Distributed Algorithms for Secure Multipath Routing in Attack-Resistant Networks

Patrick P. C. Lee, Vishal Misra, and Dan Rubenstein

***Abstract*— To proactively defend against intruders from readily jeopardizing single-path data sessions, we propose a *distributed secure multipath solution* to route data across multiple paths so that intruders require much more resources to mount successful attacks. Our work exhibits several important properties that include: (1) routing decisions are made locally by network nodes without the centralized information of the entire network topology, (2) routing decisions minimize throughput loss under a single-link attack with respect to different session models, and (3) routing decisions address multiple link attacks via lexicographic optimization. We devise two algorithms termed the *Bound-Control algorithm* and the *Lex-Control algorithm*, both of which provide provably optimal solutions. Experiments show that the Bound-Control algorithm is more effective to prevent the worst-case single-link attack when compared to the single-path approach, and that the Lex-Control algorithm further enhances the Bound-Control algorithm by countering severe single-link attacks and various types of multi-link attacks. Moreover, the Lex-Control algorithm offers prominent protection after only a few execution rounds, implying that we can sacrifice minimal routing protection for significantly improved algorithm performance. Finally, we examine the applicability of our proposed algorithms in a specialized defensive network architecture called the attack-resistant network and analyze how the algorithms address resiliency and security in different network settings.**

***Index Terms*— Resilience, security, multipath routing, optimization, maximum-flow problems, preflow-push, attack-resistant networks.**

I. Introduction

In conventional routing protocols such as OSPF [29] and RIP [26], a network selects a least-cost path to route data from a source to a sink. While these protocols deliver data efficiently, the use of a single path is vulnerable to general failures and security threats. For instance, intruders can disrupt the data session simply by attacking one of the intermediate links along the utilized path. This singularity enables intruders to readily devote their resources to attacking the only path.

Such networks can be protected via a *secure multipath approach* in which data are dispersed across multiple paths destined for the sink. Each path conveys a portion of data from the source, and the sink assembles the data fragments received from the various paths. If some paths fail to deliver data, then as long as the scale of failure is modest, the sink can still recover *all* data using redundant routing [27] or threshold secret sharing [25]. Therefore, to successfully compromise the data session, intruders must subvert a sufficient number of routing paths and hence require more resources than those needed to attack a single path. We point out that using multiple paths can complicate the packet-reordering problem [31]. However, it can be remedied via sophisticated coding solutions (e.g., [8]) for non-real-time data transfers or standard pre-buffering techniques (e.g., [24]) for real-time data transfers. In addition, more recent application-layer architectures such as overlay networks (e.g., RON [3] and SOS [20]) provide a more promising platform for deploying multipath routing as compared to conventional layer-3 architectures. Therefore, it is feasible to adopt the secure multipath approach to proactively accomplish routing resilience.

One major challenge is to design a *distributed* solution that implements the process of selecting the "best" data allocation across multiple paths through a network. The distributed solution enhances traditional centralized solutions for secure multipath routing such as [4], [6], [18] in different ways. First, it does not require any network node to have full knowledge of the entire network topology. It is therefore adequate for decentralized peer systems, such as RON [3], whose nodes are located in different domains and are often administered independently. In addition, it allows network nodes to locally decide security costs, bandwidth constraints, and choices of routes, and thus improves flexibility as compared to the centralized approach.

To characterize the "best" data allocation across multiple paths, our primary security objective is to minimize the maximum damage incurred by a *single-link attack* (or failure), i.e., an intruder compromises data along a single link in a given network. There are two reasons to justify our preliminary analysis on a single-link attack. First, there are many attack and failure scenarios where a single-link failure is likely to cause the majority of problems, as the network can often be repaired, or routes are adjusted, to account for the failure before a subsequent outage occurs. Nevertheless, we still want to mitigate the damage of a single-link failure since it can cause severe throughput loss in a high-speed network within only a few seconds. For example, a 10-second outage of an OC-48 link can incur a loss of 3 million 1-KB packets [23]. Second, our experiments show that our solution that is designed for preventing a single-link attack provides substantial resilience to multiple simultaneous attacks as well. Thus, our analysis can serve as a baseline for future work that focuses on multi-link attacks.

Unlike traditional load-balancing solutions that minimize the maximum link utilization (i.e., the maximum ratio of the link throughput to the link bandwidth), our objective is to guarantee resilience using all available network resources. Given different session requirements, we seek to minimize the worst-case single-link attack while attaining the desired

P. Lee and V. Misra are with the Dept of Computer Science, Columbia University (emails: {pclee,misra}@cs.columbia.edu).

D. Rubenstein is with the Dept of Electrical Engineering, Columbia University (email: danr@ee.columbia.edu).

A conference version of this paper appeared in IEEE INFOCOM '05 [22].

throughput rates with the provisioned network bandwidth.

In this paper, we devise a distributed secure multipath solution that determines the multipath routes to maximize the security with respect to single-link attacks. Our work is suitable for two session models, namely:

- *Fixed-rate session*: a session that wishes to send data from the source to the sink at a pre-determined rate; and
- *Maximal-rate session*: a session that wishes to send data from the source to the sink at the fastest rate allowed by all available paths in the underlying network.

Given the above session models, we first propose a distributed solution called the *Bound-Control algorithm*, which provably minimizes the maximum throughput loss when a link is attacked. We formulate this solution as a maximum-flow problem that can be solved in a distributed fashion based on the extension of the Preflow-Push algorithm [16].

Using the Bound-Control algorithm as a building block, we devise a higher-complexity, but more resilient distributed solution called the *Lex-Control algorithm*. It defends not only against the worst-case link attack, but also against link attacks that do not cause the worst damage but are still severe (e.g., the second and third worst-case link attacks). To achieve this property, the Lex-Control algorithm scatters the costs incurred by the link attacks as evenly as possible over all the links in a network, or equivalently solves a *lexicographic-optimization* problem [13], in a distributed manner.

By simulation, we evaluate the resilience of the Bound-Control and Lex-Control algorithms against different types of attacks on single or multiple links. In comparison to single-path alternatives, our results indicate that the Bound-Control algorithm substantially decreases the cost of the worst-case single-link attack (e.g., by 78% in a 200-node, 1000-link network). Also, the Lex-Control algorithm can further reduce, by more than 50%, the number of links that incur severe damage due to single-link attacks, and such reduction is realized after only three or four iterations. While the resilience enhancement of the Lex-Control algorithm over the Bound-Control algorithm comes at the expense of higher complexity, our simulation results show that we can limit this increase in complexity without much loss in resilience by executing only the first few iterations of the Lex-Control algorithm.

Finally, we demonstrate the applicability of both Bound-Control and Lex-Control algorithms in an *attack-resistant network* (e.g., SOS [20]), a specialized network that protects end hosts with a defensive architecture. Using [7] as our foundation, we analyze how our proposed multipath algorithms can be deployed to provide routing resilience and in the meantime secure the network against malicious attacks.

The paper proceeds as follows. In Section II, we formulate the secure multipath approach. Sections III and IV present the Bound-Control and Lex-Control algorithms, respectively. In Section V, we report several experiments that evaluate the algorithms under different classes of link attacks. Section VI discusses how to apply our algorithms in an attack-resistant network and presents simulation results of their performance. Section VII reviews related work. Section VIII discusses the practical issues of our work and suggests future directions. Section IX concludes.

TABLE I

MAJOR NOTATION USED IN THIS PAPER.

Defined in Section II:	
$\mathcal{N}$	set of nodes
$\mathcal{L}$	set of links
$\mathcal{G}$	network $(\mathcal{N}, \mathcal{L})$
s	source node
t	sink node
$\mathcal{L}(u)$	set of outgoing links $l \in \mathcal{L}$ of node $u \in \mathcal{N}$
X	session throughput from source s to sink t
x_l	proportion of session data carried by link $l \in \mathcal{L}$
$\mathbf{x}$	proportion vector $(x_l, l \in \mathcal{L})$
c_l	security constant of link $l \in \mathcal{L}$
a_l	attack cost $c_l x_l$ of link $l \in \mathcal{L}$
a^*	minimized worst-cast attack cost
$cap(l)$	capacity of link $l \in \mathcal{L}$ in maximum-flow problems
f_l	flow of link $l \in \mathcal{L}$ in maximum-flow problems
$\mathbf{f}$	flow vector $(f_l, l \in \mathcal{L})$ in maximum-flow problems
f^*	resulting maximum-flow value
B_l	bandwidth of link $l \in \mathcal{L}$
b_l	fraction bound of link $l \in \mathcal{L}$
$\mathbf{a}$	non-increasing attack-cost sequence
$\mathbf{a}^*$	lexicographically optimized $\mathbf{a}$
Defined in Sections III and IV:	
f_s	flow value broadcast by source s
U	sufficiently large value
$\mathcal{G}_{f^*}$	residual network with respect to f^*
Defined in Section VI:	
$\mathcal{A}$	set of access points (APs)
$\mathcal{T}$	set of targets
$\mathcal{P}$	set of paths between APs and targets
$\mathcal{G}_a$	attack-resistant network $(\mathcal{A}, \mathcal{T}, \mathcal{P})$
$\mathcal{A}(j)$	set of APs from which target $j \in \mathcal{T}$ is reachable
$\mathcal{T}(i)$	set of targets that can be reachable from AP $i \in \mathcal{A}$
C_u	event that node $u \in \mathcal{A} \cup \mathcal{T}$ is compromised
D_u	event that node $u \in \mathcal{A} \cup \mathcal{T}$ is under DoS attacks
$P(E)$	probability that event E occurs
p_i	blocking probability of AP $i \in \mathcal{A}$

II. PROBLEM FORMULATION

In this section, we formalize the secure multipath approach as a minimax-optimization problem and hence its equivalent maximum-flow problem. This formulation will also be used later when we include link-bandwidth constraints and lexicographic optimization. Note that the following formulation is generally based on [1], [2], [4], [5], [13], [15], [16], [25]. To aid our discussion, Table I summarizes the major notation that we use in this paper.

Our discussion relies on the concepts of the maximum flow and the minimum cut [2]. Given a network with a number of nodes and links, the *maximum-flow problem* is to determine the maximum flow that can be sent from a source node s to a sink node t subject to the capacity constraints (i.e., each link has flow bounded by the link capacity) and the flow-conservation constraints (i.e., the net flow entering any node except the source and the sink equals zero). Suppose that we partition the nodes into two sets $\mathcal{S}$ and $\overline{\mathcal{S}}$, where $s \in \mathcal{S}$ and $t \in \overline{\mathcal{S}}$. A *cut* refers to the set of links directed from $\mathcal{S}$ to $\overline{\mathcal{S}}$. A *minimum cut* is the cut that has the minimum capacity (i.e., the minimum sum of capacities of all links in the cut). The *max-flow min-cut theorem* states that the maximum-flow value equals the capacity of the minimum cut.

