Policies in Routing

Cheng Tien Ee

Electrical Engineering and Computer Sciences
University of California at Berkeley

Technical Report No. UCB/EECS-2007-66
http://www.eecs.berkeley.edu/Pubs/TechRpts/2007/EECS-2007-66.html

May 18, 2007

Acknowledgement

I would like to thank Scott Shenker and Ion Stoica for their patience and guidance, and also Ruzena Bajcsy who first took me under her wing.

I would also like to acknowledge the assistance of those who worked with me at some point in time or another. Brent Chun, Sylvia Ratnasamy, Pavlin Radoslavov, Lakshminarayanan Subramanian, Vijay Ramachandran, Byung-Gon Chun. I appreciate all the effort put in to help me towards my goal.

These five years have been made more bearable by a group of friends. Due to space constraints I am unable to name them all.

Last but not least, I thank my family for their understanding and putting up with my long periods of absence and lack of updates.

Policies in Routing

by

Cheng Tien Ee

B.Eng. (National University of Singapore) 2002
M.S. (University of California at Berkeley) 2005

A dissertation submitted in partial satisfaction of the
requirements for the degree of
Doctor of Philosophy

in

Computer Science

in the

GRADUATE DIVISION
of the
UNIVERSITY OF CALIFORNIA, BERKELEY

Committee in charge:
Professor Scott Shenker, Chair
Professor Ion Stoica
Professor Rhonda Righter

Spring 2007

Policies in Routing

Abstract

Policies in Routing

by

Cheng Tien Ee

Doctor of Philosophy in Computer Science

University of California, Berkeley

Professor Scott Shenker, Chair

The Internet began as a network under the control of a single organization, ARPA. The initial goals were to provide basic connectivity between end-hosts even in the event of failures, and applications running over the network, such as email, could utilize any path through the network as long as the destination could eventually be reached. The existence of a single administrative entity for the entire network and the lack of a need to distinguish between possible paths as well as packets in the network meant that a simple routing protocol is sufficient to provide connectivity. With its subsequent break up and distributed management, the Internet today needs to distinguish between different kinds of data and control information. For instance, service providers have to decide on which neighboring provider to transit their traffic, or to filter certain packets due to certain accessibility constraints. The type of policies involved vary depending on whether they are applied on an inter or intra-domain basis, and hence also the resulting problems that arise.

We address two issues in this dissertation. Firstly, the lack of visibility and independent implementing of policies in inter-domain routing can result in policy disputes causing routing to oscillate forever. We propose the Precedence Solution that enforces shortest path routing only when oscillations resulting from disputes arise. In scenarios where no such disputes exist, all routers are able to select their most preferred paths. This solution provides just enough visibility to obtain the location of routers having policy conflicts thus easing troubleshooting, without revealing additional provider policies. We prove that the Precedence Solution is able to stabilize the network, then show how it can be implemented in practice.

Secondly, the high level of access control possible in intra-domain networks has resulted in the proliferation of semantically rich policies, which are realized in the form of packet filters and physical topology manipulations. The multitude of knobs to tune in order to achieve the desired performance increases the configuration complexity of these networks. We show that using the notion of classes embedded within routing, reachability information can be automatically propagated and updated by the routing protocol, hence easing configuration.

Professor Scott Shenker
Dissertation Committee Chair

To those searching for the light at the end of the tunnel.

Contents

List of Figures

List of Tables

Acknowledgments

I would like to thank Scott Shenker and Ion Stoica for their patience and guidance, and also Ruzena Bajcsy who first took me under her wing.

I would also like to acknowledge the assistance of those who worked with me at some point in time or another. Brent Chun's words of wisdom, Sylvia Ratnasamy's guidance, Pavlin Radoslavov's enlightening explanations on the inner workings of Xorp, Jinyang Li's knowledge of wireless testbeds, Lakshminarayanan Subramanian's introduction to BGP and its problems in all their glory, Vijay Ramachandran's patience in explaining the difference between a corollary and proposition, Byung-Gon Chun's willingness to rush IDS through the three weeks just before SIGCOMM. The technical support group of people here at Berkeley and ICSI also deserve many thanks: the EECS security team for catching the silly things I did, Mike Howard and Jon Kuroda for setting up the wireless testbed. I appreciate all the effort put in to help me towards my goal.

These five years have been made more bearable by a group of friends. Yuen Hui Chee, Siew Leng Teng, Ailee Ho, Engling Yeo, Kah Cheong Lai, Mong Hoo Lim, Anne Chong and Von Bing Yap are the bunch of Singaporean friends with whom I celebrate many festivals. Other equally notable friends at Berkeley include Linhai He, Wilson So, Jayanth Kumar, Karthik and Kaushik Lakshminarayanan, Ling Huang, Sukun Kim, Jaein Jeong, Fred Jiang, Boon Thau Loo, Matthew Kam, Weehong Tan, Li Zhuang, Feng Zhou, Matthew Caesar, Stanley Bo-Ting Wang, Weilun Chao, Jie-Hong Roland Jiang, Karl Chen, Dilip Joseph, Gautam Altekar, Rabin Patra, Sergiu Nedevschi, Sanjeev Kohli and many more.

Last but not least, I thank my family for their understanding and putting up with my long periods of absence and lack of updates.

Chapter 1

Introduction

The Internet began as an amalgamation of smaller networks under the control of a single organization, ARPA. As a military project, the primary goal of the Internet was to provide basic connectivity between end-hosts in the event of failures. Applications such as email could utilize any path through the network as long as the destination can eventually be reached. Since it was a network that was administered by a single organization and did not require the need to distinguish between packets and different paths to destination, basic routing protocols, first distance-vector then link-state, were sufficient to meet the primary goal of providing connectivity.

The subsequent break up of the Internet and distributed management by commercial internet service providers (ISPs), or autonomous systems (ASes), shifted the focus from providing basic connectivity to maximizing profits obtained from transiting customers' or neighboring providers' traffic. The selected routes to destinations are correspondingly important for all ISPs; that is, there is henceforth a need to distinguish between different routes. To provide route selection flexibility necessary for meeting service agreements, the Border Gateway Protocol (BGP) [40], a path-vector protocol, replaced link-state routing between autonomous systems. The high degree of policy selection freedom granted by path-vector routing allowed ASes to individually implement their policies. Unfortunately, coupled with the lack of visibility into other networks (because of commercial reasons), policy conflicts can exist between ASes, resulting in permanent route oscillations.

On the other hand, complete visibility and control over the intra-networks provided ISPs the freedom to deploy complex solutions that improve the performance of their networks. With the surge of malicious activities such as worm and denial of service (DoS) attacks and in general the lack of control over end-hosts (the customers of the ISPs), network-level policies on handling of packets have risen in importance. Since the Internet was created initially with the intelligence pushed to the end-hosts, implementation of the network-level policies have been more of an afterthought, and distributed amongst various types of components such as middle-boxes and even routing protocols. The problem of having varied and large numbers of independent components to configure is compounded by the difficulty in mapping high-level policies to low-level details such as IP addresses and port numbers.

This dissertation attempts to answer the following two questions:

1. Using some form of global constraint in a network administered in a decentralized manner and utilizing path-vector routing, is it possible to minimize loss of autonomy and impact on route selection without revealing ISPs' policies and yet ensure network convergence?

2. Is it possible to couple the functionalities of the various components used for enforce-

ment of access control, such that the number of knobs to tune is reduced and thus also the space of possible configurations?

1.1 Contribution I: Precedence Solution

The first major contribution of this dissertation is a global constraint that ensures routing convergence without revealing ISPs' policies. We show that the Precedence Solution has an impact on route selection only when there exists policy disputes resulting in oscillations; thus, all ISPs have complete freedom over the routes they choose when no such oscillations exist. The correctness of the algorithm is a two-step process, and shown first theoretically based on an assumption that is subsequently eliminated by the implementation of the algorithm in practice.

Theory: By assuming that each router maintains only knowledge of routes encountered during oscillations caused by disputes, routers indicate the possible presence of disputes by incrementing a precedence metric, which is simply a numerical value carried with each route advertisement. Using the notion of circular routing dependency present in policy disputes, transmission and reception of such indications confirm the presence of such disputes.

Practice: To obtain knowledge of routes that are seen during oscillations caused by disputes, we begin by testing whether recently available routes are contributing to disputes via the precedence metric. Such infeasible routes eventually expire if they are not involved in disputes. This constitutes the dispute detection phase. Next, routes confirmed to contribute to disputes are stored, and again relying on the circular dependency nature of disputes to remove stale state.

A minor contribution stems from usage of the precedence metric. Requiring only local information, *i.e.* observing metrics associated with incoming advertisements, a router is able to determine if local policies conflict. Using this knowledge, we can propagate troubleshooting information, such as the AS and router identifier number, together with the advertisements. This provides the right level of visibility across ISPs, easing troubleshooting.

1.2 Contribution II: Access Control Routing

The second major contribution is a method to implement network-level access control, while replacing (1) current distributed firewalls, (2) sophisticated routing protocols (*e.g.* BGP) and (3) methods such as physical topology manipulation for purposes of channeling traffic through network choke points. Using the notion of *class*, which can refer to a particular role (*e.g.* researchers), type of packet (*e.g.* HTML), or class of traffic (*e.g.* Voice-over-IP), we couple routing together with access control, letting routing propagate and update access information as well as the paths to be traversed by certain classes of traffic. This coupling reduces the number of knobs to tune in order to achieve the desired control, hence the effort required for configuration and therefore also the likelihood of errors.

As a minor contribution, we describe how access policies in a network can be modeled using an Access Graph. Using this graph, we propose an algorithm that heuristically determines the minimum number of classes required. Such an algorithm is important as it reduces the amount of state stored in the routers, as well as computation time required during routing updates.

1.3 Discussion

A commonly asked question is, “Does a common framework for routing policies across and within ISPs exist?” Although we believe that the answer is positive, such a framework is not likely to have much impact because (1) it is not likely to provide service differentiation between ISPs and (2) decoupling inter and intra-domain routing will in general lessen the impact of one on the other, thus improving overall system stability.

Firstly, although the general notion of traffic classes, be they assigned to control, data or network links, is applicable to both inter and intra-domain networks, the need for traffic differentiation across domains is in general absent. The primary reason is an economical one: ISPs generate profits from differentiating themselves from their competitors, not from collaboration with them. Thus, although a common framework is possible, there is insufficient motivation.

Secondly, one of the main problems with the Internet today is the lack of routing event isolation in one part of the network from another [48]. This results in churn that does not scale with network size. To minimize churn, one might simplify the interface and hence amount of control information sent across inter-domains. The general approach to access control in intra-domains supports rich policies, requiring the dissemination of non-trivial control information. Application of this approach (*e.g.* propagation of classes across domains) to the inter-domain can possibly lead to greater coupling between ISPs, causing the Internet to experience more churn.

1.4 Organization

The organization of this dissertation is as follows: Chapter 2 describes our Precedence Solution, including the theoretical proofs as well as details in practice. We describe the changes necessary in today's routers, and evaluate the solution using AS-level graphs generated from RouteViews [44]. Since ISPs' misbehavior involving manipulation of the precedence metric can cause routing to oscillate again, we end the chapter with a discussion on the detection of misbehavior.

Chapter 3 begins with general models of access control implemented in networks today. We next describe the general notion of classes and their motivation, followed by the design of Access Control Routing (ACR). The implementation of ACR on Click [32] is detailed, and we evaluate its performance, complexity and quantitative-wise. We end the chapter with a discussion on dealing with mobile nodes, as well as an algorithm to minimize the number of classes required in a network.

Finally, Chapter 4 concludes the dissertation, and ends with directions for future work.

Chapter 2

Policies in Inter-Domain Routing

2.1 Brief Overview

The Border Gateway Protocol (BGP) [40] establishes connectivity between the independent networks, called *autonomous systems* (ASes), that together form the Internet. BGP computes routes by a series of local decisions based on each ASes' individual routing policies. These policies are semantically rich in order to accommodate the complex rules that govern route choices in today's commercial Internet, such as business relationships and traffic engineering. However, this expressiveness in routing-policy configuration, coupled with ASes' freedom in implementing their policies autonomously, can cause instability in interdomain routing manifesting in the form of persistent route oscillations [50].

The problem of understanding and preventing policy-induced routing anomalies has been the subject of much recent study. While some work characterized these anomalies using global models [19, 20, 46], other research proved that global and local constraints on policies could guarantee routing stability. The good and bad news from this literature can be summarized as follows:

Good news: If the AS graph has an underlying business hierarchy and local policies obey sensible constraints arising from this hierarchy, then routing converges [16, 23].

Bad news: If ASes have complete freedom to filter routes (that is, exclude routes from consideration) then the only policies that are *a priori* guaranteed to converge are generalizations of shortest-path routing [13].

Thus, there are two choices: we can hope that natural business arrangements provide a stabilizing hierarchy, or we can remove all policy autonomy (but not filtering autonomy) by imposing some generalized form of shortest-path routing.

This paper advocates a "third way". Rather than rely on the vagaries of the marketplace to define a suitable hierarchy, or eliminate policy autonomy because of its potential to induce route oscillations, we propose a simple extension to BGP that constrains policy choices only after an oscillation is detected. Oscillations can be characterized by the presence of *dispute wheels* in the network [20], and our method provably finds and breaks dispute wheels, including those involving non-strict preferences. We tag each route advertisement with a *precedence value*, where a lower value corresponds to higher precedence. This goes at the top of the BGP decision process: available routes are chosen first based on their advertised precedence, with ties broken using the usual BGP decision process. The precedence attribute changes only in the presence of a persistent oscillation; if there is no oscillation, we effectively use only the normal BGP decision process. Since configuration is not constrained unless absolutely necessary, ASes' freedom to decide on local policies is preserved.

2.2 Dispute Wheels

We begin by describing the notation used in this paper. The network is represented as the AS graph $G = (V, E)$, where each node $v \in V$ corresponds to one AS, and each edge $\{u, v\} \in E$ corresponds to a *BGP session* between ASes u and v, meaning that these ASes are physically connected and share route advertisements. We assume that links between ASes are reliable FIFO message queues with arbitrary delays; this accounts for network asynchrony. At most one link is assumed to exist between ASes, and all the internal and border routers of an AS are condensed into one node (or one point of routing-policy control).

A path P is a sequence of nodes $v_1 v_2 \cdots v_k$ such that $\{v_i, v_{i+1}\} \in E$; we write $v \in P$ if path P traverses node v. Paths can be concatenated with other nodes or paths; *e.g.*, if $P = u \cdots v$, $Q = v \cdots w$, and $\{w, d\} \in E$, we may write PQd to represent the path starting at node u, following P to node v, then following Q to node w, and finally traversing the edge (w, d). We assume that paths are directed from source to destination.

BGP, at a schematic level, computes routes using the following iterative process: (1) Nodes receive *route advertisements* from their neighbors, indicating which destinations are reachable and by what routes; (2) for each destination, a node chooses the best route from those available, based on local policy; (3) if the current route to a given destination has changed, an advertisement is sent to neighboring nodes. The content of advertisements, or update messages, is also governed by routing policy; nodes are not required to share or consider all available routes, *i.e.* routes may be *filtered.* The process begins when a destination advertises itself to its neighboring ASes; routes to that destination then propagate through the network as transit nodes choose routes and send updates. Because route choices are computed independently for each destination, we will focus our attention on, without loss of generality, on a single destination node $d \in V$.

We say the network has converged when each AS $v \in V$ is assigned a path $\pi(v)$ to the destination, such that the assignment is *stable*, *consistent* and *safe*. By consistent, we mean that the paths form a forwarding tree to the destination; if $\pi(v) = vuP$, then $\pi(u) = uP$. By stable, we mean that $\pi(v)$ is the "best" available route for each node v, given the other nodes' path assignments, where "best" is determined by node v's routing policy; that is, if $\pi(v) = v\pi(u)$, there is no other node w such that the path $v\pi(w)$ is more preferred at v than $\pi(v)$.

Safety is slightly more subtle. By unsafe, we meant that there is some sequence of route updates that does not converge, in which every node gets a chance to update infinitely often. Because there are only a finite set of route choices, such a sequence must be a route oscillation. The sequence may or may not be dependent on particular delays in receiving route updates. A configuration is *safe* if any sequence of route updates, in which no node is shut out, converges.

Griffin, Shepherd, and Wilfong [20] showed that any such oscillation can be characterized by a *dispute wheel* in the network, shown in Figure 2.1. The dispute wheel captures the interaction amongst the routing policies of a set of nodes that are involved in a route oscillation. Formally, we have the following.

Definition 2.2.1 *A* dispute wheel *is a set of nodes* $p_0, p_1, \ldots, p_{k-1}$ *(assume all subscripts are modulo k) called* pivots*, such that*

1. *at each pivot* p_i*, there exists a* spoke path Q_i *from* p_i *to the destination;*

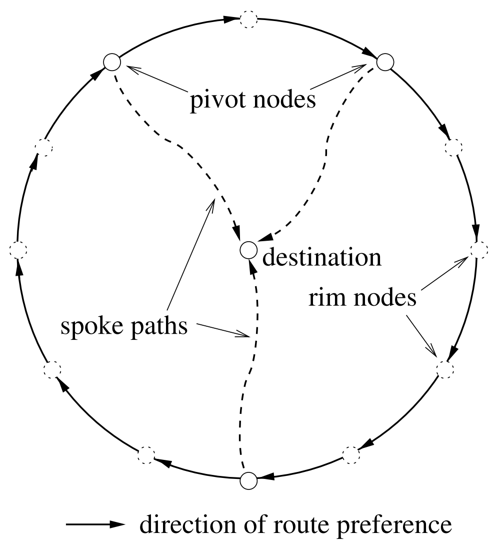


Figure 2.1: *Example of a dispute wheel: elements of the wheel include the spoke paths, pivot nodes, and rim nodes.*

2. *at each pivot* p_i*, there exists a* rim path R_{i+1} *to the next pivot* p_{i+1}*;*

3. *each pivot prefers the path* $p_i R_{i+1} p_{i+1} Q_{i+1} d$ *over the path* $p_i Q_i d$*.*

Note that the rim and spoke paths are not necessarily disjoint. We refer to non-pivot nodes along the rim paths R_i as *rim nodes.*

Since dispute wheels lie at the heart of BGP policy instabilities, we now walk through an example of BGP dynamics in the presence of a dispute wheel. Consider the four-node network shown in Figure 2.2. In the figure, paths considered by a node are listed in the shaded box next to that node in decreasing order of preference. The oscillation is shown in Figure 2.3. (i) Assume that the destination node D sends an initial advertisement to nodes A, B, and C. (ii) Nodes A, B, and C then choose the direct paths to D and advertise their choices to nodes C, A, and B, respectively.(iii) Upon receiving this advertisement, each node prefers the route through its neighbor, rather than the direct path to D, and chooses

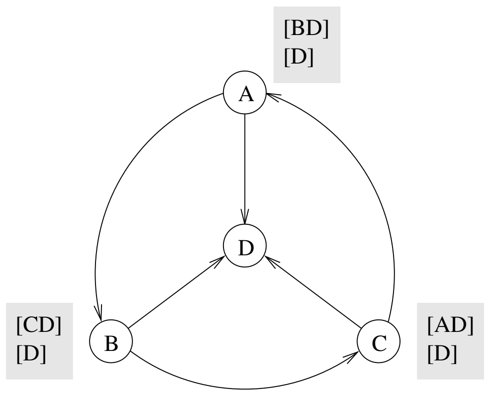


Figure 2.2: A simple dispute wheel: node D is the destination. Shaded boxes show route choices in order of preference.

