Multi-Commodity Flow with In-Network Processing

Moses Charikar, Yonatan Naamad, Jennifer Rexford, and X. Kelvin Zou

Department of Computer Science, Princeton University
{moses,ynaamad,jrex,xuanz}@cs.princeton.edu

Abstract

Recent industry trends towards virtualization of network functions has led to a growing interest in the problems of placement and configuration of so-called "middleboxes" to perform services on the network traffic. The goal is to determine: how many middleboxes to run, where to place them, and how to direct traffic through them. Towards this end, we introduce and study a new class of multi-commodity flow problems. Here, in addition to demands on flows and capacity constraints on edges in the network, there is an additional requirement that flows be processed by nodes in the network.

We study the problems that arise from jointly optimizing the: (1) allocation of middleboxes over a pool of server resources, (2) steering of traffic through middleboxes, and (3) routing of the traffic between the servers over efficient network paths. We introduce and study several problems in this class from the exact and approximation point of view.

We consider the problem of allocating resources within a given network to maximize the processed flow and show that this can be optimized exactly via an LP formulation, and to arbitrary accuracy via an efficient combinatorial algorithm. We also study a class of network design problems where the goal is to purchase processing capacity in order to process and route a given set of demands in a network. We design approximation algorithms as well as obtain hardness of approximation results for four natural problems in this class: the minimization problem (minimize purchase cost so as to process all the demand) and maximization problem (maximize flow processed subject to budget on purchase cost) for both undirected and directed graphs.

1 Introduction

1.1 Background In addition to delivering data efficiently, today's computer networks often perform services on the traffic in flight to enhance security, privacy, or performance, or provide new features. Network administrators frequently install so-called "middleboxes" such as firewalls, network address translators, server load balancers, Web caches, video transcoders, and devices that compress or encrypt the traffic. In fact, many networks have as many middleboxes as they do underlying routers or switches. Often a single conversation, or *connection*, must traverse multiple middleboxes, and different connections may go through different sequences of middleboxes. For example, while Web traffic may go through a firewall followed by a server load balancer, video traffic may simply go through a transcoder. In some cases, the traffic volume is so high that an organization needs to run multiple instances of the same middlebox to keep up with the demand. Deciding how many middleboxes to run, where to place them, and how to direct traffic through them is a major challenge facing network administrators.

Until recently, each middlebox was a dedicated appliance, consisting of both software and hardware. Administrators tended to install these appliances at critical locations that naturally see most of the traffic, such as the gateway connecting a campus or company to the rest of the Internet. A network could easily have a long chain of these appliances at one location, forcing all connections to traverse every appliance—whether they need all of the services or not. In addition, placing middleboxes only at the gateway does not serve the organization's many *internal* connections, unless the internal traffic is routed circuitously through the gateway.

Over the last few years, middleboxes are increasingly *virtualized*, with the software service separate from the physical hardware—an industry trend called Network Functions Virtualization (NFV) [NFV12, OPN]. Middleboxes now run as virtual machines that can easily spin up (or down) on any physical server, as needed. This has led to a growing interest in good algorithms for optimizing the (i) *allocation* of middleboxes over a pool of server resources, (ii) *steering* of traffic through a suitable sequence of middleboxes based on a high-level policy, and (iii) *routing* of the traffic between the servers over efficient network paths [QTC+13, ABFL15].

1.2 The General Problem Rather than solving these optimization problems separately, we introduce—and solve—a joint optimization problem. Since server resources are fungible, we argue that each compute node could subdivide its resources arbitrarily across any of the middlebox functions, as needed. That is, the *allocation* problem is more naturally a question of what fraction of each node's computational (or memory) resources to allocate to each middlebox function. Similarly, each connection can have its middlebox processing performed on any node, or set of nodes, that have sufficient resources. That is, the *steering* problem is more naturally a question of deciding which nodes should devote a share of its processing resources to a particular portion of the traffic. Hence, the joint optimization problem ultimately devolves to a new kind of *routing* problem, where we must compute paths through the network based on both the bandwidth and processing requirements of the traffic between each source-sink pair. That is, a flow from source to sink must be allocated (i) a certain amount of bandwidth on *every* link in its path and (ii) a *total* amount of computation across all of the nodes on its path.

We can formulate the above—the flow with in-network processing model—in the following way: there is a flow demand with multi-sources and multi-sinks, and each flow requires a certain amount of in-network processing. The in-network processing required for a flow is proportional to the flow size and, without losing generality, we assume one unit of flow requires one unit of processing. For a flow from a source to a sink, we assume it is an aggregate flow of many connections so the routing and in-network processing for a flow are both divisible. In this model there are two types of constraints: edge capacity and vertex capacity, which represent bandwidth and node processing capacity, respectively. A feasible flow pattern satisfies: (i) the sum of flows on each edge is bounded by the edge capacity, (ii) the sum

of in-network processing done at each vertex is bounded by the vertex capacity, and (iii) the processing done at all vertices for a flow is equal to the flow size. Though ignoring vertex capacity constraints reduces our class of problems to those of the standard multicommodity flow variety, the introduction of these constraints yields a new class of problems that (to our knowledge) has not yet been studied in the literature.

We aim to solve two variants of the problem in this paper: (i) the basic processed flow routing problem: how to route the flow and steer it through processing nodes; and (ii) a class of network design problems: where to place the physical server resources in the network, for the best utilization of the network and server resources. The two problems are natural for the flow with in-network processing model we introduce, and are both important in their own right: network operators need guidance to purchase and place server resources to run middlebox functions, and to allocate the deployed resources. This paper provides a systematic approach to this new class of network problems, in both directed and undirected graphs.

The main goal of our work is to introduce this interesting class of practically motivated flow problems to the theory community. We explore several problems in this space and give exact and approximation algorithms. For some problems we study, there are tantalizing gaps between algorithms and hardness results ($\Omega(1/\log(n))$ versus constant). Closing these gaps is an interesting challenge for future work.

1.3 Outline of this paper In Section 2, we introduce the PROCESSED FLOW ROUTING class of problems, in which we discuss how to optimize processed flow routed in a fixed network. Our main result here is that given a network with edge capacities, node processing capacities, and flow demands, we give an LP-based algorithm to find a maximum feasible multi-commodity flow with processing assigned to nodes, and an efficient multiplicative weight update algorithm to compute the maximum flow to within arbitrary accuracy. Our result shows an equivalence between an exponential size path-based LP and a polynomial size edge based LP—a generalization of the well known equivalence for max flow. However, the proof here needs a more careful argument. The LP can be adapted to optimize several other objective functions, e.g., sum of congestions. We also discuss the case when multiple processing steps are required before a packet reaches its destination, as may arise in onion routing or while monitoring processed traffic. In Section 3, we discuss the MIDDLEBOX NODE PURCHASE class of problems, in which the goal is to purchase middlebox nodes in a network optimally (the capacities of edges are fixed). Unless stated otherwise, we consider problems where each flow needs a single processing step. We study two natural variants here. (1) MIN MIDDLEBOX NODE PURCHASE: minimize middlebox node costs so to satisfy a given set of demands. We show an $O(\log(n)/\delta^2)$ approximation for node costs and an associated multi-commodity flow that satisfies $(1-\delta)$ fraction of the demands and satisfies all edge capacities. We show that in the directed case, the problem is hard to approximate better than a logarithmic factor, even if the demand requirements are relaxed. We show that the undirected case is at least as hard as VERTEX COVER. We also show that minimizing node costs for the problem with two processing steps is LABEL COVER hard. (2) BUDGETED MIDDLEBOX NODE PURCHASE: purchase middlebox nodes within a specified budget so as to maximize the flow that can be processed and routed in the network. Although it's tempting to conjecture that the problem is an instance of BUDGETED SUBMODULAR MAXIMIZATION, one can construct instances for both directed and undirected graphs where the amount of routable processed flow is *not* submodular in the set of purchased nodes, so black-box submodular maximization techniques cannot be used here. We show an $\Omega(1/\log(n))$ approximation for this problem. For the undirected case with a single source-sink pair, we show a constant factor approximation. For the directed case, we show approximation hardness of $1-1/e$ and constant factor hardness in the undirected case.

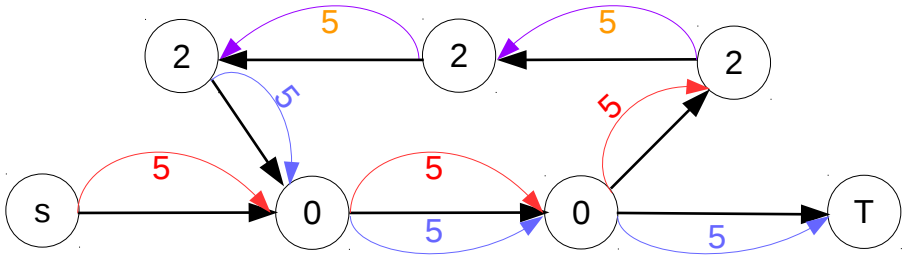

Figure 1: The edge capacity is 10 for all edges and the node capacities are denoted in each node. Here, we can send maximum flow size 5, by routing it along the red arcs, have it processed at the nodes at the top, and then sent to T along the blue arcs. The capacity of the bottom middle edge forms the bottleneck here, as all flow must pass through it twice before reaching T.

2 Flow Routing with In-Network Processing

2.1 The Basic Problem We begin by introducing the routing problem in the presence of processing demands. In this problem, we are given a directed graph $G = (V, E)$ along with edge capacities $B : E \to \mathbb{R}^+$, vertex capacities $C : V \to [0, \infty)$, and a collection of demanded integer flows $D = \{(s_1, t_1, k_1), (s_2, t_2, k_2), \cdots\} \subseteq V \times V \times \mathbb{R}^+$. While the edge capacities are used in a manner entirely analogous to its uses in standard multicommodity flow problems, we also require that each unit of flow undergo one unit of processing at an intermediate vertex. In particular, while edge capacities limit the *total* amount of flow that may pass through an edge, vertex capacities only bottleneck the amount of processing that may be done at a given vertex, regardless of the total amount of flow that uses the vertex as an intermediate node. The goal is then either to route as much flow as possible, or to satisfy all flow demand subject to a congestion-minimization objective function.

2.2 Flow Maximization We begin by showing how to express the maximization version of the problem both as an *edge-based* and as a *walk-based* linear program. While neither of these constructions is particularly difficult, it is not obvious that either is enough to solve the flow problem in polynomial time. In particular, while the walk-based LP requires exponential size, the polynomial-sized edge-based LP may a-priori not correspond to a valid routing pattern at all. In subsubsection A.1.1, we resolve this problem by showing that the two linear programs are equivalent, and so the edge-based LP inherits the correctness of the walk-based program, ensuring that we can indeed find a valid solution in polynomial time. We summarize this result in the following theorem.

