Using Link Cuts to Attack Internet Routing

Steven M. Bellovin Emden R. Gansner
{smb,erg}@research.att.com
AT&T Labs Research

Abstract

Attacks on the routing system, with the goal of diverting traffic past an enemy-controlled point for purposes of eavesdropping or connection-hijacking, have long been known. In principle, at least, these attacks can be countered by use of appropriate authentication techniques. We demonstrate a new attack, based on link-cutting, that cannot be countered in this fashion. Armed with a topology map and a list of already-compromised links and routers, an attacker can calculate which links to disable, in order to force selected traffic to pass the compromised elements. The calculations necessary to launch this attack are quite efficient; in our implementation, most runs took less than half a second, on databases of several hundred nodes. We also suggest a number of work-arounds, including one based on using intrusion detection systems to modify routing metrics.

1 Introduction

Attacks on the routing system, with the goal of diverting traffic past an enemy-controlled point for purposes of eavesdropping or connection-hijacking [1], have long been known [2, 3]. In such attacks, an enemy advertises a false route. That is, some node claims to have a better (lower-cost) route to a given destination. That will induce other nodes to send traffic for that destination to (or towards) the subverted node, where it can be captured by the enemy.

The false advertisement can be done in several ways. The simplest mechanism is for the enemy to gain control of a router and configure it to emit the false advertisements. Note that by "gain control", we do not necessarily mean "subvert"; gaining control might be as simple as buying a certain grade of service from an ISP. Alternatively, an attacker with suitable access to a network link can simply inject false packets onto the wire.

At least in principle, packet injection is relatively easy to defend against. OSPF [4] can use Keyed MD5 [5] to guard against this. Similarly, many ISPs use Keyed MD5 to protect BGP transmissions [6].

Defending against false advertisements by legitimate speakers is harder. [7] describes how to use digitally signed statements in OSPF; the routers' certificates include the authorized address ranges. [8, 9] extends that concept to BGP [10, 11].

We present a new attack, based on cutting links and forcing the new path to traverse enemy-controlled links or nodes. Briefly, given knowledge of the network topology, an enemy can easily calculate a set of links to cut. The usual routing mechanisms will detect the failures and route around them, in such a way that selected traffic will flow through the compromised elements.

Section 2 explains how routing works on the Internet. Section 3 shows how to implement the attack, given the nature of Internet routing.

The graph-theoretic algorithm is presented in Section 4. We implemented the algorithm and applied it to some experimental topologies; this is discussed in Section 5. We discuss countermeasures in Section 6 before outlining future work.

1.1 Practical Link-Cutting

There are a number of ways to cut links. The most obvious is physical: sever the wire or fiber optic cable. While not a common hacking technique, it might be employed by more serious enemies. Besides, no network operator will be surprised to hear that a rampaging backhoe has severed a fiber.

Other problems have existed in the past that could crash links. The original specification for PPP over SONET rings [12] did not specify a "scrambler"; this would have allowed an attacker to crash a link by sending packets with certain bit patterns. It took five years for the standard to be updated [13].

Finally, it may be possible to make a link appear dead to routing protocols by flooding it via a distributed denial of service attack [14]. This technique is trickier, since once the link appears to be down, all traffic—including the attack traffic—will be routed around it, thus causing it to appear to resurrect itself.

A variant of this attack involves disabling routers instead of links. This, too, is feasible. Over the years, there have been a number of bugs that can crash various routers (i.e., CERT Advisory CA-2002-03). Certain sorts of packets directed to the router itself can drive up its CPU load and effectively disable it; this form of attack has already been seen on the Internet.

1.2 Terminology

This paper speaks of both Internet routing and graph theory. In each section, we try to use the appropriate language. Thus, when we are discussing routing, we speak of routers (also called nodes) and links (also called wires). When we are discussing graph-theoretic concepts, the same entities are called vertices and edges.

2 Routing on the Internet

Most of this paper deals with routing as a simple graph-theoretic problem. This does not match the reality of the Internet. In this section, we (briefly) sketch Internet routing and explain how to map it to the simple graphs we are considering.

The Internet is too big for a single routing algorithm. Consequently, a hierarchical approach is used. Routing tables within an *autonomous system* (AS)—roughly speaking, an ISP or major customer—are determined by an *interior gateway protocol* (IGP); routing between ASs is determined by an *exterior gateway protocol* (EGP). This hierarchical routing has two effects on our attack. First, by constraining the size of the ordinary routing calculation, it limits limits the work needed to perform the attack. Second, the different routing protocols that are in fact used have different implications for the practicality of the attack.

2.1 Exterior Routing: BGP

Exterior routing—that is, routing between autonomous systems—is calculated via the *Border Gateway Protocol* (BGP) [10]. BGP is a path vector protocol. Each node applies local policy to the paths received from its peers in order to calculate the best path to destinations; additionally, it appends its AS number to the path and passes that path to its peers.

There are two major implications of this. First, individual nodes (or eavesdroppers on the links leading to these nodes) do not have a complete picture of the inter-AS topology. Maps can be drawn, but as is pointed out in [15] and [16], the task is difficult. Still, sufficient public data exists that it may be possible to construct a good-enough approximation. The best publicly-available data is described in [17]. Second, routing policy is very important. We sketch how to deal with it in Section 2.4, but in general it is a hard problem.

After applying policy constraints, BGP uses hop count as its metric. In order to help achieve routing policy goals—individual ASs can and do filter BGP data in order to achieve certain goals—it is common to pad the AS path with repetitions of the local AS, thus lengthening the apparent cost as seen from the outside. See the discussion of routing policy below.

2.2 Interior Routing: OSPF and IS-IS

Interior routing within an AS is commonly calculated by OSPF (*Open Shortest Path First*) [4] or IS-IS (Intermediate System-Intermediate System) [18]. (Some small sites use RIP (Routing Information Protocol) [19]. RIP is a distance vector protocol; for our purposes, it has similar characteristics to BGP, in that sites do not receive a complete topology map. We will not be discuss it further here. Other small sites use E-IGRP, a proprietary distance vector protocol.) OSPF and IS-IS are link-state protocols. That is,

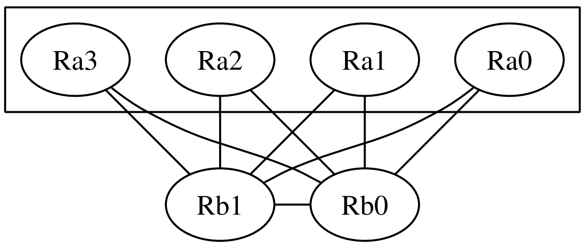


Figure 1: Design of a typical Point of Presence (POP).

each node advertises the existence and status of each of its links. Furthermore, each node readvertises the link information it has learned from its neighbors. Because of this readvertisement, all nodes learn the entire topology of the AS. They then calculate the path to each destination using Dijkstra's shortest path algorithm.

Even a single AS can grow too large for this process to be feasible. Accordingly, OSPF and IS-IS permit a two-level hierarchy. Routers within a area know only the topology within their area; only the interarea routers know the topology of more than one area. One distinguished area, known as Area 0, is the core; the other areas must connect to it. Intra-area paths never traverse other areas; this is sometimes used to constrain routing within POPs (see below).

Metrics for the routing calculation are arbitrary; however, they must be consistent within an AS.

