On the Robustness of Distributed Computing Networks

Jianan Zhang, Hyang-Won Lee, and Eytan Modiano

arXiv:1901.02636v3 [cs.NI] 26 Nov 2021

Abstract—Traffic flows in a distributed computing network require both transmission and processing, and can be interdicted by removing either communication or computation resources. We study the robustness of a distributed computing network under the failures of communication links and computation nodes. We define cut metrics that measure the connectivity, and show a non-zero gap between the maximum flow and the minimum cut. Moreover, we study a network flow interdiction problem that minimizes the maximum flow by removing communication and computation resources within a given budget. We develop mathematical programs to compute the optimal interdiction, and polynomial-time approximation algorithms that achieve near-optimal interdiction in simulation.

I. INTRODUCTION

Cloud computing has been growing rapidly in recent years. For example, over one millon servers have been deployed for Amazon Web Service, which generates billions of dollars in revenue each year and grew by over 40 percent in revenue in 2018. Cloud networks, and computing networks in general, facilitate agile, reliable and cost effective implementations for a variety of applications. The robustness of computing networks is essential for web access, online database, video streaming, among other applications deployed in the cloud.

Network flows in a computing network rely on both communication resources for transmission and computation resources for processing. The unavailability of either type of resources may lead to the failure of flows. Hundreds of thousands of websites were down due to the computation resource failure in a data center for Amazon Web Service [1]. In 2006, Internet services in Asia were disrupted by communication failure due to the broken of submarine cables by earthquake [2].

The dependence of network flow on various types of resources brings challenges to the reliability of a computing network [3], [4], [5]. Previous research proposed new computing network architectures to improve reliability [6], [7], [8], and developed models to study failure cascading and protection strategies [9], [10]. However, limited works focus on the rigorous analysis of network flow reduction under the failures of network resources, which is a key metric for computing network performance and is the focus of this paper.

Flow interdiction problems have been extensively studied based on the classical flow network model. The problem of minimizing the maximum flow by removing network links within a budget is strongly NP-hard [11]. Integer linear programs were developed to compute the optimal interdiction [11]. Approximation hardness results and a $2(n-1)$-approximation algorithm was developed in [12]. A pseudoapproximation algorithm was developed in [13] based on linear programming relaxation, and developed in [14] with faster combinatorial algorithm implementation. NP-hardness result and a polynomial-time approximation scheme were developed for network flow interdiction on planer graphs [15], [16].

In a traditional flow network, the maximum flow between a source-destination $(s-t)$ pair equals the minimum cut, which is the minimum-capacity link removals that disconnect the $s-t$ pair [17]. In a computing network, we show that there is a non-zero gap between the maximum flow and the minimum cut. The non-zero gap between maximum flow and minimum cut exists in a wide range of network interdiction scenarios, abstracted by the shared risk group model [18], where a single failure event may destroy multiple network components. For example, in layered communication networks, such as IP-over-WDM networks, the failure of a physical link may affect multiple logical links, and the maximum number of failure-disjoint paths could be smaller than the minimum number of physical link failures that induce a cut [19], [20]. In geographically correlated failure models [21], [22], [23], one geographical failure affects multiple nodes and links, and the minimum cut can also be greater than the maximum flow [24]. Although seemingly unrelated, we show that a computing network can be analyzed using a layered graph where link failures are coupled, and thus the gap exists.

The main contributions of this paper are as follows. We propose a model for a computing network to characterize the dependence of network flow on both communication and computation resources. The model facilitates the analysis of computing network robustness, by integrating the modeling of the computation resource to a classical graph model. By extending the classical cut metric for a graph, we define cut metrics that characterize computing network robustness under the failures of communication and computation resources. We prove the computation complexity, and develop integer programs and approximation algorithms to compute the minimum cuts. Moreover, we formulate a maximum flow interdiction problem, where the objective is to minimize the maximum $s-t$ flow by removing network resources within a given budget. We prove the computation complexity, and develop exact and approximation algorithms to compute optimal interdiction

strategies. A preliminary version of this paper appeared in [25].

The rest of this paper is organized as follows. In Section II, we introduce the model for a distributed computing network, and define cut metrics to evaluate the network robustness. In Section III, we develop algorithms to evaluate the maximum flow and minimum cuts. In Section IV, we formulate and solve a maximum flow interdiction problem with an interdiction budget. Section V provides numerical results. Section VI concludes the paper.

II. Model

In this section, we develop a model for a distributed computing network, and define metrics for network robustness.

A distributed computing network is modeled by a directed graph $G(V, E)$, where V denotes the set of forwarding and computation nodes, and E denotes the set of communication links. Computation nodes can process and forward packets, while forwarding nodes can only forward packets. A computation node $u \in V$ has processing capacity μ_u, and a communication link $(u, v) \in E$ has transmission capacity μ_{uv}.

Unlike the traditional data network where flows require minimal fixed computation tasks such as routing table lookup and checksum, flows in the distributed computing network can require vastly different computation resources, and hence computation capacities at servers (as well as communication bandwidth) are essential to process traffic. The classical robustness metric such as minimum cut is not able to capture the robustness of such a computing network. We extend classical flow and cut metrics to computing networks, to characterize the need to incorporate both communication and computation resources in network operation.

We first define *computation path* which supports both the processing and the delivery of data packets in the network.

Definition 1. A *computation path* (P, w) from a source s to a destination t is characterized by a sequence of connected edges (and their end nodes) P that start at s and end at t, and includes a computation node $w \in P$.

A network flow consists of packets that are originated at a source, processed at one or more computation nodes, and delivered to a destination. A flow can be decomposed into computation paths. We illustrate an $s - t$ flow and computation paths decomposition using an example in Fig. 1, where computation nodes are illustrated by squares and forwarding nodes are illustrated by circles, and the numbers represent capacities. The maximum $s - t$ flow is four, and can be decomposed into one unit flow on each of the four computation paths $(\{s - u_1 - v_1 - t\}, u_1), (\{s - u_1 - v_1 - t\}, v_1), (\{s - u_2 - v_2 - t\}, u_2), (\{s - u_2 - v_2 - t\}, v_2)$.

In order to reduce the flow carried by a computation path to zero, either any communication link or the computation resource in the path should be removed. Note that we consider the removal of computation resources without removing the node, i.e., the node can still forward packets without processing them.

In general, there are multiple computation paths from a source to a destination. To interdict the flow, a combination of
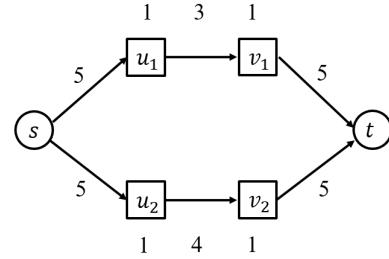

Fig. 1: An example of a distributed computing network.

communication and computation resources can be removed. We next define cuts that measure the connectivity of a pair of nodes in a computing network.

Definition 2. A *communication cut* for an $s - t$ pair is a set of communication links E_c such that the $s - t$ flow is reduced to zero after removing E_c.

Note that a communication cut can be different from the classical edge cut, since a path with zero computation resource does not need to be disconnected.

Definition 3. A *computation cut* for an $s - t$ pair is a set of computation nodes V_c such that the $s - t$ flow is reduced to zero after removing the computation resources at V_c.

Definition 4. A *joint communication and computation cut* (abbr. *joint cut*) for an $s - t$ pair is a set of communication links E_c and computation nodes V_c such that the $s - t$ flow is reduced to zero after removing E_c and computation resources at V_c.

We illustrate these cuts using the example in Fig. 1. Edges $\{(u_1, v_1), (u_2, v_2)\}$ form a communication cut, since s and t are disconnected after removing the two links. Nodes $\{u_1, v_1, u_2, v_2\}$ form a computation cut, since no flow can be processed after removing the computation resources at the four computation nodes. The union of edge $\{(u_1, v_1)\}$ and nodes $\{u_2, v_2\}$ is a joint cut, since the upper path is disconnected after removing (u_1, v_1), and the lower path cannot process flow after removing the computation resources at nodes $\{u_2, v_2\}$.

To simplify the analysis for network robustness, we assume that *all flows have the same resource requirement,* i.e., every unit flow requires a fixed amount of communication and computation resources. The identical resource requirement of flows can be justified by the statistical multiplexing of individual flows in a network, although individual flows for different applications may have different resource requirements. For example, video streaming is communication intensive, while search is computation intensive. By normalizing units and ignoring flow scaling, we further assume that *every unit flow requires one unit of computation resource for processing, and outputs one unit processed flow.* Under this assumption, one unit flow on a computation path occupies one unit communication resource at every link along the path, and one unit computation resource at the computation node.