THEOREM 2.1. *There exists a polynomial-sized linear program solving the Maximum Processed Flow problem. Further, the full routing pattern can be extracted from the LP solution by decomposing it into its composing s_i, t_i walks in $O(|V| \cdot |E| \cdot |D|)$ time.*

To express the walk-based linear program, we require one variable $p^v_{i,\pi}$ for each walk-vertex-demand triplet, representing the total amount of flow from s_i, t_i exactly utilizing walk π and processed at v. The aggregate (s_i, t_i) flow sent along a given walk π is then simply denoted by $p_{i,\pi}$, and the set of all walks is given by P. The linear program is then the standard multicommodity-flow LP augmented with the new processing capacity constraints.

The edge-based formulation can be thought of as sending two flows for each D_i: f_i represents the flow being sent from s_i to t_i and w_i is the processing demand of this flow. While f_i is absorbed (non-conserved) only at the terminals, w_i is absorbed only at the processing vertices. The variables $f_i(e)$ and $w_i(e)$ measure how much of f_i and w_i passes through edge e. We use the notation $\delta^+(v)$ and $\delta^-(v)$ to denote the edges leaving and entering vertex v, respectively. The two linear programs are given below:

Walk-based formulation:

$$\text{MAXIMIZE} \qquad \sum_{i=1}^{|D|} \sum_{\pi \in P} p_{i,\pi}$$

SUBJECT TO

$$p_{i,\pi} = \sum_{v \in \pi} p_{i,\pi}^{v} \qquad \forall i \in [|D|], \forall \pi \in P$$

$$\sum_{i=1}^{|D|} \sum_{\substack{\pi \in P \\ \pi \ni e}} p_{i,\pi} \leq B(e) \qquad \forall e \in E$$

$$\sum_{i=1}^{|D|} \sum_{\pi \in P} p_{i,\pi}^{v} \leq C(v) \qquad \forall v \in V$$

$$p_{i,\pi}^{v} \geq 0 \qquad \forall i \in [|D|], \forall \pi \in P, \forall v \in V$$

Edge-based formulation:

$$\text{MAXIMIZE} \qquad \sum_{i=1}^{|D|} \sum_{e \in \delta^{+}(s_i)} f_i(e)$$

SUBJECT TO

$$\sum_{e \in \delta^{-}(v)} f_i(e) = \sum_{e \in \delta^{+}(v)} f_i(e) \qquad \forall i \in [|D|], \forall v \in V \setminus \{s_i, t_i\}$$

$$p_i(v) = \sum_{e \in \delta^{-}(v)} w_i(e) - \sum_{e \in \delta^{+}(v)} w_i(e) \qquad \forall i \in [|D|], \forall v \in V$$

$$\sum_{i=1}^{[D]} f_i(e) \leq B(e) \qquad \forall e \in E$$

$$\sum_{i=1}^{|D|} p_i(v) \leq C(v) \qquad \forall v \in V$$

$$w_i(e) \leq f_i(e) \qquad \forall i \in [D], \forall e \in E$$

$$w_i(e) = f_i(e) \qquad \forall i \in [D], \forall e \in \delta^{+}(s_i)$$

$$w_i(e) = 0 \qquad \forall i \in [D], \forall e \in \delta^{-}(t_i)$$

$$w_i(e), p_i(v) \geq 0 \qquad \forall i \in [D], \forall e \in E$$

2.3 Multiplicative Weight Update for Flow Maximization Solving LPs can be expensive, and people have studied applying multiplicative weight update (MWU) [AHK12, PST91] method to more efficiently compute optimal solutions to the multicommodity flow LP.

We show that the same approach can be used in the presence of processing constraints, giving a combinatorial $(1-\epsilon)$ approximation to the problem in time $O(km\log(m)\cdot(m+n\log n)/\epsilon^2)$ (here k is the number of source sink pairs). Since we have both edge capacities and vertex capacities that behave differently, our algorithm and analysis differs from the standard application of the multiplicative weights framework to maximum multicommodity flow. One difference is in the update step, where we could be limited either by the lowest capacity edge along the path, or the sum of node processing capacities. This complicates the analysis.

2.3.1 Formulation We use the walk based LP for Multiplicative Weight Updates (MWU). For each edge e, there is a constraint

$$\sum_{\pi \ni e} p_\pi \leq B(e)$$

and for each node v, there is a constraint

$$\sum_{\pi \ni v} p_\pi^v \leq C(v)$$

We associate one expert and the corresponding weight for each of the two types of capacity constraints. For each edge e and for each vertex v, let w_e and q_v respectively denote the weights of their corresponding experts. We use p to denote a generalized flow, that is, p is the shorthand for all values of p_π^v.

For any p, let $P(p)$ be the amount of flow in p, that is

$$P(p) = \sum_{\pi} \sum_{v} p_\pi^v.$$

For expert e and generalized flow p, let the gain $M(e,p)$ be defined as

$$M(e,p) = \frac{1}{B(e)} \sum_{\pi \ni e} p_\pi,$$

we can think this as the fractional utilization of the edge by the flow, and for expert v define the gain $M(v,p)$ as

$$M(v,p) = \frac{1}{C(v)} \sum_{\pi \ni v} p_\pi^v,$$

corresponding to the fractional utilization of the node's processing capacity.

Let $\mathcal{D}$ be the distribution over experts where the probability of choosing a given expert is proportional to its weight. The expected gain over a random sample from $\mathcal{D}$ is

$$M(\mathcal{D},p) = \frac{\sum_e w_e M(e,p) + \sum_v q_v M(v,p)}{\sum_e w_e + \sum_v q_v}$$

Now we simulate the MWU algorithm. At the beginning we start with weight $w_e = 1/\delta$ and $q_v = 1/\delta$ with $\delta = (1+\epsilon)((1+\epsilon)m)^{-1/\epsilon}$. At each timestep t, given the weights w_e^t and q_v^t of the experts, we provide the experts the event

$$p^t = \operatorname*{argmin}_{\pi \in \Pi} \sum_{e \in \pi} \frac{w_e}{B(e)} + \min_{v \in \pi} \frac{q_v}{C(v)}$$

The algorithm for computing such paths is described in Appendix subsubsection A.1.3.

2.3.2 Update Once the path with smallest cost is computed, we break into two cases:

If $\sum_{v \in \pi^t} C(v) \geq \min_{e \in \pi^t} B(e)$, then let $e^t = \operatorname{argmin}_{e \in \pi^t} B(e)$, and let the generalized flow p^t be the one satisfying

$$p_\pi^{t,v} = \begin{cases} \frac{C_v}{\sum_{v \in \pi^t} C_v} \cdot B_{e^t} & \text{if } \pi = \pi^t, v \in \pi^t \\ 0 & \text{o.w.} \end{cases}$$

If $\sum_{v \in \pi^t} C(v) < \min_{e \in \pi^t} B(e)$, let generalized flow p^t be the one satisfying

$$p_\pi^{t,v} = \begin{cases} C(v) & \text{if } \pi = \pi^t, v \in \pi^t \\ 0 & \text{o.w.} \end{cases}$$

Update condition: Given the signal p^t, the experts gains are $M(e,p^t)$ or $M(v,p^t)$, and they update their weights w_e or q_v to be $w_e(1+\epsilon)^{M(e,p^t)}$ or $q_v(1+\epsilon)^{M(v,p^t)}$, respectively. The algorithm stops when one of the weights w_e or q_v is larger than 1. Once the algorithm terminates, we scale down the computed flow p^t at each round by dividing it by $\log_{1+\epsilon} \frac{1+\epsilon}{\delta} = 1 - \frac{\ln \delta}{\ln 1+\epsilon}$.

Note that at each round, depending on whether or not $\sum_{v \in \pi^t} C(v) \geq \min_{e \in \pi^t} B(e)$, we either increase the weight of one w_e by a factor of $(1+\epsilon)$, or increase all of the q_v's on a path π^t by a factor of $(1+\epsilon)$. Since each w_e and each q_v can only be increased by such a factor at most $\frac{\ln 1/(\delta m)}{\epsilon}$ times before its weight exceeds 1 and the initial weight is $1/\delta$, $T \leq (m+n)\frac{\ln 1/(\delta m)}{\epsilon} \cdot T_{sp} = O(m \log m/\epsilon^2 \cdot T_{sp})$, where T_{sp} is the time to compute the generalized shortest path for each of the k flows. For each of the k different (s_i, t_i) flows, we use the algorithm proposed in subsubsection A.1.3, which can be implemented in time $O(m + n\log n)$ time each (so $O(k \cdot (m + n\log n)$) in total) using Fibonacci heaps. The result is similar to previous work [GK07] for the maximum multicommodity flow problem without the node processing requirement. The analysis details are in the Appendix.

2.4 Multiple Types of In-Network Processing as a Chain Sometimes packets need to be processed in multiple, distinct stages [GJVP$^+$14]. For example, onion routing requires the data to visit a number of intermediaries, each with its own decryption key, before reaching its ultimate destination. Further, certain nodes may be fit for only certain types of computations with inter-dependencies, e.g., decrypting of files after passing through a firewall, or encrypting a file after being compressed. Thus, it is natural to attempt to generalize the above formulation into one that can handle multiple tasks.

One common formulation of this problem, which we will call *dependency routing*, requires a chain of tasks as an additional input. If there T tasks in the network, for each (s_i, t_i) pair, we are given a chain of $\{k_i : k_i \leq T\}$ tasks: $Chain_i = \{P_i^1, P_i^2 \dots P_i^{k_i}\}$, with each P_i being a subset of V. We require that the (s_i, t_i) flow get processed at vertices in the various $\{P_i^j\}$ in sequence, so that the processing of the $(j+1)^{th}$ task in some vertex of P_i^{j+1} only begins after j^{th} task completed its processing in a member of P_i^j. Interestingly, we can encode *dependency routing* into the above edge-based linear program. To do so, each vertex v needs to be given T different processing capacities $C^1(v) \cdots C^T(v)$, one for each task. A naive approach to ensuring feasibility in a general case requires $2^{k_i} = O(2^T)$ new flows be created for each (s_i, t_i) pair. However, the linear dependency allows us to encode the problem with just $k_i + 1 = O(T+1)$ flows. The LP formulation and its analysis is in the Appendix.

3 Middlebox Node Purchase Optimization

In this section, we discuss the problem of how to optimally purchase processing capacity so to satisfy a given flow demand. Although this can be modeled in multiple ways, we limit our discussion to the case where each vertex v has a potential processing capacity C, which can only be utilized if v is "purchased". Flow processed elsewhere can be routed through v regardless of whether or not v is purchased. As in the previous section, this yields two general categories of optimization problems

1. The *minimization* version of the problem (Min Middlebox Node Purchase), where the goal is to pick the smallest set of vertices such that all flow is routable.

2. The *maximization* version of the problem (Budgeted Middlebox Node Purchase), where we try to maximize the amount of routable flow while subject to a budget constraint of k.

Formally, the input to Min Middlebox Node Purchase is a graph $G = (V, E)$, which can be either directed or undirected, with nonnegative costs q_v on its vertices, a potential processing capacity $C : V \to [0, \infty)$, and a collection of (s_i, t_i) pairs with demands R_i. The goal is to select a set $T \subseteq V$ of vertices such that all demands are satisfied. Budgeted Middlebox Node Purchase is given the same collection of inputs along with a budget integer k, and the goal is to route as much of the demand as possible.

