

Game Theoretic Stochastic Routing for Fault Tolerance and Security in Computer Networks

Stephan Bohacek, *Member, IEEE*, João P. Hespanha, *Senior Member, IEEE*, Junsoo Lee, *Member, IEEE*, Chansook Lim, *Member, IEEE*, and Katia Obraczka, *Member, IEEE*

Abstract—We introduce the Game-Theoretic Stochastic Routing (GTSR) framework, a proactive alternative to today's reactive approaches to route repair. GTSR minimizes the impact of link and router failure by 1) computing multiple paths between source and destination and 2) selecting among these paths randomly to forward packets. Besides improving fault tolerance, the fact that GTSR makes packets take random paths from source to destination also improves security. In particular, it makes connection eavesdropping attacks maximally difficult as the attacker would have to listen on all possible routes. The approaches developed are suitable for network layer routing, as well as for application layer overlay routing and multipath transport protocols such as the Stream Control Transmission Protocol (SCTP). Through simulations, we validate our theoretical results and show how the resulting routing algorithms perform in terms of the security/fault-tolerant/delay/throughput trade-off. We also show that a beneficial side effect of these algorithms is an increase in throughput, as they make use of multiple paths.

Index Terms—Multipath routing, stochastic routing, game theory, network security, fault tolerance.

1 INTRODUCTION

“TRADITIONAL” routing protocols forward packets over a single path from source to destination. This means that, even if redundant resources are available, a single failure (accidental or due to malicious activities) temporarily interrupts all connections that use the compromised route. Eventually, the underlying routing protocol will react to the failure and correct it, but this can take a nonnegligible amount of time, depending on how often routers ping/poll one another and how frequently routing updates are distributed. Another undesirable aspect of single-path routing is that packet interception or eavesdropping can be achieved with a minimum amount of resources because the path over which the packets travel is predictable. For example, by tapping into one of the links along a path, an attacker can reconstruct an unencrypted file transfer from eavesdropped packets and, by breaking into one router, a man-in-the-middle attack can be launched on encrypted transfers that utilize key exchange [1].

In this paper, we describe *game-theoretic stochastic routing* (GTSR), which can be viewed as a proactive alternative to today's reactive approaches to route repair. GTSR explores

the existence of multiple paths between network nodes and routes packets to minimize predictability. It discovers all paths between a source-destination pair and determines the next-hop probabilities, that is, the probability with which a packet takes a particular next hop along one of the possible paths. Unlike security and fault-tolerant mechanisms that are based on detection and response, GTSR takes a *proactive* approach to making connections less vulnerable to failures or attacks.

Throughout this paper, we use *nodes* to refer to the network elements that forward packets. Such nodes can be network layer routers, end systems (or hosts) that participate in application layer routing (for example, in an overlay network such as Resilient Overlay Networks (RON) [2]), or act as transport layer end-points (for example, in Stream Control Transmission Protocol (SCTP) that supports splitting of data streams over multiple paths [3]). From this perspective, GTSR can operate at any layer of the protocol stack. In the remainder of this section, we put GTSR in perspective by reviewing some existing approaches to failure prevention and recovery.

1.1 Failures, Interception, and Eavesdropping

The Internet is particularly vulnerable to failures and attacks because data packets travel along a physical infrastructure that is easily probed and are composed of thousands of elements, each one of them vulnerable to attacks and failures. Dynamic routing protocols are designed to try to circumvent failures in network nodes and links. However, these protocols are purely reactive and wait for failures to be detected before taking corrective measures. Although some failures can be easily discovered (for example, if there is no carrier signal, the physical layer interface will report an error), intermittent failures are especially difficult to detect and routing often takes considerable time to recover from them. In addition, although some link-state algorithms can find alternate

- S. Bohacek and C. Lim are with the Department of Electrical and Computer Engineering, University of Delaware, Newark, DE 19716. E-mail: bohacek@eece.udel.edu, cslim@usc.edu.
- J.P. Hespanha is with the Department of Electrical and Computer Engineering, University of California Santa Barbara, Santa Barbara, CA 93106-9560. E-mail: hespanha@eece.ucsb.edu.
- J. Lee is with the Department of Computer Science, Sookmyung Women's University, Yongsan-ku, Seoul 140-742, Republic of Korea. E-mail: jslee@sookmyung.ac.kr.
- K. Obraczka is with the Computer Engineering Department, University of California Santa Cruz, Santa Cruz, CA 95064. E-mail: katia@cse.ucsc.edu.

Manuscript received 18 Aug. 2005; revised 15 Aug. 2004; accepted 4 Dec. 2006; published online 2 Jan. 2007.

Recommended for acceptance by G. Lee.

For information on obtaining reprints of this article, please send e-mail to: tpds@computer.org, and reference IEEECS Log Number TPDS-0375-0805. Digital Object Identifier no. 10.1109/TPDS.2007.1000.

paths relatively quickly, distance-vector algorithms may require considerably more time in large, poorly connected networks. In either case, during the detection of the cut/failure and the search for a new route, the connection remains broken.

Selective faults/attacks are especially challenging. Suppose, for example, that a node of the network is compromised and that it selectively intercepts packets from specific end-to-end connections. If routing hello packets are not intercepted, other nodes do not compute new routes to exclude the compromised node. This means that application or transport-layer retransmissions will keep following the same route and will continue to be subject to interception. A more “active” form of attack exploiting compromised nodes is to have them generate spurious routing updates, each reporting very low costs to all or some destinations. As a result, other nodes will funnel packets to these compromised nodes.

Privacy and integrity techniques (for example, end-to-end encryption, Virtual Private Networks (VPNs), and secure tunnels) are effective in protecting against eavesdropping and some forms of interception (for example, selective interception). However, if an encryption key is stolen or when defending against attackers with inside information, these techniques lose much of their effectiveness. Furthermore, man-in-the-middle attacks may still be possible. To protect a data transmission network against a wide range of attacks, one needs a suite of mechanisms spanning several layers of the protocol stack. In particular, end-to-end security mechanisms (for example, end-to-end encryption for privacy and message authentication for integrity/authenticity) should be complemented by security mechanisms at other layers (for example, link-layer encryption). GTSR is one such mechanism and complements security measures taken at other layers.

1.2 Exploring Multiple Paths

(Deterministic) flooding—that is, sending each packet along every possible path—is the simplest form of multipath routing and provides a routing mechanism that is extremely robust with respect to link/node failures. However, this type of routing is not practical because it greatly increases overhead. Flooding also significantly simplifies packet eavesdropping. We propose exploring multiple paths through *statistical flooding*. In statistical flooding, packets are sent through all available paths, but, generally, no packet is sent more than once (except for the possibility that error control at the transport layer may require a packet to be resent). Flooding thus occurs “on the average” and is randomized to minimize predictability.

Stochastic routing is the mechanism used to achieve statistical flooding. In this type of routing, nodes select the next hop to which to forward a packet in a random fashion. The next-hop probabilities—that is, the probabilities of selecting particular next-hop destinations—are design parameters and determining these probabilities is the subject of this paper. Our main challenge is determining next-hop probabilities that ensure

1. *delivery*, that is, all packets reach their desired destination with probability one,

2. *timeliness*, that is, the delay is acceptable, and
3. *statistical flooding* is achieved, that is, all paths are explored by the packet ensemble.

With respect to item 3, some network elements may be more susceptible to failures/attacks than others and, therefore, it may be desirable to have nonuniform statistical flooding. For example, as a fault/attack detector starts to suspect that a particular element is compromised, it can adjust the next-hop probabilities to start avoiding that element. With GTSR, this can be done progressively, which allows for a measured response and provides additional flexibility in addressing the trade-off between the false alarm rate and detection speed (cf., [4] for a discussion on this trade-off).

1.3 Scope

One question to ask is “where can the GTSR methodology be effectively applied in today’s networks?” Clearly, GTSR will only be effective if there are multiple paths to explore. At the network layer, we see at least two domains in which this technique may prove useful:

1. Inside Internet service providers (ISPs), where there are often multiple independent paths to the exit point for that ISP. GTSR would be part of the suite of tools that the ISP utilizes to provide secure networking to its clients.
2. Inside an organization that builds multipath redundancy in its network to improve security (and also to augment throughput). An organization may utilize multiple connections to a single ISP or even use multiple ISPs to connect to the outside network and employ GTSR to spread data across independent paths provided by distinct ISPs.

Although GTSR can also be used for interdomain routing, it has been noted that Autonomous Systems (ASs) typically have limited information about the network topology and may have different (and often conflicting) path-selection goals [5]. This can limit the viability of GTSR for interdomain routing.

At application layer overlay networks, the applicability of GTSR depends on the degree to which links in the overlay network are truly independent (that is, do not share any physical links/nodes). Although the presence of dependent paths does not affect the applicability of GTSR, if there are no independent paths at the network layer, then GTSR may not significantly increase security/reliability. On the other hand, if attacks target application layer routers instead of packet transmission (that is, if the risk is that end hosts taking part in the forwarding may be compromised), then GTSR can increase reliability and security as long as the overlay network has multiple distinct paths between the source and the destination.

At the transport layer, protocols such as SCTP allow data streams to be split over several paths. Again, the utility of splitting the flow depends on whether the different physical paths are distinct.

1.4 Outline

The remainder of this paper is organized as follows: In Section 2, we review related work. Section 3 develops

techniques for designing “optimal” stochastic routing policies. Simulations of these stochastic routing policies are presented in Section 4. Finally, Section 5 provides concluding remarks and points toward future research. Detailed derivations of the results in Section 4 are included in the Appendix. A subset of the results in this paper was presented in the *Proceedings of the 11th IEEE International Conference on Computer Communications and Networks* [6].

2 RELATED WORK

2.1 Network Robustness and Security

VPNs [7] have been used as a way to securely interconnect a (typically small) number of sites. Although private networks use dedicated lines, VPNs try to implement private networks atop a publicly accessible communication infrastructure like the Internet. VPNs typically employ some combination of encryption, authentication, and access control techniques to allow participating sites to communicate securely.

The emergence of Internet Protocol security (IPsec) [8] as an Internet Engineering Task Force (IETF) standardized protocol has prompted VPN solutions to use IPsec as the underlying network-layer protocol. As in any encryption-based mechanism, the key challenge in IPsec is that secret keys must remain secret. As previously discussed, if an attacker is able to infiltrate a node or has “inside” information, shared and/or private keys may be compromised and the corresponding communication channels become insecure.

