Optimizing Data Plane Resources for Multi-Path Flows

Gabi Nakibly   Reuven Cohen   Liran Katzir

Abstract—**In many modern networks, such as datacenters, optical networks, and MPLS, the delivery of a traffic flow with a certain bandwidth demand over a single network path is either not possible or not cost effective. In these cases, it is very often possible to improve the network's bandwidth utilization by splitting the traffic flow over multiple efficient paths. While using multiple paths for the same traffic flow increases the efficiency of the network, it consumes expensive *forwarding resources* from the network nodes, such as TCAM entries of Ethernet/MPLS switches and wavelengths/lightpaths of optical switches. In this paper we define several problems related to splitting a traffic flow over multiple paths while minimizing the consumption of forwarding resources, and present efficient algorithms for solving these problems.**

I. Introduction

In computer networks, a traffic flow is a flow of data packets sharing the same source and destination network nodes (switches or routers). A traffic flow can often be split into multiple traffic subflows, usually using information in the packet header, such as the IP/MAC addresses, the Port fields in the UDP/TCP header, or the VLAN number. Because these traffic subflows are generated by different applications, or even by different hosts, it is possible to route each of them over a different network path. Using multiple paths for a traffic flow is useful when routing over a single path is impossible or too expensive.

A simple example of the advantages of multipath routing is illustrated in Figure 1. Suppose that we would like to route a 2Gb/s traffic flow from a to f. Suppose that the default (shortest) path, $a \rightarrow b \rightarrow f$ has only 1 Gb/s available bandwidth, and the other (longer, and therefore less cost effective) path has only 1.5 Gb/s available bandwidth. In this case, the traffic flow can be split such that 1 Gb/s will be routed over the upper path and 1 Gb/s over the lower path. But splitting a traffic flow over multiple paths consumes extra "forwarding resources" from the network nodes. These resources are proportional to the number of paths (2 in Figure 1), and the number of nodes/links traversed by these paths (6 links in Figure 1), as we now describe for several network technologies:

1. Optical Networks: In optical networks, each path is an optical, λ-switched, lightpath. Such lightpaths can be set up and taken down in real time. The dominating cost in the setup of a lightpath is of the transponders at the two ends of it, which convert optical to electronic signals and vice versa (see [5], [12], [29] and references therein). Therefore, network operators either impose a strict upper bound on the number of lightpaths that can be established, or seek to minimize this number subject to other constraints. In addition, every lightpath requires a wavelength on each optical link it traverses. Since wavelengths are also a scarce resource, it is often desirable to minimize not only the number of lightpaths, but also the number of nodes traversed by each one.

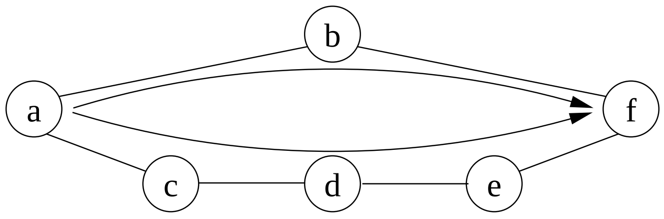


Fig. 1. A simple example of a multi-path flow

2. Ethernet switches in datacenters: Ethernet is the default technology for connecting hardware in datacenters. Using SDN (Software Defined network), network operators can establish multiple paths between source-destination pairs in their datacenters. However, each path requires an entry in the expensive (TCAM) forwarding table of each switch it traverses. According to [9], a large network may require hundreds of thousands of path flow table entries at each switch, while commodity switches have much smaller flow tables. In [25] it is also indicated that datacenter scaling is made difficult by the forwarding table size, which increases linearly with the size of the system. In addition to the forwarding cost associated with every traversed node, there is an extra forwarding cost associated with every path The source of this extra cost can be found in the high speed NIC (Network Interface Card) used to connect the servers between which most datacenter traffic is transmitted. The NICs have a very limited forwarding table, much smaller than a typical switch. The division of a traffic flow into multiple paths is performed by the NIC using a classification logic, which consumes one entry in the forwarding table for every path over which packets of this traffic flow are forwarded.

3. MPLS networks: MPLS Traffic Engineering (MPLS-TE) is used today by most network operators for building an IP infrastructure based on traffic engineering and QoS (Quality of Service) considerations [3]. Using routing protocols such as OSPF-TE, an MPLS-TE router is provided with a map of the network topology and with information about the bandwidth available on each link. Each ingress router uses this map to find a route with sufficient bandwidth for a given traffic flow

G. Nakibly and R. Cohen are with the Department of Computer Science, Technion, Haifa, Israel.
L. Katzir is with Yahoo! Israel Labs, Haifa, Israel.

to an egress router. An ingress MPLS router establishes an MPLS LSP (Label Switched Path) over a selected route and uses it to deliver the traffic flow through the network. The challenge of minimizing the number of LSPs and the number of nodes they traverse is similar to that described above for Ethernet. Each LSP requires one entry in the costly forwarding table of each node it traverses In addition, the division of the traffic flow's packets into multiple LSPs is performed by end "pseudowire" switches, each of which needs to allocate an end-to-end forwarding entry in its expensive forwarding table for every used path [8].

We study the problem of reducing the forwarding cost in two different cases. In the first case, called **DMO** (Decomposition with Minimum Overhead), we are given a traffic demand (source, destination and bandwidth demand) and a network flow[1] that satisfies the bandwidth demand between the source and destination nodes. This network flow is pre-determined according to some bandwidth efficiency criterion, such that bandwidth cost, and the problem is to break it into a set of simple paths between the source and destination nodes, while minimizing the number of paths or the number of nodes they traverse. In the second case, called **RMO** (Routing with Minimum Overhead), only a traffic demand (source, destination and bandwidth demand) is given, and the problem is to find a set of simple paths between the source and destination nodes over which the bandwidth demand can be delivered, while minimizing the number of paths or the number of nodes they traverse. At first glance it seems that RMO should be solved using a solution for DMO as a sub-routine. We indeed find this approach to perform very well, but in the general case it may be better to build a solution for RMO as a collection of paths, rather than starting with an initial network flow.

For both problems we aim at *minimizing the forwarding cost*, measured as the number of paths or the number of nodes traversed by the paths. Thus, we actually solve two pairs of problems: (a) DMO(p) and RMO(p) for minimizing the number of paths; (b) DMO(n) and RMO(n) for minimizing the number of nodes.

Throughout the paper we focus on the case where traffic flows are admitted one by one. While there are scenarios where the operator can admit many traffic flows at the same time, we believe that the "one traffic flow at a time" scenario is very important for the following three reasons. First, in many relevant applications, traffic flows are admitted for a pre-specified duration. The starting times and due dates of the flows are usually independent. Thus, when a new traffic flow has to be admitted, previous traffic flows already use their own paths. Second, an operator may decide to set up a new set of paths between two nodes in order to respond better to periodic congestion. This is an on-line decision, which is captured by the "one traffic flow at a time" approach. Third, when a link or a node fails, each path that crosses this link or node has to be re-routed, which is again an on-line problem. Our goal in this work is thus to minimize the forwarding cost associated with the delivery of each traffic flow, and not to set an upper bound (in each switch) on the total forwarding cost associated with all traffic flows.

[1]Given a network graph $G(V, E)$, a network flow is a real valued function $f : V \text{x} V \rightarrow R$ that satisfies the capacity constraint, the skew symmetry, and the flow conversation [1]. Figure 2(c) shows an example of a network flow for the graph in Figure 2(a).

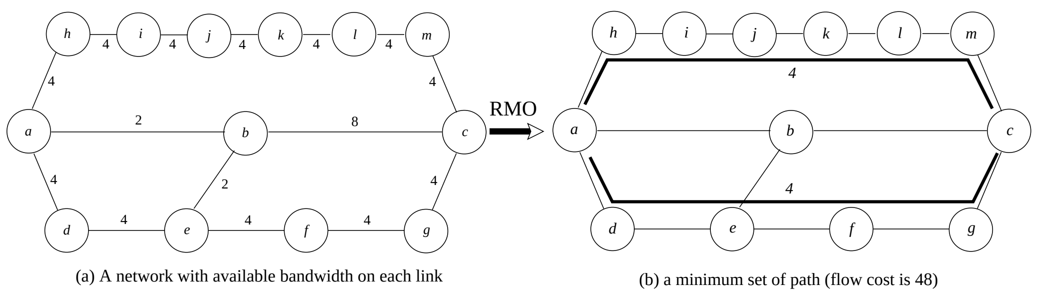
Some operational issues have to be addressed before the algorithms proposed in this paper can be applied: how the network knows the volume of traffic for each flow and how sophisticated routing decisions can be made in a distributed environment. These issues are relatively easy when a centralized controller is employed, as is usually the case for all the application scenarios considered above. A centralized controller can make centralized routing decisions and can inform each switch how to forward each flow. Other issues, such as how to divide a traffic flow into several paths and how to avoid congestion – e.g., by limiting the volume of traffic forwarded on each path – are orthogonal to the algorithms we present, and are therefore beyond the scope of this paper.

