On Static Reachability Analysis of IP Networks

Geoffrey G. Xie, Jibin Zhan, David A. Maltz, Hui Zhang[1]
Albert Greenberg, Gisli Hjalmtysson, Jennifer Rexford[2]

June 2004

CMU-CS-04-146

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

[1] Carnegie Mellon University. Emails:{geoffxie,jibin,dmaltz,hzhang}@cs.cmu.edu. Geoffrey Xie is visiting from Naval Postgraduate School.

[2] AT&T Labs–Research. Emails: {albert,gisli,jrex}@research.att.com. Gisli Hjalmtysson is also at Reykjavík University

This research was sponsored by the NSF under ITR Awards ANI-0085920, ANI-0331653, and ANI-0114014. Views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of AT&T, NSF, or the U.S. government.

A condensed version of this report appears in IEEE INFOCOMM 2005 Proceedings

Keywords: routing protocols, routing design, routing analysis

Abstract

The primary purpose of a network is to provide reachability between applications running on end hosts. In this paper, we describe how to compute the reachability a network provides from a snapshot of the configuration state from each of the routers. Our primary contribution is the precise definition of the *potential reachability* of a network and a substantial simplification of the problem through a unified modeling of packet filters and routing protocols. In the end, we reduce a complex, important practical problem to computing the transitive closure to set union and intersection operations on reachability set representations. We then extend our algorithm to model the influence of packet transformations (e.g., by NATs or ToS remapping) along the path. Our technique for static analysis of network reachability is valuable for verifying the intent of the network designer, troubleshooting reachability problems, and performing "what-if" analysis of failure scenarios.

Contents

1 Introduction

While the ultimate goal of networking is to enable communication between hosts that are not directly connected, a wide variety of mechanisms are being used to *limit* the set of destinations the hosts can reach. For example, backbone networks may provide Virtual Private Network services to connect only remote offices belonging to the same enterprise, and enterprise networks themselves are often segmented into departments or offices whose hosts must be isolated for business or security reasons. Also, due to a configuration or design mistake, two hosts may not be able to communicate under certain failure scenarios, even though the network remains connected; knowing when these vulnerabilities exist is crucial to building a more reliable network.

Determining what kinds of packets can be exchanged between two hosts connected to a network is a difficult and critical problem facing network designers and operators. To our knowledge, the problem is largely unexamined in the networking research literature. Solving the problem requires knowing far more than the network's topology or the routing protocols it uses. For example, despite having a route to a remote end-point, a sender's packets may be discarded by a packet filter on one of the links in the path. The network's packet filters, routing policies, and packet transformations all must be taken into account to even ask the simple and very important question of "can these two hosts communicate?"

This paper crystallizes the problem of calculating the *reachability* provided by a network. By mapping packet filters, routing information, and packet transformations to a single unified model of reachability we have determined how to transform this seemingly intractable problem into a classical graph problem that can be solved with polynomial time algorithms such as transitive closure. This is the primary contribution of this paper.

1.1 Advantages of Automated Static Analysis

Currently, the common practice to determine if packets can reach from one point in a network to another is to use tools such as `ping` and `traceroute` to send probe traffic that experimentally test whether reachability exists. In contrast, we have developed a *static-analysis* approach that can be applied even if only a description of the network is available. Static analysis has many advantages over `ping` and `traceroute`, including:

- The ability to determine a description of the set of packets that *could* traverse the network from a given starting point to a given ending point, whereas experimental techniques can only check the reachability of the specific probe traffic they send.

- The ability to calculate the set of routers and hosts that a given packet could potentially reach, whereas `ping` and `traceroute` can only check reachability along the path currently selected by the routing protocols.

- The ability to evaluate the reachability of a network during its *design phase*—before the network has been deployed or a problem has arisen. Network operators can perform our static analysis using only the configuration files used to program the network's routers, and these files are readily available to them.

- The ability to verify whether the reachability a network actually provides matches the designer's intent. Static analysis can verify that Virtual Private Networks are, in fact, isolated from other traffic. It can also be used to conduct "what-if" analysis—predicting the effects of equipment failures and planned maintenance on the communication between end hosts. While syntax verification of router configuration has been evaluated [3, 8], there is little understanding of the power and limitation of semantic verification based on static analysis.

Manually calculating the static reachability of a network is often impractical, as data show that campus, enterprise, and backbone networks vary in size from 5 to 500 routers, with the largest networks having on the order of 1,000 routers, and that real networks use a wide variety of mechanisms to control the reachability they provide. A survey of 31 production networks [10], including examples of both carrier backbone and enterprise networks, found that 10 out of the 27 enterprise networks had packet filters applied to their *internal* links. Several of the networks deliberately prevented some hosts from reaching others by preventing the distribution of routing information needed to direct packets between the hosts. Further complicating the question of a network's reachability is the use of mechanisms that actually *transform* packets as they travel across the network. For example, Network Address Translators (NATs) [16] that change a packet's source and destination address were found in the *interior* of 10 of the 31 networks. Understanding the reachability "matrix" created by a network requires a framework for reasoning about the effects of all these different mechanisms—packet filters, routing policy, and packet transformations—at the same time.

1.2 Our Contributions

First, we *formulate the problem of computing the reachability of a network* and argue for the importance of crafting good solutions. We focus on the value of computing reachability through static analysis. We rigorously define the reachability of a network, and we define expressions for upper and lower bounds on the reachability.

Second, we describe a *tractable framework for jointly reasoning about how packet filters, routing, and packet transformations affect the reachability* that a network provides. Bringing together these three very different types of mechanisms is critical to accurately computing the reachability of a network.

Third, we present an *algorithm for the static analysis of reachability* for IP networks and explain how the network model can be populated by static analysis of the network's router configuration files.

1.3 Structure of the Paper

In Section 2, we present a brief overview of the most relevant aspects of how routers operate and are configured. We then formally describe our framework for analyzing a network's reachability in Section 3, beginning our analysis by focusing on packet filters. We present our algorithm for calculating reachability in Section 4. In Section 5 we show how to map routing information to packet filters and how this model of routing is populated by analyzing

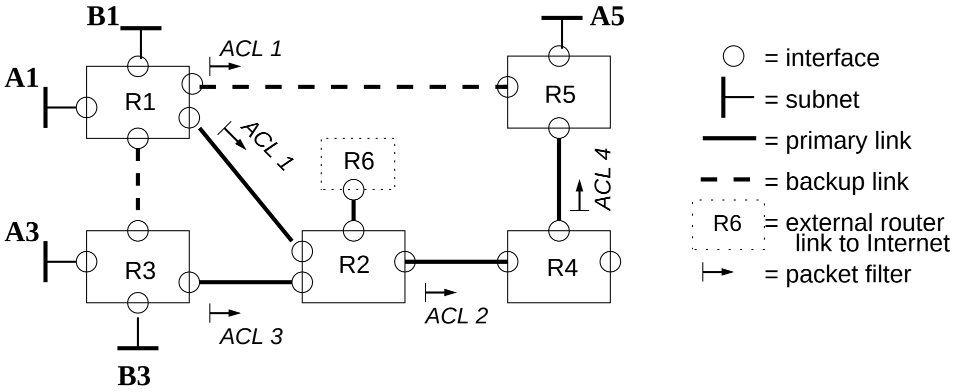


Figure 1: An example enterprise network with five routers

the router configuration files. In Section 6 we describe how packet transforming mechanisms are handled. Section 8 discusses the applications and limitations of our approach. After a brief overview of related work in Section 9, the paper concludes in Section 10 with a summary of our contributions.

2 Background on Reachability Configuration

In addition to forming the physical topology of routers and links, network operators must configure the protocols and mechanisms that collectively determine which hosts can communicate. Today's routers offer a wealth of configuration options for enabling and tuning packet filters, routing protocols, and packet transformations. Our analysis techniques operate on a snapshot of the configuration state for each of the routers in the network, as recorded in a *configuration file*. In well-managed networks, these files are routinely captured and archived for backup purposes, and are available to network operators.

To make our discussion of the different reachability configuration options more concrete, we focus on the example enterprise network in Figure 1. The network has five routers R1 to R5 (depicted as solid rectangles) connected via physical links (depicted as solid lines) that terminate at interfaces (depicted as small circles). R1 and R3 are remote sales offices connected directly to the central office where R2, R4, and R5 reside. R6 represents the external router in the service provider's network where the enterprise connects to the Internet.

Each sales office has two subnets, A and B. Critical accounting applications are run by hosts connected to subnet A, and general purpose computers are connected to subnet B. Hosts on subnets A1 and A3 must be able to communicate with corporate servers in subnet A5, but the network's policy is to prevent any other hosts from communicating with the servers on A5 to reduce the chances of a server compromise. To make the network more resilient to link failures, the operators are planning to add two backup links (shown with dashed lines). In Section 4 and beyond, we show that our reachability analysis technique can predict the effect of adding these links and prevent a design error that would violate the network's goals.

ACL	Definition
ACL 1	permit tcp A1 A5 port eq 1433 deny tcp any any port eq 1433
ACL 2	deny 77 any any
ACL 3	permit tcp A3 A5 port eq 1433 deny tcp any any port eq 1433 deny ip any 224.0.0.0/8
ACL 4	deny 55 any any

Table 1: Four packet filters instantiated in Figure 1

2.1 Packet Filters

The simplest way to control reachability is to configure an interface to filter unwanted packets in the data plane. Today's routers allow operators to filter packets based on a combination of fields in the packet header, such as source and destination IP addresses, type-of-service (ToS) bits, port numbers, and protocol. Each packet filter consists of a sequence of clauses that that permit or deny certain packets based on their header fields. A filter can be instantiated on a particular interface to filter incoming or outgoing packets. An interface may have different filters for incoming and outgoing packets, and different interfaces may be assigned different filters.

Table 1 shows four access-control list (ACL) specifications, defined in the Cisco IOS language; packets not matching any clause are permitted by default. Figure 1 shows where these ACLs are used to filter outgoing packets on four interfaces. ACL1 permits TCP packets destined to Microsoft SQL servers (port 1433) in subnet A5 from hosts in A1, but denies them from any other subnet; instantiating this packet filter on the link from R1 to R5 is meant to prevent other subnets from accessing the corporate servers on subnet A5. ACL2 drops all Sun ND protocol packets (protocol 77), which were implicated in an earlier attack on Cisco routers. Like ACL1, ACL3 permits TCP packets to the Microsoft SQL server (port 1433) from hosts in A3, but denies them from any other subnet. ACL 3 also prevents multicast packets (in the IP address range 224.0.0.0/8) from leaving the office containing R3. ACL4 drops all Mobile IP packets (protocol 55), which were also implicated in an earlier attack on Cisco routers.