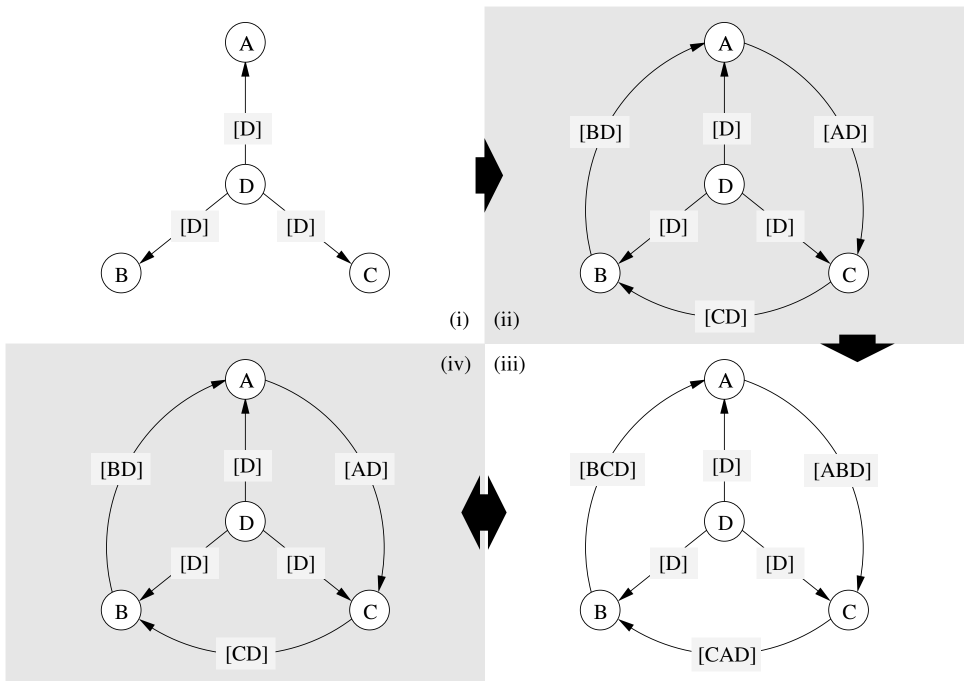


Figure 2.3: Simple example of dispute wheel oscillation: The simple local policy enforced at each node is the import filtering of routes with more than 2 hops. Routing oscillates between (iii) and (iv).

it. Doing so requires advertisement of these new paths; with the longer paths selected, the direct paths to D are no longer advertised. (iv) When node A learns that node B has selected BCD, its preferred choice of ABD is no longer available; so node A reverts to choosing the direct path to D. By symmetry, this occurs at nodes B and C as well. This state is identical to (ii); therefore, the sequence of route updates repeats, and nodes A, B, and C oscillate forever between their two route choices.

Any policy-induced oscillation can be characterized by a dispute wheel; thus, the absence of dispute wheels is sufficient to guarantee that BGP is always safe. However, the presence of a dispute wheel does not necessarily guarantee an oscillation; even if there are some initial conditions that will lead to an oscillation, BGP could non-deterministically converge.[1] Rather than exclude all potentially troublesome policy relationships *a priori*, the method we describe in the next section triggers a mechanism to resolve the corresponding dispute wheel whenever an oscillation is detected.

2.3 The Precedence Metric

We begin by augmenting BGP's decision process, prepending it with an additional step that utilizes a new metric which we call the precedence metric. We describe this metric below, and show that it eliminates route oscillations due to dispute wheels.

Each route advertisement is tagged with a *global*[2] *precedence value* that is non-negative: a numerically greater value translates to a lower precedence. We denote the global precedence value, say v, associated with path P by $(P{:}v)$. Each AS maintains a *history* of

[1]For instance, a four node dispute wheel can converge into one of two stable configurations.

[2]Again, the term global only means that this precedence value has meaning across more than one AS, not that all ASes share this precedence value.

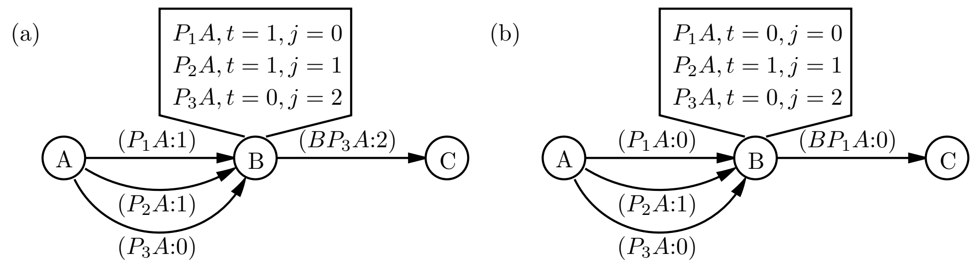


Figure 2.4: *(a) AS B's local preference for route* $(P_3A{:}0)$ *to destination AS A is ranked third. Selection and propagation of this route to AS C will result in the increasing of its global precedence by 2. (b) AS B now considers the route* $(P_1A{:}0)$ *to AS A to be the most locally preferred. Selection and propagation of this route to AS C will not alter its global precedence value.*

observed route advertisements from its immediate neighbors. In this history, we associate every route with a *local precedence value* starting from 0. This local precedence value is obtained from the route's rank, and is determined via the usual BGP decision process. Thus the route ranked ith has a local precedence of i-1 and is preferred over all routes with local precedence greater than that. Strict ranking is performed, such that no two routes of equal local precedence exist.

Suppose the selected route has an incoming global precedence of t, and a local precedence value of j. Then, the outgoing route advertisement is tagged with t+j. Thus, a route that is most preferred for all ASes along its path is tagged with 0 at all hops. Figure 2.4 gives an example of this update process. Without loss of generality, we assume for the rest of this paper that the destination AS advertises routes with global precedence value of 0. We next show that this precedence metric prevents the formation of dispute wheels.

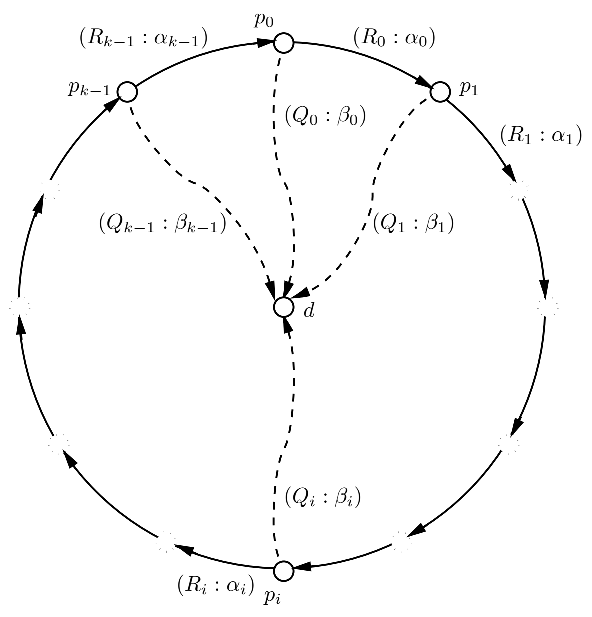


Figure 2.5: *Dispute wheel illustration and notation used in our proof.*

2.3.1 Dispute Wheel Elimination

Proposition 2.3.1 *If routes encountered during previous policy-induced oscillations are stored and the precedence metric is used, then no further policy-induced oscillations can occur.*

Proof: It is proven in [20] that the absence of dispute wheels is sufficient for safety, and hence it suffices to show that the precedence mechanism precludes dispute wheels. Using proof by contradiction, we begin by assuming that a dispute wheel exists.

Figure 2.5 is used to illustrate our proof, in which we consider a single destination d. Nodes $p_0, p_1, \ldots, p_{k-1}$ are the subset of nodes that are in the dispute wheel and have stable paths to the destination, that is, these are the pivot nodes. $(Q_i{:}\beta_i)$ is the tuple consisting of Q_i, the spoke path from source p_i to destination d, and β_i, the precedence value associated with path Q_i. The tuple $(R_i{:}\alpha_i)$ on the other hand consists of the rim path R_i, which leads from p_{i+1} to p_i, and α_i, the change in precedence along R_i, including node p_{i+1}. In other words, if γ is the precedence value for path $R_i p_{i+1} Q_{i+1} d$, then $\gamma{=}\beta_{i+1}{+}\alpha_i$.

Suppose $p_0, p_1, \ldots, p_{k-1}$ each receive route advertisements from their immediate next hops along $Q_0, Q_1, \ldots, Q_{k-1}$ with global precedence values $\beta_0, \beta_1, \ldots, \beta_{k-1}$, respectively. Node p_i then selects the route Q_i, updates the value, and advertises that.

We next assume that the dispute occurs: node p_i prefers path $(R_i p_{i+1} Q_{i+1} d{:}\beta_{i+1}{+}\alpha_i)$, over route $(Q_i d{:}\beta_i)$. In Figure 2.5, this corresponds to each node picking its immediate neighbor, in the clockwise direction, as the next hop. In this proof, we assume that the route advertisements received and stored as part of the history include those encountered during oscillations.[3] Note that we do not need *all* routes encountered during one oscillation period to be stored, merely one that has higher local precedence than the stable spoke route. Then, the dispute wheel implies

$$\begin{aligned}
\beta_1 + \alpha_0 &\leq \beta_0 \\
\beta_2 + \alpha_1 &\leq \beta_1 \\
&\vdots \\
\beta_0 + \alpha_{k-1} &\leq \beta_{k-1}
\end{aligned}$$

Summing, we obtain

[3]Other routes will at most merely increase the precedence value, and not affect the correctness of the proof.

$$\begin{aligned}\sum_{i=0}^{k-1}\beta_i + \sum_{i=0}^{k-1}\alpha_i &\leq \sum_{i=0}^{k-1}\beta_i \\ \text{or} \quad \sum_{i=0}^{k-1}\alpha_i &\leq 0\end{aligned}$$

Since, by definition, $\alpha_0, \alpha_1, \ldots, \alpha_{k-1}$ are non-negative, we have

$$\alpha_i = 0 \qquad \forall\ i$$

which implies that all nodes $p_0, p_1, \ldots, p_{k-1}$ locally prefer routes through $Q_0, Q_1, \ldots, Q_{k-1}$ respectively. This means that if the dispute wheel exists and each $R_i p_{i+1} Q_{i+1}$ is chosen over Q_i, it must be because of the global precedence values.

Thus, for the system to oscillate, we will require

$$\begin{aligned}\beta_i + \alpha_{i-1} &< \beta_{i-1} \qquad \forall\ i \\ \text{or} \qquad \beta_{k-1} < \beta_{k-2} < &\cdots < \beta_0 < \beta_{k-1}\end{aligned}$$

which is not possible. Therefore, by contradiction, no oscillations due to dispute wheels can exist. ■

Proposition 2.3.2 *If there are non-zero precedence values advertised once the protocol converges, this must mean that dispute wheels exist.*

Proof: Assume that the destination node advertises routes with precedence value 0, and that the network has converged. Thus, a non-zero value advertised somewhere means that there exists some node v with an incoming set S of routes of precedence value 0, $|S| > 0$, and an advertised route vP, $P \in S$, with positive precedence value. If this happens, then

P must not be the most *locally preferred* route; suppose that route is Q. The precedence value of Q must be positive, otherwise v would have chosen it. This means there must be some node w along Q that increases its precedence value; w is similar to v, in that it must have some other path Q' with positive global precedence, causing it to choose Q. Thus, we can repeat this process at w and subsequent similar nodes. As the destination node is never encountered, because it always advertises routes with precedence value 0, we must ultimately encounter a node already traversed. The resulting cycle of nodes naturally form a dispute wheel that has been resolved using the precedence mechanism. ■

Corollary 2.3.3 *From Propositions 2.3.1 and 2.3.2, global precedence values greater than that advertised by the destination exist when routing converges if and only if dispute wheels that can cause oscillations exist.*

Corollary 2.3.4 *A route traversing resolved disputes cannot advertise the same global precedence at all hops.*

Proof: Assume that such a route exists. Since the precedence value advertised by all hops are the same, this implies that the route selected by each node is its most preferred. This in turn implies that the destination node must be part of the dispute wheel, which is a contradiction. ■

2.3.2 Autonomy Loss in Presence of Disputes

Corollary 2.3.3 showed that only the presence of dispute wheels can cause positive global precedence values to exist after routing converges. The increased value advertised

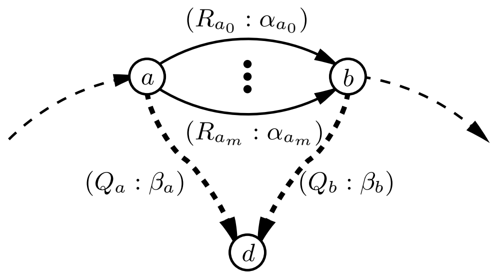


Figure 2.6: *Multiple paths advertised by neighboring nodes can cause the global precedence value of a route to increase by more than 1.*

Table 2.1: History of Node b in Figure 2.6

Route	Global Precedence	Local Precedence
$R_{a_0}aQ_ad$	$\alpha_{a_0} + \beta_a$	0
$R_{a_1}aQ_ad$	$\alpha_{a_1} + \beta_a$	1
$\cdots$	$\cdots$	$\cdots$
$R_{a_m}aQ_ad$	$\alpha_{a_m} + \beta_a$	m
Q_bd	β_b	$m+1$

by the pivot nodes depends on the number of paths advertised in parallel by immediate neighboring pivot nodes.

We use Figure 2.6 to explain this. Here, node b has a spoke path Q_b to destination d. Assuming that b locally prefers routes advertised by neighboring pivot node a along $R_{a_0}, R_{a_1}, \ldots, R_{a_m}$ compared to Q_b, we have the history state shown in Table 2.1. Clearly, if the spoke path is selected, it will be advertised as $(bQ_bd{:}\beta_b{+}m{+}1)$.

A non-uniform increase[4] in global precedence values around the dispute wheel causes the rest of the network, *i.e.* nodes not in dispute and not along spoke paths, to lose autonomy. To correct this, instead of increasing the selected route's value by its local precedence, we bound the increase by 1. We call this the *precedence+* metric.

Proposition 2.3.5 *Usage of the precedence+ metric eliminates oscillations caused by dis-*

[4]in the sense that some pivots increase outgoing routes' global precedence by x, and others by y, where $x \neq y$.

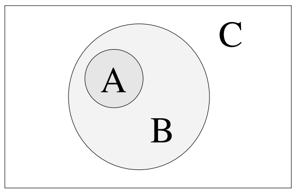


Figure 2.7: *Region A is encompassed by nodes involved in a dispute wheel. Routes advertised in the external region B have global precedence values one higher than those in A. Similarly, if the nodes around the edges of B are in dispute, the global values in C will be one higher than those in B.*

pute wheels.

Proof: The following constraint is added to the proof of Proposition 2.3.1:

$$\alpha_i \leq r_i \qquad \forall\ i$$

where r_i is the total number of nodes along R_i, including p_{i+1} and excluding p_i. The rest of the proof follows. ■

Proposition 2.3.6 *Usage of the precedence+ metric results in an increment in global precedence value at steady state only in the presence of dispute wheels that result in route oscillations.*

Proof: Same as that for Proposition 2.3.2. ■

Corollary 2.3.7 *From Propositions 2.3.5 and 2.3.6, the global precedence value increases by one if and only if a dispute wheel exists and causes routes to oscillate.*

Precedence values can take on multiple non-negative values as opposed to just binary 0 or 1 values. With reference to Figure 2.7, the presence of a dispute wheel causes routes beyond the nodes in and within the wheel, that is, nodes in region B and not A, to be advertised with the same incremented value. Nodes in region B can still be in dispute, in which case the global precedence will be incremented again.

Corollary 2.3.8 *Only nodes that prefer routes through nodes in dispute may lose autonomy.*

Proof: Trivial. ■

For the rest of this paper, we focus solely on the precedence+ metric.

2.3.3 Accounting for Non-Strict Preferences

The precedence+ metric is proven to eliminate dispute-based oscillations for *strict preferences*; that is, routes can be ranked independent of others. In general, preferences are non-strict, and are encountered for instance in BGP's Multi-Exit Discriminators (MEDs) [30]. In this subsection we propose a minor extension to account for this.

The primary effect of having non-strict preferences is that an incoming route R_i causes route R_{cs} to be selected, where $R_i \neq R_{cs}$ and R_{cs} is not the previous route selected (R_{ps}). This is an *Independent Route Ranking (IRR)* violation [24]. In terms of strict preferences, it appears as though the existence of R_i results in the disappearance of R_{ps} from the most locally preferred rank. Thus, to capture this as part of the history of routes encountered, we associate a logical route R'_{ps} with R_i, and comparison of R'_{ps} with any other route should ignore the presence of R_i. This slight tweak is necessary for computing the local preference of the selected route. Since the goal is to determine if the global precedence should be incremented, we will be comparing R'_{ps} with R_{cs}, ignoring R_i. Since $R_i \neq R_{cs}$, we will not encounter the scenario when the two will be compared. In the case where R_{cs}

becomes unavailable in the future and is replaced by R_i, we evict R'_{ps}.

2.4 From Theory To Practice

In §2.3, we showed that usage of the precedence+ metric, coupled with the knowledge of routes encountered during oscillations, can cause the network to converge. The primary difficulty in implementing the solution is knowing precisely the relevant set of routes encountered during oscillations and not others. In this section we describe how this is achieved in practice. We begin by defining our goals:

One: We distinguish between transient and permanent oscillations, where the former disappear with the convergence of the network. The association of routes with disputes should be removed if the latter is found to be transient. Further, changes in network topology affecting resolved disputes should cause the removal of stored state associated with those disputes.

Two: The solution should not reveal any ISP policies.

Three: Only local information associated with incoming advertised routes is necessary; no global knowledge is required.

Four: Knowledge of potential pivot nodes should be provided as feedback by the protocol. The presence of resolved disputes causes precedence values to increase, thereby possibly restricting the choices of routes. In general we believe it is preferable to react by altering the local preferences at a subset of the pivot nodes so that disputes do not arise in the first place and route choices become unconstrained. Since access to the global view is not assumed and is probably unattainable, we seek an alternative means of identifying the potential pivots.