The rest of this paper is organized as follows. In Section II we illustrate in greater detail the DMO and RMO problems. In Section III we discuss related work. In Section IV we formally define the DMO(p) problem, discuss its computational complexity, and present approximation algorithms. In Section V we do the same for the RMO(p) problem. In Section VI we address DMO and RMO while minimizing the number of nodes rather than the number of paths. The actual performance of the proposed algorithms is evaluated through simulations in Section VII. Finally, Section VIII concludes the paper.

II. DMO VS. RMO

DMO and RMO are illustrated in Figure 2. The bandwidth cost of a flow on a link is the link cost times the volume of flow it carries. For the sake of simplicity, let the cost of each link in this example be 1. Figure 2(a) shows a network with the capacity (available bandwidth) of each link. First, suppose that the operator needs to accommodate a 1Gb/s traffic demand from node a to c. The most efficient routing solution is to use the shortest path $a - b - c$. The bandwidth cost of this solution is 2. It is cheap and can be delivered using a single path. However, if the operator needs to accommodate an 8Gb/s traffic demand between the same nodes, the shortest path cannot carry it. When the main optimization criterion is to minimize the forwarding cost, the operator can use this traffic demand as an input to RMO. Figure 2(b) shows a routing of the traffic demand over two paths of 4Gb/s: $a-d-e-f-g-c$ and $a-h-i-j-k-l-m-c$. In this example, this solution minimizes both the number of paths (2) and the number of nodes that carry them: $6+8=14$ (we count nodes a and c twice), but this is not always the case. The bandwidth cost of this solution is $4*5+4*7=48$.

Now, suppose that the operator's main optimization criterion is to minimize the bandwidth cost of carrying 8Gb/s from a to c. The operator can use a standard algorithm for finding a minimum-cost network flow [1] whose output is illustrated in Figure 2(c). The bandwidth cost of this network flow is 36. The operator can use this network flow as an input to DMO in order to decompose it into a set of paths which minimizes the forwarding cost. Figure 2(d) shows a decomposition of the


RMO
4
2
8
(a) A network with available bandwidth on each link
(b) a minimum set of path (flow cost is 48)

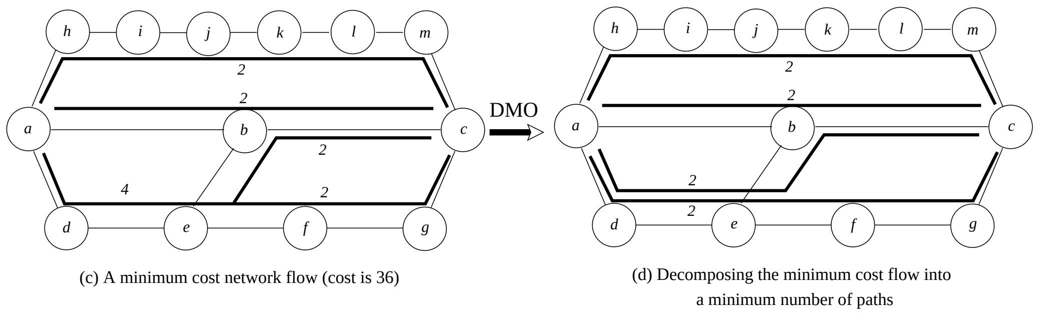

DMO
2
4
(c) A minimum cost network flow (cost is 36)
(d) Decomposing the minimum cost flow into a minimum number of paths

Fig. 2. An example for the two optimization problems considered in this paper, for accommodating an 8Mb/s traffic flow from node a to node c

Problem	Description
DMO(p)	Decompose a given network flow into a minimum number of paths.
DMO(n)	Decompose a given network flow into a set of paths traversing a minimum number of nodes.
RMO(p)	For a given traffic demand (source, destination and bandwidth demand), find a minimum set of paths, which satisfies it.
RMO(n)	For a given traffic demand (source, destination and bandwidth demand), find a set of paths, which satisfies this flow while traversing a minimum number of nodes.

TABLE I
DESCRIPTIONS OF THE PROBLEMS WE TACKLE IN THIS PAPER

network flow in Figure 2(c), which minimizes both the number of paths (4) and the number of nodes that carry these paths (22).

Figure 3 gives an overview of the scope of this paper, and Table I summarizes the four addressed problems. We make two theoretical and one practical contributions:

1) We are the first to define and solve the RMO(n) and DMO(n) problems. We present for these problems approximation algorithms with performance guarantees.
2) We show that simple greedy decomposition algorithms for DMO have an approximation ratio that is independent of the size of the network.
3) We compare the performance of the RMO and DMO algorithms. The purpose of this comparison is to better understand the trade-off between bandwidth efficiency and forwarding cost. This comparison allows us to identify an algorithm that has the best performance for both objectives.

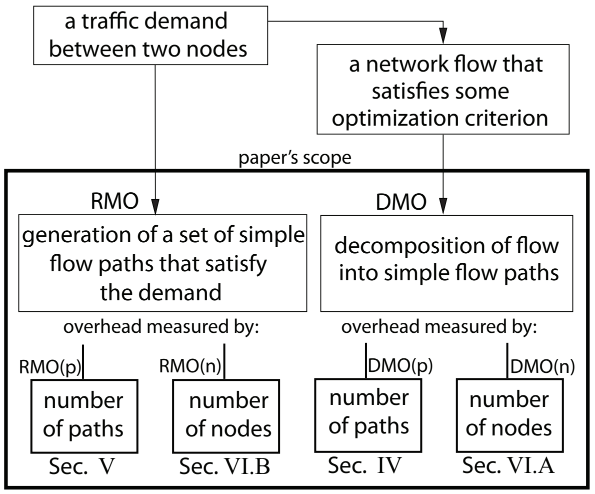

a traffic demand between two nodes
a network flow that satisfies some optimization criterion
paper's scope
RMO
DMO
generation of a set of simple flow paths that satisfy the demand
decomposition of flow into simple flow paths
overhead measured by:
overhead measured by:
RMO(p)
RMO(n)
DMO(p)
DMO(n)
number of paths
number of nodes
number of paths
number of nodes
Sec. V
Sec. VI.B
Sec. IV
Sec. VI.A

Fig. 3. The scope of this paper

Table II summarizes our main results from a computational complexity perspective. In this table, B denotes the bandwidth demand, b denotes the quantum of the edge capacities, *opt* denotes the value of the optimal solution and α is a tuning parameter. For each problem the table indicates a lower bound on its approximation ratio and the approximation ratio achieved by the algorithms we present for it. Throughout the paper we consider the minimization of the bandwidth cost as the bandwidth efficiency criterion. However, our results are applicable to any other bandwidth utilization criterion, such as throughput maximization or maximal load minimization.

problem	minimum bound	approximation ratio
DMO(p)	-	$O(\log(B/b))$
DMO(n)	-	$O(\log(B/b))$
RMO(p)	$3/2$	$O(\frac{B}{\text{opt}\cdot\alpha})$
RMO(n)	$3/2-\epsilon$	$O(\frac{B}{\alpha})$

TABLE II
OUR MAIN COMPUTATIONAL COMPLEXITY RESULTS

III. RELATED WORK

To the best of our knowledge, no prior work deals with minimizing the number of nodes traversed by paths that satisfy a given traffic demand (RMO(n)). Moreover, no prior work deals with the decomposition of a given network flow while minimizing the number of nodes traversed by the paths (DMO(n)). There are, however, a few works that address the DMO(p) and RMO(p) problems. We note that if minimizing the number of paths is not important, it is easy to decompose a given network flow with at most $O(|E|)$ paths [1].

In [6], the RMO(p) problem is addressed. In this work, the number of paths that satisfy a given bandwidth demand is minimized while guaranteeing an upper bound on the load imposed on the network links. This work presents an algorithm that may violate the maximum load bound. The violation gets smaller as the number of paths increases. The actual performance of the proposed algorithm is not studied.

The objective of [27] is to decompose a given (maximum) flow into a minimum number of paths. The authors prove that the problem is NP-hard, present several heuristics, and evaluate their performance using simulations. A greedy algorithm that iteratively decomposes the maximum flow path is shown to achieve the best performance. The problem of decomposing a given flow into a minimum number of paths is also studied in [26]. That paper is mainly concerned with decompositions that produce independent paths. Such paths are iteratively produced by reducing to 0 the flow on at least one edge during each step. Such decompositions are shown to have an approximation ratio of $n-1-\frac{n^2-3n+1}{m}$, where n is the number of vertices and m is the number of edges in the graph. Two decomposition algorithms are evaluated: one is the greedy algorithm, and the other chooses the shortest path during each step. As in [27], the greedy algorithm is shown to have the best performance. The flow decomposition problem has also been studied in [15]. Its main contribution is an approximation algorithm that decomposes all but an ϵ-fraction of a flow into at most O($1/\epsilon^2$) times the smallest possible number of decomposed paths.

Another relevant branch of work deals with the k-splittable flow problem. This problem can be viewed as the reverse version of our RMO(p) problem. An upper bound k on the number of paths is given and the objective is to maximize the satisfied bandwidth demand [4], [17], [18]. In [4], the directed graph version of this problem is proven to be NP-hard. Moreover, it is proven that it cannot be approximated within a factor of $3/2$. A 2-approximation algorithm is also presented. In [17], an optimal polynomial solution is given for the cases where k is constant and the graph has a special property called bounded treewidth [23]. In addition, a polynomial time approximation is presented for the case where k is part of the input. In [18], a comprehensive study of the k-splittable flow problem is presented and proven to be NP-hard for undirected graphs. Moreover, it is proven that for a constant k the problem cannot be approximated within a factor of 5/6. Finally, it is also proven that the problem is NP-hard for $2 \leq k \leq m-n+1$, and that it is polynomially solvable for any other k.

