Distributed Algorithms for Secure Multipath Routing

Patrick P. C. Lee*, Vishal Misra*, and Dan Rubenstein†
*Department of Computer Science †Department of Electrical Engineering
Columbia University
New York, NY
{pclee,misra}@cs.columbia.edu, danr@ee.columbia.edu

***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 crucial properties that differentiate itself from previous approaches. They include (1) *distributed routing decisions*: routing decisions are made without the centralized information of the entire network topology, (2) *bandwidth-constraint adaptation*: the worst-case link attack is mitigated for any feasible session throughput subject to the link-bandwidth constraints, and (3) *lexicographic protection*: severe link attacks are suppressed based on lexicographic optimization. We devise two algorithms for the solution, termed the *Bound-Control algorithm* and the *Lex-Control algorithm*, and prove their convergence to the respective 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 models of multi-link attacks. Moreover, the Lex-Control algorithm offers prominent protection after only a few execution rounds. Thus, system designers can sacrifice minimal routing security for significantly improved algorithm performance when deploying the distributed secure multipath solution.**

***Index Terms*— security, multipath routing, minimax optimization, maximum-flow problems, graph theory**

I. Introduction

In conventional routing protocols such as OSPF [22] and RIP [20], a network selects the least-cost path for routing data from a sender to its targeted receiver. While this type of path selection addresses the performance issue regarding how data can be delivered 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 associated path. Such a denial-of-service (DoS) attack is feasible since only one single path is chosen, and this singularity enables intruders to readily devote their resources to attacking the only path.

Such networks can be secured with a *secure multipath approach* in which the sender achieves routing security by dispersing its data across multiple paths destined for the receiver. Each path conveys a portion of data from the sender, and the receiver assembles the data fragments arrived from various paths. To completely compromise the data session, intruders must subvert all the routing paths, and thus require more resources than those needed for attacking a single path. We point out that using multiple paths can complicate the packet-reordering problem [25]. However, it can be tackled via sophisticated coding solutions (e.g., [6]) for non-real-time data transfers or standard pre-buffering techniques (e.g., [18]) for real-time data transfers. Therefore, it is feasible to adopt the multipath scheme, and routing security is accomplished *proactively* by exploiting the network diversity.

While implementing secure multipath routing within conventional layer-3 architectures is a daunting task, more recent application-layer architectures such as overlay networks (e.g., RON [3] and SOS [16]) and programmable router paradigms (e.g., CROSS/Linux [12]) provide a promising platform for deploying this multipath service. For instance, SOS [16] is built upon an overlay network to proactively prevent DoS attacks. The overlay routing scheme proposed in SOS is Chord [24], which is a single-path approach. To offer multipath availability, we can instead use another overlay routing protocol CAN [23] with multiple coordinate spaces so that each overlay node is assigned multiple neighbors. With the secure multipath approach, the security strength of SOS is further elevated.

One major challenge is to design a *distributed* solution that implements the process of selecting the "best" data allocation across multiple paths on the above architectures. The distributed solution enhances the traditional centralized solutions for secure multipath routing such as [5], [14] in several aspects. First, it does not require any network node to have full knowledge of the entire network topology, and is therefore more easily applied in large networks. Also, it allows network nodes to locally decide the security costs, bandwidths, and choices of routes, and thus improves the flexibility over centralized coordination. Furthermore, it is adequate for decentralized peer systems, such as RON [3], whose nodes are located in different domains and are often administered independently.

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

- **Fixed-rate session**: A session wishes to send data from the source to the sink at a pre-determined rate.
- **Maximal-rate session**: A session wishes to send data from the source to the sink at the fastest rate allowed by the network using all available paths.

Our primary security objective is to minimize the maximum

This material was supported in part by the National Science Foundation under grant numbers CAREER ANI-560153 and NSF ANI-0238299, and by gifts from the Intel IT Research Council and CISCO, and IBM. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

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. 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 the 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 offer security guarantees using all available network resources. For example, in the maximal-rate session model where the link utilization is always unity, our solution minimizes the worst-case single-link attack while attaining the maximum possible throughput with the provisioned network bandwidth.

