Optimizing Flow Bandwidth Consumption with Traffic-diminishing Middlebox Placement

Yang Chen
Temple University
yang.chen@temple.edu

Jie Wu
Temple University
jiewu@temple.edu

Bo Ji
Temple University
boji@temple.edu

ABSTRACT

The implementation of network services is changed from dedicated hardware to software middleboxes with the evolution of Network Function Virtualization (NFV). The placement of such middleboxes are complicated not only by the selection of multiple available hosting servers, but also by the traffic-changing effect of middleboxes. In this paper, we address the placement problem of a single type of traffic-diminishing middlebox (e.g., spam filters), where the objective is to minimize the total bandwidth consumption when the total number of placed middleboxes is limited. We prove the NP-hardness of checking the feasibility of our problem in general topologies. Then we propose a greedy solution and prove that it is performance-guaranteed when it generates a feasible deployment. Next we narrow down to tree-structured networks and propose an optimal dynamic programming based strategy. In order to improve the time efficiency, we also introduce an efficient greedy solution with an intuitive insight. Extensive simulations are conducted on a real-world dataset to evaluate the performance of our algorithms.

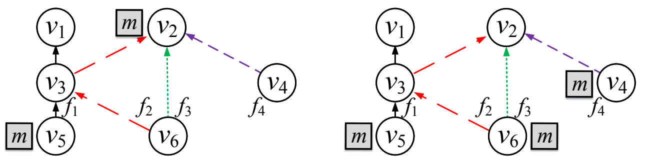
KEYWORDS

Bandwidth consumption, traffic-diminishing effect, middlebox deployment, NFV.

ACM Reference Format:
Yang Chen, Jie Wu, and Bo Ji. 2020. Optimizing Flow Bandwidth Consumption with Traffic-diminishing Middlebox Placement. In *49th International Conference on Parallel Processing - ICPP (ICPP '20), August 17–20, 2020, Edmonton, AB, Canada.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3404397.3404458

1 INTRODUCTION

Network Function Virtualization (NFV) changes the way we implement network services from expensive hardware to software functions (middleboxes), which run on switch-connected commodity servers [18]. The choice of middlebox service location is complicated by not only the availability of multiple hosting servers but also the *traffic-changing* effect of middleboxes [22]. Middleboxes with traffic-diminishing capability are quite common. For example, the Citrix CloudBridge Wide Area Network Optimizer reduces traffic volume by up to 80% by compressing traffic [2]. Redundancy Eliminator would reduce the difference between peak and minimum traffic more significantly by 25% for the university and by 52% for the data center [15]. *Spam filters* intercept all suspicious flows by cutting down 100% spam rates. Link bandwidth is a valuable resource in most networks such as data centers [21], WANs [4], and LANs [13]. Efficiently placing such kind of traffic-diminishing middleboxes is important for today's high-performance networks/systems [14]. Additionally, server resources for running middleboxes, such as CPU and memory, are also valuable and finite



(a) Two middleboxes. **(b) Three middleboxes.**

Figure 1: A motivating example.

in today's networks [1]. Then there is usually a constraint on the total number of middleboxes that we can deploy [26].

In this paper, we aim at minimizing the total flow bandwidth consumption by placing a limited number of a single type of traffic-diminishing middleboxes with a given number of copies. The flow bandwidth consumption is defined as the sum of a flow's occupied bandwidth on each link along its path. A middlebox does not have a capacity limit. Note that even under a simplified assumption of only one type of middleboxes, our formulated problem is non-trivial (Section V). There is a delicate trade-off between saving more link bandwidth and sharing more middleboxes among flows. Intuitively, deploying traffic-diminishing middleboxes as close to flows' sources as possible along all paths consumes less link bandwidth resources. However, this simple strategy reduces middleboxes' sharing opportunities and forces the launch of more middleboxes. Similarly, when a middlebox is deployed on the vertex of a flow's source, the traffic rate of the flow diminishes in its earliest position and the bandwidth consumption of the flow is the minimum. However, more middleboxes are needed if we deploy one on each flow's source.

We illustrate the complexity of deployment by showing an example in Fig. 1, where circles and squares represent switches and middleboxes, respectively. Middleboxes are assigned to servers (not shown in the figures) that are attached to switches. All flows need to be served by a middlebox m before reaching their destinations. The traffic-diminishing ratio, which is the ratio of a flow's traffic rate before and after being processed by the middlebox, is 0.5. There are four flows: f_1, f_2, f_3, and f_4, and their initial traffic rates are 4, 2, 2, and 2, respectively. All flow paths are pre-determined, shown in different types of lines. Fig. 1(a) shows the optimal deployment with only two middleboxes allowed. A middlebox is deployed on v_5 to process f_1, and another one on v_2 processes f_2, f_3, and f_4. Take f_1 as an example of a flow's bandwidth consumption calculation. Its initial traffic rate is 4 and its consumed bandwidths on both links (v_5 to v_3 and v_3 to v_1) are $0.5 \cdot 4 = 2$ because of the deployed traffic-diminishing middlebox at its source v_5. Then the bandwidth consumption of f_1 is $2+2=4$. The total bandwidth consumption of all flows is calculated as $0.5 \cdot 4 \cdot 2 + 2 \cdot 2 + 2 + 2 = 12$. If we are allowed to deploy three middleboxes, shown in Fig. 1(b), we should deploy one on each flow's source, which reduces the bandwidth consumption in the earliest locations for all flows. Additionally, the total flow bandwidth consumption is reduced to $0.5 \cdot (4 \cdot 2 + 2 \cdot 2 + 2 + 2) = 8$, which is the minimum.

Most existing works focus on multiple middlebox deployment, but frequently formulate as a complex Integer Programming problem with no efficiency-guaranteed solvers, or are limited to design no performance-guaranteed heuristic solutions. Additionally, we find that when it comes to security inspection or analytic services, only one kind of middleboxes is needed for each flow, such as spam filters, Intrusion Detection Systems (IDSs), Intrusion Prevention Systems (IPSs), Deep Packet Inspection (DPI), and network analytics/ billing services [28]. Thus, we narrow down to the deployment problem of one single type of middleboxes per flow, and propose performance-guaranteed solutions for middleboxes with traffic-diminishing effects in order to minimize the total flow bandwidth consumption. Our main contributions are summarized as follows:

- We formulate a new optimization problem, called Traffic-diminishing Middlebox Deployment (TDMD), where the objective is to minimize the total bandwidth consumption in a given network using a fixed number of middleboxes. The solution of TDMD is particularly useful in allocating spam filters to minimize the total spam traffic using a fixed number of spam filters.
- We prove the NP-hardness of the middlebox deployment in general topologies in Section 4 (Theorem 1). A heuristic algorithm with a complexity of $O(|V|^2 \log |V|)$ of oracle queries is proposed, which has a performance-guaranteed ratio of $(1 - 1/e)$ based on k middleboxes derived from the algorithm.
- We propose one optimal dynamic programming based algorithm and one efficient greedy algorithm for the tree-structured networks. Their time complexities are $O(|V| \cdot (\log |V|)^3 \cdot r_{\max})$ and $O(|V|^2 \log |V|)$ respectively, where V is the vertex set, and $r_{\max}$ is the integral largest flow rate. When flows have the same rate, the time complexity is reduced to $O(|V|^3 (\log |V|)^2)$. We also present a time-efficient greedy solution with complexity of $O(|V|^3 \log |V|)$.
- We conduct extensive simulations to evaluate our algorithms' efficiency with the CAIDA data set [5].

2 RELATED WORK

NFV frameworks have recently drawn a lot of attention, especially in the middlebox deployment[9, 10, 24]. For placing a single type of middleboxes for all flows, Casado *et al.* [6] propose a deployment model and present a heuristic solution. Sang *et al.* [28] study the joint deployment and allocation of a single type of middleboxes, where flows can be split and served by several middleboxes. They propose several performance-guaranteed algorithms to minimize the number of middleboxes. Sallam *et al.* [27] maximize the total amount of network flows that are fully processed by certain nodes while respecting deployment budget and node capacity. However, none of them considers middlebox traffic-changing effects or focuses on the bandwidth consumption objective.

For placing multiple types of middleboxes, most research on middlebox deployment focuses on placing a totally-ordered set, which is known as a *service chain* [32]. Mehraghdam *et al.* [23] propose a context-free language to formalize the chaining of middleboxes and describe the middlebox resource allocation problem as a mixed integer quadratically constrained program. Rami *et al.* [8] locate middleboxes in a way that minimizes both new middlebox setup costs and the distance cost between middleboxes and flows' paths. Both [19] and [20] aim to maximize the total number of requests for service chains. Kuo *et al.* [19] propose a systematic way to tune the proper link consumption and the middlebox setup costs in a joint problem of middlebox deployment and path selection. Li *et al.* [20] present the design and implementation of NFV-RT, a system that dynamically provisions resources in an NFV environment to provide timing guarantees so that the assigned flows meet their deadlines. Fei *et al.* [11] propose a proactive approach to provision new middleboxes in order to minimize the cost incurred by inaccurate prediction of middlebox deployment. However, none of the above mentioned works on service chain considers the traffic-changing effect.