2.3 POP Design

As described in [17], ISPs are typically organized as POPs (*Points of Presence*) connected by a backbone. A typical POP layout is shown in Figure 1. The routers labeled Rb0 and Rb1 are *backbone routers*; they are connected to other POPs. Backbone routers within a POP are connected in a dense mesh by very high speed links.

Ra0, Ra1, etc., are *access routers*. A typical POP will have many of these; they're used to connect to customer sites. Links to other ISPs are often through specialized access routers; these are sometimes known as *gateway routers*. Access routers are generally connected to two or more backbone routers, usually by links that are slower than those used for interconnections between the backbone routers.

In a large POP (Figure 2), there are likely to be too many access routers to permit direct connection to the backbone routers. In such cases, an intermediate layer of router is used. Access routers are connected to two or more intermediate routers; intermediate routers are connected to two or more backbone routers.

Because of the link speeds used, it is undesirable to have traffic between two backbone routers within a POP traverse intermediate or access routers. This is prevented artificially, either by setting the cost metrics very high on the links to the access routers, or by putting the access routers within a POP in a separate OSPF area.

In some POPs, a LAN—a single, multi-access network fabric—is used for interconnection. We model LANs with a hidden pseudo-node with links to all of the routers on the LAN.

2.4 Routing Policy

For a variety of reasons, ISPs do not use simple cost metrics, especially for inter-AS routes. Rather, various policies constrain routing decisions. Sometimes, these policies are driven by business relationships, such as agreemnts in which one ISP has paid another to carry some types of traffic, but not others. Other policies are driven by load balancing, varying link speeds, bandwidth pricing, and more.

There have been a few attempts to discover routing polices; see, for example, [20, 21, 22, 23]. Policies are often considered confidential, and are not easy to determine from the outside. [22] shows that determining AS policies is NP-complete; however, they also show that there are efficient approximation algorithms that handle most of the actual Internet. We note, though, that in the security arena, enemies may not restrict themselves to technical means of gathering information. Non-technical mechanisms—i.e., spying—is far from unknown.

One simple case is the so-called "stub AS": one that never carries any transit traffic. A multi-homed customer site falls into this category. We assign infinitely high weights to all links entering such an AS,

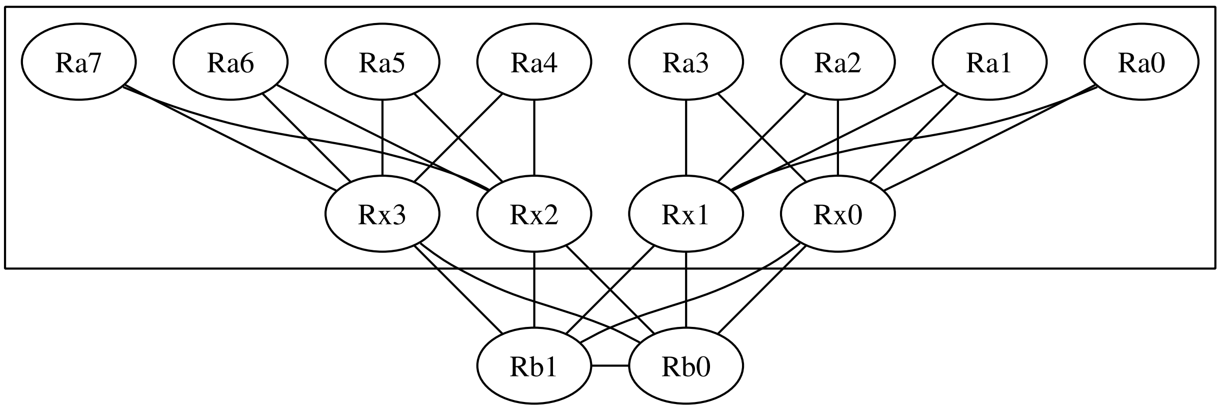


Figure 2: Design of a large POP. The routers labeled Rx*i* avoid having too much fanout from the backbone routers.

so that no route will ever be computed that traverses such an AS.

Moving beyond that is complex. If the attacker has perfect knowledge of the relevant policies, there is a simple heuristic that will suffice: delete all edges from the graph that are not available for traffic between the targeted pair of nodes. That is, if the traffic of interest is between hosts A and D, any links over which that traffic cannot flow effectively do not exist. Deleting them will also simplify the graph.

An example can make this clearer. In Figure 3a, X, Y, and Z are major (i.e., "Tier 1") ISPs that peer with each other. By policy, however, the links X–Y, Y–Z, and Z–X are only usable when the source or destination is a customer of the receiving ISP. Thus, traffic from B to A will only flow via B–Y–X–A; it will *never* follow the path B–Y–Z–X–A, even if link Y–X is down. If we are trying to monitor traffic between those two nodes, link Y–Z effectively does not exist, and can be deleted from the graph (Figure 3b).

On the other hand, if B is multihomed to Y and Z (Figure 3c), its traffic will be allowed to flow over Y–Z. B is thus protected against failures of not just B–Y (perhaps Y is its primary ISP), but also against other link failures. In such cases, we cannot delete these links.

2.5 Complex Layer 2 Topologies

As noted in [17], some ISPs have complex inter-POP topologies. These links are generally virtual circuits implemented via ATM, MPLS, or all-optical switching.

It is tempting to try to add the layer 2 nodes to the topology. However, not all links that physically exist are used in all possible ways. In Figure 4, for example, there is no direct virtual circuit from R1 to R4; thus, IP traffic between the two routers would go via either R1–R2–R4 or R1–R3–R4.

Even in simpler topologies, what appear to be independent links may, in fact, share link provider facilities. The problem of geographic link diversity is well-known to organizations that desire maximum availability; they have not found it easy to solve. Accordingly, for purposes of this paper we are ignoring the actual implementation of such links, and are assuming that they are, in fact, independent physical wires.

3 Modeling Attacks

Given these constraints, we need to demonstrate how an attacker can actually implement the attack described here. Put another way, if an enemy controls certain links or routers, what is the precise procedure to follow to launch the attack?