Onion routing [9] is another approach to security that focuses on hiding the identities of the communicators. It uses several layers of encryption, where each layer is used to encrypt the transmission between routers on each end of a link. Because of the many layers of encryption, routers are unable to decrypt the data or even the source and destination addresses. All that a router can decipher is the next-hop information. Although onion routing is very effective for anonymity, it is computationally heavy: Each connection must be built and torn down, routers must encode and decode packets, and memory-intensive source routing is used. The Secure Border Gateway Protocol (S-BGP) [10] makes use of public key and an authorization infrastructure, as well as IPsec to verify the authenticity and authorization of control traffic generated by the Border Gateway Protocol.

Another related effort is the RON project [2] whose goal is to improve the performance and robustness of network-layer routing. RON nodes monitor current routing paths and decide whether to choose other routes (by selecting alternate application-layer paths through other RON nodes) in order to meet application-specific performance requirements. GTSR could be implemented in such networks to improve security and fault tolerance.

The approach developed here is, in some ways, similar to that presented in [11], where members of a group cooperate to maintain their anonymity to the server. Specifically, users send their request not directly to the server but to random users in the group. Each user can then forward the request either to the server or to another member of the group. In

GTSR, randomization is applied at the network layer with the goal of protecting the data packet not from the destination but from intermediate attackers.

Equal-Cost Multipath (ECMP) [12] and Open Shortest Path First Protocol (OSPF) Optimized Multipath (OSPF-OMP) [13] are, in a way, multipath routing algorithms. Although these methods insist that the alternative paths be of equal cost, MPLS allows different paths besides shortest paths to be utilized. However, none of these methods is stochastic; rather, routers employ techniques to ensure that a connection will utilize a single path.¹ Hence, a link failure will result in cut connections and eavesdropping at a single link will expose connections going through that link. Furthermore, these algorithms were developed to increase throughput, but not to make routing robust to attacks or failures. Hence, there is no mechanism in place to avoid links that may have been compromised or have been subject to high failure rates. In short, these, as well as other load balancing techniques, seek to optimize performance metrics such as bit rate and delay, whereas, in GTSR, we optimize fault tolerance and security.

2.2 Game Theoretical Approaches to Network Routing

Game theoretical approaches have been proposed for many resource allocation problems in computer networks, but none is very closely related to GTSR. We review the most relevant results in routing and refer the reader to the survey in [14] for applications of game theory to other networking problems such as bandwidth allocation, congestion control, power control in wireless networks, and medium access control.

Economides and Silvester [15] consider the selection of one among L parallel queuing systems to transport (or process) data. Two classes of packets need to be processed: The α -class packets can be queued, whereas the β -class packets are always blocked when the server is busy. The problem is formulated as a game played between two players P_α and P_β , each one controlling the fraction of packets of a particular class that goes through each of the L queues. This problem has a unique Nash equilibrium (NE) when P_α minimizes the queuing delay and P_β minimizes the blocking probability.

The paper by Yamaoka and Sakai [16] considers a network with arbitrary topology in which every link has two queues with different priorities. Both queues are first in first out (FIFO), but transmission of packets from the low-priority queue only takes place when the high-priority queue becomes empty. An independent nonzero-sum game is played at each node of the network, where one player (router) decides which outgoing link to use and the other player (network) decides whether the packet should be placed in the high or the low-priority queue. The authors show by simulation that the resulting routing algorithm decreases the overall number of packets that are dropped by time-out.

1. One reason that a connection is restricted to a single path is that many of today’s implementations of TCP do not work well under persistent packet reordering. However, as is discussed in Section 4, advancements in TCP have relieved this limitation.

In [17], a group of nodes in a sensor network wants to construct a routing tree rooted at a destination node. This problem is formulated as a nonzero-sum game in which every node is viewed as a player that independently decides its next hop to maximize an estimate of end-to-end path reliability from which it subtracts a next-hop communication cost. For each candidate next-hop node, the path-reliability estimate depends on the reliability of the next-hop node and the probability that the link between the two nodes will fail. It turns out that computing the NE for this game is NP-hard.

Orda et al. [18] study the uniqueness of NE for the “selfish routing” problem, where a number of players share a network. Each player generates data at a fixed rate and needs to decide how to split its transmission among a set of alternative end-to-end paths. The players are selfish in the sense that each one attempts to minimize its own objective function, which depends on the overall distribution of flows through the shared network. Under a similar problem setup, [19] and [20] investigate how far the noncooperative NE is from the cooperative social optimum. Under appropriate simplifying assumptions on the network structure and the cost functions, these papers establish the worst-case bounds for the ratio between the noncooperative and the cooperative solutions. This line of research was further extended in [21] to the framework of repeated games, where the routing game is played repeatedly with a discounting factor. Although the abovementioned papers are focused on theoretical studies, Qui et al. [22] present an extensive simulation study showing that selfish routing can achieve close to optimal average latency in Internet-like environments.

The paper by Feigenbaum et al. [23] considers incentive-based routing in which interdomain lowest cost paths are computed based on per-packet costs reported by a network of ASs. The game aspect derives from the fact that the ASs may not report true costs in an attempt to engineer the flows so as to maximize their own profits. It is shown that a Vickrey-Clarke-Groves (VCG) mechanism produces “strategy-proof” prices to be paid to each AS for forwarding packets in transit. Strategy-proof essentially means that the ASs have no incentive for lying about their costs and should therefore report true values. However, Afergan [24] later showed that the VCG mechanism is not strategy-proof in the context of repeated games, which they argue that have more practical significance. Blanc et al. [25] also consider the issue of avoiding selfish behavior in routing. They propose a reputation-based scheme to prevent some nodes from refusing to carry other node’s traffic (the “free-loader’s” problem).

3 ROUTING GAMES

Within each deterministic router, there exists a routing table that associates each possible destination IP address with the address and physical interface of a next-hop router. The goal is to, at every hop, get “one step closer” to the destination or, in case the final destination is in the local subnet, the address and physical interface of the host.

Stochastic routing utilizes a distinct concept called *next-hop probabilities*, which map each possible destination IP

with a probability distribution of the next hop. Packets are forwarded by selecting the next hop at random but according to the next-hop probability distribution. From an end-to-end perspective, this results in data packets following random paths. The main challenge in stochastic routing is the determination of next-hop probabilities that ensure *delivery*, *timeliness*, and *statistical flooding*.

The key technical insight explored here is to formalize the stochastic routing problem as an abstract game between two players: the designer of the routing algorithm, which is represented by *routers*, and an *attacker* that attempts to intercept packets. In practice, minimizing the impact of an attack is equivalent to minimizing the impact of a worst-case failure.

We consider zero-sum games in which routers target at minimizing the time it takes for a packet to be safely transmitted, whereas the attacker’s goal is to maximize this time. It is well known from the game theory literature that the solution to such games requires the use of mixed (randomized) policies (cf., for example, [26]). In practice, the mixed solution of the minimax problem provides the probability distributions needed for the next-hop probabilities. By formalizing the problem as an optimization with a time cost, we achieve both delivery and timeliness. Statistical flooding is a consequence of the saddle solution.

There are several alternatives to formalize a game that results in adequate routing policies. In this paper, we consider two alternatives: offline *routing games* and online *games*.

In offline *games*, the attacker starts by selecting one link or one physical interface at a particular node that she will scan for packets. This choice is made before routing starts, but is not conveyed to the router, whose task is to design the next-hop probabilities to minimize the overall probability that the packet will be intercepted, assuming that the attacker made an intelligent choice. This setup can be generalized to attacks at nodes and even mixed attacks at both nodes and links. We shall see that the computation of routing policies for online games amounts to solving a linear program, requiring information about the overall network topology as in link-state routing.

In online *games*, the attacker is not forced to select a single link/node before routing starts. Instead, the attacker is allowed to scan one physical interface at every node. However, she will not be able to catch all packets that travel through the interface selected, only a fraction of these. For highly secure nodes, this fraction would be zero, whereas it would have high values for less-secure nodes. The computation of the routing policies that arise from this game can be done using dynamic programming, amenable to distributed computation as in distance-vector routing.

It is important to emphasize that the abstract games described above *are not intended to represent realistic attack scenarios*; they mostly intend to capture the facts that 1) simultaneous failures in multiple links/nodes are unlikely and, therefore, one should optimize routing for robustness against a finite number of worst-case faults and that 2) an attacker will (hopefully!) have a finite set of resources available. Moreover, we will show that the attack models described above provide computational tools to

obtain stochastic routing policies with adequate properties. Similarly, although the number of hops is generally a poor measure of routing-path quality, minimum-hop optimization is a useful tool to compute deterministic routing tables because it generally provides adequate routing trees.

As with deterministic routing, topology changes require the recomputation of the next-hop probabilities, regardless of the type of game considered. Failure to do so will generally result in lost packets. However, with stochastic routing, some degree of functionality is preserved even before the next-hop probabilities are updated. This is because some of the packets will still make it through as long as at least one of the paths from source to destination remains viable.

3.1 Stochastic Routing Policies

We consider a data transmission network with nodes $\mathcal{N} := \{1, 2, \dots, n\}$ connected by unidirectional links. We denote by \mathcal{L} the set of all links and use the notation $\vec{j_i}$ to represent a link from node j to node i . We assume that all of the nodes in the network are connected in the sense that it is possible to reach any node from any other node through a finite sequence of links. The source and destination nodes are denoted by n_{src} and n_{end} , respectively.

In a stochastic routing framework, each *stochastic routing policy* is characterized by a list of probabilities $R := \{r_\ell : \ell \in \mathcal{L}\}$ such that

$$\sum_{\ell \in \mathcal{L}[k]} r_\ell = 1, \quad \forall k \in \mathcal{N}, \quad (1)$$

where the summation is taken over the set of links $\mathcal{L}[k] \subset \mathcal{L}$ that exit from node k . Under this policy, when a packet arrives at a node $k \in \mathcal{N}$, it will be routed with probability r_ℓ through the link $\ell := k k' \in \mathcal{L}$ to the next-hop node k' . The distribution $\{r_\ell : \ell \in \mathcal{L}[k]\}$ determines the entries of the next-hop probabilities at node k , associated with the path from node n_{src} to node n_{end} . In this case, the routing table will actually be a matrix, as the next hop may depend not only on the final destination node n_{end} but also on the source node n_{src} .

