a *single* hop but 3 units of λ_1 are consumed. However if option2 is used then there is a delay of 3 hops for the traffic of r_4 but a saving $(3 - 1) = 2$ units of λ_1. So a decision is required whether to choose a single-hop or a multi-hop path for routing traffic, keeping in mind the delay on one hand and the number of resources consumed on the other hand. The scheme employed in our proposed heuristic for STG, reduces consumption of wavelengths to the extent possible and on the other hand also attempts to limit the number of traffic hops.

IV. OUR CONTRIBUTIONS

Among all the works [3]–[35] that deal with the problem of traffic grooming, only a few [6], [8], [16], [20], [24] and [32] have studied the combined problem of traffic grooming and survivable network design (the STG problem). The STG problem is significantly important from the network operator's perspective but even recent papers [33]–[35] have addressed traffic grooming without considering survivability. In this paper we address the problem of STG in a static scenario.

Resources reserved for backup provisioning, causes network resources to be unavailable during normal fault-free operation. So any scheme for network survivability must ensure to minimize the use of network resources as much as possible. If the connection requests with their corresponding traffic streams are known in advance, then the objective of any strategy for STG should be to maximize throughput (the number of accommodated traffic demands) to the extent possible with the available number of resources. Since wavelengths are considered to be primary resource for lightpath establishment, throughput can be maximized by minimizing wavelength consumption.

The work [24] an extension of [8] shows that STGLTD results in significant resource saving compared to schemes that employ path protection (DPP and SPP). The heuristic in [24] is designed to maximize throughput by efficiently utilizing the bandwidth of the existing logical topology. The authors however, do not consider employing sub-problem level heuristic for STGLTD. Further the heuristic in [24] is designed to address survivability at connection level only. In this paper we investigate whether the throughput obtained using STGLTD can be improved upon further by proposing efficient heuristics at the sub-problem level and by employing the other form of survivability namely survivability at lightpath level. It must be noted that we do not include any Integer Linear Program (ILP) formulation for the problem. This is because an ILP formulation for maximizing the total traffic groomed is presented in [24]. So we only compare our results with the optimal values obtained from the ILP. We contribute by proposing and studying the following.

a) Proposing an efficient heuristic which can save significant network resources and consequently enhance throughput. The heuristic saves resources by adding extra lightpath(s) to the logical topology if and only if the existing capacity of the logical topology is insufficient to carry additional traffic.

b) Proposing an efficient heuristic RWA strategy for survivable routing of logical topology. The proposed RWA strategy leads to establishment of lightpaths in a way that reduces the number of wavelengths consumed in every link to the extent possible.

c) Proposing an efficient traffic routing strategy that makes a decision on the choice of logical path whenever an additional lightpath is to be included in the existing logical topology. Our strategy reduces the consumption of wavelengths to establish new lightpaths with restriction on the number of traffic hops (delay).

d) Extensively studying the throughput performance of our heuristic for three different categories of traffic demand and two types of survivability, namely survivability at connection and survivability at lightpath.

V. HEURISTIC FOR SURVIVABLE TRAFFIC GROOMING

We present our heuristic *TATG* (Throughput Aware Traffic Grooming). In our heuristic, RWA of lightpaths are carried out in a manner that a single link failure will not be able to disconnect the logical topology. It must be noted that our heuristic does not reserve resources for dedicated or shared backup paths to replace working paths in the event of a link failure. This saves the setup time and wastage of resources for backup lightpaths. Our heuristic ensures that the generated logical topology will always have a survivable routing over the physical topology.

The proposed heuristic can be thought of as a two-step method. In the first step, the aim is to generate a logical topology for supporting the given set of connections under a fault-free scenario. In the second step, the aim is to perform a survivable routing of the generated logical topology in a manner that any single link failure will not cause the network to be disconnected. This is carried out by considering the potential failure of each physical link in turn and adding extra lightpath(s), if and only if the current capacity of the logical topology due to the failure is not sufficient to accommodate the disrupted connections. The second step proceeds if and only if the first step succeeds.

The set of connection requests to be served is sorted in decreasing order of traffic demand. We use the well-known *FF* (First-Fit) wavelength assignment policy for assigning wavelengths to lightpaths. We assume 1) full-wavelength conversion capability in nodes, 2) all links to have same capacity in terms of the number of wavelengths and 3) sufficient number of wavelength converters and transceivers to be present at the nodes so that no lightpaths block due to want of these resources. In our scheme connection requests can get blocked if and only if wavelengths are not available. Table 1 provides the various notations used in *TATG* and their meanings. Given a physical topology $G_P(V_P, E_P)$, a matrix $Cost\,[1{:}\,|E_P|]$ is used where $Cost\,[i]$ denotes the cost in terms of the number of wavelengths consumed in link i. Initially $Cost\,[i] = 0, \forall i,\ 1 \le i \le |E_P|$. A matrix $CS\,[1{:}\,|E_P|][1{:}\,W]$ is used to denote the current status of wavelengths in a link. The value of $CS\,[i][j]$ can be either 1 or 0 and denotes that wavelength j is available or not available respectively in link i. Initially $CS\,[i][j] = 1\ \forall i, j,\ 1 \le i \le |_P|,\ 1 \le j \le W$. *TATG* is run both in a) *non-blocking* condition i.e. with sufficient link capacity (sufficient number of wavelengths in links) and b) *blocking* conditions i.e. with limited link capacity. We next describe our heuristic whose details are provided in Fig.2.

Inputs: The inputs are G_P, W, $Cost$, CS.

Outputs: a) In non-blocking conditions the outputs are the generated survivable logical topology $G_L(V_L, E_L)$, R and W_{min}. b) In blocking conditions, the output is *NOS*, the throughput measured as the number of accommodated or satisfied connections.

Method: The heuristic has access to the global variable C, the set of connection requests to be served. Two variables *NOF* and *NOS* are used to record the number of failures and the number of success respectively. First, the procedure

TABLE 1. Notations used in *TATG*.

Symbol	Meaning
$G_P(V_P, E_P)$	Physical topology with its vertex set and edge set.
C	A global variable that denotes the set of connection requests to be served, sorted in decreasing order of traffic demand.
W	Link capacity or the number of wavelengths per link.
$G_L(V_L, E_L)$	Logical topology with its vertex set and edge set.
R	Routing of G_L in G_P.
W_{min}	Minimum link capacity required to support the given set of connections C.
NOS	Number of success or number of accommodated connections.
NOF	Number of failures or number of blocked connections
$Cost\,[1{:}\,\|E_P\|]$	A vector which represents the cost in terms of the number of wavelengths consumed in each link.
$CS[1{:}\,\|E_P\|]\,[1{:}\,W]$	A matrix which represents the current status of wavelengths or the number of wavelengths currently available in each of the $\|E_P\|$ links.
N_{cf}	Number of connection that could not be restored for the failure of a physical link.

```
Heuristic TATG (G_P, W, Cost, CS)
begin
// the heuristic has access to the global variable C.
1  NOS ← 0;  NOF ← 0;
2  (G_L, R, success, CS, Cost) ← genInitTopology(G_P,
                                   CS, Cost, W);
3  if (success = TRUE)  // success in creating initial
                        //topology
4      for each e ∈ E_P do
5      begin
6        (G_L, R, CS, Cost, N_cf) ← findSurvPaths(G_P,
                                   e, G_L, R, CS, Cost, W);
7        NOF ← NOF + N_cf;
8      end for
9      if (NOF = 0)     // non-blocking condition
10         W_min ← getMax(Cost);
11         return (G_L, R, W_min);
12     else             // blocking condition
13         NOS ← |C| - NOF;
14         return (X, X, NOS);
15     end if
16 else                 // failure in creating initial topology
17     return (X, X, X);
18 end if
end heuristic
```

FIGURE 2. Heuristic *TATG*.

genInitTopology() is used (line 2, Fig.2) to obtain the logical topology G_L and its routing R on the physical topology generated initially under fault-free conditions. The procedure

also returns *success*, a Boolean variable whose value indicates success or failure in creating the initial logical topology. The procedure also returns the updated *CS* and *Cost*.