All four problems (maximization or minimization, directed or undirected), are **NP**-hard. In this section, we present approximation algorithms and hardness results for each version of the problem, as well as for some restricted variants. Due to space constraints, much of the analysis will be relegated to the appendix. Our results are summarized in Table 1.

Table 1: Network Design Results

		Directed	**Undirected**
Budgeted	**Approximation**	$\Omega(1/\log n)$	$.078^{(\dagger)}$
	Hardness	$1 - 1/e - \epsilon$	.999
Minimization	**Approximation**	$O(\log n)^{(*)}$	$O(\log n)^{(*)}$
	Hardness	$O(\log n)$	$2 - \epsilon$

* All demands are satisfied only up to an $(1 - \epsilon)$ fraction.

† Assuming 1 source-sink pair. For multiple pairs, we adapt the $\Omega(1/\log n)$-approximation digraph algorithm.

3.1 Min Middlebox Node Purchase

3.1.1 Bicriterion Approximation Algorithm for (Un)directed Min Middlebox Node Purchase We first describe an algorithm for directed Min Middlebox Node Purchase that satisfies all flow requirements up to a factor of $1-\delta$ fraction with expected cost bounded by $O(\log n/\delta^2)$ times the optimum.

We begin our approximation algorithm for directed Min Middlebox Node Purchase by modifying the walk-based LP formulation with additional variables x_v corresponding to whether or not processing capacity at vertex v has been purchased. We further give a polynomial sized edge-based LP formulation with flow variables $f_i^{1,v}(e)$ and $f_i^{2,v}(e)$ for each commodity i, each vertex $v \in V$ and each edge $e \in E$. The variables $f_i^{1,v}(e)$ correspond to the (processed) commodity i flow that has been processed by vertex v: these variables describe a flow from v to t_i. The variables $f_i^{2,v}(e)$ correspond to the (unprocessed) commodity i flow that will be processed by vertex v: these variables describe a flow from s_i to v.

Walk-based formulation:

$$\text{MINIMIZE} \sum_{v\in V} q_v x_v$$

SUBJECT TO

$$x_v \le 1 \qquad \forall v \in V$$

$$p_{i,\pi} = \sum_{v\in\pi} p_{i,\pi}^v \qquad \forall i \in [|D|], \pi \in P$$

$$\sum_{\pi\in P} p_{i,\pi} \ge R_i \qquad \forall i \in [|D|]$$

$$\sum_{i=1}^{|D|} \sum_{\substack{\pi\in P\\ \pi\ni e}} p_{i,\pi} \le B(e) \qquad \forall e \in E$$

$$\sum_{i=1}^{|D|} \sum_{\pi\in P} p_{i,\pi}^v \le C(v) x_v \qquad \forall v \in V$$

$$\sum_{i=1}^{|D|} \sum_{\substack{\pi\in P\\ \pi\ni e}} p_{i,\pi}^v \le B(e) x_v \qquad \forall e \in E, v \in V$$

$$\sum_{\pi\in P} p_{i,\pi}^v \le R_i x_v \qquad \forall i \in [|D|], v \in V,$$

$$p_{i,\pi}^v \ge 0 \qquad \forall i \in [|D|], \pi \in P, v \in \pi$$

$$x_v \ge 0 \qquad \forall v \in V$$

Edge-based formulation:

$$\text{MINIMIZE} \sum_{v\in V} q_v x_v$$

SUBJECT TO

$$x_v \le 1 \qquad \forall v \in V$$

$$\sum_{e\in\delta^-(u)} f_i^{j,v}(e) = \sum_{e\in\delta^+(u)} f_i^{j,v}(e) \qquad \forall i \in [|D|], j \in \{1,2\}, v \in V, \forall u \in V \setminus \{s_i, t_i, v\}$$

$$\sum_{e\in\delta^-(v)} f_i^{2,v}(e) = \sum_{e\in\delta^+(v)} f_i^{1,v}(e) \qquad \forall i \in [|D|], v \in V,$$

$$\sum_{v\in V} \sum_{e\in\delta^+(s_i)} f_i^{2,v}(e) \ge R_i \qquad \forall i \in [|D|]$$

$$\sum_{i=1}^{|D|} \sum_{v\in V} (f_i^{1,v}(e) + f_i^{2,v}(e)) \le B(e) \qquad \forall e \in E$$

$$\sum_{i=1}^{|D|} \sum_{e\in\delta^-(v)} f_i^{2,v}(e) \le C(v) x_v \qquad \forall v \in V$$

$$\sum_{i=1}^{|D|} (f_i^{1,v}(e) + f_i^{2,v}(e)) \le B(e) x_v \qquad \forall e \in E, v \in V$$

$$\sum_{e\in\delta^+(s_i)} f_i^{2,v}(e) \le R_i x_v \qquad \forall i \in [|D|], v \in V$$

$$f_i^{2,v}(e) = 0 \qquad \forall i \in [|D|], v \in V, e \in \delta^-(s_i)$$

$$f_i^{1,v}(e) = 0 \qquad \forall i \in [|D|], v \in V, e \in \delta^+(t_i)$$

$$p_i^{1,v}(e), p_i^{2,v}(e), x_v \ge 0 \qquad \forall i \in [|D|], v \in V, e \in E$$

Given an optimal solution to this LP, we pick vertices to install processing capacity on by randomized rounding: pick vertex v with probability x_v. if x_v is picked, then all flows processed by v are rounded up in the following way: $\hat{F}_i^{j,v}(e) = f_i^{j,v}(e)/x_v$ for all $i \in [|D|], j \in \{1,2\}, e \in E$. If v is not picked, then all flows processed by v are set to zero, i.e. $\hat{F}_i^{j,v}(e) = 0$.

By design, $E[\hat{F}_i^{j,v}(e)] = f_i^{j,v}(e)$. In the solution produced by the rounding algorithm, the total flow through edge e is $\sum_{v \in V} \sum_{i=1}^{|D|} ((\hat{F}_i^{1,v}(e) + \hat{F}_i^{2,v}(e))$. This is a random variable whose expectation is at most $B(e)$, and is the sum of independent random variables, one for each vertex v. The constraints of the LP ensure that if v is selected, then the total processing done by vertex v is at most $C(v)$. Further, the total contribution of vertex v to the flow on edge e does not exceed the capacity $B(e)$, i.e. $\sum_{i=1}^{|D|} (\hat{F}_i^{1,v}(e) + \hat{F}_i^{2,v}(e)) \leq B(e)$. Also, the total contribution of vertex v to the commodity i flow is at most R_i, i.e. $\sum_{e \in \delta^+(s_i)} \hat{F}_i^{2,v}(e) \leq R_i$.

We repeat this randomized rounding process $t = O(\log(n)/\epsilon^2)$ times. Let $g^k(e)$ denote the total flow along edge e, and h_i^k denote the total amount of commodity i flow in the solution produced by the kth round of the randomized rounding process. The following lemma follows easily by Chernoff-Hoeffding bounds:

LEMMA 3.1.

$$Pr\left[\sum_{k=1}^{t} g^k(e) \geq (1+\epsilon)t \cdot B(e)\right] \leq e^{-t\epsilon^2/3} \qquad \forall e \in E \tag{3.5}$$

$$Pr\left[\sum_{k=1}^{t} h_i^k \leq (1-\epsilon)t \cdot R_i\right] \leq e^{-t\epsilon^2/2} \qquad \forall i \in [|D|] \tag{3.6}$$

We set $t = O(\log(n)/\epsilon^2)$ so that the above probabilities are at most $1/n^3$ for each edge $e \in E$ and each commodity i. With high probability, none of the associated events occurs. The final solution is constructed as follows: A vertex is purchased if it is selected in any of the t rounds of randomized rounding. Thus the expected cost of the solution is at most $t = O(\log(n)/\epsilon^2)$ times the LP optimum. We consider the superposition of all flows produced by the t solutions and scale down the sum by $t(1+\epsilon)$. This ensures that the capacity constraints are satisfied. Note that the vertex processing constraints are also satisfied by the scaled solution. The total amount of commodity i flow is at least $\frac{1-\epsilon}{1+\epsilon} R_i \geq (1-2\epsilon)R_i$. Hence we get the following result:

THEOREM 3.1. *For directed* MIN MIDDLEBOX NODE PURCHASE*, there is a polynomial time randomized algorithm that satisfies all flow requirements up to factor $1-\delta$ and produces a solution that respects all capacities, with expected cost bounded by $O(\log(n)/\delta^2)$ times the optimal cost.*

We can modify the LP to simulate the inclusion of an undirected edge with capacity $B(e)$ by adding the constraints for two arcs between its endpoints with capacity $B(e)$ each, as well as an additional constraint requiring that the sum of flows over these two arcs is bounded by $B(e)$. The analysis done above carries through line-by-line, giving the following result.

THEOREM 3.2. *For undirected* MIN MIDDLEBOX NODE PURCHASE*, there is a polynomial time randomized algorithm that satisfies all flow requirements up to factor $1-\delta$ and produces a solution that respects all capacities, with expected cost bounded by $O(\log(n)/\delta^2)$ times the optimal cost.*

3.1.2 Hardness of Directed Min Middlebox Node Purchase We now prove that directed MIN MIDDLEBOX NODE PURCHASE is NP-hard to approximate to a factor better than $(1-\epsilon)\ln n$

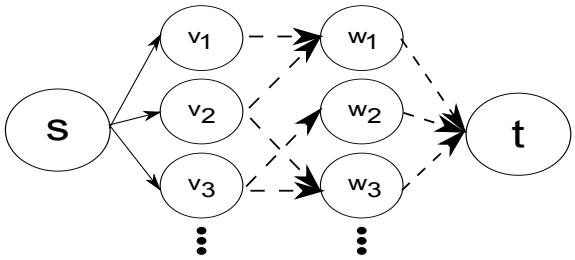

Figure 2: Approximation-preserving reduction from Set Cover and Max k-Coverage to directed Min Middlebox Node Purchase and directed Budgeted Middlebox Node Purchase. Solid edges have infinite capacity, dashed edges have capacity 1. v_i vertices have infinite processing potential, at a cost of 1 each.

by showing an approximation-preserving reduction from Set Cover, a problem already known to have the aforementioned $(1-\epsilon)\ln n$ hardness [DS14].

Given a Set Cover instance with set system $\mathcal{S} = \{S_1, S_2, \cdots\}$ and universe of elements $\mathcal{U}$, we create one vertex v_S for each $S \in \mathcal{S}$ and one vertex w_u for each $u \in \mathcal{U}$. Further, we create one source vertex s and one sink vertex t, where t demands $|\mathcal{U}|$ units of processed flow from s. We add one capacity-n arc from s to each v_S, and one capacity-1 arc from each w_u to t. We then add a capacity-1 arc from each v_S to w_u whenever $S \ni u$. Finally, we give each v_S vertex n units of processing capacity at a cost of 1 each.