In the sequel, we denote by \mathcal{R}_{sto} the set of lists that satisfy (1) and, therefore, \mathcal{R}_{sto} represents the set of all stochastic routing policies. In general, stochastic routing policies can exhibit cycles. Since cycles introduce delivery delays and can only increase the probability of packet interception, we restrict our attention to *cycle-free* routing policies, that is, policies for which a packet will never pass through the same node twice. Formally, $R \in \mathcal{R}_{\text{sto}}$ is cycle free when there is no sequence of links

$$\mathcal{S} := \{\vec{k_1 k_2}, \vec{k_2 k_3}, \dots, \vec{k_{k-1} k_k}, \vec{k_k k_1}\} \subset \mathcal{L} \quad (2)$$

with positive probabilities $r_\ell > 0$ for all $\ell \in \mathcal{S}$ starting and ending at the same node k_1 . We denote by $\mathcal{R}_{\text{no-cycle}}$ the subset of \mathcal{R}_{sto} consisting of cycle-free policies. Fig. 1 shows an example network and a cycle-free stochastic routing policy.

3.2 Online Games

In online games, the attacker has the capability of scanning a certain percentage of the packets that go through every

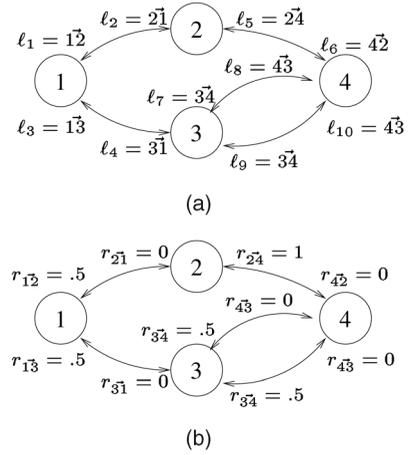


Fig. 1. (a) Example of a network with four nodes $\mathcal{N} := \{1, 2, 3, 4\}$ and 10 links $\mathcal{L} := \{\ell_1, \ell_2, \dots, \ell_{10}\}$. In this network, all connections between the nodes are assumed bidirectional, which corresponds to two unidirectional links between every two nodes that are connected. (b) Stochastic routing policy from the source node $n_{\text{src}} = 1$ to the destination node $n_{\text{end}} = 4$. Under this policy, at each node, the packets are routed with equal probability among the possible options. This policy is cycle

free. She then has to decide on which physical interfaces she should concentrate her efforts. For each node $k \in \mathcal{N}$, we denote by p_k the total percentage of packets that the attacker can scan on node k .

In the context of online games, a *stochastic attack policy* is a list of probabilities $A := \{a_\ell : \ell \in \mathcal{L}\} \in \mathcal{R}_{\text{sto}}$ that specifies which percentage of the packets will be scanned on each output interface of each node. The probability of a packet being scanned in the node k interface connected to the link $\ell := k k' \in \mathcal{L}$ is therefore given by $p_k a_\ell$. Since $A \in \mathcal{R}_{\text{sto}}$, all probabilities a_ℓ that exit node k must add up to one and, therefore, the total percentage of packets scanned in node k is indeed p_k . The selection of links by the router is done according to a stochastic routing policy $R := \{r_\ell : \ell \in \mathcal{L}\} \in \mathcal{R}_{\text{sto}}$ as described before.

For each link $\ell \in \mathcal{L}$, we denote by $\tau_\ell > 0$ the time it takes for a packet to traverse it assuming that it was not intercepted. When there is interception, we assume that this time increases by T_ℓ . Denoting by \mathbf{T}^* the amount of time that takes a packet to get from the source node n_{src} to the destination node n_{end} , the router selects a stochastic routing policy $R \in \mathcal{R}_{\text{sto}}$ so as to minimize the expected value of \mathbf{T}^* , whereas the attacker selects a stochastic attack policy $A \in \mathcal{R}_{\text{sto}}$ to maximize it. The problem just formulated is a zero-sum game for which we will attempt to find an optimal saddle pair of policies $R^* \in \mathcal{R}_{\text{sto}}$, $A^* \in \mathcal{R}_{\text{sto}}$ for which

$$\begin{aligned} E_{R^*, A^*}[\mathbf{T}^*] &= \min_{R \in \mathcal{R}_{\text{sto}}} \max_{A \in \mathcal{R}_{\text{sto}}} E_{R, A}[\mathbf{T}^*] \\ &= \max_{A \in \mathcal{R}_{\text{sto}}} \min_{R \in \mathcal{R}_{\text{sto}}} E_{R, A}[\mathbf{T}^*]. \end{aligned} \quad (3)$$

In the above equation, the subscripts on the expected value E emphasize the fact that the expectation depends on the routing/attack policies. The choice of a routing policy R^* for which (3) holds guarantees the *best possible timeliness*

(smallest \mathbf{T}^*), assuming an intelligent attacker that does her best to prevent this. This is achieved by statistical flooding as not fully exploring the available paths would be used by the attacker to her advantage.

The computation of the optimal routing and attack policies can be done using dynamic programming. To this effect, for each node $k \in \mathcal{N}$, we denote by V_k^* the *optimal cost to go from node k* , which is the average time it takes to send a packet from node k to node n_{end} using optimal policies for each player. Once the optimal cost to go has been computed for all nodes, the optimal routing policy can be computed as follows: For each node k , the next-hop routing distribution $\{r_\ell^* : \ell \in \mathcal{L}[k]\}$ over the set of links $\mathcal{L}[k]$ that exit node k is the solution to the following minimization:

$$\arg \min_{\sum_{\ell} r_\ell = 1} \max_{a_\ell: \ell \in \mathcal{L}[k]} \sum_{\vec{k}\vec{i}, \vec{k}\vec{j} \in \mathcal{L}[k]} r_{\vec{k}\vec{i}} a_{\vec{k}\vec{j}} m_{ijk}^*,$$

$$m_{ijk}^* := \begin{cases} V_i^* + \tau_{\vec{k}\vec{i}} & i \neq j, k \neq n_{\text{end}} \\ V_i^* + \tau_{\vec{k}\vec{i}} + p_k T_{\vec{k}\vec{i}} & i = j, k \neq n_{\text{end}} \\ 0 & k = n_{\text{end}}. \end{cases} \quad (4)$$

The optimal cost-to-go, V_k^* , can be computed with the help of the following function T that transforms a vector of cost-to-go $V := [V_1 \ V_2 \ \dots \ V_N]$ into another vector of cost-to-go $V' := [V'_1 \ V'_2 \ \dots \ V'_N] = T([V_1 \ V_2 \ \dots \ V_N])$, where the k th element of V' is given by

$$V'_k = \min_{\sum_{\ell} r_\ell = 1} \max_{a_\ell: \ell \in \mathcal{L}[k]} \sum_{\vec{k}\vec{i}, \vec{k}\vec{j} \in \mathcal{L}[k]} r_{\vec{k}\vec{i}} a_{\vec{k}\vec{j}} m_{ijk}$$

$$m_{ijk} := \begin{cases} V_i + \tau_{\vec{k}\vec{i}} & i \neq j, k \neq n_{\text{end}} \\ V_i + \tau_{\vec{k}\vec{i}} + p_k T_{\vec{k}\vec{i}} & i = j, k \neq n_{\text{end}} \\ 0 & k = n_{\text{end}}. \end{cases} \quad (5)$$

We defer to Section 3.2.2 how these optimizations can be performed and proceed to state several important properties of the policies just defined (refer to the Appendix for their proofs).

Theorem 1. Assume that all of the $\tau_\ell > 0$, $\ell \in \mathcal{L}$. Then:

1. The vector $V^* := [V_1^* \ V_2^* \ \dots \ V_N^*]$ of optimal cost-to-go is the unique fixed point of the function T defined by (5).
2. For any (not necessarily optimal) vector of cost-to-go $V(0)$, the optimal cost-to-go can be obtained using

$$V^* = \lim_{i \rightarrow \infty} \underbrace{T(T(\dots(T(V(0)))) \dots)}_{i \text{ times}}.$$

3. The stochastic routing policy $R^* \in \mathcal{R}_{\text{sto}}$ defined by (4) and the stochastic attacker policy $A^* \in \mathcal{R}_{\text{sto}}$ defined by a similar expression with the *min* and *max* interchanged form a saddle point pair for the game, that is, (3) holds.

3.2.1 Heterogeneous Attacks

By choosing distinct percentages p_k for different nodes, we can take into account the fact that some nodes may be more secure than others. In practice, by choosing high values for p_k , we are implicitly assuming that the node k is less secure.

One can also encode in the p_k external information about where an attack is more likely to occur and/or succeed. This information could be obtained, for example, from intrusion detection sensors that provide indications of where an attack may be occurring. In this case, the routing algorithm could actually adapt to the changing perception as the intrusion detection sensors gain more confidence about which hosts have been compromised.

The percentages p_k can also be viewed as design parameters that allow a traffic engineer to shape the amount of data sent that goes through different sections of the network. For example, by increasing the values of the p_k , one can obtain routing policies that avoid links that are especially costly or that are somehow undesirable. To some extent, the percentages p_k can play the role of link costs in the shortest path routing.

3.2.2 Computational Issues

As mentioned above, the optimal routing policies can be obtained from (4) as soon as the vector of optimal cost-to-go $V^* := [V_1^* \ V_2^* \ \dots \ V_N^*]$ is available. According to Statement 2 of Theorem 1, one can obtain V^* using an iteration of the form

$$V(t+1) = T(V(t)), \quad (6)$$

where T is defined by (5) and the initial vector $V(0)$ can be anything (for example, equal to zeros for every node). In general, V^* is only obtained as $t \rightarrow \infty$, but one can usually stop the iteration after a finite number of steps at the expense of getting a slightly suboptimal policy. Typically, the iteration is stopped when there is a small change in $V(t)$. The number of steps needed is usually on the order of the maximum diameter of the graph.