2.2 Routing Protocols

Routing protocols influence reachability by controlling the construction of the forwarding table on each router. Conceptually, a *route* is a network address (e.g., an IP address and a mask length, such as 10.0.0.0/8) along with additional attributes (e.g., numerical weights, AS paths, or next-hop IP address) that a router can use to determine which outgoing link to use to reach that subnet. A router can learn a route in several ways. First, a router knows locally how to reach all directly-connected subnets—the incident links themselves. Second, the router may be configured with static routes that map a destination subnet directly to one or more outgoing interfaces. Third, the router may learn the information dynamically through a routing protocol, such as OSPF [11], IS-IS [4], BGP [13], RIP [9], or EIGRP [18].

To control the sharing of routing information, each instance of a routing protocol runs as a separate *routing process* on the router. Just as with operating system process boundaries, by default no information is exchanged between these entities, and they operate completely independently. Each routing process has a *Routing Information Base (RIB)* that stores the routes on which it operates, similar to the virtual memory space of a process. To simplify the discussion, we consider the directly-connected subnets and static routes as belonging to a single process that creates a *local RIB*. A router can run multiple routing processes simultaneously, including multiple instances of the same routing protocol. For example, Figure 2 illustrates the routing processes (as represented by their RIBs) for the network in Figure 1, after the two backup links have been added. Router 2 runs two instances of OSPF and one instance of BGP, and has a local RIB.

Routing processes do not exchange information unless specifically configured to do so. The dashed lines in Figure 2 indicate adjacencies between routing processes on different routers, or route redistribution between RIBs on the same router. For example, router 2 exchanges routing information via OSPF with router 1 and router 3; routes from the OSPF RIB are redistributed to BGP and advertised via external BGP (EBGP) to router 6. The other instance of OSPF on router 2 does the same for routers 4 and 5. The routes from router 2's local RIB are also redistributed to the BGP RIB, and onward to router 6. Thus, router 2 takes responsibility for ensuring that the subnets in the enterprise network are reachable from the rest of the Internet via router 6.

Rather than exchanging routes to every subnet, the distribution of routes is governed by routing policies. A policy can be thought of as an annotation on the dashed line denoting the exchange (e.g., routing policies 1 (RP1) and 2 (RP2) in Figure 2). For example, router 2 could be configured to filter the route to subnet A5 (i.e., the sensitive corporate servers) when distributing routes via eBGP to router 6. Modern routers have rich languages for specifying routing policies, including the ability to select which routes should be imported or exported based on any of the attributes associated with the route (e.g., the subnet or the AS path). Routing policies can also alter the attributes of the routes they accept (e.g., changing metrics or adding an AS number onto an AS path).

Upon receiving multiple routes for the same subnet, the routing process must select a single best route. The selection of the best route depends on the route attributes and logic defined for the particular protocol. For example, BGP has a complex multi-stage process for identifying the best route [17], whereas OSPF selects the path with the smallest cost as the sum of the link weights [11]. If multiple RIBs on the same router have a best route for the same subnet, the router must determine which routing process should control the entry in the forwarding table. For example, the router may impose a static ranking on the routing processes (e.g., giving the local RIB priority over BGP-learned routes).

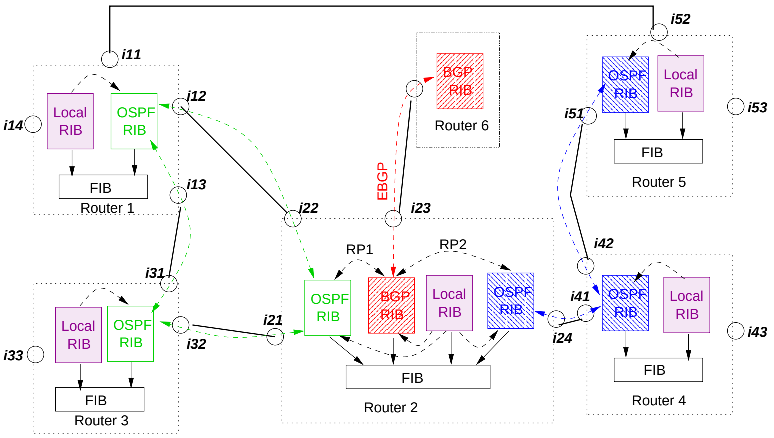


Figure 2: Interactions of routing processes in the example network in Figure 1. Each routing process is depicted by the RIB that stores its routes. Dashed lines indicate the import, export, and redistribution of routes. Interfaces are marked with identifiers and the solid lines between interfaces are the physical links.

2.3 Packet Transformations

The routers make packet filtering and forwarding decisions based on fields in the header of each packet. However, these header fields may change as a packet flows through the network. For example, the network operator may configure router R2 in Figure 1 to reset the ToS bits of incoming packets from R6. If the enterprise network assigns packets to different queues based on the ToS bits, setting the ToS bits to a default value would ensure that traffic coming from the Internet does not enter the same queue as high-priority internal traffic. Similarly, R2 could be configured to map the source IP addresses of packets leaving the network via R6, in order to use private IP addresses inside the enterprise and public addresses in communicating with the external Internet. Although stateful Network Address Translator (NAT) and firewall devices may transform or rate-limit packets in complex ways, the functionality supported (and enabled) directly in the routers is often much simpler. In our analysis, we focus on this simpler form of statically-configured transformations and how they influence the reachability between end hosts.

3 Problem Formulation

In this section, we formulate the reachability analysis problem. We first describe a graph model for computing reachability which allows joint reasoning of the effects of packet filters and routing protocols. We then formally define the reachability metrics targeted by our analysis. In particular, we introduce the concept of the instantaneous reachability provided by a network, and explain why it is useful to develop bounds on the reachability provided by the network. We end this section with an example illustrating the potential value of being able to compute the reachability bounds.

3.1 A Unifying Model

The crux of determining the reachability of a network is finding a way to unify two very different views of the network. The first is the graph of routers and links, illustrated in Figure 1, where vertices are routers and edges are physical links that may have packet filters applied on them. The second is the routing process graph, illustrated in Figure 2, where vertices are routing processes and edges are adjacencies that implement routing policy. Unifying these views requires combining the policies governing redistribution of routes with the packet filters governing which packets can traverse a link. This unified framework underlies our reachability analysis, and will be extended to address packet transforms in Section 6.

We define the reachability analysis problem by extending the graph of links and routers – annotating the edges of the graph more elaborately. Formally, we define the graph $G = (V, E, \mathcal{F})$ where V is the set of routers, E is the set of directed edges defining the connectivity between the routers, and $\mathcal{F}$ is a labeling function that annotates the edges in E. As the graph is directed, two routers directly connected by a physical link will have two edges between them, one in each direction. For each edge $< u, v > \in E$, $F_{u,v} \in \mathcal{F}$ represents the policies governing the flow of packets from u to v.[1]

The challenge in building this graph model G from the static analysis of configuration data is that $F_{u,v}$ cannot be a simple metric like an integer weight. It must embody the effects of the complex collection of packet filters and routing protocols used by the network, but still be amenable to efficient arithmetic-like manipulations. We define $F_{u,v}$ to be *the set of packets that the network is able to carry from u to v.* $F_{u,v}$ can also be represented by a packet filter $f_{u,v}$ containing predicates that test properties of packet p, returning true if the packet should be in the set $F_{u,v}$. Determining which packets can flow from router u to router v to router w can then be written simply as $F_{u,v} \cap F_{v,w}$ or $f_{u,v} \wedge f_{v,w}$.

The advantage of representing the reachability problem as a graph $G = (V, E, \mathcal{F})$ is that it exposes the similarity between our problem and the class of well-known problems such as transitive closure and shortest-path computation, allowing us to use their efficient solutions [1, 6] when computing the reachability of a network.

Packet filters defined by the network can be easily represented in the graph model G. Network configuration files define a packet filter f as a series of predicates over packet elements. For example, f may be "p.src_addr $\in$ 128.2/16 $\wedge$ p.dest_port $\neq$ 135", which accepts all packets from the 128.2/16 subnet except for those going to port 135. By parsing the configuration files we can extract the predicate f applied to the link from router u to router v, and annotate the edge $< u, v > \in E$ with the set of packets that f accepts, i.e., $F_{u,v} = \{p \mid f(p) = 1\}$.

The intuition behind our framework for jointly modeling routing and packet filtering is that routing can be thought of as a kind of dynamically constructed packet filter. If routing process on router **A** holds a route for subnet d with a next hop of interface i, it means **A** might forward packets to d out that interface. Therefore, we can treat this route as if it were a permit clause for d in the packet filter on interface i. Inversely, if router **A** holds no routes

[1] We believe our framework can be trivially extended to handle multiple physical links between u and v, but for the remainder of this paper we assume there is at most one physical link between each pair of routers.

that could possibly send packets to destination d out interface i, then we can add a clause to the packet filter on interface i to drop all packets headed to destination d.

3.2 Formal Definitions of Reachability Metrics

We describe the reachability between two points in a network in terms of the the subset of packets (from the universe of all IP packets) that the network will carry between those points. Thus, reachability from router i to router j is given by the subset of packets that the network will carry from i to j and is denoted as $R_{i,j}$. Note that it is common for $R_{i,j}$ to include packets that are neither sourced by a host connected to i nor destined to a host connected to j — this must be true if routers are to forward packets along multiple hops.

Clearly, the action of the network's routing protocols will directly influence $R_{i,j}$, for if router i has no routes for destination d, then packets to d cannot be elements of $R_{i,j}$, since i will be dropping those packets. More generally, the network is continually affected by events such as link failures and changes in routing advertisements received from peer networks. Through the action of routing protocols and other mechanisms, each router will populate its Forwarding Information Base (FIB) with information determining the interface(s) out which each packet should be sent. We define the collective contents of the FIB on each router in the network to be the network's *forwarding state,* denoted by s. We also define $\mathcal{S}$ to represent the set of all possible forwarding states that the network can possibly enter, as it responds to any imaginable set of external advertisements, link failures, etc.[2]

3.2.1 Instantaneous reachability

The Reachability provided by the network will change as a function of the network's forwarding state s, which may change from instant to instant as the network responds to events. Therefore, our first step is to precisely define the reachability provided by a network at a single instant in time, assuming that the forwarding state s in effect at that instant is known.