If *success* equals FALSE, signifying failure in creating the logical topology under fault-free condition then the heuristic proceeds no further. If *success* equals TRUE, then failure scenario of each fiber-link $e \in E_P$ is considered in turn. For every such failure scenario, the procedure *findSurvPaths*() is used (line 6) to obtain the survivable routing of the affected logical topology to accommodate the disrupted connection requests. The procedure *findSurvPaths*() returns the new G_L, the new R, updated *CS*, updated *Cost* and the number of failures N_{cf}. These new and updated records are input to the procedure when the next failure scenario is considered. The value of N_{cf} is added to *NOF*. In non-blocking conditions, the number of connection failures N_{cf} returned is always zero. So in non-blocking condition the value of *NOF* is 0 and the heuristic returns G_L, R and W_{min}. Note that W_{min} is the maximum value of *Cost* [i] over all $1 \leq i \leq |E_P|$. This value is obtained using procedure *getMax*() (line 10). In blocking condition, the heuristic returns X, X and *NOS*. Here 'X' denotes failure in creating survivable logical topology and its survivable routing. If *success* is 0, then the heuristic returns X, X and X and stops.

We next provide the description of the procedures *genInitTopology*() and *findSurvPaths*() whose details are provided in Fig.3 and Fig.4 respectively. The description of these procedures is essential for understanding the strategy used in *TATG*.

A. PROCEDURE genInitTopology

This procedure is used by *TATG* to generate the initial logical topology under fault-free condition. The procedure has access to the global variable C, the set of connection requests to be served. The procedure begins by initializing the variable *success* to TRUE and initializing the empty logical topology G_L. Every connection request $r_i \in C$, procedure *findPath*() is used (line 5, Fig.3) for finding a logical path connecting s_i and d_i in G_L such that the logical edges will be capable of accommodating t_i. If such a path *lpath* is found, then t_i is routed over it and the logical topology is updated. This is done by using procedure *updateTopology*() (line 15) which returns the updated G_L and R. If *lpath* is NULL or not found, then a new lightpath has to be set up. The procedure *searchRoute*() is used (line 7) to find a lightpath route rt_{min} (a route having minimum number of physical hops) and the wavelength list WL_{min} for assignment.

If WL_{min} is empty, signifying failure of RWA, *success* is set to FALSE to indicate failure in generating the initial logical topology. In case of non-empty WL_{min}, procedure *updateTopology*() is used (line 12) for establishing a new lightpath and updating G_L and R. The procedure *genInitTopology*() returns the generated initial logical topology G_L, the routing R of G_L, the value of *success* and the updated *CS* and *Cost* to *TATG*. The description of procedures *findPath*() and *updateTopology*() is provided later and the details of *searchRoute*() is shown in Fig.5.

```
Procedure genInitTopology (G_P, CS, Cost, W)
begin
// the procedure has access to the global variable C.
1   success ← TRUE;
2   initialize G_L by setting V_L ← V_P and E_L ← Φ;
    R ← Φ;
3   for each request r_i ∈ C do
4   begin
5     lpath ← findPath(G_L, r_i);
6     if (lpath = NULL)
7         (v_f, rt_min, WL_min, CS, Cost) ←
                searchRoute(G_L, G_P, r_i, CS, Cost, W);
8         if (WL_min = Φ) // no spare wavelengths in
                          // links of rt_min
9             success ← FALSE;
10            return (NULL, Φ, success, NULL,
              NULL);
11        else          // links of rt_min have spare
                        // wavelengths
12            (G_L, R) ← updateTopology(G_L, R, r_i,
                         rt_min, v_f, WL_min);
13        end if
14    else                      // lpath is not NULL
15        (G_L, R) ← updateTopology(G_L, R, r_i, lpath,
                         NULL, Φ);
16    end if
17  end for
18  return (G_L, R, success, CS, Cost);
end procedure
```

FIGURE 3. Procedure *genInitTopology*.

B. PROCEDURE findSurvPaths

The procedure (in Fig.4) is used by *TATG* to deal with the failure scenario of every link $e \in E_P$. Since a lightpath carries the traffic demand of one or more connections, a link failure in the lightpath route causes all connections accommodated by the lightpath to get disrupted. The procedure identifies the set of lightpaths L_e that can be interrupted due to the failure of link e and also identifies the set of connection requests C_e that can get disrupted. A new logical topology G_L' is formed with the logical edges corresponding to the lightpaths in L_e removed. The procedure then tries to restore each connection $r_j \in C_e$ by utilizing the available bandwidth of G_L'. Extra lightpaths are added to G_L' if and only if the existing capacity is insufficient to accommodate the traffic of a disrupted connection. In presence of resource restrictions the procedure may fail to restore a connection $r_j \in C_e$. The number of connection failures is recorded in N_{cf}.

Since a lightpath route can traverse one or more links in G_P, it is necessary that a connection, which could not be restored for a certain link failure scenario is not considered for restoration in subsequent link failure scenarios. The procedure ensures this by *marking* connection r_j as "FAILED" (line 14, Fig.4). Connection requests are assumed to be global

Procedure *findSurvPaths* (G_P, e, G_L, R, CS, $Cost$, W)
begin
1 $L_e \leftarrow$ {all lightpaths traversing edge e};
2 $C_e \leftarrow$ {all *unmarked* connection requests carried by the lightpaths in L_e};
3 create a modified physical topology G_P'=(V_P, E_P') where $E_P' \leftarrow E_P - \{e\}$;
4 sort connections in C_e in *decreasing* order of traffic demand;
5 create a modified logical topology $G_L' = (V_L, E_L')$ where $E_L' \leftarrow E_L$ – {edges corresponding to lightpaths in L_e};
6 $N_{cf} \leftarrow 0$;
7 **for** each request $r_j \in C_e$ do
8 **begin**
9 $lpath \leftarrow findPath(G_L', r_j)$;
10 **if** ($lpath$ = NULL)
11 $(v_f, rt_{min}, WL_{min}, CS, Cost) \leftarrow searchRoute(G_L', G_P', r_j, CS, Cost, W)$;
12 **if** ($WL_{min} = \Phi$)
13 $N_{cf} \leftarrow N_{cf} + 1$;
14 mark r_j as "FAILED";
15 **else** // the links in rt_{min} have spare // wavelengths
16 $(G_L', R) \leftarrow updateTopology(G_L', R, r_j, rt_{min}, v_f, WL_{min})$;
17 **end if**
18 **else** // $lpath$ is not NULL
19 $(G_L', R) \leftarrow updateTopology(G_L', R, r_j, lpath,$ NULL, $\Phi)$;
20 **end if**
21 **end for**
22 $E_L \leftarrow E_L' \cup$ {edges corresponding to lightpaths in L_e};
23 generate $G_L = (V_L, E_L)$;
24 **return** (G_L, R, CS, $Cost$, N_{cf});
end procedure

FIGURE 4. Procedure ***findSurvPaths***.

Procedure *searchRoute* (G_L, G_P, r, CS, $Cost$, W)
begin
1 $minCost \leftarrow getMin(Cost)$;
2 $V_a \leftarrow$ {all vertices v such that $(s, v) \in E_L$ with sufficient capacity to accommodate t};
3 $found \leftarrow$ FALSE; $v_f \leftarrow$ NULL;
4 $S_{RW} \leftarrow \Phi$; $rt_{min} \leftarrow$ NULL; $WL_{min} \leftarrow \Phi$;
5 **while** ($found$ = FALSE **and** $minCost < W$) do
6 **begin**
// find the indirect routes that connects $v \in V_a$ and // destination d of r.
7 **for** each $v \in V_a$ do
8 **begin**
9 $(rt_{ind}, WL_{ind}, CS) \leftarrow getRWL(v, G_P, r, CS, Cost, minCost)$;
10 $S_{RW} \leftarrow S_{RW} \cup \{(rt_{ind}, WL_{ind})\}$;
11 **end for**
// find the direct route connecting s and d of r
12 $(rt_d, WL_d, CS) \leftarrow getRWL$(NULL, G_P, r, CS, $Cost$, $minCost$);
13 **if** ($S_{RW} = \Phi$ **and** $WL_d = \Phi$) // no route is // found
14 $minCost \leftarrow minCost + 1$;
15 **else** // route is found
16 $(rt_{min}, WL_{min}) \leftarrow findMin(S_{RW}, \{(rt_d, WL_d)\})$;
17 $found \leftarrow$ TRUE;
18 $Cost \leftarrow incrCost(rt_{min}, Cost)$;
19 $CS \leftarrow update(rt_{min}, WL_{min}, CS)$;
20 **if** $((rt_{min}, WL_{min}) \in S_{RW})$ // route found // is indirect
21 $v_f \leftarrow getFirstNode(rt_{min})$;
22 **end if**
23 **end if**
24 **wend**
25 **return** (v_f, rt_{min}, WL_{min}, CS, $Cost$);
end procedure

FIGURE 5. Procedure ***searchRoute***.

variables and so this *marking* is available for subsequent failure scenarios. The set C_e thus contains only the *unmarked* connections that may be disrupted. Note all connections are initially *unmarked*.