We first propose a distributed solution called the *Bound-Control algorithm* that 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 [11]. In particular, our Bound-Control algorithm adapts to the network where every link has a specified bandwidth that bounds the link throughput. Therefore, it supports both fixed-rate and maximal-rate session models subject to the link-bandwidth constraints.

We then extend the Bound-Control algorithm to another distributed solution called the *Lex-Control algorithm* that defends against not only the worst-case link attack, but also the link attacks that do not cause the worst damage but are still severe (e.g., the second and third worst-case link attacks). The Lex-Control algorithm achieves this property by scattering the costs incurred by the link attacks as evenly as possible over all the links in a network, or equivalently solving a *lexicographic-optimization* problem. This type of problem was well studied in [9], in which a centralized solution is proposed for load balancing. Our Lex-Control algorithm, instead, provides a distributed method that solves lexicographic optimization to counter severe link attacks.

By simulation, we evaluate the resilience of the Bound-Control algorithm and the Lex-Control algorithm against uniform, proportional, and worst-case attacks on single or multiple links. Our results indicate substantial improvement over single-path alternatives. For instance, in a 200-node, 1000-link network, the Bound-Control algorithm decreases the cost incurred by the worst-case single-link attack by 78% when compared to the single-path approach. The simulation also illustrates that the Lex-Control algorithm reduces by more than 50% the number of links that can incur severe damage due to single-link attacks, and this reduction is realized after only three or four iterations. Thus, by executing only a few rounds of the Lex-Control algorithm, we can improve algorithm efficiency with only minor degradations in routing security.

The paper proceeds as follows. In Section II, we formulate the secure multipath approach. Sections III and IV present and validate the Bound-Control algorithm and the Lex-Control algorithm, respectively. In Section V, we report several experiments that evaluate the algorithms. Specifically, we assess the Lex-Control algorithm in response to the uniform, proportional, and worst-case link attacks. Section VI compares our algorithms with related work. Section VII discusses the limitations of our work and suggests future directions. Section VIII concludes.

TABLE I
IMPORTANT NOTATION USED IN THIS PAPER

$\mathcal{N}$	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 $\mathbf{a}$
f_s	flow value broadcast by source s (see Section III)
U	sufficiently large value (see Sections III and IV)
$\mathcal{G}_{f^*}$	residual network with respect to f^* (see Section IV)

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], [7], [9], [11], [19]. To aid our discussion, Table I summarizes the notation that we use throughout the paper.

Our discussion relies on the concepts of the maximum flow and the minimum cut. 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) [2]. Suppose that we partition the nodes into two sets $\mathcal{S}$ and $\mathcal{T}$, where $s \in \mathcal{S}$ and $t \in \mathcal{T}$. A *cut* refers to the set of links directed from $\mathcal{S}$ to $\mathcal{T}$. 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 [2].

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 focuses on a single-link attack (see Section I). Quantifying the damage of the attack is very difficult, and we leave its investigation as future work. In this paper, we assume that the damage of the attack on link $l \in \mathcal{L}$ is characterized as an *attack cost* $a_l = c_l x_l$, where c_l denotes the *security constant* of link l. The security constant 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 [4], the failure probability of link l [7], or the proportion of loss of data traversing link l when it is attacked. Note that for the security constant c_l to have a consistent interpretation across different links l, it has to be calibrated with respect to an agreed upon definition of an attack. An example would be quantifying the resources employed by an intruder, and then computing the success probability of an attack for different links given the same amount of the intruder's resources. A precise quantification of c_l however is not the focus of this work, and we assume the existence of an agreed upon definition of c_l. Every node u can determine in advance the security constants c_l for its own outgoing links $l \in \mathcal{L}(u)$, where $\mathcal{L}(u)$ is the set of all outgoing links of node u, using security monitoring systems [19] or statistical measurements [7]. For instance, in a hybrid wired/wireless network, those applications of measuring security constants naturally lead to higher security constants for wireless links as opposed to wired links, indicating the higher susceptibility of wireless links to subversion. In order to be consistent with the interpretation of c_l as a probability or proportion, we require that $0 \leq c_l \leq 1$.