To compute each iteration of (6), we need to solve the minimax optimization that appears in the definition (5) of the operator T . This can be done using the following linear program:

$$\frac{1}{V'_k} = \max \sum_{\vec{k}\vec{i} \in \mathcal{L}[k]} \tilde{r}_{\vec{k}\vec{i}}$$

subject to $\tilde{r}_{\vec{k}\vec{i}} \geq 0$, $\sum_{\vec{k}\vec{j} \in \mathcal{L}[k]} \tilde{r}_{\vec{k}\vec{j}} m_{ijk} \leq 1$, $\forall \vec{k}\vec{j} \in \mathcal{L}[k]$, (7)

from which one obtains V'_k , which is the k th element of $T(V)$ [26, Section 2.3]. The constants m_{ijk} that appear in (7) are defined in (5). Once the optimal cost-to-go V^* has been obtained, the optimal next-hop routing distribution $\{r_\ell^* : \ell \in \mathcal{L}[k]\}$ over the set of links that exit node k can be obtained from

$$r_{\vec{k}\vec{i}}^* = \frac{\tilde{r}_{\vec{k}\vec{i}}}{\sum_{\vec{k}\vec{j} \in \mathcal{L}[k]} \tilde{r}_{\vec{k}\vec{j}}},$$

where the $\tilde{r}_{\vec{k}\vec{i}}$ are the solutions to a linear program like (7), but with the m_{ijk} replaced by the m_{ijk}^* that appear in (4).

The computation of V'_k in (7) only requires knowledge of the entries V_i corresponding to the neighbors i of k . In practice, this means that the computation of the k th element of the vector of cost-to-go $V(t+1)$ can be performed at the node k , provided that this node knows the elements of the vector $V(t)$ that correspond to their neighbors. Hence, the

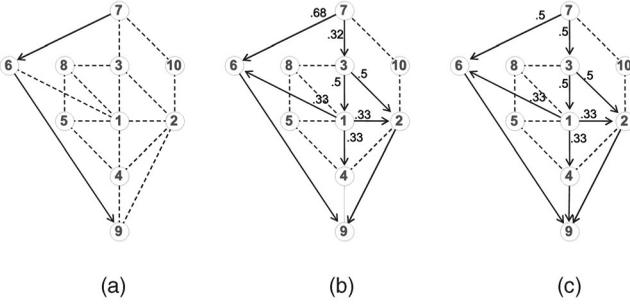


Fig. 2. Examples of routing policies obtained for an online game. When the probability of traversing a link is not zero or one, the probability is labeled near to the source end of the link. Dashed links are not used by the policies. The destination is the bottom node, labeled 9, and the total percentage of packets p_k scanned at each node k was set to 10 percent. (a) shows the routing policy obtained when $T_\ell \approx 0, \forall \ell \in \mathcal{L}$, that is, when interception poses no penalty. In this case, one obtains exactly ECMP routing [12] with hop count as the metric. In (c), we see the case $T_\ell = 100, \forall \ell \in \mathcal{L}$. In this case, there is a heavy penalty incurred when a packet is intercepted and, therefore, routing favors maximal flooding, utilizing all possible paths. (b) shows the intermediate case ($T_\ell = 45, \forall \ell \in \mathcal{L}$). Note that the link from node 7 to node 10 is not utilized, whereas the link from node 7 to node 3 is. The reason for this is that, although node 3 is no closer to the destination than node 10, there are many alternate paths from node 3 to the destination, whereas there is only one path from node 10 to the destination. Therefore, from the defender's perspective, node 3 is more appealing than node 10 because the presence of alternate paths makes the attacker's task more difficult. The path $\{7, 6, 9\}$ is similar to the path $\{7, 10, 2, 9\}$ in that, once node 6 or node 10 is selected, the rest of the path is deterministic and an online attacker could easily catch the packets. However, the path $\{7, 6, 9\}$ is shorter. Hence, under no circumstances would the path $\{7, 10, 2, 9\}$ be selected. These results should be compared to the offline example in Section 3.3.

computation of the optimal cost-to-go V^* can be distributed in the same way that RIP is. In RIP or similar distance-vector routing computations, at each stage, the router performs an operation with complexity $O(d)$, where d is the out-degree of the router. In the case of online games, the router must solve a linear programming problem with worst-case complexity $O(d^3)$, but for which there are methods that typically require much more modest complexity. Fig. 2 shows an example of routing policies obtained for online games.

3.3 Offline Games

In offline games, the attacker is only allowed to select a single link for scanning and this choice must be done ahead of time. As in online games, we assume that the attacker may not possess sufficient resources to scan every packet in the link selected. To this effect, we take as given percentages of the packets $p_\ell, \ell \in \mathcal{L}$, that the attacker would be able to scan if she were to select link ℓ . By choosing different percentages for different links, we can take into account the fact that some links may be more secure or more desirable than others (cf., discussion in Section 3.2.1). Shortly, this formulation will be extended to node attacks as opposed to link attacks.

Assuming that there is more than one independent path from source to destination, the optimal action for the attacker is to select links according to some distribution. In the context of offline games, a *stochastic attack policy* $A = \{a_\ell : \sum_\ell a_\ell = 1, \ell \in \mathcal{L}\}$ consists of a distribution over the set of links \mathcal{L} , where a_ℓ denotes the probability that the attacker

will scan link ℓ . We denote by \mathcal{A} the set of all stochastic attack policies, that is, the set of distributions over \mathcal{L} .

In offline games, the router selects a cycle-free stochastic routing policy $R \in \mathcal{R}_{\text{no-cycle}}$ so as to minimize the probability of a packet being intercepted, whereas the attacker selects a stochastic attack policy $A \in \mathcal{A}$ so as to maximize it. Denoting by χ , a random variable that takes the value one when a packet is intercepted and zero, otherwise, the problem just formulated is a zero-sum game for which we will attempt to find an optimal saddle pair of policies $R^* \in \mathcal{R}_{\text{no-cycle}}, A^* \in \mathcal{A}$ for which

$$\begin{aligned} E_{R^*, A^*}[\chi] &= \min_{R \in \mathcal{R}_{\text{no-cycle}}} \max_{A \in \mathcal{A}} E_{R, A}[\chi] \\ &= \max_{A \in \mathcal{A}} \min_{R \in \mathcal{R}_{\text{no-cycle}}} E_{R, A}[\chi]. \end{aligned} \quad (8)$$

Note that the expected value of χ is precisely the probability of a packet being intercepted. We will see shortly how this quantity can be computed from the values of the policies A and R .

The cost in (8) places no penalization on the number of links that a packet will cross from the source to the destination. However, in some cases, one may want to favor shorter paths. This can be done by introducing a new random variable $\chi_\epsilon, \epsilon \geq 0$, that is equal to zero in case the packet is not intercepted and equal to $(1 + \epsilon)^{t-1}$ when the packet is intercepted at the t hop. Suppose now that we consider the zero-sum game

$$\begin{aligned} E_{R^*, A^*}[\chi_\epsilon] &= \min_{R \in \mathcal{R}_{\text{no-cycle}}} \max_{A \in \mathcal{A}} E_{R, A}[\chi_\epsilon] \\ &= \max_{A \in \mathcal{A}} \min_{R \in \mathcal{R}_{\text{no-cycle}}} E_{R, A}[\chi_\epsilon]. \end{aligned} \quad (9)$$

For $\epsilon = 0, \chi_\epsilon = \chi$, and the cost is the same as in (8). However, for $\epsilon > 0$, (9) penalizes longer paths since the cost incurred increases as the number of hops increases. In fact, as $\epsilon \rightarrow \infty$, the potential burden of an extra hop is so large that the optimal solution will only consider paths for which the number of hops is minimal, leading to shortest path routing, but not necessarily single path. In the remainder of this section, we consider the game in (9), as (8) is a special case of the former for $\epsilon = 0$.

To compute $E_{R, A}[\chi_\epsilon]$, we need to construct the matrix C that encodes the network connectivity. We define C to be a square matrix with one column/row per link, with the (ℓ_1, ℓ_2) entry equal to one if the link $\ell_2 = i\vec{j}$ ends at the node j , where link $\ell_1 = j\vec{k}$ starts, and zero otherwise. We also define a column vector s with one entry per link, which is equal to one if the corresponding link exits the source node n_{src} and zero otherwise. In particular

$$\begin{aligned} C &:= \left[c_{\ell, \ell'} := \begin{cases} 1 & \exists i, j, k \in \mathcal{N} : \ell' = i\vec{j}, \ell = j\vec{k} \\ 0 & \text{otherwise} \end{cases} \right]_{\ell, \ell' \in \mathcal{L}}, \\ s &:= \left[s_\ell := \begin{cases} 1 & \exists k \in \mathcal{N} : \ell = n_{\text{src}}\vec{k} \\ 0 & \text{otherwise.} \end{cases} \right]_{\ell \in \mathcal{L}}. \end{aligned}$$

In the construction of C and s , we should exclude all links that enter the source node n_{src} and exit the end node n_{end} as these links will never be used in cycle-free routing policies. The order in which the links are associated with the rows/columns of C and s must be consistent. The following result,

proved in the Appendix, allows one to explicitly compute the value of $E_{R,A}[\chi_\epsilon]$.

Lemma 2. *Given $\epsilon \geq 0$, $R \in \mathcal{R}_{\text{no-cycle}}$, $A := \{a_\ell : \sum_\ell a_\ell = 1, \ell \in \mathcal{L}\} \in \mathcal{A}$*

$$E_{R,A}[\chi_\epsilon] = \text{row}[A]x, \quad (10)$$

where $\text{row}[A]$ denotes a row vector with one entry per link and $p_\ell a_\ell$ in the entry corresponding to the link $\ell \in \mathcal{L}$; x is the unique solution to

$$x = (1 + \epsilon) \text{diag}[R]Cx + \text{diag}[R]s \quad (11)$$

and $\text{diag}[R]$ is a square diagonal matrix with the r_ℓ in the main diagonal. \square

The main difficulty in solving the routing game (9) is that the cost (10) is generally not convex on $R \in \mathcal{R}_{\text{no-cycle}}$. However, the relation specified by (11) between the vectors x and policies R is in some sense one-to-one and will allow us to “convexify” the cost (10) by searching for “optimal” vectors x instead of policies R . To this effect, let C_{out} and C_{in} be matrices with one row per node and one column per link such that the entry of C_{out} corresponding to the node k and link ℓ is equal to one if link ℓ exits node k and zero otherwise, and the entry of C_{in} corresponding to the node k and link ℓ is equal to one if link ℓ enters node k , and this is not the destination node n_{end} and equal to zero otherwise, that is, $C_{\text{out}} := [c_{k,\ell}]$ and $C_{\text{in}} := [d_{k,\ell}]$ with

$$c_{k,\ell} := \begin{cases} 1 & \exists i \in \mathcal{N} : \ell = \vec{k}i \\ 0 & \text{otherwise,} \end{cases}$$

$$d_{k,\ell} := \begin{cases} 1 & \exists i \in \mathcal{N} : \ell = i\vec{k}, k \neq n_{\text{end}} \\ 0 & \text{otherwise.} \end{cases}$$

We further define s_{src} to be a column vector with one entry per node such that the entry corresponding to the source node n_{src} is equal to one and all others are equal to zero, that is

$$s_{\text{src}} := \left[s_k := \begin{cases} 1 & k = n_{\text{src}} \\ 0 & \text{otherwise} \end{cases} \right]_{k \in \mathcal{N}}.$$

In the construction of these matrices, we should exclude all links that enter the source node n_{src} and exit the end node n_{end} .

