Network Interdiction Using Adversarial Traffic Flows

Citation: Fu, Xinzhe and Eytan Modiano. "Network Interdiction Using Adversarial Traffic Flows." IEEE INFOCOM 2019, Paris, April 29-May 2, 2019, IEEE © 2019 The Author(s)

As Published: 10.1109/INFOCOM.2019.8737475

Publisher: IEEE

Persistent URL: https://hdl.handle.net/1721.1/126285

Version: Author's final manuscript: final author's manuscript post peer review, without publisher's formatting or copy editing

Network Interdiction Using Adversarial Traffic Flows

Xinzhe Fu and Eytan Modiano
Lab for Information and Decision Systems, MIT

***Abstract*—Traditional network interdiction refers to the problem of an interdictor trying to reduce the throughput of network users by removing network edges. In this paper, we propose a new paradigm for network interdiction that models scenarios, such as stealth DoS attack, where the interdiction is performed through injecting adversarial traffic flows. Under this paradigm, we first study the deterministic flow interdiction problem, where the interdictor has perfect knowledge of the operation of network users. We show that the problem is highly inapproximable on general networks and is NP-hard even when the network is acyclic. We then propose an algorithm that achieves a logarithmic approximation ratio and quasi-polynomial time complexity for acyclic networks through harnessing the submodularity of the problem. Next, we investigate the robust flow interdiction problem, which adopts the robust optimization framework to capture the case where definitive knowledge of the operation of network users is not available. We design an approximation framework that integrates the aforementioned algorithm, yielding a quasi-polynomial time procedure with poly-logarithmic approximation ratio for the more challenging robust flow interdiction. Finally, we evaluate the performance of the proposed algorithms through simulations, showing that they can be efficiently implemented and yield near-optimal solutions.**

I. Introduction

Network interdiction, originally proposed in [1], [3] models the scenarios where a budget-constrained *interdictor* tries to limit the throughput available for *users* of a capacitated network by removing network edges. The throughput is given by the optimal value of a single-commodity max-flow problem and the goal of the interdictor is to compute an interdiction strategy that specifies which edges to remove in order to minimize the throughput, or maximize the throughput reduction, subject to the budget constraint. Since the problem is NP-hard even when the network has special topologies, previous works focus on designing approximation algorithms [3], [5], [6] or formulating integer programs and solving them using traditional optimization techniques (e.g. branch and bound) [1], [7]. Subsequent generalizations include extensions to the case where the throughput is given by multi-commodity max-flow problem [2] and allowing the interdictor to use mixed strategy that takes advantage of randomization [4], [8]. We refer the readers to [17] for a comprehensive survey.

As a generalization of the renowned max-flow min-cut theorem, network interdiction provides valuable insights to the robustness of networks. Projecting the interdictor to an adversarial position, network interdiction can characterize the impact of natural disasters on fiber-optic networks [9], evaluate the vulnerability of network infrastructures [10], and provide guidelines to the design of security-enhancing strategies for cyber-physical systems [11].

In this paper, we propose a new paradigm for network interdiction where the interdiction is performed through injecting adversarial traffic flow to the network in an intelligent way, encroaching the capacity of network links, thereby reducing the throughput of network users. It captures applications that have eluded the traditional network interdiction paradigm based on edge removals. One of the most prominent examples is the stealth denial of service (DoS) attack in communication networks, including wireless ad hoc networks [12], software defined networks [13] and cloud services [14]. The interdictor (attacker) injects low-rate data into the network that consumes network resources and compromises the capacity available to the users. In this paradigm, we model the network as a capacitated directed graph with n nodes, where the network users are sending flow on a set P of user paths and the interdictor aims to reduce the throughput of the users through sending adversarial flow from its source s to its destination t. Mirroring the situations in [12], [13], [15], we assume the interdictor to be low-rate and undetectable, which will be formally defined later. The interdiction strategy is defined as a probability distribution over the set of s-t flows with value less than the given budget, which resembles the mixed strategy in the game theory literature [4]. The throughput reduction achieved is equal to the difference between the network throughput before the interdiction, which is defined as the sum of initial flow values on paths in P and the network throughput after the interdiction, which is determined by the optimal value of a path-based max-flow problem on the residual network that subsumes both single and multi-commodity max-flows.

Under the proposed interdiction paradigm, we study two problems that differ in the availability of the knowledge on the operation of network users captured by the set of user paths P. The first, deterministic flow interdiction, assumes that the interdictor has perfect knowledge of P and seeks an interdiction strategy that maximizes the (expected) throughput reduction with respect to P. We show that there does not exist any polynomial time algorithm that approximates the problem on general networks within an $O(n^{1-\delta})$ factor for any $\delta > 0$ unless P = NP, and the problem is NP-hard even when the network is acyclic. Thus, we focus on designing efficient algorithms with good performance guarantees on acyclic networks. Specifically, utilizing the submodularity of the problem, we propose a recursive algorithm that is capable of achieving $O(\log n)$-approximation. The second problem, robust flow interdiction, deals with the situation where definitive knowledge of P is not available. In particular, we assume that

This work was supported by DTRA grants HDTRA1-13-1-0021 and HDTRA1-14-1-0058, and NSF grant CNS-1735463.

the set of user paths lies in some uncertainty set $\mathcal{U}$ that contains all possible candidates for P. The goal of the interdictor is to compute an interdiction strategy that maximizes the throughput reduction for the worst case in $\mathcal{U}$. As a generalization of its deterministic counterpart, robust flow interdiction inherits the computational complexity results and is more challenging to solve due to its inherent maximin objective. In this context, we design an approximation framework that integrates the algorithm for deterministic flow interdiction and yields a quasi-polynomial time procedure with a polylogarithmic approximation guarantee. Finally, we evaluate the performance of the proposed algorithms through simulations. The simulation results suggest that our algorithms compute solutions that are at least 70% of the optimal and are efficiently implementable.

The rest of the paper is organized as follows. We formally present our paradigm on acyclic networks in Section II. In Sections III and IV, we introduce formal definitions, show the computational complexity and describe our proposed algorithms for the two flow interdiction problems, respectively. We evaluate the performance of our algorithms through simulations in Section V. Section VI is devoted to the extension of our paradigm and the interdiction problems to general networks. We conclude the paper in Section VII. *Some of the proofs are omitted due to space constraints, and interested readers can refer to our technical report [26] for the details.*

II. Network Interdiction Paradigm

In this section, we first formalize our network interdiction paradigm, and then show two important structural properties of it. Note that currently, we focus on acyclic networks, and provide extensions to general networks in Section VI.

Consider a network represented as a directed acyclic graph $G(V, E)$ with vertex set V and edge set $E \subseteq V \times V$. Let $n = |V|$ be the number of nodes and $m = |E|$ be the number of edges. We assume G to be simple (with no multi-edges). Let C be an $|E|$-dimensional non-negative capacity vector with $C(e)$ indicating the capacity of edge e. We define $s, t \in V$ as the source and the destination of the interdictor, and assume without loss of generality that they are connected. An s-t flow is defined as an $|E|$-dimensional vector $\mathbf{f}$ that satisfies capacity constraints: $\forall e \in E, 0 \leq \mathbf{f}(e) \leq C(e)$ and flow conservation constraints: $\forall v \in V \backslash \{s,t\}, \sum_{(u,v)\in E} \mathbf{f}(u,v) = \sum_{(v,u)\in E} \mathbf{f}(v,u)$. We define $val(\mathbf{f}) = \sum_{(s,u)\in E} \mathbf{f}(s,u)$ to be the value of $\mathbf{f}$, i.e., the total flow out of the source.