In order for t to get $|\mathcal{U}|$ units of flow, each w_u must get at least one unit of processed flow itself. Thus, at least one of of its incoming v_S neighbors must be able to process flow. Therefore, this instance of directed Min Middlebox Node Purchase can be seen as the problem of purchasing as few of the v_S vertices so that each u_W vertex has one (or more) incoming v_S vertex. This provides a direct one-to-one mapping between solutions to our constructed instance and the initial Set Cover instance, and the values of the solutions are conserved by the mapping. Therefore, we have an approximation-preserving reduction between the two problems, and directed Min Middlebox Node Purchase acquires the known $(1-\epsilon)\ln n$ inapproximability of Set Cover, summarized in the following result:

Theorem 3.3. *For every $\epsilon > 0$, it is* **NP**-hard *to approximate directed* Min Middlebox Node Purchase *to within a factor of $(1-\epsilon)\ln n$.*

Note that this construction provides the same hardness even when all demands are only to be satisfied up to a $(1-\delta)$ fraction, showing the asymptotic tightness of the approximation factor in Theorem 3.1.

3.1.3 Hardness of Undirected Min Middlebox Node Purchase One can derive an approximation-preserving reduction from Min Vertex Cover to undirected Min Middlebox Node Purchase by requiring one unit of flow be sent from every vertex to each of its neighbors on a graph with edge capacities 2. A complete analysis is presented in subsubsection A.2.3. The result is summarized below:

Theorem 3.4. *Approximating Undirected* Min Middlebox Node Purchase *is at least as hard as approximating* Min Vertex Cover. *In particular, it is* **NP**-hard *to approximate within a factor of* 1.36 *and* **UGC**-hard *to approximate within a factor of $2-\epsilon$, for any $\epsilon > 0$.*

3.2 Budgeted Middlebox Node Purchase

3.2.1 Algorithms for Budgeted Middlebox Node Purchase Although it's tempting to conclude that Budgeted Middlebox Node Purchase is an instance of Budgeted Submodular Maximization, one can construct instances where the amount of routable processed flow is *not* submodular in

the set of purchased nodes, so black-box submodular maximization techniques cannot be used here (an example non-submodular instance can be found in subsubsection A.3.1). Instead, we show that the randomized rounding algorithm used above for Min Middlebox Node Purchase can be reworked to work for Budgeted Middlebox Node Purchase, as well. Using a slightly modified LP, we derive an $\Omega(1/\log n)$ approximation algorithm for Budgeted Middlebox Node Purchase. Analysis is deferred to subsubsection A.2.1. The results of that section are summarized below:

Theorem 3.5. *For both the directed and undirected versions of* Budgeted Middlebox Node Purchase*, there is a polynomial-time randomized algorithm producing an* $\Omega(1/\log(n))$ *approximation to the optimal solution.*

In the case that an undirected Budgeted Middlebox Node Purchase instance has a single source, we show that the problem admits a constant-factor approximation algorithm by showing how to find a solution within a $(1-1/e)/8 \approx .078$ factor of the optimum. Details are left to subsubsection A.2.2. The result is summarized below:

Theorem 3.6. *For undirected* Budgeted Middlebox Node Purchase *with a single source, there is a deterministic polynomial time algorithm that produces a solution that can route at least* $(1-1/e)/8 \approx .078$ *times the optimal solution.*

3.2.2 Hardness of Budgeted Middlebox Node Purchase In subsubsection A.2.4, we provide an approximation-preserving reduction from Max k-Coverage reminiscient of the Set Cover reduction from subsubsection 3.1.2. The result is given below:

Theorem 3.7. *For every* $\epsilon > 0$*, it is* **NP**-hard *to approximate Directed* Budgeted Middlebox Node Purchase *to within a factor of* $(1 - 1/e + \epsilon)$*.*

NP-hardness of the undirected version of the problem can be shown trivially by reducing Knapsack to Budgeted Middlebox Node Purchase on a clique with infinite edge capacities. In subsubsection A.2.5, we show a stronger statement by ruling out the existence of a PTAS for Budgeted Middlebox Node Purchase by providing a reduction from Max Bisection on 3-uniform graphs. In particular, we provide a .999 hardness of approximation for the problem. No attempt to optimize this constant. The result is summarized below:

Theorem 3.8. *It is* **NP**-hard *to approximate undirected* Budgeted Middlebox Node Purchase *to within a factor better than* .999*.*

3.3 Hardness of Dependency Routing The natural extension of Min Middlebox Node Purchase to dependency routing (as introduced in subsection 2.4) naturally encodes Min Rep as defined in [Kor01], acquiring its Label-Cover hardness. Details about the construction are given in **??**, and the conclusion is summarized as follows:

Theorem 3.9. *For every* $\epsilon > 0$*, there is no polynomial-time algorithm approximating the* Dependency Min Middlebox Node Purchase *problem to within an* $O(2^{\log^{(1-\epsilon)} n})$ *factor unless* $NP \subseteq$ $\mathbf{DTIME}(n^{\text{polylog}\, n})$*.*

References

[ABFL15] Bilal Anwer, Theophilus Benson, Nick Feamster, and Dave Levin. Programming Slick network functions. In *Proceedings of Symposium on SDN Research*, June 2015.

[AHK12] Sanjeev Arora, Elad Hazan, and Satyen Kale. The multiplicative weights update method: A meta-algorithm and applications. *Theory of Computing*, 8(1):121–164, 2012.

[BK99] Piotr Berman and Marek Karpinski. *On Some Tighter Inapproximability Results*. Springer, 1999.

[CKLN13] Deeparnab Chakrabarty, Ravishankar Krishnaswamy, Shi Li, and Srivatsan Narayanan. Capacitated network design on undirected graphs. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques*, pages 71–80. Springer, 2013.

[DS05] Irit Dinur and Samuel Safra. On the hardness of approximating minimum vertex cover. *Annals of Mathematics*, pages 439–485, 2005.

[DS14] Irit Dinur and David Steurer. Analytical approach to parallel repetition. In *Proceedings of the Annual ACM Symposium on Theory of Computing*, pages 624–633, New York, NY, USA, 2014. ACM.

[Fei98] Uriel Feige. A threshold of ln n for approximating set cover. *Journal of the ACM (JACM)*, 45(4):634–652, 1998.

[FT87] Michael L Fredman and Robert Endre Tarjan. Fibonacci heaps and their uses in improved network optimization algorithms. *Journal of the ACM (JACM)*, 34(3):596–615, 1987.

[GJVP+14] Aaron Gember-Jacobson, Raajay Viswanathan, Chaithan Prakash, Robert Grandl, Junaid Khalid, Sourav Das, and Aditya Akella. OpenNF: Enabling innovation in network function control. In *Proceedings of the ACM Conference on SIGCOMM*, pages 163–174. ACM, 2014.

[GK07] Naveen Garg and Jochen Koenemann. Faster and simpler algorithms for multicommodity flow and other fractional packing problems. *SIAM Journal on Computing*, 37(2):630–652, 2007.

[Kor01] Guy Kortsarz. On the hardness of approximating spanners. *Algorithmica*, 30(3):432–450, 2001.

[KR08] Subhash Khot and Oded Regev. Vertex cover might be hard to approximate to within 2- ε. *Journal of Computer and System Sciences*, 74(3):335–349, 2008.

[NFV12] Network Functions Virtualisation: Introductory white paper. In *SDN and OpenFlow World Congress*, Oct 2012. `https://portal.etsi.org/NFV/NFV_White_Paper.pdf`.

[OPN] OPNFV. Linux Foundation, `https://www.opnfv.org/`.

[PST91] Serge A. Plotkin, David B. Shmoys, and E. Tardos. Fast approximation algorithms for fractional packing and covering problems. In *Proceedings of the Symposium on the Foundations of Computer Science*, pages 495–504, Oct 1991.

[QTC+13] Zafar Ayyub Qazi, Cheng-Chun Tu, Luis Chiang, Rui Miao, Vyas Sekar, and Minlan Yu. SIMPLE-fying Middlebox Policy Enforcement Using SDN. In *Proceedings of ACM SIGCOMM*, pages 27–38. ACM, 2013.

[Ski90] S Skiena. Dijkstra's algorithm. *Implementing Discrete Mathematics: Combinatorics and Graph Theory with Mathematica, Reading, MA: Addison-Wesley*, pages 225–227, 1990.

[Svi04] Maxim Sviridenko. A note on maximizing a submodular set function subject to a knapsack constraint. *Operations Research Letters*, 32(1):41–43, 2004.

A Appendix

A.1 Flow Routing with In-Network Processing

A.1.1 Proof of Equivalence between Two LPs in subsection 2.2 Proof sketch: we first show that we can compose an *edge-based* solution based on a *walk-based* solution and vice versa for a single flow, and then show that we can iteratively place multi-commodity flows.

1. show *Direction A:* If there is a *walk-based* LP solution, there is an *edge-based* solution.

2. show *Direction B:* If there is an *edge-based* LP solution, there is a *walk-based* solution using walk decomposition.

3. show the formulations for multi-commodity flows are also equivalent via extending the above approach.

***Walk-based* solution → *Edge-based* solution**

Proof. we show that we can easily convert a walk-based solution to an edge-based solution and all the constraints in the edge-based formulation hold.

For each edge e, $f_i(e) = \sum_{\pi \in P: e \in \pi} p_{i,\pi}$.

For each vertex v, $w_i(e) = \sum_{v' \in \pi, v' \leq v} p^v_{i,\pi}$ ($v' \leq v$ means e' is topologically at or after e on the walk π).

Flow conservation holds $\sum_{(u,v) \in E} f_i(e) = \sum_{\pi \in P, v \in \pi} p_{i,\pi} = \sum_{(v,w) \in E} f_i(e)$.

Constraints in terms of $B(e), C(v)$ also hold. (A2d,2e)

Relations between $w_i(e), f_i(e)$ also hold: $w_i(e) = \sum_{v' \in \pi, v' \leq v} p^v_{i,\pi} \leq \sum_{v \in \pi} p^v_{i,\pi} = f_i(e)$, and $w_i(s,v) = f_i(s,v)$ and $w_i(v,t) = 0$ are special cases.(A.2f,2g,2h)

***Edge-based* solution → *Walk-based* solution**

To prove this; we need to show:

1. We can always construct a walk if there is some residual flow left in the graph.
2. All constraints holds for the updated residual graph.

Setup: For simplicity, we only construct all walks for a flow each time, so notation wise we can remove i. A directed graph $G(V, E)$ with an *edge-based* LP solution, where $f(e)$ is the flow for each edge, $w(e)$ is workload demand at the same edge and $p(v)$ process work done at each vertex v. Build a new graph G': all vertices V, and for $\forall e \in E$, if $f(e) > 0$, we put a direct edge e in the graph. To help proof, divide a flow into two states, processed and unprocessed f^1 and f^2; in terms of flow volume $f^1 = w$ and $f^2 = f - w$.

LEMMA A.1. *loops for flows f^1 and f^2 respectively can be cancelled via flow cancellation without affecting (s,t) flow.*

Proof. it is similar to flow cancellation in a simple graph model:

(i) for e=(u,v) whereas $min(f^1) > 0$, we can simply cancel the unprocessed flow demand by small amount ϵ, and it does not affect the outcome of the flow outside the loop, while we can reduce the flow load and workload demand in the loop without side effect.