Several papers address the problem of minimizing the maximum load while bounding the number of paths. In [21], the splittable version of this problem is optimally solved. The number of paths is kept below $m+d$, where m is the number of edges and d is the number of demands.

ECMP (Equal Cost Multi-Path) is the standard approach for using multi-paths in today's IP networks. This concept has been recently adopted for Ethernet networks, mainly for datacenters (E.g., see [14]), and for MPLS [8]. The idea is that when multiple best paths exist between a [source,destination] pair, each switch/router can split the traffic between its next hops, e.g., using random hashing, without creating loops. In [22], a concept known as MTCP (Multipath TCP) is presented in the context of large datacenters. The idea is that by exploring multiple paths simultaneously, MTCP will lead to both higher network utilization and fairer allocation of capacity to flows. The main advantage of ECMP compared to the algorithms proposed in this paper is that it does not require a centralized controller. On the other hand, the algorithms proposed in this paper can take advantage of multiple paths that are not necessarily of equal cost. In the simulation section we show that due to this advantage, our algorithms perform much better than ECMP.

Another relevant branch of work deals with the embedding of virtual networks into physical networks. In this line of work, the nodes and links of the virtual network have to be embedded onto those of the physical network. Thus, one can view a virtual link as a traffic flow that has to be accommodated into the physical network. In most of the works that consider bandwidth constraints on the physical links, such as [11], [19], a single physical network path is chosen for each virtual link. In [28], a virtual link may be split over several physical paths, but no effort is made to minimize the number of such paths.

IV. DMO WITH PATH MINIMIZATION (DMO(P))

In this section we define the DMO(p) problem, discuss its computational complexity and propose approximation algorithms. Throughout the paper, a network flow that does not violate the capacity constraints is referred to as a *feasible network flow*. In addition, we refer to a flow carried by a path as a *single-path flow*.

Problem 1 [DMO(p)]:

Instance: Let $G=(V,E)$ be a directed graph. Let $s,t \in V$ be the source and target nodes. Let f be a feasible network flow from s to t and $f(e)$ be the bandwidth of f carried on edge $e \in E$.

Objective: Find a minimum path decomposition of f. A decomposition of f is a set $p_1, p_2, \ldots, p_k$ of simple directed paths from s to t, where path p_i carries a single-path flow of bandwidth w_i, and on each edge e the sum of bandwidths carried by the paths traversing the edge equals $f(e)$.

Using a reduction from the partition problem, the authors of [27] prove that DMO(p) is NP-hard in the strong sense. Thus, a pseudo-polynomial algorithm that finds an optimal solution for it is unlikely to exist.

Consider a greedy algorithm for the DMO(p), which iteratively decomposes the remaining network flow at each step into the widest feasible single-path flow. This intuitive algorithm was previously studied in [26] and [27]. In [26], this algorithm is proven to have an approximation ratio of $|V|-1-\frac{|V|^2-3|V|+2}{|E|}$. Our contribution here is to prove that this algorithm also yields an approximation ratio that does not depend on the size of the network.

In the following discussion we assume that the edge capacities are b-integral, where b is an integer greater than 0. For instance, b might be 1Kb/s or 1Mb/s. We show that the approximation ratio for the greedy algorithm is $\log(B/b)$, where B is the total bandwidth of the network flow. For dense networks, this approximation ratio is tighter than the one proposed in [26], since $|V|-1-\frac{|V|^2-3|V|+2}{|E|}$ might be in the order of several hundreds while $\log(B/b)$ is in the order of 10.

As indicated above, the input network flow for DMO(p) is f, while $f(e)$ is the value of f carried over edge e. Let $f(p)$ denote the value of a single-path flow carried over path p, i.e., $f(p) = \min_{e\in p}\{f(e)\}$. Let $f\backslash p$ be the network flow whose value $\forall e \in p$ is $f(e)-f(p)$ and its value $\forall e \notin p$ is $f(e)$.

The greedy algorithm iteratively finds a single path whose bandwidth is maximal until it reaches a total bandwidth of B or more.

Algorithm 1: (A greedy algorithm for DMO(p))

```
1) $B^0 \leftarrow B$, $f^0 \leftarrow f$, $P \leftarrow \phi$, $i \leftarrow 0$.
2) Repeat until $B^i = 0$:
    a) Choose the path $p$ that can provide the largest portion of $f^i$ from the source to the destination. This can be found using the extended Dijkstra algorithm [1] in time $O(|E|\log(|V|))$.
    b) $B^{i+1} \leftarrow B^i - f^i(p)$, $f^{i+1} \leftarrow f^i\backslash p$, $P \leftarrow P \cup p$, $i \leftarrow i+1$.
3) Return $P$.   □
```

It is easy to see that the algorithm returns a feasible solution that carries a total bandwidth of B, because on each path $p \in P$ we can route a bandwidth of $f^i(p)$, where i is the step during which p is selected.

During each step of the algorithm, the flow carried on at least one edge in f^i is reduced to 0. Thus, the number of steps is bounded by $|E|$. Since each step can be performed in time $O(|E|)$, the total running time of Algorithm 1 is $O(|E|^2\log(|V|))$.

Theorem 1: The approximation ratio of Algorithm 1 is $O(\log(B/b))$.

Proof: The proof is similar in spirit to the one used in [16] for the Minimum Set Cover problem. Let G' be the same as G, but with edge capacities scaled down by a factor of b. Denote the number of paths in the optimal solution by OPT and each path in the optimal solution by p_j^*, where $1 \le j \le OPT$. Let p_i be the path chosen by the algorithm in the i-th iteration. Since at each step the chosen path is the widest one, then for every j

$$f^i(p_i) \ge f^i(p_j^*).$$

Hence,

$$OPT \cdot f^i(p_i) \ge \sum_{j=1}^{OPT} f^i(p_j^*) \ge B^i/b.$$

The right inequality is due to the fact that the entire set of optimal paths can decompose the network flow f and hence any network flow f^i of value B^i. This leads to

$$\frac{1}{f^i(p_i)} \le \frac{OPT}{B^i/b}.$$

Each iteration reduces B^i by at least b units of bandwidth. Thus,

$$\frac{OPT}{B^i/b} \le \frac{OPT}{B/b-(i+1)}.$$

The left side of this inequality can be viewed as the "cost" of each b bandwidth units routed by p_i. Summing up the cost for all the sets of b bandwidth units in B results in the number of paths chosen by the algorithm, denoted as ALG. We now order the sets of b bandwidth units according to the order of the algorithm steps during which they are routed. Multiple sets that are routed at the same step are arbitrarily ordered. For each set k of b bandwidth units routed at step i, $\frac{OPT}{B/b-(i+1)} \le \frac{OPT}{B/b-(k+1)}$ holds. Hence, we get

$$\begin{aligned} ALG &= \textstyle\sum_{k=1}^{B/b} \frac{OPT}{B/b-(k+1)} = OPT \cdot (1 + \frac{1}{2} + \cdots + \frac{1}{B/b}) \\ &\le OPT \cdot \log B/b, \end{aligned}$$

which concludes the proof. ■

V. RMO with Path Minimization (RMO(p))

In this section we formally define the RMO(p) problem. Unlike DMO, here the network flow is not given in advance but only the traffic demand. We discuss the computational complexity of RMO and propose approximation algorithms with bounded performance guarantees.

Problem 2 [RMO(p)]:

Instance: Let $G = (V, E)$ be a directed graph. Let $c(e)$ be the bandwidth capacity of edge $e \in E$. Let $s, t \in V$ be the source and target nodes and $B \in \mathcal{R}^+$ be the bandwidth demand from s to t.

Objective: Find a minimum set of simple directed paths from s to t, that carry together a feasible network flow of B from s to t.

To solve RMO, we first construct a feasible network flow that satisfies the bandwidth demand. After the flow is constructed, it is decomposed into paths.

Theorem 2: RMO(p) is NP-complete.

Proof: We prove this by a reduction from DMO(p). Consider an instance of DMO(p) that consists of a directed graph G and a network flow f from s to t. We transform this instance into an instance of RMO(p) in the following way. We take the same graph G and set its edge capacities such that $\forall e \in E,\ c(e) = f(e)$. We take the bandwidth demand B of RMO(p) to be equal to the value of flow f of DMO(p) and consider the same source and destination nodes. By construction, in the resulting graph there is only one possible network flow of value B from s to t. This flow is exactly f of DMO(p). Hence, an optimal solution for the constructed RMO(p) instance is also an optimal solution for the original DMO(p) instance. ■

Theorem 3: RMO(p) cannot be approximated within a factor of 3/2.