Before developing algorithms to evaluate the maximum flow

and the minimum cuts, we prove the complexity of computing the cut metrics. First, we show the hardness of evaluating the minimum communication cut, whose proof is in the Appendix.

Lemma 1. *Computing the minimum communication cut for an $s - t$ pair is NP-hard, if there is more than one computation node.*

Lemma 1 implies that computing the minimum joint cut is NP-hard, since the minimum communication cut can be viewed as a special case of the minimum joint cut when the computation resources are abundant at computation nodes.

Theorem 1. *Computing the minimum joint communication and computation cut for an $s - t$ pair is NP-hard, if there is more than one computation node.*

III. COMPUTATION OF MAX-FLOW AND MIN-CUTS

In this section, we study the computation of the maximum flow and minimum cuts for a source-destination pair. We develop polynomial-time algorithms to evaluate the maximum flow and the minimum computation cut, and integer programs to evaluate the minimum communication cut and the minimum joint cut. In Section III-A, we develop mathematical programs to evaluate the maximum flow and the minimum cut using path-based formulation, which are intuitive but have an exponential number of variables or constraints. In Section III-B, we develop a layered graph representation to simplify their computations, and develop mathematical programs of polynomial sizes. Finally, in Section III-C, we study the gap between the maximum flow and the minimum cuts.

A. Path-based formulations

We first develop mathematical programs to compute the maximum flow and minimum cuts using path-based formulations. While the formulations have an exponential number of variables or constraints, they highlight the connections between flow and cuts in a computing network to those in a classical flow network.

We formulate a linear program to compute the maximum flow in a computing network. Let $\mathcal{P}$ denote the set of $s - t$ paths. Let $x_{P,w}$ denote the amount of flow transmitted through path P and processed at a computation node $w \in P$. The maximum flow can be computed by the following linear program.

$$\max \quad \sum_{P \in \mathcal{P}, w \in P} x_{P,w} \tag{1}$$

$$\text{s.t.} \quad \sum_{P \in \mathcal{P}, w \in P: (u,v) \in P} x_{P,w} \leq \mu_{uv}, \quad \forall (u,v) \in E, \tag{2}$$

$$\sum_{P \in \mathcal{P}: w \in P} x_{P,w} \leq \mu_w, \quad \forall w \in V, \tag{3}$$

$$x_{P,w} \geq 0, \quad \forall P \in \mathcal{P}, w \in P.$$

The communication capacity constraints are guaranteed by (2), and the computation capacity constraints are guaranteed by (3), by restricting the total amount of flow transmitted by a link or processed at a computation node. The objective is to maximize the total amount of flow supported by the computation paths.

We then develop an integer program to evaluate the minimum joint communication and computation cut using the path-based formulation. Indicator variable y_{uv} represents whether link (u,v) is removed. Indicator variable y_w represents whether the computation resource at node w is removed. Constraint (5) guarantees that for each path, either one of the link is removed, or all the computation resources are removed.

$$\min \quad \sum_{(u,v) \in E} \mu_{uv} y_{uv} + \sum_{w \in V} \mu_w y_w \tag{4}$$

$$\text{s.t.} \quad \sum_{(u,v) \in P} y_{uv} + y_w \geq 1, \quad \forall P \in \mathcal{P}, w \in P \tag{5}$$

$$y_{uv} \in \{0, 1\}, \quad \forall (u,v) \in E$$

$$y_w \in \{0, 1\}, \quad \forall w \in V.$$

Pure communication or computation cuts can be obtained by the above integer program with additional constraints. A minimum communication cut can be obtained by setting $y_w = 0, \forall w \in V$. A minimum computation cut can be obtained by setting $y_{uv} = 0, \forall (u,v) \in E$.

The number of paths $|\mathcal{P}|$ can be exponential in the size of the network. Both the linear program (1) and the integer program (4) have exponential sizes. Compared with the classical maximum flow and minimum cut formulations, the main difference is that a computation path in the computing network depends on a computation node in addition to a sequence of connected links. The coupling of constraints by the computation nodes brings challenges to the evaluation of the flow and cut metrics.

B. Layered graph formulations

To address the challenges, we develop a layered graph representation to simplify the evaluation of flow and cuts. Based on the layered graph, in Sections III-B1 and III-B2, we develop modified mathematical programs with a polynomial number of variables and constraints to evaluate the maximum flow and the minimum cuts, respectively.

We consider a two-layer graph, where every layer has the same topology as the original graph. An edge connects the two copies of each computation node across the two layers. Unprocessed flows are transmitted thought links in the upper layer $G(V, E)$, while processed flows are transmitted in the lower layer $G'(V', E')$. Flows across the two layers represent processing at computation nodes. For example, in Fig. 2, a flow is transmitted through (s, u), processed at u, and then transmitted through (u, v) and (v, t). In the layered graph, unprocessed flow is transmitted through (s, u) in the upper layer, then transmitted through (u, u'), which represents the processing at u, and finally transmitted through (u', v') and (v', t') in the lower layer. Every flow from s to t and processed at computation nodes in the original graph can be represented by a flow from s to t' in the layered graph. We next show that the network resource failures that disconnect (s, t) in the computing network can be mapped to failures that disconnect (s, t') in the layered graph.

Lemma 2. *Let S be an $s - t$ cut in the computing network. In the layered graph, removing edges $S' = \{(u,v),(u',v')|(u,v) \in S\} \cup \{(w,w')|w \in S\}$ disconnects s and t'.*

Proof. We prove by contradiction. Suppose that a path P exists between s and t' in the layered graph after removing S'. The path P must contain a link from the upper layer to the lower layer, denoted by (a, a'). There is a path P_1 from s to a in the upper layer, and a path P_2' from a' to t' in the lower layer. Let $P_2 = \{(u,v)|(u',v') \in P_2'\}$. Since none of the edges $P_1 \cup P_2'$ belong to cut S', none of the edges $P_1 \cup P_2$ belong to cut S, under the construction of S'.

In the computing network, there is a path P_1 from s to a, and a path P_2 from a to t. Moreover, the computation resource at a is not removed, since (a, a') remains in the layered graph. The path $(P_1 \cup P_2, a)$ is a computation path from s to t, which contradicts with the fact that S is an $s - t$ cut. $\square$
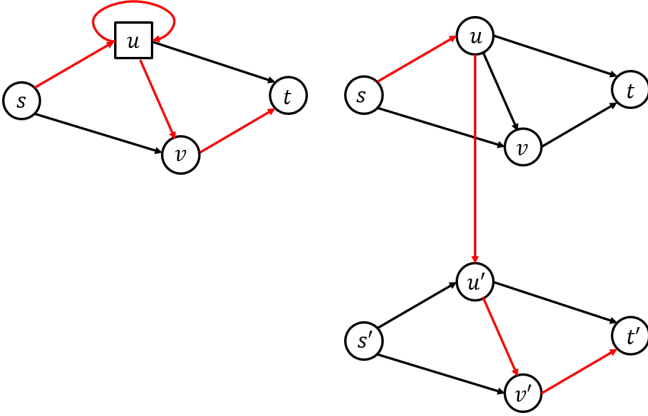

Fig. 2: Flows in the original and layered graphs.

1) Evaluation of maximum flow: Flow conservation holds in the layered graph, since communication and computation units are normalized and flow scalings are ignored. The difference between a flow in the layered graph and the classical network flow is that the sum of flows on the two copies of a link should not exceed the transmission capacity. Let $\tilde{E}$ denote the union of the set of links in the layered graph and a link from t' to s that has an infinite capacity. Let $\tilde{V} = V \cup V'$ denote the set of nodes in the layered graph. Let f_e denote the amount of flow on link e in the layered graph. The maximum flow from s to t' can be computed using the following linear program.

$$\max \quad f_{t's} \tag{6}$$
$$\text{s.t.} \sum_{u \in \tilde{V}:(u,v)\in\tilde{E}} f_{uv} - \sum_{w \in \tilde{V}:(v,w)\in\tilde{E}} f_{vw} = 0, \forall v \in \tilde{V}, \tag{7}$$
$$f_{ww'} \leq \mu_w, \quad \forall w \in V, \tag{8}$$
$$f_{uv} + f_{u'v'} \leq \mu_{uv}, \quad \forall(u,v) \in E, \tag{9}$$
$$f_{uv} \geq 0, f_{u'v'} \geq 0, \quad \forall(u,v) \in E,$$
$$f_{ww'} \geq 0, \quad \forall w \in V.$$

Flow conservation constraints are guaranteed by (7). Computation capacity constraints are guaranteed by (8) for each computation node. Communication capacity constraints are guaranteed by (9) for each communication link. The linear program has $O(|E|)$ variables and $O(|E|)$ constraints, which has a significantly smaller size compared with the path-based formulation. To conclude, the maximum flow can be computed by the linear program in polynomial time.