The interdiction is performed by injecting flow from s to t. The interdictor has a flow budget γ that specifies the maximum value of flow that it can inject. In this paper, we are primarily concerned with low-rate interdictor, and thus assume that $\gamma \leq \min_{e\in E} C(e)$ and is bounded by some polynomial of n. Let $\mathcal{F}_{\leq\gamma}$ be the set of s-t flows $\mathbf{f}$ with $val(\mathbf{f}) \leq \gamma$. We allow the interdictor to use randomized flow injection, which is captured by the concept of interdiction strategy formally defined below.

Definition 1 (Interdiction Strategy). *An interdiction strategy w is a probability distribution $w : \mathcal{F}_{\leq\gamma} \mapsto [0, 1]$ such that $\sum_{\mathbf{f}\in\mathcal{F}_{\leq\gamma}} w(\mathbf{f}) = 1$*

The interdiction strategy bears resemblance to the mixed strategy in the game theory literature. It can be alternatively interpreted as injecting flows in a time sharing way. Furthermore, a deterministic flow injection $\mathbf{f}$ (similar to a pure strategy in game theory) can be represented as a strategy with $w(\mathbf{f}) = 1$.

Before the interdiction, the network users are sending flow on a set of user paths $P = \{p_1, p_2, \ldots, p_k\}$ in the network. Each path is a subset of edges and we use $e \in p_i$ to represent that edge e is on path p_i. The user paths may not share the same source and destination, and are not necessarily disjoint. Initially, the values of the flows on the paths are $\lambda_1, \lambda_2 \ldots, \lambda_k$ respectively, which satisfy capacity constraints: $\forall e \in E,\ \sum_{p_i\ni e} \lambda_i \leq C(e)$. The network throughput of the users before the interdiction is defined as $\sum_{i=1}^{k} \lambda_i$. Note that the involvement of the initial flows gives our paradigm the flexibility to capture the case where the users are not fully utilizing the paths before interdiction.

After the interdictor injects flow $\mathbf{f}$, the residual capacity of the edges becomes $\tilde{C}_{\mathbf{f}}$ such that $\tilde{C}_{\mathbf{f}}(e) = C(e) - \mathbf{f}(e)$ for all $e \in E$. The throughput of the users after interdiction is given by the optimal value of the (path-based) max-flow problem:

$$\text{maximize } \textstyle\sum_i \tilde{\lambda}_i \tag{1}$$

$$\textbf{s.t. } \textstyle\sum_{p_i\ni e} \tilde{\lambda}_i \leq \tilde{C}_{\mathbf{f}}(e), \quad \forall e \in E \tag{2}$$

$$0 \leq \tilde{\lambda}_i \leq \lambda_i, \quad \forall i \tag{3}$$

where constraints (2) are the capacity constraints after the interdiction and constraints (3) specify that the users will not actively push more flows on the paths after the interdiction, which can be attributed to the undetectability of the interdictor or that the users have no more flow to send. Let $T(\mathbf{f}, P)$ be the optimal value of (1). We define the throughput reduction achieved by injecting flow $\mathbf{f}$ as the difference between the throughput of the network before and after interdiction, i.e., $\Lambda(\mathbf{f}, P) = \sum_i \lambda_i - T(\mathbf{f}, P)$. Naturally, under an interdiction strategy w, the expected throughput reduction achieved by the interdictor is defined as $\Lambda(w, P) = \sum_{\mathbf{f}\in\mathcal{F}_{\leq\gamma}} w(\mathbf{f})\Lambda(\mathbf{f}, P)$.

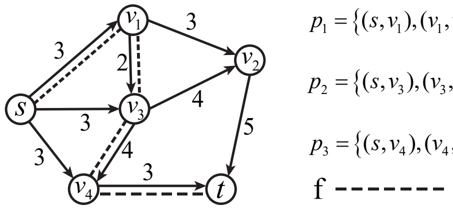
A. Structural Properties of the Paradigm

The proposed network interdiction paradigm has two important structural properties that will play a key role in the problems we study. The first property shows that if we want to maximize the throughput reduction, we can restrict our consideration to the set of s-t flows with value γ. It follows straightforwardly from the monotonicity of $\Lambda(\mathbf{f}, P)$ with respect to the value of $\mathbf{f}$ and that $\gamma \leq \min_e C(e)$.

Observation 1. *For any s-t flow $\mathbf{f}$ with $val(\mathbf{f}) < \gamma$, there exists a flow $\mathbf{f}'$ such that $val(\mathbf{f}') = \gamma$ and $\Lambda(\mathbf{f}', P) \geq \Lambda(\mathbf{f}, P)$ for all possible P.*

We denote $\mathcal{F}_\gamma$ to be the set of flows with value γ. We further define *single-path flows* as the s-t flows that have positive values on edges of one s-t path. The second property establishes the optimality of interdiction strategies taking positive value on only single-path flows in the maximization of Λ.

Proposition 1. *For any interdiction strategy w, there exists an interdiction strategy w' that is a probability distribution on*



v_1
v_2
v_3
v_4
s
t
3
3
2
3
4
3
4
3
5
$p_1 = \{(s,v_1),(v_1,v_2)\}$
$p_2 = \{(s,v_3),(v_3,v_2)\}$
$p_3 = \{(s,v_4),(v_4,t)\}$
f

Fig. 1. An example network of the flow interdiction problems.

the set of single-path flows such that $\Lambda(w',P) \geq \Lambda(w,P)$ for all possible P.

Proof. We prove the proposition through flow decomposition and linear programming duality. The proof is omitted due to space constraints. □

III. Deterministic Flow Interdiction

In this section, we study the deterministic flow interdiction problem. We first formally define the problem, then prove its computational complexity, and finally introduce our proposed approximation algorithm.

A. Problem Formulation

The deterministic flow interdiction deals with the case where the interdictor has full knowledge of P and seeks the interdiction strategy that causes the maximum expected throughput reduction.

Definition 2 (Deterministic Flow Interdiction)**.** *Given the set of user paths $P = \{p_1,\dots,p_k\}$ with initial flow values $\{\lambda_1,\dots,\lambda_k\}$, the deterministic flow interdiction problem seeks an interdiction strategy w that maximizes $\Lambda(w,P)$.*

Example: We give an example of the problem. Consider the network in Figure 1, where the capacities are labeled along the edges. The source and the destination of the interdictor are nodes s and t. The interdictor has budget $\gamma = 2$. The user paths $P = \{p_1,p_2,p_3\}$ all have an initial flow value of 3. Let $\mathbf{f}$ be the s-t flow such that $\mathbf{f}(s,v_1) = \mathbf{f}(v_1,v_3) = \mathbf{f}(v_3,v_4) = \mathbf{f}(v_4,t) = 2$. In this example, the interdiciton strategy w such that $w(\mathbf{f}) = 1$ is optimal with $\Lambda(w,P) = 4$.

B. Computational Complexity

Before establishing the computational complexity, we first show a structural property that is specific to the deterministic flow interdiction problem. Following from Proposition 1, there exists an interdiction strategy on the set of single-path flows that is optimal for the deterministic flow interdiction. We further extend this property, showing that there exists an optimal pure interdiction strategy.

Proposition 2. *For the deterministic flow interdiction problem, there exists an optimal (pure) interdiction strategy w such that $w(\mathbf{f}^*) = 1$ for some single-path flow $\mathbf{f}^*$.*

Proof. Building on proposition 1, let w be an optimal interdiction strategy that takes positive values only on single-path flows $\mathbf{q}_1,\dots,\mathbf{q}_r$. Let $\mathbf{q}^* \in \arg\max_i \Lambda(\mathbf{q}_i,P)$. Consider the pure strategy w' with $w'(\mathbf{q}^*) = 1$. It follows that $\Lambda(w',P) = \Lambda(\mathbf{q}^*,P) \geq \sum_i w(\mathbf{q}_i)\Lambda(\mathbf{q}_i,P) = \Lambda(w,P)$, which proves the existence of an optimal pure strategy. □

Based on Propositions 1 and 2, we prove the NP-hardness of the deterministic flow interdiction problems.