Ma *et al.* [22] are the first to take the traffic-changing effects into consideration. They target load balancing instead of middlebox setting-up costs. They propose a dynamic programming based algorithm to deploy a totally-ordered set, an optimal greedy solution for the middlebox deployment of a non-ordered set, and prove the NP-hardness of placing a partially-ordered set. However, this work only processes a single flow and always builds new, private middleboxes without sharing with other flows, which excessively increases the number of located middleboxes. Chen *et al.* [7] consider both traffic-changing effects and multiple flows. However, their results only can apply to a special restrictive network called the double-tree structure network when flows have the same rate.

3 MODEL AND PROBLEM FORMULATION

3.1 Network model

Our scenario is based on a directed network, $G = (V, E)$, where V is a set of vertices (i.e., switches), and $E \subseteq V^2$ is a set of directed edges (i.e., links). We use v and e to denote a vertex (node) and an edge (link). We assume each link is bidirectional and has enough bandwidth to hold all bypass flows with their initial traffic rates, which eliminates congestion and ensures that the routing of all flows is successful. Middlebox m has a pre-defined traffic-changing ratio $\lambda \geq 0$, serving as the ratio of a flow's traffic rate before and after being processed by m if the flow requires to be processed by m. We focus on the deployment of a single type of traffic-diminishing middleboxes, whose traffic-changing ratio is $\lambda \leq 1$. We use an indicator function, m_v, to represent whether there is a middlebox of m deployed on v. If a middlebox is deployed on v, $m_v = 1$; otherwise, $m_v = 0$.

We are given a set of unsplittable flows $F = \{f\}$. Because flow splitting may not be feasible for applications that are sensitive to TCP packet ordering (e.g. video applications). In any case, split flows can be treated as multiple unsplittable flows. We use f to denote a single flow that has an initial traffic rate of r_f. Its path p_f is an ordered set of edges from f's source, src_f, to its destination, dst_f. All flows' paths are predetermined and valid. We introduce the indicator function, f_v, for flow f using the middlebox m deployed on the vertex v. We define $l_v(f)$ as the minimum number of edges from a vertex v to src_f. We use $b(f)$ to denote f's total bandwidth

Symbols	Definitions
V, E, F	the set of vertices, edges, and flows
v, e, f, m	a vertex,an edge, a flow, and a middlebox
src_f, dst_f, p_f, r_f	source, destination, path, initial rate of f
λ	traffic-changing ratio of middlebox m
m_v	indicator function of placing m on v
f_v	indicator function of f using m on v
$l_v(f)$	minimum number of edges from v to src_f
$\mathcal{P}, \mathcal{F}$	deployment and allocation plans
$b(f), b(\mathcal{P}, \mathcal{F})$	consumed bandwidth of f and the solution
k	the maximum number of middleboxes

Table 1: Symbols and definitions.

consumption on all edges along its path. If f requires to get processed by m, its traffic rate (occupied bandwidth) on e equals to r_f before a flow f is processed by m and $\lambda \cdot r_f$ after the processing; otherwise, its traffic rate remains unchanged as r_f. We assume that each packet in a flow is served by the middlebox only once, even if there are several middleboxes along its path. This is because being served by middleboxes will add an extra transmission delay, which should be avoided as much as possible, in order to improve the network performance.

The problem consists of two sub-problems: the middlebox deployment, which vertices to deploy middleboxes; and the flow allocation, which middlebox to process each flow. Note that the flow allocation is trivial once the middlebox deployment is determined because assigning each flow f with the first deployed middlebox along its path always minimizes its total bandwidth consumption. We use $\mathcal{P}$ and $\mathcal{F}$ to denote the deployment and allocation plans. $b(\mathcal{P}, \mathcal{F})$ is the total bandwidth consumption of all flows being processed by the plans. We have $\mathcal{P} = \{v \mid m_v = 1, \forall v \in V\}$ where $\mathcal{P}$ is a subset of V, i.e., $\mathcal{P} \subseteq V$, which contains all vertices with deployed middleboxes. The allocation plan consists of indicator values for all flows, meaning $\mathcal{F} = \{f_v | \forall f \in F\}$. The decision variables include f_v and m_v for all flows and all vertices. We are given a priori the maximum number of middleboxes that are allowed to be deployed in the whole network, denoted by k. For ease of reference, we summarize notations in Tab. 1.

3.2 Problem formulation

Based on the above model, our Traffic-Diminishing Middlebox Deployment (TDMD) problem includes two properties: *feasibility* and *optimality*. The feasibility of our TDMD problem is whether we are able to use k middleboxes to ensure all flows being processed. For the optimality of middlebox deployment and allocation, we formulate it as a mathematical optimization problem on minimizing the total bandwidth consumption in the following:

$$\min_{\{m_v, f_v | v \in V\}} b(\mathcal{P}, \mathcal{F}) = \sum_{f \in F} b(f) = \sum_{f_v = 1, f \in F} r_f(|p_f| - (1-\lambda) l_v(f)) \quad (1)$$

$$\text{s.t. } \mathcal{P} = \{v \mid m_v = 1, \ \forall v \in V\} \quad (2)$$

$$|\mathcal{P}| = \sum\nolimits_{v \in V} m_v \leq k \qquad \forall m \in M \quad (3)$$

$$\sum\nolimits_{v \in p_f} f_v = 1 \qquad \forall f \in F \quad (4)$$

$$f_v \leq m_v, \ \mathcal{F} = \{f_v | \forall f \in F\} \qquad \forall v \in V \quad (5)$$

$$m_v = \{0, 1\}, \ f_v = \{0, 1\} \qquad \forall v \in V \quad (6)$$

Eq. (1) is our objective: minimizing the total bandwidth consumption, which is the sum of all flows' bandwidth consumption.

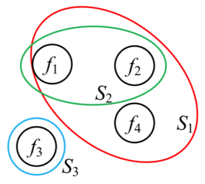


Figure 2: Reduction from feasibility check to set-cover.

A flow's bandwidth consumption is the sum of its occupied bandwidths on each link along its path. We mathematically define $\mathcal{P}$ in Eq. (2) as a set of vertices with middleboxes deployed on them. Eq. (3) states that the total number of deployed middleboxes is no more than k. ($|\cdot|$ denotes the set cardinality.) $|\mathcal{P}|$ is the total number of selected vertices with deployed middleboxes, which equals the sum of m_v, $\forall v \in V$. This is because $m_v = 1$ when a middlebox is deployed on v; otherwise, $m_v = 0$. Eq. (4) requires that each flow $f \in F$ be served by the middlebox once and only once. Eq. (5) ensures that a flow only can be processed on v when one m has been deployed on v. Eq. (6) shows m_v and f_v can only be either 0 or 1.

4 SOLUTION FOR GENERAL NETWORKS

4.1 Problem hardness in a general topolgoy

Theorem 1. *The feasibility of the TDMD problem is NP-hard to check in a general topology.*

Proof: The TDMD problem's feasibility is whether a deployment with k middleboxes ensuring all flows being processed exists or not. First, the feasibility of a given deployment can be verified in a polynomial time as it takes $O(|F|)$ time to check that all flows are processed when reaching their destinations.

Second, we show that the set-cover decision problem is reducible to the feasibility of our TDMD problem. Consider a case of set-cover decision: given a set of elements $\{1, 2, ..., n\}$ (called the universe) and a collection S of $|S|$ sets, whose union equals the universe, we need to identify whether there is a sub-collection of S with k sets, whose union equals the universe. We can always construct an equivalent case of the TDMD problem: we construct a flow f corresponds to each element, then the universe equals their universal set $F = \{f\}$. Each flow requires to be processed by a same type of traffic-changing middlebox m. For each set and its elements in collection S, we construct a set of corresponding flows and assume it is the set of flows that can be processed by deploying a middlebox m on a vertex v. Intuitively, the union of all such sets equals to F. The topology of the TDMD instance consists of all these vertices and is designed as fully-connected between every pair of vertices. The path p_f of each flow f is a directed line connecting each vertex that can process f. We claim that there is a sub-collection of S with k sets, whose union equals the universe, if and only if there exists k vertices to deploy middleboxes, which can process all flows. For suppose there is a sub-collection of S with k sets whose union equals the universe, then we can deploy middleboxes on k vertices with their corresponding sets of processed flows. We illustrate the reduction using an example in Fig. 2. The elements are all the flows $F = \{f\}$, i.e., $\{f_1, f_2, f_3, f_4\}$. In the example, $S_1 = \{f_1, f_2, f_4\}$, $S_2 = \{f_1, f_2\}$ and $S_3 = \{f_3\}$. We need to find the minimum number of subsets whose union equals the universe set, which is S_1 and S_3. Then we deploy one middlebox on v_1 and another on v_3 to serve all flows.