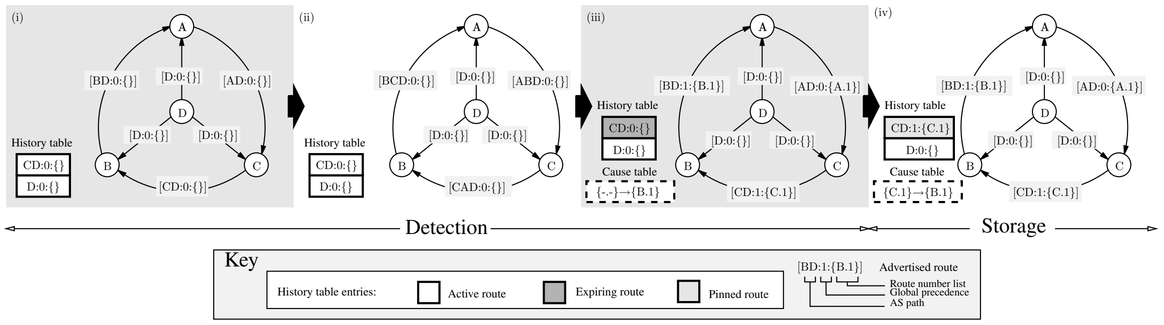


Figure 2.8: Dispute wheel formation and elimination: the simple local policy enforced at each node is the import filtering of routes with more than 2 hops. The history table entries are ordered according to local preference. Since the network is symmetrical, only the history and cause tables for node B are shown. In phase 1, detection of disputes takes place, with -.- in (iii) indicating an increase in advertised route's precedence due to an expiring, more preferred route (as opposed to a feasible one). The route number $B.1$ represents route BD:1. In phase 2, incoming feasible routes cause outgoing ones to have increased values, and the cause tables are updated accordingly.

Our complete Precedence Solution consists of two main phases: detection and storage. We detect routes involved in oscillations, then store and maintain their associated information so that previously encountered disputes remain resolved and corresponding oscillations do not reoccur. *As discussed later in §3.7, if AS policies are based solely on next hop neighbors, then only the first phase, detection, is required, and the overall solution is simplified.*

2.4.1 Detection & Short-Term Memory

In the detection phase, *short-term* memory of more preferred routes that are *infeasible*, result in less preferred but more stable routes being advertised with larger global precedence values. During this period of time, such routes are said to be *expiring*. This mechanism determines if a possible dispute exists and operates locally, without requiring information beyond the routes received. Short term memories need only exist until it can be confirmed whether disputes resulting in permanent oscillations exist. This ensures that transient oscillations do not cause unnecessary suppression of routes. In general this amount of time is determined by the number of rim nodes between neighboring pivots, since rim nodes can be thought of as delaying route advertisements. As the number of rim nodes is difficult to obtain in practice, we can upper-bound it by using routes' path lengths. We store short term memory in a *history table.*

2.4.2 Storage & Long-Term Memory

Subsequently in the storage phase, incoming routes with larger global precedence values result in more stable ones being advertised. In this phase, short-term memories of the last unavailable routes are no longer required, but, as we shall show shortly, *long-term* memories are needed instead. Thus, the main difference between the two phases is the global precedence value of incoming routes: the first phase has incoming routes of the same value, whereas the latter has less preferred routes having smaller values. The two different types of memories together make up the required *history* mentioned earlier in §2.3.

An incoming, more preferred route causing a less preferred one to be advertised with increased global precedence is stored in long term memory by being *pinned.* At the time of pinning, these incoming routes must either be *feasible*, i.e. currently being advertised by the neighbors, or their *ignore* list (containing the route demoting the current one's local rank in the presence of IRR violations) must be non-empty. Pinned routes are never evicted automatically from long term memory, and may only be considered when selecting routes if they are currently being advertised by neighbors. Also, pinned routes are unpinned only when the *causes* of their increased value have been eliminated, or a more preferred route is received and selected, or, in the case of IRR violations, a more preferred route (in the absence of the ignored route) is received. Thus, this implies that only routes associated with some number of causes can be pinned.

A *cause* of an increase in the selected route's value refers to a more preferred route that is currently advertised by a neighbor. It can be thought of as a pointer to a route, thus using it in place of the referred route reduces storage and communication overhead. We use a unique *route number* to represent a particular cause, obtaining a globally distinct number by concatenating the router's IP (say A) with a locally generated sequence number (say 24) giving us $A.24$. Rather than propagate route numbers all around the wheel, we instead maintain cause mappings in the *cause table*: when a new route is advertised with increased global precedence, we assign it a new route number, and associate with that the corresponding causes. Thus, an outgoing route number is no longer valid once all its incoming causes are removed. Long term memory consists of pinned routes and the cause mappings, which is stored in the cause table.

2.4.3 A Simple Example

We next use a simple example (Figure 2.8) to illustrate the resolution process. We denote a route using the 3-tuple P:V:L, where P refers to the AS path, V the global precedence value and L the list of route numbers. We assume strict preferences in this example and disregard the ignore list for now. We also assume that routes longer than two hops are filtered. Since the network and policies are symmetrical, we focus on a single node

B. We begin with dispute detection; in (iii), the expiring of route CD:0:{} causes the less preferred route from D to be advertised with precedence value 1. Since route CD:0:{} is expiring and there is an absence of more preferred, feasible routes, we denote the cause by {-.-}, which we call the *null* route number. Note that the expiring route CD:0:{} is not pinned since its route number list is empty. In (iv), incoming route CD:1:{C.1} *causes* route BD to be advertised with precedence 1, and the cause table is updated accordingly. In this case, CD:1:{C.1} is pinned.

The astute reader will notice that in this example, long term memory, in the form of the cause table and pinned routes, is unnecessary to ensure convergence, since incoming routes with value 1 are always feasible. As described later in §3.7, if AS policies are based solely on next hops, the solution as described in this simple example would be sufficient. We motivate the necessity of long term memory in the next example.

2.4.4 A Complex Example

Figure 2.9 shows a scenario where multiple dispute wheels intersect at node X, where R_A denotes a route to destination with A as the last hop node. Again we assume strict preferences in this example. In (i), we assume that the first dispute, involving nodes A, B, X and C, has been resolved, with X selecting the less preferred route via Z. Subsequently in (ii), we have the converged network after the second dispute, involving nodes Y, X and Z, has been resolved. We note that:

1. Pinned routes ensure that previously encountered disputes that should still be resolved remain so. At pivot A, the absence of pinned route BXR_Z:1:{X.1} will result in the

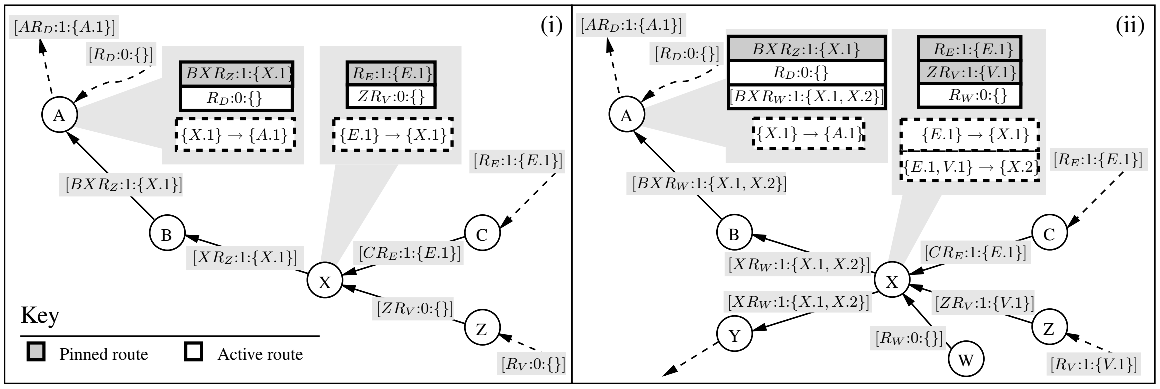


Figure 2.9: R$_A$ denotes a route with A being the last hop node. (i) A simple dispute wheel is first resolved, with nodes A and X being the pivots, B and C the rim nodes, and Z is the last node along the least preferred route. (ii) Node X is involved in a second dispute, where pinned routes ensure that the first dispute remains resolved.

selected route R_D:0:{} being advertised with global precedence value of 0, which eventually un-resolves the first dispute.

2. Route numbers associated with selected routes are propagated unchanged. For instance, rim node B advertises numbers $X.1$ and $X.2$, ensuring that the next pivot (A) keeps route BXR_Z:1:{$X.1$} pinned.

3. The new route advertised by X is associated with a different route number $X.2$. In this case, since the routes of $E.1$ and $V.1$ are more preferred than R_W:0:{}, they are deemed to be causes of the latter route's increase in global precedence. Note that the cause table entries for previous route numbers, for instance $X.1$, are no longer updated.

4. At A, since received route BXR_W:1:{$X.1$,$X.2$} is the least preferred, *X.2* does not become a cause of *A.1*.

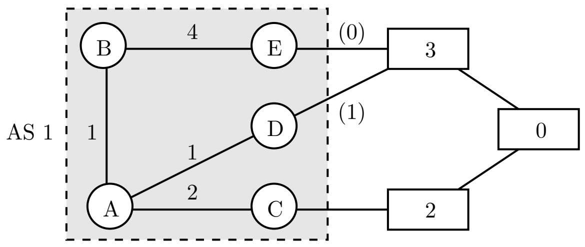


Figure 2.10: The MED-EVIL example from [18].

5. Changes in network topology is taken into account. For instance, breakage of link XC will, at X, eliminate $E.1$, which in turn removes the corresponding entry in X's cause table and thus $X.1$, and thereafter unpin (and remove expired) route BXR_Z:1:{X.1} at A, and so forth. This corresponds to the elimination of the first dispute wheel.

2.4.5 A MED Example

A significant problem in BGP today is the occurrence of oscillations due to MED [30]. MED selection rules are different from local preferences, AS path lengths etc. because they result in non-strict preferences. Figure 2.10 shows an example from [18]. Here, link weights in brackets denote MED values assigned to links from external ASes, whereas weights within AS 1 indicate the link's iBGP cost. Table 2.2 shows the sequence of routes advertised during an oscillation period.

We observe from Table 2.2 that the primary issue is the change in the most preferred route, from $D30$ to $C20$, with the reception of $BE30$. That is, the cause of $D30$ being demoted in rank is brought about by $BE30$. In order for the dispute detection to be effective, we create a logically different, expiring route $D30'$ that is still the most preferred (in the absence of $BE30$). $BE30$ is associated with $D30'$, the former is ignored when com-

Table 2.2: MED Oscillation in Figure 2.10

	A		B	
Step	Available	Advertised	Available	Advertised
1	D30, C20	AD30	E30	BE30
2	BE30, D30, C20	AC20	E30, D30	BE30
3	BE30, D30, C20	AC20	E30, AC20	BAC20
4	D30, C20	AD30	E30, AC20	BAC20
5	D30, C20	AD30	E30, D30	BE30
Repeat from step 2.				

paring the latter with other routes (for instance when determining whether other routes are more preferred). Subsequently, the selected route $AC20$ will be advertised with increased precedence. Pinning of the logical route $D30'$ at A (via incoming route $BE30$) and route $AC20$ at B takes place (long term memory), and the dispute is resolved. For the logical route $D30'$ to be unpinned, there must be an incoming route that is more preferred than it in the absence of $BE30$. As before, we note that no additional policies are revealed.

Denoting a stored route by the 4-tuple $P{:}V{:}L{:}I$, where I refers to the list of incoming routes to be ignored, or the *ignore list*, when computing this route's local precedence, the sequence of route updates is shown in Table 2.3. Note that the ignore list is not sent to neighboring nodes.

2.4.6 Convergence Proof

We now show that the correct routes are pinned so that disputes are resolved and remain so. The proof focuses on individual pivot nodes.

Proposition 2.4.1 *With long term memory, the network eventually converges.*

Table 2.3: MED Oscillation Elimination

Step	A	B

1	<u>Available</u>
	D30:0:{}:{}
	C20:0:{}:{}
	<u>Advertised</u>
	AD30:0:{}
		<u>Available</u>
		E30:0:{}:{}
		<u>Advertised</u>
		BE30:0:{}

2	<u>Available</u>
	D30':0:{}*:*{*BE30*}
	BE30:0:{}:{}
	C20:0:{}:{}
	D30:0:{}:{}
	<u>Advertised</u>
	AC20:1:{A.1}
		<u>Available</u>
		AD30:0:{}:{}
		E30:0:{}:{}
		<u>Advertised</u>
		BE30:0:{}

3	<u>Available</u>
	D30':0:{}*:*{*BE30*}
	BE30:0:{}:{}
	C20:0:{}:{}
	D30:0:{}:{}
	<u>Advertised</u>
	AC20:1:{A.1}
		<u>Available</u>
		AD30:0:{}*:*{}
		E30:0:{}:{}
		AC20:1:{A.1}:{} - pinned
		<u>Advertised</u>
		BE30:1:{B.1}

4	<u>Available</u>
	D30':0:{}*:*{*BE30*} - pinned
	BE30:1:{B.1}:{}
	C20:0:{}:{}
	D30:0:{}:{}
	<u>Advertised</u>
	AC20:1:{A.1}
		<u>Available</u>
		E30:0:{}:{}
		AC20:1:{A.1}:{} - pinned
		<u>Advertised</u>
		BE30:1:{B.1}

Proof: Let the new available route be R_n, the previous selected route be R_w, and the set of non-empty routes (including pinned routes) more preferred than R_w be $\mathcal{R}_p$.

Suppose R_n is not selected. This implies that R_n's global precedence is at least as low as R_w's. If R_n is less preferred than R_w, then there is no further effect. Otherwise R_n becomes one of the causes of the increase in global of R_w. In either case, no oscillations are introduced as the advertised route's AS path and global precedence remains unchanged.

On the other hand suppose R_n is selected and advertised. **Case 1: If R_n is more preferred than R_w,** then its precedence is at most as high as R_w's, and we unpin (and remove expired) pinned routes less preferred than R_n, as well as remove their corresponding outgoing route numbers. Previous dispute-induced oscillations associated with the unpinned routes cannot reappear due to the presence of R_n, that is, we will not again encounter exactly the same routes advertised during previous oscillations. Since more preferred, pinned routes are not unpinned, and the maximum number of these routes is bounded, this scenario cannot occur indefinitely.

Case 2: If R_n is less preferred than R_w, then its precedence value must be strictly lower than R_w's. Since all pinned routes less preferred than R_w must have been unpinned before, none less preferred than R_n will exist. In other words, disputes previously resolved remain so.

■

2.4.7 Achievement of Goals

Based on the Precedence Solution proposed earlier, we next describe how our goals are met.

Handling transient and permanent oscillations: An oscillation that is eliminated is said to be permanent if the route numbers associated with outgoing, increased precedence value routes at each pivot are always advertised. In the case of transient oscillations, the expiration of more preferred routes eliminates the corresponding outgoing route numbers, thereby unpinning upstream routes. Alternatively, more preferred routes can be encountered later, in which case those less preferred and pinned will be unpinned and also removed if expired.

If instead network topology changes, such as link breakage, occur, incoming route numbers via that link is removed, the cause table is updated accordingly and its effect is subsequently propagated around the wheel. Similarly, the cause and history tables can be flushed whenever policy changes occur. Thus, the Precedence Solution allows for both transient oscillations and changes in network topology, and does not permanently suppress any routes.

Minimal revealing of policies: If the input routes of a router are known, the global precedence of the advertised route indicates whether the chosen route is the most preferred: if it is not, then its value increases. This is not much different from today's network: given the inputs, a route is not the most preferred if it is not advertised.

For routes stored in the history table, all have been previously advertised before and have been intended to be used for routing, none have been explicitly propagated for purposes of eliminating oscillations. Route numbers have been explicitly associated with these routes, and do not contain any additional information. Thus, we do not expose any additional AS policies.

No requirement for global knowledge: Both the detection and storage mechanisms operate solely on route advertisements received from neighbors, and are fully decentralized. No third party is required to gather and compute optimal routes for all ASes. The route numbers are propagated upstream only for the purposes of ensuring that disputes remain resolved, and do not affect nodes elsewhere, including other parts of the wheel.

Identification of potential pivot nodes: Although it is possible to use some other unique number as the route number, we believe that inclusion of the router IP gives the right amount of visibility to assist in network troubleshooting. If a node is forced to select a less preferred route, the cause table maps incoming route numbers to an outgoing one, appended to those already associated with the selected route. Otherwise the numbers associated with a selected route is propagated unchanged. Thus, the set of nodes identified by the list of numbers includes all potential pivots encountered downstream. Although not all pivots along the wheel can be identified from a single viewpoint, adjustment of just one such node's preferences is sufficient to break the dispute, reducing global precedence values and relaxing constraints on route selection.

2.5 Router Changes

In this section we describe extensions to a BGP router necessary to implement the Precedence Solution. The two main additions are the history and cause tables.

2.5.1 History Table

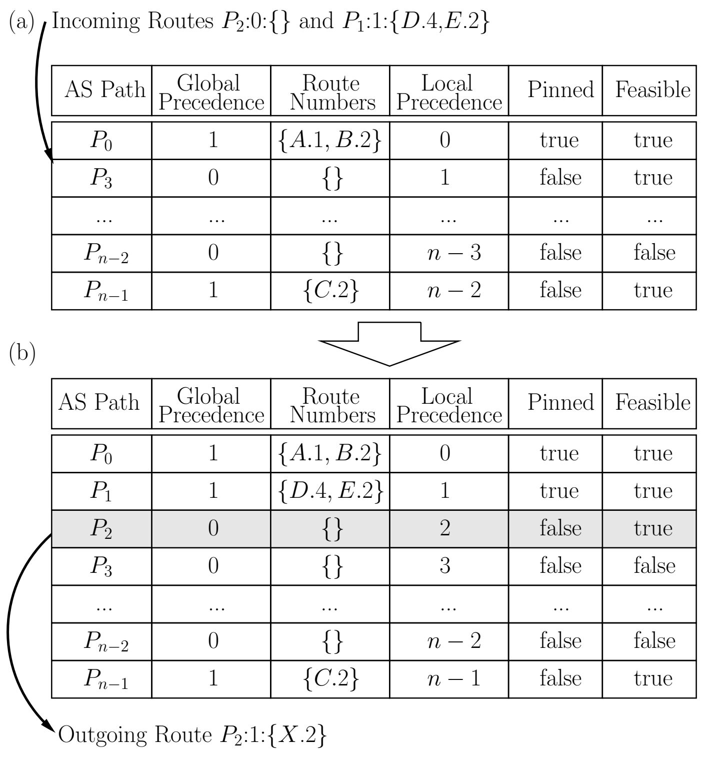
The history table stores routes received from neighbors, as well as information relevant to short and long term memories necessary for dispute detection and storage. Memory used to store routes may be shared amongst the different data structures, and is dependent on actual implementation. Thus, the history table can be thought of as an extension to other structures.

A route that is currently being advertised is *feasible*. Infeasible routes that are not pinned are removed, and only feasible routes can be considered for selection and pinning. A route is unpinned if there exists a selected route that is more preferred (or if no route is selected). Pinning of a route occurs when it has a non-empty route number list and it causes a less preferred route to be advertised with increased global precedence value.