The influence of any given forwarding state $s \in \mathcal{S}$ on the reachability in the network can be accounted for by incorporating additional packet filters into $F_{u,v}$. In doing so, the policy annotation at each edge in the reachability analysis graph becomes a function of s, written as $F_{u,v}(s)$. Assume $I_u(s, d)$ to be a function that returns the set of next hop routers to which router u will forward packets destined to IP subnet d while the network is in forwarding state s. $F_{u,v}(s)$ can then be formally defined as an extension to the statically configured packet filters $F_{u,v}$.

$$F_{u,v}(s) = F_{u,v} \cap \{p \mid p.\text{dst\_addr} \in \{d \mid v \in I_u(s,d)\}\} \tag{1}$$

Let $\mathcal{P}(i, j)$ be the set of all loop-free paths from i to j in the network's physical topology. Using all these concepts, we can now precisely define the instantaneous reachability from i

[2] When conducting a particular analysis of a particular network, the human conducting the analysis might want to restrict $\mathcal{S}$ to the forwarding states reachable under a more restricted set of events, such as "no more than one link or router will fail at a time."

to j provided by the network while at routing state s as:

$$R_{i,j}(s) = \bigcup_{\pi \in \mathcal{P}(i,j)} \bigcap_{<u,v> \in \pi} F_{u,v}(s) \tag{2}$$

3.2.2 Bounding the Instantaneous Reachability

In theory, it should be possible to compute exactly what forwarding state, and thus what reachability, a network provides at any instant in time. After all, each router in the network is a computing device with its behavior programmed and controlled by configuration commands. Unfortunately, computing the instantaneous reachability of a network requires knowing the *current* topology (e.g., which links and routers are up or down) and the exact information given to the network by neighboring domains in the outside world (e.g., the routing updates from BGP peers). Dynamic information of this kind might not be available (e.g., the network is not deployed yet), and its use makes the instantaneous reachability results depend heavily on the exact inputs used. For example, if the exact set of routes offered by external peers to the network under analysis is known, then the reachability to those destinations at that instant could be calculated. However, the calculated reachability is applicable only in situations where the external peers offer exactly those routes, which severely limits the usefulness of the reachability analysis.

Further, computing the instantaneous reachability of a network requires knowing not only the configuration state of each router, it requires the tedious and error-prone coding of an exact bug-for-bug emulation of the decision logic used by the particular version of the software running on each router. (More than 200 different software versions were used by the routers in the 31 production networks we recently examined in our study of IP routing design [10].) While the routing protocols are defined by standards, each vendor has implemented them differently. For example, the Border Gateway Protocol (BGP) [13] defines a seven-step process for selecting a route to a destination, but Cisco has added several more decision steps in their implementation [17].

The goal of most network designers is to ensure that the network's behavior remains within some "acceptable operating region" under reasonable predictions of how routers/links might fail or outside events might change. This means that more useful than calculating the instantaneous reachability of a network is the ability to calculate *bounds* on the reachability provided by the network. That is, given some set of reasonable events, predict the "operating region" of the network. We do this by defining two key bounds: the upper bound on reachability, which is the largest set of packets the network will *ever* deliver between two points, and the lower bound on reachability, which is the largest set of packets the network will *always* deliver between two points.

Reachability upper bound: Formally, we define the upper bound of the reachability over all routing states as follows:

$$R^U_{i,j} = \bigcup_{s \in \mathcal{S}} R_{i,j}(s) \tag{3}$$

Conceptually, $R^U_{i,j}$ captures the notion that as external events change, the path the network chooses for a packet moving from i to j will change as a function of the route selection logic and the external routing advertisements. Therefore, taking the union of the set of packets that can traverse each path from i to j under each state s produces a superset of the instantaneous reachability — that is, $R^U_{i,j}$ is the set of packets that could *potentially* reach from i to j if the routing decisions were made appropriately. The set negation of $R^U_{i,j}$ is particularly useful, as a packet appearing in this complement of $R^U_{i,j}$ *cannot ever* reach from i to j. Essentially, the packet is blocked along every possible path. This allows us to verify whether the network enforces security policies intended to isolate traffic.

Reachability lower bound: Formally, we define the lower bound for reachability as follows:

$$R^L_{i,j} = \bigcap_{s \in \mathcal{S}} R_{i,j}(s) \tag{4}$$

Conceptually, $R^L_{i,j}$ captures the notion that a packet permitted to reach between i and j under all possible forwarding states $s \in \mathcal{S}$ will always be able to get from i to j. For the lower bound to give useful information about the network's routing design, we first need to restrict $\mathcal{S}$ to those routing states induced from a set of network events targeted by the analysis. In particular, $\mathcal{S}$ should not include any forwarding states corresponding to failure scenarios that would *physically* disconnect i and j (or $R^L_{i,j}$ will be trivially $\emptyset$).

$R^L_{i,j}$ is useful to network designers because the network's routing design guarantees that packets appearing in this set will be deliverable between i and j as long as the network is not physically partitioned. Designers can then verify that traffic requiring robustness appears in this set.

3.2.3 Approximating the Reachability Bounds

As discussed earlier, it is difficult and error-prone to precisely model the route selection logic and external routing advertisements under all events. Further, the size of $\mathcal{S}$ is enormous, even for small networks. Combined together, these two factors make it seem impossible to accurately compute $\mathcal{S}$ or $F_{u,v}(s)$ for every s. Therefore, we cannot use equation (3) to exactly compute $R^U_{i,j}$ or equation (4) to compute $R^L_{i,j}$. Instead, we must develop estimators for $R^U_{i,j}$ and $R^L_{i,j}$.

We denote estimators to $R^U_{i,j}$ and $R^L_{i,j}$ as $\hat{R}^U_{i,j}$ and $\hat{R}^L_{i,j}$, respectively. Ideally, these estimators should be looser bounds, that is:

$$\hat{R}^L_{i,j} \subseteq R^L_{i,j} \subseteq R_{i,j}(s) \subseteq R^U_{i,j} \subseteq \hat{R}^U_{i,j}$$

as this property maximizes the utility of $\hat{R}^U_{i,j}$ and $\hat{R}^L_{i,j}$ in verifying network properties. For example, a $\hat{R}^U_{i,j}$ that is looser than $R^U_{i,j}$ may incorrectly warn an operator that the packets in $\hat{R}^U_{i,j} - R^U_{i,j}$ could violate the network's traffic isolation policies, but in this situation a false-positive is much better than a false-negative.

Even simple estimators to $R^U_{i,j}$ and $R^L_{i,j}$ still have value in predicting network properties. For example, we can obtain simple estimators by ignoring the effect of the routing protocols entirely, so that $F_{u,v}$ models only the static packet filters defined on edge $< u, v >$. The upper bound can then be calculated by finding the set of packets that at least one path through the network will allow to pass from i to j, since there could be some routing state that chooses this path for the packets.

$$\hat{R}^U_{i,j} = \bigcup_{\pi \in \mathcal{P}(i,j)} \quad \bigcap_{<u,v> \in \pi} F_{u,v} \tag{5}$$

Similarly, the lower bound can be calculated by finding the set of packets that all paths from i to j allow to pass, since, so long as i and j are not partitioned, at least one of these paths will exist and *could* be chosen by the routing protocols.

$$\hat{R}^L_{i,j} = \bigcap_{\pi \in \mathcal{P}(i,j)} \quad \bigcap_{<u,v> \in \pi} F_{u,v} \tag{6}$$

It is straightforward to prove that this estimator $\hat{R}^U_{i,j} \supseteq R^U_{i,j}$ when combining the results of Theorem 2 and Theorem 3 presented in the Appendix. It can be shown that $\hat{R}^L_{i,j} \subseteq R^L_{i,j}$ if we can assume that when only one path exists from i to j the routing design of network is such that the path will be used.[3]

In Section 5, we describe an approach to approximating the effect of the routing protocols on reachability that yields tighter estimators. Our expectation is that further research will lead to better and better estimators.

3.3 Example Application of Reachability Analysis

In this subsection, we illustrate the value of the concept of upper and lower bounds on reachability even if we use only those simple estimators as given above. To do so, let us revisit the example network defined in Section 2 where network operators were considering adding two backup links. At a first glance, it may seem to be sufficient to reconfigure routing parameters on the routers to use the backup links under failure scenarios. However, checking the reachability bounds reveals that such a design is incorrect. Specifically, the table below compares two particular reachability bounds before and after the backup links are added, where $ACL\{1, 2, 4\}$ represents the set of packets permitted by ACL 1, 2 and 4, and so on.

	Before	After
$\hat{R}^L_{1,5}$	$ACL\{1, 2, 4\}$	$ACL\{1, 2, 3, 4\}$
$\hat{R}^U_{3,5}$	$ACL\{3, 2, 4\}$	$ACL\{1\} \cup ACL\{3, 2, 4\}$

(The algorithms for computing these bounds are given in Section 4.) On one hand, the lower bound from router R1 to router R5 is further constrained by the addition of $ACL3$. Recall

[3] It is completely conceivable that a network could have a routing design such that not all paths can be used, meaning that i and j can be effectively partitioned even when there are still physical paths that connect them.

that for TCP packets with port number 1433 (SQL traffic), $ACL1$ permits only those from hosts in `A1` to hosts in `A5` and $ACL3$ permits only those from `A3` to `A5`. Together, $ACL1$ and $ACL3$ will deny *all* TCP packets with port number 1433 from R1 to R5 as `A1` and `A3` use distinct address ranges. This defeats the purpose of adding the new backup links as they will be totally ineffective for SQL traffic from R1 to R5 under failure scenarios. On the other hand, the upper bound from R3 to R5 is expanded, allowing a portion of multicast traffic to spill out of R3 against the security policy established by $ACL3$. Since the backup links are not used under normal conditions, `ping` and `traceroute` tools would not be of much help in detecting these problems without destructive tests (e.g., by shutting down a primary path).

4 Computing the Reachability Bounds

In this section, we present basic algorithms for computing the simple reachability bound estimators defined by equations (5) and (6). The same algorithms can also be used to calculate other (potentially tighter) reachability bound estimators as long as the approximation is based on adding additional static restrictions to pre-configured packet filters.[4] Section 5 describes such an approximation method.

We assume that there are no packet transformers in the network. We will relax this condition in Section 6.

Lower bound calculation. To compute $\hat{R}^L_{i,j}$, we first prune all the edges $< u, v >\in E$ that cannot be in any path from i to j. This is accomplished by applying the "Articulation Points and Biconnected Components" algorithm for any pair of i and j, which is $O(E+V)$ [1]. After that, $\hat{R}^L_{i,j}$ is simply the intersection of $F_{u,v}$ for t he remaining edges.

