Filtering Undesirable Flows in Networks

Gleb Polevoy*[1], Stojan Trajanovski**[1,2], Paola Grosso[1], and Cees de Laat[1]

[1] University of Amsterdam
[2] Philips Research

Abstract. We study the problem of fully mitigating the effects of denial of service by filtering the minimum necessary set of the undesirable flows. First, we model this problem and then we concentrate on a subproblem where every good flow has a bottleneck. We prove that unless P = NP, this subproblem is inapproximable within factor $2^{\log^{1-1/\log\log^c(n)}(n)}$, for $n = |E| + |GF|$ and any $c < 0.5$. We provide a $b(k+1)$-factor polynomial approximation, where k bounds the number of the desirable flows that a desirable flow intersects, and b bounds the number of the undesirable flows that can intersect a desirable one at a given edge. Our algorithm uses the local ratio technique.

Keywords: flow, filter, MMSA, set cover, approximation, local ratio algorithm

1 Introduction

Denial of Service (DoS) and Distributed DoS [18] are widespread network attacks. These attacks negatively impact functionality, especially when the system needs to be quick (soft real time, for example) [17]. Consequently, fighting the problem is highly important [22]. Filtering the attacking flows [16] is one of the main ways to fight the problem. Filtering is also preferred among practitioners and network operators, rather than, for example, the more complicated and expensive link addition or removal. If we properly select a flow we want to filter, filtering always succeeds, but the required efforts depend on the filtered flow. For example, defining in the firewall which flows to filter is sometimes simple (say, filter all the UDP), but sometimes contrived (e.g., no simple pattern of what to filter exists) [11]. Unlike admission control, here we do not decide whether to allow a connection, but rather how to handle an existing one.

A similar problem is having less important but not malicious flows in the network. We then remove the less important flows to allow the more important ones to optimally utilize the network, and we want to incur the least possible cost from removing the less important flows. This pertains to both computer networks and transportation networks. In computer networks, streaming video to prioritized customers may be contractually binding, forcing flows to the other

* G.Polevoy@uva.nl

** The research was started while S.T. was with the University of Amsterdam. He is now with Philips Research.

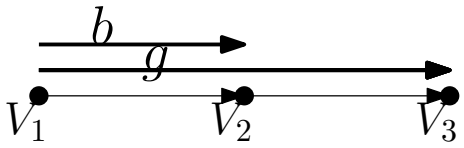

Fig. 1. Consider the network represented by the path graph with the 3 vertices V_1, V_2, V_3. We have a bad flow, b, and a good one, denoted by g.

customers to give space to the prioritizes ones. In transportation, for example, a less important freight connection may be removed in favor of the more crucial ones [19].

We define a flow as a single path from the source to the sink and consider a system with some desirable (call them *good*) and undesirable (name them *bad*) flows. Undesirable flows can either model malicious flows or, alternatively, legitimate but dispensable flows. In particular, we model DoS as a set of bad flows that take up the available bandwidth. We study filtering as a coping method, possible within traffic engineering [1, 2]. If we filter some bad flows, we can allocate the good flows more *value* because of the freed capacity. We aim to maximally increase the good flows, while spending the minimum necessary filtering effort, or losing the least from filtering the less important flows. Indeed, in the context of DoS, minimizing the filtering effort is practically important: Koning et al. [16] show that the filtering cost can have a significant effect on the overall effectiveness of the response. Therefore, we should not simply filter everything: Example 1 demonstrates that filtering all the bad flows can take arbitrarily more effort than the minimum effort necessary to maximally increase the good flows. In order to autonomously decide which flows to filter, as suggested in [16], we need an algorithm to find which bad flows to filter. In order to cope with large instances in real time, the algorithm has to be polynomial.

Example 1. In Figure 1, assume the capacity of edge (V_1, V_2) is $2c$ and the capacity of (V_2, V_3) is c. Let the original flows have the value of c each: $v(b) = v(g) = c$, and let $w(b)$ be positive. Because of the saturated edge (V_2, V_3), filtering b would not allow increasing g. Therefore, the optimal set to filter is $\emptyset$ and it costs zero, infinitely smaller than filtering anything.

We assume we know which flows are good and which are bad, either because we know all the flows, they are all good, and we decide which are dispensable and which are not, or, when malicious flows exist, we can identify them by frequent access trials from the same IP group.

Take a look at the following example of using an algorithm that decides which bad flows to filter.

Example 2. In a system where the flows belong to the same organization and carry equally important traffic, assume that our intrusion detection system discovers a DoS attack, and determines which flows are attacking. We need to respond quickly and efficiently. Having determined which flows are desirable,

which are attacking and how large the flows are, we first estimate how hard filtering each attacking flow would be. Now, we run our algorithm to obtain an (approximately) easiest set of attacking flows to filter, such that the desirable flows will be able to fully utilize the system.