2) Evaluation of minimum cuts: Recall that an $s - t$ flow can be interdicted by removing either communication or computation resources, or a combination of both. We first develop an integer program to compute the minimum joint communication and computation cut, which can be easily modified to compute the minimum communication cut and the minimum computation cut. The formulation is based on disconnecting s and t' in the layered graph, which equivalently reduces the $s-t$ flow to zero in the original graph by Lemma 2.

We use different *node potentials* to indicate the separation of nodes in $\tilde{V}$ after removing the joint communication and computation cut. The potential of a node can be interpreted as its distance to t', where the edges in the cut have unit lengths and the remaining edges have zero lengths. Let p_v indicate the potential of a node. Suppose that $p(s) - p(t') \geq 1$. There is no path between s and t' that only consists of zero-length edges. Therefore, s and t' are disconnected after removing the edges in the cut. Let y_{uv} indicate whether link (u,v) is removed. Let y_w indicate whether the computation resource at node w is removed. The node potential never decreases along a connected path, guaranteed by constraints (11), (12), and (13) when $y_{uv} = 0$ and $y_w = 0$. Disconnected nodes may have different potentials, guaranteed by the same constraints when $y_{uv} = 1$ or $y_w = 1$. If all the constraints are satisfied, s and t' are disconnected, since the potential cannot decrease along a connected path. The cut include the communication links where $y_{uv} = 1$ and computation nodes where $y_w = 1$. Notice that if link (u,v) is removed, no flow can pass through either (u,v) or (u',v'). Therefore, y_{uv} appears in both Eqs. (11) and (12).

$$\min \quad \sum_{(u,v)\in E} \mu_{uv} y_{uv} + \sum_{w \in V} \mu_w y_w \tag{10}$$
$$\text{s.t.} \quad p_v - p_u + y_{uv} \geq 0, \quad \forall(u,v) \in E, \tag{11}$$
$$p_{v'} - p_{u'} + y_{uv} \geq 0, \quad \forall(u,v) \in E, \tag{12}$$
$$-p_w + p_{w'} + y_w \geq 0, \quad \forall w \in V, \tag{13}$$
$$p_s - p_{t'} \geq 1,$$
$$y_{uv} \in \{0,1\}, \quad \forall(u,v) \in E,$$
$$y_w \in \{0,1\}, \quad \forall w \in V.$$

To obtain the minimum computation cut, it suffices to set $y_{uv} = 0$ for all $(u,v) \in E$, and compute the optimal solution to the integer program. To obtain the minimum communication cut, it suffices to set $y_w = 0$ for all $w \in V$, and then compute the optimal solution to the integer program.

Since it is inefficient to compute the optimal solution of an integer program, we next develop a polynomial-time algorithm for evaluating the minimum computation cut, and approxima-

tion algorithms for evaluating the minimum communication cut and the joint cut.

Minimum computation cut: Since a flow needs to be processed by computation nodes along the paths from the source to the destination, removing all the computation resources along $s - t$ paths is sufficient and necessary to reduce the flow to zero. Such computation resources can be identified by computing the intersection of the set of nodes reachable from the source and the set of nodes that can reach the destination. Both sets can be computed by depth first search. The algorithm is summarized as follows, with time complexity $O(|E|)$.

Algorithm 1 Algorithm for evaluating the minimum computation cut for an $s - t$ pair

---

1) Compute the set of nodes V_s such that there exists at least one path from s to every node in V_s.
2) Compute the set of nodes V_t such that there exists at least one path from every node in V_t to t.
3) The minimum computation cut for the $s-t$ pair is $V_s \cap V_t$.

Minimum communication cut: If there is a single computation node u, then the minimum communication cut is the minimum of 1) the minimum cut that disconnects s and u, and 2) the minimum cut that disconnects u and t.

However, if there is more than one computation node, computing the minimum communication cut is NP-hard (Lemma 1). Besides the integer program (10), we develop a 2-approximation algorithm, which runs in polynomial time and outputs a communication cut whose value is at most twice the minimum communication cut.

Algorithm 2 Approximation algorithm for the minimum communication cut for an $s - t$ pair

---

1) Construct a layered graph. Assign an arbitrarily high cost to every link across two layers. Assign μ_{uv} cost to each of the links (u, v) and (u', v').
2) Compute a minimum cut C that separates s and t'.
3) The communication cut is given by links $\{(u, v)|(u, v) \in C$ or $(u', v') \in C\}$.

Theorem 2. *The communication cut obtained by Algorithm 2 has a value that is at most twice the value of the minimum communication cut.*

Proof. Let S^* be the minimum $s-t$ communication cut, which has value w. By Lemma 2, in the layered graph, removing edges $S' = \{(u, v), (u', v')|(u, v) \in S^*\}$ disconnects s and t'. The cost of S' in the layered graph is $2w$.

The minimum communication cut C obtained by Algorithm 2 has value at most $2w$, since C is the minimum $s - t'$ cut in the layered graph and is no larger than S'. Note that C contains no link across the two layers because every crossing link has an arbitrarily high cost. Consequently, $L = \{(u, v)|(u, v) \in C$ or $(u', v') \in C\}$ is a communication cut in the original graph. Furthermore, the cost of removing links L is no more than the cost of removing C. Therefore, L has at most twice the value of the minimum communication cut. $\square$

Minimum joint communication and computation cut: Algorithm 2 can be modified to compute a joint cut whose value is at most twice the minimum joint cut. In the first step of Algorithm 2, instead of assigning an arbitrarily high cost to links across two layers, μ_w cost is assigned to link (w, w'). Using a similar proof to the proof of Theorem 2, we obtain the performance of the modified algorithm.

Theorem 3. *The joint communication and computation cut obtained by the modified algorithm has a value that is at most twice the value of the minimum joint cut.*

C. Relationship between max-flow and min-cuts

The classical max-flow min-cut theorem states that the maximum amount of flow from s to t equals the value of the minimum cut that separates s and t. In a computing network, we study the connections between maximum flow and various types of minimum cuts. Since either communication or computation could be the bottleneck to support a flow, the gap between the maximum flow and the minimum communication cut or the minimum computation cut could be arbitrarily large. For example, Fig. 3 illustrates that the gap between the minimum communication cut and the maximum flow can grow arbitrarily large as the communication bandwidth increases while the computation capacity stays the same, where the numbers adjacent to links and nodes represent the communication capacity and computation capacity, respectively. Similarly, Fig. 4 illustrates that the gap between the minimum computation cut and the maximum flow can be arbitrarily large.
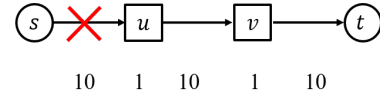

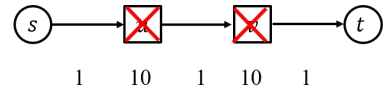Fig. 3: Gap between the max flow and min communication cut: max flow = 2, min communication cut = 10.

Fig. 4: Gap between the max flow and min computation cut: max flow = 1, min computation cut = 20.

Since the joint communication and computation cut include pure communication cut and pure computation cut as special cases, the minimum joint cut is at most the smaller of the two pure cuts. In Fig. 3, the minimum joint cut is 2, by removing the two units computation resources, while in Fig. 4, the minimum joint cut is 1, by removing any one of the three communication links. Note that the joint cut can be smaller than both pure cuts. For example, consider two paths in parallel between s and t, illustrated by Figs. 3 and 4, respectively. The minimum joint cut is 3, while the minimum communication cut is 11 and the minimum computation cut is 22.

The following theorem bounds the gap between the maximum flow and the minimum joint cut.

Theorem 4. *The minimum value (cf. (10)) of the joint communication and computation cut is at most twice the maximum flow between a source-destination pair.*

Proof. In the layered graph, the sum of flows on two copies of a communication link should not exceed the capacity of the link. By relaxing the capacity constraints, and restricting that the flow on each copy of the link should not exceed the capacity of the link, we obtain a *modified* layered graph. Since the sum of flows in the two copies of a link is at most twice the link capacity, the capacity constraints in the original graph are satisfied by reducing the flow by half in the modified layered graph. Therefore, the maximum flow in the modified layered graph is at most twice the maximum flow in the original graph.