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* [11], 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 [11], the authors explain how to implement the Preflow-Push algorithm in a distributed and asynchronous fashion. We refer readers there for a detailed discussion.

Let $\mathbf{f} = (f_l, l \in \mathcal{L})$ be the flow vector where f_l denotes the flow carried by link l, and f be the net flow entering sink t. Problem 1 can therefore 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:

$$\begin{aligned} a^* &= 1/f^*, \\ x_l &= f_l/f^*, \ \forall l \in \mathcal{L}. \end{aligned}$$

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:

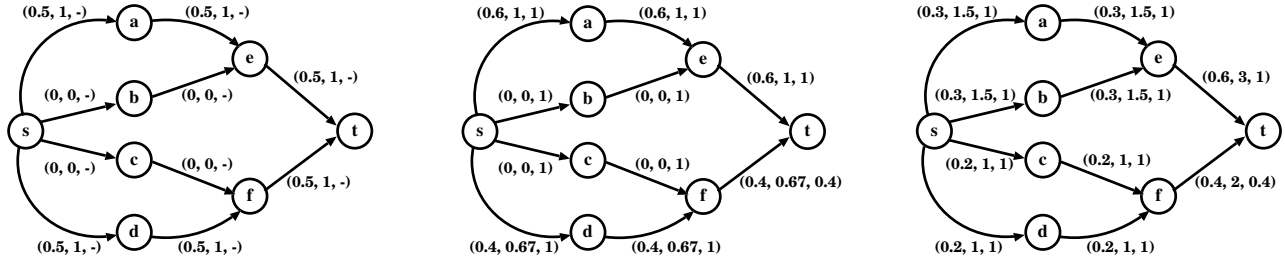
$$\begin{aligned} a^* &= \min_{\mathbf{x}} \max_{l \in \mathcal{L}} a_l = \min_{\mathbf{x}} \max_{l \in \mathcal{L}} c_l x_l \\ \text{subject to} &\quad 0 \leq x_l \leq b_l, \ \forall l \in \mathcal{L}. \end{aligned} \tag{3}$$

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, where $l \in \mathcal{L}$) represents the maximum amount of data that can be sent across a link, and the term *capacity* (i.e., $cap(l) = \min(1/c_l, b_l f)$, where $l \in \mathcal{L}$) 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.

[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).

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. The solutions $\mathbf{f}$ and $\mathbf{x}$ are adjusted accordingly to satisfy the fraction bounds.

Similar to Problem 1, we can solve Problem 3 in a centralized manner via linear programming. To implement a distributed approach, we develop the *Bound-Control algorithm* that is built upon the Preflow-Push algorithm to solve Problem 4 and hence Problem 3. Section III describes the algorithm and formally proves its correctness.

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 some 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 $\text{lexmin}\,(.)$ be the function that returns the lexicographically minimum sequence $\mathbf{a}^*$. We then express the lexicographic-optimization problem as:

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

Hence, the corresponding maximum-flow problem is:

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

This type of lexicographic-optimization problem was analyzed in [9], whose solution is centralized and requires the knowledge of the whole network state. 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 Problem 6 and hence Problem 5 in a distributed fashion.

III. Bound-Control Algorithm

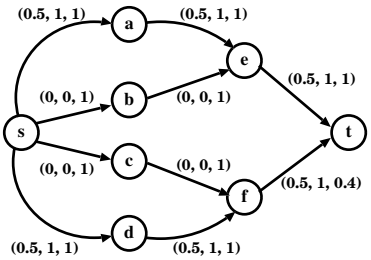
This section presents the *Bound-Control algorithm*, which solves Problem 4, the maximum-flow problem in which the fraction bound b_l is imposed on every link $l \in \mathcal{L}$. We describe its working mechanism, prove its correctness, and finally address how it supports both fixed-rate and maximal-rate session models described in Section I.