The first step is to force the traffic through an AS

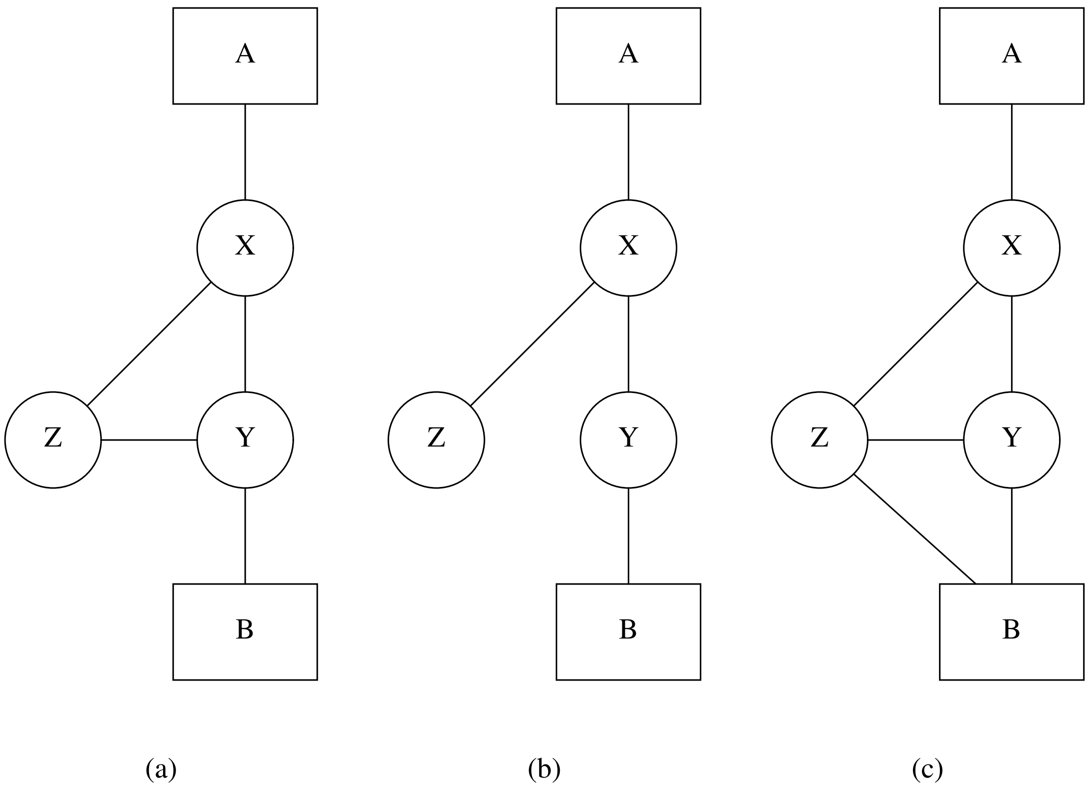


Figure 3: The effects of routing policy on single- and multi-homed hosts.

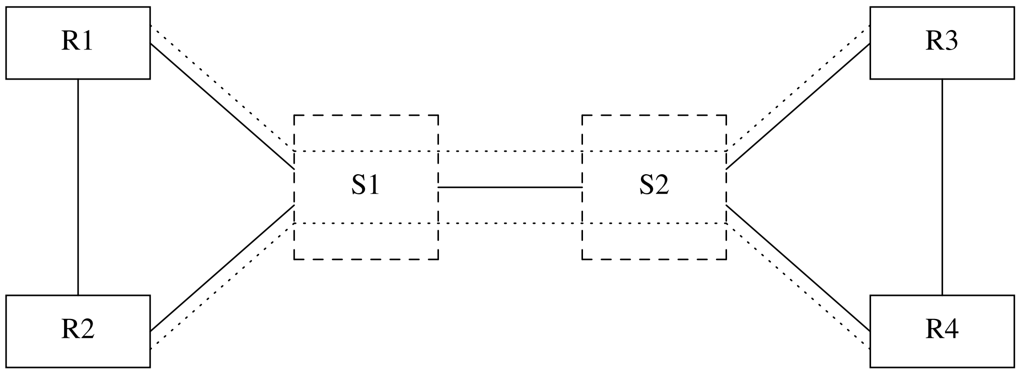


Figure 4: A layer 3 topology implemented via layer 2 virtual circuits, shown as dotted lines. The physical links are solid lines. There is no IP-layer link from R1 to R4, even though the physical path exists.

containing a compromised link or router. We run the attack algorithm on the AS graph, treating each such AS as a compromised vertex that we wish the traffic to traverse. If two ASs peer at more than one point, delete all but one link, but scale the difficulty factor for the link to reflect the fact of multiple connections; at the BGP level, one cannot control which such link is used. The output of this step will be a set of inter-AS links that must be cut.

Assuming that those links have been cut, we then rerun the BGP algorithm to see which compromised ASs will be traversed. Recall that our goal is not to traverse every compromised AS; any one will suffice. For each AS that will be traversed, we run the attack algorithm on its topology. Upstream and downstream ASs as represented as pseudo-nodes. Multiple links to these ASs are included in this graph, as they may need to be cut to force traffic over the proper path.

At this point, the attacker has the necessary information. Cutting the links identified in the first step will force the traffic through one or more compromised ASs. For one or more of them, cut the links within the AS, to force traffic over its compromised links.

On the Internet itself, major providers are connected by multiple links. Severing all of the links between a pair of such providers is difficult. Although we take into account the difficulty of severing a link, our current approach only makes limited use of this information. We hope to extend our work to make better use of this information.

4 Link-Cutting Attacks

We now present an algorithm for deciding what links to cut. Let $\mathcal{A}$ be the topology graph, with n vertices and m edges, for some routing area. Assume that the enemy controls some set of vertices $\mathcal{E}$. Obviously, $|\mathcal{E}| \leq n$; most likely, $|\mathcal{E}| \ll n$, and quite possibly $|\mathcal{E}|$ is very small, perhaps one or two. We also assume that each edge in $\mathcal{A}$ has a non-negative integral cost, which represents the cost or difficulty in cutting that edge. The goal of the enemy is, given two vertices s and t, to ensure that the shortest path between them transits at least one member of $\mathcal{E}$, preferably by cutting edges whose total cost is as little as possible.

The algorithm is as follows:

1. First, check the shortest path from s to t in $\mathcal{A}$. If it contains any component of $\mathcal{E}$, we are done.
2. For each vertex v in $\mathcal{E}$, calculate a shortest simple (i.e., no repeated vertices) path from s and t containing v.
3. If the set of paths is empty, the problem has no solution. Otherwise, from these paths, pick a shortest one, say $\mathcal{P}$, with length l. Assign each edge on this path an infinite cost.
4. Calculate a minimal cost s-t cut[1] of $\mathcal{A}-\mathcal{E}$. Note that s and t must be connected in $\mathcal{A} - \mathcal{E}$; otherwise, all paths from s to t in $\mathcal{A}$ would pass through $\mathcal{E}$ and we would have stopped at the first step above.
5. For each member e of the cut, calculate the shortest path from s to t containing e in $\mathcal{A} - \mathcal{E}$. If this path has length less than or equal l, remove the edge.

When the algorithm terminates, we end up with a graph $\mathcal{A}'$, which is $\mathcal{A}$ with the edges selected in Step 5 deleted. Since every path from s to t in $\mathcal{A}-\mathcal{E}$ must contain an edge in the cut, every path between s and t in $\mathcal{A}' - \mathcal{E}$ will have length greater than l. Any shorter path in $\mathcal{A}'$ must use an element of $\mathcal{E}$. In addition, the path $\mathcal{P}$ of length l still exists in $\mathcal{A}'$ since each of its edges had infinite cost and would not be selected in the cut. Indeed, the minimum cut can have infinite cost if and only if there is path in $\mathcal{A} - \mathcal{E}$ from s to t using only edges from $\mathcal{P}$, an immediate consequence of the max flow-min cut theorem [24]. Since $\mathcal{P}$ is simple, such a path cannot exist in $\mathcal{A}' - \mathcal{E}$. We thus find that any shortest path connecting s and t in $\mathcal{A}'$ must traverse $\mathcal{E}$, as desired.

Our method finds some set of edges whose deletion forces a shortest path through $\mathcal{E}$, and the heuristics provide some local minimization of the cost. Note that, for expediency, we are only attempting to solve a restricted version of the full problem. We are fixing a path which uses $\mathcal{E}$ and looking for a minimum cost set of edges whose removal makes the

[1] An *s-t cut* is a set of edges that, if deleted, creates a disconnected graph with s and t in separate components. The cost of the cut is the sum of the cost of the edges in the cut.

given path the shortest path. The general problem, though, allows paths which need not be shortest in the original graph. Figure 5 presents a simple example where this approach fails to find a minimum cut, even when $\mathcal{E}$ contains a single vertex. The dashed

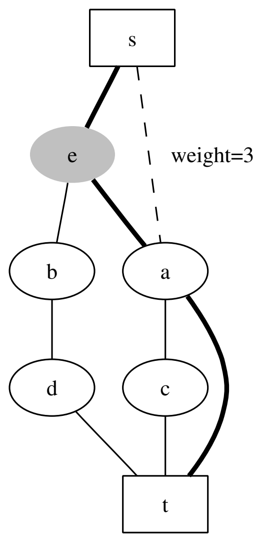


Figure 5: Missing the minimum cut.

edge has cost 3; the rest have cost 1. The shortest path containing e is shown in bold, having length 3. Our algorithm will then cut the single edge of cost 3. A better choice would be to cut edges $a - c$ and $a - t$. This forces communication along the path $s - e - b - d - t$. This is a longer path, but the total cost is 2.