(ii) for e=(u,v) whereas $min(f^2) > 0$, we can cancel the processed flow demand by small amount ϵ, and this does not affect the outcome of the flow outside of the loop while we can reduce the flow load in the loop without side effect.

The intuition behind this is that loop exists due to that some flow needs to borrow some processing capacity from some node(s), so it would "detour" a flow in an unprocessed state and get back the flow in a processed state.

Introduce an intermediate variable ρ for each edge e where $\rho_e = \frac{w(e)}{f(e)} = \frac{f^1}{f^1+f^2}$. Run flow loop cancellation for f^1 and f^2 respectively in G'.

Note: after loop cancellation we may still have loops for f as a unitiy.

LEMMA A.2. *ρ has the following property: if there is a cycle for unity flow f, there is always at least one edge with $\rho = 1$ and one edge with $\rho = 0$.*

Proof. This can be easily inferred from Lemma A.1.

Algorithm 1 Walk Decomposition

Data: $G'(V, E)$, $w(e)$, $f(e)$ for $\forall e \in E$ and $p(v)$ for $\forall v \in V$
Result: $f(\pi)$, $p(\pi, v)$ where $v \in \pi$
Algorithm `Walk Construction()`

//Construct walk from $s \to v$ and $v \to t$ From v run backward traversal, pick an incoming directed edge with $max(\rho_{in})$ where $\rho_{in} \equiv \frac{w(e_{in})}{f(e_{in})}$ From v run forward traversal, pick an outgoing directed edge with $min(\rho_{out})$ where $\rho_{out} \equiv \frac{w(e_{out})}{f(e_{out})}$ **return** π

Algorithm `Flow Placement()`

while $\exists v; p(v) > 0$ **do**
  //walk representation $\pi \equiv< v_1, \ldots, v_k >\equiv< e_1, \ldots, e_{k-1} >$ π = `Walk Construction()`
  $p_\pi = \min\{f^1(e^a), f^2(e^b), p(v)\}$, $e^a \in< e_1, \ldots, u \to v >, e^b \in< v \to w, \ldots, e_{k-1} >$ $p^v_\pi = p_\pi$
  for $u \in \pi$ *and* $u \neq v$ **do**
    $p^v_\pi = 0$
  end
  $C(v) = C(v) - p_\pi$ $p(v) = p(v) - p_\pi$ **for** $i \leftarrow 1$ **to** $k - 1$ **do**
    $f(e_i) = f(e_i) - p_\pi$ $B(e_i) = B(e_i) - p_\pi$
  end
end

LEMMA A.3. (WALK CONSTRUCTION) *algorithm 1 can always generate a walk with non-zero flow from source to sink if there exists any v where $p(v) > 0$, and the complexity of the algorithm is $O(|V| \cdot |E|)$*

Proof. First, from Lemma A.2, the walk cannot loop a cycle twice from [Walk Construction]. Since downstream traversal keeps picking $\min \rho$ while upstreaming traversal keeps picking $\max \rho$, so we never pick the same edge twice. Since $p(v) > 0$ so at the same node there must be one upstream edge with $\rho > 0$ and downstream edge with $\rho < 1$. Since the same edge is never picked twice so there is no loop in terms of f^1 and f^2. The walk consists of two DAGs, one is from source to v and one is from v to sink, the walk is a DAG as well.

Second we need to show for a certain walk $\pi; p_\pi > 0$. Since $p_\pi = \min\{f^1(e^a), f^2(e^b), p(v)\}$; at node v where $p(v) > 0$, so we have $f^1(e_{in}) > 0$ and $f^2(e_{out}) > 0$ at vertex v. Since we only pick $\max\{\rho\}$ for upstream traversal, so for $\forall e^a; f^1(e^a) > 0$. The same reason we have $\forall e^b; f^2(e^b) > 0$.

For a single flow, after each iteration, we either take out one edge or one vertex, and the runtime for each iteration is $O(|V|)$ for traversal. As we iterate through $O(|V|)$ vertices and $O(|E|)$ for edges so the runtime total will be $O(|E| + |V|) \cdot |V|) = O(|V| \cdot |E|)$.

LEMMA A.4. (FLOW PLACEMENT) *algorithm 1 conserves all the constraints for the reduced graph.*

Proof. we show that all the constraints are satisfied:
for A.2b: $\forall v \in \pi; \sum_{in} f(e) - \sum_{out} f(e) = \sum_{in \neq e_i} f(e) - \sum_{out \neq e_{i+1}} f(e) + [f(e_i) - p_\pi] - [f(e_{i+1})p_\pi] = 0$
A.2d: $\forall e \in \pi; f(e) = f(e) - p_\pi \leq B(e) - p_\pi = B^{new}(e)$
A.2e: $\forall v \in \pi; p(v) - p_\pi \leq C(v) - p_\pi = C^{new}(v)$
A.2f and A.2g are ensured by the algorithm, since $v \neq s$ and $v \neq t$.
A.2h constraints are satisfied by numerical relations.

Multi-Commodity Flow For MCF, we can use the same approach above. For a graph with K source-sink paired flows, we iterate $i = 1 \ldots K$, for each flow we genrate a G' and exhaustively decompose

walks for f_i and it is easy to see that all the constraints still hold after flow i has been removed. In particular, we have : A.2b, A.2c, A.2f and A.2g hold for all the flows left after one flow is removed;

A.2d: $\forall i, \forall e; \sum_{l=i}^{K} f_l(e) - f_i \leq B(e) - f_i(e)$;

A.2e: $\forall i, \forall v; \sum_{l=i}^{K} p_l(v) - p_i(v) \leq C(v) - p_i(v)$.

A.1.2 Multiplicative Weight Update Analysis To remind you, the definition of $\mathcal{D}$ is the distribution over experts where the probability of choosing a given expert is proportional to its weight. The expected gain over a random sample from $\mathcal{D}$ is

$$M(\mathcal{D}, p) = \frac{\sum_e w_e M(e,p) + \sum_v q_v M(v,p)}{\sum_e w_e + \sum_v q_v}$$

We first make two observations:

Observation 1: For any feasible flow p, then $0 \leq M(\mathcal{D}, p) \leq 1$. This is because $M(e,p) \leq 1$ and $M(v,p) \leq 1$.

Observation 2: For any flow p and weights w, q, if $\pi^* = \text{argmin}_\pi \left(\sum_{e\in\pi} w_e/B_e + \min_{v\in\pi} q_v/C_v\right)$, then

$$M(\mathcal{D}, p) \geq \frac{P(p)(\sum_{e\in\pi^*} w_e/B_e + \min_{v\in\pi^*} q_v/C_v)}{\sum_e w_e + \sum_v q_v}$$

The proof is due to the fact that:

$$M(\mathcal{D}, p) = \frac{\sum_e w_e M(e,p) + \sum_v q_v M(v,p)}{\sum_e w_e + \sum_v q_v} = \frac{\sum_\pi \left(p_\pi(\sum_e w_e/B_e) + \sum_{v\in\pi} p_\pi^v q_v/C_v\right)}{\sum_e w_e + \sum_v q_v} \tag{A.1}$$

$$\geq \frac{\sum_\pi \left(p_\pi(\sum_e w_e/B_e) + \sum_{v\in\pi} p_\pi^v \cdot \min_{v\in\pi} q_v/C_v\right)}{\sum_e w_e + \sum_v q_v} \tag{A.2}$$

$$\geq \frac{\sum_\pi \left(p_\pi(\sum_e w_e/B_e + \min_{v\in\pi} q_v/C_v\right)}{\sum_e w_e + \sum_v q_v} \tag{A.3}$$

$$\geq \frac{\sum_\pi p_\pi \cdot \min_\pi(\sum_{e\in\pi} w_e/B_e + \min_{v\in\pi} q_v/C_v)}{\sum_e w_e + \sum_v q_v} \tag{A.4}$$

$$\geq \frac{P(p)(\sum_{e\in\pi^*} w_e/B_e + \min_{v\in\pi^*} q_v/C_v)}{\sum_e w_e + \sum_v q_v} \tag{A.5}$$

Where π^* is the path with the smallest value. This would lead to the objective we compute at each round — the path π with minimum cost.

Let $\bar{p} = \sum_{t=1}^{t} p^t$, i.e., the total number of flow placed after t rounds. By the guarantee of MWU (Theorem 2.5 in [AHK12], or Analysis in Section 3.4.1 in [AHK12]), we have that for any e and any v

$$\sum_{t=1}^{T} M(\mathcal{D}^t, p^t) \geq \frac{\ln(1+\epsilon)}{\epsilon} M(e, \bar{p}) - \frac{\ln m}{\epsilon}$$

$$\sum_{t=1}^{T} M(\mathcal{D}^t, p^t) \geq \frac{\ln(1+\epsilon)}{\epsilon} M(v, \bar{p}) - \frac{\ln m}{\epsilon}$$

Since at time T, $w_e^T = w_e^0(1+\epsilon)^{M(e,\bar{p})}$, and $q_v^T = w_v^0(1+\epsilon)^{M(v,\bar{p})}$, and the stopping rule that at the end there exists e or v such that $w_e^T \geq 1$ or $q_v^T \geq 1$, we have that there exists e such that $M(e,\bar{p}) \geq \frac{\ln 1/\delta}{\ln(1+\epsilon)}$

or there exists v such that $M(v, \bar{p}) \geq \frac{\ln 1/\delta}{\ln(1+\epsilon)}$. Therefore we have by guarantee of MWU,

$$\sum_{t=1}^{T} M(\mathcal{D}^t, p^t) \geq \frac{\ln 1/\delta}{\epsilon} - \frac{\ln m}{\epsilon}$$

We bound the LHS of the inequality above now. Note that

$$M(\mathcal{D}^t, p^t) = \frac{\sum_e w_e^t M(e^t, p^t) + \sum_v q_v^t M(v^t, p^t)}{\sum_e w_e^t + \sum_v q_v^t} = \frac{P(p^t) \cdot (\sum_{e \in \pi^t} w_e^t / B_e + \min_{v \in \pi^t} q_v^t / C_v)}{\sum_e w_e^t + \sum_v q_v^t}$$

By the definition of π^t and Observation 2, we have

$$M(\mathcal{D}^t, p^t) = \frac{P(p^t)(\sum_{e \in \pi^t} w_e^t / B_e + \min_{v \in \pi^t} q_v^t / C_v)}{\sum_e w_e^t + \sum_v q_v^t} \leq P(p^{opt})^{-1} P(p^t)$$

Therefore we have that

$$P(p^{opt})^{-1} P(\bar{p}) \geq \sum_{t=1}^{T} M(\mathcal{D}^t, p^t) \geq \frac{\ln(\delta^{-1} m^{-1})}{\epsilon}$$