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.

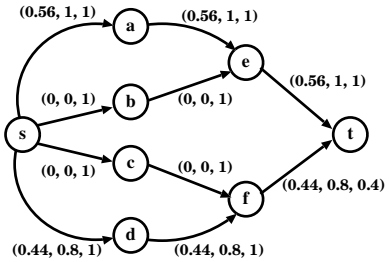
A. Description of the Bound-Control Algorithm

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 shown in Algorithm 1.

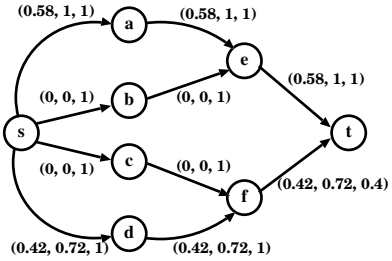
In Algorithm 1, source s first broadcasts a sufficiently large value $f_s = U$ to initiate the Bound-Control algorithm (line 1). Next, all network nodes execute the Preflow-Push algorithm subject to the link-capacity constraint $cap(l) = \min(1/c_l, b_l f_s) = 1/c_l$ for every link $l \in \mathcal{L}$ (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 maximum flow obtained from the Preflow-Push algorithm



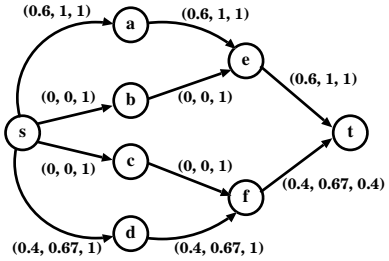
(a) 1st Preflow-Push: maximum flow = 2



(b) 2nd Preflow-Push: maximum flow = 1.8



(c) 3rd Preflow-Push: maximum flow = 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 flow values after the first three executions of the Preflow-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 *Preflow-Push*
6: **repeat**
7: source s sets f_s to be the maximum-flow 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 *Preflow-Push*
13: **until** source s finds that f_s equals the maximum-flow result

equals the flow value f_s that has just been broadcast (line 13). The optimal value f^* is given by f_s (as will be shown later).

Figure 2 illustrates the Bound-Control algorithm in Algorithm 1 for the network shown in Figure 1. Figure 2(a) shows the values of f_l and x_l for every link $l \in \mathcal{L}$ after the first execution of the Preflow-Push algorithm (line 5). Source s then broadcasts $f_s = 2$ to the network (line 8). Node f subsequently sets the capacity of its outgoing link to sink t to be $cap(l) = \min(1/c_l, b_l f_s) = \min(1, 0.8) = 0.8$, while the capacities $cap(l)$ of other links remain one. Figures 2(b) and 2(c) show the flow values after the second and third executions of the Preflow-Push algorithm (line 12). Upon termination, the Bound-Control algorithm returns the maximum flow $f^* = 1.67$ and hence the optimal attack cost $a^* = 1/f^* = 0.6$, as shown in Figure 2(d).

B. 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 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. ■

C. Discussion of the Bound-Control Algorithm

The correctness of Theorem 1 relies on the condition that $\sum_{l \in \mathcal{C}} b_l \geq 1$ for any cut $\mathcal{C}$. As stated in Section II, the fraction bound b_l is expressed as a normalized value $\min(B_l/X, 1)$ for all $l \in \mathcal{L}$, where X and B_l refer to the feasible session throughput and the bandwidth of link l, respectively. As X is a feasible throughput, we must have $\sum_{l \in \mathcal{C}} B_l \geq X$ for any cut $\mathcal{C}$. Thus, we have the scaled sum $\sum_{l \in \mathcal{C}} b_l \geq 1$ for any cut $\mathcal{C}$, and Theorem 1 is applicable.