The following lemma establishes a one-to-one relation between the set of cycle-free stochastic policies (which is not convex in general) and a convex set. This will be the basis to find a solution to the routing game (9).

Lemma 3. *Let \mathcal{X} denote the convex set of vectors $x := \{x_\ell \geq 0 : \ell \in \mathcal{L}\}$ that satisfy*

$$C_{\text{out}}x = (1 + \epsilon)C_{\text{in}}x + s_{\text{src}}. \quad (12)$$

1. *Given any $R \in \mathcal{R}_{\text{no-cycle}}$, there exists an $x \in \mathcal{X}$ for which*

$$x = (1 + \epsilon) \text{diag}[R]Cx + \text{diag}[R]s \quad (13)$$

and the norms of the vectors $x \in \mathcal{X}$ can be bounded by a constant that is independent of $R \in \mathcal{R}_{\text{no-cycle}}$.

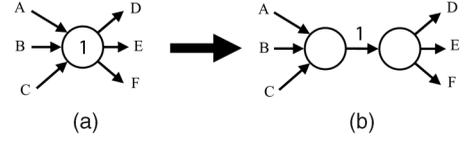


Fig. 3. Converting a node attack to a link attack. An attack on (a) node 1 is transformed into an attack on (b) link 1.

2. *Given any $x := \{x_\ell \geq 0 : \ell \in \mathcal{L}\} \in \mathcal{X}$ that is cycle-free in the sense that there is no sequence of links (2) with $x_\ell > 0, \forall \ell \in \mathcal{S}$, there exists an $R := \{r_\ell : \ell \in \mathcal{L}\} \in \mathcal{R}_{\text{no-cycle}}$ for which (13) holds, which is given by*

$$r_\ell := \frac{x_\ell}{\sum_{\ell' \in \mathcal{L}[\ell]} x_{\ell'}}, \quad \forall \ell \in \mathcal{L}, \quad (14)$$

where the summation is over the set $\mathcal{L}[\ell]$ of links that exit from the same node as ℓ .

Equation (12) can be interpreted as the *flow-conservation law*, which states that the incoming flow to a node is equal to the outgoing flow from the same node, possibly amplified by $(1 + \epsilon)$ when $\epsilon > 0$. Because of this, we call the elements of \mathcal{X} *flow vectors*.

We are now ready to state the main result of this section (proved in the Appendix), which provides the solution to offline routing games. We defer to Section 3.3.2 details on the computation of the optimal policies.

Theorem 4. *For every $\epsilon \geq 0$, the routing game (9) has saddle-point policies. Moreover, the flow game defined by*

$$\begin{aligned} \text{row}[A^*]x^* &= \min_{x \in \mathcal{X}} \max_{A \in \mathcal{A}} (\text{row}[A]x) \\ &= \max_{A \in \mathcal{A}} \min_{x \in \mathcal{X}} (\text{row}[A]x) \end{aligned} \quad (15)$$

always has a saddle point $(x^*, A^*) \in \mathcal{X} \times \mathcal{A}$ with x^* cycle-free from which we can construct a saddle point $(R^*, A^*) \in \mathcal{R}_{\text{no-cycle}} \times \mathcal{A}$ to the original routing game (9) by using (14) to compute R^* from x^* .

3.3.1 Node-Attack Variations

Although, so far, we focused our discussion on link attacks, one can easily solve the node attack problem by a suitable transformation of the network graph. When one wants to consider the possibility that one node can be attacked, one simply expands the graph by unfolding the original node into two nodes, one that is fed by all the incoming links and another from which all outgoing links exit. The two new nodes are connected by a single link from the former to the latter that will carry all packets that pass through the original node (cf., Fig. 3). An attack on the thus created “bottleneck” link in the new graph is equivalent to an attack to the node in the original graph. By choosing a suitable percentage of packets scanned p_ℓ in the new link, we effectively select the percentage of packets that can be scanned in the original node. The percentage of packets that can be scanned in the old links should be set to zero (unless one wants to consider mixed link-node attacks) and, therefore, the total number of links that are relevant for

optimization actually decreases, making the problem computationally simpler.

3.3.2 Computational Issues and Max-Flow/Load-Balancing Problems

Theorem 4 allows us to use the formula (14) to compute the optimal routing policy R^* from the saddle point (x^*, A^*) to the flow game (15), where x^* is any vector in \mathcal{X} that achieves the following minimum:

$$\min_{x \in \mathcal{X}} \max_{A \in \mathcal{A}} (\text{row}[A]x) \quad (16)$$

[26, Theorem 2.4]. It turns out that the computation of x^* in (16) can be reduced to a linear-program optimization: Since the cost is linear on A for fixed x , the inner maximization in (16) is always achieved at a distribution of the form

$$A_\ell = \left\{ \underbrace{0, 0, \dots, 0, 1, 0, \dots, 0}_{\text{single 1 at the } \ell\text{th entry}} \right\}$$

for some $\ell \in \mathcal{L}$. For such A_ℓ , we have that $\text{row}[A_\ell]x = p_\ell x_\ell$ and, therefore, (16) is equal to

$$\min_{x \in \mathcal{X}} \max_{\ell \in \mathcal{L}} (p_\ell x_\ell) = \min_{x \in \mathcal{X}} \min_{p_\ell x_\ell \leq \mu, \forall \ell} \mu = \min_{\substack{0 \leq p_\ell x_\ell \leq \mu, \forall \ell \\ C_{\text{out}} \bar{x} = (1+\epsilon)C_{\text{in}} \bar{x} + \bar{\mu} s_{\text{src}}}} \mu. \quad (17)$$

In the first equality in (17), we used the fact that $\max\{\alpha_1, \alpha_2, \dots, \alpha_k\} = \min_{\alpha_i \leq \mu, \forall i} \mu$. Although (17) is already in the form of a linear program, it is instructive to reformulate it as a maximization by making the change of variables $\bar{\mu} := \mu^{-1}$ and $\bar{x} := \mu^{-1}x$, which leads to

$$\frac{1}{V^*} = \max_{\substack{0 \leq p_\ell \bar{x}_\ell \leq 1, \forall \ell \\ C_{\text{out}} \bar{x} = (1+\epsilon)C_{\text{in}} \bar{x} + \bar{\mu} s_{\text{src}}}} \bar{\mu}. \quad (18)$$

One can recognize in this linear program a max-flow optimization showing that the value of the game is equal to the inverse of the maximum flow $\bar{\mu}$ from source to destination, consistent with the flow conservation law $C_{\text{out}} \bar{x} = (1+\epsilon)C_{\text{in}} \bar{x} + \bar{\mu} s_{\text{src}}$ and subject to the link-bandwidth constraints:

$$\bar{x}_\ell \leq \frac{1}{p_\ell}, \quad \forall \ell \in \mathcal{L}. \quad (19)$$

Note also that, since \bar{x} is simply a scaled version of x , the optimal routing policy can be computed directly from \bar{x} using (14). This means that the routing policies that arise from the routing game (9) also *maximize throughput subject to the constraint* (19). For $\epsilon = 0$, this turns out to be a standard max-flow problem, whereas, for $\epsilon > 0$, we obtain a generalized maximum flow with gain [27]. When this algorithm is applied to the transformed graph that encodes node attacks, the maximum node-flow is minimized. This means that the policies arising from the routing game (9) also achieve *load balancing*. If the links and nodes have heterogeneous p_ℓ , then a weighted load balancing is achieved. Fig. 4 shows an example of routing policies obtained for offline games.

The routing computations in (18) require that each node have knowledge of the entire topology, that is, this is a link-state algorithm. However, it seems likely that a distributed approach is feasible. In particular, in the case that $\epsilon = 0$, there are several algorithms that solve the max-flow problem in a distributed fashion [28], [29]. The time

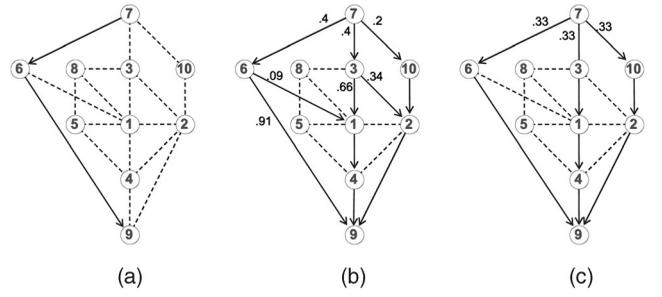


Fig. 4. Examples of routing policies obtained for an offline game. When the probability of traversing a link is not zero or one, the probability is labeled near to the source end of the link. Dashed links are not used by the policies. The source is the top node, labeled 7, and the destination is the bottom node, labeled 9. The percentage p_ℓ of packets that the attacker could scan at each link was assumed to be the same for every link ℓ , which is consistent with a network where all links are equally secure. (c) shows the routing policy obtained when one selects $\epsilon = 0$ and, therefore, delay is not penalized, leading to the maximal spreading of packets. In (a), we see that, when each extra hop corresponds to a severe cost penalty ($\epsilon = 100$), one obtains minimum-hop routing. (b) shows an intermediate case ($\epsilon = 0.1$). In contrast to that in Fig. 2, now, the link from node 7 to node 10 is, in general, utilized because it is part of one of the three independent paths from source to destination. However, in online games, if a packet was sent through node 10, it was going to be caught with high probability since it necessarily had to be sent toward node 2, so this option was not considered.

complexity of these algorithms can be as low as $O(n^2)$, where n is the number of nodes. Furthermore, a hierarchical version of GTSR would likely yield significant computational saving.