The minimum cut in the modified layered graph is the same as the maximum flow in the modified layered graph. The minimum joint cut in the original graph is at most the minimum cut in the modified layered graph, since removing two copies of a link incurs double cost in the modified layered graph and a single cost in the original graph. Therefore, the minimum joint cut in the original graph is at most twice the maximum flow in the original graph. $\qquad\square$

The gap is shown to be tight by the example in Fig. 5. In this computing network, each link has capacity 2. Node v is the only computation node with processing capacity 2. The maximum $s - t$ flow is 1, since the flow has to traverse link (s, t) twice in order to be first processed and then delivered to t. Meanwhile, the minimum $s - t$ joint cut is 2.
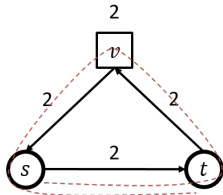
Fig. 5: Gap between the max flow and min joint cut: max flow = 1, min joint cut = 2.

Unlike the classical communication network where links in a minimum $s - t$ cut are saturated by a maximum $s - t$ flow, a computing network may have links and nodes whose capacities are not saturated but still belong to the minimum cut. We next provide examples to support this observation.

Unsaturated node in minimum cut: Consider a computing network represented by Fig. 5, where the processing capacity at node v is reduced to 1.5 and the other capacities remain the same. The maximum flow remains 1. The minimum joint cut is node v, which has value 1.5. However, only one unit processing capacity at v is utilized by the maximum flow, and 0.5 unit processing capacity remains idle.

Unsaturated link in minimum cut: In Fig. 6, the maximum flow remains 1. The minimum cut is link (u, t), which has capacity 1.5 and is not saturated by the maximum flow.

IV. FLOW INTERDICTION

In this section, we study network flow interdiction problems in a computing network. The objective is to minimize the
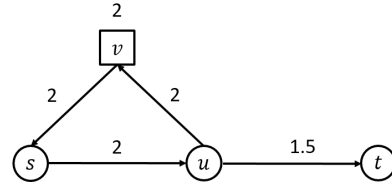
Fig. 6: Link in min cut may not be saturated by max flow.

maximum $s - t$ flow by removing communication links and computation resources using a given budget. We first discuss the complexity of flow interdiction problems in a computing network, and then develop mathematical programs and approximation algorithms to compute optimal and near-optimal interdiction strategies.

A. Flow interdiction variants and complexity

There are two major types of network flow interdiction problems – binary interdiction and partial interdiction. For binary interdiction, the capacity of an interdicted link or a node is removed in full, at a fixed cost. For partial interdiction, the capacity can be removed by a fraction at a fractional cost.

We start by reviewing the complexity of network flow interdiction problems in a communication network, based on the classical flow network model, which imply the complexity of interdiction problems in a computing network. Suppose that the interdiction cost c_{uv} for a link (u, v) is arbitrary, and is independent of its capacity μ_{uv}. The binary interdiction is NP-hard for a communication network, by a reduction from the knapsack problem [11], [15]. Moreover, the binary interdiction remains NP-hard even if every link has one unit interdiction cost [11]. The optimal set of interdicted links belong to some *minimal* cut, and the optimal partial interdiction strategy to attack a minimal cut is greedy in the decreasing value of μ_{uv}/c_{uv}. Partial interdiction, on a network with unit link interdiction cost and an integer interdiction budget, reduces to binary interdiction, and is therefore NP-hard [15].

One special case for network interdiction is that the link interdiction cost equals its capacity. The binary interdiction problem remains NP-hard, by a reduction from the subset sum problem. On the other hand, the partial interdiction problem can be solved in polynomial time, and the optimal interdiction is to interdict the links in any minimum cut. The maximum flow is $\max\{C - B, 0\}$ after the partial interdiction, where C is the minimum cut value and B is the interdiction budget. The same solution can be extended to the problem where link interdiction cost is proportional to its capacity (i.e., $c_{uv} = \alpha\mu_{uv}$, where α is identical for all links).

Since the flow interdiction problem in a computing network includes the flow interdiction problem in a communication network as a special case, the interdiction problem in a computing network is NP-hard for binary interdiction, and for partial interdiction with arbitrary costs. Nevertheless, the optimal partial interdiction is non-trivial even if the interdiction cost equals capacity. The optimal interdiction may not be the minimum cut. For example, in Fig. 6, for budget $B < 1$, the optimal strategy is to interdict link (s, u), while for budget

$B > 1$, the optimal strategy is to interdict link (u, t). The maximum flow under the optimal interdiction is:

$$f = \begin{cases} 1 - 0.5B, & B \leq 1 \\ 1.5 - B, & 1 < B \leq 1.5 \\ 0, & B > 1.5 \end{cases}$$

We further discuss the properties of the optimal partial interdiction when attack cost equals removed capacity. A flow can be decomposed into computation paths. After cycle canceling of the unprocessed flow, and the processed flow, respectively, a flow on a computation path traverses the same link no more than twice, once before processing and once after processing. By removing link capacity, the rate of flow decrease is 0, 0.5, or 1. By removing node capacity, the rate of flow decrease is 0 or 1. The maximum flow after the optimal partial interdiction is a piecewise linear function in the budget.

The function is neither convex nor concave in general, as shown by an example illustrated by Fig. 7. The maximum flow before interdiction is 2, which is carried by computation pathes $(\{s - w - t\}, w)$ and $(\{s - u - v - s - u - t\}, v)$. For interdiction budget $B \leq 1$, the optimal strategy is to attack link (s, w). For $1 < B \leq 2$, the optimal strategy is to attack link (s, u) in addition to (s, w). For $2 < B \leq 2.5$, the optimal strategy is to attack link (u, t) in addition to (s, w). The rates of max flow decrease are 1, 0.5, and 1, respectively.
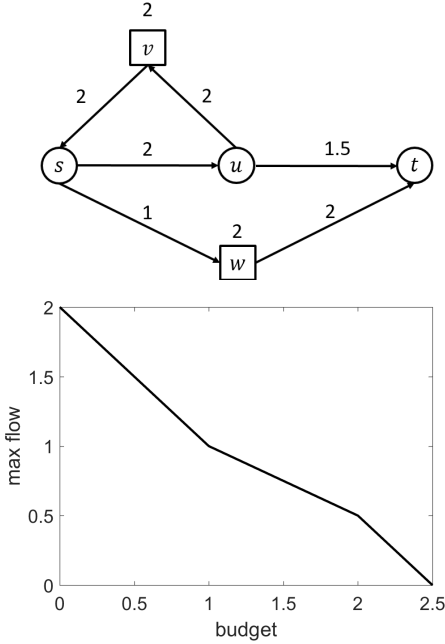

Fig. 7: Example for max flow after optimal interdiction.

B. Exact solutions

We develop mathematical programs to compute the optimal interdiction. The key is to transform the minimax problem (i.e., minimizing the maximum flow) to a minimization problem. Using linear programming duality, the maximum flow is equivalent to the minimum cut without integrality constraints. For binary interdiction, let z_{uv} indicate whether link (u, v) is removed, and let z_w indicate whether the computation resource

at node w is removed. Let c_{uv} denote the cost of removing link (u, v). Let c_w denote the cost of removing the computation resource at node w. Let B denote the interdiction budget. The budget constraint is guaranteed by Eq. (15). The objective (14) minimizes the maximum flow after interdiction. Informally, $\mu_{uv}\beta_{uv}$ indicates the amortized amount of flow contributed by link (u, v), which is zero either by removing (u, v) (i.e., $z_{uv} = 1$) or if link (u, v) is not in a cut (i.e., $p_v = p_u$ and $p'_v = p'_u$). Similarly, $\mu_w\beta_w$ indicates the amortized flow contributed by computation node w. The detailed derivation of this formulation can be found in the Appendix.