Figure 2.11 shows an example of a history table being updated, and Figure 2.12 provides the pseudo-code. Entries in the table are arranged in order of local precedence, that is, the ordering is determined using the same rules as the decision process in use today. This ordering provides the local precedence value: the most locally preferred has value 0, the next 1, and so forth.

2.5.2 Cause Table

The cause table contains entries that map incoming causes' route numbers to outgoing ones. To recapitulate, the route number of a cause consists of a router interface's IP address and a locally unique sequence number. Figure 2.13 shows an example of a cause table being updated corresponding to the update in Figure 2.11, with Figure 2.14 providing



(a) Incoming Routes P_2:0:{} and P_1:1:{D.4,E.2}

AS Path	Global Precedence	Route Numbers	Local Precedence	Pinned	Feasible
P_0	1	$\{A.1, B.2\}$	0	true	true
P_3	0	$\{\}$	1	false	true
...	...	...	...	...	...
P_{n-2}	0	$\{\}$	$n-3$	false	false
P_{n-1}	1	$\{C.2\}$	$n-2$	false	true

(b)

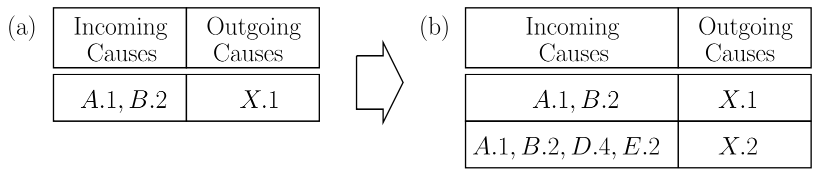
AS Path	Global Precedence	Route Numbers	Local Precedence	Pinned	Feasible
P_0	1	$\{A.1, B.2\}$	0	true	true
P_1	1	$\{D.4, E.2\}$	1	true	true
P_2	0	$\{\}$	2	false	true
P_3	0	$\{\}$	3	false	false
...	...	...	...	...	...
P_{n-2}	0	$\{\}$	$n-2$	false	false
P_{n-1}	1	$\{C.2\}$	$n-1$	false	true

Outgoing Route P_2:1:{X.2}

Figure 2.11: (a) Before and (b) after a history table is updated.

```
 1: if local routing has converged (no routing changes) then
 2:     for each entry in history table do
 3:         if not feasible and not pinned then
 4:             remove entry
 5: for each route $R$ received do
 6:     if route from neighbor $N$ is filtered then
 7:         set previous feasible route $R_p$ from $N$ infeasible
 8:     else if different route received from neighbor $N$ then
 9:         set previous route $R_p$ from $N$ infeasible
10:         compute $R$'s local precedence
11:         insert $R$ into history table, set feasible
12:     else if same route received from neighbor then
13:         if previous feasible route $R_p \neq R$ then
14:             set $R_p$ infeasible
15:         else
16:             update $R$'s global precedence value and route numbers
17:         set $R$ feasible
18:     else if first route $R$ is received from neighbor then
19:         compute $R$'s local precedence
20:         insert $R$ into history table, set feasible
21: select set $\mathcal{S}$ of eligible routes
22: select set $\mathcal{S}'$ with lowest global precedence, $\mathcal{S}' \subset \mathcal{S}$
23: select route $R$ with lowest local precedence, $R \in \mathcal{S}'$
24: for each entry in history table do
25:     if route $R_{mp}$ more preferred than $R$, feasible, has non-null route number list then
26:         pin $R_{mp}$
27:     else if route $R_{lp}$ less preferred than $R$ and pinned then
28:         unpin $R_{lp}$
29: return $R$
```

Figure 2.12: Pseudo-code for updating history table and determination of the selected route for each destination.



(a)

Incoming Causes	Outgoing Causes
$A.1, B.2$	$X.1$

(b)

Incoming Causes	Outgoing Causes
$A.1, B.2$	$X.1$
$A.1, B.2, D.4, E.2$	$X.2$

Figure 2.13: Updating of cause table corresponding to history table update in Figure 2.11. A change in selected route triggers the generation of a new sequence number, and only most recent outgoing causes are updated.

```
create empty route number set $\mathcal{S}_r$
let selected route be $R_s$
**for** each entry in history table **do**
    **if** route $R$ more preferred than $R_s$ and feasible **then**
        pin $R$
    **else if** $R_s$ more preferred than $R$ **then**
        unpin $R$
    **if** route $R$ is pinned **then**
        add $R$'s route numbers to $\mathcal{S}_r$
**for** each entry in history table **do**
    **if** route $R$ is pinned and none of $R$'s route numbers $\in \mathcal{S}_r$ **then**
        unpin $R$
**if** $R_s$ is different from previous **then**
    create new route number
**if** $\mathcal{S}_r \equiv \emptyset$ and $R_s$ is not most preferred and feasible **then**
    add null route number to $\mathcal{S}_r$
set current outgoing cause's incoming route numbers to $\mathcal{S}_r$
**for** each entry in cause table **do**
    **if** no incoming route numbers are present in $\mathcal{S}_r$ **then**
        remove cause entry
```

Figure 2.14: Pseudo-code for updating cause table for each destination prefix.

the pseudo-code. An entry exists until all its incoming causes cease to be received. In the figure, the cause entry for $X.1$ will be evicted if both $A.1$ and $B.2$ are no longer advertised by the neighbors. A new route number $X.2$ is created with a change in the selected route. Only current causes, that is, those using the most recent sequence numbers, are updated based on received routes. Thus in Figure 2.13 new incoming causes will be added to the entry for $X.2$, but not for $X.1$.

2.5.3 Adaptive Convergence Window

As elaborated in §2.4, we require the use of short term memory to detect disputes. The convergence window is the period of time during which received routes are kept in memory. Assuming that one-hop route propagation delay W is similar to the Minimum Route Advertisement Interval (MRAI), the rim nodes (say there are r of them) can be

```
1: for each adj-RIB-in do
2:     process incoming routes, update route table
3: update history table
4: update cause table
5: for each adj-RIB-out do
6:     update new routes' route number list
7:     advertise route to peer
```

Figure 2.15: Primary steps in router batch updates.

thought of as delaying route advertisements from one pivot to another by rW, thus the window size should be proportional to this number.

The convergence window begins with a short duration (one MRAI), so that networks not containing disputes can converge relatively quickly. We double its duration when convergence does not occur after some time, so that disputes involving large numbers of rim nodes can be resolved quickly. An upper bound can be determined using the maximum observed path length during this period. Lastly, its duration is reset after the network stabilizes to remove effects of transient convergence.

2.6 Evaluation

2.6.1 Simulator

We built an event-based, packet-level and asynchronous simulator. Route updates are *batched*, and take place every Minimum Route Advertisement Interval (MRAI). Figure 2.15 shows the main steps of the batch update process, whereas Figures 2.12 and 2.14 describe maintenance of the history and cause tables respectively. We set MRAI to 30 seconds, processing delay jitter to 1 second, and link propagation delay to 10 milliseconds.

2.6.2 Metholodgy

To better understand the basic performance of our solution, we use simple graphs, which consist only of rim, pivot and destination nodes. Figure 2.8 shows an example. Whilst these graphs are not representative of a real network in general, they are still useful in determining properties of a dispute wheel.

To evaluate the effectiveness of the Precedence Solution in practice, we use an AS-level network topology constructed using routing table dumps from RouteViews [44]. Route dumps from January 3rd 2007 were used to construct an AS-level network which consists of 24307 ASes and 56914 inter-AS links. Since complete policy information is impossible to obtain [14,47], we sought an alternative method of generating local preferences. Restricting ourselves to next hop preferences, we note that a dispute-free configuration can be obtained as long as the most preferred neighbor lies along a cycle-free path to the destination. Thus, a shortest-path algorithm will generate local preferences that can guarantee convergence.

However, inter-domain routing typically does not result in shortest paths [49], and as we show later, the network convergence time as well as the degree of route exploration (and hence the number of routes encountered) are dependent on the ratio of actual versus shortest path lengths (*i.e.* route inflation). Thus, we focus on routing algorithms that provide approximately the same route inflation. We use a combination of depth-limited and breadth-first searches to obtain routing trees: depth-limited search is used whilst within the limit at each stage, otherwise BFS is used. In general, increasing the maximum depth at each stage results in greater route path inflation. The remaining neighbors' preferences are set in a random fashion. Finally, we simulated misconfigurations by selecting a subset of

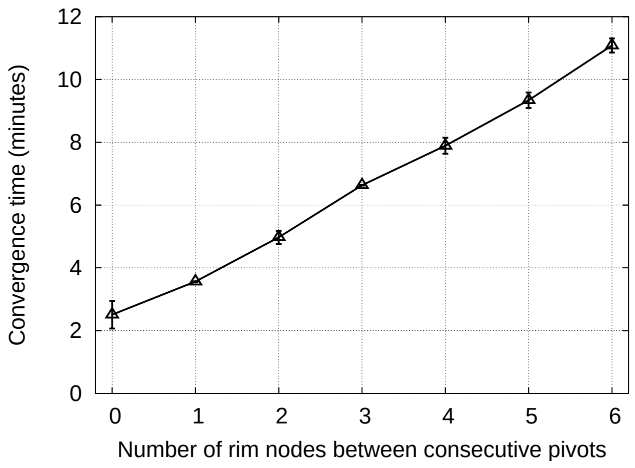


Figure 2.16: Simple graphs: 3-pivot network's convergence time against rim-to-pivot ratio.

routers and randomly assigning local preferences.

2.6.3 Metrics

We use convergence time and memory requirement as metrics. We say that a node has converged at a certain time if its routing table no longer changes thereafter. As for memory requirements, we look at the ratio of routes stored when using our solution against normal BGP. This allows comparison across the entire network, taking into account routers with varying numbers of neighbors.

2.6.4 Results

Simple graphs Using simple graphs, we determined that the convergence time is dependent on the rim-to-pivot ratio and not the total size of the network. We show representative results in Figure 2.16, where the number of pivot nodes is 3. Each data point in the figure is obtained from 20 samples; we see that the mean convergence time increases with this ratio, and there is little deviation in all cases. In all experiments the networks converged.

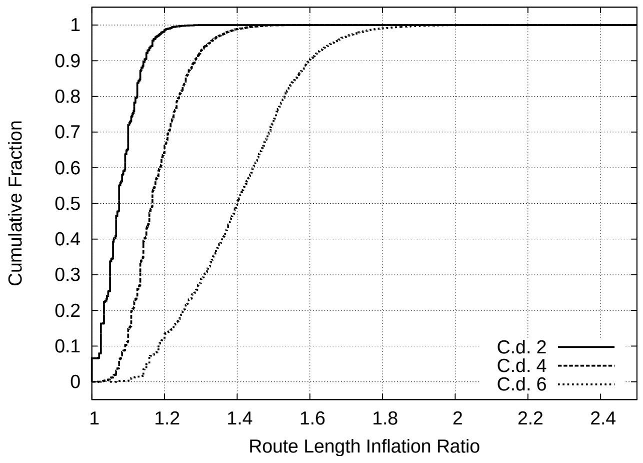


Figure 2.17: An increase in the maximum depth of constrained depth-first routing results in more inflated routes. For constrained depth (c.d.) of 6, we obtain paths with inflation close to that in practice [49].

RouteView graph We varied the maximum depth of each constrained depth-first iteration, obtaining the mean route length inflation ratios shown in Figure 2.17. A maximum depth of 6 results in route inflation that most closely match that in the Internet today [49].

Next, we investigated the impact of additional memory requirements for Precedence+ by varying route inflation. In all cases, we verified that usage of Precedence+ in networks with no disputes resulted in all nodes selecting their most preferred next hops: Precedence+ does not unnecessarily suppress routes. For normal BGP, the amount of memory required at a router is proportional to the number of its neighbors. From Figure 2.18, we observed that deviation from the shortest path results in more routes being explored and hence more being stored before convergence in the case of Precedence+. On average, Precedence+ requires 50% more memory for each destination prefix, which can be amortized across the network by jittering initial prefix advertisements. Furthermore, actual route exploration in the Internet may be to a lesser extent since route advertisement will

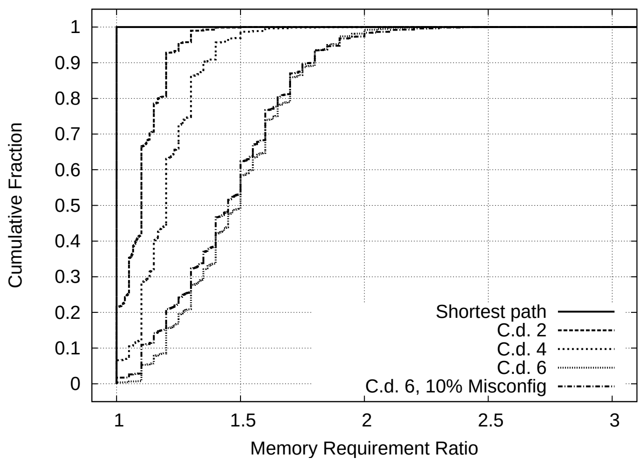


Figure 2.18: As path lengths deviate away from shortest, the exploration of more paths before convergence results in more routes being stored for Precedence+.

be constrained by economic policies.

To investigate policy disputes, we randomly assigned next hop preferences to 10% of the nodes. We verified that dispute wheels do exist (normal BGP does not converge), and that the networks converged when Precedence+ is used. As shown in Figure 2.18, we required approximately the same amount of memory as before.

Finally, we looked at the network convergence times (Figure 2.19). As we expected, local preferences assigned based on shortest-paths results in faster convergence. More importantly, convergence time is not significantly affected by usage of Precedence+, nor by the presence of misconfigurations (disputes) in the network.

2.7 Discussion

In this section we discuss two issues encountered in practice, namely constrained policies and misbehavior.

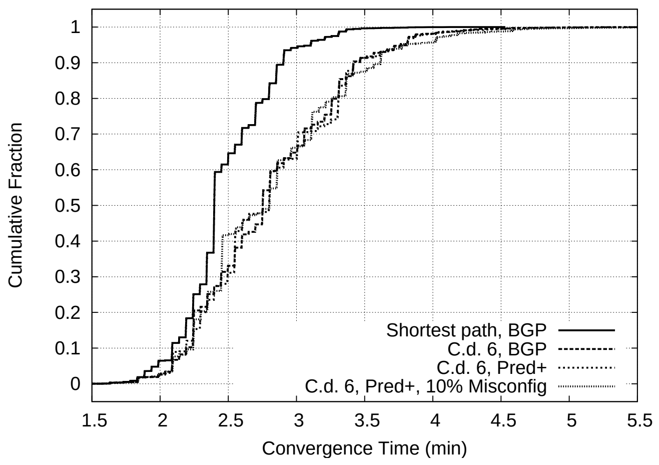


Figure 2.19: Inflated paths result in an increase in convergence times. Precedence+ does not delay convergence, even in the presence of misconfigurations.

2.7.1 Constrained Policies

If the local precedence value of a route is determined first by the last hop, that is, if the first step of the BGP decision process selects routes based on next-hop ASes, then the Precedence Solution can be significantly simplified. In Figure 2.9(ii) at node A, if neighbor B's routes are more preferred than D's, then route BXR_W:1:$\{X.1,X.2\}$ will be ranked higher than R_D:0:$\{\}$, thus the eviction of BXR_Z:1:$\{X.1\}$ will not un-resolve the first dispute. In other words, pinning of routes and long-term storage, including cause tables and route numbers, are unnecessary. Communication overhead is reduced as well, since only the global precedence value need to be carried with each route advertisement.

2.7.2 Misbehavior

Since the global precedence metric can in general restrict the autonomy of an AS, there may be incentives for not adhering to the general rule. We discuss various ways whereby ASes can misbehave, and detection methods that rely on the ability to observe

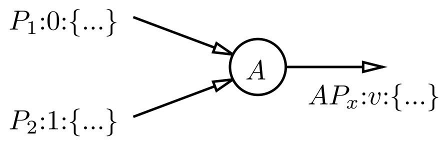


Figure 2.20: Basic scenario used to describe misbehavior: node A receives two routes and advertises one. Detection of misbehavior can be performed by observing incoming and outgoing routes.

the incoming and outgoing routes. Clearly, one type of misbehavior is the selection of an available route with the highest local precedence regardless of its global value. We describe several scenarios using Figure 2.20, focusing on the routes advertised from A.

$\mathbf{P}_2$:0:{...} there is definite misconduct, since the outgoing route's global precedence is less than its incoming's. This is true even if A filters P_1:0:{...}.

$\mathbf{P}_2$:1:{...} there is no misconduct only if A permanently filters route P_1:0:{...}. In this case, route P_2:1:{...} is the only incoming route and therefore also the most locally preferred. Thus, the outgoing route's global precedence is not incremented.

$\mathbf{P}_x$:v:{...} where v>2 for x=1 and x=2. In this case, node A is artificially increasing the outgoing precedence value. This has the effect of not allowing upstream ASes to select a route traversing this AS. While some may construe this as misbehavior, it may be used as a means of indicating that certain links are used as backup. For instance, the destination node can advertise a global precedence value of 1 on backup links, and 0 on normal links.

From this simple example, we can determine that an AS is misbehaving if one of these two conditions are satisfied: (1) an outgoing route has a global precedence value that is less than its corresponding incoming route, or (2) an outgoing route has a global

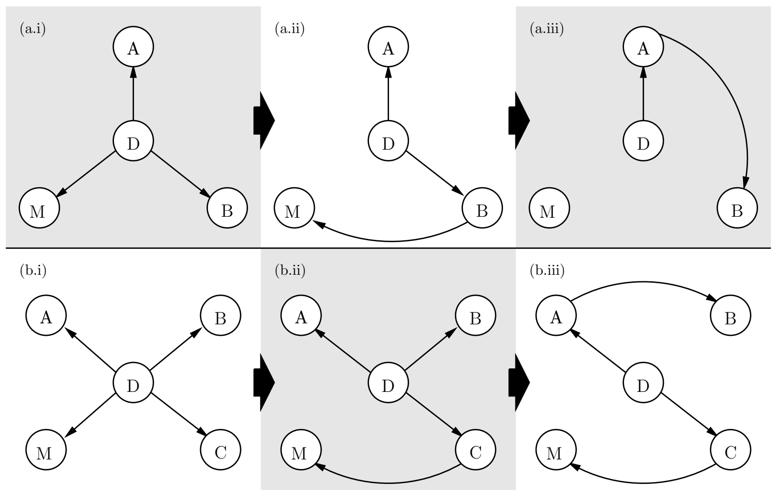


Figure 2.21: A misbehaving AS, represented by node M, can have differing effects on the network. (a) For a dispute wheel with an odd number of nodes, M eventually lacks a route if it initially filters the spoke one. (b) For a wheel with an even number of nodes, M does not destabilize the network.

precedence value that is greater than its corresponding incoming route by more than one.

2.7.3 Adaptive Filtering

Misbehavior that is more difficult to detect involves *adaptive filtering*, which we now describe. Let M be the node representing a misbehaving AS. Clearly, if M is always filtering its spoke path, it will never become a pivot node, and thus cannot influence the convergence process. However, M involved in a dispute can initially accept routes from neighbors along the spoke and rim. When routing stabilizes and the precedence+ metric forces selection of the spoke path, M can subsequently decide to effectively filter that in order to select the locally preferred path along the rim.