The connections requests in C_e are first sorted in the order of decreasing traffic demand. The procedure handles the restoration of an unmarked $r_j \in C_e$ in a way similar to how connections are handled in *genInitTopology*() with the following exceptions. Procedure *findPath*() is used (line 9, Fig.4) to find a logical path in G_L' instead of G_L, procedure *updateTopology*() is used (lines 16 and 19) to update G_L' instead of G_L and procedure *searchRoute*() is used (line 7) to find a lightpath route in a modified physical topology G_P' with link e removed. Since the lightpaths in L_e should be available for subsequent failure scenarios, after handling a failure scenario the edges corresponding the lightpaths in L_e are added to obtain G_L from G_L'. Note that G_P' is not restored as the failure scenario is handled locally and does not affect the original G_P. Procedure *findSurvPaths*() returns updated G_L, R, CS, $Cost$ and N_{cf}.

C. PROCEDURE findPath

The procedure is used by both *genInitTopology*() and *findSurvPaths*() to find the logical path with sufficient capacity to route the traffic of a connection request. If the required logical path ($lpath$) is found then it is returned and if not found NULL is returned. We use a modified form of Breadth First Search (BFS) [36], not only capable of traversing the logical

topology (G_L or G_L') but also having the capability of finding logical paths with sufficient spare capacity. Since BFS is employed, the shortest logical path connecting s_i and d_i of r_i having capacity sufficient to accommodate t_i if available is always returned. This ensures that single-hop grooming is always tried first and only if it fails, multi-hop grooming is used. This results in restricting the number of traffic hops. We do not include the details of this procedure.

D. PROCEDURE searchRoute

The details of this procedure are shown in Fig.5. The notations used and their meanings are included in Table 2. The procedure *searchRoute*() is used by both *genInitTopology*() and *findSurvPaths*() whenever a new lightpath is to be included in the logical topology. The procedure implements the decision of whether to employ single-hop or multi-hop grooming when the existing capacity of the logical topology is insufficient to accommodate additional traffic. Employing multi-hop grooming can lead to increase in delay, but in our case if single-hop grooming is unsuccessful, the number of traffic hops is restricted to *two only*. If a new lightpath has to be included in the logical topology, RWA has to be performed. The objective of this procedure is to select the route of a new lightpath in a way that the consumption of wavelengths per link is minimized to the extent possible, the number of wavelengths used for establishing the new lightpath is also reduced to the extent possible while restricting the number of traffic hops.

TABLE 2. Notations used in *searchRoute*.

Symbol	Meaning
R	A connection request.
s	Source node of r.
d	Destination node of r.
T	Traffic stream of r.
$minCost$	Minimum value in $Cost$ [1:$\|E_P\|$].
V_a	Set of all vertices v in G_L such that $(s, v) \in E_L$ with sufficient capacity to accommodate t.
rt_{ind}	Indirect route connecting a vertex $v \in V_a$ and d in G_p.
WL_{ind}	Set of wavelengths for assignment to rt_{ind}, one for each link in rt_{ind}.
rt_d	Direct route connecting s and d in G_p.
WL_d	Set of wavelengths for assignment to rt_d, one for each link in rt_d.
S_{RW}	Set of (rt_{ind}, WL_{ind}) pairs.
$Found$	Boolean variable that indicates success or failure in finding the required lightpath route.
v_f	First node of an indirect route.
rt_{min}	The route with minimum number of physical hops
WL_{min}	Set of wavelengths corresponding to rt_{min}, one wavelength for each link in rt_{min}.

The inputs to the procedure are $G_L, G_P, r, CS, Cost, W$. The $Cost$ vector represents the cost incurred in the links so far due to wavelength consumption. The procedure *getMin*() (line 1, Fig.5) is used to perform a search of $|E_P|$ values in $Cost$ to find $minCost$, the minimum value of cost in the $Cost$ vector. The vertex set V_a is initialized with all vertices v, such that $(s, v) \in E_L$ with capacity sufficient to accommodate t. The Boolean variable *found* is used to indicate success or failure in finding the required lightpath route, initially it is FALSE.

The procedure *searchRoute*() works if and only if $minCost$ has not exceeded W and *found* is FALSE (line 5). For each vertex $v \in V_a$, the procedure *getRWL*() (line 9) is used to obtain an *indirect* route rt_{ind} connecting v and d, constrained by $minCost$, satisfying the objective of *searchRoute*(). The set of wavelengths WL_{ind} for assignment to rt_{ind} is also computed by *getRWL* Pairs of the form (rt_{ind}, WL_{ind}) are added to S_{RW}. The procedure *getRWL*() is then used (line 12) to obtain a *direct* route rt_d connecting s and d constrained by $minCost$, satisfying the objective of *searchRoute*(). The set of wavelengths WL_d for assignment to rt_d is also obtained.

The procedure *findMin*() (line 16) is then used to obtain the pair (rt_{min}, WL_{min}) out of all pairs $S_{RW} \cup \{(rt_d, WL_d)\}$ such that the number of physical hops in the route is minimum. If the *direct* route rt_d and an *indirect* route in S_{RW} has the same minimum number of hops, then the pair (rt_{min}, WL_{min}) returned by *findMin*() is (rt_d, WL_d) so that the number of traffic hops is kept to a minimum. The variable *found* is set to TRUE indicating successful route finding. Procedure *incrCost*() (line 18) is used to update $Cost$ by incrementing the cost value, $Cost[i]$ for each link $i \in rt_{min}$. The procedure *update*() is then used to update CS by allocating wavelengths for rt_{min}. If rt_{min} is an indirect route i.e. if the pair (rt_{min}, WL_{min}) $\in S_{RW}$, then procedure *getFirstNode*() is used to obtain the first node v_f in rt_{min}. This is required because in case of multi-hop grooming a lightpath has to be set up between v_f and d of r.

If no route (neither direct nor indirect) is found with the present value of constraint $minCost$ i.e. if both $S_{RW} = \Phi$ and $WL_d = \Phi$, the value of $minCost$ is incremented and the entire process is repeated with the new value of $minCost$. This entire process (lines 6-23) is repeated in every iteration with increased value of $minCost$ until a route is found and till the value of $minCost$ do not exceed the link capacity W (line 5). The procedure *searchRoute*() returns v_f, rt_{min} and WL_{min} and the updated CS and $Cost$. We do not include the details of *getMin*(), *findMin*(), *incrCost*(), *update*() and *getFirstNode*() as they are easy to understand.

E. PROCEDURE getRWL

The details of procedure *getRWL*() is shown in Fig.6. This procedure is used by *searchRoute*() to obtain the route of a lightpath and also the list of wavelengths that can be assigned to the links of the route. The inputs to the procedure are v, G_P, r_i, CS, $Cost$, $minCost$. If v is NULL, BFS [36] is initiated for a route rt_d in G_P that connects s_i and d_i of r_i with each link i of rt_d having $Cost[i] < minCost + 1$. If v is not NULL then BFS is initiated for a route rt_{ind} in G_P that connects v and d_i with each link i of rt_{ind} having $Cost[i] < minCost + 1$. The procedure also uses CS to find the list of wavelengths WL_d (WL_{ind}) that can be assigned following the FF-policy

Procedure *getRWL* (v, G_P, r, CS, $Cost$, $minCost$)
begin
  if (v = NULL) then
    find a route rt_d in G_P using BFS connecting s and d of r such that for every link i in rt_d, $Cost[i] < minCost + 1$;
    Use CS to find wavelength list WL_d consisting of wavelengths that can be assigned to the links of rt_d using First-Fit policy;
    return (rt_d, WL_d);
  else
    find a route rt_{ind} in G_P using BFS connecting v and d of r such that for every link i in rt_{ind}, $Cost[i] < minCost + 1$;
    Use CS to find wavelength list WL_{ind} consisting of wavelengths that be can assigned to the links of rt_{ind} using First-Fit policy;
    return (rt_{ind}, WL_{ind});
  end if
end procedure

FIGURE 6. Procedure ***getRWL***.

to the links of rt_d (rt_{ind}). The matrix CS is updated by setting $CS[i][j]$ to 0 for every link i in rt_d (rt_{ind}) and every wavelength j in WL_d (WL_{ind}) to indicate current wavelength status. The procedure returns rt_d and WL_d if v is NULL or rt_{ind} and WL_{ind} if v is not NULL.