Proposition 3. *The deterministic flow interdiction problem is NP-hard.*

Proof. The proof is done by reduction from the 3-satisfiability problem, which is a classical NP-Complete problem [25]. We refer to [26] for the full proof. □

Remark: From the proof of Proposition 3, we have that even when the user paths are disjoint, the deterministic problem is still NP-hard.

C. Approximation Algorithm

Before presenting the algorithm, we extend some previous definitions. For any subset of edges $A \subseteq E$, imagine that the interdictor can interdict the edges in A by reducing their capacities by γ. We extend the definition of $\Lambda(\cdot,P)$ to A as $\Lambda(A,P) = \sum_i \lambda_i - T(A,P)$, where $T(A,P)$ is the optimal value of the maximization problem (1) with $\tilde{C}_A(e) = C(e) - \gamma \cdot \mathbb{1}_{\{e \in A\}}$. This provides an interpretation of $\Lambda(\cdot,P)$ as a set function on all subsets of E. Note that each single-path flow $\mathbf{f}$ can be equivalently represented as a set of edges $E_{\mathbf{f}}$ with $\mathbf{f}(e) = \gamma$ if and only if $e \in E_{\mathbf{f}}$. It follows that $\Lambda(\mathbf{f},P) = \Lambda(E_{\mathbf{f}},P)$, which links the definition of $\Lambda(\cdot,P)$ on single-path flows to that on sets of edges.

Our algorithm works on the optimization problem below.

$$\begin{aligned} \text{maximize} \quad & \Lambda(E_{\mathbf{f}},P) \\ \textbf{s.t.} \quad & E_{\mathbf{f}} \text{ forms an } s\text{-}t \text{ path.} \end{aligned} \tag{4}$$

Let $E_{\mathbf{f}^*}$ be the optimal solution to (4) and $\mathbf{f}^*$ be its corresponding single-path flow. By Proposition 1, the strategy w with $w(\mathbf{f}^*)$ is an optimal interdiction strategy, and $\Lambda(E_{\mathbf{f}^*},P) = \Lambda(w,P)$. Therefore, through approximating problem (4), our algorithm translates to an approximation to the deterministic flow interdiction problem. In the sequel, to better present the main idea of our algorithm, we first discuss the case where the user paths are edge-disjoint. After that, we generalize the results to the non-disjoint case.

1) Disjoint User Paths: When the user paths are edge-disjoint, for some interdicted edges $A \subseteq E$ and user paths $\{p_1,\dots,p_k\}$ with initial values $\{\lambda_1,\dots,\lambda_k\}$, the optimal solution to the max-flow problem (1) can be easily obtained as $\tilde{\lambda}_{iA} = \min\left(\lambda_i, \min_{e \in p_i} \tilde{C}_A(e)\right)$ for all i. It follows that the throughput reduction can be written as the sum of the throughput reduction on each paths, i.e., $\Lambda(A,P) = \sum_i (\lambda_i - \tilde{\lambda}_{iA})$. Based on this, we reason below that the set function $\Lambda(\cdot,P)$ has two important properties: *monotonicity and submodularity*.

Lemma 1. *Consider $\Lambda(\cdot,P) : 2^E \mapsto \mathbf{R}^*$ as a set function. Λ is:*

1) *Monotone: $\Lambda(A,P) \leq \Lambda(B,P)$ for all $A \subseteq B$;*
2) *Submodular: for all $A,B \subseteq E, e \in E$, if $A \subseteq B$, then $\Lambda(A \cup \{e\},P) - \Lambda(A,P) \geq \Lambda(B \cup \{e\},P) - \Lambda(B,P)$.*

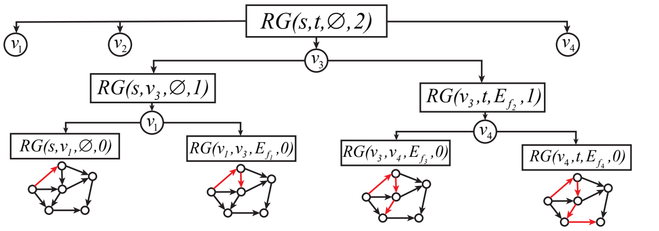


Fig. 2. Illustration of the Recursive Greedy algorithm (with some intermediate steps omitted) on the example of Figure 1, where $E_{\mathbf{f}_1}, \ldots, E_{\mathbf{f}_4}$ are used to denote the sub-paths constructed during the recursion for ease of notation.

Proof. The monotonicity is easily seen from the definition of Λ. The proof of submodularity is also straightforward. Note that for each user path i,

$$\begin{aligned}\lambda_i - \tilde{\lambda}_{iA} &= \lambda_i - \min(\lambda_i, \min_{e \in p_i}\{C(e) - \mathbb{1}_{\{e \in A\}}\}) \\ &= \lambda_i + \max(-\lambda_i, \max_{e \in p_i}\{-C(e) + \mathbb{1}_{\{e \in A\}}\})\end{aligned}$$

Since constant and linear functions are submodular, and the maximum of a set of submodular functions is also submodular, it follows that $\Lambda(A, P) = \sum_i (\lambda_i - \tilde{\lambda}_{iA})$ is submodular. □

Intuitively, an s-t path with large throughput reduction should have many intersections with different user paths. This intuition, combined with the monotonicity and submodularity of Λ, may suggest an efficient greedy approach to the optimization problem (4) that iteratively selects the edge with the maximum marginal gain with respect to Λ while sharing some s-t path with the edges that have already been selected. However, this is not the whole picture since such greedy selection might get stuck in some short s-t path and lose the chance of further including the edges that contribute to the throughput reduction. The latter aspect indicates the necessity of extensive search over the set of all s-t paths, but the number of s-t paths grows exponentially with n. Therefore, an algorithm with good performance guarantee and low time complexity must strike a balance between greedy optimization that harnesses the properties of Λ, and extensive search that avoids prematurely committing to some short path. The algorithm we propose, named as the Recursive Greedy algorithm, achieves such balance. It is based on the idea of [18]. The details of the algorithm are presented in **Algorithm 1** . In the description and analysis of the algorithm, $\Lambda_X(A, P) = \Lambda(A \cup X, P) - \Lambda(X, P)$ for $X, A \subseteq E$ represents the marginal gain of set A with respect to X. We use $\log$ to denote the logarithm with base two. For two nodes $u_1, u_2 \in V$, the shortest u_1-u_2 path is defined as the u_1-u_2 path with the smallest number of edges.

The recursive function RG lies at the heart of the Recursive Greedy algorithm. RG takes four parameters: source u_1, destination u_2, constructed subpath X and recursion depth i. It constructs a path from u_1 to u_2 that has a large value of $\Lambda_X(\cdot, P)$ by recursively searching for a sequence of good anchors and greedily concatenating the sub-paths between anchors. The base case of the recursion is when the depth i reaches zero, then RG returns the shortest path between u_1 and u_2 if there exists one (step 2). Otherwise, it goes over all the nodes v in V (step 8), using v as an anchor to divide the search into two parts. For each v, it first calls a sub-procedure to search for sub-path from u_1 to v that maximizes $\Lambda_X(\cdot, P)$, with i decremented by 1 (step 9). After the first sub-procedure returns $E_{\mathbf{f}_1}$, it calls a second sub-procedure for sub-paths from v to u_2 (step 10). Note that the second sub-procedure is performed on the basis of the result of the first one, which reflects the greedy aspect of the algorithm. The two sub-paths concatenated serve as the u_1-u_2 path that RG obtains for anchor v. Finally, RG returns the path that maximizes $\Lambda_X(\cdot, P)$ over the ones that it has examined over all anchors (steps 11, 12 and 13).