Fixing any edge e, it's initial weight is $1/\delta$ and the final weight is at most $1 + \epsilon$, therefore the $\bar{p}$ has at most $B_e \log_{1+\epsilon}(1+\epsilon)/\delta$ flow on it. Similarly for the v's. In other words, scaling down $\bar{p}$ by $\log_{1+\epsilon}(1+\epsilon)/\delta$ will result in a feasible flow. Let $p' = \bar{p} / \log_{1+\epsilon} \frac{1+\epsilon}{\delta}$. Therefore we have

$$P(p^{opt})^{-1} P(p') \geq P(p^{opt})^{-1} P(\bar{p}) / \log_{1+\epsilon} \frac{1+\epsilon}{\delta} \geq \frac{\ln(\delta^{-1} m^{-1})}{\epsilon} / \log_{1+\epsilon} \frac{1+\epsilon}{\delta}$$

Taking $\delta = (1+\epsilon)((1+\epsilon)m)^{-1/\epsilon}$, we have that

$$\frac{P(p')}{P(p^{opt})} \geq (1 - \epsilon)$$

A.1.3 Computing Min-Cost Path To compute the path with minimum cost, we use a dynamic programming algorithm reminiscent of Dijkstra's shortest path algorithm. Given a graph $G(V, E)$, with weights $w(e)$ on edges, weights $n(v)$ on nodes, and some source-sink pair s, t, we are interested in computing the following quantity

$$gsp(s, t) = \min_{p: s \to t, v \in p} cost(p, v) \tag{A.6}$$

where $cost(p, v)$ is defined as

$$cost(p, v) := \left(\sum_{e \in p} w(e) + n(v) \right)$$

where $w(e)$ and $n(v)$ are set to $\frac{w_e}{B(e)}$ and $\frac{q_v}{C(v)}$ from the initial problem. We also use the notation $cost(p)$ to represent the minimum value $cost(p, v)$ takes over all choices of v:

$$cost(p) := \min_v cost(p, v) = \left(\sum_{e \in p} w(e) + \min_{v \in p} n(v) \right)$$

We use the following algorithm to calculate $gsp(s, t)$ for a fixed s and all $t \in V$.

Require: Graph $G = (V, E)$ with edge weights $w(e)$, node weights $n(v)$, and a designated source s.
Ensure: $r(v) = gsp(s, v)$ for every $v \in V$.
Calculate $d(v)$, the shortest path between s and v using Dijkstra's algorithm.
Initialize $r(v) \leftarrow d(v) + n(v)$ for all $v \in V$. $S \leftarrow \{s\}$.
while $S \neq V$ **do**
Let $u^* = \text{argmin}_{v \in V \setminus S} r(v)$. Add u^* to S.
For all neighbors z of u^* that are not already in S, let $r(z) \leftarrow \min\{r(u^*) + w(u, z), r(z)\}$
end while

Proof. First of all, it is not hard to see that the value of $r(u)$ only decreases, and at any time, there exists a path p from s to u and $v \in p$ such that $cost(p, v) = r(u)$. We prove the following statement by induction:

Claim 1: At the beginning of every while loop, for any node $u \notin S$, and any path p from s to u with last but one endpoint being $z \in S$, we should have $r(u) \leq cost(p)$.

Claim 2: After the first line of each while loop, we have $r(u^*) = \min_{p:s \to u^*} cost(p)$.

We first prove that if claim 1 is true for loop i and Claim 2 is true for loop $1, 2, \ldots, i-1$, then Claim 2 is true for loop i as well. Assuming Claim 1 is true, we are going to prove that for any path p from s to u^*, $r(u^*) \leq cost(p)$. We divide into three cases. If p has last but one endpoint being $z \in S$, then by Claim 1, $r(u^*) \leq cost(p)$. In the second case, if $v \notin S$, we claim $r(u^*) \leq d(v) + n(v) \leq cost(p, v)$, where the first inequality is by the definition of u^*, and the fact that $r(v) \leq d(v)+n(v)$ for any v. (Note that r is decreasing and $r(v)$ is initialized as $d(v)+n(v)$.). Therefore the only cases that are left for consideration are those cases when p has last but one endpoint $z \notin S$, and $v \in S$. We claim this case never appears by showing a contradiction. Let v' be the last point in p that belongs to S, and let z' be the following node in p. Under the assumption of this case, such v' and z' exist. Consider the subpath p' of p from s to z'. Note that $v' \in S$, and by Claim 2 at previous loops, when v' was added to S, we have $gsp(s, v') = r(v')$ and then in the same loop, $r(z')$ was updated with a value that is not greater than $r(v') + w(v', z')$. Because the value of r never increases, we have $r(z') \leq gsp(s, v') + w(v', z') = r(v') + w(v', z')$ at the loop i. Therefore certainly $r(z') \leq gsp(s, v') + w(v', z') \leq cost(p, v')$ since v is on the subpath of p from s to v'.

Then we show that if claim 2 is true for loop i, then claim 1 is true for loop $i + 1$. This is straightforward because after the loop i, for any $z \in S$, we have that $r(z) = \min_{p:s \to z} cost(p)$. Let u^* be the newly-added node in S. Therefore, at the beginning of loop $i+1$, for any node $u \notin S$, and any path p from s to u with last but one endpoint being $z \in S$, there are two possible cases: a) if $z \in S \setminus \{u^*\}$, then Claim 1 at loop i, we know that $r(u) \leq cost(p)$. b) if $z = u^*$, then since at loop i, $r(u)$ has been updated to be the minimum of $r(u)$ and $r(u^*) + w(u^*, u)$. Therefore, $r(u) \leq r(u^*) + w(u^*, u) \leq cost(p, v)$ for any $v \neq u$, since p passes u^* and $v \neq u^*$. On the other hand, for $v = u$, we also have that $r(u) \leq d(u) + n(u) \leq cost(p, u)$. Therefore, for any v in path p, we have that $r(u) \leq cost(p, v)$ and therefore, $r(u) \leq cost(p)$.

Analysis of Runtime: the algorithm clearly runs in n while loops. At each while loop, we need also access the smallest value in R and remove it, and update the values in R, where R is the min heap data structure that stores the $r(z)$ values. The total number of updates is $O(m)$. Therefore, using a Fibonacci heap to maintain R, the runtim will be $O(m + n \log n)$. The analysis is similar to Dijstra's algorithm [FT87, Ski90]. If we compute for k different flows, the runtime becomes $O(k \cdot (m + n \log n))$ since we have to compute k times.

A.1.4 Dependency Routing Analysis We first introduce some notation for the following LP formulation. For a dependency routing with k_i tasks, f_i^j represents the amount of i^{th} flow that has

passed j^{th} task, $p_i^j(v)$ represents the amount of j^{th} task processed at node v for the i^{th} flow. We have a total of T tasks and each $C^t(v)$ represents the capacity for t^{th} task at node v. We have a mapping function $m_i(j) = t$ that maps the index of j^{th} task to one of the T tasks.

MAXIMIZE

$$\sum_{i=1}^{|D|} \sum_{e \in \delta^+(s_i)} f_i^0(e) \tag{A.7a}$$

SUBJECT TO

$$\sum_{e \in \delta^-(v)} \sum_{0 \le j \le k} f_i^j(e) = \sum_{e \in \delta^+(v)} \sum_{0 \le j \le k_i} f_i^j(e) \qquad \forall i \in [|D|], \forall v \in V \setminus \{s_i, t_i\} \tag{A.7b}$$

$$p_i^j(v) = \sum_{j \le l \le k_i} \Big(\sum_{e \in \delta^-(v)} f_i^l(e) - \sum_{e \in \delta^+(v)} f_i^l(e) \Big) \qquad \forall i \in [|D|], \forall v \in V, \forall j \in [0, k_i] \tag{A.7c}$$

$$\sum_{i=1}^{[D]} \sum_{0 \le j \le k_i} f_i^j(e) \le B(e) \qquad \forall e \in E \tag{A.7d}$$

$$\sum_{i=1}^{|D|} p_i^j(v) \le C^t(v) \qquad \forall j \in [0, k_i], \forall i \in [|D|], m_i(j) = t, \forall v \in V \tag{A.7e}$$

$$f_i^j(e) = 0 \qquad \forall i \in [|D|], \forall j \in [1, k_i], e \in \delta^+(v), \forall v \in V \setminus \{s_i\} \tag{A.7f}$$

$$f_i^j(e) = 0 \qquad \forall i \in [|D|], \forall j \in [0, k_i - 1], e \in \delta^-(v), \forall v \in V \setminus \{t_i\} \tag{A.7g}$$

$$f_i^j(e), p_i^j(v) \ge 0 \qquad \forall i \in [|D|], \forall j \in [0, k_i], \forall e \in E, \forall v \in V \tag{A.7h}$$

Proof. To simplify the proof, again, we only focus on a single flow with (s, t); we can easily extend to the multi-commodity case. We show that we can convert the LP solution to a routing and allocation solution, where we know the paths and for each path we know how to allocate the processing among nodes for different tasks.

Round One: We can first focus on the last process (e.g., the k^{th} process during a k staged processing), and the way to compute the walks for the last process is very analogous to that of a single type of processing problem, we can think of that there are two types of flows, $\sum_{0 \le j \le k-1} f^j$ as $\hat{f}^1$ and f^k as $\hat{f}^2$. It is not hard to see that the above LP is exactly the same as the previous problem once we replace with $\hat{f}^1$ and $\hat{f}^2$. Save the result in the format of $(\pi, p_\pi^k(v))$.

Round Two: Once we get routing and processing pattern from *Round One*, we can compute the processing and routing pattern from the LP output and *Round One* output. The allocation is some variant of single type processing problem in A1. At each node, we know $p^{k-1}(v) = \sum_{e \in \delta^-(v)} f^{k-1}(e) - \sum_{e \in \delta^+(v)} f^{k-1}(e) + \sum_{e \in \delta^-(v)} f^k(e) - \sum_{e \in \delta^+(v)} f^k(e)$ with f^k values computed. We again group the flows in two types, $\sum_{0 \le j \le k-2} f^j$ as $\hat{f}^1$ and $f^k + f^{k-1}$ as $\hat{f}^2$. We pick a node with $p^{k-1}(v) > 0$, and pick one upstream and one downstream link at that node from the *Round One* result with different $\frac{\hat{f}^1}{\hat{f}^1 + \hat{f}^2}$ values, allocate processing at node v, based on flow size on the path, and the available flow processing at node v. Allocate until there is no node v with $p^{k-1}(v) > 0$. Note at each step, we may have a portion of flow from *Round One* path π with node processing. Since we preserve the flow pattern as in *Round One*, we consolidate the result as the format of $(\pi, p_\pi^k(v), p_\pi^{k-1}(v))$, and preserve the number of paths from step one.

...

Round K: We recursively compute the flow and processing allocation for all k stages and get the final routing and processing pattern. Note every step only relies on the previous step in the algorithm. Save the result as the format of $(\pi, p_\pi^k(v), p_\pi^{k-1}(v) \dots p_\pi^1(v))$.