Upper bound calculation. While the calculation of $\hat{R}^U_{i,j}$ is not as straightforward, we observe that it closely relates to the classical transitive closure algorithm. Consider $F_{u,v}$ as describing the set of packets that $F_{u,v}$ accepts, with empty set $\emptyset$ and the set of all possible packets denoted as Φ. Our labeling function, $F_{u,v}$, is a map from E to the power set of Φ, $P(\Phi)$, which is closed under the operators $\cup$ and $\cap$. It follows that properties and algorithms in classical literature apply; in particular, solutions to compute transitive closure [1] and classical all-pairs shortest paths algorithms [6]

Below is a dynamic programming formulation (as in [1]) for calculating $\hat{R}^U_{i,j}$, with the recurrence relation $R(i,j)^m = \bigcup_{k \in V} R(i,k) \cap R(k,j)^{m-1}$, where $R(i,j)^m$ represents the set of packets that can go from i to j in up to m hops. The calculation starts from the destination router j and extends the path by one hop with each iteration of the outermost loop, and eventually taking all paths from i to j into consideration.

```
//Computing reachability upper bound matrix column j
1. Initialize $R(i,j)$ to $F_{i,j}$ for all $i$;

2. for ($m = 1$ to $\|V\| - 2$) do
```

[4] Also, the upper bound algorithm can compute the instantaneous reachability if $F_{u,v}(s)$ is known for every edge.

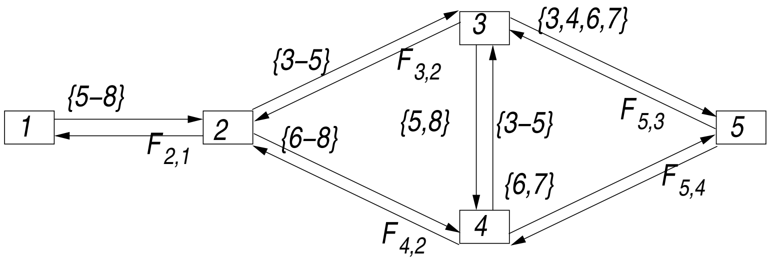


Figure 3: Example network for illustrating execution of algorithm.

Table 2: Example Execution of Basic Algorithm

	$m = 0$	$m = 1$	$m = 2$	$\mathbf{m = 3}$
$R(1,5)$	$\emptyset$	$\emptyset$	$6,7$	$\mathbf{6,7}$
$R(2,5)$	$\emptyset$	$3,4,6,7$	$3,4,6,7$	$\mathbf{3,4,6,7}$
$R(3,5)$	$3,4,6,7$	$3,4,6,7$	$3,4,6,7$	$\mathbf{3,4,6,7}$
$R(4,5)$	$6,7$	$3,4,6,7$	$3,4,6,7$	$\mathbf{3,4,6,7}$

```
3.  for ($i = 1$ to $\|V\|$) do
4.      $R'(i,j) = \emptyset$;
5.      for ($k = 1$ to $\|V\|$) do
6.          if ($< i,k > \in E$)
                then $R'(i,j) = R'(i,j) \bigcup \{F_{i,k} \bigcap R(k,j)\}$;
7.      $R(i,j) = R'(i,j)$;
```

Table 2 shows the intermediate results of $R(i,j)^m$, when running the dynamic program on the example network shown in Figure 3. (This network has a more complex structure than the one defined in Section 2.) For links with packet filters defined, the figure shows the set of packets the filters will pass. For simplicity, packets are represented by integers: $\{5-8\}$ refers to packets 5,6,7, and 8. An uninstantiated $F_{u,v}$ set indicates a filter that passes all packets. The destination router j is set to 5. The last column, when $m = 3$, gives the final result of the reachability from routers 1–4 to router 5.

Algorithm Complexity. The complexity of the illustrative upper bound algorithm above is $O(V^3)$.[5] However, the reachability between *all* pairs of routers can be computed also in $O(V^3)$ via the same techniques used in the Floyd-Warshall method [6] for computing all-pairs shortest paths.

Our reachability analysis framework is targeted at computing the reachability for a network operated and controlled by a single organization, rather than the Internet as a whole. As discussed earlier, the sizes of such networks typically range from 5 to 1,000 routers. With

[5] It should be noted that we have made a simplifying assumption that step 6 has complexity $O(1)$. In real networks, $F_{u,v}$ is often a nontrivial predicate representation of a set of packets. Performing set operations over such representations may incur higher cost than $O(1)$. We are currently investigating this issue.

V bounded like that, the $O(V^3)$ time complexity is very reasonable. It should also be noted that the algorithm will be run mainly as part of a design time tool installed on an off-line system. In that case, timely execution is not a primary concern.

For on-line troubleshooting, the size of V can be reduced by refining the reachability analysis graph model to incorporate the *routing realm* abstraction so that a node in the graph may represent a collection of routing processes with the same external reachability [10]. The details are presented in Section 7.

5 Converting Routing Information into Packet Filters

In this section, we explain how the effects of routing on reachability can be incorporated into our unified framework by adding additional terms to the static packet filters defined in router configuration files. (We will use $F_{u,v}$ to denote the intersection of all packet filters configured over edge $< u, v >$.) These terms restrict the set of packets that can travel from u to v to those packets that the network might route over the link $< u, v >$. The following subsections define the key elements in our model and then describe a four step algorithm for computing the additional terms that must be added to $F_{u,v}$. The algorithm starts with the routes that are explicitly specified in the configuration of the network. It then computes the maximal set of routes that could possibly end up in each router, subject to the network's routing policies. Finally, it uses these maximal sets of routes to compute the additional terms.

We have formally established that our algorithm computes a tighter estimator for the reachability upper bound than the simple one defined by equation (5). The details are presented in the Appendix, which also discusses a limitation of our algorithm in producing a tight estimator for the reachability lower bound.

5.1 Definitions for Modeling Routes and RIBs

A destination subnet is traditionally defined as an address and netmask (Section 2). However, we need the ability to reason about how routers will handle a *set* of destinations. In particular, we will need a means to describe the set of all possible destinations. The conceptual representation of this set as a list of all 2^{32} possible IPv4 destinations is unwieldy to work with in practice, so we must find a more concise notation.

We adopt the representation defined by Cisco, where a set of destination subnets is represented by a list of {address/netmask-range}. For example, {128.2/16-24} represents the set of all destinations whose first bits are 128.2 and whose netmasks are from 16 to 24 bits long; {0/0-32} represents the set of all possible IPv4 destination subnets; and {0/1-32,128/1-32} represents the set of all possible destinations with the default route {0/0} removed. Our algorithms require that union and subtraction be well defined on these sets of destinations, and this is easily proven. Where an algorithm in this paper calls for a destination d, we can use either a single destination subnet or a set of destination subnets interchangeably.

As described in Section 2, each router contains one RIB for each routing process that it runs. A RIB rib is conceptually a function $rib(d)$ that maps destination address d to a list of "routes." A route rt is a tuple with the following fields defined:

•rt.d = set of destination subnets this route applies to

•rt.interfaces = the set of interfaces on the router that packets matching rt.d might be routed out

•rt.next_hop_ip = the set of routers (identified by their IP address) whom packets matching rt.d might be routed towards

•rt.type = {interface,static} original source of this route

•attributes.... = a list of key-value pairs

The Router RIB (i.e., the routing table) of each router maps the complete IP address space onto the set of interfaces according to a longest prefix match. If there is no default route, all packets not matching a more specific route are dropped. We formalize the action of the Router RIB on router u as $I_u(s, d)$, which returns the set of interfaces that packets to destination d should be sent out when in forwarding state s.[6] In this paper, we only consider converged forwarding states — analysis of transient states is beyond the scope of this work. Previous works have shown how network configurations can be statically checked to verify the forwarding state will converge.

For computing the upper and lower bounds on reachability, which predict the network's reachability over all $s \in \mathcal{S}$, we do not need to compute $I_u(s, d)$ but rather the function $\hat{I}_u(d)$ that specifies all the interfaces router u *might potentially* use to forward packets to destination d. That is, $\hat{I}_u(d) = \bigcup_{s \in \mathcal{S}} I_u(s, d)$. Step 2 below shows how we calculate $\hat{I}(d)$ by flooding routes through the network and identifying on each router the interfaces that will be candidates for carrying traffic to d.

While functions like $\hat{I}$, $\mathcal{D}$, and others are router specific, for brevity we will omit the subscript (e.g., using $\hat{I}(d)$ instead of $\hat{I}_u(d)$) when it is clear from the context which router these functions are associated with.

5.2 Step 1: Initializing the RIBs

Initially all RIBs are cleared of all routes. Then the Local RIB on each router is populated with all the routes that are explicitly created on the router by its configuration. For each router r, each interface i on r will be assigned a subnet d by the configuration file: this is represented by setting LocalRIB(d) $=< d, \{i\}, \{r\}, \text{type=interface} >$. Routes manually configured to direct packets to destination d out interface i are represented in the same way. Static routes, which are manually configured routes that direct packets to destination d out whichever interface is used to reach address v, are represented as LocalRIB(d) $=< d, \{\}, \{v\}, \text{type=static} >$. The outgoing interface for a static route is determined in Step 3 using a recursive lookup.

If we are computing the reachability upper bound, the RIBs of all routers external to the network are populated with a single route with destination {0/0-32} — the set of all

[6] $I_u(s, d)$ usually maps d to a single interface, but may contain several interfaces if Equal Cost Multiple Path (ECMP) is in use.

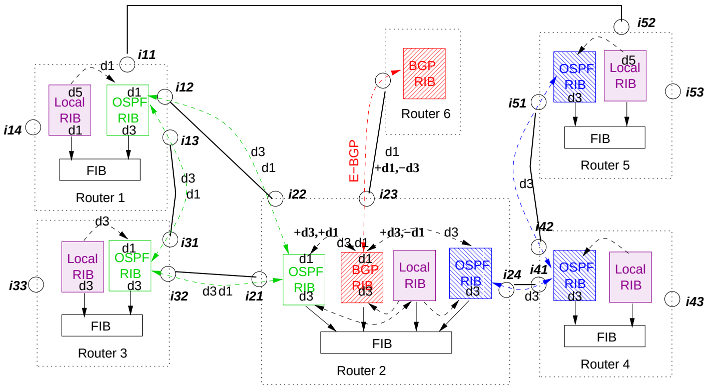


Figure 4: RIB level view of the network illustrating the movement of routes between RIBs. Routes, such as “d1”, “d3”, route filters, such as “+d1,-d3” (“+” as permit, “-” as deny), and flow of routes according to redistribution policies are shown. Following the route flow diagram, we can identify the origins of routes and through which interfaces the routes are imported or exported.

possible destinations. This is a conservative approximation consistent with computing the upper bound on reachability, since whatever destinations the peer does advertise will be covered by {0/0-32}.

If we are computing the lower bound on reachability, the RIBs of all routers external to the network are left empty. This conservative approximation is consistent with computing the lower bound on reachability, since in the worst case the external routers will export no routes whatsoever to our routers, perhaps due to misconfiguration, bugs, crashes, etc.

If the routes the external peers are expected to export are known, the RIBs in our model can be initialized accordingly and the bounds computed on reachability will be correspondingly tighter.