Proof: In [4], a reduction from SAT to the 2-splittable flow problem is shown. In the 2-splittable flow problem, the objective is to find a maximum flow that can be decomposed into at most 2 paths. The reduction constructs a graph with source and destination nodes such that a satisfiable SAT instance, for which there is a truth assignment that satisfies all its clauses, yields a feasible flow with 2 paths that carry together 3 flow units. In contrast, an unsatisfiable SAT instance, for which there is no truth assignment that satisfies all its clauses, yields a feasible flow with 2 paths that carry together only 2 flow units. In the latter case, the flow can be augmented by a third path that carries 1 flow unit. Consequently, an unsatisfiable SAT instance yields a feasible flow of 3 paths that carry together only 3 flow units. In both cases we have flows of 3 units delivered by either 2 or 3 paths, which implies that even for $B = 3$ it is NP-hard to determine whether 2 or 3 paths are needed to accommodate the demand. Therefore, it is NP-hard to approximate RMO(p) with a ratio of $3/2$. ■

We now present an approximation algorithm for RMO(p), which uses the following observation:

Observation 1: A network flow of value B in a network with integral capacities can be decomposed into $\lceil B \rceil$ paths. □
At first glance, this observation does not seem to be very helpful, because B may be larger than the number $|E|$ of edges in the network, which is a straightforward upper bound on an optimal solution. However, we can scale down the edge capacities by a significant factor such that each unit of flow will be larger in relation to the total network flow. This scaling process reduces the original demand B into a small number that makes a solution of unit-flow paths more attractive.

Algorithm 2 below uses a parameter α for the scaling process. The algorithm finds a network flow whose value is slightly less than B using no more than $\left\lceil \frac{B}{\alpha} \right\rceil$ paths. Choosing a larger α would yield fewer paths whose total bandwidth is smaller.

Algorithm 2: (A basic scaling algorithm for RMO(p))

1) Scale the capacities by α, i.e., $\forall e \in E\ c'(e) \leftarrow \left\lfloor \frac{c(e)}{\alpha} \right\rfloor$.
2) Find a network flow f whose value is not larger than $\left\lceil \frac{B}{\alpha} \right\rceil$ in the scaled network.
3) Find any decomposition of f into paths. Let the resulting set of paths be $P = p_1, ..., p_k$, where path p_i carries a single-path flow of f_i.
4) Use every path $p_i \in P$ to carry a single-path flow of αf_i in the original graph. □

The network flow in Step 2 and its decomposition in Step 3 can be arbitrary. Furthermore, we denote the computational complexity of this step by $O(\text{Flow-Alg})$. The total computational complexity of Algorithm 2 is $O(\text{Flow-Alg} + |E| \cdot \frac{B}{\alpha})$, because the time complexity of the scaling process in Step 1 is linear in the size of the network and the time complexity of the decomposition process in Step 3 is $O(|E| \cdot \frac{B}{\alpha})$. Note that the above time complexity is only pseudo-polynomial because it depends on B. Algorithm 3 presented later refines this and yields a polynomial running time complexity.

Theorem 4: Algorithm 2 returns a set of at most $\left\lceil \frac{B}{\alpha} \right\rceil$ paths whose total bandwidth is at least $B - k^* \cdot \alpha$, where k^* is the number of paths in an optimal solution.

Proof: The scaled network has integral capacities. From Observation 1 follows that the decomposition step produces no more than $\left\lceil \frac{B}{\alpha} \right\rceil$ paths, which is the value of the flow found in Step 2. We now prove the lower bound on the bandwidth. Let $p_1, p_2, \ldots, p_{k^*}$ be the set of paths in an optimal solution. Each of them is a simple path from s to t. Each path p_i carries a single-path flow of w_i, where $\sum_i w_i = B$. Consider the same set of paths in the scaled network, and let each path p_i carry a single-path flow of $w'_i = \left\lfloor \frac{w_i}{\alpha} \right\rfloor$. This scaled solution is a feasible solution in the scaled network due to the following set of equations, which holds $\forall e \in E$:

$$\begin{aligned} c(e) &\geq \sum_{e \in p_i} w_i \\ c(e) &= \Delta + \sum_{e \in p_i} w_i,\ \Delta \geq 0 \\ \frac{c(e)}{\alpha} &= \frac{\Delta}{\alpha} + \frac{\sum_{e \in p_i} w_i}{\alpha},\ \Delta \geq 0 \\ \left\lfloor \frac{c(e)}{\alpha} \right\rfloor &\geq \left\lfloor \frac{\Delta}{\alpha} \right\rfloor + \left\lfloor \frac{\sum_{e \in p_i} w_i}{\alpha} \right\rfloor,\ \Delta \geq 0 \\ \left\lfloor \frac{c(e)}{\alpha} \right\rfloor &\geq \left\lfloor \frac{\sum_{e \in p_i} w_i}{\alpha} \right\rfloor . \end{aligned}$$

The first equation holds because the optimal solution must be feasible in the original graph. The second and third equations follow from the first one. The fourth equation holds because $\lfloor \sum x_i \rfloor \geq \sum \lfloor x_i \rfloor$, and the last one follows from the fourth.

We also note that:

$$\begin{aligned} \sum_{i=0}^{k^*} w_i &= B \\ \sum_{i=0}^{k^*} \frac{w_i}{\alpha} &= \frac{B}{\alpha} \\ \sum_{i=0}^{k^*} \left\lfloor \frac{w_i}{\alpha} \right\rfloor &\leq \left\lceil \frac{B}{\alpha} \right\rceil \\ \sum_{i=0}^{k^*} \alpha \left\lfloor \frac{w_i}{\alpha} \right\rfloor &\leq \alpha \left\lceil \frac{B}{\alpha} \right\rceil , \end{aligned}$$

where $\alpha\left\lceil\frac{B}{\alpha}\right\rceil$ is the value (bandwidth) of the flow returned by Algorithm 2. This can be lower bounded as follows:

$$\alpha\left\lceil\frac{B}{\alpha}\right\rceil \geq \alpha\cdot\sum_{i=1}^{k^*}\left\lfloor\frac{w_i}{\alpha}\right\rfloor \geq \alpha\cdot\sum_{i=1}^{k^*}\left(\frac{w_i}{\alpha}-1\right) = \sum_{i=1}^{k^*}(w_i-\alpha)$$
$$= \sum_{i=1}^{k^*} w_i - k^*\cdot\alpha = B - k^*\cdot\alpha.$$

■

Corollary 1: Let k^* be the number of paths in an optimal solution. For $\alpha = \frac{B}{k^*\cdot\beta}$, Algorithm 2 produces a solution with at most $k^*\cdot\beta+1$ paths whose value is no less than $B\cdot\left(1-\frac{1}{\beta}\right)$.

The parameter β can be considered as a tuning parameter. As β increases, the value of the output flow of Algorithm 2 approaches the original demand B, but the number of paths increases. Since k^* is not known in advance, it is not easy to find the value of α. One can try all values of k^*, and find the minimum one that yields a network flow whose total bandwidth is larger than $B\cdot\left(1-\frac{1}{\beta}\right)$. This requires running Algorithm 2 on all possible values of k^*, which is $O(|E|)$. To improve the total time complexity, Algorithm 3 below uses the output returned by Algorithm 2 for a given k as the initial network flow when running Algorithm 2 with $k+1$. This is possible because the scaling parameter α decreases as k increases. Thus, the capacities of the scaled network increase.

Algorithm 3: (A scaling approximation algorithm for RMO(p) using a tuning parameter β)

```
1) $k \leftarrow 1$.
2) Let $f$ be an initial network flow such that $f(e) \leftarrow 0$ for
   every $e \in E$.
3) While $k \leq |E|$ and the total value of $f$ is smaller than
   $B\cdot\left(1-\frac{1}{\beta}\right)$ do
   a) Run Algorithm 2 with a scaling factor $\alpha = \frac{B}{k\cdot\beta}$
      and use $f$ as the initial network flow for Step 2 in
      Algorithm 2.
   b) Set $f$ as the flow returned by Algorithm 2.
   c) $k \leftarrow k+1$.
4) Return the set paths output by the last execution of
   Algorithm 2.                                              □
```

Assuming that the capacities are integral, if Algorithm 3 is invoked with $\beta = B$, the resulting value of the network flow is guaranteed to be at least B. However, there is no guarantee on the number of paths it uses.

The running time complexity of each iteration of Step 3 is the time complexity of Algorithm 2. Since Algorithm 2 does not need to construct a flow from scratch, its running time is $O(|E|\cdot\frac{B}{\alpha})$. Since the number of iterations does not exceed $|E|$, the running time complexity of Algorithm 3 is $O(|E|^2\cdot\frac{B}{\alpha})$, i.e., $O(|E|^2\cdot k^*\cdot\beta)$.

Algorithm 2 and 3 have theoretical value because they have worst case performance guarantees. However, simulation results indicate that their actual average performance is not good. Specifically, when the bandwidth provided by the flow is close to B, the number of paths increases very rapidly. We therefore present another algorithm for RMO(p). While this algorithm has no worst case performance guarantee, its actual performance is shown later to be very good.

The main idea behind the new algorithm is to break the RMO(p) solution into two stages. First, a network flow that provides a bandwidth of at least B is found. Then, this flow is decomposed using Algorithm 1.