A.2 Middlebox Node Purchase Optimization

A.2.1 Directed Maximization Algorithm

The algorithm here proceeds similarly to that in subsubsection 3.1.1. The LPs we use are the natural maximization variant of those used for the minimization problem, with the added restriction that we only use a 1/2 fraction of the budget. It is easy to see that this additional restriction does not reduce the objective value of the optimal LP solution by more than an 1/2-fraction. We also assume (without loss of generality) that no vertex has cost greater than the budget. The LPs are formulated as follows:

Walk-based formulation:

$$\text{MAXIMIZE} \sum_{i=1}^{|D|} \sum_{\pi \in P} p_{i,\pi}$$

SUBJECT TO

$$\sum_{v \in V} c_v x_v \leq k/2$$

$$x_v \leq 1 \qquad \forall v \in V$$

$$p_{i,\pi} = \sum_{v \in \pi} p_{i,\pi}^v \qquad \forall i \in [|D|], \pi \in P$$

$$\sum_{\pi \in P} p_{i,\pi} \geq R_i \qquad \forall i \in [|D|]$$

$$\sum_{i=1}^{|D|} \sum_{\substack{\pi \in P \\ \pi \ni e}} p_{i,\pi} \leq B(e) \qquad \forall e \in E$$

$$\sum_{i=1}^{|D|} \sum_{\pi \in P} p_{i,\pi}^v \leq C(v) x_v \qquad \forall v \in V$$

$$\sum_{i=1}^{|D|} \sum_{\substack{\pi \in P \\ \pi \ni e}} p_{i,\pi}^v \leq B(e) x_v \qquad \forall e \in E, v \in V$$

$$\sum_{\pi \in P} p_{i,\pi}^v \leq R_i x_v \qquad \forall i \in [|D|], v \in V,$$

$$p_{i,\pi}^v \geq 0 \qquad \forall i \in [|D|], \pi \in P, v \in \pi$$

$$0 \leq x_v \leq 1 \qquad \forall v \in V$$

Edge-based formulation:

$$\text{MAXIMIZE} \sum_{v \in V} \sum_{i=1}^{|D|} \sum_{e \in \delta^-(v)} f_i^{2,v}(e)$$

SUBJECT TO

$$\sum_{v \in V} c_v x_v \leq k/2$$

$$\sum_{e \in \delta^-(u)} f_i^{j,v}(e) = \sum_{e \in \delta^+(u)} f_i^{j,v}(e) \qquad \forall i \in [|D|], j \in \{1,2\}, v \in V, \forall u \in V \setminus \{s_i, t_i, v\}$$

$$\sum_{e \in \delta^-(v)} f_i^{2,v}(e) = \sum_{e \in \delta^+(v)} f_i^{1,v}(e) \qquad \forall i \in [|D|], v \in V,$$

$$\sum_{v \in V} \sum_{e \in \delta^+(s_i)} f_i^{2,v}(e) \geq R_i \qquad \forall i \in [|D|]$$

$$\sum_{i=1}^{|D|} \sum_{v \in V} (f_i^{1,v}(e) + f_i^{2,v}(e)) \leq B(e) \qquad \forall e \in E$$

$$\sum_{i=1}^{|D|} \sum_{e \in \delta^-(v)} f_i^{2,v}(e) \leq C(v) x_v \qquad \forall v \in V$$

$$\sum_{i=1}^{|D|} (f_i^{1,v}(e) + f_i^{2,v}(e)) \leq B(e) x_v \qquad \forall e \in E, v \in V$$

$$\sum_{e \in \delta^+(s_i)} f_i^{2,v}(e) \leq R_i x_v \qquad \forall i \in [|D|], v \in V$$

$$f_i^{2,v}(e) = 0 \qquad \forall i \in [|D|], v \in V, e \in \delta^-(s_i)$$

$$f_i^{1,v}(e) = 0 \qquad \forall i \in [|D|], v \in V, e \in \delta^+(t_i)$$

$$p_i^{1,v}(e), p_i^{2,v}(e), x_v \geq 0 \qquad \forall i \in [|D|], v \in V, e \in E$$

$$0 \leq x_v \leq 1 \qquad \forall v \in V$$

If purchasing a single vertex allows us to route a $1/(2 \ln n)$ fraction of the objective value of the above LP, we purchase only this vertex. Otherwise, we can remove the potential for processing at each vertex v with $c_v \geq k/\ln n$ and re-solve the LP to get a solution with objective value at least half as large

as before. Thus, from now on we can assume that no c_v exceeds $k/\ln n$ and therefore that the optimal LP solution puts support on at least a $1/\ln n$ fraction of the x_vs (at a cost of 2 in our approximation factor). We will call the objective value of this modified linear program $\mathrm{OPT}_{\mathrm{LP}'}$.

Again, we pick the vertices on which to install processing capacity on by randomized rounding: each vertex v is picked with probability x_v. If x_v is picked, then all flows processed by v are rounded so that $\hat{F}_i^{j,v}(e) = f_i^{j,v}(e)/(4x_v \ln n)$ for all $i \in [|D|], j \in \{1,2\}, e \in E$. If v is not picked, then all flows processed by v are set to zero, i.e. $\hat{F}_i^{j,v}(e) = 0$.

By design, $E[\hat{F}_i^{j,v}(e)] = f_i^{j,v}(e)/(4\ln n)$ and thus the total amount of flow processed, P, satisfies $E[P] = E\left[\sum_{v\in V}\sum_{i=1}^{|D|}\sum_{e\in\delta^-(v)}\hat{F}_i^{2,v}(e)\right] = \mathrm{OPT}_{\mathrm{LP}'}/(4\ln n)$. In the solution produced by the rounding algorithm, the total flow through edge e is $\sum_{v\in V}\sum_{i=1}^{|D|}((\hat{F}_i^{1,v}(e) + \hat{F}_i^{2,v}(e))$. This sum of random variables is $\hat{B}(e) = B(e)/(4\ln n)$ in expectation. Letting $g(e)$ denote the flow along edge e, standard bounds give

Lemma A.5.

$$Pr\left[g(e) \geq (4\lg n)\cdot\hat{B}(e)\right] \leq e^{-4\ln n} = n^{-4} \qquad \forall e \in E \tag{A.10}$$

$$Pr\left[P \leq (1/4)\cdot(1/(4\lg n)\cdot\mathrm{OPT}_{\mathrm{LP}'})\right] \leq e^{-4\ln n} = n^{-4} \qquad \forall e \in E \tag{A.11}$$

so by the union bound, with probability higher than $1 - 1/n$ every edge is assigned $\leq B(e)$ total flow and the amount of flow processed and routed is within a $1/16\ln n$ factor of $\mathrm{OPT}_{\mathrm{LP}'}$.

Finally, by Markov's inequality, the original budget constraint is satisfied with probability at least $1/2$. Combining this with lemma A.5, the algorithm fails with probability at most $1/2 + 1/n$. Repeating the algorithm $O(\log n)$ times and taking the best feasible solution therefore provides an $\Omega(1/\log n)$ approximation with probability at least $1 - 1/\mathrm{poly}(n)$. This can be summarized in the following result:

Theorem A.1. *For directed* Budgeted Middlebox Node Purchase*, there is a polynomial-time randomized algorithm producing an* $\Omega(1/\log(n))$ *approximation.*

We can also apply this algorithm to undirected instances by adding additional constraints the as we did in subsubsection 3.1.1, with the analysis carrying through as before. Thus, we attain the following:

Theorem A.2. *For undirected* Budgeted Middlebox Node Purchase*, there is a polynomial-time randomized algorithm producing an* $\Omega(1/\log(n))$ *approximation.*

A.2.2 Undirected Maximization Algorithm

We now show that the undirected Budgeted Middlebox Node Purchase admits a constant-factor approximation algorithm when restricted to a single source s. Let $\mathrm{OPT}(G,k)$ denote the value of the optimal solution to an instance with graph G and budget k. Our algorithm works by splitting the problem into both a *processing step* and a *routing step*. The algorithm begins by reserving a $1/2$ fraction of each edge for use in the processing step and the remaining $1/2$ fraction for use in the routing step. Calling the reserved-capacity graphs G_{proc} and G_{route}, respectively, the algorithm proceeds as follows:

Processing step A well known fact in capacitated network design is that the maximum amount of flow routable (sans processing) from a set $S \subseteq V$ of source vertices to a single sink forms a monotone, submodular function in S [CKLN13]. Although this problem is usually defined in the context of sources that can produce an arbitrary amount of flow (should the network support it), we can bottleneck each source s_i into producing at most some c_i units of flow by replacing it with a pair of vertices connected by a capacity c_i edge, without changing the submodularity of the routable flow function, $f_G(S)$. For the purpose of this lemma, redefining s as our "sink" and the set P of processing nodes as our source set S, we immediately attain that the function $f_G(P)$ is submodular, where $P \subset V$ is the set of nodes purchased for processing.

Let H be a copy of G_{proc} with all edge capacities halved. Because f_H is a submodular function, the problem of using our budget to purchase a set $P \subseteq V$ of processing nodes so to maximize $f_H(P)$ is simply an instance of a monotone, submodular maximization subject to knapsack constraints. Such problems are known to admit simple $(1 - 1/e)$-approximation algorithms [Svi04]. Let $P(H, k)$ be the optimal solution to this *processable flow problem* on H with budget k and $\text{ALG}_1(H, K)$ denote the value of the solution found by our algorithm. Because $P(H, k)$ is an upper bound on $\text{OPT}(H, k)$ (indeed, the former is simply an instance of the former without the need to account for post-processing routing), the $(1 - 1/e)$ approximation we get has value at least equal to $(1 - 1/e)$ times the value of $\text{OPT}(H, k)$. In particular

$$\begin{aligned}
\text{ALG}_1(H, k) &\geq (1 - 1/e)P(H, k) \\
&\geq (1 - 1/e)\text{OPT}(H, k) \\
&\geq (1 - 1/e)(1/2)\text{OPT}(G_{\text{proc}}, k) \\
&\geq (1 - 1/e)(1/2)(1/2)\text{OPT}(G, k) && = (1 - 1/e)/4 \cdot \text{OPT}(G, k)
\end{aligned}$$

Further, because our solution only uses at most half of the capacity of any edge in G_{proc}, we can use the remaining, unused half of the capacities to route all flow we managed to process back to s.

Routing Step All flow residing in s after the end of the processing step is already processed, all of it can be routed directly to the sinks using the 1/2 fraction of edge capacities we reserved for G_{route}. Because multiplying all edge capacities by 1/2 reduces the amount of routable flow by the same (multiplicative) amount, we can route at least $(1/2)\min(\text{ALG}_1(H, k), \text{MaxFlow}_G(s, t))$ units of the processed flow from s to t. As $\text{MaxFlow}_G(s, t)$ is a (trivial) upper bound on $\text{OPT}(G, k)$, this means we can route at least $(1/2)(1 - 1/e)/4\text{OPT}(G, k)$ units of the processed flow from s to the sinks, giving a $(1 - 1/e)/8 > .078$ approximation algorithm.