$$\min \quad \sum_{(u,v)\in E} \mu_{uv}\beta_{uv} + \sum_{w\in V} \mu_w\beta_w \quad (14)$$

$$\begin{aligned} \text{s.t.} \quad & p_v - p_u + \beta_{uv} + z_{uv} \geq 0, \quad \forall(u,v)\in E \\ & p_{v'} - p_{u'} + \beta_{uv} + z_{uv} \geq 0, \quad \forall(u,v)\in E \\ & -p_w + p_{w'} + \beta_w + z_w \geq 0, \quad \forall w\in V \\ & p_s - p_{t'} \geq 1, \\ & \sum_{(u,v)\in E} c_{uv}z_{uv} + \sum_{w\in V} c_w z_w \leq B, \quad (15) \\ & 0 \leq \beta_{uv} \leq 1, z_{uv} \in \{0,1\}, \quad \forall(u,v)\in E, \\ & 0 \leq \beta_w \leq 1, z_w \in \{0,1\}, \quad \forall w\in V. \end{aligned}$$

We next develop a bilinear program to compute the optimal partial interdiction. We use the same variables to represent costs and capacities as in the integer linear program (14), except that the variables z_{uv} and z_w now denote the fraction of removed link and node capacities. In Eq. (18), $\mu_{uv}(1-z_{uv})$ denotes the remaining transmission capacity of link (u, v), and $\mu_w(1-z_w)$ denotes the remaining processing capacity at node w. The objective is the dual of the maximum flow Eq. (6) with reduced capacities after interdiction. Although there is no integral constraints, the bilinear program is difficult to solve since the objective function is non-convex.

$$\min \quad \sum_{(u,v)\in E} \mu_{uv}(1 - z_{uv})y_{uv} + \sum_{w\in V} \mu_w(1 - z_w)y_w \quad (16)$$

$$\begin{aligned} \text{s.t.} \quad & p_v - p_u + y_{uv} \geq 0, \quad \forall(u,v)\in E, \\ & p_{v'} - p_{u'} + y_{uv} \geq 0, \quad \forall(u,v)\in E, \quad (17) \\ & -p_w + p_{w'} + y_w \geq 0, \quad \forall w\in V, \\ & p_s - p_{t'} \geq 1, \\ & \sum_{(u,v)\in E} c_{uv}z_{uv} + \sum_{w\in V} c_w z_w \leq B, \\ & 0 \leq y_{uv} \leq 1, 0 \leq z_{uv} \leq 1, \quad \forall(u,v)\in E, \\ & 0 \leq y_w \leq 1, 0 \leq z_w \leq 1, \quad \forall w\in V. \end{aligned}$$

C. Approximation algorithms

We develop approximation algorithms based on the sensitivity analysis of the linear program (6). Namely, we study the change of the maximum flow under the changes of link capacity μ_{uv} and node capacity μ_w. Our algorithms are in contrast with previous algorithms for classical flow interdiction based on minimizing the min-cut [15], [11], [14]. Instead,

our algorithms directly work with the max-flow, and the performance is not deteriorated by the non-zero gap between max-flow and min-cut in a computing network.

We start by considering the case where the attack cost equals the removed capacity. The *shadow price* associated with a constraint in a linear program is the rate of change of the objective for one unit change of the right-hand side value of the constraint. Therefore, the shadow price associated with constraint (8) represents the rate of max-flow decrease for each unit processing capacity decrease at node $w \in V$. The shadow price associated with constraint (9) represents the rate of max-flow decrease for each unit transmission capacity decrease at link $(u, v) \in E$. Although the shadow price is a local property for a small change of the right-hand side of the constraint, it gives a conservative estimate of the impact on reducing the max-flow by reducing the capacity of a node or a link. The reason is that the rate of max-flow decrease is a monotone non-decreasing function as the capacity decreases, due to Lemma 3.

Lemma 3. *The maximum $s - t$ flow $F^*(\boldsymbol{\mu})$ given by linear program (6) is a concave function of $\boldsymbol{\mu}$, where $\boldsymbol{\mu}$ is a vector representing link and node capacities.*

Proof. Let $F^*(\boldsymbol{\mu}^i)$ be the maximum $s-t$ flow given a capacity vector $\boldsymbol{\mu}^i$, $\forall i \in \{1, 2\}$. Let $\boldsymbol{f}^*(\boldsymbol{\mu}^i)$ denote the flow vector on each link that support the maximum flow, given capacity $\boldsymbol{\mu}^i$. Consider a capacity vector $\boldsymbol{\mu}^3 = \alpha\boldsymbol{\mu}^1 + (1-\alpha)\boldsymbol{\mu}^2$, where $0 \le \alpha \le 1$. The flow vector $\boldsymbol{f}(\boldsymbol{\mu}^3) = \alpha\boldsymbol{f}^*(\boldsymbol{\mu}^1) + (1 - \alpha)\boldsymbol{f}^*(\boldsymbol{\mu}^2)$ is a feasible solution to the linear program (6), which supports $F(\boldsymbol{\mu}^3) = \alpha F^*(\boldsymbol{\mu}^1) + (1 - \alpha)F^*(\boldsymbol{\mu}^2)$ flow from s to t. The maximum flow $F^*(\boldsymbol{\mu}^3)$ is at least $F(\boldsymbol{\mu}^3)$. Therefore, $F^*(\boldsymbol{\mu})$ is a concave function of $\boldsymbol{\mu}$. $\square$

We propose Algorithm 3 that greedily computes the attack using the shadow price information in linear program (6).

Algorithm 3 Greedy binary interdiction using budget B on a network where interdiction cost equals removed capacity

---

1) Solve linear program (6) and obtain shadow prices for constraints (8) and (9).
2) Choose a link or a node whose capacity is no more than B and is associated with a constraint that has the largest shallow price. Denote the capacity by μ^*.
3) Update B by $B - \mu^*$. Repeat Step 1 until $B \le 0$.

Algorithm 3 can be naturally extended to the partial interdiction case. In Step 2, a link or a node associated with the constraint that has the largest shadow price is chosen. All its capacity is removed if the remaining budget is sufficient, while partial capacity is removed otherwise. The other steps of the algorithm remain the same.

We then develop Algorithm 4 that computes an attack strategy when the interdiction cost is arbitrary, and not necessarily equal to the removed capacity. The algorithm can also be extended to solve the partial interdiction problem, in the same manner as Algorithm 3.

The drawback of Algorithm 4 is that the shadow price is only dependent on the capacity, but not cost. Under arbitrary

---

Algorithm 4 Greedy binary interdiction using budget B on a network with arbitrary interdiction cost

---

1) Solve linear program (6) and obtain shadow prices $\boldsymbol{q}$ for constraints (8) and (9).
2) Choose a link or a node whose capacity is no more than B and is associated with a constraint that has the largest $q_i\mu_i/c_i$, where μ_i is the capacity and c_i is the interdiction cost.
3) Update B by $B - \mu_i$. Repeat Step 1 until $B \le 0$.

---

interdiction cost, it is possible that a cut has a small capacity (i.e., bottleneck for traffic) but a high interdiction cost, in which case it is wise to attack a cut that has a lower interdiction cost but may allow more traffic to go through. The shadow prices associated with links in larger cuts are always zero, since reducing their capacities by a small amount would not reduce the maximum flow.

To overcome this difficulty, we develop a *cost-aware greedy algorithm*. The algorithm is based on the linear program (6), where the capacities $\boldsymbol{\mu}$ in the right-hand side of the constraints are replaced by interdiction costs $\boldsymbol{c}$. The remaining steps are identical to Algorithm 4. The reason of using $\boldsymbol{c}$ as the new capacities is that the linear program serves as an approximation of the minimum-cost cut by relaxing the integral constraints.

V. NUMERICAL RESULTS

In this section, we provide numerical examples to illustrate the applications of our proposed metrics and algorithms to study computing network robustness. First, we study the robustness based on the Abilene network topology in Fig. 8, which has 11 nodes and 14 links. Since we study directed graphs throughout the paper, we consider each edge in the figure as bidirectional links. In the last part of the section, we compare the accuracy and running time of the algorithms on CenturyLink (Level 3) network in U.S., which has 170 nodes and 230 links.
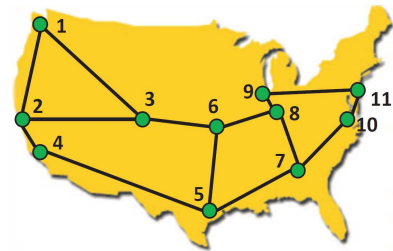

Fig. 8: Abilene network topology.

A. Max-flow and min-cuts

The maximum flow equals the minimum computation cut between an $s-t$ pair if computation resource is the bottleneck to support a network flow. Suppose that each directed link has transmission capacity 1, and that each of nodes 6 and 11 has processing capacity 0.5. The maximum flow between each pair of nodes is 1, which matches the value of minimum computation cut (i.e., nodes 6 and 11).