Algorithm 4: (A 2-phase algorithm for RMO(p))

```
1) Find an initial feasible network flow of bandwidth $B$ or
   more from $s$ to $t$.
2) Use Algorithm 1 for decomposing the flow into a
   minimum number of paths that provide bandwidth $B$.
3) Return the set of paths produced by Algorithm 1.          □
```

We now present several procedures for finding an initial network flow. In Section VII we compare the performance of Algorithm 4 using each of these procedures. All of the procedures produce a *maximum* network flow between s and t although the algorithm only requires that the bandwidth of the initial network flow will be greater than or equal to B. We found that starting with a maximum flow gives the algorithm greater flexibility in minimizing the number of paths. When we evaluated similar procedures that limit the bandwidth of the initial flow to B, the number of decomposed paths was larger.

The procedures for finding an initial network flow are as follows.

- The *Maximum Widest Path Flow (WIDE)* procedure: Here, to find an initial feasible network flow, the procedure iteratively augments the *widest path* available from s to t until the maximum flow is reached. If there are multiple paths, one of them is selected arbitrarily. The rationale behind this procedure is to greedily use the available paths in the network. The running time of this procedure is $O(|E|^2\log(|V|)\log(C_{max}))$ [1], where C_{max} is the maximal capacity of an edge in the network.
- The *Maximum Shortest Path Flow (SHORT)* procedure: Here, to find an initial feasible network flow, the procedure iteratively augments the *shortest path* available from s to t until the maximum flow is reached. If there are multiple paths, one of them is selected arbitrarily. This is the well-known Edmonds-Karp algorithm [10] for finding a maximum flow. The rationale behind this procedure is to use short paths, which traverse fewer nodes. The running time of this procedure is $O(|V||E|^2)$.
- The *Maximum Shortest Widest Path Flow (S-WIDE)* procedure: This procedure is similar to WIDE, except that when there is more than one path of maximum width in any iteration, the shortest is chosen. The rationale behind this procedure is to consume less bandwidth in each iteration as compared to WIDE, in the hope that the next iterations will be able to choose wider paths. This procedure can be implemented using a simple dynamic programming algorithm with a running time of $O(|V||E|^2\log(C_{max}))$.
- The *Maximum Widest Shortest Path Flow (W-SHORT)* procedure: This procedure is similar to SHORT, except that when there is more than one path of minimum length in any iteration, the widest of them is chosen. It

is expected that this procedure will need less iterations to achieve the maximum flow compared to SHORT. Hence, the decomposition algorithm is likely to use fewer paths. This procedure can be implemented using a simple dynamic programming algorithm with a running time of $O(|V||E|^2)$.

- The *Maximum Width/Length Path Flow (WID/LEN)* procedure: This procedure iteratively chooses the path with the largest width-length ratio. The rationale behind this procedure is to have a better trade-off between the width and length of the chosen paths compared to the previous procedures. This procedure can be implemented using a dynamic programming algorithm with a running time of $O(|V|^2|E|^3\log^2(|V|))$.

VI. DMO and RMO with Node Minimization

In many cases, a network operator seeks to minimize the number of nodes that carry the paths rather than the number of paths. This is because each node traversal requires one entry in the forwarding table of that node. In such a case, it may be better to set up many short paths rather than fewer long ones. To address such cases, we now change our optimization problem and view the forwarding cost as the number of nodes that carry the paths. More formally, given a set Π of simple directed paths from s to t, the forwarding cost by Π is measured by $\sum_{p\in\Pi}|p|$, where $|p|$ is the number of nodes along the path p.

A. DMO with Node Minimization (DMO(n))

We now show that DMO(n) can not be solved in polynomial time. Then, we propose an approximation algorithm for it.

Theorem 5: DMO(n) is NP-complete.

Proof: We show this using a reduction from DMO(p) to DMO(n). Given an instance of the former problem, we construct an instance of the latter. We set the source of DMO(n) to be a new node, s', which is connected to s using a chain of $|V||E|$ links whose capacity is B. The network flow of DMO(n) is carried over the new chain from s' to s, and then to t as the network flow in the DMO(p) instance. We now show that the minimum-node decomposition of the DMO(n) flow, P_n^*, has the same number of paths as the number of paths in the minimum-path decomposition of the DMO(p) flow, P_p^*. First, $|P_n^*|\geq|P_p^*|$ must hold, because otherwise P_p^* is not a minimum-path decomposition in DMO(p). Second, if $|P_n^*|>|P_p^*|$ then P_p^* induces a decomposition for the DMO(n) flow with a smaller number of nodes than that imposed by P_n^* (because each additional path in the DMO(n) decomposition increases the number of nodes by $|V||E|$, which is greater than the number of nodes of any decomposition in DMO(p)).

Therefore, an optimal solution for the DMO(p) instance can be derived from an optimal solution for the constructed DMO(n) instance. ■

Algorithm 1 can be modified to approximate DMO(n) with the same approximation ratio $O(\log(B/b))$. The idea is to choose in each iteration the path with the greatest ratio between the bandwidth it carries and the number of nodes it traverses:

Algorithm 5: (A greedy algorithm for DMO(n))

```
1) $B^0 \leftarrow B$, $f^0 \leftarrow f$, $P \leftarrow \phi$, $i \leftarrow 0$.
2) Repeat until $B^i = 0$:
    a) Choose the path $p$ from the source to the destination for which $f^i(p)/n_p$ is maximum, where $f^i(p)$ is the bandwidth of $p$ in $f^i$ and $n_p$ is the number of nodes $p$ traverses.
    b) $B^{i+1} \leftarrow B^i - f^i(p)$, $f^{i+1} \leftarrow f^i\backslash p$, $P \leftarrow P\cup p$, $i \leftarrow i+1$.
3) Return $P$.                                              □
```

Algorithm 5 has the same computational complexity as Algorithm 1.

Theorem 6: The approximation ratio of Algorithm 5 is $O(\log(B/b))$.

Proof: The proof is similar to that of Theorem 1. Denote the number of paths in the optimal solution by OPT and each path in the optimal solution by p_j^*, where $1\leq j\leq OPT$. Let p_i be the path chosen by the algorithm in step i, and u_p^i be the ratio $f^i(p)/n_p$. The path chosen in each step is the one with the greatest ratio u. Thus, for every j $u_{p_i}^i \geq u_{p_j^*}^i$ and $u_{p_i}^i\cdot n_{p_j^*}\geq u_{p_j^*}^i\cdot n_{p_j^*}$ hold. Therefore,

$$OPT\cdot u_{p_i}^i \geq \sum_{j=1}^{OPT} u_{p_j^*}^i\cdot n_{p_j^*}\geq B^i/b. \tag{1}$$

The second inequality holds because $u_{p_j^*}^i\cdot n_{p_j^*}=f^i(p_j^*)$ and because the entire set of optimal paths is a decomposition of the network flow f and, therefore, of any network flow f^i. This leads to

$$\frac{1}{u_{p_i}^i}\leq\frac{OPT}{B^i/b}\leq\frac{OPT}{B/b-(i+1)}.$$

The rest of the proof is identical to the proof of Theorem 1. ■

B. RMO with Node Minimization (RMO(n))

Using a proof similar to that of Theorem 5 for DMO(n), it can be shown that

Theorem 7: RMO(n) is NP-complete.

Theorem 8: An α-approximation algorithm for RMO(n) yields an $(\alpha+\epsilon)$-approximation algorithm for RMO(p), where $\epsilon>0$ is arbitrarily small.

Proof: The reduction used in the proof of Theorem 5 can be used again, but this time the length of the added chain is $M=|V||E|\cdot\alpha\cdot(1/\epsilon)$. If we have an α-approximation algorithm for RMO(n), we can apply it to the new flow constructed by the reduction. Let ALG_p and ALG_n be the number of paths and the number of nodes in the solutions found by the two approximation algorithms. Let OPT_p and OPT_n be the number of paths and the number of nodes in the corresponding optimal solutions. From the reduction it is obvious that

$$ALG_n\geq ALG_p\cdot M.$$

On the other hand, we have:

$$ALG_n\leq\alpha\cdot OPT_n\leq\alpha\cdot(OPT_p\cdot M+X),$$

where X is the number of nodes in the minimum-node decomposition on the original graph, which is obviously smaller than $|V||E|$. The right inequality holds because in the proof of Theorem 5 we showed that the number of paths of the minimum-node decomposition must be equal to the number of paths in the minimum-path decomposition. Hence we have

$$\alpha(OPT_p + \frac{X}{M}) \geq ALG_p$$
$$OPT_p(\alpha + \epsilon) \geq ALG_p.$$

■

From Theorems 3 and 8 we derive the following corollary:

Corollary 2: RMO(n) cannot be approximated within a factor of $3/2 - \epsilon$.

We now present an approximation algorithm for RMO(n). The algorithm is based on Algorithm 2 for RMO(p). In Algorithm 2 each flow unit is transformed into a path. Therefore, if the network nodes were assigned a cost of 1, the number of nodes for the decomposed paths returned by Algorithm 2 would be equal to the cost of their total network flow. Hence, we shall modify Algorithm 2 to find a minimum cost network flow before it is decomposed. Clearly, Corollary 1 still holds for this minimum cost flow version of the algorithm.

Since an algorithm for finding a minimum cost network flow addresses the case where the edges, rather than the nodes, have a cost, we will consider the following simple reduction. Consider a network where every unit of flow on a node v incurs a cost of 1. Every node v is transformed into two nodes, v_i and v_o, connected by an edge $v_i \to v_o$ with infinite capacity and a cost of 1. All the other edges have zero cost. All edges going into v will go into v_i and all edges from v will go out from v_o.