5.3 Step 2: Computing the Potential Set of Routes

In this step, we compute the set of routes that could potentially occupy each RIB by flooding routes from the local RIBs and external RIBs throughout the network. The flooding process is governed by the routing policies between adjacent RIBs that determine which routes are passed, modified, or dropped. At the end of the step, we will have calculated for each RIB rib the maximal set of routes that rib could potentially hold and $\mathcal{D}(rib)$, the set of destinations that rib covers.

As illustrated in Figure 4, the routing design of a network forms a graph $G_{\text{RIB}} = (V_{\text{RIB}}, E_{\text{RIB}}, \mathcal{P})$, where V_{RIB} is the set of RIBs in the network and E_{RIB} describes the adjacencies between RIBs over which routes are imported, exported, and redistributed. $\mathcal{P}$ is the set of routing policies that govern how routes move between RIBs, i.e., for $< x, y > \in E_{\text{RIB}}$, the policy $P_{x,y} \in \mathcal{P}$ determines which routes can move from RIB x to RIB y. Unlike packet filters in $\mathcal{F}$, routing policies in $\mathcal{P}$ can transform the routes they are applied to by changing the route's attributes.

Note that graph G_{RIB} may be partitioned and that adjacencies among RIBs need not follow the physical links of the network. That is, the edge set of the RIB graph E_{RIB} can be different from the edge set E of the physical graph G in Section 3. For example, there is a physical link between routers 1 and 5 in Figure 4, but no RIB adjacency traverses it, or, in the case of networks using internal BGP (IBGP), a single edge in G_{RIB} representing an IBGP adjacency may traverse multiple physical links.

We first define a helper function $\mathsf{push}(rib, rt)$ that takes route rt found in rib and pushes it into all the adjacent RIBs. Lines 2-5 prepare a candidate route for entry into the adjacent RIB. Line 6 applies the routing policy governing which routes can be pushed into the adjacent RIB, potentially altering or dropping the route in the process. Lines 7-8 add the candidate route into the adjacent RIB.

```
push(RIB $x$,Route $rt$) =
1. **Forall** $< x, y > \in E_{\text{RIB}}$
2.     $r$ = router on which RIB $x$ resides
3.     $v$ = router on which RIB $y$ resides
4.     $rt$.interfaces = {interfaces on $v$ where
                   edge $< x, y >$ could arrive}
5.     $rt$.next_hop_ip = $r$
6.     $rt' = P_{x,y}(rt)$
7.     $y(rt'.\text{d}) = y(rt'.\text{d}) \bigcup rt'$
8.     $\mathcal{D}(y) = \mathcal{D}(y) \bigcup rt'.\text{d}$
```

Using $\mathsf{push}(rib, rt)$, we compute rib for each RIB on each router by iterative relaxation: applying $\mathsf{push}()$ to each route in each RIB until there are no changes in the contents of any RIB.

Much of the work in modeling routing lies with the policy $P_{x,y}(rt)$ in step 6. However, these expressions can be directly extracted by parsing the description of the network (e.g., the router configuration files). $P_{x,y}$ must implement the export policy of RIB x and the import policy of y, but has tremendous flexibility given its ability to modify the routes it is applied to. Typical policies seen in real networks include:

– A policy that passes routes to destination subnets 1/8 and 128.2/16 and drops all other routes.

$$\begin{aligned} P_{x,y}(rt) &= rt'.\text{d} = rt.\text{d} - \{1/8 - 8, 128.2/16 - 16\} \\ &\quad \text{if } rt'.\text{d} \neq \emptyset \text{ then return } rt' \text{ else } \emptyset \end{aligned}$$

– A policy that governs the EBGP adjacency between AS1 and AS2, where all routes are passed, but AS1 must prepend its AS number to the route's AS path.

$$P_{x,y}(rt) = rt.\text{as\_path} = \text{concatenate}(AS1, rt.\text{as\_path})$$

– A policy that governs the EBGP adjacency between AS1 and AS2, where routes whose AS path matches a regular expression looking for AS3 are dropped.

$$\begin{aligned} P_{x,y}(rt) &= rt'.\text{as\_path} = rt.\text{as\_path} - /\text{AS3}/ \\ &\quad \text{if } rt.\text{as\_path} \neq \emptyset \text{ then return } rt' \text{ else } \emptyset \end{aligned}$$

The time complexity of this step is $O(|V_{\text{RIB}}| \times l \times |rt| \times 2^{\alpha \times l})$ where l is the length of the longest cycle in the graph G_{RIB} and $|rt|$ is the number of initial routes. The $2^{\alpha \times l}$ factor results from the potential need to split a route into multiple routes each time it is pushed, where α is the fraction of policies that require splitting routes. From our experience so far, α is small for real networks.

5.4 Step 3: Computing $\hat{I}(d)$ for each router

Recall that $\hat{I}(d)$ is a router specific function that returns the set of interfaces out which the corresponding router might forward a packet destined to d. To calculate $\hat{I}(d)$ for a router, we go through all the RIBs on that router looking for routes that cover d, and then union together the interfaces for those routes. $\hat{I}(d)$ is defined recursively, and the base cases are generally routes found in the Local RIB. For readability, we introduce a helper function $\mathsf{ifs}(rt, rib)$ that computes the interfaces to which route rt in rib might direct packets.

$$\hat{I}(d) = \bigcup_{rib\colon\ d \in \mathcal{D}(rib)} \mathsf{ifs}(rt, rib), \quad \text{where } rt = rib(d)$$

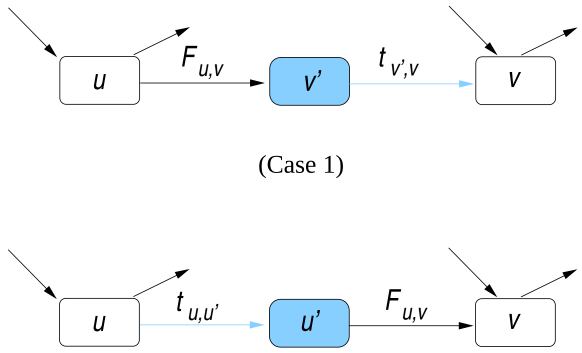
$$\mathsf{ifs}(rt, b) = \begin{cases} rt.\text{interfaces} & \text{if } b \text{ is a LocalRIB, and} \\ & \quad \text{rt.type = interface;} \\ \bigcup_{d \in \text{rt.next\_hop\_ips}} \hat{I}(d) & \text{if } b \text{ is a LocalRIB, and} \\ & \quad \text{rt.type = static;} \\ \bigcup_{d \in \text{rt.next\_hop\_ips}} \hat{I}(d) & \text{if } b \text{ is a BGP-RIB;} \\ & \\ rt.\text{interfaces} & \textit{otherwise.} \end{cases}$$

Case 1 handles the base case of a simple route that forwards packets out a specific interface. Case 2 handles static routes, which require a recursive lookup to determine which interfaces are used to reach the next-hop specified in the route. Case 3 handles BGP sessions, which are carried in TCP session that can traverse multiple routers. A recursive lookup for the address at the other end of the session is required to determine which interfaces the TCP session might arrive on, and thus what outgoing interfaces might be used for routes learned from that session.[7] Case 4 handles all other routing protocols, where the potential outgoing interfaces are those leading to the neighbor routers from which the router imported route d.

5.5 Step 4: Computing Packet Filters that Represent the Effects of Routing

With $\hat{I}(d)$ in hand, we know the set of interfaces out which packets destined to d might be sent. We first compute the inverse mapping of $\hat{I}(d)$, $\hat{D}(i)$, which returns the set of destination

[7] In IBGP it is possible to explicitly set a "third-party" next-hop, but this is unusual. If we see the configuration commands for this, we set $\hat{I}(d)$ to be all interfaces on the router for all d potentially learned over this session.



(Case 1)

(Case 2)

Figure 5: Explicitly modeling packet transformations using $t_{*,*}$ and a virtual node.

subnets that potentially map to interface i, i.e., $\hat{D}(i) = \{d \mid i \in \hat{I}(d)\}$. Using $\hat{D}(i)$ we map the routing table information to packet filters as follows:

Let r denote the router under consideration. For all interfaces i on r and for all routers v that are directly connected to r via interface i, add the following clauses to $F_{r,v}$.

$$F_{r,v} = F_{r,v} \cap \{p \mid \text{p.dst\_addr} \in \hat{D}(i)\} \tag{7}$$

This filter will pass any packet going to a destination that r might possibly route out the link to v, and drop all the packets that r would never route via v.

6 Handling Packet Transforms

In this section, we refine the basic algorithm presented in Section 4 so that it will work with networks that include packet transforming filters.[8]

We have found that this refinement can be accomplished without changing the fundamental structure of the basic algorithm. Specifically, we separate the packet transforming parts from these filters and introduce virtual components to represent them explicitly in the reachability analysis graph G. This is illustrated in Figure 5. There are two cases: (1) the packet transformation t is applied after the packet filtering (e.g., ToS remarking), and (2) the transformation t is applied before the packet filtering (e.g., NAT). In ether case, a virtual node-edge pair is introduced to model the separate processing stage. Each virtual edge is labeled with a t function representing a packet transform.

We have discovered that packet transforms may have two undesirable properties that can complicate the reachability analysis. First, a transform might not be one-to-one. For example, in the case of ToS remarking, multiple ToS values may be mapped into one single

[8] For networks containing packet transformers, the set of packets that a destination can receive may be different than the set a source can send to that destination. For this paper, we calculate reachability as the set of packets the source can send, although our results can be extended to also calculate a set describing what those packets might look like on arrival at the destination.

ToS value. Also, in the case of NAT, one external address pool is typically reused for many hosts as long as no two hosts use the same external address and port number at the same time.

Second, a transform may not even be a deterministic function. In some modes of NAT, a packet is not always transformed into the same packet; the source address the packet gets depends on the current availability of the address pool. To address these problems, we define a generalized inverse function of t, over an arbitrary packet set F, as: $t^{-1}(F) = \bigcup_{q \in F} \{p \mid q \in t(p)\}$, which returns the set of all possible packets that can be transformed using t to a packet in F.

Using the inverse transform function, we have refined the basic algorithm to handle packet transforms. Specifically, only steps 1 and 6 of the original algorithm need to be changed.

```
1'.For all $i$, initialize $R(i,j)$ as follows:
    to $F_{i,j}$, if $< i,j >$ is filter
    to set of all packets, if $< i,j >$ is transformer
    to $\emptyset$, if $< i,j > \notin E$

6'.        if $(< i,k > \in E$ and $< i,k >$ is transformer)
             then $R'(i,j) = R'(i,j) \bigcup t_{i,k}^{-1}(R(k,j))$;
           else if $(< i,k > \in E$ and $< i,k >$ is filter)
             then $R'(i,j) = R'(i,j) \bigcup \{F_{i,k} \bigcap R(k,j)\}$;
```

The intuition behind the new clause in 6' is that if the set of packets described by $R(k,j)$ can reach from k to j, then only those packets arriving at i that $t_{i,k}$ transforms into a packet in the set $R(k,j)$ will be able to reach from i to j. To find this set of packets that t will transform into $R(k,j)$, we calculate $t_{i,k}^{-1}(R(k,j))$.