The Recursive Greedy algorithm starts by invoking $RG(s, t, \emptyset, I)$ with I as the initial recursion depth. In the following, we show that the algorithm achieves a desirable performance guarantee as long as I is greater than certain threshold. An illustration of the algorithm with $I = 2$ on the previous example is shown in Figure 2. The optimal solution is the path with anchors v_1, v_3, v_4.

Algorithm 1 The Recursive Greedy Algorithm

```
Input: Network graph $G(V, E)$, user paths $P = \{p_1, \ldots, p_2\}$
    with initial flow values $\{f_1, \ldots, f_k\}$
Output: The optimal $s$-$t$ path $E_\mathbf{f}$
 1: Run: $RG(s, t, \emptyset, I)$
    The Recursive Function $RG(u_1, u_2, X, i)$:
 2: $E_\mathbf{f}$ := shortest $u_1$-$u_2$ path.
 3: if $E_\mathbf{f}$ does not exist then
 4:     return Infeasible
 5: if $i = 0$ then
 6:     return $E_\mathbf{f}$
 7: $r := \Lambda_X(E_\mathbf{f}, P)$.
 8: for all $v \in V$ do
 9:     $E_{\mathbf{f}_1} := RG(u_1, v, X, i-1)$.
10:     $E_{\mathbf{f}_2} := RG(v, u_2, X \cup E_{\mathbf{f}_1}, i-1)$.
11:     if $\Lambda_X(E_{\mathbf{f}_1} \cup E_{\mathbf{f}_2}, P) > r$ then
12:         $r := \Lambda_X(E_{\mathbf{f}_1} \cup E_{\mathbf{f}_2}, P)$, $E_\mathbf{f} := E_{\mathbf{f}_1} \cup E_{\mathbf{f}_2}$.
13: return $E_\mathbf{f}$
```

Theorem 1. *If $I \geq \lceil \log d \rceil$, the Recursive Greedy algorithm returns an s-t path $E_\mathbf{f}$ with $\Lambda(E_\mathbf{f}, P) \geq \frac{1}{\lceil \log d \rceil + 1} \Lambda(E_{\mathbf{f}^*}, P)$, where d is the length of $E_{\mathbf{f}^*}$.*

Proof. We prove a more general claim, that for all $u_1, u_2 \in V$, $X \subseteq E$, if $I \geq \lceil \log d \rceil$, the procedure $RG(u_1, u_2, X, I)$ returns an u_1-u_2 path $E_\mathbf{f}$ with $\Lambda_X(E_\mathbf{f}, P) \geq \frac{1}{\lceil \log d \rceil + 1} \Lambda_X(E_{\mathbf{f}^*}, P)$, where $E_{\mathbf{f}^*}$ is the u_1-u_2 path that maximizes $\Lambda(\cdot, P)$ and d is the length of $E_{\mathbf{f}^*}$. The theorem follows from the claim by setting $u_1 = s$, $u_2 = t$ and $X = \emptyset$.

Let the nodes on the path $E_{\mathbf{f}^*}$ be $\{u_1 = v_0, \ldots, v_d = u_2\}$. The proof is done by induction on d. First, for the base step, when $d = 1$, it means that there exists an edge between u_1 and u_2, which must be the shortest u_1-u_2 path. Obviously the procedure examines this path at step 2, and the claim follows. Next, suppose the claim holds for $d \leq l$. When $d = l + 1$, $I \geq 1$. Let $v^* = v_{\lceil \frac{d}{2} \rceil}$ and $E_{\mathbf{f}_1^*}, E_{\mathbf{f}_2^*}$ be the subpaths of $E_{\mathbf{f}^*}$ from u_1 to v^* and v^* to t, respectively. When RG uses v^*

as an anchor, it first invokes $RG(u_1, v^*, X, I-1)$ that returns $E_{\mathbf{f}_1}$ and then invokes $RG(v^*, u_2, X \cup E_{\mathbf{f}_1}, I-1)$ that returns $E_{\mathbf{f}_2}$. Let $E'_{\mathbf{f}} = E_{\mathbf{f}_1} \cup E_{\mathbf{f}_2}$. Our goal is to show that

$$\Lambda_X(E'_{\mathbf{f}}, P) \geq \frac{1}{\lceil \log d \rceil + 1} \Lambda_X\left(E_{\mathbf{f}^*}, P\right), \tag{5}$$

which proves the induction step, since the path $E_{\mathbf{f}}$ that $RG(u_1, u_2, X, I)$ returns must satisfy $\Lambda_X(E_{\mathbf{f}}, P) \geq \Lambda_X(E'_{\mathbf{f}}, P)$. The goal is achieved by invoking the induction hypothesis on the two sub-procedures $RG(u_1, v^*, X, I-1)$ and $RG(v^*, t, X \cup E_{\mathbf{f}_1}, I-1)$, and combining the results using the monotonicity and submodularity of Λ (Lemma 1). Further details are deferred to [26] due to space constraints. □

Time Complexity: As we invoke at most $2n$ sub-procedures at each level of recursion and the computation of Λ takes $O(m)$ time, the time complexity of the algorithm is $O((2n)^I m)$. Taking $I = \log n + 1 \geq \lceil \log d \rceil$, we get an algorithm with a logarithmic approximation ratio of $1/(\lceil \log d \rceil + 1)$ with a quasi-polynomial time complexity of $O((2n)^{\log n + 1} m)$.

Remark: Note that the proof of Theorem 1 only relies on the monotonicity and submodularity of Λ. Therefore, the Recursive Greedy algorithm works for any monotone and submodular function on the subsets of E.

2) Non-disjoint User Paths: When the user paths are not disjoint, the problem becomes more challenging. First, notice that $\tilde{\lambda}_{iA} = \min\left(\lambda_i, \min_{e \in p_i} \tilde{C}_A(e)\right)$ no longer holds due to the constraints in (2) that couple different $\tilde{\lambda}_i$'s together. More importantly, Λ actually loses the submodular property,which prevents the direct application of the Recursive Greedy algorithm. We tackle the issues through approximating Λ with a submodular function $\bar{\Lambda}$, and run the Recursive Greedy algorithm on $\bar{\Lambda}$. The performance guarantee of the algorithm can be obtained by bounding the gap between Λ and $\bar{\Lambda}$.

Let $E_0 \subseteq E$ be the set of edges that belong to some user path. This is also the set of edges that appear in constraints (2). We partition E_0 into two sets E_1 and E_2, where E_1 is the set of edges that belong to only one user path, and E_2 is the set of edges that belong to at least two (intersecting) user paths. Following this, we define $\bar{\Lambda}(A, P), A \subseteq E$ to be evaluated through the two-phase procedure below. The procedure first goes edges in E_1 (Phase I), setting

$$\tilde{\lambda}_{iA}^{(1)} := \min\left(\lambda_i, \min_{e \in p_i, e \in E_1} \{\tilde{C}_A(e)\}\right), \quad \forall i.$$

Then, it goes over edges in E_2 (Phase II), setting

$$\tilde{\lambda}_{iA}^{(2)} := \tilde{\lambda}_{iA}^{(1)} \cdot \prod_{e \in p_i, e \in E_2, \tilde{C}_A(e) \leq \sum_{p_j \ni e} \lambda_j} \frac{\tilde{C}_A(e)}{\sum_{p_j \ni e} \lambda_j}, \quad \forall i.$$

Finally, it sets $\bar{\Lambda}(A, P) = \sum_i \lambda_i - \sum_i \tilde{\lambda}_{iA}^{(2)}$.