Algorithm 6: (A scaling algorithm for RMO(n))

1) Assign to each node in the network a cost of 1.
2) Transform the network to one with costs on the edges (as described above).
3) Add a source node s' and an edge $s \to s'$ with capacity B.
4) For $k = 1 \ldots |E|$.
   a) Run a minimum cost network flow version of Algorithm 2 with a scaling factor $\alpha = \frac{B}{k \cdot \beta}$.
   b) Store the result as f_k.
5) Return f_i with minimum cost whose value is at least $B\left(1 - \frac{1}{\beta}\right)$. □

Theorem 9: The solution returned by Algorithm 6 has a value greater than $B\left(1 - \frac{1}{\beta}\right)$ and a cost smaller than $\beta k^* N^*$, where k^* and N^* are the number of paths and the number of nodes in the optimal solution.

Proof

Corollary 1 holds during the iteration where $k = k^*$. Hence, the value of f_{k^*} is $\geq B\left(1 - \frac{1}{\beta}\right)$. Let $p_1, p_2, \ldots, p_{k^*}$ be the set of paths in an optimal solution. The optimal scaled solution is feasible. The value of this solution is the number of nodes in $p_1, p_2, \ldots, p_{k^*}$. Since every path p_i carries a single-path flow of w_i, where $\sum_i w_i = B$, this value is $\sum_{i=1}^{k^*} |p_i| > \max_i |p_i|$. In addition, the cost of the scaled optimal solution is

$$\sum_{i=1}^{k^*} \left\lfloor \frac{w_i}{\alpha} \right\rfloor |p_i| < \left\lfloor \frac{B}{\alpha} \right\rfloor \max_i |p_i|.$$

Clearly, the cost of f_{k^*} is less than that of the scaled optimal solution. Hence,

$$\frac{f_{k^*}}{N^*} < \frac{k^* \cdot \beta \cdot \max_i |p_i|}{\max_i |p_i|} = k^* \cdot \beta.$$

□

As in RMO(p), the above algorithm for RMO(n) has theoretical value. However, our simulation results indicate that its actual average performance is not good enough. Therefore, we present an additional algorithm that has no worst case performance guarantee, but a very good actual performance.

The algorithm is similar to Algorithm 4 presented for RMO(p). Its main idea is to break the RMO(n) solution into two stages. First, a network flow that provides a bandwidth of at least B is found. Then, this flow is decomposed using Algorithm 5.

Algorithm 7: (A 2-phase algorithm for RMO(n))

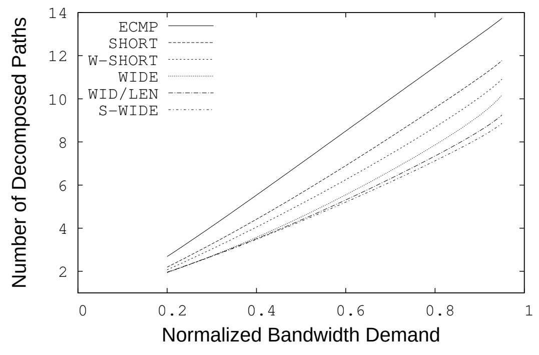
1) Find an initial feasible network flow of bandwidth B or more from s to t.
2) Use Algorithm 5 for decomposing the network flow into a set of paths that traverse a minimum number of nodes and deliver together a bandwidth of B.
3) Return the set of paths produced by Algorithm 5. □

The initial network flow can be found by one of the five procedures (WIDE, SHORT, S-WIDE, W-SHORT, and WID/LEN) described in Section V.

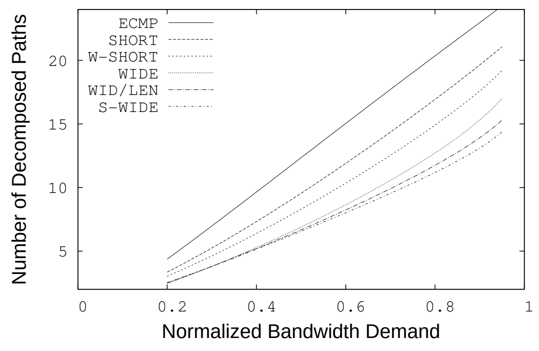
VII. Simulation Study

In this section we evaluate the performance of the algorithms for RMO and DMO. We first examine the performance of the two variants of the RMO algorithms. Then we evaluate the trade-off between the bandwidth cost and the forwarding cost of a network flow by comparing the performance of the RMO algorithms to the performance of the DMO algorithms as they apply to a network flow of minimum bandwidth cost.

We use the BRITE simulator [20] to simulate network domain topologies according to the “preferential attachment model” of [7]. This model captures two important characteristics of network topologies: incremental growth and preferential connectivity of a new node to well-connected existing nodes. These characteristics yield a power-law degree distribution of the nodes. In addition, we also run our algorithms on actual ISP topologies, as inferred from the RocketFuel project [24]. These topologies reflect better the model presented in [2]. For each synthetic or realtopology, we generate a bandwidth demand between a source and a destination. The characteristics of the simulated topologies and the methods for choosing the bandwidth demands are described for each setting. A network topology together with a bandwidth demand are considered as one simulation instance. We apply the various algorithms for each such instance.



(a) 100 nodes with average degree = 5



(b) 100 nodes with average degree = 10

Fig. 4. The number of paths found by Algorithm 4 with the various procedures as a function of the normalized bandwidth demand for various sizes of network domains

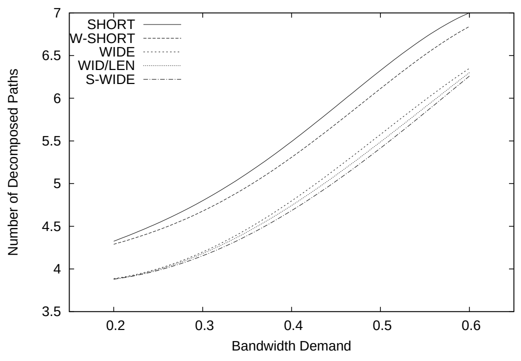
A. Minimizing the Number of Paths

Figure 4 depicts the number of paths over which the required bandwidth can be delivered as a function of the bandwidth demand for networks with 100 nodes whose average degree is 5 links (Figure 4(a)), and networks with 100 nodes whose average degree is 10 links (Figure 4(b)). For each such network, the edge capacities are uniformly distributed in $[0.5C, 1.5C]$. C is a normalizing factor for the edge capacities and the volume of bandwidth demands. The y-axis of all the graphs in Figure 4 represents the number of decomposed paths produced by the various algorithms. The x-axis represents the normalized bandwidth demand from s to t, i.e., the bandwidth demand divided by the value of the largest maximum network flow that exists between any pair of nodes in the network. For each network instance and for each average bandwidth value B, we generate 100 instances of bandwidth demands that are uniformly distributed on the interval $[0.9B, 1.1B]$. For each demand, the source and destination nodes are uniformly selected from among the network nodes, and the five variants of Algorithm 4 are executed. As a benchmark, we also simulate the well known equal cost multi-path (ECMP) algorithm. For ECMP, the bandwidth of a traffic flow is equally divided between the paths whose length/cost are minimum. If the total bandwidth of the least-cost paths is insufficient, the set of second least-cost paths is used for the remaining bandwidth, and so on.

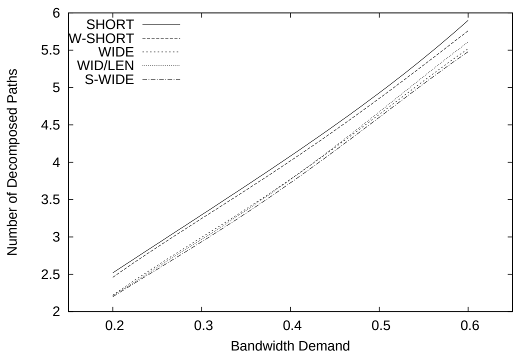
As clearly indicated by all the graphs in Figure 4, ECMP always produces more paths than Algorithm 4 under any flow construction scheme. In addition, it is evident that Algorithm 4 minimizes the number of paths needed for delivering the requested bandwidth when it uses S-WIDE in Step 1. W-SHORT performs better than SHORT because it produces network flows with larger average bandwidth on each edge. This allows Algorithm 1 (in Step 2 of Algorithm 4) to choose wider, and consequently fewer, paths. S-WIDE and WID/LEN perform better than WIDE because they take into account the length of the paths. Hence, less bandwidth is consumed during each iteration, and more bandwidth is left for latter iterations. Consequently, wider paths are found. S-WIDE is still slightly better than WID/LEN since it chooses wider paths. The above results are consistent across all network sizes and average node degrees, which indicates that the network size and node average degree do not have a significant impact on the relative characteristics of the network flows generated by the 5 procedures. For example, W-SHORT always chooses wider paths than SHORT, and S-WIDE always chooses shorter paths than WIDE, regardless of the network size or the node average degree. This makes W-SHORT and S-WIDE perform better than SHORT and WIDE, respectively. To continue the example, since it is better to choose wider paths than shorter ones when generating a network flow, WIDE and S-WIDE will always perform better than SHORT and W-SHORT, regardless of the network size and node average degree.