We are interested in a connected, directed, and acyclic network that is viewed as a graph $\mathcal{G} = (\mathcal{N}, \mathcal{L})$, where $\mathcal{N}$ is the set of nodes and $\mathcal{L}$ is the set of directed links. Our analysis is based on a single data session with a source node s and a sink node t. We emphasize that our analysis can be generalized to a homogeneous class of multiple data sessions by mapping source s and sink t to the ingress and egress points of the network, respectively. Suppose that source s sends data to sink t with a session throughput given by X (say, in Mb/s). We let x_l, $0 \leq x_l \leq 1$, be the *proportion* of the entire session data carried by link $l \in \mathcal{L}$ (i.e., x_l equals the throughput of link l divided by X) and let $\mathbf{x} = (x_l, l \in \mathcal{L})$ be the corresponding proportion vector.

Our analysis mainly focuses on a single-link attack, but we also address a single-node attack in Section VI. In this paper, we focus on a threat model in which the damage due to the attack on link $l \in \mathcal{L}$ not only is proportional to the throughput sent over link l, but also depends on other factors such as the likelihood that an attack can successfully bring down link l. We characterize such damage as an *attack cost* $a_l = c_l x_l$, where c_l, which we term the *security constant* of link l, specifies the vulnerability of link l. Intuitively, the attack cost is used to measure the scale of throughput loss due to a single-link attack. Note that c_l can have several physical interpretations, such as the probability that link l is successfully attacked given that the intruder attempts to attack link l [5], the failure probability of link l [4], or the proportion of loss of data traversing link l when it is attacked. To ensure that every link l has a consistent interpretation of c_l, every node has to calibrate c_l with respect to an agreed-upon definition of an attack. Also, to enable us to interpret c_l as a probability or proportion, we require that $0 \leq c_l \leq 1$ for every link $l \in \mathcal{L}$. With an agreed-upon attack model, every node u can then determine in advance c_l for each of its own outgoing links $l \in \mathcal{L}(u)$, where $\mathcal{L}(u)$ is the set of all outgoing links of node u, using vulnerability modeling [10], statistical measurements of reliability indexes [15], or security monitoring systems [25]. We point out that if an accurate estimate of c_l is not available, we can set $c_l = 1$, meaning that link l has all its data lost when it is under attack, and our analysis still applies to this worst-case scenario.

A. Minimax Optimization

To mitigate the worst damage due to a single-link attack, our objective is to decide a feasible proportion vector $\mathbf{x}$ that *minimizes the maximum attack cost* over all links in the network. This can be viewed as the following *minimax* optimization problem[1]:

$$\begin{aligned} a^* &= \min_{\mathbf{x}} \max_{l \in \mathcal{L}} a_l = \min_{\mathbf{x}} \max_{l \in \mathcal{L}} c_l x_l \\ \text{subject to} &\quad 0 \leq x_l \leq 1, \ \forall l \in \mathcal{L}. \end{aligned} \tag{1}$$

Problem 1 can be solved in polynomial time via linear programming, but this is a centralized solution and requires the information of the entire network topology. To implement a distributed solution, we can first transform the problem into a maximum-flow problem by setting the capacity of every link l, denoted by $cap(l)$, as the reciprocal of c_l [1], and then solve for the maximum flow using the distributed *Preflow-Push algorithm* [16], which is summarized as follows. Source s first initiates the algorithm by pushing the maximum possible flow to its neighbor nodes. All nodes except source s and sink t then attempt to push the flow toward sink t along the estimated shortest paths until the resulting maximum flow reaches sink t. Any excess flow is pushed back to source s. In [16], it explains how to implement the Preflow-Push algorithm in a distributed and asynchronous fashion. We refer readers there for a detailed discussion. For completeness, we include the pseudo-code of the Preflow-Push algorithm in Appendix I.

Let $\mathbf{f} = (f_l, l \in \mathcal{L})$ be the flow vector where f_l denotes the flow of link l, and f be the net flow entering sink t. Problem 1 can thus be mapped to the following maximum-flow problem:

$$\begin{aligned} f^* &= \max_{\mathbf{f}} f \\ \text{subject to} &\quad 0 \leq f_l \leq 1/c_l, \ \forall l \in \mathcal{L}, \end{aligned} \tag{2}$$

where the solutions to Problems 1 and 2 are related by $a^* = 1/f^*$ and $x_l = f_l/f^*, \ \forall l \in \mathcal{L}$.

To illustrate both problems, Figure 1(a) depicts a network where $c_l = 1$ for all links l. From the Preflow-Push algorithm, we know the maximum flow is $f^* = 2$ and thus the worst-case attack cost is minimized at $a^* = 0.5$. Also, the algorithm returns the corresponding vectors $\mathbf{f}$ and $\mathbf{x}$.

B. Minimax Optimization with Bandwidth Constraint

One limitation of Problem 1 is that every link is assumed to have infinite bandwidth so that it can accommodate the entire session data. To incorporate the *link-bandwidth constraints*, we assume that each node u specifies a priori a bandwidth B_l (say, in Mb/s) for its outgoing links $l \in \mathcal{L}(u)$. We let $b_l = \min(B_l/X, 1)$, where $0 \leq b_l \leq 1$, denote the *fraction bound* of link l that bounds from above the *proportion* of data that can be sent through link l for a given session throughput X. We then incorporate the fraction bound into Problem 1 as:

$$\begin{aligned} a^* &= \min_{\mathbf{x}} \max_{l \in \mathcal{L}} a_l = \min_{\mathbf{x}} \max_{l \in \mathcal{L}} c_l x_l \\ \text{subject to} &\quad 0 \leq x_l \leq b_l, \ \forall l \in \mathcal{L}. \end{aligned} \tag{3}$$

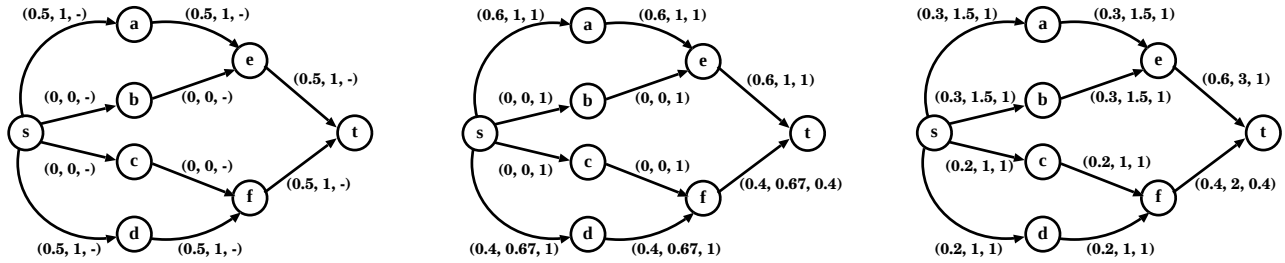
The corresponding maximum-flow problem becomes:

$$\begin{aligned} f^* &= \max_{\mathbf{f}} f \\ \text{subject to} &\quad 0 \leq f_l \leq \min(1/c_l, b_l f), \ \forall l \in \mathcal{L}. \end{aligned} \tag{4}$$

For clarity, the term *bandwidth* (i.e., B_l) represents the maximum amount of data that can be sent across a link, and the term *capacity* (i.e., $cap(l) = \min(1/c_l, b_l f)$) denotes the upper bound of the link flow in the transformed maximum-flow problem. While the bandwidth B_l is fixed, the capacity $cap(l)$ varies depending on the flow value f that reaches sink t.

Figure 1(b) depicts the case where we assign the fraction bound $b_l = 0.4$ to the link from node f to sink t and $b_l = 1$ to the rest. Similar to Problem 1, we can solve

[1]All problems presented in this paper are under flow-conservation constraints, although the convention is omitted for brevity.



(a) $a^* = 0.5$ (b) $a^* = 0.6$ (c) $a^* = 0.6$

Fig. 1. Optimal solutions to the three optimization problems: (a) minimax optimization, (b) minimax optimization with the bandwidth constraints, and (c) lexicographic optimization. Every link l has $c_l = 1$ and is associated with a triple (x_l, f_l, b_l), where x_l and f_l are the solutions after the optimization problems are solved, and b_l (defined for (b) and (c) only) denotes the *initial* fraction bound assigned to link l. Note that b_l is different from its initial value after the lexicographic-optimization problem is solved (see Section IV and Figure 3 for details).

Problem 3 in a centralized manner via linear programming. To implement a distributed approach, in Section III, we develop the *Bound-Control algorithm*, which is built upon the Preflow-Push algorithm to solve Problem 4 and hence Problem 3.

C. Lexicographic Optimization

A limitation of the previous problems is that they are concerned only with how to minimize the worst-case attack cost, but do not attempt to reduce the costs of severe link attacks. For example, in Figures 1(a) and 1(b), the attack costs are unevenly distributed. Specifically, in Figure 1(b), there are six links whose attack costs are at least 0.4 each. By evenly distributing the costs as shown in Figure 1(c), only two such links exist. Thus, we reduce the number of links where the single-link attacks can lead to severe damage.