To pose the problem, we first model it in Section 2 and define k as the largest number of good flows that any good flow intersects in a network. We will also need b, defined as the largest number of bad flows that flow through an edge where a good flow also flows. The measures k and b will be used later on. Section 2.1 provides a short primer to the *local ratio* technique, which we employ to approximate the subproblem where every good flow has a bottleneck. At the outset, we prove in Section 3 that for any $k \geq 0$, the problem is NP-hard, using a reduction from Set Cover. Even when no bad edges intersect one another, we prove by reduction from Minimum-Monotone-Satisfying-Assignment that the problem is not even approximable within $2^{\log^{1-1/\log\log^c(n)}(n)}$, for $n =$ |edges in the network| + |desirable flows| and any $c < 0.5$, unless P $\neq NP$. In Section 4 we provide a polynomial approximation of the problem, with the tight approximation ratio of $b(k+1)$. The algorithm uses the *local ratio* technique [4, 3, 5]. We conclude and suggest further research directions in Section 5.

Our approximation can facilitate remedying the distributed DoS and similar congestion scenarios.

1.1 Related Work

We are not aware of any theoretical flow filtering approximations, but there is literature studying related flow problems. First, we aim to maximize the desirable flows, each flow being on a given path, while the famous max-flow – min-cut problem [8, Chapter 26] aims to maximize the total flow from source to sink, without predefined paths. There exist many famous generalizations of max-flow, such as maximum circulation [15, Chapter 7] and multi-commodity flow [10]. Regarding the allowed actions, we study filtering, thereby completing the studies of network design: edge addition [14], edge deletion [21, 13], etc. In particular, deleting edges that can disconnect all the flows from a source to a sink is a famous problem, and Menger's theorem [7, Chapter 3.2] characterizes the minimum number of edges one has to remove in order to disconnect the source from the sink. Of course, finding a minimum cut mentioned above and disconnecting it is an optimal algorithm for this problem.

2 Model

We model the flow network as a directed graph $G = (N, E)$ with (edge) capacities $c\colon E \to \mathbb{R}_+$. A flow f from node a to node z in this network is a path from source a to sink z, each of which edges carries the value of the flow. Formally, $f = (v(f), P(f))$, where $v(f) \in \mathbb{R}_+$ is the value of the flow and $P(f)$ is the set of the edges of the path that the flow takes from a to z. Flow in this paper are not

splittable, which meanings that a flow takes a single path. This can also model a splitting flow as separate flows with partially overlapping paths. All the flows together fulfill the capacity constraint, meaning that for every edge $e \in E$, all the passing flows together are bounded in their values by the capacity of the edge, i.e.

$$\sum_{f:e\in P(f)} v(f) \leq c(e).$$

Let us define the basic problem we are considering.

Definition 1 *The* Bad Flow Filtering problem (BFF) *receives the input* $(G = (N, E), c\colon E \to \mathbb{R}_+, F, GF, BF, w\colon BF \to \mathbb{R}_+)$. *Here,* $G = (N, E)$ *is a capacitated network with capacities* c *and flows* $F = \{f_i\}$*, where some flows, denoted* $GF = \{g_i\} \subseteq F$*, are marked as good (desirable), and the rest, denoted* $BF = \{b_i\} \triangleq F \setminus GF$*, are bad (undesirable). The values of the good flows are not given in the input. Every bad flow* f *is endowed with a weight* $w(f)$*, designating how hard filtering that flow would be, or how important the bad flow is, if bad flows model dispensable but legitimate flows.*
A solution S *is a subset of bad flows to filter.*
A feasible solution *is a solution such that the good flows can be allocated values such that the total value of the good flows is the maximum possible (i.e. equal to the total value that can be allocated if all the bad flows are removed).*
We aim *to find a feasible solution with the minimum total weight. Intuitively, we aim to optimize the total good flow while investing the minimum filtering effort, or while losing the minimum of the less important flows, depending on what bad flows model.*

BFF is monotonic with respect to inclusion, in the sense that filtering more bad flows after having filtered a feasible solution preserves feasibility.

We now define a constrained version of BFF, such that the algorithm can always provably approximate the solution. We need to avoid a situation when decreasing the value of a good flow can allow multiple good flows increase. Intuitively, we achieve this by always having a bottleneck that connects all the good flows that intersect one another, so that decreasing one of them will never be used multiple times to increase others. Formally,

Definition 2 *For any good flow* $g \in GF$*, define a* bottleneck of g *be a set of edges* $S(g) \subseteq P(G)$ *such that every other good flow* g' *that intersects* g *contains all these edges, and for every edge* i *where* g *intersects another good flow and for every solution* $BF' \subseteq BF$*, there exists an edge* $e \in S(g)$ *such that* $c(e) - \sum_{b\in BF\setminus BF':e\in b} v(b) \leq c(i) - \sum_{b'\in BF\setminus BF':i\in b'} v(b')$.
A BFF problem where every good flow has a bottleneck is called a Bottleneck-BFF (BBFF).

We prove BBFF, and therefore, BFF, is hard and approximate BBFF. Let us now present two cases that fall under BBFF.

Common Narrow Link. If every good flow g and all the flows that it intersects pass through an edge of a much smaller capacity than the other edges on the path of this flow, then this edge constitutes a bottleneck of g. This happens in practice when the flows pass through a physically common link.