3.3.3 Routing Tables versus Routing Matrices

For online games without the back-to-start variation, the function T used to compute the optimal cost-to-go and, eventually, the optimal stochastic routing policies only depend on the destination node n_{end} and not on the source node n_{src} . In practice, this means that the optimal routing policies only depend on the final destination, as in the usual deterministic routing algorithms such as RIP, OSPF, and so forth. However, for offline games, the matrices C_{in} and C_{out} used to compute the optimal routing policies depend both on the source and destination nodes (recall that we need to remove from these matrices the contribution of the nodes entering n_{src} and exiting n_{end}). In practice, this means that the routing tables will actually be matrices as the choice of the probability distributions for the next hop depends not only on the final destination node n_{end} , but also on the source node n_{src} . A hierarchical approach would result in next-hop probabilities only depending on the source and destination prefix.

Another implementation approach is to use source routing, whereby the end hosts define the path to be taken by each packet and are then responsible for the randomization of paths. The Dynamic Source Routing (DSR) protocol is an example of a source routing mechanism which has been proposed for use in multihop wireless ad hoc networks (or MANETs) [30]. The use of source routing to enhance security has been previously suggested in [31].

4 SIMULATION RESULTS

To evaluate the algorithm proposed in Section 3 applied to network layer routing, we simulated the network in Figs. 2

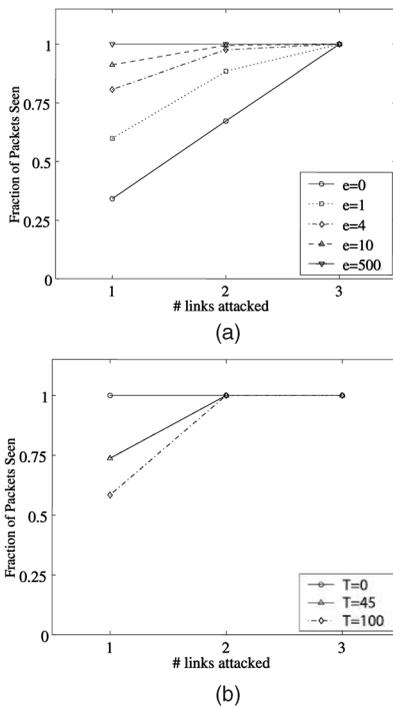


Fig. 5. Worst-case fraction of packets from CBR traffic seen by an attacker eavesdropping on one, two, or three links, using (a) offline and (b) online games.

and 4 using the ns-2 network simulator [32]. In all simulations presented, all links have a propagation delay of 20 ms and a bandwidth of 10 megabits per second (Mbps). Each queue implements drop-tail queuing discipline with maximum queue size set to 200 packets for the case of the constant bit rate (CBR) simulations and 100 packets for the Transmission Control Protocol (TCP) simulations. All packets are 1,000 bytes long. The simulation time for each trial was 100 seconds. Two types of experiments were performed. In the first type, only CBR traffic was used. In the second type, TCP-SACK traffic was used. In both types of experiments, the security parameter ϵ is fixed during a trial, but varies from trial to trial.

We were interested in determining the effect of stochastic routing on reliability, throughput, and packet transmission delay. To evaluate how secure a particular routing policy is, we determine the maximum fraction of packets that an attacker could see by eavesdropping on one, two, or three links. We assumed here that the attacker chooses the set of links that maximizes the fraction of packets seen (that is, the “worst”-case scenario). This is equivalent to determining how reliable the routing policy is by examining the worst-case fraction of packets lost when one, two, or three links fail. Fig. 5 shows the simulation results obtained for offline games with several values of the parameter ϵ and for online games with several values of the delay T_l (the same for every link). CBR traffic was used in all of these simulations. As expected, routing is most secure for offline games with $\epsilon = 0$ since, in this case, all paths are equally good and the packets will be spread the most across all paths. As ϵ increases, more packets are routed through the shorter paths and, therefore, an attacker eavesdropping on those will see a larger percentage of packets. In fact, for $\epsilon = 100$,

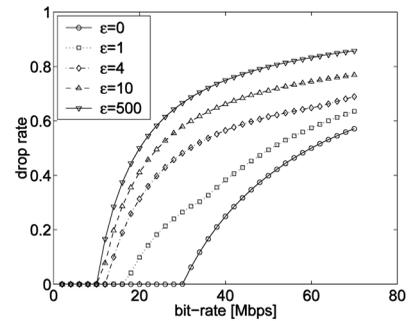


Fig. 6. Drop rate versus sending rate.

we essentially have minimum-hop routing and, by attacking a single link, it is possible to see every packet. Because, in the network tested, there are only three independent paths, when the attacker is allowed to eavesdrop on three links, she will be able to see every packet, regardless of which type of routing is used. Online games generally do not result in maximum flow and, in this particular topology, explore at most two of the three independent paths. As mentioned before, this can be explained by the fact that node 10 is heavily penalized by online games for having a single output interface.

For offline games with $\epsilon = 0$, we achieve maximum throughput (from a sender’s perspective) since we make use of all available paths (including the minimum cost one). This is supported by the data in Fig. 6, where we plot the drop rate as a function of the source’s sending rate. For $\epsilon = 100$, drops start occurring at sending rates around 10 Mbps, whereas, for $\epsilon = 0$, drops only become significant for sending rates higher than 30 Mbps. The price to pay for security comes in terms of the average latency per packet. This is because, to achieve high security, one needs to explore all the paths from source to destination, including those with high latency. Fig. 7 confirms that the delay does increase as the value of ϵ decreases.

Problems may arise when multipath routing is used in conjunction with standard TCP (Reno, New Reno, or SACK). Fig. 8a shows the average throughput of a long-lived TCP-SACK (circles) and a modified version of TCP called TCP-PR (hash marks), both operating under GTSR. For TCP-SACK, the throughput for $\epsilon = 100$ (essentially minimum-hop routing) is roughly six times larger than that

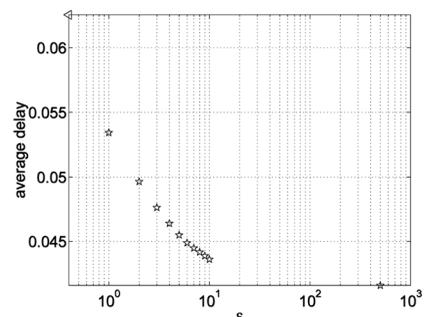


Fig. 7. Average packet transmission delay (CBR traffic). The triangle over the vertical axis corresponds to $\epsilon = 0$ (which could not be represented in log scale).

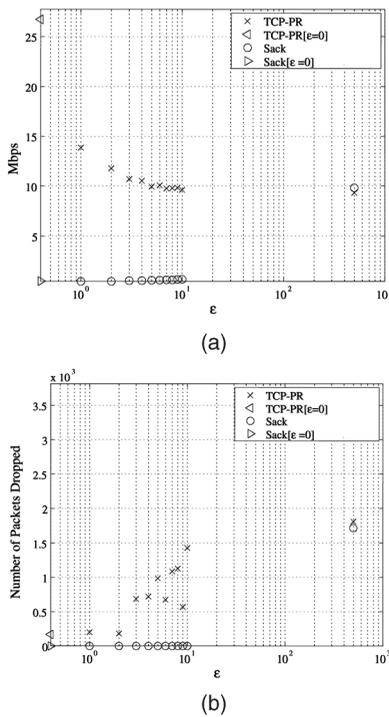


Fig. 8. (a) Transmission rate and (b) drop rate of TCP-SACK and TCP-PR for different security levels ϵ . For $\epsilon < 10$, the TCP-SACK flow experienced no drops. The point $\epsilon = 0$ is indicated with a triangle on top of the y -axis.

for any other value of ϵ . This is because packets sent through longer paths are often assumed dropped by TCP as they only arrived after several packets that were sent later, but traveled through shorter paths. Because fast retransmit is only triggered when three duplicate acknowledgments are received, TCP-SACK is able to handle some degree of out-of-order packets. However, in all of our simulations, this proved insufficient to handle packets traveling through paths with distinct propagation times. The fact that standard TCP performs poorly when used in conjunction with multipath routing has been observed in [13], [33].

TCP-PR was proposed in [34] and is one of several transport layers capable of coping with out-of-order packet delivery [35]. TCP-PR does not assume a packet is dropped when out-of-order sequence numbers are observed but only when a time-out occurs. Fig. 8a shows that, for $\epsilon = 0$, the throughput of TCP-PR is larger than all other configurations. That is, the maximum-security posture actually increases throughput as a by-product because it explores all alternative paths and, hence, uses all available bandwidth between the source and destination. Fig. 8b shows the drop rate for the different TCP implementations and different ϵ .

A complete discussion of TCP-PR is beyond the scope of this paper. However, it was shown in [34] that TCP-PR and TCP-SACK compete fairly over a network that does not implement multipath routing. Hence, it is possible to maintain a single transport layer that is compatible with today's network and is also suitable for networks that implement stochastic routing. One implication of this is that multipath routing could be incrementally deployed. For

example, packets could be marked as being willing to be routed stochastically. Then, flows that use TCP-PR and multipath routing would compete fairly with flows that request single path routing and use traditional implementations of TCP.

5 CONCLUSIONS

We proposed GTSR and demonstrated through simulations that it improves routing security. By proactively forcing packets to probabilistically take alternate paths, stochastic routing mitigates the effects of interception, eavesdropping, and improves fault tolerance. The routing policies proposed proved efficient in achieving statistical flooding at the expense of some increase in average packet delay. A beneficial side effect of these policies is an increase in throughput, as they explore multiple paths. The algorithms developed are applicable not only to the network layer, but also to application layer overlay networks and multipath transport protocols such as SCTP. However, when used at the network layer, a transport layer that is robust to out-of-order packets such as TCP-PR [34] must be employed.

We presented two alternative techniques to generate multipath routing tables. Routing based on offline games maximizes the spread of packets across the network, but requires link-state information and generally results in routing matrices. Alternatively, routing based on online games does not achieve the same degree of statistical flooding, but the routing computation can be distributed as in distance-vector algorithms.