To formalize the concept of the even distribution of attack costs, we let $\mathbf{a} = \langle c_{l_1} x_{l_1}, c_{l_2} x_{l_1}, \cdots, c_{l_{|\mathcal{L}|}} x_{l_{|\mathcal{L}|}} \rangle$, where $l_1, l_2, \cdots, l_{|\mathcal{L}|} \in \mathcal{L}$, be a non-increasing attack-cost sequence. The distribution of the attack costs is said to be the *most even* if the associated attack-cost sequence $\mathbf{a}$ is *lexicographically minimized*, i.e., for any other non-increasing attack-cost sequence $\mathbf{a}' = \langle c_{l_1} x'_{l_1}, c_{l_2} x'_{l_2}, \cdots, c_{l_L} x'_{l_{|\mathcal{L}|}} \rangle \neq \mathbf{a}$, there exists i, where $1 \leq i < |\mathcal{L}|$, such that $c_{l_j} x_{l_j} = c_{l_j} x'_{l_j}$ for $j < i$ and $c_{l_i} x_{l_i} < c_{l_i} x'_{l_i}$. Let $\operatorname{lexmin}(.)$ be the function that returns the lexicographically minimum sequence $\mathbf{a}^*$. We then express the lexicographic-optimization problem as:

$$\begin{aligned} \mathbf{a}^* &= \operatorname*{lexmin}_{\mathbf{x}} \mathbf{a} = \operatorname*{lexmin}_{\mathbf{x}} \langle c_{l_1} x_{l_1}, \cdots, c_{l_L} x_{l_L} \rangle \\ \text{subject to} \quad & \mathbf{x} = \arg\min_{\mathbf{x}} \max_{l \in \mathcal{L}} c_l x_l, \\ & 0 \leq x_l \leq b_l, \ \forall l \in \mathcal{L}. \end{aligned} \tag{5}$$

Hence, the corresponding maximum-flow problem is:

$$\begin{aligned} \mathbf{a}^* &= \operatorname*{lexmin}_{\mathbf{f}} \mathbf{a} = \operatorname*{lexmin}_{\mathbf{f}} \langle \frac{c_{l_1} f_{l_1}}{f}, \cdots, \frac{c_{l_L} f_{l_L}}{f} \rangle \\ \text{subject to} \quad & \mathbf{f} = \arg\max_{\mathbf{f}} f, \\ & 0 \leq f_l \leq \min(1/c_l, b_l f), \ \forall l \in \mathcal{L}. \end{aligned} \tag{6}$$

In Section IV, we propose the *Lex-Control algorithm* to address this problem. By extending the Bound-Control algorithm and setting the fraction bounds of the links appropriately, the Lex-Control algorithm can determine the lexicographically optimal solutions for Problems 5 and 6 in a distributed fashion. This type of lexicographic-optimization problem was first analyzed in [13], from which our Lex-Control algorithm has two main distinctions. First, while the analysis in [13] assumes no link-bandwidth constraint, we explicitly incorporate this constraint into our algorithm. Furthermore, our algorithm allows distributed implementation, while the solution in [13] is centralized and requires the knowledge of the whole network state.

III. Bound-Control Algorithm

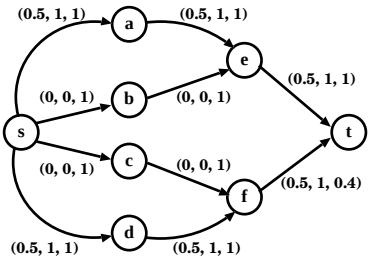
This section presents the *Bound-Control algorithm*, which solves Problems 3 and 4, in which a fraction bound b_l is imposed on every link $l \in \mathcal{L}$. We describe how it operates and how it supports both fixed-rate and maximal-rate session models described in Section I. We refer readers to Appendix II-A for the proof of its correctness.

Here, we let f_s be the flow value that source s broadcasts to the network in the Bound-Control algorithm. We also let U be a sufficiently large value that approximates infinity. For instance, U can be the largest value that can be processed by the implementation.

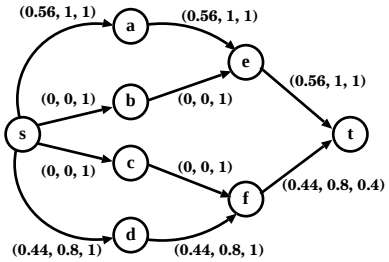
A. Description of the Bound-Control Algorithm

The idea of the Bound-Control algorithm is to repeatedly solve a maximum-flow problem via the Preflow-Push algorithm and adjust the link capacities until the maximum-flow result converges to the optimal solution. The Bound-Control algorithm is presented in Algorithm 1.

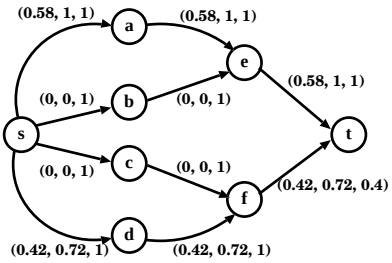
In Algorithm 1, source s first broadcasts a sufficiently large value $f_s = U$ to initiate the Bound-Control algorithm (line 1). Next, all network nodes execute the Preflow-Push algorithm subject to the link-capacity constraint $cap(l) = \min(1/c_l, b_l f_s) = 1/c_l$ for every link $l \in \mathcal{L}$ (lines 2-5). By checking the amount of flow that has been sent out, source s can determine the maximum-flow result. Source s then broadcasts the computed maximum-flow result represented by f_s to the network (lines 7-8) so that every network node can adjust the capacities of its outgoing links (lines 9-11). Afterward, all nodes execute again the Preflow-Push algorithm under the new link capacities (line 12). The algorithm iterates in the repeat-until loop (lines 7-12), and terminates if the



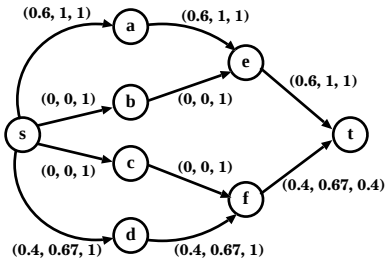
(a) 1st Prefbw-Push: maximum fbw = 2



(b) 2nd Prefbw-Push: maximum fbw = 1.8



(c) 3rd Prefbw-Push: maximum fbw = 1.72



(d) Optimal solution: $f^* = 1.67, a^* = 1/f^* = 0.6$

Fig. 2. Example of the Bound-Control algorithm in Algorithm 1 for the network shown in Figure 1. Every link l has $c_l = 1$ and is associated with a triple (x_l, f_l, b_l). The figures illustrate: (a)-(c) the fbw values after the first three executions of the Prefbw-Push algorithm (lines 5 and 11) and (d) the optimal solution returned from the Bound-Control algorithm.

Algorithm 1 Bound-Control

1: source s broadcasts $f_s = U$ to all nodes $u \in \mathcal{N}$
2: **for all** $u \in \mathcal{N}$ **do**
3: **for all** $l \in \mathcal{L}(u)$ **do**
4: node u sets $cap(l) = \min(1/c_l, b_l f_s)$
5: all nodes run *Prefbw-Push*
6: **repeat**
7: source s sets f_s to be the maximum-fbw result
8: source s broadcasts f_s to all nodes $u \in \mathcal{N}$
9: **for all** $u \in \mathcal{N}$ **do**
10: **for all** $l \in \mathcal{L}(u)$ **do**
11: node u sets $cap(l) = \min(1/c_l, b_l f_s)$
12: all nodes run *Prefbw-Push*
13: **until** source s finds that f_s equals the maximum-fbw result

maximum flow obtained from the Preflow-Push algorithm equals the flow value f_s that has just been broadcast (line 13). The optimal value f^* is given by f_s. Figure 2 illustrates how the Bound-Control algorithm works.

B. Discussion of the Bound-Control Algorithm

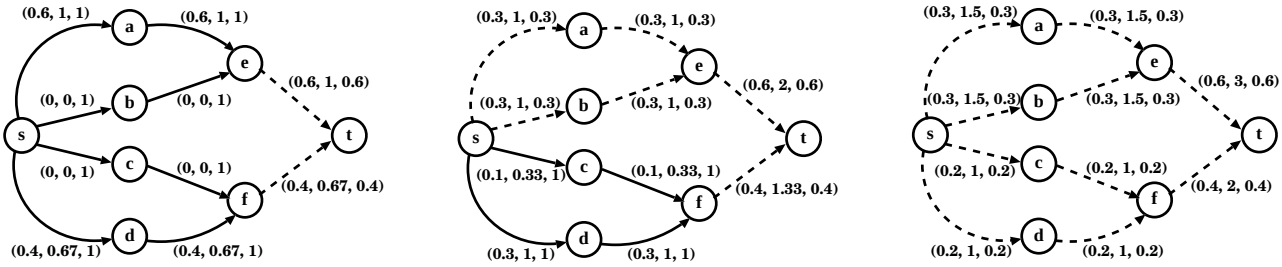
In actual implementation, we can support both fixed-rate and maximal-rate session models (see Section I) by determining the feasible session throughput X and hence the fraction bound b_l in a *distributed* fashion. Source s first initiates the Preflow-Push algorithm to decide the feasible session throughput X subject to the bandwidth constraint B_l for all $l \in \mathcal{L}$, and then broadcasts X to all the nodes in the network so that they can specify the fraction bound b_l for their associated links l. The fixed-rate session model is thus provided by sending data at the fixed rate X. If X is the maximum flow returned from the Preflow-Push algorithm, then we can achieve the maximum security under the maximum session throughput using the Bound-Control algorithm. Thus, the maximal-rate session model is supported.

We can further enhance the efficiency of the implementation of the Bound-Control algorithm via *bisection search* to locate the optimal value f^* in the Bound-Control algorithm as follows. Suppose that f_{low} and f_{high} denote the lower and upper bounds, respectively. Source s first initializes f_{low} to be zero and f_{high} to be twice the maximum-flow result determined by the first execution of the Preflow-Push algorithm (i.e., line 5 of Algorithm 1). It then broadcasts $f_s = (f_{low} + f_{high})/2$ to the network. If the next execution of the Preflow-Push algorithm returns the maximum flow less than f_s, then source s assigns the maximum-flow result to f_{high}. Otherwise, the result is assigned to f_{low} instead. Source s repeatedly searches for f_s, and the algorithm terminates if the most recently broadcast value f_s and the latest maximum-flow result are equal (or different by some tolerance value depending on the implementation).

With bisection search, the complexity of the Bound-Control algorithm is $O(pT)$, where p is the number of precision digits describing all possible flow values and T is the complexity of executing the Preflow-Push algorithm. For instance, if the Bound-Control algorithm implements the distributed and asynchronous version of the Preflow-Push algorithm [16], it introduces $O(p|\mathcal{N}|^2|\mathcal{L}|)$ messages and takes $O(p|\mathcal{N}|^2)$ time to converge.

IV. Lex-Control Algorithm

In this section, we present the Lex-Control algorithm, which solves the lexicographic optimization specified in Problems 5 and 6. We explain how the Lex-Control algorithm is extended



(a) 1st Bound-Control: maximum fbw=1.67 (b) 2nd Bound-Control: maximum fbw=3.33 (c) 3rd Bound-Control: maximum fbw=5

Fig. 3. Example of the Lex-Control algorithm in Algorithm 2 for the network shown in Figure 1. Every link l has $c_l = 1$ and is associated with a triple (x_l, f_l, b_l). After every execution of the Bound-Control algorithm (lines 1 and 11), the nodes identify the critical links (in dashed arrows) and adjust the fraction bounds b_l accordingly (lines 6-10).

from the Bound-Control algorithm. Its proof of correctness can be found in Appendix II-B.