The complexity of the new algorithm is still $O(V^3)$. It should be noted that the algorithm requires two additional elements to be complete: (i) an efficient method to compute the inverse function, and (ii) a condition to throw out paths with loops because a looping path containing a packet transform edge may alter the outcome. Luckily, the inverse function for commonly used transforms, such as NAT and ToS remarking, are very simple — though in general the inverse of other transforms may be more complicated. For brevity, the details of (ii) are omitted.

Let's revisit the example network in Figure 3 to illustrate the steps of the refined algorithm. Suppose node 1 now uses a leading packet transform: $\{1,2\} \rightarrow \{5,6\}$, meaning that packets 1 and 2 each will be mapped into either packet 5 or 6 before processed by node 1's packet filter. The new reachability analysis graph becomes Figure 6 and the execution steps of the refined algorithm are shown in Table 3. At the last step ($m = 4$), $R(1,5)$ is changed due to the transform.

Our framework currently requires that packet transforms be maps over sets of packets. They cannot test a property of a packet and behave one way if the property is true and another way if the property is false. In particular, some networks include functionality called a "stateful firewall". These are like a NAT, but only create the mapping when a

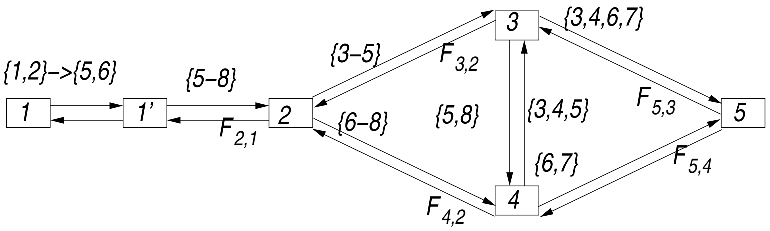


Figure 6: Packet transform example

Table 3: Example Execution of Refined Algorithm

	$m = 0$	$m = 1$	$m = 2$	$m = 3$	$\mathbf{m = 4}$
$R(1,5)$	$\emptyset$	$\emptyset$	$6,7$	$6,7$	$\mathbf{1,2,6,7}$
$R(2,5)$	$\emptyset$	$3,4,6,7$	$3,4,6,7$	$3,4,6,7$	$\mathbf{3,4,6,7}$
$R(3,5)$	$3,4,6,7$	$3,4,6,7$	$3,4,6,7$	$3,4,6,7$	$\mathbf{3,4,6,7}$
$R(4,5)$	$6,7$	$3,4,6,7$	$3,4,6,7$	$3,4,6,7$	$\mathbf{3,4,6,7}$

packet traverses from the inside of the firewall to the outside. Since our framework currently has no notion of whether a packet has already been sent through the stateful firewall from inside to outside, we cannot directly model the reachability the stateful firewall provides. However, we can calculate the reachability assuming a packet has traversed the firewall, in which case the firewall functions as a NAT described above, and again assuming no packet has traversed it, in which case the firewall functions as a block.

7 Improving Scalability with Routing Realm Abstraction

As discussed in Section 4, calculating the router level reachability matrix with our approach has time complexity of at least $O(V^3)$, where V is the number of routers in the network. Although polynomial, such complexity may still pose a challenge for analyzing large networks that have thousands of routers. To make our approach more scalable, we refine the reachability analysis graph model by incorporating the *routing realm* abstraction so that a node in the graph may represent a collection of RIBs and routers.

A routing realm represents a set of routing processes that have exactly the same set of routable subnets in their RIBs. Algorithmically, a routing realm groups together all the routing processes that are adjacent to each other so long as the adjacencies are free of packet filters and route filters, regardless of the router boundaries. More specifically, routing realms can be constructed by analyzing router configuration files as follows. First, select from the network a routing process that has not yet been assigned a routing realm and assign to it a new unique realm number. Then locate all the adjacencies of that process which are free of packet filters and route filters, and compute the transitive closure within the AS boundary to find the set of routers and routing processes belonging to the new routing realm. The process is then repeated until all routing processes have been assigned to a routing realm.

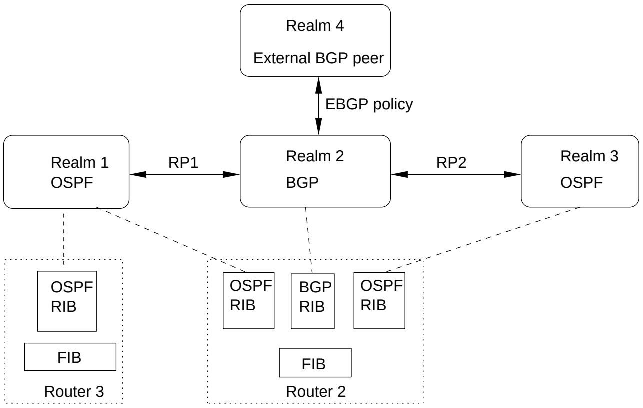


Figure 7: A depiction of routing realm abstraction for the network in Figure 1.

The concept of a routing realm is related to the "routing instance" defined as part of our prior work on reverse engineering of routing design [10] — the primary difference is that a routing realm stops expanding when it hits any policy applied to a link or adjacency, while a routing instance grows to the transitive closure of routing process adjacencies. A single routing instance can contain multiple routing realms, and from observations of real production networks this is not uncommon.

Figure 7 shows the result of applying the routing realm abstraction to the example network in Figure 1. All internal routers in the network have been removed and replaced with the routing realms of which their routing processes are part. The solid arrows between realms denote where route redistribution occurs, and can be annotated with a key to the redistribution policies (e.g., "RP1" in the figure). A dotted box connecting to a routing realm represents a typical member router in that realm. In the figure, "Router 2" is illustrated as an example member of three different routing realms.

Because of the way routing realms are defined and constructed, the potential reachability between two routers in the same routing realm is equal to the set of all routable packets at the routers, i.e., {packet p | p.dst is part of a routable subnet in routers' RIB}. And the potential reachability between two nodes that are in two different routing realms is the same as the potential reachability between their routing realms. Therefore, by incorporating the routing realm abstraction, our approach scales much better to large numbers of routers without losing the ability to calculate reachability at the router level.

8 Reachability Analysis in Larger Context

In this section, we describe how static reachability analysis relates to our larger goal of understanding and improving the design of IP networks and routing policies. We also discuss the limitations of static analysis and how to move beyond them.

8.1 Understanding and Improving Routing Design

Our work on static reachability analysis contributes to our broader research agenda of improving routing design and network robustness. Today, routing design is largely a complex "art" mastered by an increasingly overwhelmed community of highly-skilled human operators. We aim to uncover the fundamental abstractions, such as the reachability bounds $R^U_{i,j}$ and $R^L_{i,j}$, that can be used to validate, evaluate, and even generate the routing design for a network. Our analysis framework opens several avenues for ongoing work:

Verification of network design goals: A network has some (explicit or implicit) design goals for providing reachability between certain parties under certain conditions. For example, a network may need to ensure that two business competitors (customers A and B) can never reach each other under any circumstances. This property can be checked directly by ensuring that $R^U_{i,j}$ is empty for all i in customer A's network and j in B's network, and vice versa. Alternatively, a network may need to ensure that a customer can reach a data center; this can be assured by analyzing the lower bound on reachability. Repeating the reachability analysis on the subgraphs formed after link and node deletions can test that network reachability persists under certain failure modes.

Design patterns and best common practices: The same reachability goals can be satisfied by a wide variety of different routing designs. Our concise representation of reachability provides an appealing way to characterize and compare routing designs and identify common ways of configuring a network to satisfy the goals. Using router configuration data for several networks, we plan to identify common kinds of reachability goals and the combinations of routing protocols, routing policies, and packet filters used to achieve them. We also plan to explore the trade-offs between using routing policies and packet filters in constraining reachability, and create guidelines for selecting one mechanism over the other. In particular, we hope to understand the motivations for applying packet filters in the interior of routing domains, rather than simply at the periphery.

Influence of dynamic routing information: Our upper and lower bounds ($R^U_{i,j}$ and $R^L_{i,j}$) define an "envelope" that constrains the influence of dynamic information, such as topology changes or routes learned from neighboring domains, on network reachability. We plan to analyze existing networks in terms of the range between the upper and lower bounds. The gap between the upper and lower bounds may reflect the purpose of the network—to provide broad reachability for many client domains to the entire Internet or to to provide narrow reachability for client domains to specific network services. Alternatively, a wide range might imply the need for more protective packet and route filtering, whereas a narrow range may overly constrain the ability of the network to adapt to dynamic changes.

For each of these avenues for future work, our reachability analysis offers a general and concise way to analyze and compare routing designs at a level of abstraction well above the low-level details of router configuration commands and specific routing protocols.

8.2 Moving Beyond Static Analysis

Although static analysis provides significant insights, dynamic information determines where a network actually operates in the space between the lower and upper bounds on reachability.

Our static analysis can be extended by incorporating measurements of the dynamic state of the network and the routes learned from neighboring domains:

Dynamic network state: The configuration state defines the IP links and routing protocol adjacencies that *could* exist, without indicating whether they *do* exist at any given time. Various kinds of measurement data can provide the missing information. The up/down status of links and sessions can be tracked via the Simple Network Management Protocol (SNMP) or vendor-specific "syslog" data. In addition, a routing monitor [15] can continuously track the topology (routers and links) and configurable parameters (e.g., OSPF link weights) within each routing instance.

Routing information from neighbors: Similarly, static analysis considers the route advertisements that *could* come across links and sessions to neighboring domains, rather than the ones that *are* available at any given time. The set of routes announced by a neighboring domain could be gleaned through route monitoring or periodic dumps of the Routing Information Base (RIB) at the edge routers. In addition to identifying which prefixes are advertised, the RIB data would identify the route attributes (such as AS path in BGP) that might affect how the receiving router modifies or selects routes. In addition, the RIB would indicate whether the neighbor advertises subnets of a given prefix that would have preference over the supernet in "longest prefix match" forwarding of IP packets.

9 Related Work

Many "ping" and "traceroute" tools have been developed to help troubleshoot reachability problems in a live network. However, they are limited to the checking the instantaneous reachability for the particular type of probe packets they generate. There has been significant progress [19, 12, 5] in understanding the behavior of operating networks by measuring the routing protocols and establishing the root cause of changes. Our approach does not attempt to describe the detailed behavior of the routing protocols, and it applies to packet filters and packet transformation as well as routing.