A direction for future investigation is the development of scalable algorithms to compute next-hop probabilities. One promising approach is to make use of hierarchical decomposition of routing. Since this type of approach generally yields suboptimal routing policies, its impact on the GTSR performance requires further investigation. Preliminary results on the use of a hierarchical approach to compute routing tables based on offline games appeared recently in [36]. Alternatively, online games can be implemented under distance-vector routing and are scalable. For these algorithms, questions related to convergence need to be investigated.

APPENDIX A

DERIVATIONS FOR ONLINE GAMES

Proof of Theorem 1. The routing of a packet can be regarded as a controlled Markov chain whose state $q_t \in \mathcal{N}$ is a random variable denoting the node where the packet is before the hop $t \in \{0, 1, 2, \dots\}$. For a given routing policy $R := \{r_\ell : \ell \in \mathcal{L}\} \in \mathcal{R}_{\text{sto}}$ and attack policy $A := \{a_\ell : \ell \in \mathcal{L}\} \in \mathcal{R}_{\text{sto}}$, the transition probability function for the Markov chain is given by $P(q_{t+1} = k' | q_t = k) = r_{kk'}$, $\forall t \geq 0$. Without loss of generality, we assume that the final node n_{end} is an absorbing state, that is, that $P(q_{t+1} = n_{\text{end}} | q_t = n_{\text{end}}) = 1$, $\forall t \geq 0$. The cost to be optimized can be written as $E_{R,A}[\mathbf{T}^*] = E_{R,A} \left[\sum_{t=0}^{\infty} c(q_t) \right]$, where

$$c(k) = \sum_{\vec{k}, \vec{k}' \in \mathcal{L}[k]} r_{\vec{k}\vec{k}'} a_{\vec{k}\vec{k}'} m_{ijk},$$

$$m_{ijk} := \begin{cases} \tau_{\vec{k}\vec{k}'} & i \neq j, k \neq n_{\text{end}} \\ \tau_{\vec{k}\vec{k}'} + p_k T_{\vec{k}\vec{k}'} & i = j, k \neq n_{\text{end}} \\ 0 & k = n_{\text{end}}. \end{cases}$$

The two-person zero-sum game just defined falls in the class of stochastic shortest path games considered in [37]. In particular, it satisfies the SSP assumptions in [37]: [SSP1] There exists at least one proper policy for the minimizer, that is, a policy that will lead to a finite cost regardless of the policy chosen by the maximizer. This is true because we assume that there exists a sequence of links that connects node n_{src} to node n_{end} . [SSP2] For any policies for which there is a positive probability that the packet will not reach the destination node, the corresponding cost is infinite. This is true provided that $\tau_\ell > 0, \forall \ell \in \mathcal{L}$, because every hop that does not reach the final node n_{end} will contribute to the final cost with a positive marginal cost.

Consider now the function T defined by (5) and another function \tilde{T} defined similarly, except that the *max* and *min* in (5) appear in the reverse order. These functions satisfy the four regularity assumptions in [37]: [R1] The sets over which the controls $\{a_\ell : \ell \in \mathcal{L}[k]\}$, $\{r_\ell : \ell \in \mathcal{L}[k]\}$ take values are compact for every $k \in \mathcal{N}$. [R2] The maps from the controls $\{a_\ell : \ell \in \mathcal{L}[k]\}$, $\{r_\ell : \ell \in \mathcal{L}[k]\}$ to the transition probabilities $r_{\vec{k}\vec{k}'}$ and the costs $c(k)$ are continuous for every $k \in \mathcal{N}$. [R3-4] The minimum and maximum in the definitions of the functions T , and \tilde{T} are achieved and the two functions are actually equal [26, Minimax Theorem]. Statements 1 and 3 then follow from [37, Proposition 4.6] and statement 2 from [37, Proposition 4.7]. \square

APPENDIX B

DERIVATIONS FOR OFFLINE GAMES

Proof of Lemma 2. Denoting by $x_\ell(t)$ the probability that a packet is routed through link $\ell \in \mathcal{L}$ at time $t \in \{1, 2, \dots\}$ and has not yet been intercepted, we have that, for every $\ell \in \mathcal{L}$,

$$x_\ell(1) = \begin{cases} r_\ell & \ell \in \mathcal{L} \text{ exits from node } n_{\text{src}}, \\ 0 & \text{otherwise,} \end{cases} \quad (20)$$

$$x_\ell(t+1) = r_\ell \sum_{\ell' \in \mathcal{L}[\ell]} (1 - a_{\ell'} p_{\ell'}) x_{\ell'}(t) \quad \forall t > 1, \quad (21)$$

where the summation is taken over the set $\mathcal{L}[\ell]$ of links that enter the node from which link ℓ exits. Stacking all the $\{x_\ell : \ell \in \mathcal{L}\}$ into a column vector x , we can write (20)-(21) as

$$x(1) = \text{diag}[R] s, \quad (22)$$

$$x(t+1) = \text{diag}[R] C (I - \text{diag}[A]) x(t), \quad \forall t > 1, \quad (23)$$

where $\text{diag}[R]$ and $\text{diag}[A]$ denote diagonal matrices whose main diagonal contains the r_ℓ and the $a_{\ell} p_\ell$, respectively. From (22)-(23), we conclude that, for every $t \geq 1$,

$$x(t) = (\text{diag}[R] C (I - \text{diag}[A]))^{t-1} \text{diag}[R] s. \quad (24)$$

Since the probability that a packet is intercepted at time t is given by $\text{row}[A] x(t)$, we can write

$$E_{R,A}[\chi_\epsilon] = \sum_{t=1}^{\infty} (1 + \epsilon)^{t-1} \text{row}[A] x(t).$$

Moreover, because $x(t)$ evolves according to (24), we have that (10) holds with

$$x := \sum_{t=1}^{\infty} ((1 + \epsilon) \text{diag}[R] C (I - \text{diag}[A]))^{t-1} \text{diag}[R] s.$$

Since R is cycle free, the above series only has a finite number of nonzero terms (at most one per hop). Therefore, it must converge and be equal to

$$\sum_{t=1}^{\infty} ((1 + \epsilon) \text{diag}[R] C (I - \text{diag}[A]))^{t-1} \\ = (I - (1 + \epsilon) \text{diag}[R] C (I - \text{diag}[A]))^{-1}$$

[38]. Thus,

$$x = (I - (1 + \epsilon) \text{diag}[R] C (I - \text{diag}[A]))^{-1} \text{diag}[R] s,$$

which is equivalent to (11). \square

Proof of Lemma 3. The convexity of \mathcal{X} is trivial since \mathcal{X} is given by the intersection of the (convex) positive orthant with the (also convex) affine space corresponding to the solution of the linear equation (12). To prove Condition 1 of Lemma 3, we show that, for a given $R \in \mathcal{R}_{\text{no-cycle}}$, we can define

$$x := \sum_{t=1}^{\infty} ((1 + \epsilon) \text{diag}[R] A)^{t-1} \text{diag}[R] c. \quad (25)$$

First, note that, since $R \in \mathcal{R}_{\text{no-cycle}}$, the series converges because it only has a finite number of nonzero terms and, therefore,

$$\sum_{t=1}^{\infty} ((1 + \epsilon) \text{diag}[R] A)^{t-1} = (I - (1 + \epsilon) \text{diag}[R] A)^{-1}$$

[38]. From this and the definition of x , we immediately conclude that (13) holds. To verify that the vector x defined by (25) belongs to \mathcal{X} , note that every entry of x is nonnegative because it is the sum of nonnegative numbers and also that, left multiplying (13) by C_{out} , we obtain (12) because of the following two equalities that are straightforward to verify:

$$C_{\text{in}} = C_{\text{out}} \text{diag}[R] C, \quad s_{\text{src}} = C_{\text{out}} \text{diag}[R] s. \quad (26)$$

Note also that, because the number of nonzero terms in the series is always smaller than n and the R are bounded, one can construct a bound on the vectors x defined by (25), which is independent of $R \in \mathcal{R}_{\text{no-cycle}}$.

To prove Condition 2 of Lemma 3, note that the division in (14) guarantees that the normalization condition (1) holds and, therefore, that $R \in \mathcal{R}_{\text{sto}}$. Moreover, this definition guarantees that if x is cycle free, then R is also cycle free. To finish the proof, it remains to

show that (13) holds. To this effect, note that, from the definition of R , for every $\ell \in \mathcal{L}$, we have that

$$x_\ell = r_\ell \sum_{\ell' \in \mathcal{L}[k]} x_{\ell'} = r_\ell C_{\text{out}}[k]x,$$

where $\mathcal{L}[k]$ denotes the set of links that exit from node k from which ℓ exits and $C_{\text{out}}[k]$ denotes the row of C_{out} corresponding to node k . However, since $x \in \mathcal{X}$, we then have

$$x_\ell = (1 + \epsilon)r_\ell C_{\text{in}}[k]x + r_\ell \delta_{k, n_{\text{src}}}, \quad (27)$$

where $C_{\text{in}}[k]$ denotes the rows of C_{in} corresponding to the node k and $\delta_{k, n_{\text{src}}}$ is equal to one if $k = n_{\text{src}}$ and zero otherwise. This finishes the proof because (13) is a vector version of (27). \square

Proof of Theorem 4. First, we can conclude from von Neumann's Theorem [26] that the flow game (15) always has a saddle point $(x^*, A^*) \in \mathcal{X} \times \mathcal{A}$ because the cost of the game is bilinear on the policies x and A , which take values in compact convex sets. Moreover, since removing cycles from a policy never increases the cost, x^* can be assumed cycle free. The set \mathcal{X} is actually not bounded and therefore not compact. However, in view of Condition 1 in Lemma 3, we can restrict our attention to a bounded subset of \mathcal{X} .

Then, let $(x^*, A^*) \in \mathcal{X} \times \mathcal{A}$ be the saddle point whose existence was just proved. We show next that $(R^*, A^*) \in \mathcal{R}_{\text{no-cycle}} \times \mathcal{A}$, with R^* is constructed from x^* using (14), is a saddle point for the routing game (9). To this effect, we need to show that

$$E_{R^*, A}[\chi_\epsilon] \leq E_{R^*, A^*}[\chi_\epsilon] \leq E_{R, A^*}[\chi_\epsilon] \quad (28)$$

for every $R \in \mathcal{R}_{\text{no-cycle}}$ and every $A \in \mathcal{A}$. Because of (10), (28) is actually equivalent to

$$\text{row}[A]x^* \leq \text{row}[A^*]x^* \leq \text{row}[A^*]x, \quad (29)$$

where x is the unique solution to (11). To verify that x belongs to \mathcal{X} note that, left multiplying (11) by C_{out} , we obtain (12) because of (26). This means that (29) must hold because (x^*, A^*) is a saddle point of the flow game (15). \square

ACKNOWLEDGMENTS

The authors thank the reviewers for several comments that significantly improved this paper. This material is based upon work partially supported by the US National Science Foundation (NSF) under Grant No. ANI-0322476 and by the Institute for Collaborative Biotechnologies through grant DAAD19-03-D-0004 from the US Army Research Office. Professor Obraczka has also been partially supported by the US Army Research Office under grant W911NF-05-1-0246.