A. Description of the Lex-Control Algorithm

To understand the Lex-Control algorithm, suppose that for a particular maximum-flow problem, we have found the maximum flow f^* and the minimized worst-case attack cost $a^* = 1/f^*$. The network will then constitute a set of *critical links*, defined as the links $l \in \mathcal{L}$ whose attack costs cannot be further decreased without increasing a^*. The idea of the Lex-Control algorithm is to iteratively solve a maximum-flow problem using the Bound-Control algorithm and identify additional critical links until the lexicographically optimal solution $\mathbf{a}^*$ is obtained.

Before describing the algorithm, we define the *residual network* $\mathcal{G}_{f^*} = (\mathcal{N}, \mathcal{L}_{f^*})$ with respect to the maximum flow f^* as follows [2]. Suppose that the maximum flow f^* is solved and each link $l \in \mathcal{L}$ carries a flow f_l. To construct $\mathcal{L}_{f^*}$, for each link $l \in \mathcal{L}$ directed from node u to node v, where $u, v \in \mathcal{N}$, if $cap(l) - f_l > 0$, we include a forward link from u to v into $\mathcal{L}_{f^*}$, and if $f_l > 0$, we include a backward link from v to u into $\mathcal{L}_{f^*}$.

Algorithm 2 Lex-Control

```
1: all nodes run Bound-Control
2: source s sets f* to be the computed maximum fbw
3: while f* < U do
4:    source s broadcasts f* to all nodes u ∈ N
5:    for all u ∈ N do
6:       node u runs a connectivity-checking algorithm on G_{f*}
7:       for all l ∈ L(u) do
8:          if l is a critical link then
9:             node u sets c_l = 1/U
10:            node u sets b_l = f_l/f*
11:   all nodes run Bound-Control
12:   source s sets f* to be the computed maximum fbw
```

Algorithm 2 summarizes the Lex-Control algorithm. All nodes first run the Bound-Control algorithm to minimize the worst-case attack cost subject to the capacity constraint $cap(l) = \min(1/c_l, b_l f)$ for all $l \in \mathcal{L}$ in the transformed maximum-flow problem (line 1). Source s then broadcasts the computed maximum flow f^* (line 4). Each node runs a connectivity-checking algorithm (e.g., the breadth-first search) on $\mathcal{G}_{f^*}$ (lines 6-8). If its neighbors in $\mathcal{G}$ are not reachable in $\mathcal{G}_{f^*}$, then the corresponding links between itself and its neighbors in $\mathcal{G}$ are lying on a minimum cut and hence are critical (see Appendix II-B). It modifies c_l and b_l for each spotted critical link l (lines 9-10) so that $cap(l)$ is adjusted to bound only the proportion of flow currently carried. Here, we set $1/c_l$ to be a sufficiently large value U (defined in Section III) so that it does not affect $cap(l)$. The algorithm iteratively identifies the critical links (lines 3-12, collectively defined as a *lexicographic iteration*), and terminates when the maximum flow computed from the Bound-Control algorithm equals U. Figure 3 depicts how the Lex-Control algorithm computes the lexicographically optimal solution.

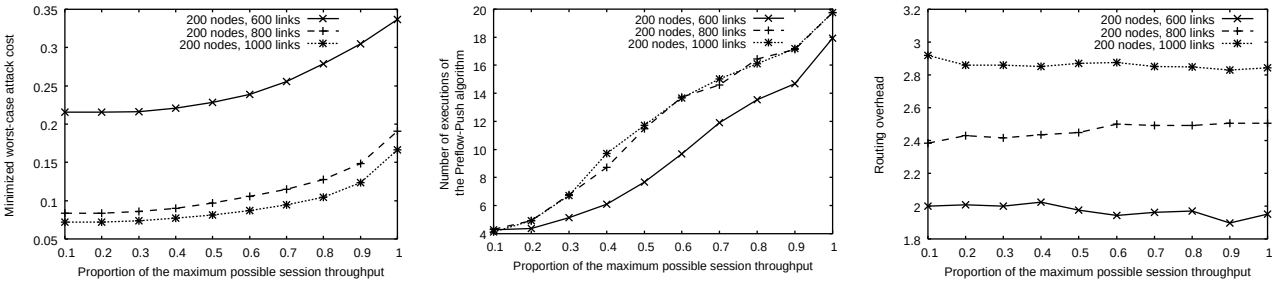
B. Discussion of the Lex-Control Algorithm

The complexity of the Lex-Control algorithm is dominated by the executions of the Bound-Control algorithm. Since each lexicographic iteration discovers at least one critical link, the Lex-Control algorithm has a complexity that is $O(|\mathcal{L}|T')$, where T' is complexity of the Bound-Control algorithm.

Instead of locating all critical links, we can simply perform a pre-determined number, say k, of lexicographic iterations to identify a subset of critical links in order to gain performance benefits in actual implementation. Since the later lexicographic iterations attempt to identify the critical links with modest attack costs, the most substantial security improvements occur during earlier lexicographic iterations. With this modification, the complexity of the Lex-Control algorithm is reduced to $O(kT')$.

V. Experiments

In this section, we perform an extensive experimental study on the proposed algorithms via simulation. We consider three network settings, each of which contains 200 nodes, connected by 600, 800, and 1000 links, respectively. We use BRITE [28], a network topology generator, to construct 50 experimental topologies for each network setting. All nodes within a topology are randomly connected and placed in a rectangular two-dimensional plane. We dedicate the nodes closest to and farthest from the origin (i.e., the bottom left-hand corner of the plane) to be source s and sink t, respectively. To construct a directed acyclic topology, for each link between any two



(a) Minimized worst-case attack cost (b) Number of executions of the Prefbw-Push algorithm (c) Routing overhead

Fig. 4. Experiment 1: Analysis of the Bound-Control algorithm at different session throughputs.

nodes u and v, we direct it from node u to node v if node u's Euclidean distance to the origin is less than that of node v. Moreover, each link l is uniformly assigned a security constant c_l between 0 and 1 and a bandwidth B_l between 1 and 5. We then analyze the average performance of the algorithms over the 50 topologies.

Our experiments focus on three metrics, namely: (1) *attack cost* (defined in Section II), which measures the resilience of the proposed algorithms toward various types of link attacks, (2) *number of executions of the Preflow-Push algorithm*, which measures the message complexity and the convergence time of the proposed algorithms. (3) *routing overhead*, which is defined as the ratio of the average hop count from source s to sink t in our multipath approach to the minimum hop count in single-path routing. We can compute the routing overhead as follows. Let $r(u)$ be the hop count from node u to sink t and $l_{uv} \in \mathcal{L}$ be the link directed from node u to node v. Recall that x_l denotes the proportion of the session data carried by link l. The average hop count of the multipath routing is thus given by the recursive equation $r(s) = \sum_{u:l_{su} \in \mathcal{L}} \frac{x_{l_{su}}}{\sum_{u:l_{su} \in \mathcal{L}} x_{l_{su}}}[1 + r(u)]$, where $r(t)$ is initialized to be zero. We then divide $r(s)$ by the minimum hop-count in single-path routing to obtain the routing overhead.

Experiment 1 (Analysis of the Bound-Control algorithm at different session throughputs): This experiment studies how the Bound-Control algorithm protects against the worst-case single-link attack at various session throughputs. For each topology, we use the Preflow-Push algorithm to determine the maximum possible session throughput subject to the link-bandwidth constraints. We then calculate the throughput rates that are given by different proportions of the determined maximum session throughput to address both fixed-rate and maximal-rate session models (see Section I). Finally, we assign the appropriate fraction bounds to all links (see Section III-B). Here, we evaluate the degree of resilience based on the *minimized worst-case attack cost*.

Figure 4 depicts the performance metrics at different session throughputs, and Table II shows the worst-case attack cost when a single path with the minimum hop-count is used. Figure 4(a) shows that the Bound-Control algorithm substantially reduces the worst-case attack cost when compared to the single-path approach (e.g., from 0.78 to 0.17, or by 78%, for the 1000-link network that uses the maximal-rate model for the maximum session throughput). Specifically, we observe two kinds of trade-offs. First, as the session throughput increases, links experience tighter fraction bounds in general. This leads to more Preflow-Push executions and higher worst-case attack cost. Second, while a network with more links attains a smaller worst-case attack cost, it also incurs more messages in running the Bound-Control algorithm as well as higher routing overhead.

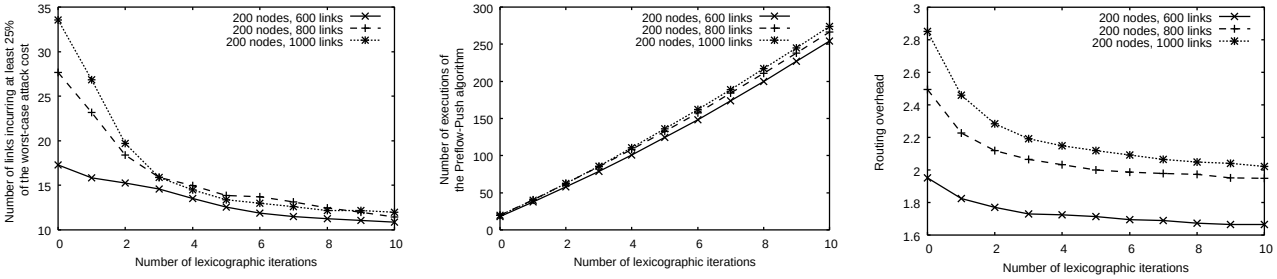
TABLE II

EXPERIMENT 1: WORST-CASE ATTACK COST WHEN A SINGLE PATH WITH THE MINIMUM HOP-COUNT IS USED.

Network setting	Attack cost
200 nodes, 600 links	0.73
200 nodes, 800 links	0.72
200 nodes, 1000 links	0.78

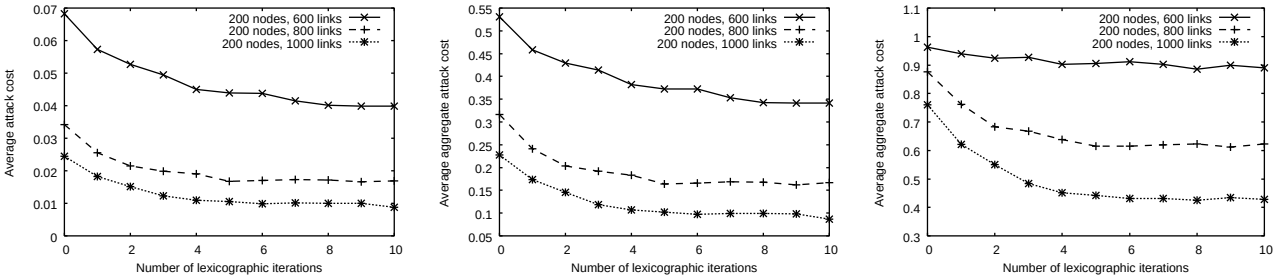
Experiment 2 (Analysis of the Lex-Control algorithm at different numbers of lexicographic iterations): This experiment considers how the Lex-Control algorithm prevents the severe single-link attacks when it executes different numbers of lexicographic iterations. We regard a single-link attack as "severe" if its resulting attack cost is at least 25% of the worst-case one. Here, for each topology, we evaluate the algorithm using the maximal-rate session model (see Section I) in which the maximum session throughput is determined as in Experiment 1. Also, we use the *number of links that incur severe attack costs* as the resilience measure.