There may be a non-zero gap between the maximum flow and the minimum communication cut even if communication resource is the bottleneck to support a network flow. Suppose that each processing capacity of nodes 6 and 11 is increased to 5. The minimum communication cut for $s = 8, t = 7$ is 3, while the maximum flow is 2.5. The flow can be decomposed as follows. One unit flow is transmitted through $8-6-5-7$ and processed at 6. One unit flow is transmitted through $8-9-11-10-7$ and processed at 11. Half unit flow is transmitted through $8-7-5-6-8-7$ (or $8-7-10-11-9-8-7$) and processed at 6 (or 11). Part of the flow has to traverse link $8-7$ twice, once before processing and once after processing.

In the above two examples, the minimum joint cut equals the minimum of the pure communication cut and pure computation cut. By setting the processing capacity of nodes 6 and 11 to be 5 and 0.5, respectively, for $s = 8, t = 7$, the minimum joint cut is 2.5, smaller than both the minimum communication cut 3 and the minimum computation cut 5.5. In this example, the maximum $s - t$ flow is 2.25. One feasible decomposition of the flow is one unit flow through $8-6-5-7$ and processed at 6, half unit flow through $8-9-11-10-7$ and processed at 11, half unit flow through $8-9-11-10-7-5-6-8-7$ and processed at 6, and 0.25 unit flow through $8-7-5-6-8-7$ and processed at 6.

B. Flow interdiction

We then study flow interdiction using randomly generated capacities. For simplicity, the capacity of each link is independently and uniformly chosen from $(0, 1)$. The capacity of each node is independently and uniformly chosen from $(0, 0.1)$.

First, we consider the network flow interdiction problem where the cost of interdiction equals the capacity. For $s = 1, t = 2$, the values of max-flow after optimal binary interdiction (solving the integer-linear program), approximate binary interdiction based on Algorithm 3, and approximate partial interdiction based on an extension of Algorithm 3 are presented in Fig. 9. The curve for the optimal binary interdiction is smooth, because computation resource is the bottleneck for the flow from node 1 to node 2 and computation capacity has finer granularity due to the small random number generation range. We observe that the approximate binary interdiction algorithm has good performance. Moreover, the approximate partial interdiction algorithm gives exact solutions, since the slope of the red curve is -1 and thus there is a unit max-flow decrease by removing each unit capacity, which is the maximum possible decrease.

For $s = 1, t = 10$, the values of max-flow after optimal binary interdiction, approximate binary interdiction, and approximate partial interdiction are presented in Fig. 10. The curve for optimal binary interdiction has larger steps, because communication resource is the bottleneck for the flow from node 1 to node 10 and the cost of removing a link is relatively high. The steps in the curve illustrates that the interdiction problem has the same nature as the knapsack problem where the knapsack size represents the interdiction budget and item sizes represent interdiction cost (i.e., link capacity). We observe that Algorithm 3 and its extension still have good performance for both binary and partial interdictions.
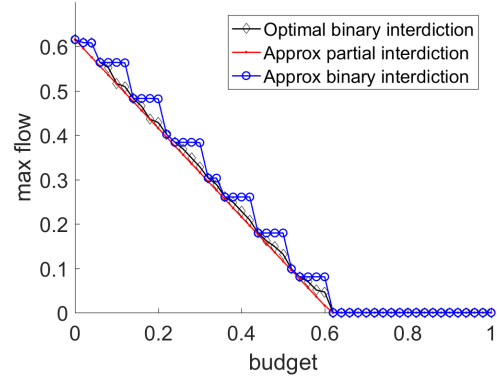

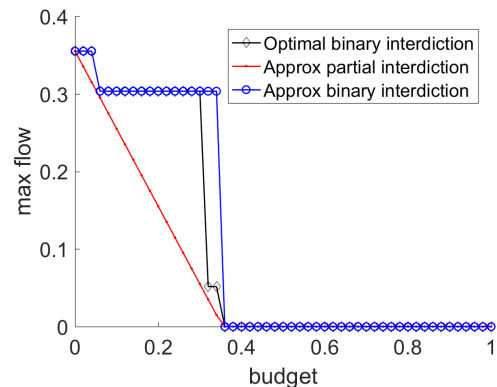Fig. 9: Cost equals capacity, $s = 1, t = 2$.

Fig. 10: Cost equals capacity, $s = 1, t = 10$.

Then, we study the performance of the interdiction algorithms under arbitrary interdiction costs. The cost of removing each link is independently and uniformly chosen from $(0, 1)$. The cost of removing the computation resource at each node is independently and uniformly chosen from $(0, 0.1)$.

For $s = 1, t = 2$, the values of max-flow after the optimal binary interdiction, approximate binary interdiction using Algorithm 4, and approximate partial interdiction using an extension from Algorithm 4 are presented in Fig. 11. The curve for optimal binary interdiction is steeper for small budgets compared with Fig. 9, since it is possible to remove large computation resource at small cost due to the independence between cost and capacity. However, the performance of the greedy algorithms deteriorates. It is worth noting that the objective is not monotone under the greedy algorithm. The reason is that, in the greedy algorithm, a saturated link that has a high cost can be ruled out when the budget is small, which allows a larger cut that has a smaller cost to be removed. Similarly, the performance of the algorithms for $s = 1, t = 10$ is illustrated in Fig. 12.

We next compare the greedy algorithm and the cost-aware greedy algorithm on network interdiction with arbitrary cost. We observe in Fig. 13 that the cost-aware greedy algorithm has significantly better performance for $s = 1, t = 2$. This can be explained by that a large number of computation nodes are attacked under the optimal strategy, and minimizing the attack
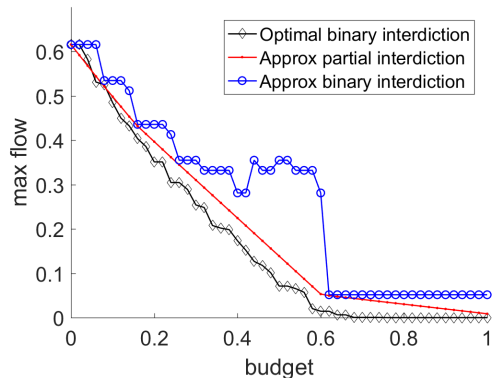
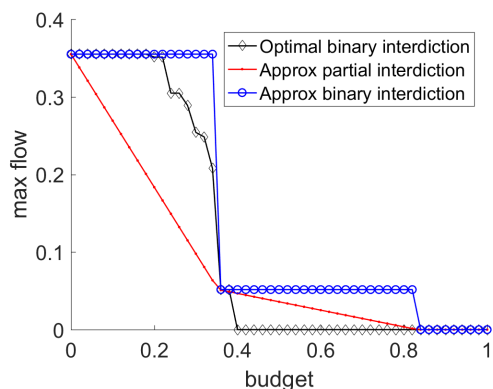Fig. 11: Cost independent of capacity, $s = 1, t = 2$.

Fig. 12: Cost independent of capacity, $s = 1, t = 10$.

cost becomes more important. However, the improvement is not significant for $s = 1, t = 10$, where a small number of links are attacked under the optimal strategy. The cost-aware greedy algorithm rely on the interdiction cost instead of capacity to compute the attacked links, and may attack links whose removal does not have a significant impact on the max-flow.
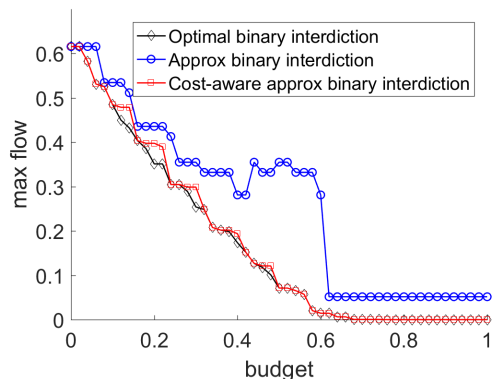

Fig. 13: Comparison of greedy and cost-aware greedy algorithms, cost independent of capacity, $s = 1, t = 2$.

C. Scaling of the algorithms on larger network

Finally, we study the performance of the algorithms by solving interdiction problems on the CenturyLink network illustrated by Fig. 14. We observe that the running time of solving the integer linear program (14) is sensitive to input parameters. The exact solution cannot be obtained within a pre-defined time limit for some problem instances. On the other hand, the greedy algorithms have good performance and have much shorter running time. The detailed results are reported below.
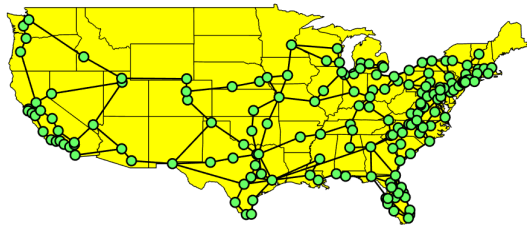

Fig. 14: CenturyLink (Level 3) network topology [26].