Uniform Intersection. Intuitively, we require that a set of intersecting good flows all intersect each other at the same edges.

Definition 3 *The* Uniform Intersection Bad Flow Filtering problem (UIBFF) *is a restriction of BFF where every $g \in GF$ has a set of edges on its path, $E(g) \subseteq P(g)$, such that every other good flow g' that intersects g shares with g exactly the edges of $E(g)$, i.e. $P(g) \cap P(g') = E(g)$.*

Since the defined $E(g)$ is a bottleneck of g, UIBFF is a subproblem of BBFF. This uniformity can happen, for instance, if the intersecting flows share a source or a destination, and intersect only near those nodes.

We now define the parameters which we will use to express the approximation ratio of our algorithm.

Definition 4 *Given an instance of BFF, let k be the largest possible number of good flows that a given good flow intersects. Formally,*

$$k \triangleq \max\left\{\left|\{g' \in GF \setminus \{g\} : P(g') \cap P(g) \neq \emptyset\}\right| : g \in G\right\}.$$

Definition 5 *For a BFF instance, let b be the largest number of bad flows that intersect a good flow at any given edge. Formally,*

$$b \triangleq \max\left\{\left|\{f \in BF : e \in P(f)\}\right| : g \in G, e \in P(g)\right\}.$$

2.1 Local Ratio Approximation

A typical local ratio r-approximation algorithm for minimization [4, 5] is easier to formulate recursively, though practical implementations are usually iterative. It works by manipulating the weights as follows.

1. If a trivial solution (often, the empty set) is feasible, return it.
2. Otherwise, if zero weight elements exist, we remove them, solve the problem without them and add them back afterwards.
3. Otherwise, decompose the weight function $w = w_1 + w_2$ such that every feasible solution would be an r-approximation with respect to w_1. We call such a w_1 weight function *r-effective* and finding it is the main challenge. Then, recursively invoke the algorithm with w_2. The returned feasible solution is an r-approximation w.r.t. w_2, by induction. Since it also is an r-approximation w.r.t. w_1, by the way we decomposed w, the following theorem implies that this solution is also an r-approximation w.r.t. w, as required.

Theorem 1 (Local Ratio Theorem [4]). *Let us have a feasible set $D \subseteq \mathbb{R}^n$.*[3] *Assume we have weight vectors $w = w_1 + w_2$ and that a feasible solution $x \in D$ is r-approximate w.r.t. w_1 and w.r.t. w_2. Then, x is r-approximate w.r.t. w as well.*

We also require from w_1 that at least one element will have the zero weight in $w - w_1$, so that the instance will shrink at the next invocation. Finding a suitable r-effective w_1 for a small r is the crux of the method, requiring an insight about all the feasible solutions of the problem.

3 Hardness

We prove that the decision version of BBFF is NP-complete, and even NP-hard to be approximated within $2^{\log^{1-1/\log\log^c(n)}(n)}$, for $n = |E| + |GF|$, thereby motivating the need to seek an approximation instead of an exact solution.

We first prove that the problem is NP-hard not merely to optimize exactly, but even to approximate.

Theorem 2. *UIBFF (and, therefore, BBFF) is not approximable within $2^{\log^{1-1/\log\log^c(n)}(n)}$, for $n = |E| + |GF|$ and any $c < 0.5$, unless $P \neq NP$. This holds even if no bad edges intersect one another.*

Proof. We prove the hardness of approximation by reducing the Minimum-Monotone-Satisfying-Assignment of depth 3 (MMSA_3) problem to UIBFF. Let us remind the definition of MMSA_3 [9].

Definition 6 *The input of the MMSA_3 problem is a monotone (with no negative literals) Boolean formula, which is a conjunction (AND) of disjunctions (OR), every such disjunction being a disjunction of conjunctions. The goal is finding a satisfying assignment that minimizes the number of variables that are assigned* 1.

An example of an MMSA_3 is $((x_1 \text{ AND } x_3 \text{ AND } x_5) \text{ OR } (x_2 \text{ AND } x_3)) \text{ AND } ((x_2 \text{ AND } x_4 \text{ AND } x_5 \text{ AND } x_6) \text{ OR } (x_1))$.

Given an instance of MMSA_3, our reduction defines the following UIBFF. For each variable x in conjunction c, which is, in turn, a part of disjunction d, define edge $e_{x,c,d}$ of capacity 1. Define also a bad flow b_x of value 1 and weight 1 that has all the edges $\{e_{x,c,d} | x \text{ appears in } c, \text{ a part of } d\}$ on its path, and no others of the above edges. The rest of the edges, if any, are arbitrary and unique for each bad flow. For each conjunction c of variables, which is a part of disjunction d, define a good flow $g_{c,d}$ that flows through the edges $\{e_{x,c,d} | x \text{ appears in } c, \text{ a part of } d\}$ and perhaps arbitrary other edges of capacity 1 without any bad flows that pass through them. For each disjunction d of conjunctions, let the respectively defined

[3] In our case, these are the incidence vectors of the bad flows that, if filtered, would allow assigning the good flows the maximum possible total value. Thus, we have $D \subseteq \mathbb{N}^n$.