Figure 5 plots the resulting metrics. It shows that the Lex-Control algorithm can reduce the number of links where the single-link attacks are severe. The reduction is more salient in the 1000-link network (e.g., by more than 50% in three or more lexicographic iterations). The trade-off is that the required number of executions of the Preflow-Push algorithm increases linearly with the number of lexicographic iterations. One interesting side benefit of the Lex-Control algorithm is that it alleviates the routing overhead as well. A possible explanation is that shorter paths incur smaller attack costs in general, so as the Lex-Control algorithm proceeds, it identifies these more secure shorter paths and hence reduces the routing overhead. From Figure 5, the benefits of the Lex-Control algorithm are more prominent in the first three lexicographic iterations. Thus, in practice, it is reasonable to run a small number of lexicographic iterations. This allows system designers to select the trade-off of diminishing returns.



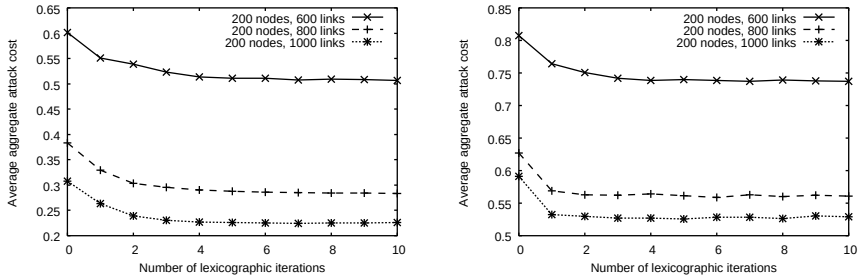
(a) Number of links incurring severe attack costs (b) Number of executions of the Prefbw-Push algorithm (c) Routing overhead

Fig. 5. Experiment 2: Analysis of the Lex-Control algorithm at different numbers of lexicographic iterations.



(a) Average attack cost under the uniform single-link attacks (b) Average aggregate attack cost under the uniform 10-link attacks (c) Average aggregate attack cost under the uniform 50-link attacks

Fig. 6. Experiment 3: Analysis of the Lex-Control algorithm subject to different scales of uniform link attacks.



(a) Average aggregate attack cost under the proportional 5-link attacks (b) Average aggregate attack cost under the worst-case 5-link attacks

Fig. 7. Experiment 4: Analysis of the Lex-Control algorithm subject to the proportional and worst-case multi-link attacks

Experiment 3 (Analysis of the Lex-Control algorithm subject to different scales of uniform link attacks): Although our analysis concentrates on the worst-case single-link attack, since the Lex-Control algorithm seeks the most balanced distribution of attack costs of all links, we envision that it also minimizes the average attack cost under *uniform link attacks*, i.e., an intruder uniformly attacks a single or multiple links that carry session data. In this experiment, we investigate this potential benefit by considering different scales of uniform link attacks.

In the experiment setup, we let the security constant c_l be the proportion of loss of data traversing link l that is being attacked (see Section II), so the attack cost of link l (i.e., $a_l = c_l x_l$) represents the actual proportion of data loss for the data session. For the single-link attack, we compute the *average attack cost* by dividing the total attack cost of all links by the number of links that carry data. For multi-link attacks, we look at the amount of remaining data that actually reach the sink in order to compute the aggregate attack cost. Then we simulate 50 multi-link attacks for each topology to obtain the *average aggregate attack cost*. Here, we focus on the maximal-rate session model as in Experiment 2.

Figure 6 illustrates the attack costs incurred by the uniform attacks on one, 10, and 50 links. It shows that the Lex-Control algorithm can mitigate the threats of uniform link attacks. For instance, given that 50 out of 1000 links are attacked, the average aggregate attack cost is reduced by 40% (or from 0.75 to 0.45) with four or more lexicographic iterations. Therefore, apart from the worst-case single-link attack, the Lex-Control algorithm also enhances the robustness of the network subject to various scales of uniform link attacks.

Experiment 4 (Analysis of the Lex-Control algorithm subject to the proportional and worst-case multi-link attacks): The final experiment assesses the Lex-Control algorithm under

the *proportional* and *worst-case multi-link* attacks. In the proportional multi-link attack, an intruder attacks a number of links such that the probability that each link is attacked is directly proportional to its attack cost. In the worst-case multi-link attack, however, the intruder deterministically attacks the links with the highest attack costs. We use the same setting as in Experiment 3 to evaluate the Lex-Control algorithm based on the maximal-rate session model.

Figure 7 illustrates the average aggregate attack costs when five links are attacked. It shows that in general, the Lex-Control algorithm can reduce the average aggregate attack costs in both proportional and worst-case attacks. For instance, in a 1000-link network, the attack cost is decreased from 0.3 to 0.23, or by 23%, in the proportional 5-link attack, and from 0.59 to 0.52, or by 12%, in the worst-case 5-link attack. Also, around four lexicographic iterations are sufficient to achieve such reduction.

Summary: The experiments show that the Bound-Control algorithm significantly protects against the worst-case single-link attack, and that the Lex-Control algorithm provides additional protection by reducing the number of links with severe attack costs. Moreover, the Lex-Control algorithm effectively defends against the uniform, proportional, and worst-case multi-link attacks, with the majority of benefits occurring within the first few lexicographic iterations.

VI. Application in Attack-Resistant Networks

To further examine the applicability of both Bound-Control and Lex-Control algorithms, we consider their use in an *attack-resistant network* [7], a specialized network that protects end hosts by surrounding them with a defensive architecture. One example is Secure Overlay Services (SOS) [20], which constructs the defensive architecture with a set of application-level overlay nodes layered atop the underlying network architecture. According to [7], an attack-resistant network should satisfy two crucial but contradicting criteria termed (1) *resiliency*: the network should offer alternate paths in the face of node failures, and (2) *security*: the network should confine the damage caused by compromised nodes. In this section, we propose how to balance these two criteria using our multipath solution and evaluate their trade-off via simulation.

A. Overview

Following [7], [20], we first overview the architecture of an attack-resistant network as illustrated in Figure 8. To communicate with a protected sink t, a source s first connects to an *access point* (AP), which authenticates incoming packets. Authenticated packets are then routed from the AP through an interconnection network to a *target* (a.k.a. *secret servlets* [20]), which relays packets to sink t. Intuitively, the AP and the target can be viewed as an entry point to the attack-resistant network and to sink t, respectively, such that no packet can reach sink t without properly going through an AP followed by a target. In a general attack-resistant network, source s can reach sink t using multiple paths through different APs and targets. Also, the APs function independently from one another [7]. Thus, we can deploy our distributed multipath solution in this type of attack-resistant network to further protect the underlying secure communication.

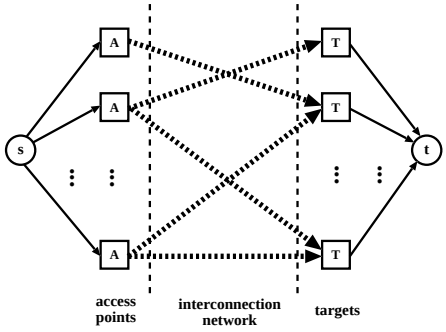


Fig. 8. Example of an attack-resistant network.

In order to secure the targets and hence the end hosts, the identities of the targets are hidden from the public and known only to a small set of nodes (a.k.a. *beacons* [20]) which are in turn known only to the associated APs. However, if an AP is compromised, then an intruder can identify and hence attack the associated targets through the compromised AP. This poses a trade-off issue between resiliency and security: for resiliency, each AP should be associated with sufficient targets so as to select an alternate target if one target becomes unavailable, while for security, each AP should not be assigned too many targets so as to suppress the number of targets being attacked when an AP is compromised.

To address the trade-off between resiliency and security, [7] formulates an assignment problem, namely, given a fixed data allocation on each AP, the objective is to find an optimal path assignment between the APs and targets that minimizes the *blocking probability*, defined as the probability that a request cannot reach a protected end host due to the attacks on the APs and targets. However, the existence of a polynomial-time optimal algorithm to this problem remains an open issue. In addition, the approximation algorithms in [7] do not take into account load balancing on the APs and targets. Therefore, we address the problem from another perspective in which our goal is to assign transmission rates at the APs instead of finding an optimal path assignment. We formalize the problem in the following discussion.

B. Model

We first formulate the problem based on [7]. We consider a network graph $\mathcal{G}_a = (\mathcal{A}, \mathcal{T}, \mathcal{P})$, where $\mathcal{A}$ is the set of APs, $\mathcal{T}$ is the set of targets, and $\mathcal{P}$ is the set of directed paths from the APs to the targets. We let $\mathcal{A}(j)$ and $\mathcal{T}(i)$ be the sets of APs and targets, respectively, such that if there exists a path in $\mathcal{P}$ from access point $i \in \mathcal{A}$ to target $j \in \mathcal{T}$, then $i \in \mathcal{A}(j)$ and $j \in \mathcal{T}(i)$.

We focus on two types of attacks: (1) *compromise attack*, in which an intruder obtains unauthorized access to a node, and (2) *denial-of-service (DoS) attack*, in which an intruder prevents a node from providing legitimate service, for example, by flooding the victim node with huge traffic. Let C_u and D_u be the respective events that node $u \in \mathcal{A} \cup \mathcal{T}$ is compromised and is vulnerable to DoS attacks, and $\overline{C_u}$ and $\overline{D_u}$

denote their complements. Let also $P(E)$ be the probability that event E occurs. In practice, the attack probabilities $P(C_u)$ and $P(D_u)$ can be estimated via the approaches described in Section II. Since the identities of the targets are hidden from the public, we assume that an intruder can only mount the attacks through the APs. It follows that $P(C_j) = 0$ and $P(D_j|\overline{C_{i_1}}, \cdots, \overline{C_{i_k}}) = 0$ for all $j \in \mathcal{T}$ and $i_1, \cdots, i_k \in \mathcal{A}(j)$. We assume that if AP i is compromised, then an intruder will identify all targets $j \in \mathcal{T}(i)$ and launch DoS attacks on them from the compromised AP i. This implies $P(D_j|C_i) = 1$ for all $j \in \mathcal{T}(i)$. Finally, we assume that $C_i's$ and $D_i's$ are mutually independent for all $i \in \mathcal{A}$.