Algorithm 1 General Topology Placement (GTP)

```
In: $V, E, F$ and traffic-changing ratio $\lambda$;
Out: Deployment and allocation plans $\mathcal{P}$ and $\mathcal{F}$;
1: Initialize $\mathcal{P}$ as an empty set $\emptyset$;
2: while not all flows are processed do
3:     deploy one $m$ on the vertex $v \in V \backslash \mathcal{P}$ with $\max d_{\mathcal{P}}(v)$;
4:     $\mathcal{P} = \mathcal{P} \bigcup \{v\}$ and $m_v = 1$;
5: $\forall f \in F, f_v = 1$, if $l_v(f) = \max\{l_{v'}(f) | m_{v'} = 1, \forall v' \in p_f\}$.
6: return $\mathcal{P}$ and $\mathcal{F} = \{f_v | \forall f \in F\}$ .
```

Conversely, if we can deploy k vertices to process all flows, then we can select their corresponding sets of elements from S. As all flows are processed, the union of the sub-collection sets of S equals the universe. Consequently, since the set-cover decision problem is a NP-complete problem, the TDMD problem is NP-hard. ■

4.2 Performance-guaranteed solution

Since the feasibility is NP-hard to check in a general topology, we propose a greedy but feasible solution that uses k middleboxes derived from the solution. We prove that it is performance-guaranteed for the k. The greedy algorithm, called General Topology Placement (GTP), is shown in Alg. 1. Line 1 initiates the deployment plan as an empty set. Lines 2-4 iteratively select $v \in V$ with the maximum value of $\max d_{\mathcal{P}(v)}$ until all flows are fully served. In each round, we add the new middlebox to the current plan $\mathcal{P}$. Line 5 generates the allocation plan The deployment plan $\mathcal{P}$ returns in line 6.

We also use Fig. 1 to illustrate the process of applying GTP. The settings are the same. The initial traffic rates of flows are $r_1 = 4$, $r_2 = 2$, $r_3 = 2$, and $r_4 = 2$, respectively. We list the values of marginal decrement for all vertices in Tab. 2. In the first round, $d_{\emptyset}(v_5)$ has the maximum value so that we deploy a middlebox on v_5 and $\mathcal{P} = \{v_5\}$. In the next round, $d_{\{v_5\}}(v_6)$ has the maximum value, but the deployment plan is not feasible. We can only deploy a middlebox on v_2 because of $k = 2$ and $\mathcal{P} = \{v_2, v_5\}$. If we are given $k = 3$, the result is shown in Fig. 1(b). In the first round, $d_{\emptyset}(v_5)$ has the maximum value so that we deploy a middlebox on v_5 and $\mathcal{P} = \{v_5\}$. In the next round, $d_{\{v_5\}}(v_6)$ has the maximum value so $\mathcal{P} = \{v_5, v_6\}$ and . Then, $d_{\{v_5,v_6\}}(v_4)$ has the maximum value so the final plan is $\{v_4, v_5, v_6\}$.

Definition 1 (decrement function). *The decrement function, denoted as $d(\mathcal{P})$, indicates the decrement of the total bandwidth consumption by a deployment plan $\mathcal{P}$, which satisfies $d(\mathcal{P}) = \sum_{f \in F} r_f \cdot |p_f| - b(\mathcal{P})$.*

Definition 2 (marginal decrement). *The marginal decrement, denoted as $d_{\mathcal{P}}(\mathcal{S}) = d(\mathcal{P} \cup \mathcal{S}) - d(\mathcal{P})$, indicates the additional bandwidth decrement of processing flows by deploying middleboxes on a new subset $\mathcal{S} \in V$ beyond vertices in the current deployment $\mathcal{P}$.*

Lemma 1. *(1) $d(\emptyset) = 0$ and $d(V) = (1 - \lambda) \cdot \sum_{f \in F} r_f \cdot |p_f|$; (2) $\max d(\mathcal{P}) = (1 - \lambda) \cdot \sum_{f \in F} r_f \cdot |p_f|$; (3) $\min d(\mathcal{P}) = 0$.*

Proof: (1) A flow f's bandwidth consumption is the sum of its occupied bandwidth on each link along its path, which is $r_f \cdot |p_f|$. When we deploy no middlebox, i.e., $\mathcal{P} = \emptyset$, the traffic rates of all flows remain unchanged. Then we have $d(\emptyset) = 0$. Similarly, when there is a middlebox deployed on each vertex, i.e., $\mathcal{P} = V$, the traffic

v	v_1	v_2	v_3	v_4	v_5	v_6
$d_{\emptyset}(v)$	0	0	3	1	4	3
$d_{\{v_5\}}(v)$	0	0	1	1	—	3
$d_{\{v_5,v_6\}}(v)$	0	0	0	1	—	—

Table 2: Marginal decrement values.

rate of each flow f changes from r_f to $\lambda \cdot r_f$ as early as its source. The bandwidth consumption of a flow f is decreased to $\lambda \cdot r_f \cdot |p_f|$. The total bandwidth consumption becomes $b(V) = \sum_{f \in F} \lambda \cdot r_f \cdot |p_f| = \lambda \cdot \sum_{f \in F} r_f \cdot |p_f|$. We have $d(V) = b(V) - (1 - \lambda) \cdot \sum_{f \in F} r_f \cdot |p_f|$. (2) The total bandwidth consumption is the smallest when all flows are processed as early as their sources because all flows' traffic rates are diminished from the first edges along their paths. We have $\min b(\mathcal{P}) = \lambda \cdot \sum_{f \in F} r_f \cdot |p_f|$ and $\max d(\mathcal{P}) = (1 - \lambda) \cdot \sum_{f \in F} r_f \cdot |p_f|$. (3) Similarly, the total bandwidth consumption is the largest when all flows are not processed because their traffic rates are not diminished. Thus, we have $\max b(\mathcal{P}) = \sum_{f \in F} r_f \cdot |p_f|$ and $\min d(\mathcal{P}) = 0$. ■

Next, we analyze the properties of the decrement function $d(\mathcal{P})$. It is known that a function d is *submodular* if and only if $\forall \mathcal{P} \subseteq \mathcal{P}' \subseteq V, \forall v \in V \backslash \mathcal{P}', d_{\mathcal{P}}(\{v\}) \geq d_{\mathcal{P}'}(\{v\})$, i.e., $d(\mathcal{P} \cup \{v\}) - d(\mathcal{P}) \geq d(\mathcal{P}' \cup \{v\}) - d(\mathcal{P}')$.

Theorem 2. *$d(\mathcal{P})$ is a submodular function.*

Proof: We assume $\mathcal{P}' = \mathcal{P} \bigcup S$. From definitions 2 and 3, we have

$$d(\mathcal{P}') - d(\mathcal{P}) = (\sum_{f \in F} r_f \cdot |p_f| - b(\mathcal{P}')) - (\sum_{f \in F} r_f \cdot |p_f| - b(\mathcal{P}))$$

$$= b(\mathcal{P}) - b(\mathcal{P}') = \sum_{f_{v'}=1, f \in F} r_f (1 - \lambda) l_{v'}(f) - \sum_{f_v=1, f \in F} r_f (1 - \lambda) l_v(f)$$

$$= \sum_{f_{v'}=1, f_v=1}^{f \in F} r_f (1 - \lambda)[l_{v'}(f) - l_v(f)].$$

We know $f_v = 1$ if v has $l_v(f) = \max\{l_w(f) | m_w = 1, \forall w \in p_f\}$. As more middleboxes are deployed in $\mathcal{P}'$ beyond $\mathcal{P}$, flows must be processed no later than in $\mathcal{P}$. Then $\forall f \in F$, we have $l_{v'}(f) \geq l_v(f)$ when $f_{v'} = 1$ in $\mathcal{P}'$ and $f_v = 1$ in $\mathcal{P}$. Thus, $d(\mathcal{P}') \geq d(\mathcal{P})$ and $d(\mathcal{P})$ is a non-decreasing function, which is monotone. Suppose a vertex u satisfies $u \in V \backslash \mathcal{P}'$. If we deploy a middlebox on u, f_u becomes 1 if u has $l_u(f) = \max\{l_w(f) | m_w, \forall w \in p_f\}$, resulting in a smaller $b(f)$; otherwise, f_u still is 0 and $b(f)$ remains the same. Then we have $l_u(f) > l_{v'}(f) \geq l_v(f), \forall f \in F$.

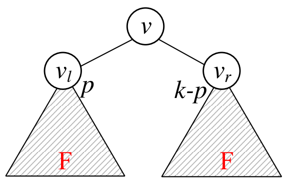
From above, we get $d(\mathcal{P} \bigcup \{u\}) - d(\mathcal{P}) = \sum_{f_u=1}^{f \in F} r_f (1 - \lambda)(l_u(f) - l_v(f))$. As $\forall f \in F$, $(l_u(f) - l_{v'}(f)) \leq (l_u(f) - l_v(f))$, we have:

$$\sum_{f_u=1}^{f \in F} r_f (1 - \lambda)(l_u(f) - l_{v'}(f)) \leq \sum_{f_u=1}^{f \in F} r_f (1 - \lambda)(l_u(f) - l_v(f)).$$

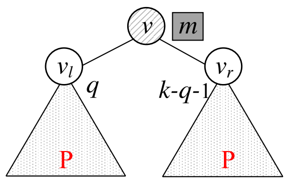
Then we have $d(\mathcal{P}' \bigcup \{u\}) - d(\mathcal{P}') \leq d(\mathcal{P} \bigcup \{u\}) - d(\mathcal{P})$, meaning $d_{\mathcal{P}'}\{v\} \leq d_{\mathcal{P}}\{v\}$. Thus, $d(\mathcal{P})$ is submodular. ■

Theorem 3. *The proposed GTP can achieve a deployment with at most $(1 - 1/e)$ times of the maximum decrement. Its time complexity is $O(|V|^2 \log |V|)$ of oracle queries.*

Proof: Our TDMD problem has the same formulation as the set cover problem and the deployment $\mathcal{P}$ follows its greedy algorithm in [16]. Hence, the approximation ratio $(1 - 1/e)$ follows from Proposition 6 in [16]. Also, Feige [12] proved that unless P = NP, no polynomial time algorithm can achieve an approximation ratio better than $(1 - 1/e)$ for the cardinality constrained maximization of this kind of



(a) Subtree fully served.