It is evident from Figure 4 that he number of decomposed paths increases linearly with the bandwidth demand. For a network with 100 nodes and an average degree of 10 (Figure 4(b)), the number of decomposed paths as well as the slope of the curves are almost doubled compared to that of networks with an average degree of 5 (Figure 4(a)). This is because we use larger bandwidth demands (recall that the bandwidth demand shown in the graphs is normalized to the largest maximum network flow in the network, which increases with the node degree). In addition, the relative difference between the number of paths using S-WIDE and the number of paths using SHORT increases: it is now roughly 50% compared to 25% for a network with an average degree of 5 (Figure 4(a)). This is because the number of possible paths between any two network nodes significantly increases. This allows S-WIDE to find wider augmenting paths. Hence, the network flow is constructed with fewer iterations, which is translated into a smaller number of decomposed paths.

In Figure 5 we examine how the distance between the source and destination influences the number of decomposed paths. As in Figure 4, the y-axis represents the number of decomposed paths for each procedure. The x-axis represents the distance between the source and destination divided by the diameter of the network. The network has 100 nodes and an average degree of 5. We generate 100 network instances of this size. For each instance we generate 100 bandwidth demands. For each demand, the distance between the source and the destination is assigned a given distance value with $\pm 10\%$ variation. The average normalized bandwidth for each demand is 0.6. For all of the procedures, the number of decomposed paths



(a) Exodus ISP, 80 routers with average degree of 1.8



(b) Telstra ISP, 115 routers with average degree of 1.3

Fig. 6. The number of paths found by Algorithm 4 with the various procedures for real ISP topologies

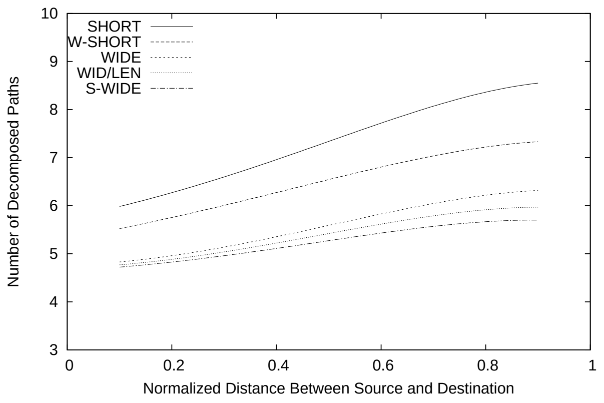


Fig. 5. The number of paths found by Algorithm 4 with the various procedures as a function of the distance between source-destination pairs

increases with the distance. Consequently, the capacities of the paths between the source and destination decrease. Thus, each decomposed path can carry less bandwidth on the average. The number of decomposed paths increases more moderately for WIDE, S-WIDE, and WID/LEN. This is because the source-destination distance has a smaller effect on the length of the widest path between them than on the length of the shortest path between them.

Furthermore, the minimum link capacity in the generated networks is 40. This is the reason for the steep change in bucket 40-50. Since the bandwidth carried by a path is dominated by the minimum of link capacities on the paths the curve for the SHORT procedure between 40 and 140 indeed resembles the probability function of the minimum of the uniform random variables. The S-WIDE curve has a different shape since it seeks to maximize the path bandwidth. The S-WIDE curve peaks around the mean of the link capacity. Below 40 are the paths whose bandwidth is the residual capacity of links on which other paths selected in earlier iteration pass.

To validate the results from the synthetic graphs, we present in Figure 6 results for real AS topologies, as inferred from the RocketFuel project [24]. These topologies reflect the model presented in [2], which may better represent a router-level ISP topology. We used the following network topologies:

1) Exodus ISP, which consists of 80 routers with average degree of 1.8
2) Telstra ISP, which consists of 115 routers with average degree of 1.3

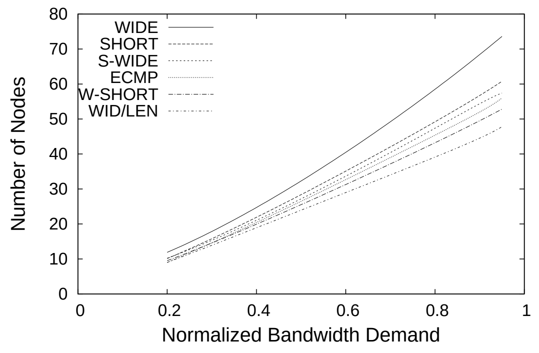
The bandwidth demands are generated as described for Fig. 4. Figure 6 shows the performance of Algorithm 4 with the various procedures. We can see that the relative performance rank is the same as for the synthetic graphs (Figure 4) that reflect the preferential attachment model. In the Telstra topology (Figure 6(b)), the performance differences are smaller because of its lower link degree, which substantially reduces the path diversity in the network.

B. Minimizing the Number of Nodes

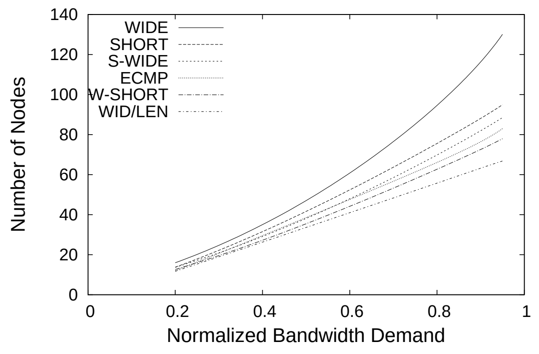
We now examine the performance of Algorithm 7, the goal of which is to minimize the number of nodes. Figure 7 depicts the number of nodes traversed by all of the paths that deliver the required bandwidth as a function of the bandwidth demand for the network domains considered in Figure 4. The network instances and bandwidth demands are generated as described for Fig. 4.

It is clear that Algorithm 7 gives the best performance when it uses WID/LEN for finding an initial network flow. S-WIDE, which was shown to yield the smallest number of paths, produces solutions with roughly 20% more nodes than WID/LEN. This result is consistent for all three routing domain sizes.

The advantage of WID/LEN over S-WIDE indicates that it is better to choose shorter paths rather than fewer wide ones in order to minimize the number of nodes. This insight is supported by the results of WIDE, which yields the worst performance. This is because WIDE is the only procedure that does not take into account the length of the chosen paths. Despite of this shortcoming, S-WIDE is slightly better than SHORT. This indicates that the number of paths still influences the number of nodes. Another evidence to the importance of using shortest paths for minimizing the number of nodes is the ECMP's performance curve, which is in the middle of Figure 7, and not the worst as in Figure 4.



(a) Num. nodes = 100, average degree = 5



(b) Num. nodes = 100, average degree = 10

Fig. 7. The number of nodes found by Algorithm 7 with the various procedures as a function of the normalized bandwidth demand for various sizes of network domains

C. *The Trade-Off Between Bandwidth Cost and Forwarding Cost*

We now study the trade-off between the bandwidth cost and the forwarding cost of a network flow. To this end, we focus on the following three questions:

- What is the extra forwarding cost when the main target is minimizing the bandwidth cost?
- What is the extra bandwidth cost when the main target is minimizing the forwarding cost?
- How do the various procedures perform with respect to this trade-off?

Finding the minimum-cost network flow in general networks is a well-studied problem [1], [10], [13]. In what follows we use the well-known Edmonds-Karp algorithm [10]. This algorithm iteratively adds to the constructed network flow the least cost path until the bandwidth demand is satisfied. We refer to this procedure as COST. We decompose the network flow using Algorithms 1 and 5 to find a solution with a minimum number of paths and nodes, respectively. Note that the difference between COST and SHORT is that SHORT constructs a maximal network flow while COST returns a network flow of bandwidth B. Consequently, the decomposition algorithm has less flexibility in the latter case.

We use the same simulation setting as described earlier, and assign an equal cost to each flow unit on every link. We then determine the bandwidth cost by summing up the cost for all links. Figures 8(a) and 8(b) show the trade-off between the bandwidth cost and the number of decomposed paths for each of the six procedures for Step 1 of Algorithm 4. The results are shown for networks with 100 nodes whose average node degree is 5 or 10. The x-axis represents the bandwidth cost normalized by the value of the actual bandwidth demand, while the y-axis shows the number of decomposed paths. The results are shown for a normalized bandwidth demand of 0.6.

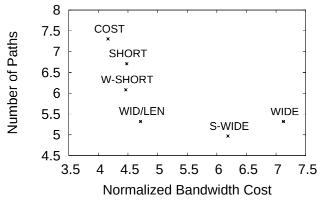
As expected, for both network sizes, the bandwidth cost is minimized using COST, but yields the largest number of paths. S-WIDE minimizes the number of paths, but its bandwidth cost is 50% more than COST. From Figures 8(a) and 8(b) we conclude that WID/LEN yields a very good trade-off between these two extremes. Its bandwidth cost is only 10% more than that of COST while it has only 5% more paths than S-WIDE.

Figures 8(c) and 8(d) show the trade-off between the bandwidth cost of a network flow and the aggregated number of nodes that participate in the setup and maintenance of all paths that carry this flowfor each of the six procedures for Step 1 of Algorithm 7. The results are shown again for networks with 100 nodes whose average degree is 5 or 10 and a normalized bandwidth demand of 0.6.As noted above, WID/LEN is the best procedure in terms of the aggregated load it imposes on the network nodes. Moreover, its bandwidth cost is almost as small as that of COST. Therefore, WID/LEN yields the best trade-off between bandwidth cost and forwarding cost.