An AP $i \in \mathcal{A}$ is said to be *blocked* if it is compromised, it is the victim of a DoS attack, or its associated targets are all victims of a DoS attack. Thus, the blocking probability p_i at AP i is given by

$$p_i \quad = \quad 1 - P(\overline{D_i})P(\bigcup_{j\in\mathcal{T}(i)} \overline{D_j}). \tag{7}$$

Note that the event $\bigcup_{j\in\mathcal{T}(i)} \overline{D_j}$ implies that AP i is not compromised. To compute $P(\bigcup_{j\in\mathcal{T}(i)} \overline{D_j})$, we note that as a result of the independence assumption, the probability that all k targets $j_1, \cdots, j_k \in \mathcal{T}$ are not vulnerable to DoS attacks is given by

$$P(\overline{D_{j_1}} \cap \cdots \cap \overline{D_{j_k}}) = \prod_{i\in\mathcal{T}(j_1)\cup\cdots\cup\mathcal{T}(j_k)} P(\overline{C_i}). \tag{8}$$

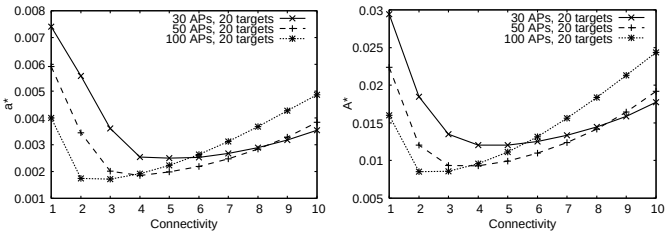
We then evaluate $P(\bigcup_{j\in\mathcal{T}(i)} \overline{D_j})$ via the *inclusion-exclusion principle* [9].

To apply our algorithms, we extend the single-link attack model in Section II to a *single-node attack* model (e.g., via *node splitting* [2]). We let x_u, where $0 \le x_u \le 1$, be the proportion of data carried by node u, where $u \in \mathcal{A} \cup \mathcal{T}$ can be an AP or a target, and $\mathbf{x} = (x_u, u \in \mathcal{A} \cup \mathcal{T})$ be the proportion vector. The attack cost of node u is thus defined as $a_u = c_u x_u$, where c_u is the security constant of node u. Using the blocking probability as our measure, for every AP $i \in \mathcal{A}$, we set $c_i = p_i$, indicating that the attack cost of AP i is quantified as the expected proportion of data loss when AP i is blocked. In contrast, since the blocking probability is calibrated only at the layer of APs, for every target $j \in \mathcal{T}$, we set $c_j = 0$. To determine the security constant, each AP can independently consult its attached targets for the compromise probabilities of their associated APs. It then computes the blocking probability based on Equations 7 and 8. Furthermore, for load balancing, each of the APs and targets can assign itself a bandwidth constraint and determine its fraction bound b_u, where $0 \le b_u \le 1$ for $u \in \mathcal{A} \cup \mathcal{T}$, using either the fixed-rate model or the maximal-rate model.

Given the above formulation, our primary objective is to decide the data allocation $\mathbf{x} = (x_u, u \in \mathcal{A} \cup \mathcal{T})$ that minimizes the worst-case attack cost of an AP subject to bandwidth constraints. Thus, we have the following optimization problem:

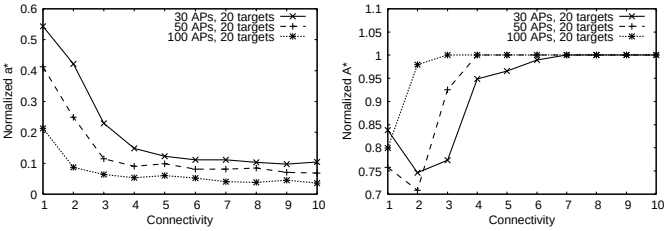
$$\begin{aligned} a^* \quad &= \quad \min_{\mathbf{x}} \max_{i\in\mathcal{A}} a_i = \min_{\mathbf{x}} \max_{i\in\mathcal{A}} c_i x_i \\ \text{subject to} \quad & \quad 0 \le x_u \le b_u, \ \forall u \in \mathcal{A} \cup \mathcal{T}. \end{aligned} \tag{9}$$

We can further extend Problem 9 to a lexicographic optimization problem. We can then readily obtain the optimal data allocation using the Bound-Control or Lex-Control algorithms.



(a) Minimized worst-case attack cost (b) Aggregate attack cost

Fig. 9. Evaluation for attack-resistant networks: attack cost vs. connectivity.



(a) Normalized minimized worst-case attack cost (b) Normalized aggregate attack cost

Fig. 10. Evaluation for attack-resistant networks: normalized attack cost vs. connectivity.

C. Evaluation

We conduct a simulation study on three attack-resistant network settings, each of which has 30, 50, and 100 APs, respectively, together with 20 targets. We define *connectivity* as the number of targets to which each AP is connected. For each network setting and connectivity, we consider the average results over 50 topologies. Within a topology, each AP is connected randomly to different targets according to the connectivity and is assigned the compromise and DoS probabilities uniformly at random between 0 and 0.08. Moreover, we assume that each AP and its paths to the targets have infinite bandwidth and that each target has the same bandwidth. We then determine the fraction bounds of the APs and targets using the maximal-rate model.

We first analyze the minimized worst-case attack cost a^* as well as the aggregate attack cost A^* of the five APs that have the highest attack costs. We compute a^* and A^* respectively via the Bound-Control algorithm and the Lex-Control algorithm with three lexicographic iterations. Figures 9(a) and 9(b) plot a^* and A^* versus the connectivity, respectively. Initially, the resilience of an attack-resistant network increases with the connectivity and both attack costs decrease. As the connectivity further increases, the targets are attached to more APs that can be compromised. Thus, they become more vulnerable to DoS attacks, and the attack costs increase. Such increase is more severe when a network has more APs (e.g., 100 APs). This shows that a network with more APs and higher connectivity does not necessarily offer better protection.

To evaluate the effectiveness of the multipath algorithms, we normalize a^* to the respective cost when no Bound-Control algorithm is used (i.e., only the maximal-rate model is satisfied), and we also normalize A^* to the respective cost when only the Bound-Control algorithm is used. Figure 10(a) plots the normalized a^* and shows that the Bound-Control

algorithm can effectively reduce the worst-case attack cost of an AP (e.g., by at least 80% for 100 APs and 20 targets). In Figure 10(b), we plot the normalized A^* and observe that the Lex-Control algorithm reduces the aggregate attack cost at low connectivities. As the connectivity increases, each AP is assigned more targets and is less constrained by the bandwidth requirement. Thus, the Bound-Control algorithm immediately minimizes the worst-case attack cost at all APs (i.e., where the minimum cut lies) and has the same result as does the Lex-Control algorithm.

VII. Related Work

Multipath routing was first studied in [27], in which data is transmitted over multiple disjoint paths as a means to provide load balancing and routing resilience. Redundancy can be added to the transmitted data so that the receiver side can fully reconstruct the data in the presence of moderate data loss. Based on this intuition, we apply multipath routing to address the presence of link attacks using optimization models.

One possible optimization model for multipath routing is based on minimax optimization. Previous studies consider the load-balancing problem (e.g., in [1], [17]), multipath solutions to combat link attacks (e.g., in [5], [6], [18]), and the network-intrusion problem (e.g., in [21]). Note that the above studies focus on centralized algorithms that assume the knowledge of the network topology. We extend beyond the previous studies by devising a distributed solution that can handle link-bandwidth constraints.

Another possible optimization model is based on lexicographic optimization, which has been studied in [11], [13] for a network setting. While [11] considers only the lexicographic optimization of the flows of the links attached to the source node, [13] extends [1] to lexicographically optimize the flows of all the network links in a centralized manner. Specifically, the idea of [13] is to solve the minimax problem via the maximum-flow problem for a given network, identify the minimum-cut links, and recursively solve the minimax problems for the subnetworks separated by those links. Our Lex-Control algorithm supports the distributed implementation in the presence of link-bandwidth constraints.

Analytical studies regarding secure multipath routing can be found in [4], [5], [6], [18], [25], [32], in which the vulnerability of each node is characterized by an attack (or failure) probability. In particular, [25], [32] consider disjoint paths among network nodes, while [4] relaxes this disjointness requirement and proposes a resilient routing scheme using two non-disjoint paths. Our algorithms, as in [5], [6], [18], explore a higher degree of network diversity by using all the paths (either disjoint or not) that are available.

In terms of the applicability of secure multipath routing, [30] studies the implementation issues of protecting an attack-resistant network (e.g., SOS [20]) against the intruders that seek to compromise a small portion of overlay nodes at random. Our work, on the other hand, provides an analytical study for protecting an attack-resistant network using a worst-case attack model. Besides attack-resistant networks, multipath routing has also been applied to improve the resilience of other architectures, such as sensor networks [12].

VIII. Practical Issues

In this section, we address several practical issues of our current work and suggest directions for future research.

Redundant routing: As mentioned in Section I, we assume that some error correction mechanism is used to reliably deliver data. Intuitively, redundant messages must be added to transmitted data so that a receiver can recover all data as long as data loss due to failed paths is modest [25], [27]. Although redundant routing provides data reliability, part of the raw network bandwidth is used to transmit redundant data, and this decreases the effective network bandwidth for carrying actual data. Determining the suitable level of redundancy that best balances the trade-off between data reliability and effective network bandwidth is challenging and hence requires further investigation.

Fault-tolerance: We currently assume that the nodes remain stable throughout the execution of the algorithms, yet in practice, nodes can experience attacks or transient failures. To offer fault-tolerance, we can either restart the algorithms, or adopt the self-stabilizing solutions in [14], [19]. In particular, [19] enhances the original Preflow-Push algorithm to adjust to the changes of link states. However, the worst-case complexity of this solution is proportional to the number of adjustments multiplied by the complexity of the original Preflow-Push algorithm, leading to severe performance degradation if the adjustments occur frequently. Hence, we need to consider the trade-offs between restarting the algorithms and invoking the self-stabilizing procedures.

Multiple sessions: Our algorithms are on a per-session basis. To support multiple sessions, one simple approach is to require each link to allocate different bandwidths (or fraction bounds) for the multiple sessions based on the application requirement. However, such an approach may not fully utilize the link bandwidth. For example, we may allocate the unused link bandwidth of one session to other sessions. If a session is given more bandwidth, it is subject to weaker fraction bounds. In Section V, Experiment 1 shows that weaker fraction bounds can achieve smaller worst-case attack cost. Thus, we have to examine how to allocate fraction bounds for multiple sessions effectively to achieve the optimal solution.