Each edge in Fig. 14 represents bidirectional links. The capacity of a link is independently and uniformly chosen from $(0, 10)$. The capacity of a node is independently and uniformly chosen from $(0, 0.1)$. We first study the case where the interdiction cost equals removed capacity. By randomly choosing ten $s - t$ pairs, and using interdiction budget $B \in \{1, 2, 3, 4, 5, 6\}$, the greedy interdiction computed by Algorithm 3 can be obtained in 5 seconds for every scenario. The integer linear program (14) fails to output an optimal solution within 10 minutes for two $s - t$ pairs. Among the scenarios where the optimal solutions are obtained, the running time ranges from 1 second to 4 minutes. The maximum flow after greedy interdiction is on average 7.7% higher than the maximum flow after the optimal interdiction.

For arbitrary interdiction cost, we assume that the cost of removing each link is independently and uniformly chosen from $(0, 10)$, and that the cost of removing the computation resource at each node is independently and uniformly chosen from $(0, 0.1)$. Among ten randomly chosen $s - t$ pairs, the integer linear program (14) fails to output an optimal solution within 10 minutes for nine $s - t$ pairs. The running time of the greedy Algorithm 4 and its cost-aware variant ranges from 1 to 40 seconds. The cost-aware greedy algorithm outperforms the greedy algorithm in 45 out of 60 scenarios, and yields 26% lower maximum flow on average after interdiction.

VI. Conclusion

We studied the robustness of a distributed computing network where traffic flows require communication and computation resources to be transmitted and processed. We defined cut metrics to evaluate network robustness under the failures of communication and computation resources. We developed algorithms to evaluate the max-flow and the min-cuts, and showed a non-zero gap between them. Moreover, we developed algorithms for flow interdiction by removing communication and computation resources within a given budget.

APPENDIX

A. Complexity

Proof of Lemma 1. We first prove that obtaining the minimum $s - t$ communication cut is NP-hard if there are two computation nodes, by a reduction from exact cover by 3-sets. The reduction follows a similar proof in [27] that shows multicut is NP-hard.

The exact cover by 3-sets problems is as follows. Given a set X of $3q$ elements, and a collection C of 3-element subsets of X, is there a subset $K \subseteq C$, such that every element in X appears in exactly one member of K?

We construct a graph from an instance of the exact cover by 3-sets problem. For each 3-set $c_i \in C$, there is a path $s_1 \to u_i \to v_i \to t_1$ from s_1 to t_1. The capacities of links $(s_1, u_i), (u_i, v_i), (v_i, t_1)$ are $k, 2, 1$, respectively. For each element $x \in X$, there is a path from s_2 to t_2. The path contains an edge (u_i, v_i) if the 3-set s_i contains x. All the other edges in the path from s_2 to t_2 have capacity k, except the edges (u_i, v_i).

Finally, the source node s is connected to each of s_1 and s_2 through a link of capacity k. Each of the two nodes t_1 and t_2 is connected to the destination t through a link of capacity k. The only two computation nodes are s_2 and t_1.

Suppose the links adjacent to s and t are not removed. In order for a computation path to connect s and t, either s_1 is connected to t_1, or s_2 is connected to both t_1 and t_2. If there exists an exact cover $K \subseteq C$ for X, a cut S_c can be constructed as follows. The edge (u_i, v_i) is in the cut if $s_i \in K$. The edge (v_j, t_1) is in the cut if $s_j \notin K$. The value of the cut S_c is $2q + (m - q) = m + q$, where $m = |C|$. This is the minimum cut that separates t_1 from s_1, and $\{t_1, t_2\}$ from s_2, for $k \geq 2m$. Therefore, S_c is the minimum communication cut that disconnect all computation paths from s to t.

To conclude, the minimum communication $s-t$ cut is $m+q$ if and only if there exist exact cover by 3-sets for X. The reduction can be done in polynomial time, since there are $O(q+m)$ edges and vertices. The computation of the minimum communication $s - t$ cut is NP-hard.

We illustrate the reduction using an example. Consider an exact cover by 3-sets problem where $X = \{1, 2, 3, 4, 5, 6\}$, $C = \{c_1 = \{1, 2, 3\}, c_2 = \{1, 2, 4\}, c_3 = \{3, 5, 6\}\}$. In this example, $m = 3, q = 2$. There exist an exact cover $K = \{c_2, c_3\}$ for X. The corresponding computing network is shown by Fig. 15. The path $s_1 \to u_i \to v_i \to t_1$ corresponds to the 3-set $c_i, \forall i \in \{1, 2, 3\}$. The path $s_2 \to u_1 \to v_1 \to u_2 \to v_2 \to t_2$ corresponds to elements 1 and 2 that appear in c_1 and c_2. The path $s_2 \to u_1 \to v_1 \to u_3 \to v_3 \to t_2$ corresponds to element 3 that appears in c_1 and c_3. The path $s_2 \to u_2 \to v_2 \to t_2$ corresponds to element 4 that appears in c_2. The path $s_2 \to u_3 \to v_3 \to t_2$ corresponds to elements 5 and 6 that appear in c_3. The thick edges each have capacity k. The numbers adjacent to the other edges indicate their capacities. The red edges $\{(u_2, v_2), (u_3, v_3), (v_1, t_1)\}$ illustrate the minimum computation cut. The value of the minimum computation cut is $5 = m + q$. □
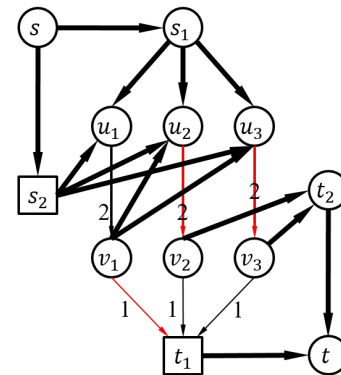

Fig. 15: Reduction from exact cover by 3-sets to minimum communication cut.

B. Correctness of the integer linear program for flow interdiction

We prove the correctness of the integer linear program formulation for flow interdiction. The dual of the maximum flow (linear program (6)) is equivalent to the minimum cut (integer program (10)) without integrality constraints. Let z_{uv} indicate whether link (u, v) is removed. Let z_w indicate whether the computation resource at node w is removed. The maximum flow after removing links where $z_{uv} = 1$ and computation resources at nodes where $z_w = 1$ is represented by Eq. (18). The mathematical program (18) computes the maximum flow after the optimal interdiction with budget B.

$$\min \quad \sum_{(u,v) \in E} \mu_{uv}(1 - z_{uv})y_{uv} + \sum_{w \in V} \mu_w(1 - z_w)y_w \tag{18}$$

$$\begin{aligned}
\text{s.t.} \quad & p_v - p_u + y_{uv} \geq 0, \quad \forall (u, v) \in E, \\
& p_{v'} - p_{u'} + y_{uv} \geq 0, \quad \forall (u, v) \in E, \tag{19}\\
& -p_w + p_{w'} + y_w \geq 0, \quad \forall w \in V, \\
& p_s - p_{t'} \geq 1, \\
& \sum_{(u,v) \in E} c_{uv} z_{uv} + \sum_{w \in V} c_w z_w \leq B, \\
& 0 \leq y_{uv} \leq 1, z_{uv} \in \{0, 1\}, \quad \forall (u, v) \in E, \\
& 0 \leq y_w \leq 1, z_w \in \{0, 1\}, \quad \forall w \in V.
\end{aligned}$$

Since z_{uv} and z_v are binary, the objective can be equivalently represented by Eq. (20), by adding constraints Eqs. (21), (22), (23), (24). To see this, note that if $z_{uv} = 0$, $\mu_{uv}\beta_{uv} \geq \mu_{uv}y_{uv}$. In the optimal solution to the integer linear program (20), $\mu_{uv}\beta_{uv}^* = \mu_{uv}y_{uv}^*$, since $\mu_{uv} \geq 0$. If $z_{uv} = 1$, $y_{uv} - z_{uv} \leq 0$, and $\mu_{uv}\beta_{uv}^* = 0$ in the optimal solution. In both cases, $\beta_{uv}^* \leq 1$. Therefore, the objective $(1 - z_{uv})y_{uv}$ can be transformed to $\mu_{uv}\beta_{uv}$. Similarly, the objective $(1 - z_w)y_w$ can be transformed to $\mu_w\beta_w$. The objective Eq. (20) exactly matches the objective Eq. (18).