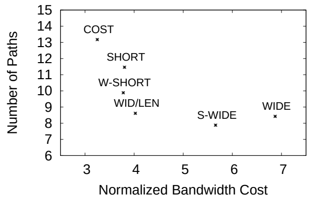
Our conclusion from these simulations is that WID/LEN is the procedure of choice for Step 1 of both Algorithm 4 and Algorithm 7.

VIII. Conclusions

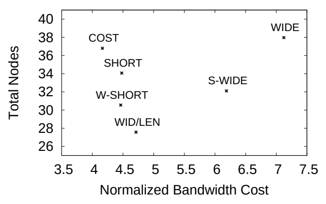
In order to improve bandwidth utilization, it is often desirable to split one traffic flow over multiple paths. This concept is supported today by state-of-the-art network technologies, such as optical networks, MPLS and datacenter SDNs. However, in such a case the network nodes need to spend more forwarding resources for every traffic flow. This raises two important optimization problems, related to splitting of a traffic flow into multiple paths while minimizing the associated forwarding cost: Decomposition with Minimum forwarding Overhead (DMO), and Routing with Minimum forwarding Overhead (RMO). We showed that both problems are NP-hard, and presented approximation algorithms. The DMO approximations were shown to perform very well, whereas the RMO approximations were shown to perform not as well. We presented efficient practical heuristics for RMO. These heuristics first find an initial network flow and then decompose it using our DMO approximation. The procedure for selecting the initial network flow was shown to have a critical impact on the performance of the algorithm. While S-WIDE was shown to be preferable when the main optimization criterion is to minimize the number of paths, WID/LEN gave the best trade-off between bandwidth cost and forwarding overhead. We also showed that these RMO algorithms perform much better than the ECMP algorithm, which is used today in many networks for routing a flow over multiple paths.



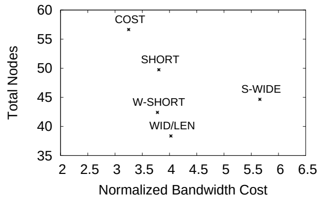
(a) 100 nodes, avg. degree = 5



(b) 100 nodes, avg. degree = 10



(c) 100 nodes, avg. degree = 5



(d) 100 nodes, avg. degree = 10

Fig. 8. The trade-off between the bandwidth cost and the number of paths/nodes that carry this bandwidth

References

[1] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin. *Network Flows: Theory, Algorithms, and Applications*. Prentice Hall, February 1993.

[2] D. Alderson, L. Li, W. Willinger, and J. C. Doyle. Understanding internet topology: principles, models, and validation. *IEEE/ACM Transactions on Networking*, 13(6):1205–1218, 2005.

[3] D. Awduche et al. Requirements for traffic engineering over MPLS. IETF RFC 2702, September 1999.

[4] G. Baier, E. Köhler, and M. Skutella. The k-splittable flow problem. *Algorithmica*, 42(3-4):231–248, 2005.

[5] D. Banerjee and B. Mukherjee. Wavelength-routed optical networks: linear formulation, resource budgeting tradeoffs, and a reconfiguration study. *IEEE/ACM Transactions on Networking*, 8(5), October 2000.

[6] R. Banner and A. Orda. Efficient multipath-routing schemes for congestion minimization. Technical report, Technion – Israel Institute of Technology, 2004.

[7] A. Barabasi, R. Albert, and H. Jeong. Scale-free characteristics of random networks: the topology of the World Wide Web. In *Physica A: Statistical Mechanics and Its Applications*, volume 281, pages 69–77, June 2006.

[8] S. Bryanti et al. Flow-aware transport of pseudowires over an MPLS packet switched network. IETF RFC 6391, November 2011.

[9] A. Curtis, J. Mogul, J. Tourrilhes, P. Yalagandula, P. Sharma, and S. Banerjee. Devoflow: scaling flow management for high-performance networks. In *SIGCOMM*, 2011.

[10] J. Edmonds and R. M. Karp. Theoretical improvements in algorithmic efficiency for network flow problems. *Journal of the ACM*, 19(2):248–264, 1972.

[11] J. Fan and M. Ammar. Dynamic topology configuration in service overlay networks: A study of reconfiguration policies. In *INFOCOM, Proceedings IEEE*, pages 1–12, 2006.

[12] O. Gerstel, R. Ramaswami, and G. H. Sasaki. Cost-effective traffic grooming in WDM rings. *IEEE/ACM Transactions on Networking*, 8:618–630, 2000.

[13] A. Goldberg and R. Tarjan. Finding minimum-cost circulations by canceling negative cycles. *Journal of the ACM*, 36(4):873–886, 1989.

[14] A. Greenberg et al. VL2: a scalable and flexible data center network. In *SIGCOMM'2009, Barcelona, Spain*.

[15] T. Hartman, A. Hassidim, H. Kaplan, D. Raz, and M. Segalov. How to split a flow? In *INFOCOM, Proceedings IEEE*, pages 828–836. IEEE, 2012.

[16] D. S. Hochbaum, editor. *Approximation Algorithms for NP-Hard Problems*. PWS Publishing Co., Boston, MA, USA, 1997.

[17] R. Koch, M. Skutella, and I. Spenke. Maximum k-splittable s,t-flows. *Theory of Computing Systems*, 43(1):56–66, 2008.

[18] R. Koch and I. Spenke. Complexity and approximability of k-splittable flows. *Theoretical Computer Science*, 369(1):338–347, 2006.

[19] J. Lu and J. Turner. Efficient mapping of virtual networks onto a shared substrate. *Washington University in St. Louis, Tech. Rep*, 2006.

[20] A. Medina, A. Lakhina, I. Matta, and J. Byers. BRITE: An approach to universal topology generation. In *Proceedings of MASCOTS*, 2001.

[21] V. S. Mirrokni, M. Thottan, H. Uzunalioglu, and S. Paul. A simple polynomial time framework to reduced path decomposition in multi-path routing. In *IEEE INFOCOM*, 2004.

[22] C. Raiciu et al. Improving datacenter performance and robustness with multipath tcp. In *SIGCOMM'2011, Toronto, Canada*.

[23] N. Robertson and P. D. Seymour. Graph minors. II. Algorithmic aspects of treewidth. *Journal of Algorithms*, 7:309–322, 1986.

[24] N. Spring, R. Mahajan, and D. Wetherall. Measuring ISP topologies with RocketFuel. In *Proceedings of the ACM SIGCOMM*, August 2002.

[25] A. Tavakoli, A. Casado, M. Koponen, and S. Shenker. Applying ”nox” to the datacenter”. In *HotNets VIII*, 2009.

[26] B. Vatinlen, F. Chauvet, P. Chrétienne, and P. Mahey. Simple bounds and greedy algorithms for decomposing a flow into a minimal set of paths. *European Journal of Operational Research*, 185(3):1390–1401, 2008.

[27] H. Wang, J. Lou, Y. Chen, Y. Sun, and X. Shen. Achieving maximum throughput with a minimum number of label switched paths in MPLS networks. In *Proceedings of ICCCN*, pages 187–192, 2005.

[28] M. Yu, Y. Yi, J. Rexford, and M. Chiang. Rethinking virtual network embedding: substrate support for path splitting and migration. *SIGCOMM Comput. Commun. Rev.*, 38(2):17–29, March 2008.

[29] K. Zhu and B. Mukherjee. Traffic grooming in an optical WDM mesh network. *Selected Areas in Communications, IEEE Journal on*, 20(1), 2002.

Gabi Nakibly received the B. Sc. in Information Systems engineering (summa cum laude) and PhD in Computer Science from the Technion - Israel Institute of Technology, Haifa, Israel, in 1999 and 2008, respectively. Gabi is a professional fellow in the National EW Research & Simulation Center at Rafael Advanced Defense Systems. He also serves as an adjunct researcher and lecturer in the Computer Science department at the Technion. Gabi received his Ph.D. in computer Science in 2008 from the Technion, and he is a recipient of the Katzir Fellowship. His main research interests include network security and traffic engineering.

Reuven Cohen received the B.Sc., M.Sc. and Ph.D. degrees in Computer Science from the Technion - Israel Institute of Technology, completing his Ph.D. studies in 1991. From 1991 to 1993, he was with the IBM T.J. Watson Research Center, working on protocols for high speed networks. Since 1993, he has been a professor in the Department of Computer Science at the Technion. He has also been a consultant for numerous companies, mainly in the context of protocols and architectures for broadband access networks. Reuven Cohen has served as an editor of the IEEE/ACM Transactions on Networking and the ACM/Kluwer Journal on Wireless Networks (WINET). He was the co-chair of the technical program committee of Infocom 2010 and headed the Israeli chapter of the IEEE Communications Society from 2002 to 2010.

Liran Katzir received his B.A., M.A., and Ph.D. degrees in computer science from the Technion Israel Institute of Technology, Haifa, Israel, in 2001, 2005, and 2008 respectively. His PhD. thesis is about scheduling in advanced wireless networks. Dr. Katzir is now a member of the networking research group in the Technion's CS department, working on technologies for the fourth generation cellular network.