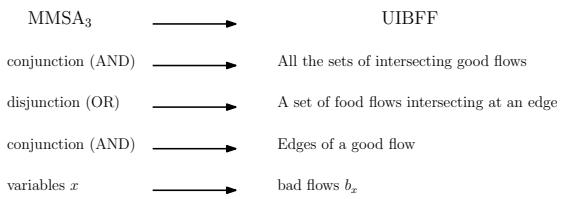

Fig. 2. The approximation preserving reduction from MMSA_3 to UIBFF.

good flows $\{g_{c,d}|c \text{ appears in } d\}$ intersect at a single edge e_d where no bad flows pass. Let these be the only intersections of good flows among themselves. The reduction is illustrated in Figure 2.

First, since we define the BFF instance such that every good flow intersects all the other good flows at a single edge, it is indeed a UIBFF. To prove validity of the reduction, note that the MMSA_3 instance is satisfied if and only if the main conjunction holds, which holds if and only if at least one conjunction in every disjunction holds. The last statement is equivalent to at least one good flow in all the intersecting sets of good flows has all its edges free and can thus be given the value of 1. Therefore, feasibility is transferred by the reduction. Since the costs are equivalent as well, the reduction preserves approximation.

Since MMSA_3 is not approximable within $2^{\log^{1-1/\log\log^c(n)}(n)}$, for any $c < 0.5$, as shown in [9], we infer that UIBFF is not approximable within the same ratio, when $n = |E| + |GF|$. This is because the size of the MMSA_3 formula translates to $|E| + |GF|$. ■

We now prove that the decision version is indeed NP-complete. We first define the decision version of BBFF.

Definition 7 *The Decision-BBFF receives (x, l) in its input, where x is an instance of BBFF and l is a natural number. The question is whether there exists a feasible solution for BBFF with weight at most l.*

We finally prove that

Theorem 3. *Decision-BBFF is NP-complete, for any $k \geq 0$.*

Proof. First, we show that Decision-BBFF is in NP. Indeed, for a candidate solution S, filter all the flows there and maximize the good ones in the remaining network. The maximization can be done polynomially by solving the Linear Program (LP):

$$\max \sum_{i \in GF} x_i \tag{1}$$

such that

$$\forall e \in E : \sum_{i \in GF : e \in P(i)} x_i + \sum_{b \in BF \setminus S : e \in P(i)} v(b) \leq c(e) \tag{2}$$

$$\forall i \in GF : x_i \geq 0 \tag{3}$$

Remark 1. This LP does not require an LP solver, since we assume that any good flow has a bottleneck set of edges which always is the constraint to any intersection. This property means that it is not important how the good flows divide a common edge, since their sum will remain the same. Consequently, we can effectively reduce the capacity taken by the bad flows in $O(|BF|\,|E|)$ time, storing the effective capacities for each edge, and subsequently maximize each good flow one after another, by checking the bottlenecks of each good flow in $O(|E|)$ time and storing the effectively remaining capacities, amounting to the total time of $O((|GF| + |BF|)\,|E|) = O(|F|\,|E|)$.

By comparing the maximum of this LP with the maximum when all the bad flows are filtered, i.e. when $S = BF$, we check whether S is feasible. If it is, then it constitutes a certificate if and only if $w(S) \leq l$. Therefore, the problem belongs to NP.

To prove the NP-hardness, notice that our reduction from MMSA_3 is also a Karp reduction for the decision versions, and since Decision-MMSA_3 is NP-hard [9], so is Decision-BBFF. However, to claim the NP-hardness for any $k \geq 0$, we now present a reduction from the decision version of Set Cover (SC) [12]. Let us remind the definition of SC.

Definition 8 *SC receives as input a universe U, a collection of its subsets $\{S_1, S_2, \ldots, S_m\}$, such that $\cup_{i=1}^{m} S_i = U$ and a natural number d. A* solution *is a subset of $\{S_1, S_2, \ldots, S_m\}$, while a solution $C \subset \{S_1, S_2, \ldots, S_m\}$ is called* feasible *or a* cover *if $\cup_{S \in C} S = U$. The question is whether there exists a cover C such that $|C| \leq d$.*[4]

Our reduction takes an input of SC, which is $U, S_1, S_2, \ldots, S_m, d$, and returns the following instance of Decision-BBFF (even Decision-UIBFF). First, we define a bad flow b_i for each set S_i, with $v(b_i) = 1$. For each element $x \in U$, let g_x be a good flow with a path consisting of the two edges: $e_x^{(1)}$ and $e_x^{(2)}$. We set the $c(e_x^{(1)})$ to be the number of sets S_i that contain x and we set $c(e_x^{(2)}) = 1$. For every S_i that contains x, let b_i intersect g_x at $e_x^{(1)}$. Besides these intersections, no more flow intersections take place. The weight of every b_i is defined to be 1.[5] The parameter l of the constructed Decision-UIBFF instance is defined to be d. The reduction is exemplified in Figure 3.