The procedure uses $\{\tilde{\lambda}_{iA}^{(2)}\}$, a set of flow values on user paths, as an approximate solution to the max-flow problem (1). The solution is obtained through first setting the flow values to $\{\lambda_i\}$ and then gradually decreasing them until the constraints are satisfied. In Phase I, the flow values are decreased to satisfy the capacity constraints posed by edge in E_1. In Phase II, the flow values are further reduced to compensate for the capacity violations on edges in E_2 through multiplying a factor $\frac{\tilde{C}_A(e)}{\sum_{p_j \ni e} \lambda_j}$, which is equal to the ratio between the capacity of e after the interdiction and the sum of flow values on e before the interdiction, to the flow value of each user path containing e, for each $e \in E_2$. Typically, Phase II overcompensates and thus $\bar{\Lambda}$ is an upper bound of Λ. But as we will show, the gap between $\bar{\Lambda}$ and Λ is moderate and such overcompensation guarantees the submodularity of $\bar{\Lambda}$.

Substituting Λ with $\bar{\Lambda}$ in **Algorithm 1**, we obtain the Recursive Greedy algorithm for the case of non-disjoint user paths. We will refer to this algorithm as the **Extended Recursive Greedy algorithm**. The name is justified by noting that when the user paths are disjoint, $E_2 = \emptyset$ and $\bar{\Lambda} = \Lambda$, the Extended Recursive Greedy algorithm degenerates to **Algorithm 1**.

Before analyzing the performance of the algorithm, we establish two lemmas that show the monotonicity and submodularity of $\bar{\Lambda}$, and bound the gap between $\bar{\Lambda}$ and Λ, respectively. The proofs of the lemmas are deferred to [26].

Lemma 2. *Consider $\bar{\Lambda}(\cdot, P) : 2^E \mapsto \mathbb{R}^*$ as a set function. $\bar{\Lambda}(\cdot, P)$ is monotone and submodular.*

Lemma 3. *$\Lambda(A, P) \leq \bar{\Lambda}(A, P) \leq (b+1) \cdot \Lambda(A, P)$ for all $A \subseteq E$, where $b = \max_i |E_2 \cap p_i|$, i.e., the maximum number of edges that a user path shares with other user paths.*

Now, we are ready to analyze the performance of the Extended Recursive Greedy algorithm.

Theorem 2. *If $I \geq \lceil \log d \rceil$, then the Extended Recursive Greedy algorithm returns an s-t path $E_{\mathbf{f}}$ that satisfies*

$$\Lambda(E_{\mathbf{f}}, P) \geq \frac{1}{(b+1) \cdot (\lceil \log d \rceil + 1)} \Lambda(E_{\mathbf{f}^*}, P),$$

where d is the length of $E_{\mathbf{f}^*}$ and $b = \max_i |E_2 \cap p_i|$.

Proof. By Lemma 2 and Theorem 1, we have $\bar{\Lambda}(E_{\mathbf{f}}, P) \geq \frac{1}{(\lceil \log d \rceil + 1)} \bar{\Lambda}(E_{\mathbf{f}^*}, P)$ when $I \geq \lceil \log d \rceil$. Invoking Lemma 3, we obtain that

$$\begin{aligned} \Lambda(E_{\mathbf{f}}, P) &\geq \frac{1}{b+1} \bar{\Lambda}(E_{\mathbf{f}}, P) \geq \frac{1}{(b+1) \cdot (\lceil \log d \rceil + 1)} \bar{\Lambda}(E_{\mathbf{f}^*}, P) \\ &\geq \frac{1}{(b+1) \cdot (\lceil \log d \rceil + 1)} \Lambda(E_{\mathbf{f}^*}, P), \end{aligned}$$

which concludes the proof. □

Note that the computation of $\bar{\Lambda}$ takes $O(m)$ time. Therefore, taking $I = \log n + 1$ we get a $\frac{1}{(b+1) \cdot (\lceil \log d \rceil + 1)}$-approximation algorithm with a time complexity of $O((2n)^{\log n + 1} m)$. Although in the worst case, b can be at the same order as n. In most cases, b is of $O(\log n)$, and the algorithm still enjoys a logarithmic approximation ratio.

IV. Robust Flow Interdiction

In this section, we investigate the robust flow interdiction problem. Following the road map of deterministic flow interdiction, we first describe the formal definition of the problem, then show its computational complexity, and finally present the approximation framework for the problem.

A. Problem Formulation

While deterministic flow interdiction considers the case where the interdictor has definitive knowledge of the user paths, robust flow interdiction concerns scenarios where such knowledge is not available. We model this more complicated situation using the robust optimization framework [16]. Instead of having certain knowledge of P, the interdictor only knows that P lies in an uncertainty set $\mathcal{U} = \{P_1, \ldots, P_\xi\}$. Each $P_l = \{p_{l1}, \ldots, p_{lk_l}\} \in \mathcal{U}$, associated with initial flow values $\{\lambda_{l1}, \ldots, \lambda_{lk_l}\}$, is a candidate set of paths that the users are operating on. The interdictor aims to hedge against the worst case, maximizing the minimum throughput reduction achieved over all candidates P.

Definition 3 (Robust Flow Interdiction)**.** *Given the uncertain set $\mathcal{U} = \{P_1, \ldots, P_\xi\}$ of the user paths and the associated initial flow values on user paths for each $P \in \mathcal{U}$, the robust flow interdiction problem seeks an interdiction strategy w that maximizes the worst case throughput reduction, i.e., $w \in \arg\max_{w'} \min_{P \in \mathcal{U}} \Lambda(w', P)$.*

Example: As an example of the robust flow interdiction problem, we again consider the network in Figure 1. The interdictor has source s, detination t and budget $\gamma = 2$. Assume that the interdictor only knows that the users are sending flow on either $\{p_1, p_2\}$ or $\{p_1, p_3\}$, and the initial flow values on p_1, p_2, p_3 are all three. This corresponds to the robust flow interdiction with $\mathcal{U} = \{\{p_1, p_2\}, \{p_1, p_3\}\}$. Let $\mathbf{f}_1$ be the single-path flow with value two on $\{(s, v_1), (v_1, v_3), (v_3, v_4), (v_4, t)\}$ and $\mathbf{f}_2$ be that on $\{(s, v_1), (v_1, v_3), (v_3, v_2), (v_2, t)\}$. The optimal strategy is $w(\mathbf{f}_1) = 1/3, w(\mathbf{f}_2) = 2/3$, and the worst case throughput reduction equals $8/3$ as $\Lambda(w, \{p_1, p_2\}) = \Lambda(w, \{p_1, p_3\}) = 8/3$. Note that in this example, no pure interdiction strategy can achieve a worst case throughput reduction of $8/3$, which demonstrates the superiority of mixed strategies in the robust flow interdiction setting.

The robust flow interdiction problem subsumes the deterministic one as a special case by setting $\mathcal{U} = \{P\}$. Therefore, we immediately have the following proposition.

Proposition 4. *The robust flow interdiction problem is NP-hard.*

Before presenting our approximation framework, we present a linear programming (LP) formulation that serves as an alternative solution to the robust flow interdiction problem. According to Proposition 1, we can restrict our attention to distributions on the set of single-path flows with value γ. Therefore, in the following, *the distributions we refer to are all on the set of single-path flows in $\mathcal{F}_\gamma$*. We enumerate such single-path flows in an arbitrary order and associate with each single-path flow $\mathbf{f}_i$ a variable w_i. Consider the linear program:

$$\begin{aligned} &\text{maximize } z && (6)\\ &\textbf{s.t. } \sum\nolimits_i w_i \Lambda(\mathbf{f}_i, P) \geq z, \quad \forall P \in \mathcal{U} \\ &\sum\nolimits_i w_i = 1 \\ &w_i \geq 0, \quad \forall i \end{aligned}$$

Clearly, the solution to the LP corresponds to an optimal interdiction strategy w to the robust flow interdiction problem with $w(\mathbf{f}_i) = w_i$. Hence, formulating and solving the LP is a natural algorithm for the robust flow interdiction. However, as the number of single-path flows can be exponential in the number of nodes n, the LP may contain an exponential number of variables. It follows that the algorithm has an undesirable exponential time complexity. We use this algorithm in the simulations to obtain optimal interdiction strategies for comparisons with our approximation framework. Furthermore, when the number of single-path flows is exponential in n, even outputting the strategy may take exponential time. This makes it impractical and unfair to compare any sub-exponential time approximation procedure to the optimal solution. We get around this issue by comparing our solution to the optimal interdiction strategy that takes non-zero values on at most N_0 single-path flows, where N_0 is a pre-specified number bounded by some polynomial of n. We refer to such strategies as N_0-bounded strategies. The optimal N_0-bounded strategy corresponds to the best strategy that uses at most N_0 different interdicting flows. Note that such restriction does not trivialize the problem since we place no limitation on the set but just the number of single-path flows that the interdictor can use.