In this case, two scenarios can occur as illustrated in Figure 2.21. In part (a), the total number of pivot nodes in dispute is an odd number. The selection of a next hop that is more locally preferred but having a higher global precedence value eventually results in

M not having a valid route. Subsequent removal of the filter causes the system to oscillate again.

In part (b), an even number of pivot nodes can cause the system to settle in a stable state even if M misbehaves. In this case, M is able to use the path it locally prefers.

In general it is difficult to determine the number of pivot nodes in dispute, and therefore hard to know if the implementation of adaptive filtering in M can result in oscillations (which ultimately does not benefit M). To provide better control of the situation, we next propose a method to detect the various types of misbehaviors discussed above.

2.7.4 Misbehavior Detection

Most ASes are comprised of multiple routers, and are unlikely to provide access to the internal network. Thus, the usual assumption that an AS can be modeled by a single router does not hold. For instance, in Figure 2.22(i), router A selects, as it should, the less preferred route R_0:0:{...} and advertises AR_0:1:{...} to B. B subsequently chooses R_2:1:{...}. If we logically collapse A and B into a single node and aggregate their inputs, we see that even though the routers are behaving correctly, the output should have been ...R_0:1:{...} instead.

We propose a slight tweak to the protocol only within an AS: when an ingress router (i.e. A in the example) advertises a route to an internal peer, it appends the route's global precedence when received (the *ingress* value) and after updates (the *egress* value). Upon reception of that route, B uses the ingress value to determine the selected route. The egress value is then updated, and is lower-bounded by the previous egress value. Advertisements to neighboring ASes carry only the egress value.

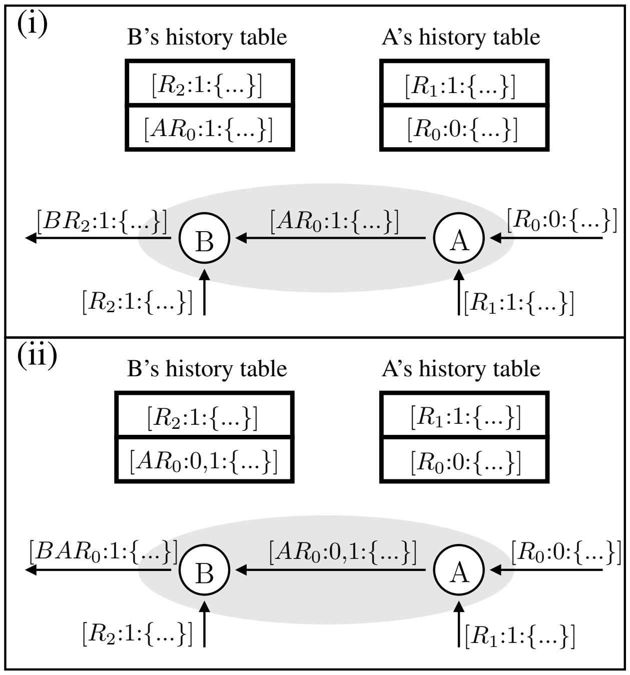


Figure 2.22: (i) With multiple routers within an AS, indicated by the shaded region, external input and output routers can appear to indicate misbehavior even if they are operating according to protocol. (ii) By tagging the route with both ingress and egress precedence values, an AS' behavior becomes similar to that of a single router.

Figure 2.22(ii) shows the same network with the tweaked protocol. Here, correct behavior will cause A to advertise AR_0:0,1:{...}, and B to advertise BAR_0:1:{...}. On the other hand, if A misbehaves and selects R_1:1:{...}, the output will clearly be incorrect.

With the slightly modified protocol, the conditions described in §2.7.2 can be used to detect the occurrence of adaptive filtering. For a dispute to occur, a less preferred route (say R_{lp}) must have been advertised before the more preferred one is selected. Thus, R_{lp} must have been observed before, but not thereafter. A monitoring mechanism can be designed based on this as follows: we detect routes that should have been selected but aren't. These are then hashed and stored. Since the monitor is maintained by a third-party, hashing of the inputs provide anonymity. Output of any of the stored routes in the future signals reuse of those routes, and therefore adaptive filtering.

2.8 Background

Varadhan, Govindan, and Estrin [50] were the first to discuss the possibility of persistent route oscillations in BGP. The cause was not the policy configuration of one AS alone; they occurred because of interaction between the policies of several ASes. These anomalies occurred without any misconfiguration and were difficult to diagnose and resolve since ASes tend to keep routing policies private.

Griffin, Shepherd, and Wilfong [20] introduced the Stable Paths Problem (SPP) as a formal model for BGP (and policy routing with path-vector protocols, in general). Using their framework, they were able to give a sufficient condition for protocol convergence, namely, the absence of *dispute wheels*. These structures characterize the conflicting policies of the nodes involved in a route oscillation (see the formal definition in §2.3). Unfortunately, the only known method to check for dispute wheels requires examining all the routing policies in a network, which is presently an impractical task. In addition, Griffin *et al.* showed that the problem of detecting whether stable routing exists, given all the policies in the network, is NP-complete. Worse yet, they showed that the existence of a stable solution does not automatically imply that a routing protocol can find it.

Gao and Rexford [16] showed that Internet economics could naturally guarantee route stability. A hierarchical business structure underlying the AS graph, along with policies that matched the various business agreements between ASes, is sufficient for protocol convergence. In this structure, it is assumed that relationships between ASes are either *customer-provider*, *i.e.*, one AS purchases connectivity from another, or *peer-peer*, *i.e.*, two ASes mutually agree to transit traffic. No customer-provider cycles are allowed (*i.e.*, no AS, through a chain of providers, is an indirect customer of itself), and additional rules exist on how to set route preferences and when routes can be shared with other ASes. These assumptions capture the structure and economics of today's commercial Internet, although violations of these assumptions due to complex agreements, business mergers, or misconfigurations can still induce route oscillation. These positive results were later confirmed by Gao, Griffin, and Rexford in [15], in which the combination of an underlying business structure and economically sensible policies was shown to prevent occurrences of dispute wheels, even when backup routing is allowed. Jaggard and Ramachandran [23] generalized this result but still required some assumption about the AS graph to prevent oscillations.

Dispute-wheel freeness and an AS business hierarchy are examples of *global con-*

straints, because they require that some condition is enforced involving the policies of many ASes at once.[5] However, policy autonomy is at the heart of the philosophy that led to BGP, and ISPs will be loathe to relinquish it. Accordingly, later research attempted to find *local constraints*—conditions that could be checked individually for each AS—that are sufficient for route stability. Unfortunately, results here were mostly negative. Sobrinho [46] and Griffin, Jaggard, and Ramachandran [19] proved that any dispute-wheel-free routing configuration is equivalent to a generalization of lowest-cost routing. This means that many seemingly sensible policies — in fact, all purely local policies not driven by some shared metric — could lead to oscillations. For example, it was shown that ASes risk oscillations if they use policies that always prefer routes through one neighbor over another—a type of policy commonly used today. Feamster, Johari, and Balakrishnan [13] further strengthened this result by showing that only generalizations of lowest-cost routing can guarantee stability while preserving the ability of ASes to *filter* routes (that is, to remove them from consideration). Overall, the theme of these results is that the only way to *a priori* guarantee stability is to essentially eliminate policy-configuration autonomy.

Most of these results exclude policies with *any possibility* of inducing routing anomalies, whether or not they actually do in a particular network. (This is because determining whether the network policies will result in oscillations is too difficult.) In this paper, we present an extension to BGP that detects oscillations and responds by breaking the corresponding dispute wheel. Griffin and Wilfong also presented such an algorithm, called SPVP, in [21]. Our protocol differs in several ways. First, SPVP records the changes

[5]In this paper, as is standard for BGP discussions, the term *global* really means "not purely local". A global value, for instance, is not one that necessarily all ASes share, but that applies to more than one AS.

in route choices due to the propagation of a route; this reveals more private policy information than necessary. Second, our protocol answers an open question left by [21], in that we present a minimal-impact solution to resolving disputes: our resolution algorithm is engaged only when an oscillation is detected, and BGP is allowed to function normally otherwise. Third, SPVP's update-message size grows with the number of nodes in an oscillation, while additional fields used by our protocol scales with the *number of resolved disputes* encountered along a path. This is similar to that in [15, 23]; however, those solutions still required a global constraint and preemptively excluded some oscillation-free policy configurations that our solution does not exclude.

Another class of runtime solution involves diffused computation [8], which uses the observation that, as long as a change in path results in reception of another with a local preference value at least as high as that of its current path, then stability is guaranteed. In this case, an AS is required to ask any other AS whose path currently traverses it if a change in path is acceptable. Such a solution would restrict a provider's route choices based on inputs from customers, which is typically not the case in practice.

Finally, we allow ASes to exercise full autonomy *unless* the particular set of policies and topology results in an oscillation, and in that case, and only in that case, AS autonomy is revoked. What distinguishes this from much of the previous literature is that it does not place *a priori* restrictions on ASes, only *post hoc* restrictions. This enables a far greater degree of freedom, and we believe that ASes might be willing to accept the limitations as the price to pay for stability.

2.9 Summary

This paper tries to reconcile two desirable, but seemingly incompatible, goals. On the one hand, it is a business reality that ASes would like to set policies according to their own specialized needs — whether these arise out of business, or traffic engineering, or other concerns — and they would like to keep these policies private. On the other hand, every AS would like to have a stable Internet, where routes didn't oscillate. Unfortunately, recent theoretical results make clear that to ensure *a priori*, without knowing the policies beforehand or relying on assumptions about the structure of business relationships, that routing will be stable, ASes must be deprived of essentially all policy autonomy. In this paper we no longer require an *a priori* guarantee, but instead seek to remove policy-induced oscillations when they arise. This allows us to preserve policy freedom when possible, and impose stability when required.

Chapter 3

Policies in Intra-Domain Routing

3.1 Brief Overview

Enterprise networks play a critical role in the security policies of the organizations they serve by enforcing *access control*. Access control typically involves two components (1) blocking packets between hosts that are not allowed to communicate at all, and (2) ensuring that a host does not receive or send packets unless they are first passed through a middle-box where they can be inspected, audited, or scrubbed.

Many techniques exist for implementing access control [2,6,22,36,37,51,52]. However, setting up and operating these systems requires maintaining the consistency of a tremendous amount of state that is distributed across all the components of the network. For example, the rules that define which hosts can communicate are typically written in terms of hosts' IP addresses, yet the middle-boxes or firewalls where these rules must be installed can be located anywhere in the network, with no obvious or direct relationship to the hosts. Any change to the IP address assigned to a host invalidates the rules, but the burden of ensuring consistency falls to the network administrators. An example of maintaining the consistency of state would be the BGP routing policies used in some enterprises to enforce coarse-grained reachability policies, and the physical manipulation of links to ensure packets traverse a particular middle-box. The tremendous amount of state that has to be considered simultaneously makes the network brittle and configuration error-prone.

This paper proposes and evaluates *Access Control Routing* (ACR), a way of using the routing protocols themselves to distribute the state needed for access control, thereby automatically maintaining consistency and reducing configuration. Our approach to network access is inspired by the model of file access control, where users belong to groups and, in turn, groups are permitted or denied access to files.

In ACR, administrators define *classes* in their network. For any entity in the network, administrators can specify the classes that entity can send packets to and the classes it can receive packets from. We use the routing protocol to create logically separate networks for each class, so that if a host is not allowed to send packets to a particular class, none of the destinations in that class will even appear routable. Packets are explicitly marked with the class they are part of, but middle-boxes are able to change the class markings on packets and re-advertise destinations from one class to another. This primitive creates an easy and flexible way to channel packets through middle-boxes.

ACR has four major benefits: (1) Configuration is simplified because the designer needs only group the appropriate end-hosts based on their roles, and thereafter the routing protocols themselves maintain the resulting reachability and path constraints. (2) ACR provides a flexible framework for channeling packets through any number of "bump-in-the-

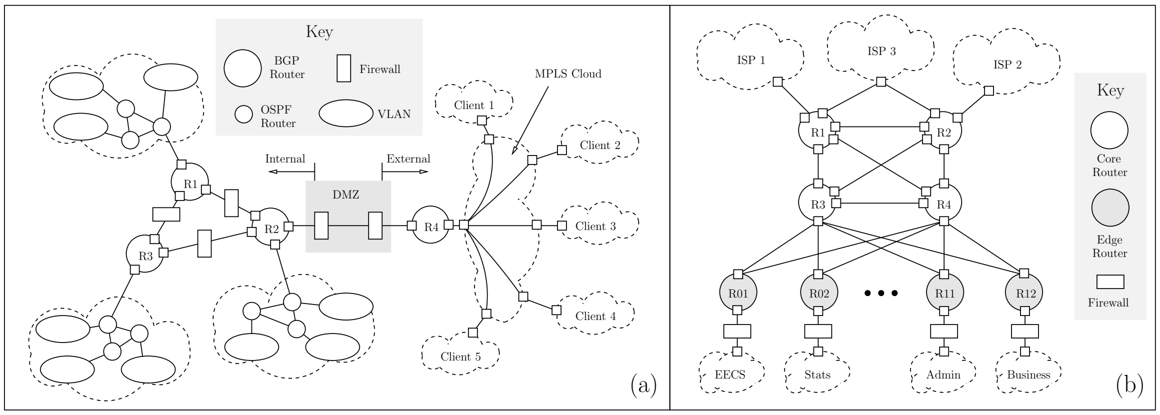


Figure 3.1: (a) General layout of the ComNet commercial network, where firewalls are placed in the core of the network, and routing is constrained for traffic to traverse these firewalls. (b) The UNet university campus network structure, firewalls are usually placed at the edges, next to the departmental subnets, and routing is unconstrained.

wire" firewalls [6,22] and deep-packet inspectors [35,36] in any order desired. (3) ACR frees administrators to assign IP addresses based on network topology rather than what will ease writing packet filters, since filters will no longer be written in terms of addresses but rather classes. (4) ACR retains the advantages of network-level access control over pure host-based access control in that Denial-of-Service attacks or worms that might overwhelm or subvert a host can be discarded before they even reach the host.

3.2 Networks in Practice

To better understand the issues with current networks and to facilitate comparison later, we gathered topology and configuration information from two large intra-domain networks: a large commercial enterprise (ComNet) that interacts with a variety of client entities and a university campus network (UNet). These networks follow two general models, which we call *core* and *edge*. We begin by describing the similarity between these two models, followed by an elaboration on the differences between them.

3.2.1 Aggregation of Hosts via VLANs

In both core and edge models, hosts that should have the same reachability policies may be plugged into different switches. To ensure these hosts all receive IP addresses from the same subnet so that IP routing policies and packet filters can be applied to them as a group, designers are forced to drag VLANs through multiple switches to gather each set of related hosts into a single virtual layer-2 LAN before connecting them to a layer-3 router. Configuring these VLANs (which are essentially multi-point permanent virtual circuits) is a painful and often manual process, requiring designers to carefully assign the order in which switches will become the root bridge and explicitly prune links to ensure the spanning tree protocols behave reasonably [7].

At layer-3, configuration methodologies between these two types of network begin to diverge, in terms of routing and middle-box set up. We next elaborate on the two models, focusing on the differences between them. We begin with the ComNet network.

3.2.2 The Core Model: A Commercial Network

As shown in Figure 3.1a, the commercial network is comprised of two main regions: an external MPLS network that provides connectivity to various client organizations, and an internal network that hosts various services. A demilitarized zone (DMZ) sits between these two network partitions, filtering unwanted packets from the external network.

The internal network consists of VLAN subnets at the periphery, and these are connected via OSPF [33]. At the center of the network, core routers running BGP [40] restrict the flow of packets in a coarse-grain manner using import and export policies. Firewalls are co-located with core routers, siphoning off packets for inspection and re-injecting them via alternate ports.

ComNet epitomizes the Core Model (CM), which has the following characteristics:

Few, Heavy-Weight Middle-boxes: A relatively small number of middle-boxes are placed at locations traversed by most traffic, such as within the network core region. Since packets can be sent from any source and destined to any end-host (as opposed to edge routers that see packets destined for or sent from its subnet), firewall rules tend to be numerous resulting in an increase in state required and processing delays. Furthermore, such rules are often difficult to debug, and rely to a large extent on the routing protocol, which ultimately dictates which traffic flows traverse the middle-boxes. On the other hand, having fewer middle-boxes may ease configuration time and complexity.

Controlled Routing: The second characteristic of CM is that routing, or more precisely the actual paths taken, is explicitly configured. In the case of ComNet, the export and import rules of BGP, which in ISPs are used to reflect economic policies, are enlisted here to enforce this control. In addition to BGP, the physical configuration of the network has to be restricted as well, as exemplified by the single link through the DMZ to ensure all packets traverse the two firewalls there. In general, currently available means for implementing security push designers towards static routing that is particularly prone to network link failures, resulting in brittle networks that lack redundancy. Furthermore, addition of new links can unintentionally result in the availability of alternate paths that bypass middle-boxes [17].

3.2.3 The Edge Model: A Campus Network

Figure 3.1b gives the high-level view of the UNet campus network. Since this is primarily a network that caters to learning and research purposes, the interior of the network has no access restrictions. As such, a single intra-domain routing protocol, OSPF, is sufficient without requiring BGP. The network itself serves multiple departments, which may or may not utilize the firewalls placed at the edge close to the departmental networks. The UNet network is representative of the Edge Model (EM), which has the following features:

Multiple, Light-Weight Edge Middle-boxes: Middle-boxes are placed as close to the client subnets as possible, allowing them to handle smaller destination and source sets thus reducing the state and number of rules required. However, the total number of middle-boxes to configure is therefore higher, increasing configuration complexity. On the up side, unintentional bypassing of middle-boxes is less likely to occur since it is easier to control the physical connections to the subnets.

Unrestricted Routing: The advantage of placing middle-boxes at network edges is that fewer constraints are needed on routing, since the physical chokepoint makes it harder to bypass the middle-boxes. Consequently, these networks allow additional links to be added, with the routing protocol automatically adjusting paths to take the new links into account.

3.2.4 Summary of Differences

In summary, current network designers either control routing to constrain traffic paths, resulting in brittle networks, or push the complexity to edges of the network, thereby necessitating the configuration of more middle-boxes. In some instances, BGP's route import and export ability has been enlisted to provide the necessary level of path control. In others, alternate paths are eliminated to produce physical chokepoints that channel the traffic. Unfortunately, both approaches decrease path diversity, making the network failure prone and increasing recovery times.

Additionally, high-level access policies need to be translated manually into the form of rules and installed at middle-boxes. The difficulty in translating between high-level policies and low-level rules means that numerous firewall rules are difficult to make correct. From anecdotal evidence, the lack of coupling between routing and access control at the protocol level can be further compounded by the fact that they may not be under the same administration. In the next section we describe our design, and show how the entire configuration process can be simplified.