We now prove that this reduction is valid. Indeed, for every element $x \in U$, any set cover includes at least one set that contains x, say S_i and this is transformed to filtering the corresponding bad flow b_i. This allows the good flow g_x to increase till it uses up all the capacity it can, i.e. 1. Note, that if more covering sets are selected as well, then the corresponding bad flows are filtered as well, but the value of flow g_x remains 1 because $c(e_x^{(2)}) = 1$; one filtering is enough to maximize $v(g_x)$, guaranteeing feasibility to Decision UIBFF.

[4] We use the unweighted set cover, where each set has the same importance, because it has the same hardness results as the weighted version.

[5] Did we reduce the weighted SC, we would define it to be the weight of the respective set.

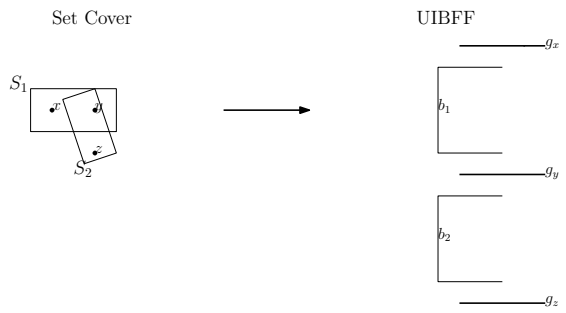

Fig. 3. The Karp reduction from Decision-SC to Decision-UIBFF.

In the other direction, a feasible solution to Decision UIBFF maximizes the sum of good flows. This requires filtering at least one bad flow from those that intersect every g_x, which means that a feasible solution to the constructed instance of Decision-UIBFF is obtained from a set cover. This completes the proof of the NP-hardness. ■

4 Approximation

Consider the local ratio approximation Algorithm 1 for BBFF. We explain it now in the terms of Section 2.1. Line 1 finds the maximum total good flow that is available at the current invocation. The recursion basis appears at line 2 and line 3 removes the zero weight bad flows. We abuse notation by writing w in the recursive call, while we actually mean the restriction of w to $BF \setminus BF_0$. The central scene of the algorithm occurs at line 4. There, we pick a good flow that would benefit from filtering bad flows at line 4a and construct the set of all the bad flows we may need to filter at line 4b. This serves us to decompose the weights at line 4c.

When choosing the flows in H at line 4a and 4b, we take the minimum possible flow each time. The idea is to select all the possible good flows that can increase, to cover all the possibilities, and a smaller flow has more chances to increase.

We now prove that in polynomial time, this algorithm returns a feasible solution approximating the optimum within $b(k+1)$. This means, for example, that if any good flow intersects at most 1 another good flow ($k = 1$), and at most one bad flow contains a given edge of a good flow ($b = 1$), then the algorithm approximates the optimal solution within $1 \cdot (1 + 1) = 2$. And in case the good flows do not intersect one another ($k = 0$), the algorithm is optimal.

Another interesting particular case is the result of the reduction of Set Cover to BBFF from Theorem 3. In the outcome of the reduction, b is the maximum number of sets that can include a given element, and $k = 0$. Therefore, Algorithm 1, acting like the Algorithm 15.2 from [20], approximates set cover within the $b(0 + 1) = b$, which is the maximum number of sets that can include an

ALGORITHM 1: MinFilter$(G = (N, E), c, F, GF, BF, w)$

1. Solve `LP` Eq. (1)–Eq. (3) and let $(x_i)_{i \in GF}$ be the obtained result (the current maximum). Note that these flow values are not necessarily unique, since we can sometimes change several intersecting good flows one on the other's expense, preserving the sum.
2. **If** no good flow in $(x_i)_{i \in GF}$ can increase by filtering the bad flows that intersect it (without changing other flows), **return** $\emptyset$.
3. **Else, if** there exist bad flows with zero weight BF_0,
   (a) $S' \leftarrow$ **MinFilter** $(G = (N, E), c, F \setminus BF_0, GF, BF \setminus BF_0, w)$.
   (b) **Return** $S \leftarrow S' \cup BF_0$.
4. **Else,**
   (a) Pick any $g \in GF$ that can be increased (without changing other flows) if we filter the bad flows that intersect it.
   This should be done by taking the minimum $v(g)$ that can be in a maximum total good flow. (Maximize the good flows that intersect g on g's expense.)
   (b) Consider all the other good flows $g_1, g_2, \ldots, g_p$ (by the definition of k, $p \leq k$) that intersect g and would grow after filtering some bad flows, if we take the minimum possible $v(g_i)$ in a maximum total good flow. Let H be the set of the considered good flows, i.e. $H \triangleq \{g, g_1, g_2, \ldots, g_p\}$ and let $D(H)$ be the set of their respective saturated edges, chosen one from a flow (may choose the same saturated edge from several good flows, so $|D(H)| \leq |H|$). Denote all the bad flows that contain edge(s) from $D(H)$ as $B(D(H))$.
   (c) Let $\delta > 0$ be the minimum total weight in $B(D(H))$,
   i.e. $\min_{b \in B(D(H))} \{w(b)\}$. Define the weight function on the bad flows:

$$w_1 \triangleq \begin{cases} \delta & \textbf{if } b \in B(D(H)), \\ 0 & \textbf{otherwise.} \end{cases}$$

   .
   (d) **Return MinFilter** $(G = (N, E), c, F, GF, BF, w - w_1)$.