To simplify the presentation, we limited $\mathcal{E}$ to vertices, but $\mathcal{E}$ might contain edges as well. The algorithm and its implementation are largely unmodified. Alternatively, we can just convert the problem to one in which $\mathcal{E}$ only contains vertices. This can be done by deleting each edge e in $\mathcal{E}$ from $\mathcal{A}$ and adding a new enemy-controlled vertex n_e to $\mathcal{A}$, with n_e connected by a single edge to each endpoint of e. The cost of the edge e is apportioned to the two new edges.

Note that the cut edges chosen in step 5 need not be independent of each other, in that deleting one may also increase the shortest path length involving others. Therefore, it is possible we may further reduce the number and total cost of deleted edges by removing each edge chosen before checking another edge in the cut.

Finding a shortest simple path in step 2 assumes there is some simple path from s to t containing the vertex v. This can be determined by looking at the block-cutpoint tree[2] of the graph $\mathcal{A}$. Let $\bar{s}$, $\bar{t}$ and $\bar{v}$ be the 3 blocks containing s, t and v, respectively. Then the desired simple path exists in $\mathcal{A}$ if and only if the path from $\bar{s}$ to $\bar{v}$ to $\bar{t}$ is simple in the block-cutpoint tree. It should be noted that this criterion is necessary and sufficient not just for a solution to our restricted version, but for the general problem. If none of the enemy vertices lies on a simple path from s to t, there is no collection of edges which can be removed to make the shortest path from s to t go through $\mathcal{E}$.

Calculating shortest paths and the block-cutpoint tree are linear in the number of edges in $\mathcal{A}$. However, we are unsure what the complexity is of computing the shortest simple paths of Step 2, given we know they exist. Our implementation relies on various heuristics, such as merging the shortest paths from s to v and from v to t; if v has degree 2, removing an adjacent edge and merging the shortest paths from s and t to the edge's endpoints with the edge; and variations on removing the shortest path $\mathcal{P}$ from s to v, and seeing if there is still a path from v to t, which can then be combined with $\mathcal{P}$ to form a simple path. Although elementary, our experience is that they perform satisfactorily in practice (cd. Section 5.

As shown by Figure 5, the approach presented here can miss significantly better solutions in terms of cost, largely because we restrict the problem before looking for a low-cost cut. This suggests we start with a minimum cut in $\mathcal{A} - \mathcal{E}$, using the original edges costs. If the entire cut is removed, there are no paths connecting s and t. We could then consider ways of removing only subsets of the cut, in hopes of finding a shortest path using $\mathcal{E}$. This would find the best solution in the example of the figure.

[2]A *cutpoint* is a node whose removal separates the graph into multiple non-trivial components. The *block-cutpoint tree* of a graph $\mathcal{G}$ is the tree whose nodes are the cutpoints c_i and biconnected components, or blocks, B_j of $\mathcal{G}$, with edges (c_i, B_j) if cutpoint c_i belongs to block B_j.

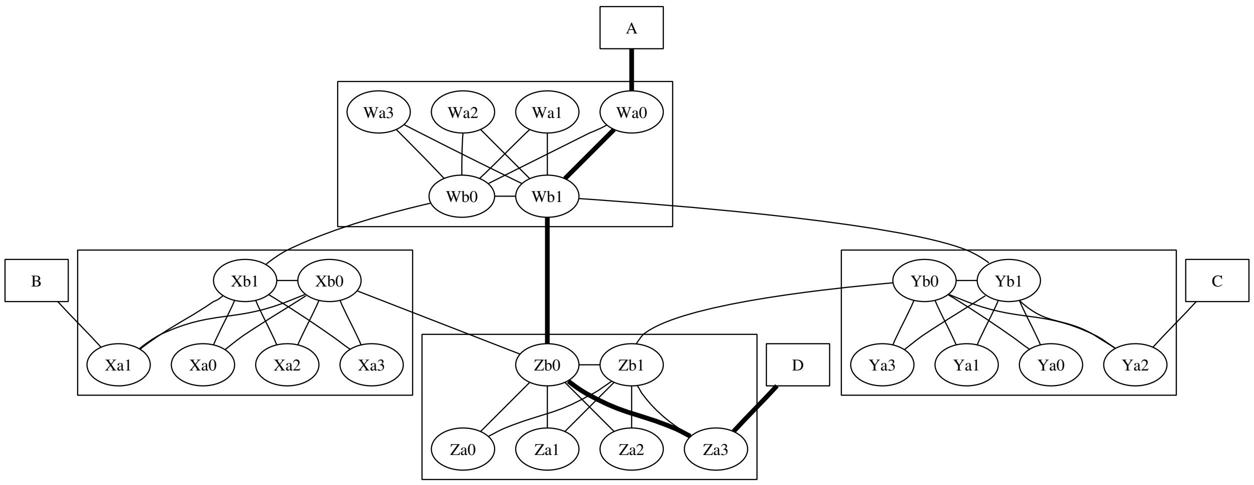


Figure 6: A simple network topology. The bold lines show the normal shortest path from A to D.

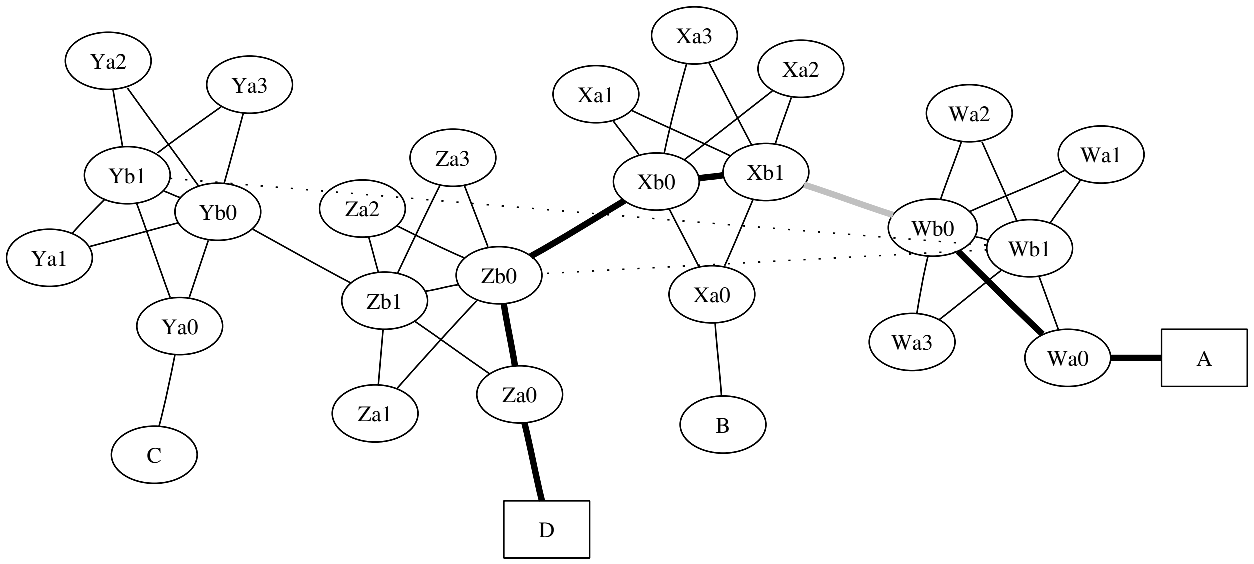


Figure 7: Monitoring a simple network topology. The bold lines show the new shortest path from A to D, when the attacker is monitoring the link Wb0–Xb1, shown in gray.