3.3 Configuration Using Classes

In this section, we describe how the basic notion of *classes* in ACR can be used to simplify configuration in enterprise environments. We begin by describing the configuration interface followed by a brief description of how ACR uses class information to achieve access control. Finally, we summarize the benefits of class-based configuration.

3.3.1 ACR Configuration Interface

The fundamental idea behind ACR is to use the abstract notion of classes to simplify enterprise configuration by establishing logically distinct networks corresponding to each class. In ACR, sources and destinations specify access control policies using classes, and traffic that flows through the network is categorized into different classes. A destination host or service specifies an *access policy* that states the classes of incoming traffic it will accept. Thus, configuration of a source's access control policies is reduced to specifying the class(es) of traffic allowed to originate from that source.

Table 3.1: Access Policy Assignment

Host	Can Receive Packets of Class
Researcher	R, A
Financial Department	F, A
Database 1	D, A
Database 2	E, A
Administrator	A

Table 3.2: Class Membership Assignment

Host	Can Send Packets of Class
Researcher	D,E
Financial Department	D
Database 1	R, F, A
Database 2	R, A
Administrator	A

We motivate this using a simple example. Consider a network where we have two databases 1 and 2, and three entities who wish to access the databases: a researcher, end-users in the finance department and the administrator. While the researcher needs access to both databases, end-users in the finance department are given access to only database 1 and the administrator can access all the resources in the network. For this example, Tables 3.1 and 3.2 specify the *access policy* configuration for each destination and the *class membership* configuration for every source respectively.

We make several important observations. First, specifying access control configuration using classes is trivial both for traffic sources and sinks. From an administrator's standpoint, a class represents an abstract category of traffic on which the administrator can set access and class membership policies. In practice, a class can refer to a particular network service, an end-host, or the current state associated with the packet as it makes its way across the network. Therefore, the translation from access control rules to class assignments need not be unique.

Next, for any given class, it is easy for an administrator to visualize as well as manually verify its configuration settings. Furthermore, all hosts that have similar access control requirements can be grouped into a single class *e.g.* class F for the finance department. This is critical to reduce the number of classes required in the network. Finally, class assignment need not be symmetric; in the example above, packets from researcher to database 1 are marked as class D, and those in the reverse direction are marked with R.

3.3.2 How Does ACR Work?

Considering the same example, we next describe how ACR translates class-based policies to achieve access control. We have again the case of databases 1 and 2, which the administrator determines are able to accept packets of class D and E respectively (Table 3.1). This information is installed in the router(s) to which the databases are connected, and it is propagated by the routing protocol. In the control plane, routers compute *reachability on a per-class granularity.* For example, if the underlying routing protocol is link-state routing, then the link to database 1 is labeled class D and this information is used in the route computation process. Therefore, all routers within the network will establish class D and class E routes to databases 1 and 2.

Next, suppose the researcher is to be given access to the databases. The class membership configuration for a host is stored in its first-hop router. When the researcher's machine generates a packet destined for database 1, the first-hop router examines the class membership list to determine if there exists any class in this list for which a class-based route to database 1 exists. Upon finding a matching class (class D), the first-hop router marks the packet as class D and forwards it along a class-D route. Every intermediary router examines the class header and forwards it along the corresponding class route. At the last-hop router, the label is removed, thus eliminating the need to alter either the researcher's or databases' machines.

Hence, ACR explicitly binds access control with the routing protocol and establishes routes on a class granularity that is in sync with reachability constraints. The role of the data plane is simplified to a straightforward verification of whether the class membership list of the source matches with any of the available class-based routes to the destinations.

3.3.3 Why Classes?

Apart from simplifying access control configuration, distinguishing packets based on logical classes, rather than physical connectivity, provides the following properties:

Network End-to-End Visibility: Since the destinations reachable by a packet are determined by its tag and enforced by routers, a packet that does not have the necessary permission to traverse the logical network in which the destination resides is dropped immediately. Verification is performed as part of the forwarding process, and takes place at all hops along the path.

Route Redundancy: The mechanism to check eligibility of access has shifted from traffic channeling and packet header inspection at firewalls to classification at first-hop routers and verification at all intermediate ones. The usage of classes as a mechanism to enforce access has therefore become orthogonal to the routing process. As a result, it is no longer necessary to constrain path diversity in order to channel packets through middle-boxes, and the network designer is free to add any number of links at any place in the network.

Topology-Independent Middle-box Placement: One of the distinctive features of the edge and core models discussed earlier in §3.2 is the placement of the middle-boxes. The locations of these boxes must be carefully chosen to align with the routing protocols, routing design, and physical topology of the network so that the desired set of packets traverse them. Otherwise, the middle-boxes have to be placed at every point along the network edge.

Current networks are designed with the physical dimension in mind: for a source A and destination B, the middle-box has to be placed along the path A$\rightsquigarrow$B. In ACR, we think in terms of the class dimension: for a source class A and destination class B, the middle-box has to bridge the two domains. In other words, the middle-box forms the "path" between the two points, and if it is the only path, then packets traversing between the two class domains must go through it. Thus, the actual network links traversed, as well as the physical location of the middle-box, no longer matter.

Ease of Understandability: Last but not least, since the network designer works with high-level policies in place of low-level details, the required inputs to the system are easier to understand. This eases the transition phase when another administrator takes over, or when changes need to be made to the network, hence reducing the likelihood of errors.

3.4 ACR Design

In this section, we describe how we realize this abstract notion of multiple logical networks in ACR. Bearing in mind the conditions commonly encountered in networks today, we begin by stating our design space that provides scope and allows us to focus on the most relevant issues. Then, we describe the ACR control and data plane operations in detail.

3.4.1 Design Space and Assumptions

As mentioned earlier, fine-grain access control to individual objects is best implemented on the hosts themselves in application-specific ways, such as through Kerberos [34], TLS [9], or file Access Control Lists (ACLs). At the network-level, however, designers may wish to have a class represent an entire organization, or a particular user, with the decision dependent on the nature of the policies they are trying to implement. Our solution provides that flexibility, and does not place any constraints on actual class assignment.

Next, we assume that routers can be trusted, that physical access to them is restricted and that they are not compromised. These assumptions are all true in typical enterprise networks, though security measures, such as the cryptographic authentication option in OSPF2 [33], can be used to reduce further the probability of successful attacks. In general, routers share fate and thus the bringing down of one is likely to significantly and negatively impact the entire network.

We believe that any solution to network-level access control must not depend on changes to hosts connected to the network, since these hosts may not be under the control of the network administrators. Further, one of the benefits of network-level control is having an independent line-of-defense even when hosts are compromised.

3.4.2 The Control Plane

We begin with the control plane, operations of which include (1) the assignment of classes to hosts, (2) dissemination of access control information by the routing protocol, and (3) installation of classification information at first-hop routers.

As a result of these operations, the control plane establishes the following state in each router. (1) For each destination reachable from that router, the latter will also know what classes the destination is willing to accept. (2) Each router with directly-connected hosts knows the classes each of them are allowed to send. §3.4.3 explains how the first-hop router uses this information to label the packets sent by a directly-connected host with the correct class.

Assignment of Classes

Network designers can implement static policies that specify which entities on their network should be allowed to communicate. A network entity be any of the following: an end-host (defined by MAC or IP address), a group of hosts (defined by IP prefix), or a well-known service (defined by port number). For each entity, the administrator defines the classes the entity is allowed to use when sending packets (the entity's *class membership*) and the classes of the packets the entity is allowed to receive (the entity's *access policy*).

Class Dissemination via Routing

Routing protocols, either link-state, distance-vector or path-vector, need only be slightly modified to account for the use of classes (details in §3.5). Access policies associated with end-hosts are installed at their corresponding first-hop routers and disseminated by the routing protocol in use. For the example in Table 3.1, the subnet to which financial department hosts belong will be advertised with classes F and A.

Next, the routing process is carried out for each class, with intermediate routers storing corresponding forwarding information for each. One of the advantages of integrating routing with access control is the ease with which traffic channeling can be performed. This is important, as middle-boxes that perform deep-packet inspection for worms *etc.*, firewall filtering, or statistics gathering are not useful unless the target traffic is routed through them.

While channelling traffic by physically manipulating network connections may sound simple and straightforward, it is much more complex at the ground level, where machine rooms are often filled with intertwined cables plugged into a multitude of switchers. This complexity increases the likelihood of errors — for instance, the administrator may inadvertently add a link and thus unintentionally allowing traffic to bypass the firewall.

ACR achieves traffic channeling through the concept of *class transformation.* The

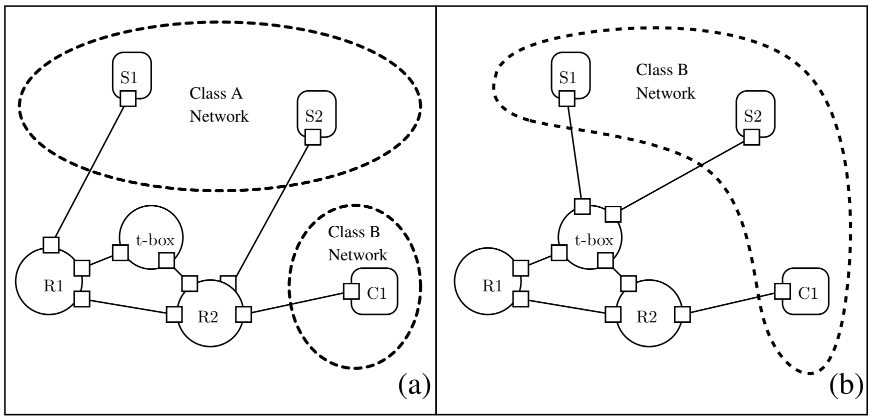


Figure 3.2: T-boxes translate packets between classes. (a) Original access configuration: servers S1 & S2 are in class A, client C1 in class B. (b) (from viewpoint of C1) By re-advertising reachability of S1 and S2 into class B, the t-box puts itself on the path for reaching S1 and S2. Packets from C1 to S1 cannot avoid the t-box, even though alternate physical paths exist, because the only route in Class B to S1 comes from the t-box.

middle-boxes, which we call *class transformation boxes* or t-boxes in short, alter the classes associated with destinations as routes are propagated. Data packets' class tags are also altered accordingly as they are forwarded through these t-boxes. For example, servers that should only receive packets that been through a scrubber t-box will accept packets of class *after-scrubber*. T-boxes that offer the scrubbing service are the only devices allowed to send packets of class *after-scrubber*. When the t-box receives a route to destination in class *after-scrubber*, it reannounces the destination in class *before-scrubber*. Hosts wishing to contact the server then send packets to class *before-scrubber*.

For link-state routing, t-boxes re-advertise reachability of destinations in the new class(es). This is similar to area border routers in OSPF [33], and is illustrated in Figure 3.2, where in (a) one can view the classes as separate logical networks, and (b) from the viewpoint of hosts in a particular network, those in the other are reachable only via the t-box. Thus, there can be multiple t-boxes placed in the network, providing robustness

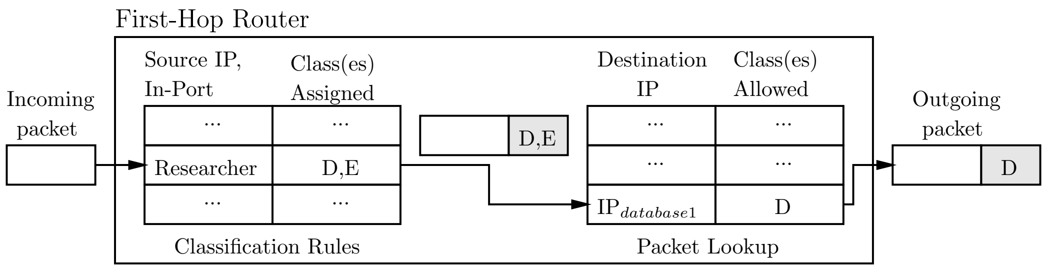


Figure 3.3: Packet forwarding at the first-hop router. The packet is first classified based on source IP, incoming interface port, *etc.*, then the intersection set $\mathcal{I}$ of assigned classes and classes permissible at destination is determined. The packet is dropped if a null set results, otherwise it is tagged with any one class in $\mathcal{I}$ and forwarded.

without reconfiguration if any one fails.[1] However, as before, if both directions of a flow has to traverse a t-box, say for reasons of completeness necessary for flow analysis, then introducing multiple t-boxes with the same functionality may cause the flows to become asymmetrical.

Class Information Installation

Given the current practice in enterprise networks, we expect that most network designers will choose to assign the classes a host can send and receive by configuring this information into the interface/port on the first-hop router when the host is connected. Alternatively, networks that already inventory their hosts' MAC and/or IP addresses might choose to configure classes based on this table.

3.4.3 Data Plane

In our framework, a host's first-hop router (that is, the first router its outgoing packets arrive at) is responsible for labeling the host's packets with an appropriate class marking. We choose this approach for three reasons: (1) thanks to the distribution of class information through the control plane, routers have the required information to know which classes a packet's destination will accept, (2) marking packets at the routers eliminates the need for changes at the hosts, and hosts typically outnumber routers by almost 100 to 1; and (3) routers are assumed to be trusted.

The classification and forwarding of data packets at the first-hop router is shown in Figure 3.3. We begin with classification based on source IP, the interface port the packet arrived on, *etc.*. A packet may be tagged with multiple classes at this time, representing permissions to access various resources. Next, we look up the next hop interface using the destination IP address. If the destination is reachable, the routing table will contain both the next-hop (as normal) and the set of classes the destination accepts (as established by the control plane extensions described in §3.4.2).

If the intersection of the tagged and permissible classes results in a non-null set, the packet is forwarded after inserting in its header one of the classes in the intersection set. Otherwise the packet is deemed not to have permission to reach the destination, and is dropped. For intermediate routers other than t-boxes, no additional classification needs to be performed. Instead, we simply check that the destination is able to accept the traffic

[1]We expect many t-boxes will be stateless, such as packet loggers or inspectors. Even if t-boxes are stateful, such that the failure of a t-box or rerouting means that connections through the t-box will need to be restarted, our proposed system is still more reliable overall that today's networks, where the failure of a t-box typically partitions the network in a manner that end-hosts cannot recover from at all.

class carried by the packet. Finally, as noted before, packets traversing t-boxes may have their class tags altered before being forwarded.

3.4.4 Using ACR in Practice

In the sections above we described the core framework and mechanisms of ACR. This section illustrates how these mechanisms are sufficiently general to handle deployment scenarios that arise in enterprise networks, both common and uncommon ones.

Connectivity to External Networks

ACR provides an easy mechanism to control reachability to external networks. The designer configures the border routers to announce all external routes into the enterprise network with a class *external*. Hosts can then be granted reachability to the outside world simply by giving them permission to send and receive packets with class *external*. In general, there is no longer a need for a rigidly structured DMZ as in the CM model. If designers are concerned about DoS traffic entering their internal networks and congesting links, they still have the option to place firewalls near their borders as described next.

Stateful Firewalls

A very common middle-box in today's networks is a stateful firewall that allows packets into a network only if they are associated with packets that have previously been sent out of the network. The intuition is that the packets sent from inside the network constitute an invitation for the response traffic, and only the response traffic, to enter the network. In the most common incarnation, TCP packets are allowed from outside to inside only if they belong to the same flow as a TCP SYN sent from inside to outside. Some such middle-boxes are also network address translators, modifying the addresses of the packets as they flow through, but this is orthogonal to the stateful firewalling.

Under ACR, the logic inside the middle-box remains exactly as it is today, and they can NAT or not as they choose. The only difference is that under ACR the middle-box will change the class of packets as they flow through. In a typical deployment, the administrator will define classes *internal* and *external*, with the hosts to be protected by the firewall sending and receiving packets of class *internal* and the border routers handling *external* packets as described above. The middle-boxes can now be placed wherever desired in the enterprise network, and the appropriate traffic will be routed through them. Even though there is a single *internal* class, packets from the outside cannot go to an internal host that does not expect them, as the existing logic will ensure that the packet is part of an established and desired flow before mapping it from *external* to *internal*.

Eliminating VLANs

Networks using ACR should no longer need VLANs, as classes serve the same function. Where a designer would have created a VLAN to separate traffic on their network, they can now create a class. As explained in §3.5, ACR can run over link-state, distance vector or path vector routing protocols, so the limitations of the spanning tree algorithms currently used to control VLANs are avoided.

Using ACR to eliminate VLANs will not increase the amount of forwarding state contained in the switches. Today, switches must store a forwarding entry for every MAC address present in the network (otherwise frames are flooded, resulting in terrible perfor-

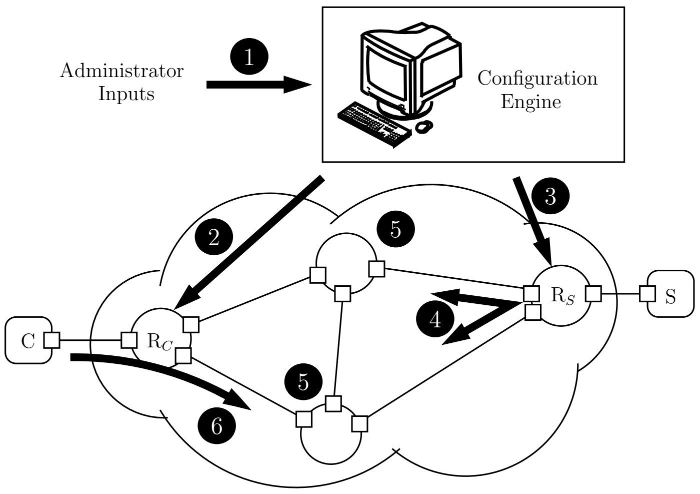


Figure 3.4: High-level view of configuration and automated network-level operations supporting ACR. C is the client machine that intends to communicate with server S. R_C and R_S are the first and last-hop routers respectively.

mance). The worst possible case for ACR is if every host in a subnet belongs to a different class. This prevents any route aggregation, and forces the routing protocol into flat-address routing – each host needs its own route advertised with its own /32 prefix so that it can list the classes it belongs to. Note, however, that this is no more state than the switches are currently storing for forwarding based on MAC addresses.

We next discuss our implementation of ACR, and evaluate the general framework as well as the implementation.

3.5 Implementation

We have implemented a version of ACR on top of the Click modular router code base [32]. Figure 3.4 illustrates the three basic entities in our implementation and how they interact: (a) Configuration engine; (b) First-hop router (which also acts the last-hop router for the destination); and (c) Intermediary router. The configuration engine acts as a centralized controller that takes in configuration inputs and sends out configuration information to the individual routers and firewalls. We first describe the configuration setup, network entities, then a version of ACR link-state routing.