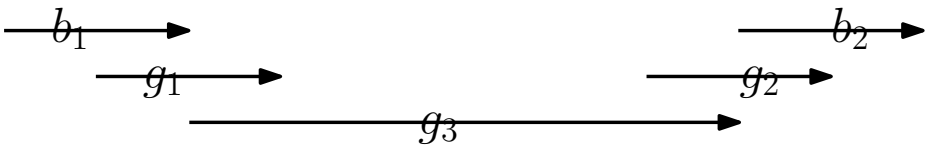

Fig. 4. Flow g_1 can grow if we filter b_1 and if g_3 decreases. Since decreasing g_3 can also help increasing g_2, if b_2 is filtered, the total good flow will increase.

element. Of course, the general BBFF is much harder than SC, being not easier than MMSA_3, as we show in Theorem 2. Approximating the general BBFF constitutes our main contribution.

First, we make a crucial observation, which guarantees that the algorithm always finds a required good flow g. This property requires the restriction of every good flow to have a bottleneck, introduced in Definition 2.

Observation 1 *A solution for an BBFF instance is infeasible if and only if in any maximal allocation of good flows for it there exists a good flow that can increase if we filter some bad flows that intersect it, without changing other flows.*

Proof. If a good flow can increase, the solution is infeasible by definition.

In the other direction, let S be an infeasible solution. This means that any allocation of good flows can increase if we filter some more bad flows. The only option where "no good flow exists that would increase if we filtered some bad flows that intersect it" needs good flows that can grow only at the expense of other good flows, like, for example, in Figure 4. However, since any good flow in BBFF has a bottleneck, where all the intersecting good flows pass, (unlike shown in Figure 4), increasing good flows at the expense of others would never increase the total good flow. ■

As in any correctness proof for a local ratio algorithm, we show that the weight function w_1 is fully $b(k+1)$-effective, meaning that any feasible solution S is a $b(k+1)$ approximation to the optimum.

Lemma 1. *Let S be any feasible solution to the instance of the problem at some invocation of Algorithm 1. Then, $w_1(S) \leq b(k+1) \cdot w_1(S^*)$, where S^* is an optimal solution at that invocation.*

Proof. Any feasible solution will either allow g to grow by filtering at least one bad flow that contains its chosen saturated edge, or it will allow at least one of the good flows that intersect it to grow by filtering at least one of the bad flows that contain their respectively chosen saturated edges. Therefore, with respect to w_1, any feasible solution will cost at least δ. On the other hand, any solution costs at most $b(k+1) \cdot \delta$. Therefore, any feasible solution costs at most $b(k+1)$ times the minimum cost. ■

We are finally set to prove the correctness and the approximation ratio of the algorithm.

Theorem 4. *Algorithm 1 always returns a feasible solution that approximates the optimal solution within the ratio of* $b(k+1)$.

Proof. We prove by induction on our recursive algorithm.

In the basis (line 2), the good flow is optimal and therefore, the empty set of bad flows is feasible. Since the empty set weighs zero, it is also optimal.

At a non-final stage, we need to prove that line 3 and line 4 both return feasible $b(k+1)$-approximation. In the case of line 3, S' is a feasible $b(k+1)$-approximation for the instance after removing BF_0, by induction on the algorithm. Now, a feasible solution for the instance after removing the flows in BF_0 remains feasible w.r.t. the original instance if we add the removed flows to the solution. Second, the approximation ratio keeps holding, since the optimum stays the same after this operation, and the solution cost remains the same as well.

Having said that, let us show that line 4 returns a feasible $b(k+1)$-approximation. First, the recursive invocation at line 4d returns a feasible solution and a $b(k+1)$-approximation with respect to the weight function $w - w_1$, by the induction hypothesis; call this solution $\hat{S}$. Set $\hat{S}$ is also a $b(k+1)$-approximation with respect to w_1, by Lemma 1. The Local Ratio Theorem 1 implies that $\hat{S}$ is also a $b(k+1)$-approximation with respect to the sum of the weight functions, i.e. $(w - w_1) + w_1 = w$. This completes the proof. ■

We finally remark that

Remark 2. The algorithm terminates in time $O(|BF|\,(l(S) + |E|\,|GF|\,|F|))$, where $l(S)$ is the time taken to solve the `LP` that corresponds to instance S. As we explain in Remark 1, $l(S) = O(|F|\,|E|)$, implying the total running time of $O(|E|\,|BF|\,|GF|\,|F|)$.

Proof. The algorithm performs $O(|BF|)$ iterations, since at least one bad flow gets filtered at each invocation of line 4.

At each invocation of line 3, we filter the zero-weight bad flows and then add them back in $|BF|$ time.