F. PROCEDURE updateTopology

This procedure is used by both *genInitTopology*() and *findSurvPaths*(). The procedure is used to update the logical topology while accommodating the traffic of a new connection. The inputs to the procedure are G_L, R, r, ($rt_{min} \backslash lpath$), ($v_f \backslash$NULL) ($WL_{min} \backslash \Phi$) The notation ($x \backslash y$) is used to denote either x or y. The following two cases may arise depending on the inputs.

Case 1: The inputs are G_L, R, r, rt_{min}, v_f, WL_{min}. It must be noted that v_f denotes the first node of an indirect physical route. The value of v_f will be NULL if rt_{min} is a direct route and not NULL if rt_{min} is an indirect route. The procedure works as follows. If $v_f \neq$ NULL and $WL_{min} \neq \Phi$ then rt_{min} is an indirect route. In this case a lightpath is established between v_f and d using rt_{min}, the wavelengths in WL_{min} are assigned and rt_{min} is added to R. A logical edge $v_f \rightarrow d$ is added to G_L. The traffic t of r is routed along the logical path $s \rightarrow v_f \rightarrow d$ in G_L and the capacities of the logical edges of the path are updated. If $v_f =$ NULL and $WL_{min} \neq \Phi$ then rt_{min} is a direct route. In this case a lightpath is established directly between s and d using rt_{min}, the wavelengths in WL_{min} are assigned and rt_{min} is added to R. A logical edge $s \rightarrow d$ is added to G_L, traffic stream t of r is routed and the capacity of the logical edge is updated. The updated G_L and its routing R are returned.

Case 2: The inputs are G_L, R, r, $lpath$, NULL, Φ. The value $WL_{min} = \Phi$ indicates that the input $lpath$ is a logical path. In this case the traffic t is routed along $lpath$ and the capacities of the logical edges comprising $lpath$ are updated. The updated G_L and its routing R are returned.

VI. TWO FORMS OF SURVIVABILITY

We study two forms of survivability namely survivability at connection and survivability at lightpath in *TATG-SC* and *TATG-SL* respectively.

1) *TATG-SC* (*TATG*-Survivability at Connection) – In *TATG-SC*, we study the problem of providing survivability to each connection request that may get disrupted due to a potential physical link failure. To address this problem, the heuristic seeks a survivable path in the logical topology for each connection that may be disrupted. The *TATG* heuristic as presented in the paper actually addresses survivability at connection.

2) *TATG-SL* (*TATG*-Survivability at Lightpath) – In *TATG-SL*, we study the problem of providing survivability to each lightpath that may get disrupted due to a potential physical link failure. To address this problem, the heuristic seeks a survivable path in the logical topology for each lightpath that may be disrupted.

The traffic of a connection request is accommodated in a lightpath. Thus the links in a lightpath route is also traversed by each connection that the lightpath accommodates. So if a lightpath gets disrupted due to a link failure then all the connections that it is carrying are disrupted as well. A connection request is assumed to survive if it can be successfully restored in the failure scenario of every physical link $e \in E_P$ that it traverses. A connection request that cannot be restored for the failure of a particular link should not be tried for restoration in subsequent links that it traverses. This is ensured by *marking* them as "FAILED". Connection requests are assumed to be global variables and so once *marked* they remain *marked* while considering subsequent link failures.

In *TATG-SL*, we not only *mark* the connections but we also *mark* lightpaths. A lightpath is *marked* "FAILED" if it cannot be restored for a link failure. It must also be noted that for every failure scenario, *findSurvPaths*() creates and returns the modified logical topology G_L (lines 22-24, Fig.4). So the *marked* lightpaths are also available when subsequent failures are considered. This ensures that a lightpath which cannot be restored in the failure scenario of one physical link is not considered in subsequent failure scenarios. In *TATG-SL* the procedure *findSurvPaths*() (Fig.4) is modified as follows. In line 4, the disrupted lightpaths in L_e are sorted in *increasing* order of residual capacity. The loop in line 7, iterates for every *unmarked* lightpath $l_i \in L_e$. If a lightpath l_i cannot be restored i.e. ($lpath =$ NULL) in line 10 and ($WL_{min} = \Phi$) line 12 then all connections accommodated by the lightpath is considered to be blocked i.e. line 13 is replaced by $N_{cf} \leftarrow N_{cf} + |C_e|$. Then all connections in C_e are *marked* FAILED in line 14. Further, l_i is *marked* "FAILED". The input r_j is replaced by l_i in the procedures *findPath*(), *searchRoute*() and *updateTopology*(). These procedures then work for lightpath l_i.

VII. TIME COMPLEXITY ANALYSIS

We analyze the worst-case time complexity of the proposed heuristic algorithm *TATG*.

Lemma: For a given physical topology $G_P(V_P, E_P)$ with link capacity W and m connections to serve, *TATG* can take at most $O(m^2 + mW(|V_P| + |E_P|) + p^2q^2|E_P| + pqW|E_P|(|V_P| + |E_P|))$ time to complete, if the average number of lightpaths that traverse a fiber link is p and the average number of connections accommodated by a lightpath is q.

Proof: The entire heuristic of *TATG* as mentioned earlier can be broken down into two steps i.e. *Step*-1: generate the initial logical topology in fault-free scenario and *Step*-2: perform survivable routing of the generated logical topology.

So we analyze the time taken by each of these two steps separately to determine the overall time complexity. Let $|C| = m$.

Step-1: The procedure *genInitTopology*() is called to create the initial logical topology under fault-free condition. For each $r_i \in C$, procedure *findPath*() is called to find a logical path (*lpath*) in G_L having sufficient capacity to accommodate t_i. Initially there are no lightpaths in G_L but lightpaths are added for connection requests for which *findPath*() fails to find the required *lpath*. Let us assume the worst-case i.e. *findPath*() fails to find the required *lpath* for every request r_i and a new lightpath is always established or in other words a logical edge is always added to G_L. With this assumption, there can be at most m logical edges in G_L and these are added sequentially and not at the same time. Thus the number of logical edge traversals in the worst-case by *findPath*() for the first request is 0, for the second request is 1, for the third request is 2 and so on. Thus the total number of logical edge traversals is $(1 + 2 + 3 + \ldots + m - 1) = m(m-1)/2 \approx m^2$.

For every failure of *findPath*(), procedure *searchRoute*() is used to find the route of the new lightpath to be established. We first analyze the time required by *searchRoute*(). Let us assume that on an average the number of vertices in V_a turns out to be x. Finding *minCost* from *Cost* causes $|E_P|$ operations to be performed. For each $v \in V_a$, *searchRoute*() uses *getRWL*() to find the route and wavelength list for RWA of the lightpath connecting v and d_i in G_P constrained by *minCost*. We employ BFS in *getRWL*() and so the time required is order of $(|V_P| + |E_P|)$. So the total time for all x nodes is $x(|V_P| + |E_P|)$. The procedure *getRWL*() is also used to find a direct route between s_i and d_i constrained by *minCost* which also takes order of $(|V_P| + |E_P|)$ time. So the total time spent by *getRWL*() for every request is order of $x(|V_P| + |E_P|) + (|V_P| + |E_P|) = (x + 1)(|V_P| + |E_P|)$. If *searchRoute*() fails for the current value of *minCost* then the value of *minCost* is incremented and the entire process is repeated. This goes on till $minCost < W$. Thus the worst-case value of *minCost* will be W. So the total time spent by *getRWL*() for every request in the worst case is order of $W(x+1)(|V_P| + |E_P|)$. So total time spent in *searchRoute*() for m connection requests in worst case is order of $m\{|E_P| + W(x + 1)(|V_P| + |E_P|)\}$. Thus the time required by *genInitTopology*() is order of $m^2 + m\{|E_P| + W(x + 1)(|V_P| + |E_P|)\}$.