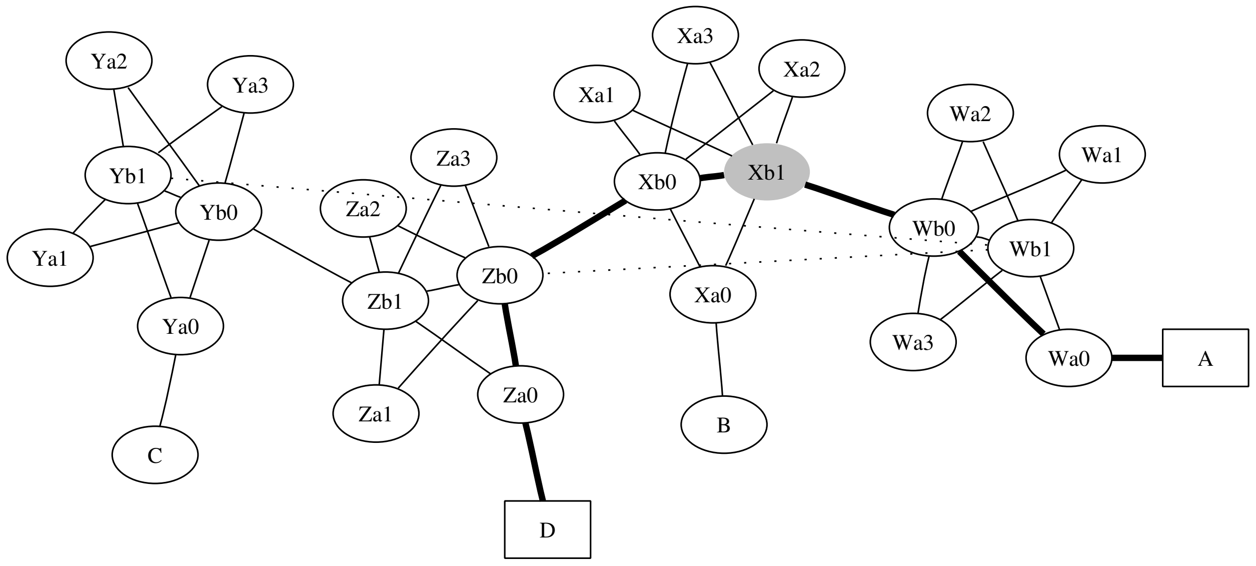


Figure 8: The attacker has compromised node Xb1.

5 Simulation Results

We implemented the algorithm and ran it on some test topologies. In all cases, the goal of the attacker was to monitor traffic from one arbitrary point to another, given control of some links.

We used three topology sets for our test runs: a simple, artificial topology modeled on a real (but small) ISP, an actual ISP backbone, and a variety of topologies derived from actual BGP data. Unless otherwise noted, all edges have equal cost.

5.1 Artificial Topology

The artificial network topology we used is shown in Figure 6. In it, the attacker wishes to monitor traffic flowing from host A to D. We use it to demonstrate several different attacks.

In the first, the attacker is monitoring interPOP link Xb1–Wb0. Cutting link Wb1–Zb0 blocks the direct path; cutting Wb1–Yb1 prevents routing through POP Y. Figure 7 shows the result.

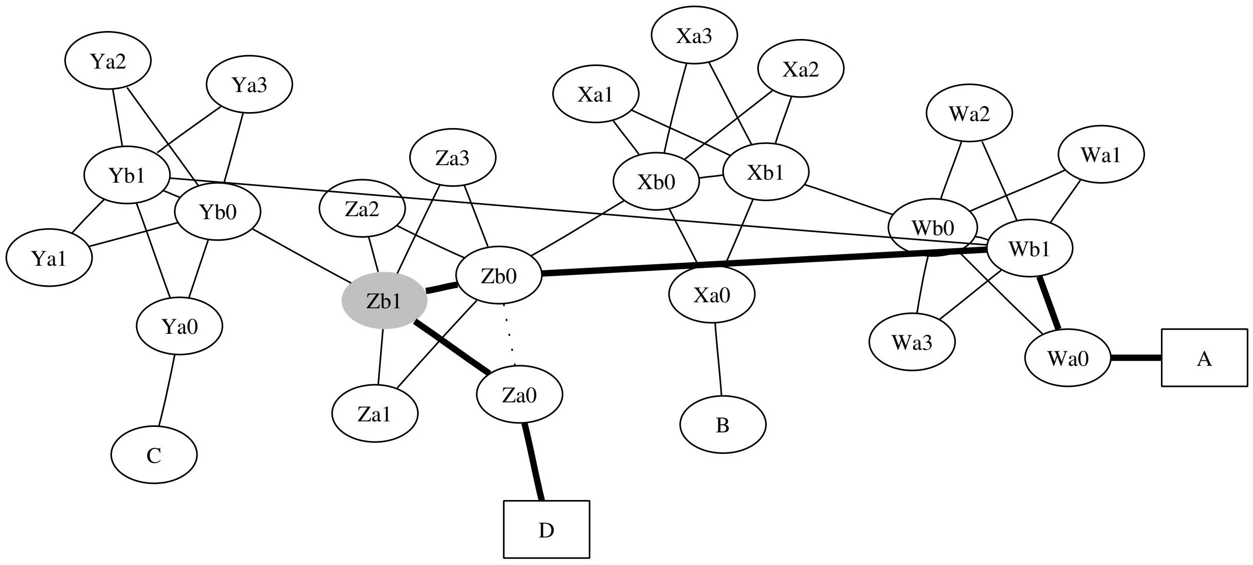
In Figure 8, the attacker controls a router in the X POP. Cutting the same links forces traffic through the compromised router in the POP.

The third example (Figure 9), where the attacker controls node Zb1, demonstrates the results of different attack difficulty levels. In Figure 9a, the shortest new path (and the fewest link cuts) are shown; in this case, the link to the access router Za0 is severed. (This example also shows why POPs are designed the way they are, to be resistant to single-element failures.) If, on the other hand, we assume, that it is easier to cut an interPOP link, we get Figure 9b; the new path is longer, and requires more link cuts, but may be easier to mount in practice.