In actual implementation, we can provide support for 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, it means 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. Based on the proof of Lemma 2, we can show that if f_s starts with a sufficiently small positive value, then the broadcast value f_s is increasing to the optimal value f^*. As a result, we can employ *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, 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 [11], it introduces $O(p|\mathcal{N}|^2|\mathcal{L}|)$ messages and takes $O(p|\mathcal{N}|^2)$ time to converge.

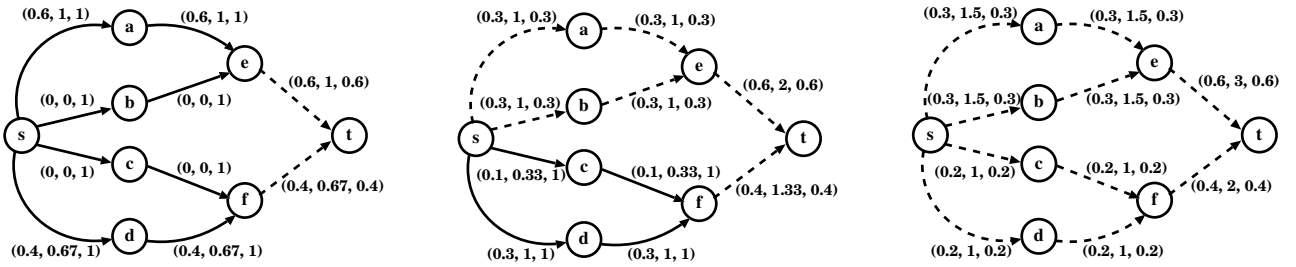
IV. Lex-Control Algorithm

In this section, we present the Lex-Control algorithm, which solves the lexicographic optimization specified in Problem 6 and hence Problem 5. We explain how the Lex-Control algorithm is extended from the Bound-Control algorithm, and then prove its correctness.

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 minimized the 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 present two properties that indicate how to pinpoint the critical links.



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

Property 1: In a maximum-flow problem, 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. ∎

To help present the next property, 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 $\mathcal{L}_{f^*}$.

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 $\mathcal{G}_{f^*}$, 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.

Algorithm 2 Lex-Control

```
1:  all nodes run Bound-Control
2:  source s sets f* to be the computed maximum flow
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 flow
```

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 $\mathcal{G}_{f^*}$ to determine if its outgoing links are critical (lines 6-8). It modifies c_l and b_l for each spotted critical link l (lines 9-10) which adjusts capacity $cap(l)$ to bound only the proportion of flow currently carried (since $1/c_l = U$ becomes very large and does not affect $cap(l)$). Since b_l is set to the proportion of flow carried by the critical link l (line 10), we still guarantee $\sum_{l \in \mathcal{C}} b_l \geq 1$ for any cut $\mathcal{C}$, and hence, by Theorem 1, guarantee the convergence of the later executions of the Bound-Control algorithm. 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 the sufficiently large value U. Figure 3 depicts how the Lex-Control algorithm evaluates the lexicographically optimal solution for the network shown in Figure 1.

B. Correctness of the Lex-Control Algorithm

Here, we formally prove that the Lex-Control algorithm converges to the lexicographically optimal solution $\mathbf{a}^*$ for Problems 5 and 6.