B. Approximation Framework

In this section, we present the approximation framework we propose for the robust flow interdiction problem. As a generalization of the deterministic version, the robust flow interdiction is more complicated since it involves maximizing the minimum of a set of functions. The (Extended) Recursive Greedy algorithm cannot be directly adapted to this case. Instead, we design an approximation framework that integrates the Extended Recursive Greedy algorithm as a sub-procedure. The framework only incurs a logarithmic loss in terms of approximation ratio. The description and analysis of the approximation framework are carried out in three steps. First, we justify that it is sufficient to consider the robust flow interdiction problem with parameters taking rational/integral values. In the second step, building on the rationality/integrality of parameter values, we convert the problem to a sequence of integer linear programs. Finally, we solve the sequence of integer programs through iteratively invoking the Extended Recursive Greedy algorithm.

1) Rationalizing the Parameters: In the first step, we show that not much is lost if we only consider the interdiction strategies that take rational values and restrict the throughput reduction to take integer values. Specifically, let $N = N_0^2 + N_0$ and $\mathbb{Q}_N = \{\frac{\beta}{N} : \beta \in \mathbb{N}, 0 \leq \beta \leq N\}$ be the set of non-negative rational numbers that can be represented with N as denominator. Further, we define $\mathcal{W}_N$ to be the set of strategies that take value in $\mathbb{Q}_N$, i.e., $\mathcal{W}_N = \{w : \mathcal{F}_\gamma \mapsto \mathbb{Q}_N, \sum_{\mathbf{f}} w(\mathbf{f}) = 1\}$. We use w^* to represent the optimal N_0-bounded interdiction strategy, and w_N^* to represent optimal strategy in $\mathcal{W}_N$. The following lemma states that w^* can be well approximated by w_N^*.

Lemma 4. *For all $P \in \mathcal{U}$, $\Lambda(w_N^*, P) \geq \frac{N_0}{N_0+1}\Lambda(w^*, P)$.*

We now proceed to argue that it suffices to consider the throughput reduction function Λ to take integral values that are bounded by some polynomial of n. First, when the integrality of Λ is not satisfied, we can always use standard scaling and rounding tricks to get a new instance of the problem, where Λ takes integral values. Our framework can be applied to the new instance, yielding an interdiction strategy that has almost the same performance guarantee for both the original and the new instances. We omit the formal statement and its proof here. Second, since γ is bounded by some polynomial of n, $\max_{w,P} \Lambda(w, P)$ is also bounded by some polynomial of n. Now, let $M = N \max_{w,P} \Lambda(w, P)$. We can thus without loss of generality assume that M is an integer bounded by some polynomial of n.

With the above results, we move into the second step, that converts the robust flow interdiction problem into a sequence of integer linear programs.

2) Converting into Integer Linear Programs: Recall the enumeration of single-path flows in the LP (6). This time, we associate each flow $\mathbf{f}_i$ with an integral variable x_i. Consider the following integer program $ILP(\kappa)$ parameterized by a positive integer $\kappa \leq M$.

$$\text{minimize } \sum\nolimits_i x_i \tag{7}$$

$$\textbf{s.t. } \sum\nolimits_i x_i \Lambda(\mathbf{f}_i, P) \geq \kappa, \quad \forall P \in \mathcal{U} \tag{8}$$

$$x_i \in \mathbb{N}, \quad \forall i \tag{9}$$

Each x_i indicates the number of times $\mathbf{f}_i$ is selected. $ILP(\kappa)$ can be interpreted as selecting the single-path flows for the minimum total number of times that achieve a throughput reduction of κ for all candidate P. For each κ, we denote by N_κ the optimal value of $ILP(\kappa)$. If we can obtain an optimal solution $\{x\}$ to $ILP(\kappa)$, then the strategy w with $w(\mathbf{f}_i) = x_i/N_\kappa$ satisfies $\min_{P \in \mathcal{U}} \Lambda(w, P) \geq \kappa/N_\kappa$. In the following lemma, we show that the strategy constructed according to the solution to the integer program with the maximum value of κ/N_κ is a close approximation to the optimal N_0-bounded strategy in terms of worst case throughput reduction.

Lemma 5. *Let $\kappa^* = \arg\max_{1 \leq \kappa \leq M}(\kappa/N_\kappa)$. We have $\frac{\kappa^*}{N_{\kappa^*}} \geq \min_{P \in \mathcal{U}} \Lambda(w_N^*, P) \geq \frac{N_0}{N_0+1} \min_{P \in \mathcal{U}} \Lambda(w^*, P)$.*

Connecting the analysis so far, we have a clear procedure to compute a near-optimal interdiction strategy for the robust flow interdiction. First, we construct and solve $ILP(\kappa)$ for $1 \leq \kappa \leq M$. Second, we take optimal solution with the maximal κ/N_κ and obtain its corresponding interdiction strategy, which is within a factor of $\frac{N_0}{N_0+1}$ to the optimal N_0-bounded strategy. The final step of our framework is devoted to solving $ILP(\kappa)$.

3) Solving the Integer Linear Programs: Resembling (6), each $ILP(\kappa)$ involves potentially exponential number of variables. What is different and important is that, we can obtain a $\frac{1}{\log M}$-approximation through a greedy scheme that iteratively chooses a single-path flow according to the following criterion: let $\{x\}$ indicate the collection of flows that have been chosen so far, i.e., each $\mathbf{f}_i$ has been chosen for x_i times. Let i^* be

$$\arg\max_i \sum_{P \in \mathcal{U}, \kappa \geq \sum_j x_j \Lambda(\mathbf{f}_j, P)} \min\{\kappa - \sum\nolimits_j x_j \Lambda(\mathbf{f}_j, P), \Lambda(\mathbf{f}_i, P)\}. \tag{10}$$

The greedy scheme chooses $\mathbf{f}_{i^*}$ at the current iteration and increments x_{i^*} by 1. The above procedure is repeated until we have $\sum_i x_i \Lambda(\mathbf{f}_i, P) \geq k$ for all $P \in \mathcal{U}$. Moreover, if we apply an α-approximate greedy scheme, which chooses $\mathbf{f}_i$ that is an α-optimal solution to (10), then the final solution we obtain is $\alpha \log M$-optimal. Essentially, (10) selects the flow that provides the maximum marginal gain with respect to satisfying the constraints (8) for all $P \in \mathcal{U}$. That the (α-approximate) greedy scheme achieves an logarithmic approximation follows from the relation of $ILP(\kappa)$ to the multiset-multicover problems and the results therein [19], which we omit here due to space limitation. Now recall the equivalence between $\Lambda(\mathbf{f}, P)$ and $\Lambda(E_\mathbf{f}, P)$ established in Section III. We proceed to show that the Recursive Greedy algorithm can be used to construct an approximate greedy scheme. First, we have the following lemma.