At each invocation of line 4, we solve the `LP` in $l(S)$ time. Next, we go over all the good flows, checking for each good flow g in $O(|P(g)|\,|F|) = O(|E|\,|F|)$ time whether filtering bad flows can help increasing this flow, by passing through all the edges of the path of the flow and checking whether the the good flows can be increased if no bad ones existed. If yes, we construct the sets H and $D(H)$ in $O(|E|\,|GF|\,|F|)$, construct $B(D(H))$ and define the weight function w_1 in $O(|E|\,|BF|)$. This takes together $O(|E|\,|GF|\,|F| + l(S))$. Finally, we make the recursive invocation.

Summing up the above time bounds and multiplying by $|BF|$, we obtain $O(|BF|\,(l(S) + |E|\,|GF|\,|F|))$. ■

Finally, we prove that the approximation ratio $b(k+1)$ is tight for our algorithm, even on a UIBFF. To this end, we employ the following example, partially inspired by Example 15.4 from [20].

Example 3. The general idea is to reduce Example 15.4 from [20] for Set Cover by our reduction from Theorem 3 to UIBFF, while also translating the weights of sets to be the weights of the corresponding bad flows. This would produce a tight

example, but the parameter k would be zero. To allow for any k, we consider several instances of such an example and make them intersect in a specific way.

Concretely, consider the following UIBFF instance, depicted in Figure 5. We have the good flows $g_1, g_2, \ldots, g_{n+1}$, each g_i with the path of two edges $e_i^{(1)}$ and $e_i^{(2)}$, where $c(e_i^{(1)}) = 2$, for $i = 1, \ldots, n-1$, $c(e_n^{(1)}) = n$, and $c(e_{n+1}^{(1)}) = 1$. For all $i = 1, \ldots, n+1$, we have $c(e_i^{(2)}) = 1$. We have the bad flows $b_{\{1,n\}}, b_{\{2,n\}}, \ldots, b_{\{n-1,n\}}$ with weight 1 each and the bad flow $b_{\{1,2,\ldots,n+1\}}$ with weight $1+\epsilon$ for a positive ϵ. Every bad flow has the value of 1. A bad flow b_S intersects the good flows corresponding to the elements of S at their respective first edges, and, for now, no more intersections exist.

Next, consider $m+1$ copies of the constructed problem instance. Let the distinct copies intersect only at the edges $e_i^{(2)}$, for $i = 1, \ldots, n+1$, where all the copies intersect.

Algorithm 1 can choose at its first recursive invocation any good edge that can increase from filtering the bad ones. Assume it chooses g_n of one of the m copies. The weights of each of the bad flows in all the copies decrease by 1 and in the next invocation we remove all the bad edges from all the copies, besides the bad edges $b_{\{1,2,\ldots,n+1\}}$. Their weights will now also go to zero and in the following invocation the empty set becomes feasible. In the unwinding of the recursion, we will add all the bad flows to the solution, accruing the total weight of $m+1$ times $\underbrace{1+\ldots+1}_{n-1}+1+\epsilon = (m+1)(n+\epsilon)$. Now, the optimal solution is just $b_{\{1,2,\ldots,n+1\}}$ of one of the copies, because the intersections among the good flows let only values 1 in every intersecting set. Therefore, the optimal weight is $1+\epsilon$, and we can obtain an arbitrarily close to $n(m+1)$ ratio, for a sufficiently small ϵ. This is exactly $b(k+1) = n(m+1)$, demonstrating the tightness of $b(k+1)$.

5 Conclusion

Aiming to optimally mitigate DoS or unintended congestion, we study the BBFF problem of filtering the minimum number of undesirable (bad) flows so as to allow the desirable (good) flows to maximally utilize the network. We demonstrate that this practical problem is also very interesting theoretically. First, we reduce the MMSA_3 to BBFF while preserving approximation, proving that approximating within $2^{\log^{1-1/\log\log^c(n)}(n)}$, for $n = |\text{edges in the network}| + |\text{desirable flows}|$ and any $c < 0.5$ is NP-hard. We then provide a local ratio approximation algorithm for BBFF.

An interesting variation of the problem would be to assume that the flows are always allocated by a given protocol, for example, by the max-min fairness algorithm [6, Section 6.5.2]. This would render the problem of filtering non-monotonic with respect to inclusion, which would make many approximation techniques fail. Another point is that we are given a continuous ranking of the