Step-2: To consider the potential failure of each physical edge, procedure *findSurvPaths*() is used. Let the average number of lightpaths that traverse an edge $e \in E_P$ be p. Since the p lightpaths are removed from G_L, the new topology G_L' can contain $m - p$ logical edges (the upper bound on the number of logical edges in G_L is m). Let q be the average number of connection requests accommodated by a lightpath. The set C_e for every $e \in E_P$ will then contain pq number of connection requests on an average. Sorting the set C_e in *descending* order of traffic demand using heap sort [36] takes order of $pq \log pq$ operations. For each $r_j \in C_e$, *findPath*() is used to find a logical path in G_L' that will be able to accommodate the traffic t_j of r_j. Let us assume the worst case for every r_j i.e. the procedure *findPath*( ) fails to find a logical path and a new lightpath have to be established. If this happens then pq additional lightpaths will be added to G_L'. But all these pq additional lightpaths are added sequentially and not at the same time. Thus the total number of logical edge traversals by *findPath*() for the pq requests $= (m - p + 0) + (m-p + 1) + (m-p + 2) + \ldots + (m - p + pq - 1) = pq(m-p) + (1+2+3+\ldots+pq-1) = pq(m-p) + pq(pq-1)/2 \approx p^2q^2$ (neglecting smaller terms).

For every failure of *findPath*(), procedure *searchRoute*() is used to find the route of the new lightpath to be established. Letting $|V_a| = x$ as in *Step*-1, the total time spent for every request by *getRWL*() in the worst case is order of $W(x+)(|V_P| + |E_P|)$. So total time spent in *searchRoute*() for pq connection requests in worst case is order of $pq\{|E_P| + W(x + 1)(|V_P| + |E_P|)\}$. Since the heuristic considers potential failure of every $e \in E_P$, the total time spent by *findSurvPaths*() is order of $|E_P|[p^2q^2 + pq\{|E_P| + W(x + 1)(|V_P| + |E_P|)\}]$.

The total time spent by *TATG* is thus the sum of the times taken by *genInitTopology*() in *Step*-1 and *findSurvPaths*() in *Step*-2. Thus the total time spent by *TATG* is $m^2 + m\{|E_P| + W(x + 1)(|V_P| + |E_P|)\} + |E_P|[p^2q^2 + pq\{|E_P| + W(x + 1)(|V_P| + |E_P|)\}] = m^2 + m|E_P| + mW(x+1)(|V_P| + |E_P|)\} + p^2q^2|E_P| + pq|E_P|^2 + W(x + 1)pq|E_P|(|V_P| + |E_P|)\} = O(m^2 + mW(|V_P| + |E_P|) + p^2q^2|E_P| + pqW|E_P|(|V_P| + |E_P|))$ neglecting all other smaller terms.

VIII. PERFORMANCE COMPARISONS

Extensive simulation experiments were performed in an environment that used 64-bit operating system Windows 10 and Intel(R) Core(TM) i5-6200U CPU @ 2.30 GHz with 4.00 GB RAM. The heuristics were implemented using programs written in Java using JDK 1.7. We compare the performance of the two heuristics *TATG-SC* and *TATG-SL* with the heuristic in [24] that uses STGLTD. We shall refer to the heuristic in [24] as *HTG*. For comparison of the heuristics a 36-node (6×6) Manhattan Street Network [37] was used. We performed experiments with randomly generated sets of connection requests of sizes 50, 100, 150, 200, 300 and 400 and with the following three categories of traffic demand.

1. Low traffic: The traffic t_i of every traffic request r_i is randomly distributed between units of OC-3 to OC-24 in a

way that t_i varies from 10% to 30% of the capacity of a lightpath.

2. Medium traffic: The traffic t_i of every traffic request r_i is randomly distributed between units of OC-3 to OC-48 in a way that t_i varies from 10% to 50% of the capacity of a lightpath.

3. High traffic: The traffic t_i of every traffic request r_i is randomly distributed between units of OC-3 to OC-96 in a way that t_i varies from 10% to 75% of the capacity of a lightpath.

We assume the capacity of a single lightpath to be OC-192. For a particular set size and for each traffic demand category, 10 sets of connection requests were generated for testing the three heuristics (*TATG-SC*, *TATG-SL* and *HTG*) and we have reported the average. A connection request r_i in a set of requests is randomly generated as a 4-tuple (s_i, d_i, g_i, k_i), where s_i represents the source, d_i the destination, g_i the granularity (for instance OC-48), k_i the amount of traffic in units of g_i. The request r_i is converted to a 3-tuple (s_i, d_i, t_i) where t_i is calculated $g_i \times k_i$. For example if in high traffic category an OC-96 request is generated as the 4-tuple (1, 7, 96, 1.5) then it is converted to a 3-tuple (1, 7, 144).

We first compare the performance of proposed heuristics with the optimal results obtained from the ILP formulation presented in [24]. The ILP is solved using ILOG CPLEX 12.8. The 6-node network shown in Fig.7 is used for the comparison. We assume medium traffic conditions for the comparisons. Each node is assumed to be equipped with 3 transmitter-receiver pairs and each link is assumed to have capacity 10 (i.e. 10 wavelengths per link). We compare the heuristics with the ILP for connection sets of sizes 15, 20, 30 and 40. For sets with more than 40 connections the ILP failed to run. Fig.8 shows the percentage of total input traffic groomed for the ILP and the proposed heuristics.

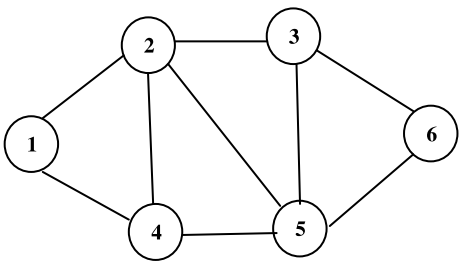

FIGURE 7. 6-Node network.

The three heuristics are compared in non-blocking as well as blocking conditions. W_{min} denotes the required link capacity or the minimum number of wavelengths required in a link to satisfy a given set of connection requests. In non-blocking condition the heuristics are compared in terms of the values of W_{min} required for generating the survivable logical topology. In blocking conditions i.e. with link capacity restrictions they are compared in terms of *NOS* or throughput (measured as the number of satisfied connection requests). Fig.9, Fig.10 and Fig.11 show the comparisons of W_{min} for different sizes of

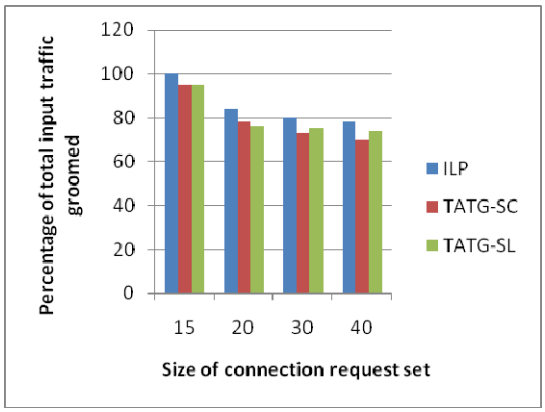

FIGURE 8. Comparison of ILP and proposed heuristics in the 6-node network.

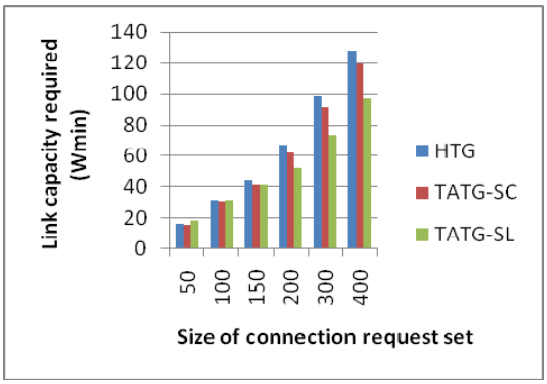

FIGURE 9. Comparison of W_{min} for the three heuristics in low traffic for 36-node manhattan street network.

request sets with low, medium and high traffic categories respectively. Fig.12, Fig.13 and Fig.14 show the percentage reduction in W_{min} obtained by using the proposed heuristics over *HTG* for different sizes of request sets with low, medium and high traffic categories respectively. Table 3 shows the throughput for 18 different experiments (E1-E18) where each experiment differs in terms of either a) request set size or b) traffic category or c) the link capacity restriction. Fig.15, Fig.16 and Fig.17 show the percentage reduction in request blocking obtained by using the proposed heuristics over *HTG* with low traffic, medium traffic and high traffic categories respectively. Results obtained from the experiments leads to the following six important observations.