REFERENCES

- [1] S. Bengio, G. Brassard, Y. Desmedt, C. Goutier, and J. Quisquater, "Secure Implementation of Identification Systems," *J. Cryptology*, vol. 4, pp. 175-183, 1991.
- [2] D.G. Andersen, H. Balakrishnan, M.F. Kaashoek, and R. Morris, "Resilient Overlay Networks," *Proc. 18th ACM Symp. Operating Systems Principles (SOSP '01)*, 2001.
- [3] L. Ong and J. Yoakum, *An Introduction to the Stream Control Transmission Protocol (SCTP)*, RFC 3286, 2002.
- [4] R. Blazek, H. Kim, B. Rozovskii, and A. Tartakovsky, "A Novel Approach to Detection of "Denial-of-Service" Attacks via Adaptive Sequential and Batch-Sequential Change Point Detection Methods," *Proc. Systems, Man, and Cybernetics Information Assurance and Security Workshop*, June 2000.
- [5] W. Xu and J. Rexford, "MIRO: Multi-Path Interdomain Routing," *Proc. ACM SIGCOMM '06*, Sept. 2006.
- [6] S. Bohacek, J.P. Hespanha, K. Obraczka, J. Lee, and C. Lim, "Enhancing Security via Stochastic Routing," *Proc. 11th IEEE Int'l Conf. Computer Comm. and Networks*, May 2002.
- [7] A. Emmett, "VPNs," *Am. Networks*, May 1998.
- [8] S. Kent, *Security Architecture for the Internet Protocol*, RFC 2401, 1998.
- [9] P.F. Syverson, M.G. Reed, and D.M. Goldschlag, "Onion Routing Access Configurations," *Proc. DARPA Information Survivability Conf. and Exposition (DISCEX '00)*, vol. I, pp. 34-40, Jan. 2000.
- [10] S. Kent, C. Lynn, J. Mikkelsen, and K. Seo, "Secure Border Gateway Protocol (s-BGP)—Real World Performance and Deployment Issues," *Proc. Network and Distributed System Security Symp. (NDSS '00)*, 2000.
- [11] M.K. Reiter and A.D. Rubin, "Crowds: Anonymity for Web Transactions," *ACM Trans. Information and System Security*, vol. 1, pp. 66-92, 1998.
- [12] C. Hopps, *Analysis of an Equal-Cost Multi-Path Algorithm*, RFC 2992, Nov. 2000.
- [13] C. Villamizar, "OSPF Optimized Multipath (OSPF-OMP)," Internet Draft (draft-ietf-ospf-omp-03), Internet Eng. Task Force, June 1999.
- [14] E. Altman, T. Boulogne, R. El-Azouzi, T. Jiménez, and L. Wynter, "A Survey on Networking Games in Telecommunications," *Computers and Operations Research*, vol. 33, no. 2, pp. 286-311, 2006.
- [15] A. Economides and J.A. Silvester, "Multi-Objective Routing in Integrated Services Networks: A Game Theory Approach," *Proc. IEEE INFOCOM*, 1991.
- [16] K. Yamaoka and Y. Sakai, "A Packet Routing Based on Game Theory," *Trans. Inst. of Electronics, Information, and Comm. Eng., B-I*, pp. 73-79, 1996.
- [17] R. Kannan, S. Sarangi, and S.S. Iyengar, "Sensor-Centric Energy-Constrained Reliable Query Routing for Wireless Sensor Networks," *J. Parallel and Distributed Computing*, vol. 64, no. 7, pp. 839-852, 2004.
- [18] A. Orda, R. Rom, and N. Shimkin, "Competitive Routing in Multiuser Communication Networks," *IEEE/ACM Trans. Networking*, vol. 1, no. 5, pp. 510-521, 1993.
- [19] E. Koutsoupias and C. Papadimitriou, "Worst-Case Equilibria," *Proc. 16th Ann. Symp. Theoretical Aspects of Computer Science*, pp. 404-413, 1999.
- [20] T. Roughgarden and E. Tardos, "How Bad Is Selfish Routing," *J. ACM*, vol. 49, no. 2, pp. 236-259, 2002.
- [21] R. La and V. Anantharam, "Optimal Routing Control: Repeated Game Approach," *IEEE Trans. Automatic Control*, Mar. 2002.
- [22] L. Qiu, Y.R. Yang, Y. Zhang, and S. Shenker, "On Selfish Routing in Internet-Like Environments," *Proc. ACM SIGCOMM '03*, pp. 151-162, 2003.
- [23] J. Feigenbaum, C. Papadimitriou, R. Sami, and S. Shenker, "A BGP-Based Mechanism for Lowest-Cost Routing," *Proc. 21st Ann. Symp. Principles of Distributed Computing (PODC '02)*, pp. 173-182, 2002.
- [24] M. Afergan, "Using Repeated Games to Design Incentive-Based Routing Systems," *Proc. IEEE INFOCOM*, 2006.
- [25] A. Blanc, Y.-K. Liu, and A. Vahdal, "Designing Incentives for Peer-to-Peer Routing," *Proc. IEEE INFOCOM*, 2005.
- [26] T. Başar and G.J. Olsder, *Dynamic Noncooperative Game Theory*. Academic Press, 1995.
- [27] D.P. Bertsekas, *Network Optimization: Continuous and Discrete Models*. Athena Scientific, 1998.
- [28] A.V. Goldberg and R.E. Tarjan, "A New Approach to the Maximum-Flow Problem," *J. ACM*, vol. 35, pp. 921-940, 1988.
- [29] D.P. Bertsekas, "An Auction Algorithm for the Max-Flow Problem," *J. Optimization Theory and Applications*, vol. 87, pp. 69-101, 1995.
- [30] D.B. Johnson, D.A. Maltz, and J. Broch, "DSR: The Dynamic Source Routing Protocol for Multi-Hop Wireless Ad Hoc Networks," *Ad Hoc Networking*, C.E. Perkins, ed., pp. 139-172, Addison-Wesley, 2001.

- [31] R. Perlman, "Network Layer Protocols with Byzantine Robustness," PhD dissertation, Massachusetts Inst. of Technology, 1988.
- [32] "The VINT Project, a Collaboration between UC Berkeley, LBL, USC/ISI, and Xerox PARC," *The ns Manual (formerly ns Notes and Documentation)*, Oct. 2000.
- [33] D. Thaler and C. Hopps, *Multipath Issues in Unicast and Multicast Next-Hop Selection*, RFC 2991, Nov. 2000.
- [34] S. Bohacek, J.P. Hespanha, J. Lee, C. Lim, and K. Obraczka, "TCP-PR: TCP for Persistent Packet Reordering," *Proc. IEEE 23rd Int'l Conf. Distributed Computing Systems*, pp. 222-231, May 2003.
- [35] E. Blanton and M. Allman, "On Making TCP More Robust to Packet Reordering," *ACM Computer Comm. Rev.*, vol. 32, 2002.
- [36] C. Lim, S. Bohacek, J.P. Hespanha, and K. Obraczka, "Hierarchical Max-Flow Routing," *Proc. IEEE Global Telecomm. Conf. (GLOBECOM '05)*, Nov. 2005.
- [37] S.D. Patek and D.P. Bertsekas, "Stochastic Shortest Path Games," *SIAM J. Control and Optimization*, vol. 37, no. 3, pp. 804-824, 1999.
- [38] R.A. Horn and C.R. Johnson, *Matrix Analysis*. Cambridge Univ. Press, 1993.



Stephan Bohacek received the BS degree in electrical engineering from the University of California, Berkeley, in 1989 and the PhD degree in electrical engineering from the University of Southern California in 1999. He is currently an assistant professor in the Department of Electrical and Computer Engineering at the University of Delaware. His research interests include design, analysis, and control of data networks. His current interests include congestion

control and routing for wireless and wireline networks, modeling mobile wireless networks, and cross-layer design for wireless networks. He is a member of the IEEE.



João P. Hespanha received the PhD degrees in electrical engineering and applied science from Yale University in 1998. He is currently a professor of electrical and computer engineering at the University of California, Santa Barbara. His current research interests include hybrid and switched systems, the modeling and control of communication networks, distributed control over communication networks, stochastic modeling in biology, and game theory. More information

about his research can be found at <http://www.ece.ucsb.edu/~hespanha>. He is a senior member of the IEEE.



Junsoo Lee received the BS and MS degrees in computer science from Seoul National University, Seoul, and the PhD degree in computer science from the University of Southern California, Los Angeles, in 2004. He is currently an assistant professor of computer science at Sookmyung Women's University of Seoul. Before joining Sookmyung Women's University, he was a postdoctoral scholar at the University of California, Santa Barbara, and the University of California, Santa Cruz. He also worked for Samsung Electronics as a senior researcher at the Network and Computer Division before pursuing the PhD degree. He is a member of the IEEE.



Chansook Lim received the PhD degree in computer science from the University of Southern California in 2006. She is currently a postdoctoral scholar at the University of Delaware. Her research interests include routing for wired and wireless networks, network measurement, and network dependability. She is a member of the IEEE.



Katia Obraczka received the PhD degree in computer science from the University of Southern California (USC) in 1994. She is currently an associate professor of computer engineering at the University of California, Santa Cruz (UCSC). Before joining UCSC, she held a research scientist position at USC's Information Sciences Institute and a research faculty appointment at USC's Computer Science Department. Her research interests include computer networks,

more specifically, network protocol design and evaluation in wireline, as well as wireless (in particular, multihop ad hoc) networks, distributed systems, and Internet information systems. More information on her research can be found at <http://www.soe.ucsc.edu/~katia>. She is a member of the IEEE.

► For more information on this or any other computing topic, please visit our Digital Library at www.computer.org/publications/dlib.