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 $\mathbf{a}^*$. ■

C. Discussion of the Lex-Control Algorithm

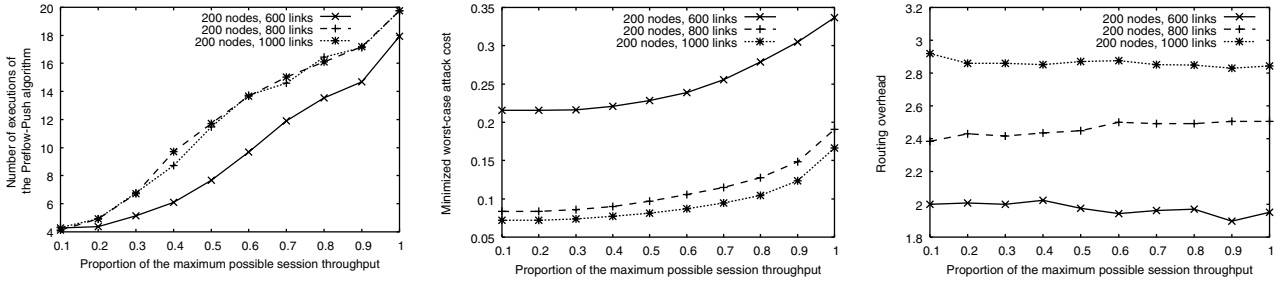
The complexity of the Lex-Control algorithm is dominated by the executions of the Bound-Control algorithm. Since each lexicographic iteration discovers at least one critical link, the Lex-Control algorithm has a complexity that is $O(|\mathcal{L}|T')$, where T' is complexity of the Bound-Control algorithm.

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 the 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 [21], a network topology generator, to construct 50 experimental topologies for each network setting. All nodes within a topology are randomly connected and randomly 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 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.

The version of the Bound-Control algorithm that we evaluate implements the bisection-search technique (see Section III-C). Also, the Lex-Control algorithm that we consider terminates after a fixed number of lexicographic iterations (see Section IV-C).



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

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

Our experiments focus on three metrics, namely:

- *Number of executions of the Preflow-Push algorithm*: We use this metric to assess the message complexity and the convergence time of the proposed algorithms.
- *Attack cost* (defined in Section II): This metric is used to measure the resilience offered by the proposed algorithms toward various types of link attacks. In the experiments, we will focus on different variants of this metric.
- *Routing overhead*: This metric is defined as the ratio of the average hop-count from source s to sink t in the multipath approach to the hop-count in the *single shortest-path approach*. By "shortest path", we mean the path that has the fewest hop-count. 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 hop-count of the shortest-path approach 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, and consider the throughput rates that are given by different proportions of the determined maximum session throughput. This addresses both fixed-rate and maximal-rate session models (see Section I). We then assign the appropriate fraction bounds to all links (see Section III-C). Finally, we apply the Bound-Control algorithm to obtain the metrics. Here, we measure the degree of resilience based on the *minimized worst-case attack cost*.

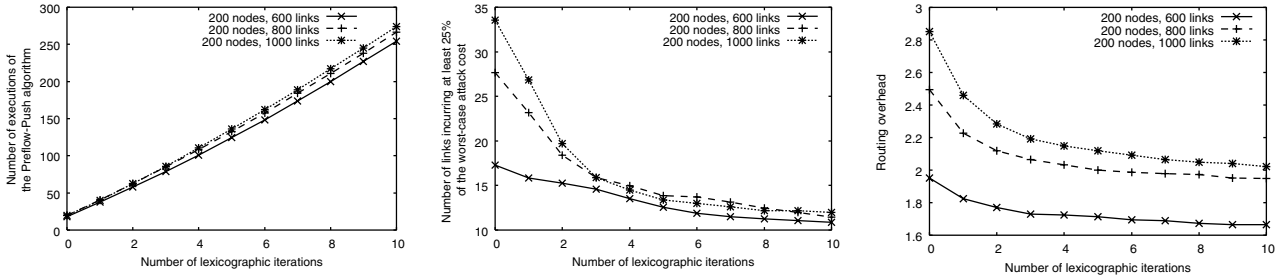
Figure 4 depicts the performance metrics at different session throughputs, and Table II shows the worst-case attack cost in the single shortest-path approach in each network setting. From Figure 4(b), we see that the Bound-Control algorithm substantially reduces the worst-case attack cost when compared to the single shortest-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 executions of the Preflow-Push 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 (since the message complexity of the Preflow-Push algorithm is proportional to the number of links according to Section III-C) as well as higher routing overhead.