Observation 1: In Fig.9, Fig.10 and Fig.11, we observe that W_{min} is lesser for the proposed heuristics for almost all experiments, suggesting that the proposed heuristics are more resource efficient. Both *TATG-SC* and *TATG-SL* outperforms *HTG* but the performance of *TATG-SL* is the best. Even with a set of 400 connections in high traffic *TATG-SL* and *TATG-SC* require a link capacity of 107 and 133 wavelengths respectively when the same required by *HTG* is 142.

Observation 2: In Fig.9, Fig.10 and Fig.11, we also observe that *TATG-SC* outperforms *TATG-SL* in terms of W_{min} for request sets of smaller size, for example with $|C| = 50$

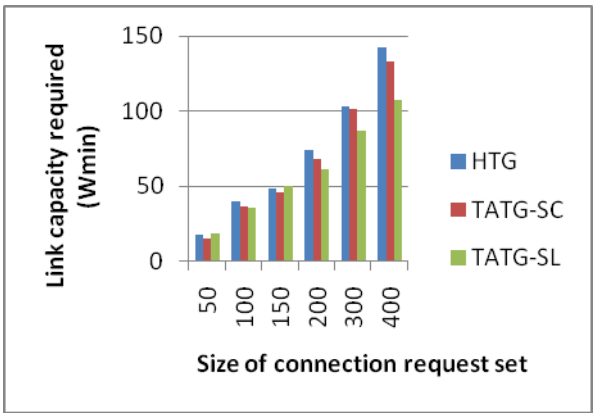

FIGURE 10. Comparison of W_{min} for the three heuristics in medium traffic for 36-node manhattan street network.

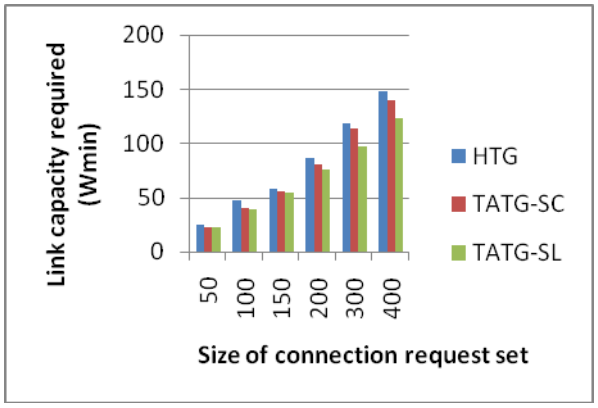

FIGURE 11. Comparison of W_{min} for the three heuristics in high traffic for 36-node manhattan street network.

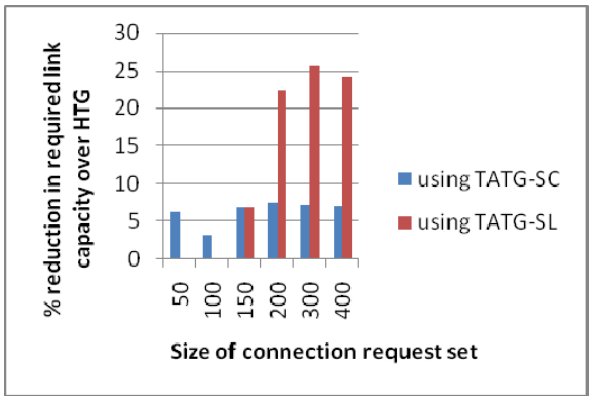

FIGURE 12. Percentage reduction in W_{min} using the proposed heuristics over HTG in low traffic.

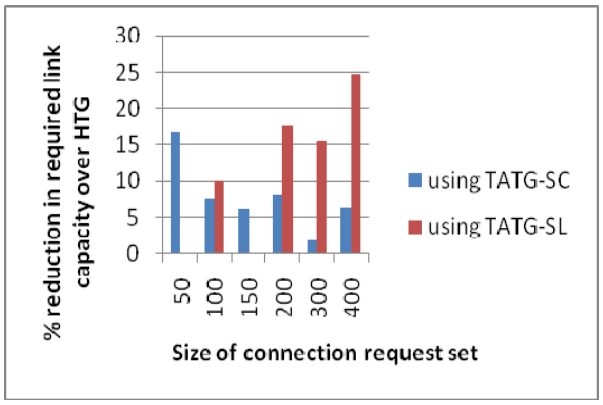

FIGURE 13. Percentage reduction in W_{min} using the proposed heuristics over HTG in medium traffic.

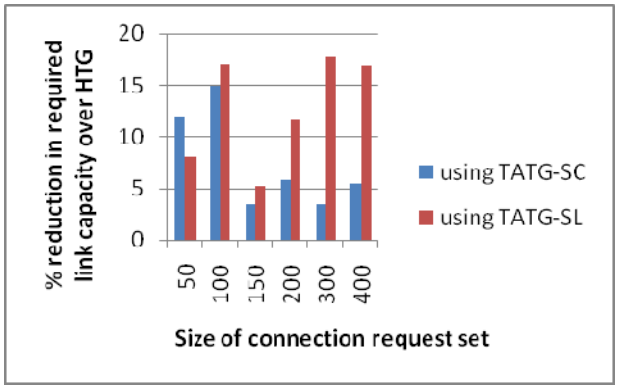

FIGURE 14. Percentage reduction in W_{min} using the proposed heuristics over *HTG* in high traffic.

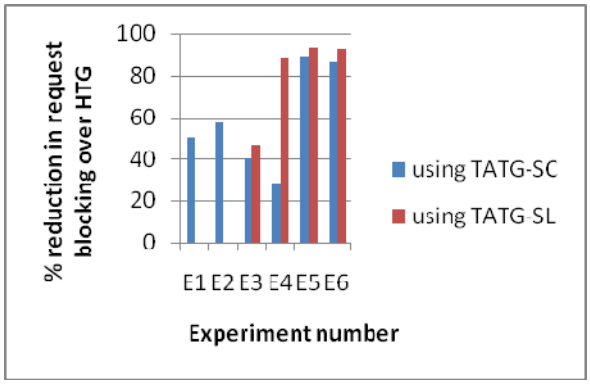

FIGURE 15. Percentage reuction in request blocking using the proposed heuristics over *HTG* in low traffic.

irrespective of the category of traffic demand whereas for request sets of larger size ($|C| \geq 200$), *TATG-SL* outperforms *TATG-SC* for all categories of traffic demand. For analysis, let us assume that the average number of connection requests accommodated by a lightpath is q and the average number of lightpaths traversing an edge e is p. So the average number of requests traversing the edge e is pq. When we consider the failure of an edge, the p lightpaths and the pq requests are assumed to be disrupted.

With request sets of smaller size, the value of p is small and consequently that of pq is also small. Also there is lesser utilization of the capacity of a lightpath and consequently the lightpaths in the logical topology have more spare capacity. Thus there is a greater chance of finding a survivable logical path with adequate spare capacity for each of the

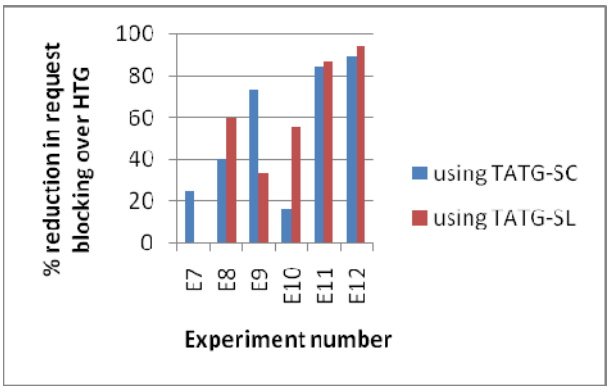

FIGURE 16. Percentage reuction in request blocking using the proposed heuristics over *HTG* in medium traffic.