(b) Served on current node v.

Figure 3: Illustration for fully served situation.

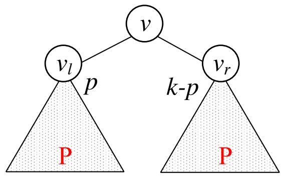


Figure 4: Illustration for partial served situation.

set cover problem. We run k rounds for placing each middlebox and $k = O(|V|)$ as we at most place one middlebox on each vertex. In each round, it takes a constant time to calculate $d_{\mathcal{P}}(v)$, $\forall v \in V \setminus \mathcal{P}$. Sorting all such $d_{\mathcal{P}}(v)$ takes at most $O(|V| \log |V|)$. The function calculation itself is usually assumed to be conducted by an oracle, and thus its complexity is not taken into account. So the algorithm complexity is $O(|V| \cdot |V| \log |V|) = O(|V|^2 \log |V|)$ times oracles. ■

5 SOLUTIONS FOR TREE NETWORKS

Because of the NP-hardness of our TDMD problem in general topologies, here we narrow down to tree topologies, which are common in streaming services [25], content delivery networks (CDNs) [30], and tree-based tiered topologies like Fat-tree [3] or BCube [14] in data centers. We require that the sources of all flows are leaves and their destinations are the same as the root of the tree. As long as the middlebox number constraint $k \geq 1$ and the destinations of all flows are the root, we can process all flows by placing a middlebox on the root so that there does not exist infeasibility.

5.1 Optimal DP-based solution

We introduce the optimal dynamic programming (DP) based solution for the tree-structured networks. Note that the solution works for general trees with an arbitrary number of branches. For simplicity, we only discuss the solution for the binary tree. Before introducing our optimal DP solution, we define two notations for the fully and partially served situations, respectively. Let $F(v, k)$ denote the minimum total occupied bandwidth of all flows with k deployed middleboxes in the subtree T_v rooted at v when all flows have been fully processed. Let $P(v, k, b)$ denote the minimum total occupied bandwidth of all flows with k deployed middleboxes in the tree T_v when flows with a total bandwidth consumption of b have been processed. Then we have the following formulations. The formulation for the full served case is:

$$F(v,k) = \min\{\min_{0<p<k}\{F(v_l,p) + F(v_r,k-p)\}+ \lambda \sum\nolimits_{f\in T_v} b(f), \min_{0\leq q<k}\{P(v_l,q,b_l) + P(v_r,k-1-q,b_r) + \lambda b_l + \lambda b_r + \sum\nolimits_{f\in T_v}(b(f) - b_l - b_r)\}\}. \tag{7}$$

Here is the explanation. When all flows have been fully processed, there are only two served situations in Fig. 3:

(a) In Fig. 3(a), the left and right subtrees of v have already been served by totally deploying k middleboxes before v. The minimum total occupied bandwidth in the tree of v, is selected from all combinations of allocating the total k middleboxes. If the left subtree deploys p ($0 < p < k$) middleboxes, then the right tree deploys the remaining $k-p$ ones. The sum of the minimum total occupied bandwidth inside v's two subtrees is $\min_{0<p<k}\{F(v_l, p) + F(v_r, k-p)\}$. The total consumed bandwidth on the two uplinks from its two subtrees to v, is $\lambda \sum_{f\in T_v} b(f)$. This is because all flows have already been processed inside its two subtrees. The minimum total occupied bandwidth in the tree of v, $F(v, k)$, is the sum of the two parts.

(b) In Fig. 3(b), its two subtrees are partially served. So we deploy one middlebox on v to make sure all flows through v have been served. The minimum total occupied bandwidth in the tree of v is selected from all combinations of allocating the remaining $k-1$ middleboxes in its subtrees. If the left subtree deploys q ($0 \leq q \leq k-1$) middleboxes, then the right tree deploys the remaining $k-q-1$ ones. Here each subtree can deploy no middlebox as we do not require all flows to be served. The sum of the minimum total occupied bandwidth inside v's two subtrees is $\min_{0\leq q\leq k}\{P(v_l, q, b_l) + P(v_r, k-1-q, b_r)\}$. The total consumed bandwidth on the two uplinks from its two subtrees to v, is $\lambda b_l + \lambda b_r + \sum_{f\in T_v}(b(f) - b_l - b_r)$. This is because its left subtree has flows with a total bandwidth of b_l processed while the right has flows with a total bandwidth of b_r processed. Additionally, there are flows with a total bandwidth of $\sum_{f\in T_v}(b(f) - b_l - b_r)$ that are not processed inside its both subtrees and will be processed by the middlebox deployed on v.

The formulation for the partial served case is:

$$P(v,k,b) = \min_{0\leq p\leq k}\{P(v_l,p,b_l) + P(v_r,k-p,b_r) + \lambda b_l + \lambda b_r + \sum\nolimits_{f\in T_v}(b(f) - b_l - b_r)\}. \tag{8}$$

We have $b = b_l + b_r$. Here is the explanation. When flows are partially processed with a total served bandwidth b, there is only one served situation, illustrated in Fig. 4: The left and right subtrees of v are both partially served. The minimum total occupied bandwidth in the tree of T_v is selected from all combinations of allocating the total k middleboxes. If the left subtree deploys p ($0 \leq p \leq k$) middleboxes, then the right tree deploys the remaining $k-p$ ones. Here each subtree can deploy no middlebox as we do not require all flows to be served. The sum of the minimum total occupied bandwidth inside v's two subtrees is $\min_{0\leq p\leq k}\{P(v_l, p, b_l) + P(v_r, k-p, b_r)\}$. The total consumed bandwidth on the two uplinks from its two subtrees to v, is $\lambda b_l + \lambda b_r + \sum_{f\in T_v}(b(f) - b_l - b_r)$. The reason is the same as the second served situation of $F(v, k)$. Then the total served flow bandwidth is the sum of the served ones in each subtree, which is $b = b_l + b_r$. Here we need to mention that $F(v, k)$ is a special case of $P(v, k, b)$ when all flows in T_v are processed and b is the smallest.

The initial value of $F(v, k)$ for each leaf node v is:

$$F(v,k) = \begin{cases} 0 & k \geq 1, \\ \infty & otherwise. \end{cases} \tag{9}$$

There is no bandwidth consumption inside a leaf node if any middlebox is deployed. If no middlebox is deployed ($k \leq 0$), there is no feasible deployment then the value of $F(v, k)$ is set as ∞. The

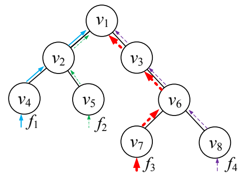


Figure 5: An example.

$k\backslash v$	1	2	3	4	5	6	7	8
1	24	3	6	0	0	6	0	0
2	16.5	1.5	3	0	0	3	0	0
3	13.5	1.5	3	0	0	3	0	0
4	12	1.5	3	0	0	3	0	0

Figure 6: Values for $F(v, k)$.

$k\backslash b$	0	1	2	3	4	5	6	7	8	9
0	24	∞	∞	∞	∞	∞	∞	∞	∞	∞
1	∞	22.5	22	22.5	∞	16.5	∞	∞	∞	24
2	∞	∞	21.5	20.5	21	16.5	15	14.5	15	16.5
3	∞	∞	∞	∞	19.5	∞	∞	14	13	13.5
4	∞	∞	∞	∞	∞	∞	∞	∞	∞	12

(a) $P(v_1, k, b)$

$k\backslash b$	0	1	5	6
0	6	∞	∞	∞
1	6	5.5	3.5	∞
2	∞	5.5	3.5	3

(b) $P(v_2, k, b)$

$k\backslash b$	0	1	5	6
0	12	∞	∞	∞
1	12	11	7	∞
2	∞	11	7	6

(c) $P(v_3, k, b)$

$k\backslash b$	0	2
0	0	∞
1	∞	0

(d) $P(v_4, k, b)$

$k\backslash b$	0	1
0	0	∞
1	∞	0

(e) $P(v_5, k, b)$

$k\backslash b$	0	1	5	6
0	6	∞	∞	∞
1	6	5.5	3.5	∞
2	∞	5.5	3.5	3

(f) $P(v_6, k, b)$

$k\backslash b$	0	5
0	0	∞
1	∞	0

(g) $P(v_7, k, b)$

$k\backslash b$	0	1
0	0	∞
1	∞	0

(h) $P(v_8, k, b)$

Figure 7: Values for $P(v, k, b)$.

initial value of $P(v, k, b)$ for each leaf v is:

$$P(v, k, b) = \begin{cases} 0 & k \geq 0 \text{ and } b \leq \sum_{f \in T_v} b_f, \\ \infty & \textit{otherwise}. \end{cases} \tag{10}$$

The upper-bound of the total processed bandwidth is $\sum_{f \in T_v} b_f$, which is the sum of all flows' initial traffic rates. Even when no middlebox is deployed ($k = 0$), $P(v, k, b)$ is 0.