Quantifying vulnerability: We currently assume that we can characterize the damage of an attack via security constants (see Section II) or attack probabilities (see Section VI). We emphasize that our algorithms can still be applied regardless of the actual values of these parameters, although a sound attack-modeling mechanism can provide better data allocation over multiple paths. We pose this problem as future work.

IX. Conclusions

We presented a distributed secure multipath solution that comprises the Bound-Control and Lex-Control algorithms, both of which proactively combat link attacks in a distributed fashion and provably converge to the desired optimal solutions. We used simulation to demonstrate the resilience of both algorithms toward different types of single-link and multi-link attacks. Specifically, simulation results demonstrate that the Lex-Control algorithm counters severe link attacks efficiently

within the first few lexicographic iterations, and hence both routing security and algorithm performance can be effectively achieved during actual implementation. Finally, we studied the applicability of both algorithms using an attack-resistant network as an example. By simulation, we evaluated their performance and analyzed how they react to resiliency and security under different attack-resistant network settings.

REFERENCES

[1] R. K. Ahuja. Algorithms for the Minimax Transportation Problem. *Naval Research Logistics Quarterly*, 33:725–740, 1986.

[2] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin. *Network Flows: Theory, Algorithm, and Applications*. Prentice Hall, 1993.

[3] D. Andersen, H. Balakrishnan, F. Kaashoek, and R. Morris. Resilient overlay networks. In *Proc. of the 18th ACM Symposium on Operating Systems Principles (SOSP)*, October 2001.

[4] R. Banner and A. Orda. The Power of Tuning: a Novel Approach for the Effi cient Design of Survivable Networks. In *Proc. of IEEE International Conference on Network Protocols (ICNP)*, October 2004.

[5] S. Bohacek, J. Hespanha, J. Lee, C. Lim, and K. Obraczka. Enhancing Security via Stochastic Routing. In *Proc. of ICCCN*, May 2002.

[6] J. P. Brumbaugh-Smith and D. R. Shier. Minimax Models for Diverse Routing. *INFORMS Journal on Computing*, 14(1):81–95, Winter 2002.

[7] T. Bu, S. Norden, and T. Woo. Trading Resiliency for Security: Model and Algorithms. In *IEEE International Conference on Network Protocols (ICNP)*, October 2004.

[8] J. Byers, M. Luby, and M. Mitzenmacher. Accessing Multiple Mirror Sites in Parallel: Using Tornado Codes to Speed Up Downloads. In *Proc. of IEEE INFOCOM*, March 1999.

[9] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. MIT Press, 2nd edition, 2001.

[10] W. Du and A. Mathur. Testing for Software Vulnerability Using Environment Perturbation. In *Proc. of the International Conference on Dependable Systems and Networks*, 2000.

[11] G. Gallo, M. D. Grigoriadis, and R. E. Tarjan. A Fast Parametric Maximum Flow Algorithm and Applications. *SIAM Journal on Computing*, 18(1):30–55, February 1989.

[12] D. Ganesan, R. Govindan, S. Shenker, and D. Estrin. Highly-resilient, Energy-effi cient Multipath Routing in Wireless Sensor Networks. *ACM SIGMOBILE Mobile Computing and Communications Review*, 5(4):11–25, October 2001.

[13] L. Georgiadis, P. Georgatsos, K. Floros, and S. Sartzetakis. Lexicographically Optimal Balanced Networks. *IEEE/ACM Transactions on Networking*, 10(6):818–829, December 2002.

[14] S. Ghosh, A. Gupta, and S. V. Pemmaraju. A Self-stabilizing Algorithm for the Maximum Flow Problem. *Distributed Computing*, 10(3):167–180, 1997.

[15] B. Gnedenko and I. A. Ushakov. *Probabilistic Reliability Engineering*. John Wiley & Sons, Inc., 1995.

[16] A. V. Goldberg and R. E. Tarjan. A New Approach to the Maximum-Flow Problem. *Journal of the Association for Computing Machinery*, 35(4):921–940, October 1988.

[17] C.-C. Han, K. G. Shin, and S. K. Yun. On Load Balancing in Multicomputer/Distributed Systems Equipped with Circuit or Cut-Through Switching Capability. *IEEE Transactions on Computers*, 49(9):947–957, September 2000.

[18] J. Hespanha and S. Bohacek. Preliminary Results in Routing Games. In *Proc. of the 2001 American Control Conference*, volume 3, pages 1904–1909, June 2001.

[19] B. Hong and V. K. Prasanna. Distributed Adaptive Task Allocation in Heterogeneous Computing Environments to Maximize Throughput. In *Proc. of IPDPS*, April 2004.

[20] A. Keromytis, V. Misra, and D. Rubenstein. SOS: An Architecture for Mitigating DDoS Attacks. *IEEE JSAC, Special Issue on Service Overlay Networks*, 22(1), January 2004.

[21] M. Kodialam and T. V. Lakshman. Detecting Network Intrusions via Sampling: A Game Theoretic Approach. In *Proc. of IEEE INFOCOM*, April 2003.

[22] P. Lee, V. Misra, and D. Rubenstein. Distributed Algorithms for Secure Multipath Routing. In *Proc. of IEEE INFOCOM*, March 2005.

[23] S. Lee, Y. Yu, S. Nelakuditi, Z.-L. Zhang, and C.-N. Chuah. Proactive vs. Reactive Approaches to Failure Resilient Routing. In *Proc. of IEEE INFOCOM*, March 2004.

[24] D. Loguinov and H. Radha. End-to-End Internet Video Traffi c Dynamics: Statistical Study and Analysis. In *Proc. of IEEE INFOCOM*, June 2002.

[25] W. Lou, W. Liu, and Y. Fang. SPREAD: Enhancing Data Confi dentiality in Mobile Ad Hoc Networks. In *Proc. of IEEE INFOCOM*, March 2004.

[26] G. Malkin. RIP Version 2, November 1998. RFC 2453.

[27] N. Maxemchuk. Dispersity Routing. *Proc. of ICC*, 1975.

[28] A. Medina, A. Lakhina, I. Matta, and J. Byers. BRITE: An Approach to Universal Topology Generation. In *Proc. of MASCOTS*, August 2001.

[29] J. Moy. OSPF Version 2, April 1998. RFC 2328.

[30] A. Stavrou and A. D. Keromytis. Countering DoS Attacks with Stateless Multipath Overlays. In *Proc. of ACM Conference on Computer and Communications Security (CCS)*, November 2005.

[31] D. Thaler and C. Hopps. Multipath Issues in Unicast and Multicast Next-Hop Selection, November 2000. RFC 2991.

[32] J. Yang and S. Papavassiliou. Improving network security by multipath traffi c dispersion. In *IEEE Military Communications Conference (MILCOM)*, October 2001.

APPENDIX I
PREFLOW-PUSH ALGORITHM

We outline the Preflow-Push algorithm in Algorithm 3, while the detailed discussion of the algorithm can be found in [2], [16].

Algorithm 3 Preflow-Push

```
source s pushes as much flow as possible to its neighbors
while ∃ node u∈N−{s,t} having flow excess do
    if node u has an admissible neighbor node v then
        node u pushes excess flow to node v
    else
        node u relabels its estimated shortest distance to sink t
source s determines the maximum flow f*
```

Consider a network $\mathcal{G} = (\mathcal{N}, \mathcal{L})$ with source $s \in \mathcal{N}$ and sink $t \in \mathcal{N}$. Initially, source s first pushes the maximum possible flow toward its neighbor nodes in $\mathcal{G}$. For every node $u \in \mathcal{N} - \{s, t\}$ that has flow excess, it seeks to push its flow excess to an *admissible* neighbor node v (a.k.a. *push* operation). By admissible, we mean node v is a neighbor node of u in the residual network (defined in Section IV) with respect to the current flow f and node v lies on the estimated shortest path from node u to sink t. If no admissible neighbor is found, then node u updates its estimated shortest distance to sink t (a.k.a. *relabel* operation). The algorithm terminates when all nodes besides source s and sink t have no excess flow. Then source s can determine the maximum flow by checking how much flow has been actually sent to sink t.

APPENDIX II
PROOFS

A. Correctness of the Bound-Control Algorithm

To prove the correctness of the Bound-Control algorithm, we first show the existence of an optimal maximum flow f^* for Problem 4 (defined in Section II) under a necessary and sufficient condition for the values of b_l. Then we prove that the flow value f_s broadcast by source s is strictly decreasing and bounded from below by f^*. This implies the Bound-Control algorithm converges to the optimal value f^*.

Lemma 1: (Existence) There always exists a maximum flow $f^* > 0$ for Problem 4 if and only if $\sum_{l \in \mathcal{C}} b_l \geq 1$ for any cut $\mathcal{C}$ in the network $\mathcal{G}$.

Proof: Necessity ($\Rightarrow$): Given $f^* > 0$, suppose that there exists a cut $\mathcal{C}$ such that $\sum_{l\in\mathcal{C}} b_l < 1$. Hence, the capacity of the cut $\mathcal{C}$ is given by $\sum_{l\in\mathcal{C}} \min(1/c_l, b_l f^*) \leq \sum_{l\in\mathcal{C}} b_l f^* = f^* \sum_{l\in\mathcal{C}} b_l < f^*$. This contradicts the max-flow min-cut theorem, which suggests that the capacity of any cut is at least the value of the maximum flow.

Sufficiency ($\Leftarrow$): We want to show that $f = 1$ is a feasible flow for Problem 4. From Problem 4, if $f = 1$, the capacity of any cut $\mathcal{C}$ is given by $\sum_{l\in\mathcal{C}} \min(1/c_l, b_l) \geq \sum_{l\in\mathcal{C}} b_l \geq 1$ (recall that c_l is normalized and so $1/c_l \geq 1$). Hence, the flow $f = 1$ is bounded from above by the capacity of any cut and is regarded as feasible. This implies the optimal maximum flow $f^* > 0$ exists. ∎

Before proceeding to the next proof, we define additional notation. Based on Algorithm 1, we first let $f_s^{(0)}$ be the flow value f_s initially broadcast (line 1). For $n \geq 1$, we let $f_s^{(n)}$ be the flow value f_s broadcast in the nth iteration of the repeat-until loop (line 8). Note that $f_s^{(n)}$ represents the maximum flow computed from the previous execution of the Preflow-Push algorithm. Moreover, we let $\mathcal{C}^{(n)}$ and $\mathcal{C}^*$ be one of the minimum cuts associated with the maximum flows $f_s^{(n)}$ and f^*, respectively.