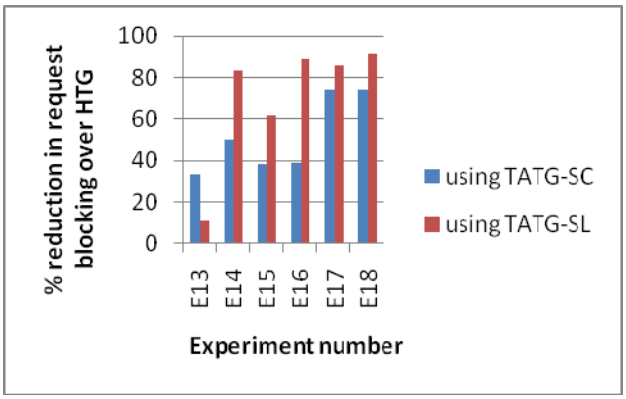

FIGURE 17. Percentage reuction in request blocking using the proposed heuristics over *HTG* in high traffic.

pq disrupted connections using *TATG-SC* resulting in lesser number of additional lightpaths being established. However, as the traffic carried by a lightpath is much greater compared to that of an individual connection request, chances of finding survivable paths with enough capacity for each of the p disrupted lightpaths using *TATG-SL* is much lesser and so more additional lightpaths are required. Since this is true for every $e \in E_P$, W_{min} of *TATG-SC* is less than that of *TATG-SL* for request sets of smaller size.

On the contrary with traffic requests of larger size, the value of p is large and so is the value of pq. Also with larger request set size there is more utilization of the capacity of a lightpath and consequently the spare capacity is lesser. Thus the chance of finding survivable paths for a disrupted request using *TATG-SC* and for a disrupted lightpath using *TATG-SL* is lesser. In the worst case it may so happen that neither *TATG-SC* nor *TATG-SL* is able to find the required survivable logical paths for the pq disrupted connections and the p disrupted lightpaths respectively. In such a case additional lightpaths have to be established by both the heuristics. But since p is much smaller than pq, lesser number of additional lightpaths is required by *TATG-SL* than by *TATG-SC*. Since this is true for every $e \in E_P$, W_{min} of *TATG-SL* is less than that of *TATG-SC*.

TABLE 3. Throughput obtained using the heuristics for the eighteen experiments conducted on manhattan street network.

Exp. No.	Request set size, traffic category and link capacity	*HTG*	*TATG-SC*	*TATG-SL*
		Throughput		
E1	50, Low, 12	46	48	45
E2	100, Low, 28	93	97	91
E3	150, Low, 40	135	141	142
E4	200, Low, 50	175	182	197
E5	300, Low, 71	238	293	296
E6	400, Low, 95	304	387	393
E7	50, Med, 13	42	44	40
E8	100, Med, 34	95	97	98
E9	150, Med, 45	135	146	130
E10	200, Med, 59	182	185	192
E11	300, Med, 85	255	293	294
E12	400, Med, 105	317	391	395
E13	50, High, 20	41	44	42
E14	100, High, 38	94	97	99
E15	150, High, 53	137	142	145
E16	200, High, 75	182	189	198
E17	300, High, 95	265	291	295
E18	400, High, 121	377	394	398

Observation 3: From Fig. 12, Fig.13 and Fig.14, we observe that a significant percentage reduction in required link capacity (W_{min}) is obtained in almost all experiments using the proposed heuristics over *HTG* for all categories of traffic demand. The maximum(average) percentage reduction using *TATG-SC* over *HTG* and *TATG-SL* over *HTG* is 7.46(6.32) and 25.51(13.15) respectively in low traffic, 16.66(7.77) and 24.64(11.29) respectively in medium traffic and 14.89(7.49) and 17.79(12.75) respectively in high traffic.

Observation 4: From the entries in Table 3, it is observed that throughput of the proposed heuristics is more compared to *HTG* with the same link capacity in almost all experiments. Both the heuristics outperforms *HTG* but the throughput of *TATG-SL* is more than that of *TATG-SC*. The maximum(average) increase in throughput using *TATG-SC* over *HTG* and *TATG-SL* over *HTG* is 83(26.16) and 89(29.33) respectively in low traffic, 74(21.66) and 78(21.66) respectively in medium traffic and 26(10.16) and 30(13.5) respectively in high traffic.

Observation 5: We observe that the throughput of *TATG-SC* is more than that of *TATG-SL* in all experiments with request sets of smaller size for example with $|C| = 50$, whereas in experiments with $|C| \geq 200$, the performance reverses. This is because for request sets of smaller size, W_{min} of *TATG-SC* is less than that of *TATG-SL* and consequently greater number of spare wavelengths becomes available. But for request sets with $|C| \geq 200$, W_{min} for *TATG-SL* is less than that of *TATG-SC* and as a result the performance reverses.

Observation 6: From Fig.15, Fig.16 and Fig.17, we observe that the percentage reduction in request blocking

obtained by using the proposed heuristics over *HTG* is significant for any category of traffic demand. Considering all the 18 experiments that were performed, the maximum(average) percentage reduction in request blocking obtained using *TATG-SC* and *TATG-SL* over *HTG* is 58.38(54.87) and 93.97(59.57) respectively.

IX. CONCLUSION

This work addresses the problem of throughput enhancement during survivable traffic grooming (STG) in a static scenario by dealing with the joint problem of STG and logical topology design (LTD) which has been referred to as STGLTD. This work shows that if efficient heuristics at the sub-problem level are employed during STGLTD then significant resource saving can be obtained. This in turn can lead to much better throughput performance when resources are scarce. Heuristics are proposed that can solve the problem in polynomial time. Performance comparison with a well-known heuristic shows that our heuristics are definitely better options to address the problem.

REFERENCES

[1] R. Ramaswami and K. N. Sivarajan, *Optical Networks: A Practical Perspective*. San Mateo, CA, USA: Morgan Kaufmann, San Francisco, CA, United States, 2002.

[2] S. Bandyopadhyay, *Dissemination of Information in Optical Networks*. New York, NY, USA: Springer-Verlag, 2008.

[3] K. Zang and B. Mukherjee, "A review of traffic grooming in WDM optical networks: Architectures and challenges," *Opt. Netw. Mag.*, vol. 4, no. 2, pp. 55–64, Mar. 2003.

[4] H. Zhu, H. Zang, K. Zhu, and B. Mukherjee, "A novel generic graph model for traffic grooming in heterogeneous WDM mesh networks," *IEEE/ACM Trans. Netw.*, vol. 11, no. 2, pp. 285–299, Apr. 2003.

[5] M. Shalom, W. Unger, and S. Zaks, "On the complexity of the traffic grooming problem in optical networks," in *Proc. Int. Conf. Fun Algorithms (FUN)* (Lecture Notes in Computer Science), vol. 4475. 2007, pp. 262–271.

[6] W. Yao and B. Ramamurthy, "Survivable traffic grooming with path protection at the connection level in WDM mesh networks," *J. Lightw. Technol.*, vol. 23, no. 10, pp. 2846–2853, Oct. 29, 2005.

[7] E. Modiano and A. Narula-Tam, "Survivable routing of logical topologies in WDM networks," in *Proc. IEEE INFOCOM*, Apr. 2001, pp. 348–357.

[8] A. Bari, Q. Rahman, A. Jaekel, and S. Bandyopadhyay, "Traffic grooming in WDM mesh networks with guaranteed survivability," in *Proc. IEEE Int. Conf. Dependable Syst. Netw.*, Jun. 2008, pp. 307–315.

[9] K. Zhu, H. Zhu, and B. Mukherjee, *Traffic Grooming in Optical WDM Mesh Networks*. Springer, 2005.

[10] H. Yao, Z. Yang, L. Ou, and X. Tan, "A transceiver saving auxiliary graph model for dynamic traffic grooming in WDM mesh networks," in *Proc. 31st IEEE Conf. Local Comput. Netw. (LCN)*, Nov. 2006, pp. 319–326.

[11] S. Huang, R. Dutta, and G. N. Rouskas, "Traffic grooming in path, star, and tree networks: Complexity, bounds, and algorithms," *IEEE J. Sel. Areas Commun.*, vol. 24, no. 4, pp. 66–82, Apr. 2006.

[12] C. Xin, B. Wang, X. Cao, and J. Li, "Logical topology design for dynamic traffic grooming in WDM optical networks," *J. Lightw. Technol.*, vol. 24, no. 6, pp. 2267–2275, Jun. 2006.

[13] Y. Wang and Q. Gu, "Maximizing throughput for traffic grooming with limited grooming resources," in *Proc. IEEE Global Telecommun. Conf. (GLOBECOM)*, Nov. 2007, pp. 2337–2341.