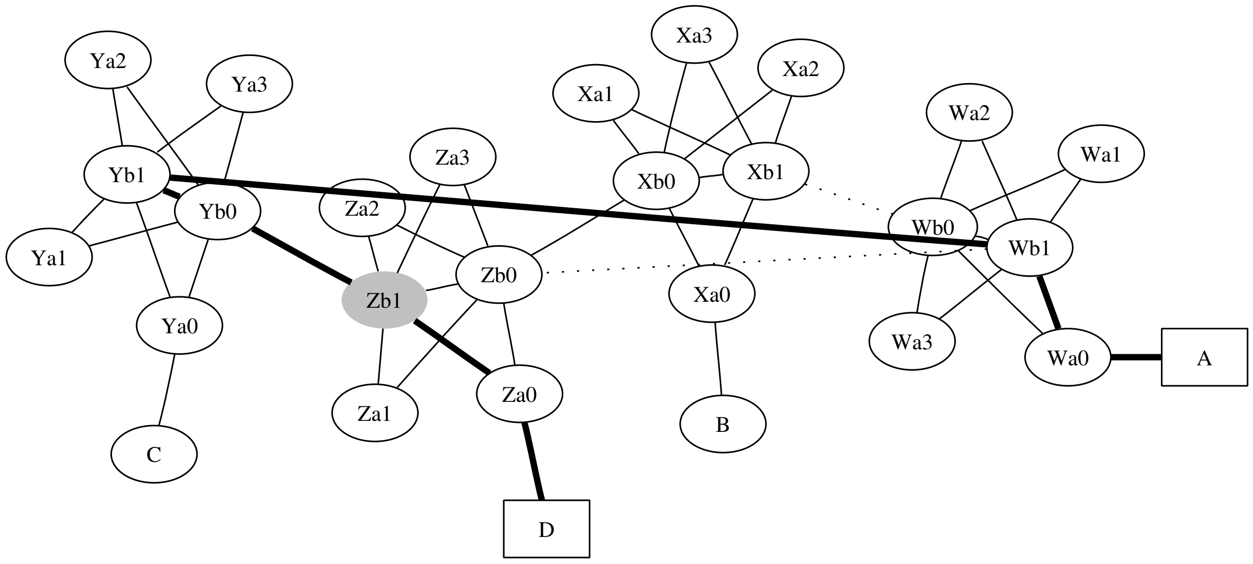
5.2 An ISP Network

To study the behavior of our algorithm on a real ISP's network, we selected one of the backbone topologies from the *Rocketfuel* collection. The dataset does not include access routers, but that is not a significant disadvantage; as noted earlier, routing never takes place through access routers except at the source or destination. This renders them largely useless for our purposes; they're not usable eavesdropping points in the center of a path, and anyone with enough access to a source or destination POP to do eavesdropping there could just as easily eavesdrop on the actual target's link.

We ran 200 tests on a network with several hundred routers. The tests were divided into four categories: compromised routers versus compromised links, and one compromised element versus five. A source node, a destination node, and one or more



(a)



(b)

Figure 9: Node Zb1 is compromised. (a) shows the route if links within a POP can be cut; (b) shows the route if interPOP links are cut.

Table 1: Test results on a real ISP's backbone.

Type	*Points*	*OrigHop*	*Hop*Δ	*Cuts*	*Fail*
Router	1	2.52	3.30	4.58	5
Router	5	3.86	2.04	4.06	0
Link	1	3.42	3.04	5.74	11
Link	5	3.84	2.40	4.56	1

Type	The network element that was compromised.
Points	The number of elements compromised by the attacker.
OrigHop	The original average hop count from the source to the destination.
HopΔ	The average increase in hop count after the attack.
Cuts	The number of link cuts required.
Fail	The number of times our algorithm could not find a solution, even though one is theoretically possible.

compromised elements were selected at random for each trial. (In the first test set, eight of the cases were impossible to solve.) The results are shown in Table 1; a number of items stand out.

The first is that the attack is practical: in 80-90% of the cases, our algorithm succeeded in finding a set of links to cut. The failure rate is due to the limitations of the heuristics we currently employ; we expect that the failure rate can be reduced.

The second is that our intuition is correct in one regard: the more nodes the attacker has compromised, the easier the attack is. Fewer links need to be cut, the increase in hop count due to the attack is not as great, and there are fewer failures.

What is less obvious is the tradeoff between compromising routers and compromising links. Fewer link cuts are necessary with compromised routers, and there are fewer failures; however, the effect on hop count is ambiguous.

The effect on hop count is greater than we would like. In retrospect, this is not surprising; ISPs engineer their backbones for efficient transport; any serious disruption will naturally cause a significant increase in path length. This does suggest a countermeasure for end-systems: monitoring path lengths to major destinations might show that a link-cutting attack is in progress [25].

Although not shown in the table, the algorithm ran extremely quickly; no test run took longer than .1 seconds.

Table 2: Test results on BGP-derived topologies. The first five entries are for graphs whose nodes have an average degree of 6.96; the next five have an average degree of 5.00. The starred entries are for a better-positioned attacker. Each line represents 800 simulations.

Type	*Points*	*OrigHop*	*Hop*Δ	*Cuts*	*Fail*
Router	1	1.29	1.69	26.67	21
Router	5	1.65	1.24	18.90	8
Link	1	1.49	2.07	32.38	71
Link	5	1.66	1.51	25.00	6
Link*	5	1.66	0.52	6.09	2
Router	1	1.29	1.63	15.79	26
Router	5	1.81	1.25	11.41	12
Link	1	1.51	2.00	20.20	137
Link	5	1.82	1.48	15.07	18
Link*	5	1.84	0.54	4.39	4

5.3 Inter-AS Topologies

We also ran the algorithm on inter-AS topologies. We derived 160 test topologies from real inter-AS maps, as described in Appendix A. To test the sensitivity of the algorithm to graph size and connectivity degree, we used successively smaller graphs, and graphs of lesser degree. Because more internet traffic originates from bigger ISPs, we used a weighted random distribution to favor ISPs—ASs, to be precise—with more connectivity. Initially, we used a uniform random distribution to select the location of the attacker; larger ISPs have more network elements subject to compromise, but they also have larger, and generally more sophisticated, staffs and procedures.

The inter-AS core is richly connected; this shows in the low original hop count (Table 2). It is also manifested in the lower failure rate, which didn't exceed 17.125%; in a rich topology, it is easy to find paths to the attacker's network elements. But the abundance of links makes the attacker's link-cutting job much harder; as shown in the table, very many links need to be cut.

We relocated the attacker by using the same weighted mechanism to select the attacker's locations. The results were dramatic: the number of links to be cut was reduced by a significant factor, the failure rate went down, and the increase in path length from the attack was diminished. This suggests that attacker location is crucial.

Intuitively, one would expect the attacker's problem to be easier for smaller networks, and for networks with smaller average degree. Figure 10 shows that this is indeed the case. The graph shows both the normal and reduced-degree versions of the second dataset from Table 2. The need to account for routing policy thus has another benefit: it will reduce the number of links that need to be cut.

We ran one test using the full BGP connectivity graph, with 15K nodes and 28K links. While in no sense statistically valid, we wanted to demonstrate that our prototype, unoptimized code could handle a network of that size. Using weighted random locations for the source, destination, and five compromised ASs, we found a new path of the same length in less than seven seconds. (All timings were done on a rather slow computer. In other words, as enemies, we weren't trying very hard and still did pretty well.)

6 Countermeasures

Link-cutting attacks are hard to counter, because they can occur even when all mechanisms are operating properly. In particular, routing protocols are *intended* to find alternate paths in the face of link failures.

Perhaps the best solution is to use encryption and not worry about the problem. Properly-encrypted traffic is immune to eavesdroppers and connection hijackers; a variety of anonymity schemes [26, 27] can be used to foil traffic analysis. (On the other hand, it may be possible to use link-cutting to enable monitoring attacks against anonymous communications networks [28].) Similarly, link encryption can prevent an attacker from exploiting an attack.