Bush and Griffin [2] formulate and derive sufficient conditions for the connectivity (reachability) constraints of Virtual Private Routed Networks (VPRNs). Our work is complementary, but broader in scope in that we frame and tackle the general problem of reachability.

10 Conclusions

This paper rigorously formulates the challenging problem of computing the reachability an IP network provides and describes a framework that can be used to calculate it.

The framework provides a unified way for jointly reasoning about the effects the three very different mechanisms of packet filters, routing policy, and packet transformations have on the network's reachability.

Finally, we show how the framework can be applied to a static description of the network's definition, allowing it to be applied either during the network design process or to a deployed network. Our technique for static analysis of network reachability is valuable for verifying the intent of the network designer, troubleshooting reachability problems, and performing "what-if" analysis of failure scenarios.

Now that we have this formal framework, our future work is focused on experimental evaluation of the algorithms on a set of networks. For example, our framework can be extended for computing finer-grain reachability bounds, such as ones that consider only a subset of packets (e.g., those carrying TCP port 1443 traffic) or a subset of paths (e.g., excluding certain links). Such extensions should be guided by analysis of a large number of production networks.

11 Appendix - Justification for Converting Routing into Packet Filters

11.1 Evaluating the Estimator as an Upper Bound

In this subsection we will prove that the method described in Section 5 for modeling the effects of routing by adding additional terms to restrict $F_{u,v}$ results in a better estimator of the upper bound on reachability, $R^U_{i,j}$, than the simple estimator shown in Section 3.2.3. To avoid confusion, we will in this discussion refer to the new and simple estimators as $\hat{R}^U_{i,j}$ and $\hat{R}^{U'}_{i,j}$, respectively. The new estimator being analyzed can be expressed as follows

$$\hat{R}^U_{i,j} = \bigcup_{\pi \in \mathcal{P}(i,j)} \quad \bigcap_{<u,v> \in \pi} \hat{F}^U_{u,v} \tag{8}$$

where $\hat{F}^U_{u,v}$ is the estimated effective filter on edge $< u, v >$, calculated using the method described in Section 5.

First we have the following theorem about the RIB calculation:

Lemma 1 (RIB Upper Bound Lemma) *Our method guarantees that at the end of calculation the routes populated into a given RIB b, denoted $\hat{b}$, is a superset of $b(s)$ — the routes that b will hold in any given network state s. Formally,*

$$\bigcup_{\forall s \in \mathcal{S}} b(s) \subseteq \hat{b} \tag{9}$$

Proof: Let rt represent an arbitrary route. Based on the definition of $\mathsf{push}(b, rt)$ in Section 5.3, we next inductively prove the following:

$$\forall s \in \mathcal{S}, rt \in b(s) \Rightarrow rt \in \hat{b} \tag{10}$$

Consider the G_{RIB} graph defined in Section 5.3. $rt \in b(s)$ means that rt is either originated from b or it is propagated there from another RIB. In the former case, it is straightforward to show that $rt \in \hat{b}$.

Now let's consider the latter case. rt may propagate to b via multiple paths. for example, if b is running a link state protocol, propagation will occur over every adjacency to b where the link is up and there is no special policy to filter the route. Suppose we pick one path along which the route rt propagates, denoted $b_0 \to b_1 \to b_2 \to ... \to b_n$ where b_0 is the RIB where rt is originated and $b_n = b$. We will inductively prove that $\hat{b}_i$ contains rt for all i less than or equal to n.

When $i = 0$, then rt is originated from b_i. Such routes are statically configured, meaning that rt will be found in b_i independent of the state s. The manner in which the route is configured is protocol dependent (e.g., using static, network, or directly connected commands for the OSPF and ISIS protocols). Based on our definition, rt is in $\hat{b}_i$.

Assuming when $i = k < n$, we have $rt \in \hat{b}_i$, we need to prove that $rt \in \hat{b}_{k+1}$. Since rt is populated from b_k to b_{k+1}, we have the following: (1) there must be an edge in the routing process graph from b_k to b_{k+1}, and (2) the policy on this edge must not filter out rt. Based on our push function, this edge will be used by the push function at some point for all the routes in $\hat{b}_k$. Based on our assumption that $rt \in \hat{b}_k$ and based on our definition of push, rt will be pushed from b_k to b_{k+1}. In fact, the push routine always merges new routes into a RIB and never deletes a route, i.e., $\hat{b}_{k+1} \supseteq \bigcup_{0 \leq i \leq k} \hat{b}_i$. Therefore, $rt \in \hat{b}_{k+1}$.

■

Consequently, we have the following interface upper bound lemma.

Lemma 2 (RIB Interface Upper Bound Lemma) *Our method guarantees that the set of interfaces calculated for any given destination d at each router is a superset of all potential interfaces used by the router to forward packet to destination d. Formally,*

$$\bigcup_{s \in \mathcal{S}} I(s,d) \subseteq \hat{I}(d) \tag{11}$$

Proof: If an interface i is a member of $I(s,d)$, then there must be a route rt in the FIB for state s, such that rt.destination $= d$ and rt.interface $= i$. Since rt must come from one of the RIBs – say b – at the router, based on the above theorem, rt is in $\hat{b}$. Hence i is in $\hat{b}(d)$, so it is in $\hat{I}(d)$.

■

With the above two Lemmas, we can prove the following filter estimator upper bound theorem.

Theorem 1 (Filter Estimator Upper Bound Theorem) *Our method guarantees that the calculated filter estimator $\hat{F}^U_{u,v}$ for each edge is a superset of all potential packets that can pass through that edge with the network in any one of the forwarding states. Formally,*

$$\bigcup_{s \in \mathcal{S}} F_{u,v}(s) \subseteq \hat{F}^U_{u,v} \tag{12}$$

Proof: For any packet p, we will show that if there exists a forwarding state s such that $p \in F_{u,v}(s)$, then $p \in \hat{F}^U_{u,v}$.

$p \in F_{u,v}(s)$ implies(1) p is allowed by all the static packet filters and (2) that for at least one RIB b in router u, there exists a route $r \in b(s)$ such that $p.dest \in r.dest$ and

$I(s, r.dest) =< u, v >$. Based on the RIB Upper Bound Lemma, we have $b(s) \subseteq \hat{(b)}$; so $r \in \hat{b}$. Based on the RIB Interface Upper Bound Lemma, we have $I(s, r.dest) \subseteq \hat{I}(r.dest)$; so $< u, v > \in \hat{I}(r.dest)$. Since $p.dest \in r.dest$ and p is allowed by all the static packet filters, we can conclude that $p \in \hat{F}^U_{u,v}$.

■

We also have:

Lemma 3 (Reachability Upper Bound Lemma)

$$\bigcup_{\pi \in \mathcal{P}(i,j)} \bigcap_{<u,v> \in \pi} \bigcup_{s \in \mathcal{S}} F_{u,v}(s) \supseteq \bigcup_{s \in \mathcal{S}} \bigcup_{\pi \in \mathcal{P}(i,j)} \bigcap_{<u,v> \in \pi} F_{u,v}(s) \tag{13}$$

Proof: We prove the formula using set operations alone. First we prove for each $\pi \in \mathcal{P}(i, j)$:

$$\bigcap_{<u,v> \in \pi} \bigcup_{s \in \mathcal{S}} F_{u,v}(s) \supseteq \bigcup_{s \in \mathcal{S}} \bigcap_{<u,v> \in \pi} F_{u,v}(s) \tag{14}$$

Assume $\pi =< u_1, u_2, \cdots, u_n, u_{n+1} >$, with corresponding filter annotations $< F_1, F_2, \cdots, F_n >$ on the n edges, and $\mathcal{S} = \{s_1, s_2, \cdots, s_m\}$, then we have

$$\begin{aligned}
\bigcap_{<u,v> \in \pi} \bigcup_{s \in \mathcal{S}} F_{u,v}(s) &= \bigcap_{<u,v> \in \pi} (F_{u,v}(s_1) \cup F_{u,v}(s_2) \cup \cdots \cup F_{u,v}(s_m)) && (15)\\
&= (F_1(s_1) \cup F_1(s_2) \cup \cdots F_1(s_m)) \cap \\
&\quad\ (F_2(s_1) \cup F_2(s_2) \cup \cdots F_2(s_m)) \cap \\
&\quad\ \cdots \\
&\quad\ (F_n(s_1) \cup F_n(s_2) \cdots F_n(s_m)) && (16)\\
&\supseteq (F_1(s_1) \cap F_2(s_1) \cap \cdots F_n(s_1)) \cup \\
&\quad\ (F_1(s_2) \cap F_2(s_2) \cap \cdots F_n(s_2)) \cup \\
&\quad\ \cdots \\
&\quad\ (F_1(s_m) \cap F_2(s_m) \cap \cdots F_n(s_m)) && (17)\\
&= \bigcup_{s \in \mathcal{S}} \bigcap_{<u,v> \in \pi} F_{u,v}(s) && (18)
\end{aligned}$$

Taking the union of both sides of equation (18) over all paths in the topology, we have

$$\bigcup_{\pi \in \mathcal{P}(i,j)} \bigcap_{<u,v> \in \pi} \bigcup_{s \in \mathcal{S}} F_{u,v}(s) \supseteq \bigcup_{\pi \in \mathcal{P}(i,j)} \bigcup_{s \in \mathcal{S}} \bigcap_{<u,v> \in \pi} F_{u,v}(s) \tag{19}$$

Rearranging the unions on the right-hand side, we arrive at the statement of the theorem, completing the proof.

$$\bigcup_{\pi \in \mathcal{P}(i,j)} \bigcap_{<u,v> \in \pi} \bigcup_{s \in \mathcal{S}} F_{u,v}(s) \supseteq \bigcup_{s \in \mathcal{S}} \bigcup_{\pi \in \mathcal{P}(i,j)} \bigcap_{<u,v> \in \pi} F_{u,v}(s) \tag{20}$$

■

Now we are ready to present the main theorem.

Theorem 2 (Reachability Upper Bound Theorem) *The reachability upper bound calculated using our route model is a upper bound of the reachability upper bound. Formally,*

$$R^U_{i,j} \subseteq \hat{R}^U_{i,j} \tag{21}$$

Proof:

Using the above Lemma and Theorem 1, the proof is straightforward as follows:

$$\begin{aligned} R^U_{i,j} &= \bigcup_{s \in \mathcal{S}} \bigcup_{\pi \in \mathcal{P}(i,j)} \bigcap_{<u,v> \in \pi} F_{u,v}(s) && (22)\\ &\subseteq \bigcup_{\pi \in \mathcal{P}(i,j)} \bigcap_{<u,v> \in \pi} \bigcup_{s \in \mathcal{S}} F_{u,v}(s) && (23)\\ &\subseteq \bigcup_{\pi \in \mathcal{P}(i,j)} \bigcap_{<u,v> \in \pi} \hat{F}^U_{u,v} && (24)\\ &= \hat{R}^U_{i,j} && (25) \end{aligned}$$

■

The result of this exercise has been to show that the estimator $\hat{R}^U_{i,j}$ we compute using the method for modeling routing protocols given in Section 5 is a conservative estimator for the upper bound $R^U_{i,j}$. We can then use this estimator to verify the isolation provided by the network and know that, while there may be false positives, there will be no false negatives (i.e., a packet that managed to get between two routers when the estimator said the packet could not get through).