[14] F. Solano, L. Caro, J. De Oliveira, R. Fabregat, and J. Marzo, "G+: Enhanced traffic grooming in WDM mesh networks using lighttours," *IEEE J. Sel. Areas Commun.*, vol. 25, no. 5, pp. 1034–1047, Jun. 2007.

[15] Y. Wang and Q.-P. Gu, "A min-max optimization problem on traffic grooming in WDM optical networks," in *Proc. 16th Int. Conf. Comput. Commun. Netw. (ICCCN)*, Aug. 2007, pp. 228–233.

[16] H. Yao, Y. Yang, and Z. Yang, "Two dynamic restoration schemes for survivable traffic grooming in WDM networks," in *Proc. 33rd IEEE Conf. Local Comput. Netw. (LCN)*, Oct. 2008, pp. 560–561.

[17] M. A. Saleh and A. E. Kamal, "Many-to-many traffic grooming in WDM networks," *IEEE/OSA J. Opt. Commun. Netw.*, vol. 1, no. 5, pp. 376–391, Oct. 2009.

[18] C. Xin, "Resource planning for dynamic traffic grooming in WDM optical networks," *J. Lightw. Technol.*, vol. 27, no. 7, pp. 817–824, Apr. 1, 2009.

[19] M. A. Saleh and A. E. Kamal, "Design and provisioning of WDM networks with many-to-many traffic grooming," *IEEE/ACM Trans. Netw.*, vol. 18, no. 6, pp. 1869–1882, Dec. 2010.

[20] A. Jaekel, Y. Chen, and A. Bari, "Stable logical topologies for survivable traffic grooming of scheduled demands," *IEEE/OSA J. Opt. Commun. Netw.*, vol. 2, no. 10, pp. 793–802, Oct. 2010.

[21] A. Balma, N. B. Hadj-Alouane, and A. B. Hadj-Alouane, "A near-optimal solution approach for the multi-hop traffic grooming problem," *IEEE/OSA J. Opt. Commun. Netw.*, vol. 3, no. 11, pp. 891–901, Nov. 2011.

[22] M. A. Saleh and A. E. Kamal, "Approximation algorithms for many-to-many traffic grooming in optical WDM networks," *IEEE/ACM Trans. Netw.*, vol. 20, no. 5, pp. 1527–1540, Oct. 2012.

[23] S.-W. Wang and C.-Y. Wen, "Lightpath-level active rerouting algorithms in all-optical WDM networks with alternate routing and traffic grooming," in *Proc. Int. Conf. Inf. Netw.*, Feb. 2012, pp. 42–46.

[24] A. Jaekel, A. Bari, Q. Rahman, Y. Chen, S. Bandyopadhyay, and Y. Aneja, "Resource efficient network design and traffic grooming strategy with guaranteed survivability," *Opt. Switching Netw.*, vol. 9, no. 4, pp. 271–285, Nov. 2012.

[25] M. I. Anis, N. Amaya, G. Zervas, S. Pinna, M. Scaffardi, F. Fresi, A. Bogoni, R. Nejabati, and D. Simeonidou, "All-optical traffic grooming in elastic optical network," in *Proc. Opt. Fiber Commun. Conf./Nat. Fiber Optic Eng. Conf. (OFC/NFOEC)*, 2013, pp. 1–3.

[26] A. Bhattacharya, A. K. Saha, and M. Chatterjee, "An efficient traffic grooming policy for heterogeneous WDM mesh networks," in *Proc. IEEE Int. Conf. Adv. Netw. Telecommun. Syst. (ANTS)*, Dec. 2014, pp. 1–6.

[27] J. de Santi, A. C. Drummond, N. L. S. da Fonseca, and X. Chen, "Holding-time-aware dynamic traffic grooming algorithms based on multipath routing for WDM optical networks," *Opt. Switching Netw.*, vol. 16, pp. 21–35, Apr. 2015.

[28] W. Hou, L. Guo, and J. Lu, "Multi-domain traffic partition grooming in mixed granularity optical networks," *Opt. Switching Netw.*, vol. 17, pp. 25–37, Jul. 2015.

[29] S. R. Shinde and S. H. Patil, "Heuristics for sparse traffic grooming in dynamic WDM optical mesh networks," in *Proc. Int. Conf. Comput. Commun. Control Autom.*, Feb. 2015, pp. 159–163.

[30] M. Gagnaire and E. Doumith, "An iterative greedy algorithm for scheduled traffic grooming in WDM optical networks," in *Proc. 1st Int. Symp. Adv. Netw. Telecommun. Syst.*, 2007, pp. 1–2.

[31] H. Wang and G. N. Rouskas, "Hierarchical traffic grooming: A tutorial," *Comput. Netw.*, vol. 69, pp. 147–156, Aug. 2014.

[32] F. Dikbiyik, M. Tornatore, and B. Mukherjee, "Exploiting excess capacity for survivable traffic grooming in optical backbone networks," *IEEE/OSA J. Opt. Commun. Netw.*, vol. 6, no. 2, pp. 127–137, Feb. 2014.

[33] H.-C. Lin and Y.-X. Zhuang, "An effective algorithm for dynamic traffic grooming in light-trail WDM mesh networks," in *Proc. Opto-Electron. Commun. Conf. (OECC) Photon. Global Conf. (PGC)*, Jul. 2017, pp. 1–5.

[34] A. K. Pradhan, S. Singhi, and T. De, "Multicast dynamic traffic grooming using bin packing method in WDM mesh networks," *Opt. Switching Netw.*, vol. 23, pp. 40–51, Jan. 2017.

[35] D. Naik, Nikita, and T. De, "Congestion aware traffic grooming in elastic optical and WiMAX network," in *Proc. Technol. Smart-City Energy Secur. Power (ICSESP)*, Mar. 2018, pp. 1–9.

[36] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*. Cambridge, MA, USA: MIT Press, 2001.

[37] H. Hwang, H. Kim, Y. Choi, and C. Kim, "Multicast routing algorithms for Manhattan street network," in *Proc. 22nd EUROMICRO Conf. (EUROMICRO)*, 1996, pp. 397–404.

ASIMA BHATTACHARYA received the M.Sc. degree in physics from the University of Science and Technology, Calcutta, India, in 1995, and the Post Graduate Diploma degree in information management (PGDIM) from the Technical Teachers' Training Institute, Calcutta, in 1996. She started her carrier at Software Industry, in 1996. In her 22 years of tenure in this industry, she worked in organizations like CMC LTD, UshaComm, Reliance Infocomm, Amdocs, Accenture, iConnectiva, TEOCO, and so on., in the capacity of a Project Lead, a Project Manager, the Vice President, and the Director mainly in telecommunications domain. She is currently working as the Director of TEOCO, responsible for global product delivery and support in radio access network domain. Her research interests include solving existing problems for telecommunication mesh networks, survivability, and traffic grooming in optical networks.

MALABIKA SARDER received the B.Tech. degree in computer science and engineering from the West Bengal University of Technology, India, in 2009, and the M.E. degree in computer science and technology from Bengal Engineering and Science University, Shibpur, India, in 2012. She is currently working as a Software Engineer at Samsung India Electronics Pvt., Ltd. Her research interests include mobile ad hoc networks and WDM optical networks.

MONISH CHATTERJEE (Member, IEEE) received the Ph.D. (engineering) degree in the branch of computer science and engineering from Jadavpur University, India, in 2012, and the M.E. degree in computer science and technology from the Indian Institute of Engineering Science and Technology (IIEST), Shibpur, India, in 2000, formerly Bengal Engineering and Science University (BESU), Shibpur, one of the oldest pioneering engineering institutes in India. He started his career as a Software Engineer at IT industry, but after a short period he subsequently switched over to academia. He is currently working as an Associate Professor and the Head of the Department of Computer Science and Engineering, Asansol Engineering College, India. He has 17 years of teaching experience and two years of IT industry experience. He has publications in refereed International Journals and the IEEE International Conference Proceedings. His research interests include existing problems in WDM optical networks and elastic optical networks. He has been a member of Technical Program Committee of the many IEEE International Conferences and Workshops. He was a Sessions Chair.

DIGANTA SAHA received the B.E., M.E., and the Ph.D. (Engineering) degrees in the branch of Computer Science and Engineering from Jadavpur University, India, in 1995, 1998, and 2009, respectively. He is currently working as a Professor with the Department of Computer Science and Engineering, Jadavpur University, India. His research interests include communication networks, natural language processing, and text data mining. He is a member of IET.