In order to reduce bandwidth consumption, flows are always processed by the middlebox that is nearest to their sources, i.e., farthest to their destinations. It means that $f_v = 1$ if v has the value $l_v(f) = \max\{l_w(f) | m_w = 1, \forall w \in p_f\}$. For a flow f, if a vertex $v \in p_f$ has $f_v = 1$, we have $(|p_f| - l_v(f))$ edges consuming r_f bandwidth and $l_v(f)$ edges consuming $\lambda \cdot r_f$ bandwidth. Then $b(f) = (|p_f| - l_v(f)) \cdot r_f + \lambda \cdot r_f \cdot l_v(f) = r_f|p_f| - r_f(1 - \lambda)l_v(f)$. Thus, when $\mathcal{P}$ is decided, the optimal allocation plan is also decided. For simplicity, we omit $\mathcal{F}$ in $b(\mathcal{P}, \mathcal{F})$ as $b(\mathcal{P})$.

Theorem 4. *The dynamic programming based solution (DP) is optimal for our TDMD problem in tree-structured topologies.*

Proof: The detailed proof is omitted due to the optimal property of the dynamic programming method. ■

We use an example in Fig. 5 to explain the details of applying our DP formulation. There is a binary tree with eight vertices and four flows f_1, f_2, f_3 and f_4 with their initial rates as $r_1 = 2, r_2 = 1, r_3 = 5$ and $r_4 = 1$, respectively. The traffic changing ratio of the middlebox is $\lambda = 0.5$. We list the values of $F(v, k)$ and $P(v, k, b)$ for all combinations of feasible k and b for each vertex in the tables of Fig. 6 and Fig. 7. When all flows have been processed in the root v_1, $b = r_1+r_2+r_3+r_4 = 2+1+5+1 = 9$, and we have $F(v_1, k) = P(v_1, k, 9)$ for all k. Take $k = 3$ as an example. We have $F(v_1, 3) = P(v_1, 3, 9) = 13.5$ from tables in Fig.6 and Fig. 7(a). For finding the corresponding, we trace back. If no middlebox is deployed on v_1, all flows have been processed inside both subtrees of v_1. Then we can calculate the total consumed bandwidth inside both subtrees as $13.5-0.5\cdot(2+1+5+1) = 9$, which is exactly the sum of $F(v_2, 1) = 3$ and $F(v_3, 2) = 6$. Our assumption is correct. Then $F(v_2, 1)$ is selected, which means that we deploy only one middlebox to fully serve all flows in the tree T_2. The only feasible position is v_2. For the right subtree, $F(v_3, 2)$ is selected. Since there are only two leaf nodes in the tree T_3, we deploy one middlebox on each leaf node. Then the optimal deployment for $k = 3$ is $\{v_2, v_7, v_8\}$. Similarly, the optimal deployment for $k = 2$ is $\{v_1, v_7\}$ or $\{v_2, v_6\}$. As for partially processing with $k = 3$, if only 1 rate of flows is not processed, we have $b = 9 - 1 = 8$. For minimizing the total bandwidth consumption, we should deploy a middlebox on v_4 instead of v_2, resulting in $P(v_1, 3, 8) = 13 < P(v_1, 3, 9)$. Due to space limits, we omit calculation of other values in Figs. 6 and 7.

Theorem 5. *Time complexity of Alg. DP is $O(|V| \cdot (\log |V|)^3 \cdot r_{\max})$, where $r_{\max} = \max_{f \in F} r_f$. (We assume $r_{\max}$ is an integer.)*

Proof: This is because there are $|V|$ vertices and we need to traverse each vertex. For each vertex v, we need to calculate values of $F(v, k)$ and $P(v, k, b)$ for all combinations of feasible k and b. The largest value of k is the number of leaves, which is $O(\log |V|)$ for a binary tree. Additionally, the largest value of b is the sum of all flows' initial traffic rate $\sum_{f \in F} b_f$, which is less than $|F| \cdot r_{\max}$. Additionally, for flows from the same leaf source, we can treat them as a single flow because of their same path to the root. As a result, the maximum number of flows is the same number of the leaf nodes, i.e., $O(|F|) = O(\log |V|)$. $r_{\max}$ is the largest flow rate after the merge. When calculating the value of $F(v, k)$ for a group of fixed values v and k, we need to select the minimum value from all its combinations for two served situations, whose total number is less than $O(k+k \cdot |F| \cdot r_{\max}) = O((\log |V|)^2 \cdot r_{\max})$. When calculating the value of $P(v, k, b)$ for a group of fixed values v, k and b, we need to select the minimum value from all its combinations, whose total number is less than $k \cdot |F| \cdot r_{\max} = O((\log |V|)^2 \cdot r_{\max})$. The selection takes a constant time. As a result, the worst time complexity is $O(|V| \cdot \log |V| \cdot ((\log |V|)^2 \cdot r_{\max} + (\log |V|)^2 \cdot r_{\max}) = O(|V| \cdot (\log |V|)^3 \cdot r_{\max})$. ■

When all flows have the same initial traffic rate, the time complexity becomes $O(|V|(\log |V|)^2(|V|^2 + (\log |V|)^2) = O(|V|^3(\log |V|)^2)$, which is polynomial. When flows have various initial traffic rates , the DP algorithm is pseudo-polynomial. It is not trivial to transform our proposed DP algorithm into a polynomial-time approximation scheme (PTAS) [31] because the placement at the current vertex does not have a obvious relationship or direct impact with its parent. When the traffic rates of flows are in an arbitrary precision and order of magnitude, the DP algorithm is computationally hard. This motivates us to propose an efficient algorithm with a lower time complexity in the next subsection.

5.2 Efficient greedy solution

We study a fast, sub-optimal greedy solution, but first introduce a definition from graph theory.

Definition 3 (LCA). *Lowest common ancestor (LCA) of two vertices v and w in an acyclic graph G is the lowest vertex that has both v and w as descendants.*

Algorithm 2 Heuristic Algorithm for Trees (HAT)

```
In: Sets of vertices $V$, edges $E$, and flows $F$, traffic-changing ratio
    $\lambda$ and middlebox number constraint $k$;
Out: The deployment plan $\mathcal{P}$;
1: Initialize $\mathcal{P}$ as a set of all leaf vertices;
2: Calculate $\Delta b(i,j),\ \forall v_i, v_j \in P(i \neq j)$;
3: Construct a min-heap of $\Delta b(i,j), \forall v_i, v_j \in P(i \neq j)$.
4: while $|\mathcal{P}| > k$ do
5:     Merge the two middleboxes with the minimum $\Delta b(i,j)$,
       $\forall v_i, v_j \in \mathcal{P}(i \neq j)$.
6:     Update the heap by deleting pairs with $v_i$ or $v_j$ and inserting
       pairs with $LCA(i,j)$.
7:     $\mathcal{P} = (\mathcal{P} \setminus \{v_i, v_j\}) \bigcup \{LCA(i,j)\}$;
8: return The deployment plan $\mathcal{P}$.
```

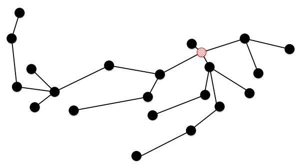
We define each vertex to be a descendant of itself. Thus, if v has a direct connection from w, w is the lowest common ancestor [29]. Take Fig. 5 as an example. LCA of vertices v_4 and v_5 is v_2 and LCA of vertices v_1 and v_6 is v_1.

Next, we define $\Delta b(i,j)$ as the difference in the total bandwidth value when we delete two middleboxes on v_i and v_j and deploy one middlebox on $LCA(i,j)$. The process of the deletion and deployment is called *merge*. We propose our solution as Heuristic Algorithm for Trees (HAT), shown in Alg. 2. Line 1 initiates the deployment plan by placing a middlebox on every leaf vertex. Line 2 calculates the value of $\Delta b(i,j)$ for each pair of vertices. Line 3 constructs the first min-heap. Lines 4-7 iteratively select the pair with the minimum value of $\Delta b(i,j)$ and merge the two middleboxes by placing one on their LCA until the number of middleboxes reaches k. In each round, we do merge to reduce the number of middleboxes by one. The min-heap is updated by deleting pairs with v_i or v_j and inserting new pairs with $LCA(i,j)$. We also delete two vertices v_i and v_j from $\mathcal{P}$ and insert their LCA into $\mathcal{P}$. The deployment $\mathcal{P}$ returns in line 8. Note that HAT is not optimal for some cases, especially when traffic has a heavily unbalanced distribution.