Lemma 2: (Monotonicity and boundedness) For any positive integer n, we have $f_s^{(n-1)} \geq f_s^{(n)} \geq f^*$. In particular, if $f_s^{(n-1)} = f_s^{(n)}$ for some n, we have $f_s^{(n-1)} = f_s^{(n)} = f^*$.

This lemma implies that prior to the termination of the Bound-Control algorithm, the flow value f_s is strictly decreasing. Furthermore, if the value f_s that has just been broadcast equals the computed maximum-flow result, the algorithm terminates with the optimal value $f^* = f_s$.

Proof: We first prove by induction on n that $f_s^{(n-1)} \geq f_s^{(n)} \geq f^*$ for any positive integer n.

- *Base case*: For $n = 1$, $f_s^{(0)}$ equals the sufficiently large value U, while $f_s^{(1)}$ is the maximum flow given by the first run of the Preflow-Push algorithm. This implies that $f_s^{(0)} \geq f_s^{(1)}$. Also, $f_s^{(1)}$ and f^* are the maximum flows subject to the capacity constraints $cap(l) = 1/c_l$ and $cap(l) = \min(1/c_l, b_l f_s)$ for every link $l \in \mathcal{L}$, respectively. Since the latter constraint is tighter, f^* is no greater than $f_s^{(1)}$.
- *Induction hypothesis*: Let $f_s^{(k-1)} \geq f_s^{(k)} \geq f^*$ for some positive integer k.
- *Induction step*: We note that $f_s^{(k)}$, $f_s^{(k+1)}$, and f^* are the maximum-flow results subject to the capacity constraints $cap(l) = \min(1/c_l, b_l f_s^{(k-1)})$, $cap(l) = \min(1/c_l, b_l f_s^{(k)})$, and $cap(l) = \min(1/c_l, b_l f^*)$ for every link $l \in \mathcal{L}$, respectively. By hypothesis, f^* is subject to the tightest capacity constraint, followed by $f_s^{(k+1)}$, and finally $f_s^{(k)}$. This implies that $f_s^{(k)} \geq f_s^{(k+1)} \geq f^*$.

By induction, we have $f_s^{(n-1)} \geq f_s^{(n)} \geq f^*$ for any positive integer n. Furthermore, if $f_s^{(n-1)} = f_s^{(n)}$, $f_s^{(n)}$ is the maximum flow satisfying the capacity constraint $cap(l) = \min(1/c_l, b_l f_s^{(n-1)}) = \min(1/c_l, b_l f_s^{(n)})$ for every link $l \in \mathcal{L}$. Thus, $f_s^{(n)}$ is a feasible flow for Problem 4. This implies $f_s^{(n)} \leq f^*$. However, we have proved that in every iteration, we have $f_s^{(n)} \geq f^*$. It follows that $f_s^{(n)} = f^*$. ∎

Theorem 1: (Convergence) The Bound-Control algorithm converges to the maximum-flow value $f^* > 0$ to Problem 4, provided that $\sum_{l\in\mathcal{C}} b_l \geq 1$ for any cut $\mathcal{C}$.

Proof: Immediate from Lemmas 1 and 2. ∎

B. Correctness of the Lex-Control Algorithm

We first present two properties that indicate how to pinpoint the critical links (see Section IV-A) in a distributed fashion.

Property 1: In a maximum-flow solution, if link $l \in \mathcal{L}$ lies on a minimum cut, then it is critical.

Proof: The attack cost of link $l \in \mathcal{L}$ is $a_l = c_l f_l / f^*$. Since c_l is fixed and f^* is the maximum flow, the attack cost a_l can only be decreased by reducing f_l. If link l lies on a minimum cut, it is saturated (i.e., flow of link l equals its link capacity). We can hence regard the reduction of f_l as the decrease in the capacity of link l. This results in the decrease in the capacity of the minimum cut and, by the max-flow-min-cut theorem, the decrease in the maximum flow f^*. Thus, the minimized worst-case attack cost $a^* = 1/f^*$ increases. By definition, link l is critical. ∎

Property 2: For every link $l \in \mathcal{L}$ directed from node u to node v, where $u, v \in \mathcal{N}$, if node v is not reachable from node u in the residual network $\mathcal{G}_{f^*}$ (see Section IV-A for its definition), then link l lies on a minimum cut.

Proof: Let $\mathcal{S}$ be the set of nodes reachable from node u in $\mathcal{G}_{f^*}$ and $\mathcal{T} = \mathcal{N} - S$. By assumption, we have $u \in \mathcal{S}$ and $v \in \mathcal{T}$. We note that link $l \in \mathcal{L}$ carries flow from u to v (otherwise, v is reachable from u in $\mathcal{G}_{f^*}$) and the flow originates from source s, so s is reachable from u in $\mathcal{G}_{f^*}$. It follows that $s \in \mathcal{S}$. Similarly, the flow arriving at v will eventually reach t, so v is reachable from t in $\mathcal{G}_{f^*}$. This implies that $t \in \mathcal{T}$ (if $t \in \mathcal{S}$ instead, v is reachable from u via t in $\mathcal{G}_{f^*}$). Moreover, since the nodes in $\mathcal{T}$ are not reachable from the nodes in $\mathcal{S}$, there are no links directed from $\mathcal{S}$ to $\mathcal{T}$ in $\mathcal{G}_{f^*}$, so the links from $\mathcal{S}$ to $\mathcal{T}$ in $\mathcal{G}$ are saturated and they must represent a minimum cut. Since l is one of the links directed from $\mathcal{S}$ to $\mathcal{T}$, l lies on the minimum cut. ∎

Based on Properties 1 and 2, each node $u \in \mathcal{N}$ can invoke any algorithm that can check the connectivity of a graph (e.g., the breadth-first search) on $\mathcal{G}_{f^*}$. This check is used to determine whether its neighbors in $\mathcal{G}$ are reachable in $\mathcal{G}_{f^*}$. If not, the corresponding links $l \in \mathcal{L}(u)$ between node u and its neighbors in $\mathcal{G}$ are lying on a minimum cut and hence are critical. This enables the identification of all critical links in a distributed fashion.

Now, we formally prove that the Lex-Control algorithm converges to the lexicographically optimal solution $\mathbf{a}^*$ (see Problems 5 and 6 in Section II).

Lemma 3: In the Lex-Control algorithm, if a link is determined to be critical in a lexicographic iteration, it remains critical in subsequent lexicographic iterations.

Proof: Consider the links that are found to be critical. By Property 1, they lie on some minimum cut. Let $\mathcal{C}$ be this minimum cut. From lines 9-10 of Algorithm 2, the capacity of the cut $\mathcal{C}$ is specified as $\sum_{l\in\mathcal{C}} b_l f$, where f is the flow value reaching the sink. By flow conservation, we have $\sum_{l\in\mathcal{C}} b_l = 1$, and thus the capacity of $\mathcal{C}$ is specified as f. In the next

lexicographic iteration, due to flow conservation, the flow across $\mathcal{C}$ is the newly computed maximum flow which also equals the specified capacity of $\mathcal{C}$. By the max-flow min-cut theorem, $\mathcal{C}$ is still a minimum cut and hence the underlying links remain critical. ■

Remark: Lemma 3 implies that the attack cost of every critical link remains unchanged in subsequent lexicographic iterations.

Lemma 4: Before the Lex-Control algorithm ends, every lexicographic iteration finds new critical links. Moreover, among the non-critical links that are identified to be critical, at least one of them has the attack cost $1/f^*$, where f^* is the maximum flow returned from the previous execution of the Bound-Control algorithm.

Proof: Suppose the algorithm proceeds to a new lexicographic iteration. This implies that f^* is less than the sufficiently large value U (due to line 3 of Algorithm 2), where f^* is the maximum flow computed from the previous execution of the Bound-Control algorithm. By the max-flow min-cut theorem, f^* equals the capacity of some minimum cut, say $\mathcal{C}$, and this capacity is equal to $\sum_{l\in\mathcal{C}} \min(1/c_l, b_l f^*)$. To achieve $f^* < U$, we must have a minimum cut $\mathcal{C}$ in which at least one link l has capacity equal to $1/c_l$ instead of $b_l f^*$ so that f^* is bounded away from U (otherwise, $f^* = U$ is the maximum flow and the algorithm terminates). This link l is previously non-critical (otherwise, its capacity is specified by $b_l f^*$ due to line 10 of Algorithm 2) and is now identified to be critical (since it lies on a minimum cut). Furthermore, its attack cost is given by $1/f^*$. ■

Remark: Lemma 4 implies that at least one newly identified critical link exhibits the minimized worst-case attack cost computed from the latest execution of the Bound-Control algorithm.

Lemma 5: Within the Lex-Control algorithm, the maximum flow computed in each execution of the Bound-Control algorithm is strictly increasing.

Proof: From Lemma 4, the maximum flow, say f^*, computed in an execution of the Bound-Control algorithm is given by $\sum_{l\in\mathcal{C}} \min(1/c_l, b_l f^*)$, where $\mathcal{C}$ denotes a minimum cut that includes a non-critical link l. Notice that $\mathcal{C}$ is not a minimum cut in the previous executions of the Bound-Control algorithm, or link l would have already been identified as critical. Thus, $\mathcal{C}$ has greater capacity. By the max-flow min-cut theorem, the computed maximum flow becomes greater, and is thus strictly increasing in each execution of the Bound-Control algorithm. ■

Theorem 2: The Lex-Control algorithm converges to the lexicographically optimal solution $\mathbf{a}^*$.

Proof: By Lemmas 3 and 4, each lexicographic iteration of the Lex-Control algorithm identifies two types of critical links: the already spotted ones (if any) and the newly spotted ones. By Lemma 3, the attack costs of the already identified critical links remain the same. Meanwhile, by the definition of a critical link and Lemma 4, the new critical links have their attack costs minimized subject to the computed minimized worst-case attack cost that is exhibited by at least one new critical link. Thus, the Lex-Control algorithm approaches the lexicographically optimal solution as more critical links are identified.

By Lemma 5, the maximum flow returned from the Bound-Control algorithm is strictly increasing, so it eventually reaches the very large value U. In this case, for any remaining non-critical link l, its attack cost is given by $a_l = c_l f_l / U$, which is negligibly small (or simply regarded as zero). Thus, the attack costs of any remaining links are at the optimized values (which are zeros). As the attack costs of the critical links are minimized (by the definition of a critical link), the Lex-Control algorithm terminates with the lexicographically optimal solution $\mathbf{a}^*$. ■