Lemma 6. *If Λ is monotone and submodular, then the objective function of (10) is also monotone and submodular.*

By Lemma 6, the Recursive Greedy algorithm (or the Extended Recursive Greedy algorithm using $\bar{\Lambda}$ instead of Λ when the user paths are not disjoint) can be applied to the maximization of (10) and enjoys the same performance guarantee as in Theorems 1 and 2. Hence, the final step can be completed by an approximate greedy scheme that iteratively invokes the (Extended) Recursive Greedy algorithm. We now summarize the three steps of our approximation framework for the robust flow interdiction as **Algorithm 2** and analyze its performance.

Theorem 3. ***Algorithm 2** returns an interdiction strategy w that satisfies*

$$\min_{P \in \mathcal{U}} \Lambda(w, P) \geq \left(\frac{N_0}{(N_0+1)(b+1)\log M \cdot (\lceil \log d \rceil + 1)}\right) \min_{P \in \mathcal{U}} \Lambda(w^*, P),$$

where w^* is the optimal N_0-bounded strategy.

Proof. Let $ILP(\kappa)$ and $\{x\}$ be the integer linear program and its solution that correspond to w. We inherit the definition of κ^* in Lemma 5 and further define $\{x^*\}$ to be the solution that **Algorithm 2** computes for $ILP(\kappa^*)$. We have

$$\min_{P \in \mathcal{U}} \Lambda(w, P) = \min_{P \in \mathcal{U}} \sum\nolimits_i w(\mathbf{f}_i)\Lambda(\mathbf{f}_i, P)$$

$$= \min_{P \in \mathcal{U}} \sum_i \frac{x_i}{\sum_j x_j} \Lambda(\mathbf{f}_i, P) \geq \frac{\kappa}{\sum_j x_j} \geq \frac{\kappa^*}{\sum_j x_j^*}$$

$$\geq \left(\frac{1}{(b+1)\log M \cdot (\lceil \log d \rceil + 1)}\right) \frac{\kappa^*}{N_{\kappa^*}} \tag{11}$$

$$\geq \left(\frac{N_0}{(N_0+1)(b+1)\log M \cdot (\lceil \log d \rceil + 1)}\right) \min_{P \in \mathcal{U}} \Lambda(w^*, P), \tag{12}$$

where inequality (11) follows from Theorem 2 and the results in [19], and inequality (12) follows from Lemma 5. □

Time Complexity: Note that **Algorithm 2** solves M integer linear programs, and it takes at most $M\xi$ calls of the (Extended) Recursive Greedy algorithm for each program since the number of iterations is bounded by $M\xi$, where $\xi = |\mathcal{U}|$. Furthermore, at the third step, there are at most $N_\kappa \leq M\xi$ variables with non-zero values in the solution $\{x\}$, which implies that w can be output in $O(M\xi)$ time. Therefore, the time complexity of **Algorithm 2** is $O\left(m(M\xi)^2(2n)^{\log n+1}\right)$.

```
Algorithm 2 Algorithm for the Robust Flow Interdiction
Input: Network graph $G$, Uncertainty set $\mathcal{U} = \{P_1, \ldots, P_\xi\}$
Output: Interdiction Strategy $w$
1: Formulate $ILP(\kappa)$ for $1 \leq \kappa \leq M$.
2: Solve each $ILP(\kappa)$ using the approximate greedy scheme
   based on the (Extended) Recursive Greedy algorithm.
3: Take the solution $\{x\}$ to $ILP(\kappa)$ with the maximum
   value of $\kappa / \sum_j x_j$ and construct $w$ by setting $w(\mathbf{f}_i) =$
   $x_i / \sum_j x_j$ for all $i$.
```

V. Simulations

In this section, we present our evaluation of the performance of the proposed algorithms. We first introduce the simulation environment in the following and then show the detailed results in subsequent sections.

A. Simulation Setting

We adopt the Gnutella network data set from [23]. We extract 20 networks of 1000 nodes, and make the networks acyclic by removing a minimal feedback edge set from each of them. The capacities of the edges are sampled from a normal distribution with mean 20 and standard deviation 3. The budget of the interdictor is set to the minimum capacity of the edges in each network.

B. Deterministic Flow Interdiction

In the deterministic flow interdiction, we divide our simulations into two parts, where the user paths are disjoint and non-disjoint respectively. In the first part, we designate k disjoint paths in each network as user paths with k varying in $\{10, 20, \ldots, 100\}$. In the second part, we follow the similar route, except that the user paths are randomly chosen without guaranteeing their disjointness. For each network, we randomly select five connected node pairs as the source and destination of the interdictor. Thus, for each number of user paths, we have 100 simulation scenarios in total (20 networks times 5 s-t pairs).

1) Algorithms Involved in Performance Comparisons: We apply the Recursive Greedy algorithm when the user paths are disjoint and run the extended one when the user paths are non-disjoint. We vary the recursion depth, i.e., the value of I in **Algorithm 1** to evaluate its influence on the algorithms' performance. Our algorithms are compared to a *brute force* algorithm that enumerates all the paths between the interdictor's source and destination, which computes the optimal interdiction strategy.

2) Performance Metric: We calculate the ratio of the throughput reduction of the interdiction strategies by our algorithms to that of the optimal solutions obtained by the brute force algorithm. The results reported are the average over all the 100 scenarios.

3) Simulation Results: We plot the results of our algorithms on deterministic flow interdiction with disjoint and non-disjoint user paths in Figures 3(a) and 3(b).

From Figure 3(a), we can see that: (i). by setting the recursion depth to two, we get interdiction strategies with throughput reduction more than 90% of the optimal (0.9-approximation) and (ii). by setting the recursion depth to three, we recover the optimal interdiction strategies. Furthermore, we find that when the recursive depth is three, the number of paths examined by the Recursive Greedy algorithm is just about one fifth of the total number of s-t paths. This suggests that the typical performance and running time are even better than what the theoretical analysis predicts. Finally, we observe that, in general, our algorithms perform better when the number of user paths is large. This observation also holds in subsequent cases. One possible explanation for this is that more user paths present more opportunities for throughput reduction, making (near-)optimal interdicting flows easier to find.

As demonstrated in Figure 3(b), the deterministic flow interdiction is harder to approximate when the user paths are non-disjoint. But we can still get 0.8-approximations with a recursion depth of three and 0.95-approximations with a recursion depth of four. Also, though we have not plotted in the figure, we have seen that increasing the recursion depth to five or six does not further improve the performance. Therefore, the gap between the Extended Recursive Greedy algorithm with depth of four and the optimal can be attributed to the loss brought by the approximate throughput reduction function $\bar{\Lambda}$.

C. Robust Flow Interdiction

In the robust flow interdiction, we randomly select 10 groups of k paths as the uncertainty set $\mathcal{U}$ for $k \in \{10, 20, \ldots, 100\}$. Similar as before, we randomly selected 5 source-destination pairs for the interdictor in each network and form 100 scenarios for each k.

1) Algorithms Involved in Performance Comparisons: We embed the Extended Recursive Greedy algorithm with different recursion depths in our proposed approximation framework (**Algorithm 2**). The optimal solution in this case is obtained by solving the LP (6).

2) Performance Metric: For all strategies w computed by our algorithms, we calculate the ratio of worst-case throughput reduction $\min_{P \in \mathcal{U}} \Lambda(w, P)$ to that of the optimal. The results reported are again averaged over all the scenarios.

3) Simulation Results: We plot the results in Figure 3(c). Taking the depth as four, our approximation framework achieves interdiction strategies that are more than 70% of the optimal (0.7-approximation). As in the previous case, we have implemented the framework with recursion depth of five and six but found that it did not improve the performance.