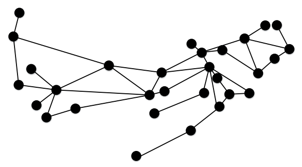
We show steps of running HAT in Fig. 5 with the same setting in the last subsection. Initially, $\mathcal{P} = \{v_4, v_5, v_7, v_8\}$, which has the minimum bandwidth consumption for all possible deployments. This is because the traffic rates of all flows are diminished from their sources and the bandwidth consumption of each flow is the smallest. If $k \geq 4$, since the while loop does not need to run, the deployment plan returned by Alg. HAT is $\mathcal{P} = \{v_4, v_5, v_7, v_8\}$. If $k = 3$, one round of the while loop needs to run. There are $\binom{4}{2} = 6$ pairs. We calculate the value of $\Delta b(i,j)$ for each pair. For example, $\Delta b(4,5) = 1.5$, $\Delta b(7,8) = 3$ and $\Delta b(4,7) = 9.5$. After calculating these six pairs, we find that $\Delta b(4,5)$ has the minimum value, 1.5. We delete v_5 and v_6 from $\mathcal{P}$ and insert their LCA v_2 into $\mathcal{P}$. Then the deployment plan returned by Alg. HAT is $\mathcal{P} = \{v_2, v_7, v_8\}$. If $k = 2$, two rounds of the while loop need to run. The first round is the same as $k = 3$. In the second round, there are $\binom{3}{2} = 3$ pairs. We have $\Delta b(2,7) = 9, \Delta b(2,8) = 3$, and $\Delta b(7,8) = 3$. $\Delta b(2,8) = 3$, and $\Delta b(7,8) = 3$ have the same minimum value. If we select to delete v_7 and v_8 from $\mathcal{P}$ and insert their LCA v_6 into $\mathcal{P}$. Then the deployment plan returned by HAT is $\mathcal{P} = \{v_2, v_6\}$. Otherwise, $\mathcal{P} = \{v_1, v_7\}$. Similarly, $\mathcal{P} = \{v_1\}$ when $k = 1$.

WASHINGTON
MONTANA
NORTH DAKOTA
MINNESOTA
Ottawa
Montreal
OREGON
IDAHO
SOUTH DAKOTA
WISCONSIN
MICHIGAN
WYOMING
IOWA
Chicago
NEVADA
UTAH
United States
NEBRASKA
ILLINOIS
INDIANA
OHIO
Philadelphia
San Francisco
COLORADO
KANSAS
MISSOURI
CALIFORNIA
Las Vegas
Los Angeles
San Diego
ARIZONA
OKLAHOMA
NEW MEXICO
ARKANSAS
TENNESSEE
Dallas
MISSISSIPPI
ALABAMA
GEORGIA
TEXAS
LOUISIANA
Houston
FLORIDA

(a) The Archipelago (Ark) Infrastructure.



(b) Tree topo (subgraph of (a)).



(c) General topo (subgraph of (a)).

Figure 8: Simulation topologies.

Theorem 6. *The time complexity of Alg. HAT is* $O(|V|^2 \log |V|)$.

Proof: There are $O(|V|/2) = O(|V|)$ leaf vertices and $O(|V|^2)$ pairs. The time of building a min-heap costs $O((|V|^2) \cdot \log(|V|^2)) = O(|V|^2 \log |V|)$. For the while loop, we need to run $O(|V|/2 - k) = O(|V|)$ rounds in order to reduce the number of middleboxes from $O(|V|/2)$ to k. In each round, it at most takes $O(|V|)$ time to delete pairs with v_i and v_j and insert pairs with $LCA(i,j)$. Thus, the total time complexity is $O(|V|^2 \log |V| + |V| \cdot |V|) = O(|V|^2 \log |V|)$. ■

6 EVALUATION

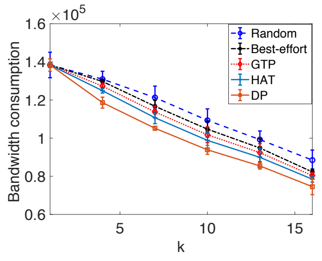
6.1 Setting

Topology: We conduct simulations by MATLAB on the Archipelago (Ark) Infrastructure topology [5] in Fig. 8(a), which is CAIDA's active measurement infrastructure serving the network research community since 2007. The tree and general topologies are reduced from Fig. 8(a). Additionally, traditional data center networks and WAN design over-provision the network with 30−40% average network utilization in order to handle traffic demand changes and failures [17]. Thus, we assume each link has enough bandwidth to hold all flows. This assumption eliminates link congestion and ensures that the transmission of all flows is successful, since routing failure is not our concern.

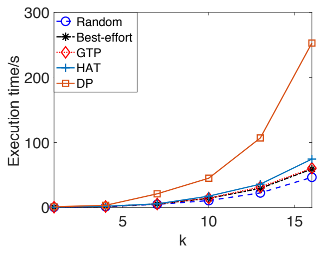
Middlebox: We only have one kind of middlebox for each deployment. The traffic changing ratio has a range from 0 (e.g., spam filters) to 0.9 (e.g., traffic optimizer) with an interval of 0.1. We do additional simulations for the spam filter, which cuts down the traffic after flows being served by them.

Traffic: All flows' paths are fixed and their traffic rates are also known a priori. We use the flow size distribution of the CAIDA center, which was collected in a 1-hour packet trace. Under the tree topology, the destination of all flows is the root of the tree. As for changing the flow density variable, we randomly select flows from the dataset in order to make our experiments more general. Here we mention that our simulations only study feasible deployments. If the algorithm can not find a feasible solution, we choose to regenerate a traffic distribution.

6.2 Metrics and comparison algorithms

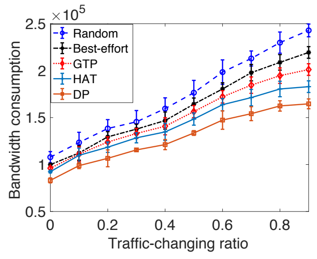


(a) Bandwidth consumption.

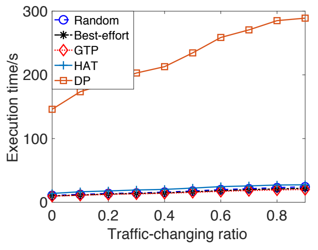


(b) Execution time.

Figure 9: Middlebox number constraint k in tree.

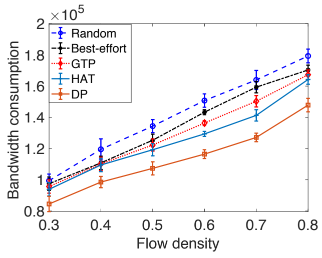


(a) Bandwidth consumption.

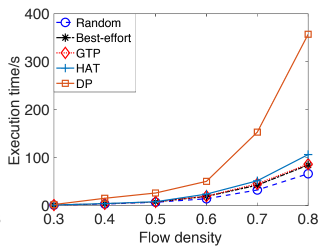


(b) Execution time.

Figure 10: Traffic-changing ratio in tree.

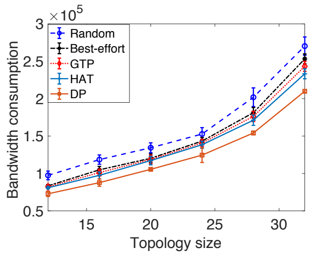


(a) Bandwidth consumption.

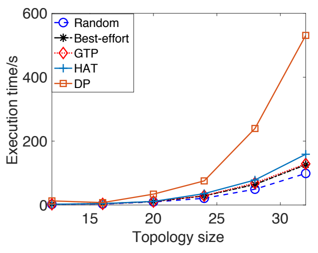


(b) Execution time.

Figure 11: Flow density in tree.



(a) Bandwidth consumption.



(b) Execution time.

Figure 12: Topology size in tree.

We use two performance metrics for our benchmark comparisons: the total bandwidth consumption, which is our objective in Eq. 1, and the execution time of each algorithm in seconds. We test the relationships among these two metrics and five variables: middlebox number constraint k (only for trees), traffic-changing ratio, flow density, topology size and topology type. The flow density is defined as the ratio of the total traffic load to the total capacity of the network. Each simulation tests one variable and keeps other variables constant. The default values of these variables are: (1) The middlebox number for the tree is $k = 8$ and for the general topology is $k = 10$; (2) The traffic-changing ratio is $\lambda = 0.5$; (3) The flow density is 0.5; (4) The topology size is 22 for the tree topology (shown in Fig. 8(b)), and 30 for the general topology (shown in Fig. 8(c)); (5) We have a tree topology and a general topology. Destinations are shown as red nodes. The root of tree topology is colored red, shown in Fig. 8(b). The topology size changes by randomly inserting and deleting vertices in the network. The independent variable in each figure is shown as the caption.

We include two benchmark schemes in our simulations: one is Random, which randomly deploys middleboxes until it deploys k middleboxes; another one is Best-effort, which deploys one middlebox on the vertex, which can reduce the bandwidth of flows mostly, until it deploys k middleboxes. Our proposed Alg. DP and Alg. HAT are for the tree, and Alg. GTP is for both the tree and the general topologies. We only discuss feasible solutions. We run each algorithm multiple times and show the error bar of each point to evaluate fluctuating situations.