TABLE II
WORST-CASE ATTACK COST IN THE SHORTEST-PATH APPROACH

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

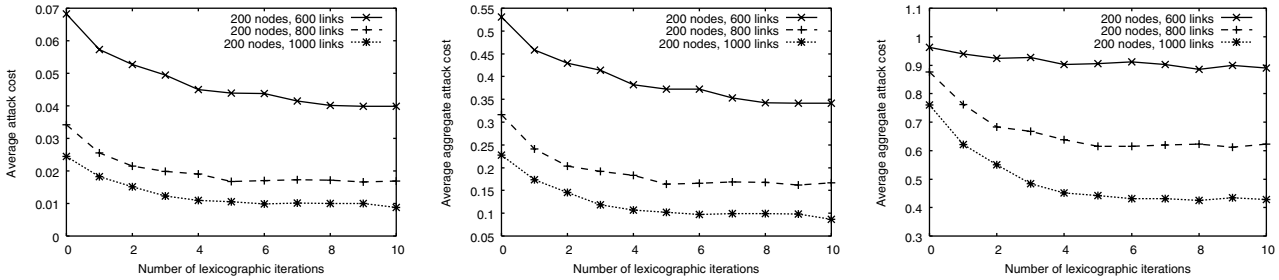
Experiment 2 (Analysis of the Lex-Control algorithm at different numbers of lexicographic iterations): This experiment considers how the Lex-Control algorithm prevents the severe single-link attacks when it executes different numbers of lexicographic iterations. We regard a single-link attack as "severe" 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 reduces 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 Figures 5(b) and 5(c), 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 executions of the Preflow-Push algorithm (b) Number of links incurring severe attack costs (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.

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.

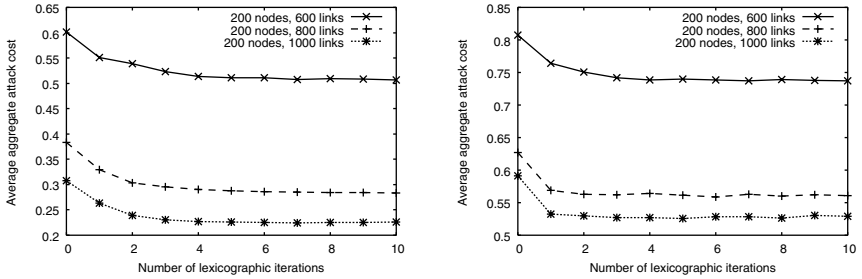
In the experiment setup, we let the security constant c_l be the proportion of loss of data traversing link l that is being attacked (see Section II), so the attack cost of link l, given by $a_l = c_l x_l$, represents the actual proportion of data loss for the data session. For the single-link attack, 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 first look at the amount of remaining data actually reaching the sink 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 abate 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. In general, the Lex-Control algorithm can mitigate the average aggregate attack costs in both proportional and worst-case attacks. For instance, in the 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 of experimental results: 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.



(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

VI. Related Work

In this section, we summarize the related work in the contexts of minimax optimization, lexicographic optimization, and secure multipath routing.

Minimax optimization has been analyzed extensively to address the issues of load balancing and network security. An excellent body of work on minimax optimization includes [1], [4], [5], [13], [14], [17], in which they consider the load-balancing problem (e.g., in [1], [13]), multipath solutions to combat link attacks (e.g., in [4], [5], [14]), and the network-intrusion problem (e.g., in [17]). While the analysis in [5] uses linear programming, [1], [13], [17] discuss how to obtain the minimax result by solving the maximum-flow problem. In particular, [13] addresses the case in which the network links are assigned different bandwidths. However, when applied to our secure multipath setting, [13] considers specifically the maximum session throughput and the zero/one security constants. For comparison, our Bound-Control algorithm is a generalization of their approach that supports the complete range of session throughputs and security constants. Note that the above studies implement the centralized algorithms that assume the knowledge of the network topology. We extend beyond this previous work by devising a distributed solution to the minimax-optimization problem subject to the link-bandwidth constraints.