Encryption is not always feasible, so we should look for other solutions.

Analytically, link-cutting attacks work because the attacker can disrupt the topology. Mechanisms that preserve the topology are thus a strong defense. Options to do this include hard-to-cut links, such as spread-spectrum radio or line-of-sight lasers, or restoration techniques such as AT&T's FASTAR system [29].

Protecting in this fashion against node disabling is harder. A replacement node would need to be topologically identical, geographically distant (against some threat models), and built on a different software base. Deployment expense for such a scheme is probably prohibitive.

A rich topology can make the attack much more difficult to carry out. The calculations remain feasible, but cutting enough links to carry out the attack will be much more difficult. Note that this countermeasure may require that these links be real, and not just virtual circuits over a shared infrastructure. Furthermore, rich topologies complicate the ordinary network engineering tasks, and slow down routing convergence after topology changes.

It is tempting to try to keep the topology secret. Experience suggests that that will not work well. Apart from mapping techniques [17, 30], topologies are hard to change; even old information is likely to be substantially correct.

Topology monitoring can provide end systems with a clue that this type of attack is in progress. As noted, the attack can have a significant effect on path length; this is observable. Wang et al. [25] describe similar monitoring to detect changes in root server accessibility. Note, though, that sometimes alternate paths can have the same cost as the original.

The most intriguing countermeasure uses an *intrusion detection system* (IDS) linked to the routing protocols. In the simplest form, an IDS could notice unusual rates of link failures. But a more sophisticated response may be feasible. We sketch a possible design here.

Assume that a network operator has a list of sensitive end customers and a list of links that are more exposed to eavesdropping. In response to link failures, traffic will be rerouted. If, at such a time, there is a marked increase in traffic between sensitive sites over exposed links, an attack may be in progress. The IDS would then respond by adjusting routing

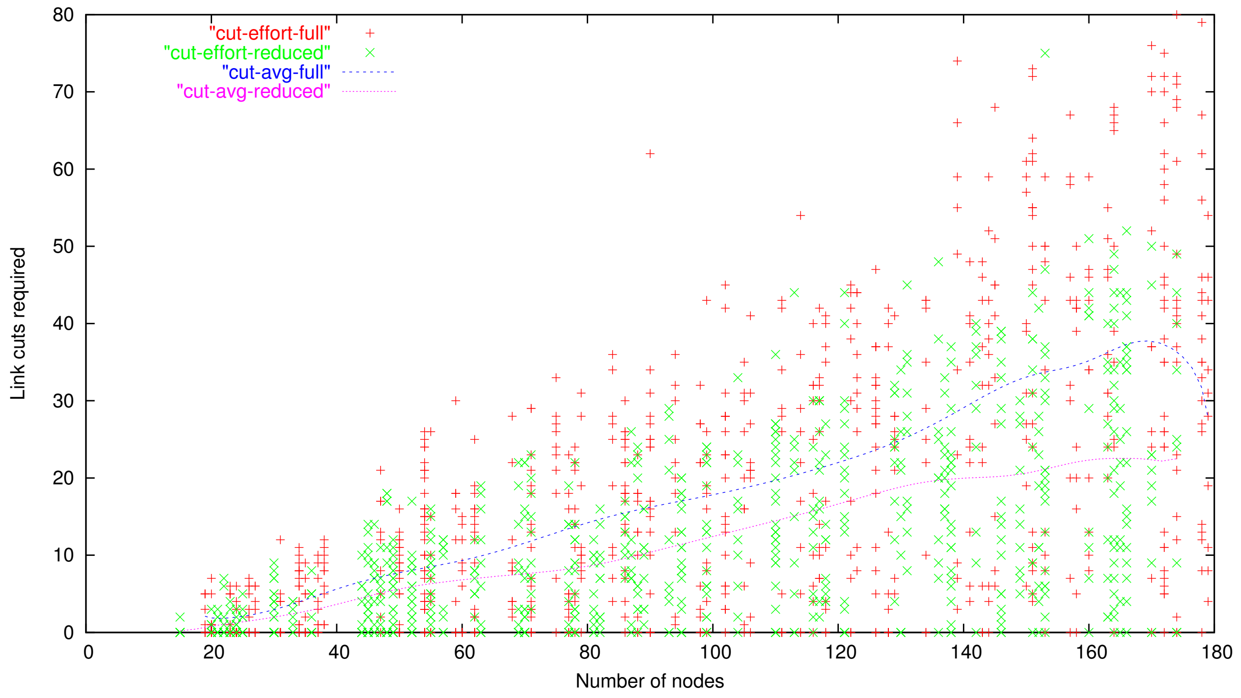


Figure 10: Link cuts as a function of nodes in the graph. The "reduced" plots are for graphs of lower degree. The lines represent smoothed averages for each number of nodes.

metrics to divert traffic, especially sensitive traffic, from such links.

To be sure, the enemy could counter by cutting more links. This sort of outage, in response to a countermeasure, would be a very strong indicator that an attack was indeed in progress.

7 Future Directions

Cutting links to facilitate eavesdropping is not new. Kahn reports that on the very first day of World War I, the British cut Germany's transatlantic telegraph cables, forcing them to use radio instead [31]. What is new here is the demonstration of how to calculate what links to cut in complex but realistic topologies. We have demonstrated that the calculations and the attack are feasible.

There are a number of important ways in this work can be extended. The first is to deal with the complexities omitted from our analysis: routing policy and complex layer 2 topologies. The two are closely related; the lack of some possible virtual circuit connections can be considered to be a form of routing policy.

More generally, we need to explore the question of fate-sharing. As noted, even simple layer 2 topologies are often implemented by multiple circuits on the same fiber pair; if one link is cut by physical means, others will be cut as well. The same issue applies to routers: in a war situation, the military may find it easier to destroy an entire POP, rather than just one or two routers.

Our current algorithm deals with interception of traffic between two nodes. Sometimes, an adversary would like to monitor traffic among three or more nodes. A more sophisticated algorithm would determine the link cuts that would satisfy multiple constraints simultaneously. If nothing else, routing on the Internet is generally asymmetric; an adversary would generally want to monitor traffic in both directions of any particular path.

As noted, the effect of attacker placement is critical. It would be interesting to design algorithms that located desirable nodes to compromise for a given number of link cuts. More generally, this algorithm can be seen as a tool in designing more sophisticated link-cutting attacks, where the attacker can minimize the number of links to cut, the increase in hop count, or the number nodes or links that must be compromised.

Even without extending the problem, there is much room for improvement in our current algorithm. As we noted in Section 4, our approach provides only an approximate solution to a restricted version of the general problem. Though we obtain reasonable solutions in practice, we are far from claiming that our results are near optimal. In particular, by restricting the problem, we only make local use of the given edge costs. Since, in reality, severing a fiber between POPs may be much easier than severing a link within a POP, it is important to broaden the algorithm to make more effective use of these numbers. In this, there are two causes for optimism. First, the current algorithm uses so little time we can afford a more expensive method. Second, at present we largely consider the problem as one involving abstract graphs. However, real network topologies have graph characteristics which, if considered, may aid in the solution.

The two-layer strategy described in Section 3 can result in cutting more links than needed. While the two-layer strategy mirrors the routing mechanisms, it might be possible to optimize the end result, minimizing the number of links that must actually be cut.

Finally, the current algorithm deals with compromised links or routers, but works by cutting links. In some situations, it's easier to take out a router than a link. This, too, can be modeled as a fate-sharing problem: all links that enter or leave a node are cut at the same time. Note, though, that in this case fate-sharing is *not* transitive; other links that share the same physical path would not be affected.