6.3 Simulation results in a tree topology

Simulation results in a tree are shown in Figs. 9, 10, 11, and 12. Fig. 9(a) shows the relationship between bandwidth consumption and k ranging from 1 to 16 with an interval of 3. Alg. DP always has the lowest bandwidth consumption as well as the smallest error bars, which verifies its optimality. When $k = 1$, there is only one feasible deployment plan so all bandwidth consumptions are the same. When k becomes larger, all their total bandwidth consumptions become lower since more flows can be processed nearer to their sources. Alg. HAT has the second lowest bandwidth consumption, while Alg. GTP has the third lowest. The error bars of Alg. Random are always the largest because its randomness of deployment results in an unsteady performance. Fig. 9(b) shows the execution time result of the five algorithms, which verifies the time complexity analysis of our proposed algorithms. When the middlebox number constraint k increases, the execution time of Alg. DP increases vastly while other four algorithms only have moderate increment. This is because the relationship between k and V is $k = O(\log |V|)$ as we discussed in Section V. It indicates the trade-off between the performance and the efficiency of this algorithm. Alg. HAT has the second longest execution time because its complexity is $O(|V|^2 \log |V|)$ larger than others, although its bandwidth consumption performance is the second best. Alg. Best-effort has a close execution time with the Alg. GTP. In the following discussion, since some results and analysis are similar, we omit the details because of limited space.

Fig. 10(a) indicates the result of the bandwidth consumption on the traffic-changing effect ranging from 0 to 0.9 with an interval of 0.1. Alg. DP still achieves the lowest bandwidth consumption for all the time. Alg. HAT has the second lowest bandwidth consumption, while Alg. GTP has the third lowest. The difference between every two algorithms becomes larger with the increase of λ. When $\lambda = 0.8$, the bandwidth consumption of Alg. HAT is only 75.4% of Alg. Best-effort and 66.1% of Alg. Random. We find the traffic-changing ratio has little influence on the execution time of all greedy algorithms, shown in Fig. 10(b). This also confirms that the time complexity is almost irrelevant of the traffic-changing ratio.

The bandwidth consumption with the flow density changing from 0.3 to 0.8 with an interval of 0.1 is shown in Fig. 11(a). The basic tendencies of all five lines are linear with the increase of

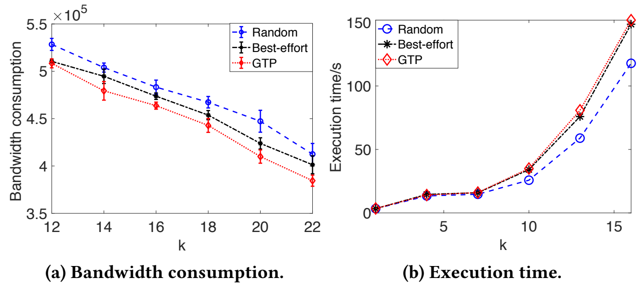

(a) Bandwidth consumption. (b) Execution time.

Figure 13: Middlebox number k in a general topology.

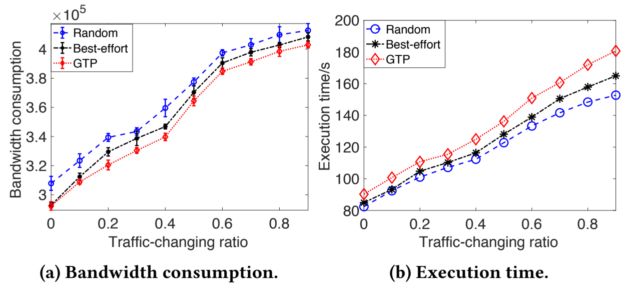

(a) Bandwidth consumption. (b) Execution time.

Figure 14: Traffic-changing ratio in a general topology.

the flow density. When the density increases from 0.5 to 0.7, the advantage of our Alg. HAT is so obvious that its consumption is at most 72.1% of the consumption of Alg. Random. When the density is high, the bandwidth consumption of Alg. Random becomes larger at a faster rate because more flows need to be handled and randomly selecting locations is much far from optimality. The execution time, shown in Fig. 11(b) has a similar tendency with Fig. 9(b). When the flow density grows, the execution time of Alg. DP increases vastly while other four algorithms only have moderate increment. When the flow density reaches the largest value as 0.8, the execution time of Alg. DP is more than 4 times than that of any of other algorithms.

Fig. 12(a) is the result of the bandwidth consumption as the topology size goes from 12 to 32 with an interval of 4. The performance of Alg. Best-effort is also good and has little difference with the bandwidth consumption of our Alg. GTP. The difference between Alg. HAT and Alg. GTP is ignoble when the topology has 20 – 25 vertices. On average, the bandwidth consumption of our Alg. DP is 10.3% less than that of Alg. GTP and 18.6% less than that of Alg. Best-effort. The tendency of the execution times in Fig. 12(b) is also similar to that in Fig. 9(b). Besides that, the increment speed with the growth of the topology size is faster than those of the previous three variables in Figs. 9(b), 10(b), and 11(b). Alg. Best-effort has a close execution time with the Alg. GTP.

6.4 Simulation results in a general topology

The simulation results in a general topology of Fig. 8(b) with k derived from Alg. GTP are shown in Figs. 13, 14, 15, and 16. Fig. 13(a) shows the relationship between bandwidth consumption and k ranging from 12 to 22 with an interval of 2. We compare our proposed Alg. GTP, with Algs. Random and Best-effort. The bandwidth consumption is around three times of that in Fig. 9(a). The possibility of an infeasible deployment plan is higher than in the tree. This is because the general topology has a larger diversity in the flows' paths and serving all flows becomes more difficult. Additionally, the

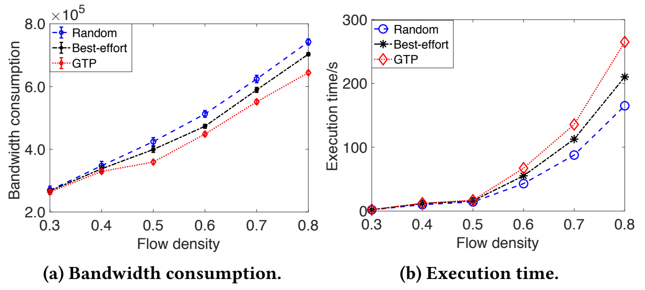

(a) Bandwidth consumption. (b) Execution time.

Figure 15: Flow density in a general topology.

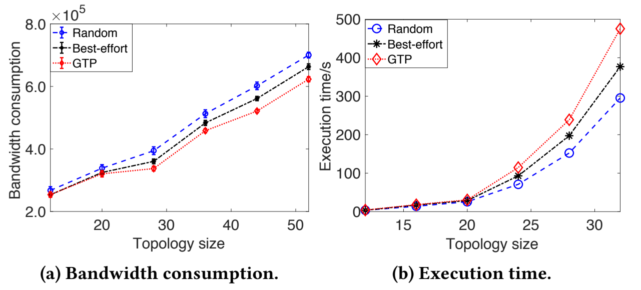

(a) Bandwidth consumption. (b) Execution time.

Figure 16: Topology size in a general topology.

error bars are smaller than in the tree. From Fig. 13(b), Alg. GTP has the longest execution time, indicating the delicate tradeoff between the bandwidth consumption performance and the time efficiency.

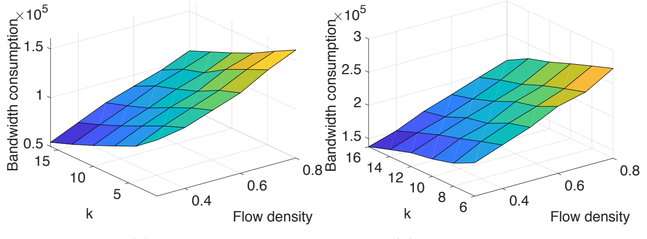
Fig. 14(a) indicates the result of the bandwidth consumption as the traffic-changing effect goes from 0 to 0.9 with an interval of 0.1. The bandwidth consumption increases faster when the traffic-changing ratio is from 0.4 to 0.6. The advantage of our Alg. GTP is less obvious as its bandwidth is only 17.3% less than that of Alg. Random and 8.3% less than that of Alg. Best-effort. The lines are not so smooth, especially when the ratio is around 0.3 to 0.6. Fig. 14(b) shows the execution time results. The tendency is almost linear, which is different from Fig. 10(b), because the general topology has more choices and is more likely to generate infeasible solutions.

The bandwidth with flow density changing from 0.3 to 0.8 with an interval of 0.1 is shown in Fig. 15(a). When the flow density is lower than 0.4, there is little bandwidth difference among the three algorithms. It may be due to the non-optimality of our Alg. GTP and the NP-hardness of our problem in a general topology. When the density is larger than 0.5, the bandwidth of our Alg. HAT is on average 91.4% of the bandwidth of Alg. Random and 93.5% of the bandwidth of Alg. Best-effort. From Fig. 15(b), Alg. GTP has the longest execution time, indicating the delicate tradeoff between the bandwidth consumption performance and the time efficiency.