The authors in [8], [9] address lexicographic optimization in the network setting. While [8] considers only the lexicographic optimization of the flows of the links attached to the source node, [9] extends [1] to lexicographically optimize the flows of all the links in the network. Specifically, the idea of [9] 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 exhibits several distinctions from [9]. First, while [9] attempts to maintain the proportion of flow of the links attached to the source node (with respect to the entire network or each subdivided network), we keep the proportions of flow of all the critical links located throughout the network. Besides, while the analysis in [9] assumes no link-bandwidth constraint, we explicitly incorporate this constraint into our algorithm in the analysis. Most importantly, our Lex-Control algorithm allows distributed implementation since no network subdivision is required, while their solution is centralized.

Extensive studies regarding secure multipath routing can be found in [4], [5], [14], [19], [26]. Instead of solving the problem via minimax optimization as in [4], [5], [14], the authors in [19], [26] explore the node-disjoint paths among the nodes that experience various attack probabilities. Our work, like [4], [5], [14], relaxes the disjointness requirement and allows shared paths. As a result, a higher degree of network diversity is acquired.

VII. Discussion and Future Work

In this section, we address several limitations in our current work and suggest directions for future research.

Data recovery: While our secure multipath approach diminishes the damage brought by link attacks, it should include a data-recovery mechanism in implementation so that receivers can recover all data as long as the scale of damage is modest. One example of the data-recovery mechanism is the *threshold secret sharing system* [19], which introduces redundancy to the transferred data to enable receivers to obtain complete information from partially received data. Although the redundancy provides data reliability, it reduces the effective session throughput as well. We therefore need further investigation on this trade-off and the implementation considerations.

Fault-tolerance: Our work focuses on proactive protection, but does not focus on reacting to link failures. We have assumed 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 [10], [15]. In particular, [15] 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. In fact, if we incorporate the data-recovery mechanism described above, we can sustain the presence of faulty links. Hence, we need to consider the trade-offs between restarting the algorithms and invoking the self-stabilizing procedures.

Implementation: Our analysis is based on the complexity of the Preflow-Push algorithm, yet the actual message complexity and the convergence time in real network settings are unknown. Thus, we need an implementation prototype for more detailed analysis.

VIII. Conclusion

We presented the distributed secure multipath approach that encompasses two algorithms: the Bound-Control algorithm and the Lex-Control algorithm, both of which can proactively combat link attacks in a distributed fashion. We validated that both algorithms converge to the desired optimal solutions, and evaluated the algorithms through simulations to demonstrate their resilience toward different patterns of single-link and multi-link attacks. In particular, the simulations demonstrate that the Lex-Control algorithm counters severe link attacks efficiently within the first few lexicographic iterations. This implies that both routing security and algorithm performance can be effectively achieved during actual implementation.

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 *Proceedings of the 18th ACM Symposium on Operating Systems Principles (SOSP)*, October 2001.

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

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

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

[7] A. Fumagalli and M. Tacca. Optimal Design of Optical Ring Networks with Differentiated Reliability (DiR). In *Proceedings of the International Workshop on Quality of Service in Multiservice IP Networks*, January 2001.

[8] 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.

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

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

[11] 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.

[12] P. Gopalan, S. C. Han, D. K. Y. Yau, X. Jiang, P. Zaroo, and J. C. S. Lui. Application Performance on the CROSS/Linux Software-Programmable Router. CS TR-01-019, Dept of Computer Sciences, Purdue University, November 2001.

[13] 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.

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

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

[16] 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.

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

[18] D. Loguinov and H. Radha. End-to-End Internet Video Traffic Dynamics: Statistical Study and Analysis. In *Proceedings of IEEE INFOCOM*, June 2002.

[19] W. Lou, W. Liu, and Y. Fang. SPREAD: Enhancing Data Confidentiality in Mobile Ad Hoc Networks. In *Proceedings of IEEE INFOCOM*, March 2004.

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

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

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

[23] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A Scalable Content-Addressable Network. In *Proceedings of ACM SIGCOMM*, 2001.

[24] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan. Chord: A Scalable Peer-To-Peer Lookup Service for Internet Applications. In *Proceedings of ACM SIGCOMM*, 2001.

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

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