Acknowledgments

Most of the networks were drawn using *graphviz*.[3]

The real network topologies we used were taken from the *Rocketfuel* collection at the University of Washington.[4]

Tim Griffin supplied the data used for the inter-AS topology; the source data was taken from the *Routeviews* collection at the University of Oregon.[5] Tim also made many useful comments on an early draft of this paper. Adam Buchsbaum and David Johnson provided helpful conversations on the related shortest path problems.

[3]`http://www.research.att.com/sw/tools/graphviz`

[4]`http://www.cs.washington.edu/research/networking/rocketfuel`

[5]`http://www.routeviews.org`

References

[1] Laurent Joncheray, "A simple active attack against TCP," in *Proceedings of the Fifth Usenix* UNIX *Security Symposium*, Salt Lake City, UT, 1995.

[2] Radia Perlman, *Network Layer Protocols with Byzantine Robustness*, Ph.D. thesis, M.I.T., 1988.

[3] Steven M. Bellovin, "Security problems in the TCP/IP protocol suite," *Computer Communications Review*, vol. 19, no. 2, pp. 32–48, April 1989.

[4] J. Moy, "OSPF version 2," RFC 2328, Internet Engineering Task Force, Apr. 1998.

[5] R. Rivest, "The MD5 message-digest algorithm," RFC 1321, Internet Engineering Task Force, Apr. 1992.

[6] A. Heffernan, "Protection of BGP sessions via the TCP MD5 signature option," RFC 2385, Internet Engineering Task Force, Aug. 1998.

[7] S. Murphy, M. Badger, and B. Wellington, "OSPF with digital signatures," RFC 2154, Internet Engineering Task Force, June 1997.

[8] Stephen Kent, Charles Lynn, and Karen Seo, "Secure border gateway protocol (Secure-BGP)," *IEEE Journal on Selected Areas in Communications*, vol. 18, no. 4, pp. 582–592, April 2000.

[9] Stephen Kent, Charles Lynn, Joanne Mikkelson, and Karen Seo, "Secure border gateway protocol (S-BGP) – real world performance and

deployment issues," in *Proceedings of the IEEE Network and Distributed System Security Symposium*, February 2000.

[10] Y. Rekhter and T. Li, "A border gateway protocol 4 (BGP-4)," RFC 1771, Internet Engineering Task Force, Mar. 1995.

[11] Y. Rekhter and P. Gross, "Application of the border gateway protocol in the internet," RFC 1772, Internet Engineering Task Force, Mar. 1995.

[12] W. Simpson, "PPP over SONET/SDH," RFC 1619, Internet Engineering Task Force, May 1994.

[13] A. Malis and W. Simpson, "PPP over SONET/SDH," RFC 2615, Internet Engineering Task Force, June 1999.

[14] Hal Burch and Bill Cheswick, "Tracing anonymous packets to their approximate source," in *Usenix LISA*, New Orleans, 2000.

[15] Qian Chen, Hyunseok Chang, Ramesh Govindan, Sugih Jamin, Scott J. Shenker, and Walter Willinger, "The origin of power laws in internet topologies revisited," in *Proceedings of IEEE Infocom 2002*, 2002, To appear.

[16] Hyunseok Chang, Ramesh Govindan, Sugih Jamin, Scott J. Shenker, and Walter Willinger, "On inferring AS-level connectivity from BGP routing tables," 2002, Draft.

[17] Neil T. Spring, Ratul Mahajan, and David Wetherall, "Measuring ISP topologies with Rocketfuel," in *Proceedings of SIGCOMM 2002*, 2002.

[18] R. W. Callon, "Use of OSI IS-IS for routing in TCP/IP and dual environments," RFC 1195, Internet Engineering Task Force, Dec. 1990.

[19] G. Malkin, "RIP version 2," RFC 2453, Internet Engineering Task Force, Nov. 1998.

[20] L. Gao, "On inferring autonomous system relationships in the Internet," in *Proc. IEEE Global Internet Symposium*, November 2000.

[21] L. Subramanian, S. Agarwal, J. Rexford, and R. Katz, "Characterizing the internet hierarchy from multiple vantage points," in *Proceedings of IEEE INFOCOM*, June 2002.

[22] Giuseppe Di Battista, Maurizio Patrignani, and Maurizio Pizzonia, "Computing the types of the relationships between autonomus systems," in *Proceedings of IEEE INFOCOM*, 2003.

[23] Neil Spring, Ratul Mahajan, and Tom Anderson, "Quanitfying the causes of path inflation," 2003, Draft.

[24] R.K. Ahuja an dT.L. Magnanti and J.B. Orlin, *Network Flows*, Prentice-Hall, Saddle River, NJ, 1993.

[25] Lan Wang, Xiaoliang Zhao, Dan Pei, Randy Bush, Dan Massey, Allison Mankin, S. Felix Wu, and Lixia Zhang, "Protecting bgp routes to top level dns servers," in *Proceedings of the 23rd International Conference on Distributed Computing Systems (ICDCS)*, May 2003.

[26] David L. Chaum, "Untraceable electronic mail, return addresses, and digital pseudonyms," *Communications of the ACM*, vol. 24, no. 2, pp. 84–90, 1981.

[27] P F Syverson, D M Goldschlag, and M G Reed, "Anonymous connections and onion routing," in *IEEE Symposium on Security and Privacy*, Oakland, California, 4–7 1997, pp. 44–54.

[28] Matt Wright, Micah Adler, Brian Neil Levine, and Clay Shields, "Defending anonymous communication against passive logging attacks," in *Proc. IEEE Symposium on Research in Security and Privacy*, Berkeley, CA, May 2003.

[29] C. W. Chao, P. M. Dollard, J. E. Weythman, L. T. Nguyen, , and H. Eslambolchi, "FASTAR — a robust system for fast ds3 restoration," in *Proceedings of IEEE Globecom '91*, 1991, pp. 39.1.1–39.1.5.

[30] Bill Cheswick, Hal Burch, and Steve Branigan, "Mapping and visualizing the internet," in *Usenix*, 2000.

[31] David Kahn, *The CodeBreakers*, Macmillan, New York, 1967.

A Derivation of the Inter-AS Graph

The inter-AS graph we used was derived from *Route-views* data from 8 April 2003. The full data set listed over 28,000 links from over 15,000 ASs; we used a small fraction of the data.

Our base graph was generated by selecting the top 200 ASs as sorted by degree of connectivity. We extracted all links that were between two members of that list. That left us with 195 ASs and 1186 links between them, for an average degree of 12.16.

For our simulations, we then generated 80 smaller graphs by randomly discarding some nodes according to a uniform random distribution. 10 were generated for each discard percentage. For each of these smaller graphs, we created a sparser topology by discarding approximately half the links. (A few nodes were deleted as well.)

The following table shows the approximate percentage discarded, and the number of remaining nodes and the average degree for the original and sparse versions.

Percent	Full		Sparse	
discarded	Nodes	Degree	Nodes	Degree
10	173.30	11.13	164.50	7.87
20	153.20	9.85	145.00	6.86
30	130.10	8.54	121.70	5.97
40	107.30	7.10	100.00	4.96
50	88.50	6.29	80.30	4.50
60	66.90	5.15	58.80	3.93
70	47.60	4.24	43.20	3.20
80	26.50	3.36	23.50	2.67