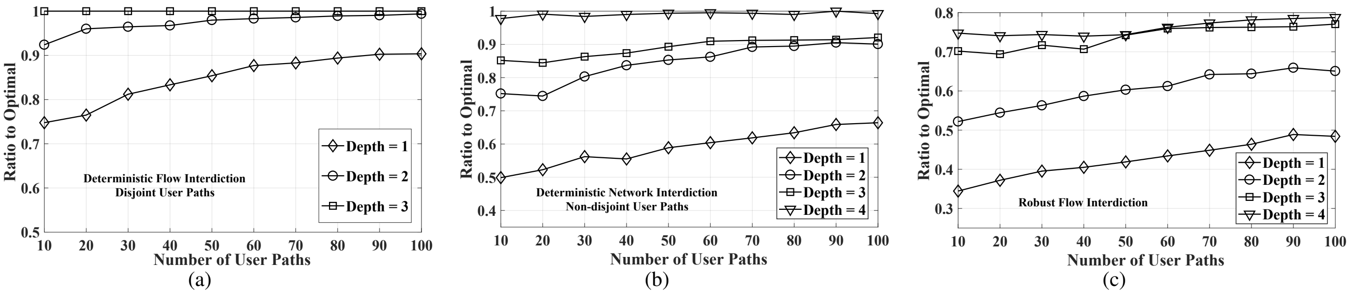


Fig. 3. Ratio of throughput reduction of the solutions by our algorithms to the optimal.

VI. Extension to General Networks

In this section, we extend our network interdiction paradigm and the two flow interdiction problems to general networks. Our network interdiction paradigm can be straightforwardly extended to general networks by allowing the network graph to be a general directed graph. One caveat is that we need to additionally restrict the flows that the interdictor injects to be free of cycles. Since otherwise, as the flow value of a cycle is zero, the interdictor would be able to consume the capacities of the edges in any cycle without spending any of its budget, which would lead to meaningless solutions. Under the generalized paradigm, the deterministic and robust flow interdiction problems can be defined in the same way as Definitions 2 and 3. For the network interdiction paradigm on general networks, Proposition 1 still holds. But the Recursive Greedy algorithm will break down since the edge set it returns will be an s-t walk instead of an s-t path (i.e. it may contain cycles). Furthermore, we can prove by an approximation-preserving reduction from the Longest Path problem in directed graphs [24] that there is no polynomial time algorithm with an approximation ratio of $O(n^{1-\delta})$ for any $\delta > 0$ unless $P = NP$. This implies that the two flow interdiction problems on general directed graph are extremely hard to approximate within a non-trivial factor in polynomial or even quasi-polynomial time. The proof is omitted due to space constraints.

VII. Conclusion

In this paper, we proposed a new paradigm for network interdiction that models the interdictor's action as injecting bounded-value flows to maximally reduce the throughput of the residual network. We studied two problems under the paradigm: deterministic flow interdiction and robust flow interdiction, where the interdictor has certain or uncertain knowledge of the operation of network users, respectively. Having proved the computation complexity of the two problems, we proposed an algorithm with logarithmic approximation ratio and quasi-polynomial running time was proposed for the deterministic flow interdiction. We further developed an approximation framework that integrates the algorithm and forms a quasi-polynomial time procedure that approximates the robust flow interdiction within a poly-logarithmic factor. Finally, we evaluated the performance of the proposed algorithms through simulations.

References

[1] R. K. Wood, "Deterministic network interdiction", in *Mathematical and Computer Modelling*, Vol. 17, No. 2, pp. 1-18, 1993.

[2] C. Lim and J. C. Smith, "Algorithms for discrete and continuous multicommodity flow network interdiction problems", in *IIE Transactions*, Vol. 39, No. 1, pp. 15-26, 2007.

[3] C. Phillips, "The network inhibition problem", in *Proc. ACM STOC*, 1993.

[4] D. Bertsimas, E. Nasrabadi and J. B. Orlin, "On the power of randomization in network interdiction", in *Operations Research Letters*, Vol. 44, No. 1, pp. 114-120, 2016.

[5] R. Zenklusen, "Network flow interdiction on planar graphs", in *Discrete Applied Mathematics*, Vol. 158, No. 13, pp. 1441-1455, 2010.

[6] C. Burch, R. Carr, S. Krumke, M. Marathe, C. Phillips and E. Sundberg, "A decomposition-based pseudoapproximation algorithm for network flow inhibition", in *Network Interdiction and Stochastic Integer Programming*, pp. 51-68, 2003.

[7] J. O. Royset, and R. K. Wood, "Solving the bi-objective maximum-flow network-interdiction problem", in *INFORMS Journal on Computing*, Vol. 19, No. 2, pp. 175-184, 2007.

[8] J. Zheng and D. A. Castañón, "Dynamic network interdiction games with imperfect information and deception", in *IEEE ICC*, pp. 7758-7763, 2012.

[9] Neumayer, Sebastian, Alon Efrat, and Eytan Modiano. ”Geographic max-flow and min-cut under a circular disk failure model.” Computer Networks 77 (2015): 117-127.

[10] S. Neumayer, G. Zussman, R. Cohen, and E. Modiano, "Assessing the vulnerability of the fiber infrastructure to disasters", in *IEEE/ACM Trans. on Networking*, Vol. 19, No. 6, pp. 1610-1623, 2011.

[11] A. Sanjab, S. Walid and T. Başar, "Prospect theory for enhanced cyber-physical security of drone delivery systems: A network interdiction game", *IEEE ICC*, 2017.

[12] I. Aad, JP. Hubaux and E. W. Knightly, "Impact of denial of service attacks on ad hoc networks", in *IEEE/ACM Trans. on Networking*, Vol. 16, No. 4, pp. 791-802, 2008.

[13] B. Wang, Y. Zheng, W. Lou and YT. Hou, "DDoS attack protection in the era of cloud computing and software-defined networking", in *Computer Networks*, Vol. 81 pp. 308-319, 2015.

[14] M. Ficco and M. Rak, "Stealthy denial of service strategy in cloud computing", in *IEEE Trans. on Cloud Computing*, Vol. 3, No. 1, pp. 80-94, 2015.

[15] G. S. Paschos and L. Tassiulas, "Sustainability of Service Provisioning Systems Under Stealth DoS Attacks", in *IEEE Trans. on Control of Network Systems*, Vol. 4, No. 4, pp. 749-760, 2017.

[16] A. Ben-Tal, L. El Ghaoui and A. Nemirovski, "Robust optimization", Vol. 28, Princeton University Press, 2009.

[17] J. C. Smith, M. Prince and J. Geunes, "Modern network interdiction problems and algorithms", in *Handbook of Combinatorial Optimization*, Springer, pp. 1949-1987, 2013.

[18] C. Chekuri and M. Pal, "A recursive greedy algorithm for walks in directed graphs", in *IEEE FOCS*, 2005.

[19] V.V. Vazirani, "Approximation algorithms", Springer Science & Business Media, 2013.

[20] D. Granata and A. Sgalambro, "Network Interdiction through Length-Bounded Critical Disruption Paths: a Bi-Objective Approach", in *Electronic Notes in Discrete Mathematics*, Vol. 52, pp. 375-382, 2016.

[21] R.K. Ahuja, T. L. Magnanti and J. B. Orlin, ”Network flows”, Pearson Education, 2014.

[22] D. Bertsimas and JN. N. Tsitsiklis, "Introduction to linear optimization", Athena Scientific, 1997.

[23] M. Ripeanu, I. Foster and A. Iamnitchi, "Mapping the Gnutella Network: Properties of Large-Scale Peer-to-Peer Systems and Implications for System Design", in *IEEE Internet Computing Journal*, 2002.

[24] A. Björklund, T. Husfeldt and S. Khanna, "Approximating longest directed paths and cycles", in *International Colloquium on Automata, Languages, and Programming*, Springer, 2004.

[25] R.M. Karp, "Reducibility among combinatorial problems", in *Complexity of computer computations*, Springer, pp. 85-103, 1972.

[26] Xinzhe Fu and Eytan Modiano, "Network Interdiction Using Adversarial Traffic Flows", arXiv:1901.02572.