$$\min \quad \sum_{(u,v)\in E} \mu_{uv}\beta_{uv} + \sum_{w\in V} \mu_w\beta_w \qquad (20)$$

$$\text{s.t.} \quad p_v - p_u + y_{uv} \ge 0, \quad \forall(u,v)\in E,$$
$$p_{v'} - p_{u'} + y_{uv} \ge 0, \quad \forall(u,v)\in E,$$
$$-p_w + p_{w'} + y_w \ge 0, \quad \forall w\in V,$$
$$p_s - p_{t'} \ge 1,$$
$$\sum_{(u,v)\in E} c_{uv}z_{uv} + \sum_{w\in V} c_w z_w \le B,$$
$$\beta_{uv} \ge y_{uv} - z_{uv}, \quad \forall(u,v)\in E, \qquad (21)$$
$$\beta_w \ge y_w - z_w, \quad \forall v\in V, \qquad (22)$$
$$0 \le \beta_{uv} \le 1, \quad \forall(u,v)\in E, \qquad (23)$$
$$0 \le \beta_w \le 1, \quad \forall w\in V, \qquad (24)$$
$$0 \le y_{uv} \le 1, z_{uv}\in\{0,1\}, \quad \forall(u,v)\in E,$$
$$0 \le y_w \le 1, z_w\in\{0,1\}, \quad \forall w\in V.$$

Finally, we show that the integer linear program (20) has the same optimal solution, if the constraints (21) and (22) are replaced by equality constraints. Suppose that in an optimal solution, $y_{uv}^* - z_{uv}^* \ge 0$. Then $\beta_{uv}^* = y_{uv}^* - z_{uv}^*$ holds in the optimal solution. If $y_{uv}^* - z_{uv}^* < 0$, y_{uv}^* can be increased to z_{uv}^* without violating any constraint and achieves the same cost, such that $\beta_{uv}^* = 0$. Therefore, the constraint (21) can be replaced by an equality constraint. The same analysis holds for replacing constraint (22) by an equality constraint. By replacing $y_{uv} = \beta_{uv} + z_{uv}$ in all the constraints, we obtain the integer linear programming formulation (14).

Remark. The network flow interdiction problem in the classical communication network was formulated as an integer linear program in [11]. We follow a similar approach that use linear programming duality to transform a minimax problem to a minimization problem. The key difference is that the classical minimum cut polytope is integral, and thus it is possible to restrict values of p_v, y_{uv} to be binary in [11]. However, the polytope of Integer program (10) is not integral. Thus, p_v, y_{uv} may take fractional values, which complicates our analysis and makes it non-trivial to extend this formulation to study partial interdiction problems.

References

[1] E. Weise, "Massive Amazon cloud service outage disrupts sites," Feb. 2017. [Online]. Available: http://usat.ly/2m41LPz

[2] W. Qiu, "Submarine cables cut after Taiwan earthquake in Dec 2006," Mar. 2011. [Online]. Available: https://www.submarinenetworks.com

[3] S. Azodolmolky, P. Wieder, and R. Yahyapour, "Cloud computing networking: Challenges and opportunities for innovations," *IEEE Communications Magazine*, vol. 51, no. 7, pp. 54–62, 2013.

[4] P. Gill, N. Jain, and N. Nagappan, "Understanding network failures in data centers: measurement, analysis, and implications," in *ACM SIGCOMM Computer Communication Review*, vol. 41, no. 4. ACM, 2011, pp. 350–361.

[5] M. Charikar, Y. Naamad, J. Rexford, and X. K. Zou, "Multi-commodity flow with in-network processing," *arXiv preprint arXiv:1802.09118*, 2018.

[6] B. Mohammed, M. Kiran, K. M. Maiyama, M. M. Kamala, and I.-U. Awan, "Failover strategy for fault tolerance in cloud computing environment," *Software: Practice and Experience*, vol. 47, no. 9, pp. 1243–1274, 2017.

[7] S. Malik and F. Huet, "Adaptive fault tolerance in real time cloud computing," in *2011 IEEE World Congress on Services (SERVICES)*. IEEE, 2011, pp. 280–287.

[8] M. N. Cheraghlou, A. Khadem-Zadeh, and M. Haghparast, "A survey of fault tolerance architecture in cloud computing," *Journal of Network and Computer Applications*, vol. 61, pp. 81–92, 2016.

[9] N. S. Rao, S. W. Poole, F. He, J. Zhuang, C. Y. Ma, and D. K. Yau, "Cloud computing infrastructure robustness: A game theory approach," in *Computing, Networking and Communications (ICNC), 2012 International Conference on*. IEEE, 2012, pp. 34–38.

[10] F. Chauvel, H. Song, N. Ferry, and F. Fleurey, "Evaluating robustness of cloud-based systems," *Journal of Cloud Computing*, vol. 4, no. 1, p. 18, 2015.

[11] R. K. Wood, "Deterministic network interdiction," *Mathematical and Computer Modelling*, vol. 17, no. 2, pp. 1–18, 1993.

[12] S. R. Chestnut and R. Zenklusen, "Hardness and approximation for network flow interdiction," *Networks*, vol. 69, no. 4, pp. 378–387, 2017.

[13] 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*. Springer, 2003, pp. 51–68.

[14] S. T. McCormick, G. Oriolo, and B. Peis, "Discrete newton algorithms for budgeted network problems," Technical report, RWTH Aachen, Tech. Rep., 2014.

[15] C. A. Phillips, "The network inhibition problem," in *Proceedings of the twenty-fifth annual ACM symposium on Theory of computing*. ACM, 1993, pp. 776–785.

[16] R. Zenklusen, "Network flow interdiction on planar graphs," *Discrete Applied Mathematics*, vol. 158, no. 13, pp. 1441–1455, 2010.

[17] L. R. Ford and D. R. Fulkerson, "Maximal flow through a network," *Canadian journal of Mathematics*, vol. 8, no. 3, pp. 399–404, 1956.

[18] D. Coudert, P. Datta, S. Pérennes, H. Rivano, and M.-E. Voge, "Shared risk resource group complexity and approximability issues," *Parallel Processing Letters*, vol. 17, no. 02, pp. 169–184, 2007.

[19] K. Lee, E. Modiano, and H.-W. Lee, "Cross-layer survivability in WDM-based networks," *IEEE/ACM Transactions on Networking*, vol. 19, no. 4, pp. 1000–1013, 2011.

[20] J. Q. Hu, "Diverse routing in optical mesh networks," *IEEE Transactions on Communications*, vol. 51, no. 3, pp. 489–494, 2003.

[21] S. Neumayer, G. Zussman, R. Cohen, and E. Modiano, "Assessing the vulnerability of the fiber infrastructure to disasters," *IEEE/ACM Transactions on Networking*, vol. 19, no. 6, pp. 1610–1623, 2011.

[22] P. K. Agarwal, A. Efrat, S. K. Ganjugunte, D. Hay, S. Sankararaman, and G. Zussman, "The resilience of wdm networks to probabilistic geographical failures," *IEEE/ACM Transactions on Networking*, vol. 21, no. 5, pp. 1525–1538, 2013.

[23] D. L. Msongaleli, F. Dikbiyik, M. Zukerman, and B. Mukherjee, "Disaster-aware submarine fiber-optic cable deployment for mesh networks," *Journal of Lightwave Technology*, vol. 34, no. 18, pp. 4293–4303, 2016.

[24] S. Neumayer, A. Efrat, and E. Modiano, "Geographic max-flow and min-cut under a circular disk failure model," *Computer Networks*, vol. 77, pp. 117–127, 2015.

[25] J. Zhang, H.-W. Lee, and E. Modiano, "On the robustness of distributed computing networks," in *Proceedings of the International Conference on the Design of Reliable Communication Networks (DRCN)*. IEEE, 2019, pp. 122–129.

[26] CenturyLink (Level 3), "Network map." [Online]. Available: http://www.level3.com/~/media/files/maps/en-network-services-level-3-network-map.ashx

[27] M. Yannakakis, P. C. Kanellakis, S. S. Cosmadakis, and C. H. Papadimitriou, "Cutting and partitioning a graph after a fixed pattern," in *International Colloquium on Automata, Languages, and Programming*. Springer, 1983, pp. 712–722.