The next theorem shows that the estimator using routing function is also a tighter bound on $R^U_{i,j}$ than the simple estimator, since the estimator that uses routing information contains more terms that limit the sets $F_{u,v}$ than the simple estimator that uses only packet filters to define $F_{u,v}$,

Theorem 3 (Upper Bound Tightness Theorem) *The estimator $\hat{R}^U_{i,j}$ of the reachability upper bound obtained by modeling routing information is a better upper bound than the estimator $\hat{R}^{U'}_{i,j}$ that ignores routing and uses only packet filters in defining $F_{u,v}$. Formally,*

$$\hat{R}^U_{i,j} \subseteq \hat{R}^{U'}_{i,j} \tag{26}$$

Proof: For any packet that is in $\hat{R}^U_{i,j}$, suppose the packet traverses a path π between i and j. For any link $< u, v >$ on π, modeling the effects of routing adds more routing-based filters to the link. That is, if we removed the routing based filters, the packet would still be able to traverse over the link $< u, v >$, and hence the entire path π; therefore, that packet will be in $\hat{R}^{U'}_{i,j}$.

■

11.2 Future Work in Improving the Estimator for the Lower Bound on Reachability

Two issues remain in determining the best estimator for the lower bound on reachability.

The first issue is the best interpretation for the set $\mathcal{S}$ of all forwarding states, over which the Lower Bound is defined. There are three candidates: (1) $\mathcal{S}_\infty$: the set of all forwarding states (including even states which the routing design of the network cannot ever put the network into); (2) $\mathcal{S}_f$: the set of all feasible forwarding states (all of the states the routing design can conceivably reach, although it might require extreme preconditions, such as the failure of all but a specific handful of links); and (3) $\mathcal{S}_i$: the set of "interesting" forwarding states the network will reach under all scenarios the network operator decides are plausible or worth considering.

We have a proof for properties of the lower bound estimator if we use set $\mathcal{S}_\infty$ (shown in the next subsection), but this proof predicts the lower bound to be larger than it really is (as it includes forwarding states that the network cannot reach). Ideally, we would be able to succinctly represent set $\mathcal{S}_i$.

The second issue is how the push() operation should be defined when computing the lower bound. As explained in Section 5.3, we compute the maximal set of routes that can be present in a RIB by taking the union of all routes pushed into the RIB. Consider, however, a case where a route is pushed into a RIB over interface i_1, but lists interface i_2 as the next_hop (or maybe the set $\{i_1, i_2\}$) for that route. This case creates a paradox, as there is potentially a scenario under which interface i_1 fails, and thus the route is never learned, but yet the route is increasing the computed lower bound reachability. It is arguable that when computing the lower bound, an additional "scrubbing" step should be added after all pushes are completed. The purpose of the scrubbing would be to remove from the routes any next_hop interface that creates the paradox outlined above. However, this scrubbing seems to imply that only routes that a router learns over all interfaces can be relied on when computing the lower bound, and that requirement seems too strict given our ultimate desire to compute the lower bound under set $\mathcal{S}_i$.

11.3 Evaluating the Estimator for the Lower Bound on Reachability

Unfortunately, the method for estimating the Lower Bound as outlined in Section 5 results in an estimator that contains packets that are not in $R^L_{i,j}$. That is, $R^L_{i,j} \subseteq \hat{R}^L_{i,j}$. This is suboptimal, as described in Section 3.2.3. However, one way of using the lower bound is for robustness testing in the network design, i.e., to check that critical traffic will survive all failures as long as the network is not partitioned. While traffic that shows up in the estimated reachability $\hat{R}^L_{i,j}$ might not be preserved across all (non-partitioning) forwarding states, if the critical traffic is not even in the lower bound estimation we calculated, then the traffic will definitely not be in the real reachability lower bound. So, based on this usage, the lower bound estimation may still be a useful tool for reachability verification.

Theorem 4 (Reachability Lower Bound Theorem) *The reachability lower bound calculated using our model in Section 5 is an upper bound of the reachability lower bound.*

Formally,

$$R^L_{i,j} \subseteq \hat{R}^L_{i,j} \tag{27}$$

Proof: First, $\hat{R}^L_{i,j}$ can be formally expressed as follows:

$$\hat{R}^L_{i,j} = \bigcap_{\pi \in \mathcal{P}(i,j)} \quad \bigcap_{<u,v> \in \pi} \hat{F}^L_{u,v} \tag{28}$$

where $\hat{F}^L_{u,v}$ is the estimated effective filter on edge $< u, v >$, calculated using the method described in Section 5.

For any physical path π between i, j, we can construct a network state s where π is the only path between i, j by removing all the other links in the network. Let us assume $s \in \mathcal{S}$. (It is certainly the case that $s \in \mathcal{S}_\infty$.)

Let X be the union of all these s for each path $\pi \in \mathcal{P}(i, j)$. Now X is a subset of the the set of all forwarding states $\mathcal{S}$, so the intersection of $R_{i,j}(s), s \in X$ is a superset of the intersection of $R_{i,j}(s), s \in \mathcal{S}$. Formally,

$$\bigcap_{s \in \mathcal{S}} R_{i,j}(s) \subseteq \bigcap_{s \in X} R_{i,j}(s) \tag{29}$$

Because the way we calculate routes (see the RIB and Interface Upper Bound Lemmas), the packet filters $\hat{F}^L_{u,v}$ used in computing $\hat{R}^L_{i,j}$ allow even more paths than those in X. That is, the packet filters constructed to model the maximal possible set of routes in each RIB is less strict than the FIBs used in the forwarding states in X. Hence, we have

$$\bigcup_{s \in X} R_{i,j}(s) \subseteq \hat{R}^L_{i,j} \tag{30}$$

Combining equations (29) and (30), we have

$$\bigcap_{s \in \mathcal{S}} R_{i,j}(s) \subseteq \bigcap_{s \in X} R_{i,j}(s) \subseteq \bigcup_{s \in X} R_{i,j}(s) \subseteq \hat{R}^L_{i,j} \tag{31}$$

■

12 Glossary

Terms used in this paper:

i, j, u, v Routers in the network

$< u, v >$ An edge in the physical topology of the network between routers u and v.

s A network's forwarding state — conceptually a snapshot of the FIB at each router in the network.

$\mathcal{S}$ The set of all possible forwarding states that the network can possibly enter, as it responds to any imaginable set of external advertisements, link failures, etc. Alternate definitions of $\mathcal{S}$ are explored in Section 11.2.

$R_{i,j}(s)$ The instantaneous reachability between router i and router j when the network is in forwarding state s.

$R^U_{i,j}$ The upper bound of the reachability between router i and router j. Packets not in this set cannot possibly get from i, to j, regardless of the network's forwarding state.

$R^L_{i,j}$ The lower bound of the reachability between router i and router j. Packets not in this set will be prevented from getting from i, to j in some forwarding states..

$\hat{R}_{i,j}$ An estimator to approximate the reachability of the network. Two different estimators are described in this paper. The simple estimators that ignore routing (Section 3.2.3) and estimators that model the effects of routing (Section 5).

References

[1] A.V. Aho, J.E. Hopcroft, and J.D. Ullman. *The Design and Analysis of Computer Algorithms.* Addison-Wesley, 1974.

[2] Randy Bush and Timothy G. Griffin. Integrity for virtual private routed networks. In *Proc. IEEE INFOCOM*, 2003.

[3] Don Caldwell, Anna Gilbert, Joel Gottlieb, Albert Greenberg, Gisli Hjalmtysson, and Jennifer Rexford. The cutting EDGE of IP router configuration. In *Proc. ACM SIGCOMM Workshop on Hot Topics in Networking*, November 2003.

[4] R. Callon. *RFC 1195 - Use of OSI IS-IS for routing in TCP/IP and dual environments*, 1990.

[5] D.-F. Chang, R. Govindan, and J. Heidemann. The temproal and topological characteristics of BGP path changes. In *Proc. International Conference on Network Protocols*, November 2003.

[6] Thomas H. Cormen, Charles E. Leiserson, and Ronald L. Rivest. *Introduction to Algorithms.* MIT Press (McGraw-Hill), 1990.

[7] Nick Feamster and Hari Balakrishnan. Verifying the correctness of wide-area Internet routing. Technical Report MIT-LCS-TR-948, Massachusetts Institute of Technology, May 2004.

[8] Anja Feldmann and Jennifer Rexford. IP network configuration for intradomain traffic engineering. *IEEE Network Magazine*, pages 46–57, September/October 2001.

[9] C. Hedrick. *RFC 1058 - Routing Information Protocol*, 1988.

[10] D. Maltz, G. Xie, J. Zhan, H. Zhang, G. Hjalmtysson, and A. Greenberg. Routing design in operational networks: A look from the inside. In *Proc. ACM SIGCOMM*, August 2004.

[11] J. Moy. *RFC 2178 - OSPF Version 2*, 1997.

[12] Packet Design, Inc. Route Explorer.
http://www.packetdesign.com/products/products.htm.

[13] Y. Rekhter and T. Li. *RFC 1771 - A Border Gateway Protocol 4 (BGP-4)*, 1995.

[14] E. Rosen and Y. Rekhter. *RFC 2547 - BGP/MPLS VPNs*, March 1999.

[15] Aman Shaikh and Albert Greenberg. OSPF monitoring: Architecture, design, and deployment experience. In *Proc. USENIX/ACM NSDI*, March 2004.

[16] P. Srisuresh and M. Holdrege. *IP Network Address Translator (NAT) Terminology and Considerations*. Internet Engineering Task Force, Aug 1999. RFC 2663.

[17] Cisco Systems. BGP Best Path Selection Algorithm. http://www.cisco.com/warp/public/459/25.shtml.

[18] Cisco Systems. *Enhanced IGRP*. http://www.cisco.com/univercd/cc/td/doc/-cisintwk/ito_doc/en_igrp.htm.

[19] Renata Teixeira and Jennifer Rexford. A measurement framework for pin-pointing routing changes. In *ACM SIGCOMM Workshop on Network Troubleshooting*, September 2004.