Fig. 16(a) is the result of bandwidth consumption as topology size goes from 12 to 52 with an interval of 8. The lines are almost linear with the increment of topology size. The bandwidth consumption is nearly three times of the one in Fig. 12(a). The advantage of Alg. GTP becomes larger when topology size increases. Fig. 16(b) is similar to Fig. 15(b).

6.5 Simulation results with spam filters

We additionally do simulations with spam filters, whose traffic-changing ratio is $\lambda = 0$. It illustrates that flows are cut off after



(a) Tree. **(b) General topology.**

Figure 17: Spam filters.

being processed by spam filters. We test the total bandwidth consumption of Alg. GTP with the relationship of flow density and k in the tree and general topologies. Results are shown in Figs. 17 (a) and (b). In order to describe the importance between k and flow density, we draw 3-D plots. From both sub-graphs, we know that flow density plays a more important role in affecting the total bandwidth consumption. This is because the slope of flow density is larger than the slope of k. Additionally, the result increases gently with flow density and decreases gradually with k. In Fig. 17(a), when the flow density doubles from 0.3 to 0.6, the total bandwidth consumption in trees increases 30.2%, while the increment is 25.6% in the general topology in Fig. 17. We find that when k is large, the bandwidth drops quickly, especially with a high density, since more flows are intercepted from sources.

Consequently, the results demonstrate the delicate trade-off between the performance and the time efficiency of our proposed algorithms. The five variables have different extents of impacts on the results while k has the largest impact on the performance of bandwidth consumption. We find that when k grows large, the bandwidth drops quickly, especially with a high flow density. The comparison Alg. Random does not have a steady enough performance, and its error bars are always the largest compared to the other four algorithms.

7 CONCLUSION

In this paper, we address the deployment problem of one single type of middleboxes with traffic-diminishing effect (e.g., spam filters). We aim at minimizing the total bandwidth consumption of all flows by placing a pre-determined number of middleboxes to serve flows. First, we formulate the traffic-diminishing middlebox deployment problem as an optimization problem. We prove that it is NP-hard to check the feasibility of our problem in a general topology. Then a greedy algorithm is proposed and prove it is performance-guaranteed when it generates a feasible deployment. Next we narrow down to the tree-structured networks, and propose both an optimal dynamic programming based strategy and an efficient heuristic strategy. Extensive simulations on CAIDA data set are conducted to evaluate the performance of our proposed algorithms in various scenarios.

8 ACKNOWLEDGMENT

This research was supported in part by NSF grants CNS 1824440, CNS 1828363, CNS 1757533, CNS 1629746, CNS 1651947, CNS 1564128.

REFERENCES

[1] 2012. Energy-aware resource allocation heuristics for efficient management of data centers for Cloud computing. *Future Generation Computer Systems* 28, 5 (2012), 755 – 768.
[2] 2015. Citrix CloudBridge Product Overview. In *Citrix CloudBridge Online*.
[3] M. Al-Fares, A. Loukissas, and A. Vahdat. 2008. A Scalable, Commodity Data Center Network Architecture. *SIGCOMM Comput. Commun. Rev.* 38, 4 (2008), 63–74.
[4] Hari Balakrishnan, Mark Stemm, Srinivasan Seshan, and Randy H. Katz. 1997. Analyzing Stability in Wide-area Network Performance. In *SIGMETRICS 1997*.
[5] CAIDA. 2018. *Archipelago Monitor Locations*. http://www.caida.org/projects/ark/locations/
[6] M. Casado, T. Koponen, R. Ramanathan, and S. Shenker. 2010. Virtualizing the Network Forwarding Plane. In *PRESTO 2010*.
[7] Y. Chen and J. Wu. 2018. NFV Middlebox Placement with Balanced Set-up Cost and Bandwidth Consumption. In *ICPP 2018*.
[8] R. Cohen, L. Lewin-Eytan, J. S. Naor, and D. Raz. 2015. Near optimal placement of virtual network functions. In *INFOCOM 2015*.
[9] Sedef Demirci and Şeref Sağıroğlu. 2019. Optimal placement of virtual network functions in software defined networks: A survey. *Journal of Network and Computer Applications* (2019), 102424.
[10] Vincenzo Eramo and Francesco Giacinto Lavacca. 2019. Optimizing the Cloud Resources, Bandwidth and Deployment Costs in Multi-Providers Network Function Virtualization Environment. *IEEE Access* 7 (2019), 46898–46916.
[11] X. Fei, F. Liu, H. Xu, and H. Jin. 2018. Adaptive VNF Scaling and Flow Routing with Proactive Demand Prediction. In *INFOCOM 2018*.
[12] U. Feige. 1998. A threshold of ln n for approximating set cover. *J. ACM* 45, 4 (1998), 634–652.
[13] H. J. Fowler and W. E. Leland. 1991. Local area network characteristics, with implications for broadband network congestion management. *IEEE Journal on Selected Areas in Communications* 9, 7 (Sep 1991), 1139–1149.
[14] C. Guo, G. Lu, D. Li, H. Wu, X. Zhang, Y. Shi, C. Tian, Y. Zhang, and S. Lu. 2009. BCube: A High Performance, Server-centric Network Architecture for Modular Data Centers. In *SIGCOMM 2009*.
[15] Archit Gupta, Aditya Akella, Srinivasan Seshan, Scott Shenker, Jia Wang, Archit Gupta, Aditya Akella, Srinivasan Seshan, Scott Shenker, and Jia Wang. 2007. Understanding and Exploiting Network Traffic Redundancy. In *SIGMETRICS 2007*.
[16] T Horel. 2015. *Notes on greedy algorithms for submodular maximization*. https://thibaut.horel.org/submodularity/notes/02-12.pdf
[17] Sushant Jain, Alok Kumar, Subhasree Mandal, Joon Ong, Leon Poutievski, Arjun Singh, Subbaiah Venkata, Jim Wanderer, Junlan Zhou, and Min Zhu. 2013. B4: Experience with a globally-deployed software defined WAN. In *ACM SIGCOMM Computer Communication Review*, Vol. 43. ACM, 3–14.
[18] Maryam Jalalitabar, Evrim Guler, Danyang Zheng, Guangchun Luo, Ling Tian, and Xiaojun Cao. 2018. Embedding dependence-aware service function chains. *Journal of Optical Communications and Networking* 10, 8 (2018), C64–C74.
[19] T. Kuo, B. Liou, K. Lin, and M. Tsai. 2016. Deploying chains of virtual network functions: On the relation between link and server usage. In *INFOCOM 2016*.
[20] Y. Li, L. T. X. Phan, and B. T. Loo. 2016. Network functions virtualization with soft real-time guarantees. In *INFOCOM 2016*.
[21] Y. Liu, J.K. Muppala, M. Veeraraghavan, D. Lin, and M. Hamdi. 2016. Data Center Networks: Topologies, Architectures and Fault-Tolerance Characteristics. In *Springer International Publishing, 2016*.
[22] W. Ma, O. Sandoval, J. Beltran, D. Pan, and N. Pissinou. 2017. Traffic aware placement of interdependent NFV middleboxes. In *INFOCOM 2017*.
[23] S. Mehraghdam, M. Keller, and H. Karl. 2014. Specifying and placing chains of virtual network functions. In *CloudNet 2014*.
[24] Montida Pattaranantakul, Ruan He, Qipeng Song, Zonghua Zhang, and Ahmed Meddahi. 2018. NFV security survey: From use case driven threat analysis to state-of-the-art countermeasures. *IEEE Communications Surveys & Tutorials* 20, 4 (2018), 3330–3368.
[25] Bangbang Ren, Deke Guo, Guoming Tang, Xu Lin, and Yudong Qin. 2018. Optimal service function tree embedding for NFV enabled multicast. In *ICDCS 2018*.
[26] G. Sallam, G. R. Gupta, B. Li, and B. Ji. 2018. Shortest Path and Maximum Flow Problems Under Service Function Chaining Constraints. In *INFOCOM 2018*.
[27] G. Sallam and B. Ji. 2019. Joint Placement and Allocation of Virtual Network Functions with Budget and Capacity Constraints. In *INFOCOM 2019*.
[28] Y. Sang, B. Ji, G. Gupta, X. Du, and L. Ye. 2017. Provably efficient algorithms for joint placement and allocation of virtual network functions. In *INFOCOM 2017*.
[29] B. Schieber and U. Vishkin. 1988. On Finding Lowest Common Ancestors: Simplification and Parallelization. *SIAM J. Comput.* 17, 6 (1988), 1253–1262.
[30] S. Seyyedi and B. Akbari. 2011. Hybrid CDN-P2P architectures for live video streaming: Comparative study of connected and unconnected meshes. In *CNDS 2011*. 175–180.
[31] Gerhard J Woeginger. 2000. When does a dynamic programming formulation guarantee the existence of a fully polynomial time approximation scheme (FPTAS)? *INFORMS Journal on Computing* 12, 1 (2000), 57–74.
[32] Bo Yi, Xingwei Wang, Keqin Li, Min Huang, et al. 2018. A comprehensive survey of network function virtualization. *Computer Networks* 133 (2018), 212–262.