3.5.1 Configuration setup and Network entities

The network administrator inputs access control policies at a single location: the Configuration Engine (CE) (Figure 3.4①). Based on the topological configuration of real world enterprises, a physically-separate control network exists such that the CE can directly connect to all the other entities in the network (routers and firewalls) and specify configuration information for each. Apart from the configuration inputs, the CE needs to be aware of the end-hosts/prefixes that connect to each first-hop and last-hop router. Using the access control rules provided, the CE determines the configuration that is to be distributed to the boundary routers (first and last-hop routers). The current CE implementation assumes that each prefix's subnet connects to a single first-hop router but it can easily be extended to the case where this assumption does not hold. At the first-hop router (Figure 3.4②), the CE specifies the source classes and at the last-hop router (Figure 3.4③), classes associated with the destination prefix(es) are modified. The only intermediary routers that are configured by the CE are those that directly connect to a firewall. Any fine-grain access control specification at the CE is installed at the firewall.

Next, we briefly elaborate on the remaining two entities. The code corresponding to the *Intermediary router* implements the simple ACR route computation and forwarding mechanism described earlier. The *First-hop router* uses the same code base as the Intermediary router with two additional functionalities: (a) identification and marking of the class corresponding to each packet; (b) propagating of the set of prefixes that the first-hop router connects to, and the set of classes associated with each prefix. The firewall in our case is similar to a router except with an additional *access filter list*. In our current implementation, the access filter list supports fine-grain access control rules of the form `<srcAddr, dstAddr, dstPort, protocol, accept/deny>` and does not support any deep-packet inspection rules. In our implementation, only rules that accept packets are used, and the default action is to drop unmatched packets.

3.5.2 Link-State ACR

Next, we describe the propagation of access information by the routing protocol. We implemented a simple version of ACR-based link-state routing where every router maintains the topology of the entire network. Every first-hop router announces a set of prefixes that it connects to, as well as their associated classes (Figure 3.4④). Here, a prefix represents the smallest granularity on which access control policy is applied. For example, if control is applied on a per-host basis, the first hop router advertises /32 IP addresses. Typical announcements can vary between /24 to /32 prefixes depending on the required granularity within the enterprise. While one can extend the implementation to support aggregation of classes across prefixes, our current implementation does not support it.

Route computation: We compute class-based routes on a per-prefix granularity (Figure 3.4⑤). The route computation process is a straightforward per-class shortest-path computation with class information along the final edges for every path. Hence, the route computation overhead is small.

For t-boxes re-advertising reachability, the intended class transformation (*e.g.* from a prefix-class, say 1.2.3.0/24-C to another prefix-class combination, say 1.2.3.0/24-D) is installed by the CE. Thereafter, the t-box first computes the shortest path cost to 1.2.3.0/24 using class C, and advertises reachability with the same cost, but with class D instead.

Forwarding process: At the ingress, or first-hop router, the data packet is tagged with the appropriate class (§3.4.3), or dropped if there is none (Figure 3.4⑥). In general, since changes to end-hosts are not necessary, class information is inserted in the form of a shim layer between the link and IP headers. To minimize the additional overhead of classification at the first hop router, we implemented an efficient hash-based process to perform the set intersection operation of classes. While we maintain forwarding entries on a per-class basis, often the amount of state associated with the forwarding table is very small. For the majority of routes not traversing t-boxes, the routing table can be simplified to maintaining a simple prefix-based routing table and a list of allowed classes per prefix. If t-boxes are to be traversed, this implies the possibility of differing next-hops depending on the class, and hence additional routing table entries.

3.6 Evaluation

In this section, we use a combination of complexity and quantitative measurements to demonstrate that ACR is practical and that it indeed simplifies access control configuration in real-world enterprise networks.

Complexity analysis: To really argue that ACR simplifies access control configuration, we need to show that ACR reduces the amount of configuration work that an administrator needs to perform to achieve a certain objective. However, there exists no standard metric to measure configuration complexity. Here, we introduce a simple metric: the configuration complexity of an access control policy event equals the number of rules in different network entities that an administrator needs to manipulate or check for that event. Based on this metric, we consider a variety of basic but commonly occurring scenarios in enterprise environments and measure the configuration complexity for each scenario across three types of networks: ACR, Core Model and Edge Model (as defined earlier in Section 3.2). We show that across all these scenarios, the configuration complexity for ACR is lower than that of current configuration models (Core and Edge models).

Quantitative Analysis: From a quantitative perspective, we consider the access control policies in four large real-world enterprise networks and demonstrate how these policies can be easily translated to configuration using classes in ACR. The real-world enterprise networks that we consider in our analysis are currently operational large enterprise networks with very tight fine-grain access control requirements. As part of the transformation from access control policies to classes, we show that the resulting number of classes required by ACR in each of the enterprise environments is not high. Finally, we perform simple performance benchmarks on various operations of ACR to show that the performance overhead incurred by ACR is not high.

3.6.1 Complexity Analysis

To perform a complexity comparison between ACR and the Edge and Core Models for controlling reachabilty, we consider three basic and regularly occurring scenarios in enterprise networks: (a) adding a new entity to the network; (b) communication cessation between two entities previously allowed to communicate; and (c) addition of a new link. For each scenario, we compare the *configuration complexity* of making the change for each of the three models (Core, Edge, ACR). We define configuration complexity to be the number of entities in the system whose configuration an administrator needs to manipulate or validate when updating an access control policy.

We use the following parameters in our analysis. We assume that there are a total of n entities for which the network is controlling the reachability, with each entity being defined by source prefix, destination prefix, and/or transport-layer port numbers. Therefore, the maximum number of rules is $O(n^2)$. With respect to the Core model, we use r_c to denote the total number of core routers in the network. Note that in the core model, firewalls are co-located with core routers; hence, the number of firewalls is also assumed to be r_c. Similarly, we use r_e to denote the number of edge routers in the Edge model.

Addition of New Entity

Core: In the core model, adding a new entity (E) to a network like ComNet requires $O(r_c)$ router checks to ensure packets to and from E traverse intended middle-boxes. On the other hand, subnets that can access E will need to know its existence, thus also necessitating $O(r_c)$ reconfiguration.

With regards to the middle-boxes, rules associated with E can be distributed amongst the firewalls, the number of which is expected to be about $O(r_c)$. Even if the network is one-connected, resulting in all complexity being pushed onto a single firewall, there is still the need to verify $O(n)$ rules.

Edge: A network structured like UNet with r_e edge routers would require $O(r_e)$ middle-boxes at the periphery. While there is no longer the need to configure routers to enforce routes when adding a new entity, *all* middle-boxes, where $r_e > r_c$ with high probability, will need to be checked to ensure their rules handle the new entity appropriately.

ACR: Addition of E requires knowledge of the classes E can access and receive packets from. This information is obtained directly from the high-level access policies, and installed just once at E's first-hop router.

Communication Cessation

In this scenario, two entities previously allowed to communicate with each other is now forbidden to do so.

Core: The simplest way of achieving this is to set the appropriate filter to drop packets sent between the two entities. An alternative, or if firewalls are not in use, is to configure the core routers such that reachability information is withdrawn. Therefore, the complexity of the operations remain unchanged, $O(r_c)$ routers and middle-boxes, and $O(n)$ for rules.

Edge: As before, the number of places at which rules have to be altered is $O(r_e)$.

ACR: For ACR, either a source's (S) permission to access a class (C) of destinations is revoked, or a destination ceases to accept packets of class C. For the former, the change can be effected once the first-hop router of the source is notified. Similarly, for the destination that stops accepting class C packets, changes in the last-hop router is sufficient to ensure that no packets tagged with that class is forwarded across the last hop. This is in spite of the convergence time required, during which a packet may *begin* to be forwarded, but will ultimately be dropped along the way.

Addition of Network Link

Core: Adding a new link in the core model may result in unintentional bypassing of middle-boxes. Link weights in the case of OSPF, and route policies for BGP, have to be carefully adjusted to ensure packets take the intended path. Additional middle-boxes may be installed to monitor traffic traversing the new link. In this case, rules may be moved from other boxes resulting in $O(r_c)$ complexity.

Edge: Inserting links in the network itself does not require additional configuration, since the routing protocol automatically takes the new link into account. However, additional links to edge subnets may require duplication of the associated middle-box.

ACR: Since the introduction of a new link triggers routing updates but does not result in destinations being assigned new classes, this event cannot result in hosts' packets reaching unintended destinations. Thus, no additional configuration is required.

Table 3.3 summarizes the complexity of each operation for current configuration models and ACR. We observe that across these three basic scenarios, the configuration

Table 3.3: Complexity Comparison Between Existing Models and ACR

	C.M.	E.M.	ACR
Entity Addition	$O(r_c)$	$O(r_e)$	$O(1)$
Communication Cessation	$O(r_c)+O(n)$	$O(r_e)$	$O(1)$
Link Addition	$O(r_c)$	$O(1)$	$O(1)$

complexity of ACR is significantly better than current configuration methods. While this notion of configuration complexity is not precise, it does provide a way of visualizing how ACR simplifies configuration when compared to the number of additional checks that we need to perform in current systems to achieve access control.

3.6.2 Complexity: Evaluating ACR in Real-world Enterprises

In this section, we evaluate the effectiveness of ACR in real-world enterprise networks. We consider real-world access control policies used in four large and varied enterprise environments,[2] all very closed networks with very tight access control requirements, and we describe how these access control policies can be transformed to configuration using classes in ACR. Our analysis is aimed at answering the following questions:

1. What do access control policies look like in enterprise networks?
2. How do we translate access control policies to class definitions?
3. How many classes would ACR need to achieve access control in these networks?
4. How would we aggregate hosts/services to define classes in ACR?

[2]for reasons of security the names of the networks are not revealed.

Description of Enterprise Networks

In our analysis, we consider four different enterprise networks all that belong to the Core Model with tightly constrained access control policies (*i.e.* each source only has access to a limited set of destinations/services):

ManageNW: The large commercial entity that we study is a huge amalgam of individual enterprises each dedicated to providing a specific functionality. ManageNW is a large management network that forms the overall management backbone of the entity that interconnects management devices in different enterprises. Within ManageNW, there are several access control policies that restrict administrators within a specific enterprise from accessing devices (routers, firewalls) within this backbone or in other enterprises. ManageNW is a large network that serves nearly 65 individual subnets ranging from /16 to /24 prefixes. Within ManageNW, there are tight restrictions on access control across subnets.

CorporateNW: CorporateNW is an internal network consisting of corporate resources such as HR and email systems for various subnets throughout the enterprise. These various subnets consist of multiple /16 prefixes. The access control policies dictate which of the corporate resources can be utilized by groups within the enterprise.

PerimeterNW: PerimeterNW is a security infrastructure network that controls access to internal servers from external networks through the use of proxy servers. Within PerimeterNW there are access control policies that dictate the proxy servers that can communicate with internal servers, this results in rules that are /32 based. PerimeterNW has over 250 individual hosts and almost all of the access control policies are specified at the host level.

AuthNW: AuthNW forms a large authorization network that entities outside of the commercial enterprise use to access resources and services within the enterprise. Given the large number of outside firms that require access to the enterprise, AuthNW is composed of over 70 prefixes (many of which being /24) and nearly 2000 individual internal hosts (not overlapping with the prefixes). Many of these internal hosts are servers, databases, proxies, firewalls, routers that outside entities connect to. Finally, many of the access control policies in AuthNW are at the level of a host or a group of hosts.

Translating Access Control Policies to Classes

There are several possible class assignments that can correspond to the same set of access control policies. We present one such class assignment mechanism which we term *Greedy-aggregation.* First, we identify the basic entities in the system which represents the smallest granularity of prefix (could be a /32 address signifying a host) over which ACL rules are specified in the network.

Once we identify these entities, we model the access control rules in the form of a bipartite graph from source entities (host or prefix) to destination entities (host or prefixes). We then identify aggregate groups of the form $(src - grp, dst - grp)$ where $src - grp$ and $dst - grp$ each represent a group of entities such that any entity within the $src - grp$ can access any in the $dst - grp$. We use a simple greedy algorithm to identify these aggregate groups such that every access control rule is captured in at least one group. For a discussion on an algorithm that reduces the number of classes required for a given access control matrix, please refer to §3.7.2.

Applying ACR to Enterprises

Table 3.4: Number of classes of ACR in real-world enterprises

Enterprise Network	Number of Entities	Number of classes
ManageNW	373	105
CorporateNW	400	140
PerimeterNW	267	40
AuthNW	2110	640

Table 3.4 indicates the total number of classes required by ACR for each of the four enterprise networks using the Greedy-Aggregation based class allocation mechanism. We make three observations. First, as per the Greedy-aggregation approach, the number of classes required by ACR is small. The number of classes should be considered relative to the total number of entities; often, the total number of entities may be much smaller than the number of end-hosts given that many entities may represent prefixes. Second, the number of classes corresponding to a single host is very small. In all these networks, this number ranged typically from 1 to 10. Hence, the class-based routing table at every first hop router is small. Later in our performance benchmark study, we show that the overhead of class-based lookup is minimal. Third, one common trend across all networks is the structure of the aggregated classes. In many of the aggregated groups, the $src-grp$ and the $dst-grp$ often referred to entities in the system which are physically not co-located. The implication of this is that in real enterprise networks, the set of entities that have similar access control policies are often dispersed. Hence, it appears that real world enterprises will greatly benefit from the way ACR decouples the access policies of a host from its IP address, allowing a class to represent directly a logical network of disparate hosts that are not physically co-located and do not share a common IP prefix.

In summary, this analysis shows that ACR can easily be adopted in real-world enterprise networks to provide access control, and that the corresponding number of classes invoked within ACR is also small.

3.6.3 Quantitative: Performance Overhead of ACR

In this section, we describe micro-benchmarks that measure the per-packet processing time in ACR in comparison to a normal routing protocol. These benchmarks are performed using our implementation on a Intel Dual-core Pentium 3.40 Ghz processor machine running Click version 1.5.0.

Table 3.5: ACR forwarding delay as a function of routing table size

Routing Table Size	ACR (μsec)	non-ACR (μsec)
10	1.58	1.19
25	1.60	1.25
50	1.62	1.29
100	1.72	1.34
250	2.23	1.61
500	2.97	2.11
1500	5.46	4.56
3000	9.39	7.77

Table 3.5 shows the average time, in microseconds, for the class-based forwarding engine to process a packet for varying routing table sizes for both ACR and normal routing (non-ACR). For ACR, we analyze the performance from the stand-point of a first-hop router which has to perform class-based lookup and forwarding for every packet. For non-ACR, in comparison, we perform a simple routing table lookup operation. For this ACR benchmark, we installed 20 classes for each source and destination prefix in the routing table. A packet was created with a random destination address (within the specified prefixes) with

a randomly generated genuine class and passed to the class-based forwarding engine. Based on these results we observe that the overhead of class-based lookup is relatively small but not insignificant (roughly $20-30\%$) and this fixed overhead (when computed as a fraction) decreases as the routing table size increases. We view this additional overhead as the tradeoff for having the class-based functionality at the routing layer.

Table 3.6: ACR forwarding delay as a function of number of classes per routing prefix

Number of Classes	Routing Table Size	Time (μsec)
10	1500	5.48
20	1500	5.59
30	1500	5.89
40	1500	6.09
50	1500	6.35

In the previous case, we fixed the number of classes per prefix to be 20 (which is a relatively high number). Table 3.6 shows the average time, in microseconds, for the class-based forwarding engine to process a packet for varying numbers of classes for each routing table prefix while keeping the routing table size constant. We observe that the additional computational overhead due to having more classes per routing table entry is small. The implementation uses an efficient hash-based process to perform the set intersection operation of classes.

To summarize, we showed that ACR scales well in terms of routing table size as well as number of classes. Furthermore, we found that the number of classes and network entities in large networks in practice can be considered small, and should be handled easily by routers today.

3.7 Discussion

In this section we discuss various miscellaneous issues related to ACR. In particular, we focus on the dynamic installation of classes at first-hop routers, and the minimizing of classes used in an enterprise network.

3.7.1 Location-Independent Class Assignment

In §3.4.2, we discussed how classification rules can be installed at first-hop routers based on inputs to the Configuration Engine. In general, end-users may be mobile and connect to the network at different locations, thus requiring the installation of rules to be dynamic. The bootstrapping process involves obtaining IP addresses *etc.* for communication with other network elements, as well as classification rule(s) installation. We describe these next.

We begin by permitting all hosts access to basic services in the network, such as DHCP. We set a default classification rule, which says that all packets that do not match any other rule will be assigned a minimal-service class C_{ms}. The set of services in C_{ms} as usual can be determined by the network administrator, and the corresponding servers must be configured to accept packets tagged C_{ms}. Since end-hosts that have been granted permission to send packets of certain classes may still require access to these basic services later (say to renew DHCP leases), they must continue to have the ability to send class C_{ms} packets.

Next, a logically centralized entity in the network has to install the relevant classification rule. In cases where end-hosts are relatively immobile, this role can be taken on by the Configuration Engine. To accommodate mobile users who authenticate themselves, say via Kerberos [34], the two roles (authentication and configuration) can be co-located at a single server. This co-location allows the end-host to contact a single network entity (Kerberos server) for the sole-purpose of authentication, with the latter installing rules at the first-hop router after a successful authentication. Thus, we eliminate the need for changes in the end-host.

In general, since access control at the network level should involve the administrator as well and not solely the user, the end-host should not be able to directly configure the first-hop router. Furthermore, since the end-host identity should be verified before being granted permission to contact corresponding services, we expect both the authentication and configuration entities to be present. Consequently, the authentication servers are the only network entities that need to be extended to accommodate class-assignment that is independent of the hosts' locations.

3.7.2 Class Optimization

In general, having a smaller set of classes results in faster route computation (and hence convergence) and state required. We now describe an algorithm to reduce the number of necessary classes, and begin with the assumption that at least one physical path exists between any two end-hosts. We model the problem using an *Access Graph*, which is a bipartite graph where each network entity[3] is represented by a unique vertex in each partition. Each entity is represented by a node in partition 1 and a node in partition 2. The former represents the entity's ability to send packets and the latter the entity's ability

[3]where an entity can be a particular user, group of users, traffic of the same port number, *etc.*.

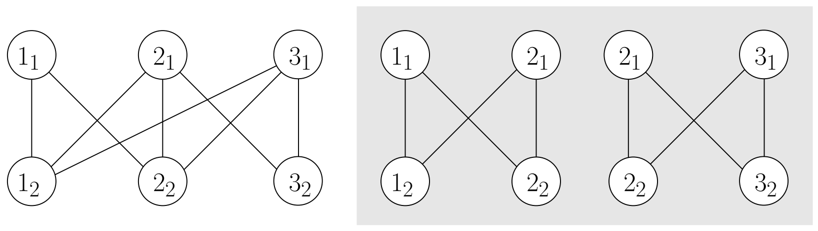


Figure 3.5: From the bipartite graph in (a), there exists two distinct complete bipartite subgraphs (b), with common vertices 2_1 and 2_2 and common edge $(2_1, 2_2)$.

to receive packets. An undirected edge (u_1, v_2) exists if client u's packets are allowed to reach server v, that is, the vertex representing u in partition 1 has an edge to v in partition 2. Since an entity can always communicate with itself, the edge (u_1, u_2) always exists for all vertices u.