Thus, we get the following theorem:

Theorem A.3. *For undirected* Budgeted Middlebox Node Purchase *with a single source, there is a deterministic polynomial time algorithm that produces a solution that can route at least* $(1-1/e)/8 \approx .078$ *times the optimal objective solution.*

A.2.3 Undirected Minimization Hardness

We now show an approximation preserving reduction from Min Vertex Cover to undirected Min Middlebox Node Purchase, proving that the latter problem is **UGC**-hard to approximate within a factor of $2-\epsilon$ for any $\epsilon > 0$ [KR08], and **NP**-hard to approximate within a factor of 1.36 [DS05].

The construction is simple. Given a VERTEX COVER instance with graph $G = (V, E)$, we create an identical graph with each vertex v demanding one unit of processed flow from each of its neighbors, and each edge's capacity is 2. Further, each vertex has n units of processing potential, at a cost of 1. Because the total demand equals the sum of all edge capacities, each unit of flow sent must use exactly one unit of edge capacity, i.e. all flow paths have length exactly one. Thus, the set of solutions exactly corresponds to vertex covers, with one unit of flow going each way across each edge, from source to sink and either to or from its point of processing. The unit costs ensure that the objective value equals the number of vertices picked, and thus that the optimal solution to this undirected MIN MIDDLEBOX NODE PURCHASE instance equals that of the original MIN VERTEX COVER. The conclusion, summarized below, follows.

THEOREM A.4. *Approximating undirected* MIN MIDDLEBOX NODE PURCHASE *is at least as hard as approximating* MIN VERTEX COVER. *In particular, it is* **NP**-hard *to approximate within a factor of* 1.36 *and* **UGC**-hard *to approximate within a factor of* $2 - \epsilon$, *for any* $\epsilon > 0$.

A.2.4 Hardness of Directed Budgeted Middlebox Node Purchase

We now prove that directed BUDGETED MIDDLEBOX NODE PURCHASE is NP-hard to approximate to a factor better than $(1 - 1/e + \epsilon)$. To show this, we reduce from MAX K-COVER, which is known to have the same hardness result [Fei98].

Given a MAX K-COVER instance with set system $\mathcal{S}$ and universe of elements $\mathcal{U}$, we create one vertex v_S for each $S \in \mathcal{S}$ and one vertex w_u for each $u \in \mathcal{U}$. Further, we create one source vertex s and one sink vertex t, where t demands $|\mathcal{U}|$ units of processed flow from s. We add one capacity-n arc from s to each v_S, and one capacity-1 arc from each w_u to t. We then add a capacity-1 arc from each v_S to w_u whenever $S \ni u$. Finally, we give each v_S vertex n units of processing capacity at a cost of 1 each. The budget for the instance is k – the same as the budget for the MAX-K-COVER instance. A diagram of the reduction is given in Figure 2.

When flow is routed maximally, each w_u contributes 1 unit of flow to the total $s - t$ flow if and only if it has a neighbor v_S that was chosen to be active. Otherwise, this vertex does not help contribute towards the $s - t$ flow. Thus, this instance of directed BUDGETED MIDDLEBOX NODE PURCHASE can be seen as the problem of buying k different v_S vertices so to maximize the number of distinct w_u vertices to which they are adjacent. Thus, there is a direct one-to-one mapping between solutions to our constructed instance and the initial MAX K-COVER instance, and the values of the solutions are conserved by the mapping. Therefore, we have an approximation-preserving reduction between the two problems, and directed BUDGETED MIDDLEBOX NODE PURCHASE acquires the known $(1 - 1/e + \epsilon)$ inapproximability of MAX K-COVER.

A.2.5 Undirected Maximization Hardness

We show that for some fixed $\epsilon_0 > 0$, the undirected version of BUDGETED MIDDLEBOX NODE PURCHASE is **NP**-hard to approximate within a factor of $1 - \epsilon$, implying that the the problem does not admit a PTAS unless $\mathbf{P} = \mathbf{NP}$. We make no attempt to maximize the value ϵ_0.

We show this hardness by reducing from MAX BISECTION on degree-3 graphs, shown to be hard to approximate within a factor of .997 in [BK99][1]. Let $G = (V, E)$ be the input to the degree-3 MAX

[1] To be precise, this paper shows the aforementioned hardness for MAX CUT. A simple approximation preserving reduction from

BISECTION instance. For each $v_i \in V$, create two vertices, u_i and w_i, joined by an edge with capacity 3. We also add a capacity-1 edge between u_i and u_j whenever v_i and v_j are adjacent in G. Each w_i vertex demands 3 units of flow from every u_j (including when $i = j$). Further, every u_i vertex can be given $3|V|$ units of processing capacity (or, equivalently, ∞ units) at a cost of 1, and the instance's budget is set to $|V|/2$.

The intuition behind the construction is as follows. With a budget of $|V|/2$, we can purchase exactly half of the u_i vertices (and all budget is used up without loss of generality); our bisection will be between the purchased u_is and the unpurchased ones. Let b be the number of edges in any such bisection. Each w_i adjacent to a purchased u_i can have 3 units of its demand satisfied by flow originating from and processed by u_i, and the only edge connecting w_i to the rest of the graph ensures w_i can never receive more than 3 units of flow regardless. Thus, such w_is are maximally satisfied, and contribute $3|V|/2$ units to our objective value. The remaining w_is must have their processed flow routed to them via edge via the b capacity-1 edges in the bisection (and, indeed, every edge in the bisection will carry 1 unit of flow when routed optimally, as witnessed by the solution where each unprocessed u_i receives flow on each cut-edge and routes it directly to w_i), so the total amount of demand satisfied by the w_i adjacent to unpurchased vertices is exactly b, so the objective value of a solution with b edges in the bisection is exactly $3|V|/2+b$.

Letting b_{OPT} denote the number of edges cut by the optimal bisection. It is a well-known fact that $b_{\text{OPT}} \geq |E|/2 = 3|V|/4$. By the theorem of [BK99] it is **NP**-hard to distinguish instances with $3|V|/2 + b_{\text{OPT}}$ units of satisfiable demand from those with only $3|V|/2 + (1 - .003)b_{\text{OPT}}$, giving an inapproximability ratio of

$$\begin{aligned}
\frac{3|V|/2 + (1-.003)b_{\text{OPT}}}{3|V|/2 + b_{\text{OPT}}} &= 1 - \frac{.003 b_{\text{OPT}}}{3|V|/2 + b_{\text{OPT}}} \\
&= 1 - \frac{.003}{3|V|/(2b_{\text{OPT}}) + 1} \\
&\leq 1 - \frac{.003}{3|V|/(2 \cdot 3|V|/4) + 1} \\
&= 1 - \frac{.003}{2+1} \\
&= .999
\end{aligned}$$

This calculation is summarized in the following result:

THEOREM A.5. *It is* **NP**-hard *to approximate undirected* BUDGETED MIDDLEBOX NODE PURCHASE *to within a factor better than* .999.

A.3 Hardness of Dependency Min Middlebox Node Purchase We can generalize the MIN MIDDLEBOX NODE PURCHASE problem to incorporate dependency routing as described in Section 2.4. We show that the dependency version of MIN MIDDLEBOX NODE PURCHASE is LABEL COVER-Hard, and thus is unlikely to admit any polylogarithmic approximation algorithm.

THEOREM A.6. *For every $\epsilon > 0$, there is no polynomial-time algorithm approximating single-commodity dependency* MIN MIDDLEBOX NODE PURCHASE *to within an $O(2^{\log^{(1-\epsilon)} n})$ factor unless $NP \subseteq$* **DTIME**$(n^{\text{polylog}\, n})$.

$\overline{\text{MAX CUT}}$ to MAX BISECTION can be derived by looking at maximum cuts of the graph formed by 2 disjoint copies of the MAX CUT instance graph.

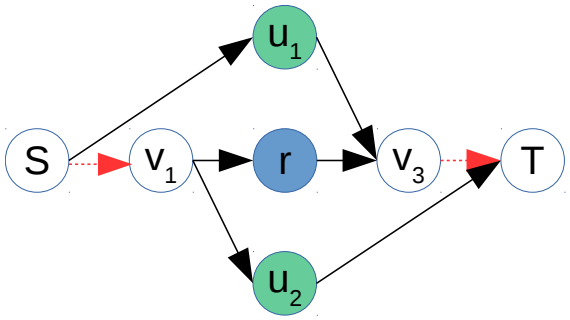

Figure 3: Example graph where vertex purchasing is not submodular. White vertices have no processing potential, colored vertices have 1 potential unit of processing. Solid black edges have capacity 2 while dashed red edges have capacity 1. If the only purchased vertex is r, no single additional purchase can increase the routable flow at all, yet buying both u_1 and u_2 simultaneously increases it to 2.

The input to Min Rep consists of a bipartite graph $G = (A; B, E)$, as well as a partitioning of A and B into equal-sized subsets, $\{A_1, A_2, \cdots, A_{m_1}\}$ and $\{B_1, B_2, \cdots, B_{m_2}\}$, respectively. The goal is to select as few vertices $C \subseteq A \cup B$ as possible while ensuring that for every pair of partitions A_i and B_j with an induced edge between them, there is at least one edge in the graph induced on $(A_i \cup B_j) \cap C$.

For each $i \in 1, 2, \cdots m_1$, we make one source vertex a_i, and similarly make a sink vertex b_j for each $j \in 1, 2, \cdots m_2$. For each vertex $v_j \in A_i$, we make a node u_j with infinite processing capacity and add an infinite capacity edge from a_i to u_j. Similarly, for each $v_j \in B_i$, we make a w_j with infinite processing capacity and add an infinite-capacity edge from it to b_i. The v_i and w_j are then connected as they were in the Min Rep instance with capacity 1 edges (directed from v_i to w_j). For each A_i adjacent to a B_j in the Min Rep instance, we add one unit of demand from a_i to b_j, with the corresponding chain $(\delta^+(a_i), \delta^-(b_j)$ (δ^+ and δ^- denoting the set of outgoing and incoming edges, respectively).

We can only route flow from a_i to b_j if there is an edge between vertices in their neighborhoods and, if there is an edge, we can always route this flow by buying its endpoints. By setting the cost of each v_i and w_j to 1, the optimum solution to our instance has the same cost as the optimum of the original Min Rep instance, proving our sought result for the problem with multiple sources. To get the same hardness for a single source, we add a single super-source and super-sink with demand equal to the number of connected A_i, B_j pairs, with the super-source adjacent via an out-edge to each a_i and the super-sink adjacent to each b_j with an in edge. We give each a_i and b_j node infinite processing capacity at a cost of 0. For each unit of demand corresponding to an A_i, B_j edge, the chain is now $(\{a_i\}, \delta^-(a_i), \delta^+(b_j), \{b_j\})$. Theorem A.6 follows.

A.3.1 Non-submodularity of Budgeted Middlebox Node Purchase As stated in subsubsection 3.2.1, it is tempting to state that the amount of routable flow is submodular in the collection of purchased vertices, which would imply that simple greedy algorithms can give a $(1-1/e)$ approximation algorithm to the problem. As shown in Figure 3, this natural supposition happens to be false, as there exist configurations where the natural greedy algorithm gets stuck at an infeasible solution.