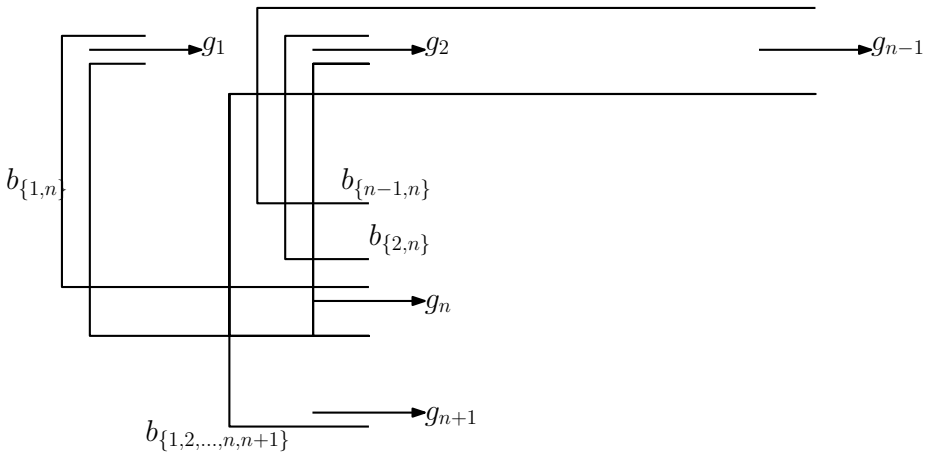

Fig. 5. The relevant part of the network. The bad flows are denoted by b with an index, while the good ones are denoted by g with an index. An optimal solution would be to filter $b_{\{1,2,\ldots,n+1\}}$, while our algorithm filters everyone.

bad flows by weight, but the distinction between the bad and the good is binary. Exploring other rankings would allow modeling other congestion domains.

To summarize, we have modeled an important NP-complete problem, proven it be not easier than MMSA_3 and approximated it.

Acknowledgments This research is funded by the Dutch Science Foundation project SARNET (grant no: CYBSEC.14.003 / 618.001.016)

References

1. Agarwal, S., Kodialam, M.S., Lakshman, T.V.: Traffic engineering in software defined networks. In: INFOCOM. pp. 2211–2219. IEEE (2013)
2. Akyildiz, I.F., Lee, A., Wang, P., Luo, M., Chou, W.: A roadmap for traffic engineering in sdn-openflow networks. Computer Networks 71, 1 – 30 (2014)
3. Bar-Noy, A., Bar-Yehuda, R., Freund, A., Naor, J., Shieber, B.: A unified approach to approximating resource allocation and scheduling. J. ACM 48(5), 1069–1090 (2001)
4. Bar-Yehuda, R., Even, S.: A local-ratio theorem for approximating the weighted vertex cover problem. North-Holland Mathematics Studies 109, 27 – 45 (1985)
5. Bar-Yehuda, R., Bendel, K., Freund, A., Rawitz, D.: Local ratio: A unified framework for approximation algorithms. in memoriam: Shimon even 1935-2004. ACM Comput. Surv. 36(4), 422–463 (Dec 2004)
6. Bertsekas, D.P., Gallager: Data networks. Prentice-Hall, Englewood Cliffs, New Jersey, 2 edn. (1992)
7. Bondy, J., Murty, U.: Graph Theory with Applications. North Holland (1976)
8. Cormen, T., Leiserson, C., Rivest, R., Stein, C.: Introduction to Algorithms, 3rd Edition:. MIT Press (2009)

9. Dinur, I., Safra, S.: On the hardness of approximating label-cover. Information Processing Letters 89(5), 247 – 254 (2004)
10. Even, S., Itai, A., Shamir, A.: On the complexity of time table and multi-commodity flow problems. In: Proceedings of the 16th Annual Symposium on Foundations of Computer Science. pp. 184–193. SFCS '75, IEEE Computer Society, Washington, DC, USA (1975)
11. Ferguson, P., Senie, D.: Network ingress filtering: Defeating denial of service attacks which employ ip source address spoofing (1998)
12. Garey, M.R., Johnson, D.S.: Computers and Intractability: A Guide to the Theory of NP-Completeness. W. H. Freeman & Co., New York, NY, USA (1979)
13. Italiano, G.F.: Finding paths and deleting edges in directed acyclic graphs. Information Processing Letters 28(1), 5 – 11 (1988)
14. Khuller, S., Thurimella, R.: Approximation algorithms for graph augmentation. Journal of Algorithms 14(2), 214 – 225 (1993)
15. Kleinberg, J., Tardos, E.: Algorithm Design. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA (2005)
16. Koning, R., de Graaff, B., de Laat, C., Meijer, R., Grosso, P.: Interactive analysis of sdn-driven defence against distributed denial of service attacks. In: 2016 IEEE NetSoft Conference and Workshops (NetSoft). pp. 483–488 (June 2016)
17. Mirkovic, J., Dietrich, S., Dittrich, D., Reiher, P.: Internet Denial of Service: Attack and Defense Mechanisms (Radia Perlman Computer Networking and Security). Prentice Hall PTR, Upper Saddle River, NJ, USA (2004)
18. Mirkovic, J., Reiher, P.: A taxonomy of ddos attack and ddos defense mechanisms. SIGCOMM Comput. Commun. Rev. 34(2), 39–53 (Apr 2004)
19. Rodrigue, J.P.: The Geography of Transport Systems. New York: Routledge, New York, 4th edn. (2017)
20. Vazirani, V.: Approximation Algorithms. Springer (2001)
21. Yannakakis, M.: Node-and edge-deletion np-complete problems. In: Proceedings of the Tenth Annual ACM Symposium on Theory of Computing. pp. 253–264. STOC '78, ACM, New York, NY, USA (1978)
22. Zargar, S.T., Joshi, J., Tipper, D.: A survey of defense mechanisms against distributed denial of service (ddos) flooding attacks. IEEE Communications Surveys Tutorials 15(4), 2046–2069 (Fourth 2013)