One of our primary focuses is on determining distinct complete bipartite graphs (CBG) in this Access Graph. Unlike convention, edges and vertices may belong in different CBGs. Thus in Figure 3.5(a), there are two such distinct graphs (Figure 3.5(b)). Next, the algorithm in Figure 3.6, where the Access Graph is denoted by G_{xs}=(V_{xs}, E_{xs}), is used to determine the minimum number of classes required, and returns that number as well as the set of labeled edges.

The intuition behind the algorithm is as follows: as far as possible, we would want to assign the same class to the set of entities that can communicate with one another, *i.e.* those that form a complete bipartite graph. Thus, we begin by finding combinations of such complete graphs (step 1), where we push as many edges into CBGs that are made as large as possible. Next, for each CBG, we assign the same class, which is distinct from those of other CBGs (steps 7-12), this is possible since every entity is allowed to communicate with the rest in the same CBG. Edges that are present in different CBGs can be assigned

```
1: find set of combinations of sets of graphs $\mathcal{G}_{cbg}$ s.t. $\forall\ G_{cbg} \in \mathcal{G}_{cbg}$, where $G_{cbg}$ consists of CBGs and
   remaining edges, and
     $no\_of\_CBG\_graphs(G_{cbg}) + no\_of\_edges\_left$ is minimal
2: set $G_{win} \leftarrow (\emptyset, \emptyset)$, $least\_class \leftarrow \infty$
3: for each set of graphs $G_{cbg} \in \mathcal{G}_{cbg}$ do
4:    set $G_{tmp} \leftarrow G_{cbg}$
5:    $class\_counter \leftarrow 0$
6:    for each CBG $g_{cbg} \in G_{tmp}$,
        where $no\_of\_incident\_vertices(g_{cbg}) > 2$ do
7:       find unused class $C$
8:       for each edge $(u, v) \in g_{cbg}$ and $u \neq v$ do
9:          set $class(u, v) \leftarrow C$
10:      $class\_counter \leftarrow class\_counter + 1$
11:   for each vertex $u$, in descending order of degree (considering unlabeled edges) do
12:      if $degree(u) = 2$ then
13:         find unused class $C$
14:         for each edge $(u, v)$, $u \neq v$ do
15:            set $class(u, v) \leftarrow C$
16:         $class\_counter \leftarrow class\_counter + 1$
17:      else
18:         if in $G_{tmp}$,
              $\exists$ class $C'$ s.t. $v_1 = u\ \forall\ class(v_1, v_2) = C'$ then
19:            $\forall\ (u, v_2) \in g_{cbg}$, set $class(u, v_2) \leftarrow C'$
20:         else
21:            find unused class $C$
22:            $\forall\ (u, v_2) \in g_{cbg}$, set $class(u, v_2) \leftarrow C$
23:            $class\_counter \leftarrow class\_counter + 1$
24:   if $least\_class > class\_counter$ then
25:      $least\_class \leftarrow class\_counter$
26:      set $G_{win} \leftarrow G_{tmp}$
27: return $least\_class$, $G_{win}$
```

Figure 3.6: Pseudo-code for heuristically determining minimum number of classes required.

either class. For the remaining edges, we use a greedy algorithm and assign the same class to edges incident on vertices (say one of which is v) with the highest degree (steps 13-27). This means that v is allowed to send or receive packets tagged with the same class. If the number of classes used is less than the previous combination of CBGs, we update the winning combination accordingly (steps 28-30). Applying the algorithm to the access graph in Figure 3.5, we will end up with three classes: one each for the CBG graphs in Figure 3.5(b), and another for the edge $(1_2, 3_1)$.

It is not difficult to see that this problem is NP-hard in the general case, so we use a greedy heuristic in obtaining a near optimal assignment. Since the access graph is unlikely to change frequently, we believe it is feasible to compute a near optimal assignment for those networks where it is desirable to minimize the number of classes.

3.8 Related Work

The access control problem, like security, affects multiple layers of the network stack. In VLANs, hosts can be grouped based on 802.1q [26], port, *etc.*. 802.1x has been used in conjunction with EAP [1] and RADIUS [41] to provide port-level access based on end-user authentication. After the user has successfully logged in, packets from the machine are allowed through the first-hop switch. Other than having scalability issues at the link layer, static assignment of tags to packets in VLANs (that is, the tag does not change whilst in transit through the VLAN) unnecessarily constrain the type of state that can be represented. For instance, it is currently impossible to distinguish between packets that have yet to traverse a firewall and those that have, and as we show later it is beneficial to capture this distinction. Also, VLAN membership does not overlap, in the sense that users cannot simultaneously belong to different groups, thus restricting communication patterns in the absence of routing. We believe that the most flexible and scalable solution therefore resides at the routing layer.

SANE [5] and its latter implementation Ethane is an extreme design in the layer 2 space, where all traffic flows have to be vetted by a logically centralized entity (the Domain Controller, or DC) before the path itself is set up by the DC and the flow allowed to begin. Capabilities [2] granted by the DC are used to ensure that compromised routers have minimal impact on the network.

In layer-2.5, MPLS [42] assigns labels to packet flows, and sets up corresponding paths through the network. Thus, the primary use of MPLS is in traffic engineering. Since ACR runs on layer-3, we believe that the two are complementary. Although we also believe that the primary components of ACR, such as classification and destination reachability, can be carried over onto MPLS and enable access control, we think that remaining at layer-3 is in general a more flexible solution.

Rather than work with individual routers, installing and manipulating filters manually when the need arises, 4D [17] proposes a unified decision plane that allows the network designer to describe, at the network level, access policies which are then installed in the form of filters in routers. The availability of network-wide views, provided by a separate discovery plane, reduces the likelihood of misconfigurations. Our approach is similar, in the sense that we advocate configuration at the high-level, and disseminate state subsequently used by routing protocols from a centralized location.

An important recent development is the proposal in IETF to have multi-topology OSPF [39]. Briefly, the authors propose running different virtual topologies on a physical network. The proposal covers primarily low-level implementation information, and is encouraging as it provides an important piece of the overall Access Control Routing solution.

Another network-level method of control is via usage of IPSec [25]. IPSec allows encryption of the data packet, ensuring authenticity and confidentiality, and is primarily used by communicating end-hosts. It can be used in tandem with Access Control Routing, providing finer-granularity control enforced at the end-hosts.

Next, in between the transport and application layers reside SOCKS [28], which facilitates tunneling of packets to a proxy before being forwarded to the original destination. SOCKS requires end-host software installation, end-user configuration using explicit knowledge of the proxy settings, and is thus susceptible to errors. On the other hand, we believe that the redirection of traffic can be simply and cleanly provided by ACR: the proxy is co-located with a t-box, which advertises reachability for the intended destination with the same class as that tagged onto users' packets.

At the application layer, access to network resources can be granted by providing certificates associated with end-users. This authentication process is automatically carried out via a network authentication protocol such as Kerberos [34] whenever, say, the user logs onto a particular Windows Domain.

3.9 Summary

This paper proposes Access Control Routing that uses classes to define virtual networks, and allows clients and servers to separately decide on the networks they can access or receive packets from. Virtual networks can also be defined to correspond to whether a packet has traversed a middle-box, thus class transformation boxes, by bridging two virtual networks, can force traversal of traffic through it. This separation of configuration, and altering of classes rather than routing or topology, simplifies configuration, which we showed qualitatively. Also, analysis of networks in practice showed that the number of classes and routes required for full access control is sufficiently small, and can readily be handled by our software-level router implementation.

Chapter 4

Conclusions and Future Work

In this chapter, we conclude by summarizing our contributions and proposing directions for future work.

4.1 Contributions

The current Internet is managed separately by service providers, that focus primarily on improving the quality of their intra-domain networks whilst maintaining basic connectivity between providers. A fundamental problem with path-vector protocols, considering the lack of visibility into other ASes and global control, is that independent implementation of policies can result in conflicts and route oscillations. We proposed the Precedence Solution, which can provably stabilize routing in the presence of policy disputes. The primary issue, that of storing the routes involved in dispute-based oscillations, can be resolved using a combination of short and long term memories: short term for detection of disputes, and long term in order for previously resolved disputes to remain so.

On the other hand, complete visibility and control within the intra-domain allowed providers to implement more policy controls. However, such controls are added as more of an afterthought to the original network, resulting in multiple types of middle-boxes and even manipulation of the physical network to perform traffic channeling. We proposed a clean slate design, where we showed that Access Control Routing, using the notion of class, can replace the multiple elements that implement that control. By introducing coupling between routing and access control using classes, we show that the amount of configuration required is less, and hence reduces the likelihood of the operator making mistakes.

4.2 Future Work

The usage of classes as a way to aggregate either control or data traffic flows (or perhaps even both) has been shown to be valuable when there is a need to distinguish between different flows. Thus, interesting areas to explore will include applications such as traffic engineering within an ISP, or path differentiation for, say VoIP and video traffic. In general, the limitations of the class-based approach is the number of classes involved: if there are too many ways to classify traffic, the overhead incurred in terms of state may be prohibitively high. On the other hand, in the case of traffic engineering for instance, we believe that having just a few classes may already provide sufficient benefits. Some formal methodology to derive the suitability of using classes for a particular application would be useful.

Bibliography

[1] B. Aboba, L. Blunk, J. Vollbrecht, J. Carlson, and H. Levkowetz. RFC 3748: Extensible Authentication Protocol (EAP), June 2004.

[2] Tom Anderson, Timothy Roscoe, and David Wetherall. Preventing internet denial-of-service with capabilities. *SIGCOMM Comput. Commun. Rev.*, 34(1):39–44, 2004.

[3] A.L. Barabasi and R. Albert. Emergence of Scaling in Random Networks. Science, October 1999.

[4] *Boston University Representative Internet Topology Generator (BRITE).* http://www.cs.bu.edu/brite.

[5] Martin Casado, Tal Garfinkle, Aditya Akella, Michael J. Freedman, Dan Boneh, Nick McKeown, and Scott Shenker. SANE: A Protection Architecture for Enterprise Networks. In *Proc. 15th USENIX Security Symposium*, Vancouver, BC, August 2006.

[6] William Cheswick, Steven Bellovin, and Aviel Rubin. *Firewalls and Internet Security: Repelling the Wily Hacker.* Addison-Wesley Professional Computing Series, 2003.

[7] Cisco Systems. Spanning tree protocol problems and related design considerations. http://www.cisco.com/warp/public/473/16.html Document ID: 10556, Aug 2005.

[8] Jorge A. Cobb, Mohamed G. Gouda, and Ravi Musunuri. A Stabilizing Solution to the Stable Paths Problem. In *Symposium on Self-Stabilizing Systems, Springer-Verlag LNCS*, pages 169–183. ACM Press, 2003.

[9] T. Dierks and E. Rescorla. RFC 4346: The Transport Layer Security (TLS) Protocol, April 2006.

[10] R. Droms. RFC 2131: Dynamic Host Configuration Protocol, March 1997.

[11] Cheng Tien Ee, Vijay Ramachandran, Byung-Gon Chun, and Scott Shenker. Resolving BGP Disputes. Technical Report UCB/EECS-2006-39, EECS Department, University of California, Berkeley, April 13 2006.

[12] K. Egevang and P. Francis. RFC 1631: The IP network address translator (NAT), May 1994.

[13] Nick Feamster, Ramesh Johari, and Hari Balakrishnan. Implications of Autonomy for the Expressiveness of Policy Routing. In *SIGCOMM '05: Proceedings of the 2005 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, New York, NY, USA, 2005. ACM Press.

[14] Lixin Gao. On inferring autonomous system relationships in the internet. *IEEE/ACM Trans. Netw.*, 9(6):733–745, 2001.

[15] Lixin Gao, Timothy G. Griffin, and Jennifer Rexford. Inherently Safe Backup Routing

with BGP. In *Proceedings of IEEE INFOCOM 2001*. IEEE Computer Society, IEEE Press, April 2001.

[16] Lixin Gao and Jennifer Rexford. Stable Internet Routing Without Global Coordination. *IEEE/ACM Transactions on Networking*, 9(6):681–692, 2001.

[17] Albert Greenberg, Gisli Hjalmtysson, David A. Maltz, Andy Myers, Jennifer Rexford, Geoffrey Xie, Hong Yan, Jibin Zhan, and Hui Zhang. A clean slate 4D approach to network control and management. *SIGCOMM Comput. Commun. Rev.*, 35(5):41–54, 2005.

[18] Timothy Griffin and Gordon T. Wilfong. Analysis of the MED Oscillation Problem in BGP. In *ICNP '02: Proceedings of the 10th IEEE International Conference on Network Protocols*, pages 90–99, Washington, DC, USA, 2002. IEEE Computer Society.

[19] Timothy G. Griffin, Aaron D. Jaggard, and Vijay Ramachandran. Design Principles of Policy Languages for Path Vector Protocols. In *SIGCOMM '03: Proceedings of the 2003 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, pages 61–72, New York, NY, USA, 2003. ACM Press.

[20] Timothy G. Griffin, F. Bruce Shepherd, and Gordon Wilfong. The Stable Paths Problem and Interdomain Routing. *ACM/IEEE Transactions on Networking*, 10(2):232–243, April 2002.

[21] Timothy G. Griffin and Gordon Wilfong. A Safe Path Vector Protocol. In *Proceedings of IEEE INFOCOM 2000*. IEEE Communications Society, IEEE Press, March 2000.

[22] Sotiris Ioannidis, Angelos D. Keromytis, Steve M. Bellovin, and Jonathan M. Smith. Implementing a distributed firewall. In *CCS '00: Proceedings of the 7th ACM conference on Computer and communications security*, pages 190–199, New York, NY, USA, 2000. ACM Press.

[23] Aaron D. Jaggard and Vijay Ramachandran. Robustness of Class-Based Path-Vector Systems. In *Proceedings of ICNP'04*, pages 84–93. IEEE Computer Society, IEEE Press, October 2004.

[24] Aaron D. Jaggard and Vijay Ramachandran. Robust Path-Vector Routing Despite Inconsistent Route Preferences. In *Proceedings of ICNP'06*. IEEE Computer Society, IEEE Press, November 2006.

[25] S. Kent and R. Atkinson. RFC 2401: Security architecture for the Internet Protocol, November 1998.

[26] LAN/MAN Standards Committee of the IEEE Computer Society. IEEE Standard for Local and Metropolitan Area Networks - Virtual Bridged Local Area Networks, May 2003.

[27] LAN/MAN Standards Committee of the IEEE Computer Society. IEEE Standard for Local and Metropolitan Area Networks - Port-Based Network Access Control, December 2004.

[28] M. Leech, M. Ganis, Y. Lee, R. Kuris, D. Koblas, and L. Jones. RFC 1928: SOCKS Protocol Version 5, March 1996.

[29] G. Malkin. RFC 2453: RIP version 2, November 1998.

[30] D. McPherson and editors V. Gill. BGP MULTI_EXIT_DISC (MED) Considerations. RFC 4451, March 2006.

[31] Alper Tugay Mizrak. Fatih: Detecting and isolating malicious routers. In *DSN '05: Proceedings of the 2005 International Conference on Dependable Systems and Networks (DSN'05)*, pages 538–547, Washington, DC, USA, 2005. IEEE Computer Society.

[32] Robert Morris, Eddie Kohler, John Jannotti, and M. Frans Kaashoek. The click modular router. In *SOSP '99: Proceedings of the seventeenth ACM symposium on Operating systems principles*, pages 217–231, New York, NY, USA, 1999. ACM Press.

[33] J. Moy. RFC 2328: OSPF version 2, April 1998.

[34] C. Neuman, T. Yu, S. Hartman, and K. Raeburn. RFC 4120: The Kerberos Network Authentication Service (V5), July 2005.

[35] Ruoming Pang, Mark Allman, Mike Bennett, Jason Lee, Vern Paxson, and Brian Tierney. A First Look at Modern Enterprise Traffic. In *Proceedings of Internet Measurement Conference*, October 2005.

[36] Vern Paxson. Bro: a system for detecting network intruders in real-time. *Comput. Networks*, 31(23-24):2435–2463, 1999.

[37] Payment Card Industry (PCI) Data Security Standard version 1.1. `https://www.pcisecuritystandards.org/`, September 2006.

[38] J. Postel. RFC 791: Internet Protocol, September 1981.

[39] P. Psenak, S. Mirtorabi, A. Roy, L. Nguyen, and P. Pillay-Esnault. Multi-Topology (MT) Routing in OSPF. `http://tools.ietf.org/wg/ospf/draft-ietf-ospf-mt/`, November 2006.

[40] Yakov Rekhter, Tony Li, and editors Susan Hares. A Border Gateway Protocol 4 (BGP-4). RFC 4271, January 2006.

[41] C. Rigney, S. Willens, A. Rubens, and W. Simpson. Remote Authentication Dial In User Service (RADIUS). `http://www.ietf.org/rfc/rfc2865.txt`, June 2000.

[42] E. Rosen, A. Viswanathan, and R. Callon. RFC 3031: Multiprotocol Label Switching Architecture, January 2001.

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

[44] *University of Oregon RouteViews Project.* http://www.routeviews.org.

[45] J. H. Saltzer, D. P. Reed, and D. D. Clark. End-to-end arguments in system design. *ACM Trans. Comput. Syst.*, 2(4):277–288, 1984.

[46] João L. Sobrinho. An Algebraic Theory of Dynamic Network Routing. *ACM/IEEE Transactions on Networking*, 13(5):1160–1173, October 2005.

[47] Lakshminarayanan Subramanian, Sharad Agarwal, Jennifer Rexford, and Randy H. Katz. Characterizing the internet hierarchy from multiple vantage points. In *Proc. of IEEE INFOCOM 2002, New York, NY*, Jun 2002.

[48] Lakshminarayanan Subramanian, Matthew Caesar, Cheng Tien Ee, Mark Handley, Morley Mao, Scott Shenker, and Ion Stoica. HLP: a next generation inter-domain

routing protocol. In *SIGCOMM '05: Proceedings of the 2005 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 13–24, New York, NY, USA, 2005. ACM Press.

[49] Hongsuda Tangmunarunkit, Ramesh Govindan, Scott Shenker, and Deborah Estrin. The impact of routing policy on internet paths. In *Proc. of IEEE INFOCOM 2001, Anchorage, AK*, Apr 2001.

[50] Kannan Varadhan, Ramesh Govindan, and Deborah Estrin. Persistent Route Oscillations in Inter-domain Routing. *Computer Networks*, 32(1):1–16, March 2000.

[51] Helen J. Wang, Chuanxiong Guo, Daniel R. Simon, and Alf Zugenmaier. Shield: vulnerability-driven network filters for preventing known vulnerability exploits. In *SIGCOMM '04: Proceedings of the 2004 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 193–204, New York, NY, USA, 2004. ACM Press.

[52] Nicholas Weaver, Dan Ellis, Stuart Staniford, and Vern Paxson. Worms vs perimeters: The case for hardLANs. In *Hot Interconnects*, August 2004.

[53] Geoffrey Xie, Jibin Zhan, David A. Maltz, Hui Zhang, Albert Greenberg, and Gísli Hjálmtýsson. Routing design in operational networks: a look from the inside. In *SIGCOMM '04: Proceedings of the 2004 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 27–40, New York, NY, USA, 2004. ACM Press.