Route Optimization in IP Networks

Jennifer Rexford

Abstract

The performance and reliability of the Internet depend, in large part, on the operation of the underlying routing protocols. Today's IP routing protocols compute paths based on the network topology and configuration parameters, without regard to the current traffic load on the routers and links. The responsibility for adapting the paths to the prevailing traffic falls to the network operators and management systems. This chapter discusses the modeling and computational challenges of optimizing the tunable parameters, starting with conventional intradomain routing protocols that compute shortest paths as the sum of configurable link weights. Then, we consider the problem of optimizing the interdomain routing policies that control the flow of traffic from one network to another. Optimization based on local search has proven quite effective in grappling with the complexity of the routing protocols and the diversity of the performance objectives, and tools based on local search are in wide use in today's large IP networks.

1 Introduction

Over the past fifteen years, the Internet has become a critical part of the world's communications infrastructure. The Internet consists of tens of thousands of domains or *Autonomous Systems* (ASes)—portions of the infrastructure that are each administered by an single institution such as a university, corporation, or Internet Service Provider (ISP). A message sent by one computer typically traverses multiple ASes before reaching its destination, making communication performance depend on the flow of traffic within and between the ASes along the path. In this chapter, we consider the role of optimization in controlling the routing of traffic through the Internet. We explain how the complexity of both the performance objectives and the network routing protocols lead to optimization problems that are not analytically tractable, forcing the use of efficient search techniques to explore a large space of tunable parameters.

1.1 Internet Architecture

Messages sent by one computer (say, a Web client) to another computer (say, a Web server) are divided into individual datagrams or *packets* that flow through the network independently. The decision to base the Internet Protocol (IP) on packet switching has its roots in the early days of the ARPAnet in the late 1960s [1, 2]. In contrast to the traditional *circuit switching* approach of the telephone network, packet

switching acknowledges that data communication is often bursty, with users and applications alternating between periods of high network activity and relative silence. Packet switching has the allure of allowing the links in the network to multiplex traffic across multiple pairs of senders and receivers. While one sender-receiver pair is inactive, another can capitalize on the unclaimed bandwidth by exchanging traffic at a higher rate. However, this flexibility comes at a cost. If too many hosts exchange packets at the same time, the aggregate traffic may overwhelm the link bandwidth, leading to delays in transmitting the data and, ultimately, to lost packets when the buffer feeding the link overflows.

The Internet, then, has a relatively simple *best effort* service model with no guarantee that a data packet reaches its ultimate destination in a timely fashion. Although many applications (such as electronic mail) can tolerate *delay* in receiving the data, *missing* information is often unacceptable. Rather than building reliable, in-order data delivery into IP, the end hosts bear the responsibility of retransmitting lost packets and reconstructing the ordered stream of data at the receiver. These functions are performed by the Transmission Control Protocol (TCP), implemented in the operating system on the end-host computers [3]. In addition, the TCP sender determines the rate of data transmission by monitoring the success (or failure) of sending packets to the receiver. When packets are lost, the sending host decreases the transmission rate to help alleviate the apparent congestion; when packets are successfully delivered, the sending host optimistically increases the sending rate to capitalize on the available bandwidth along the path to the receiver. This decentralized *congestion control* scheme ensures a form of fair sharing of the bandwidth of the links amongst the competing pairs of senders and receivers.

However, transport protocols like TCP do *not* ensure that the network operates *efficiently*. For example, one link may be heavily congested while other links in the network remain lightly loaded. Or, a voice-over-IP (VoIP) call may traverse a long path with high propagation delay when a low-latency path is available. The responsibility of selecting the path a packet follows through the network falls to the routing protocols implemented by the individual routers in the network. Rather than using hard-wired tables to forward the packets, the routers exchange control messages with each other to compute the paths through the network in a distributed fashion. The distributed approach allows a collection of routers to adapt automatically to changes in the network topology. This makes IP networks robust in the presence of link and router failures, and easily accommodates the deployment of new equipment as the network grows. However, the routing protocols deployed in most IP networks do *not* incorporate information about network load and performance into the selection of the paths. Left to their own devices, the routers continue to forward packets over heavily-

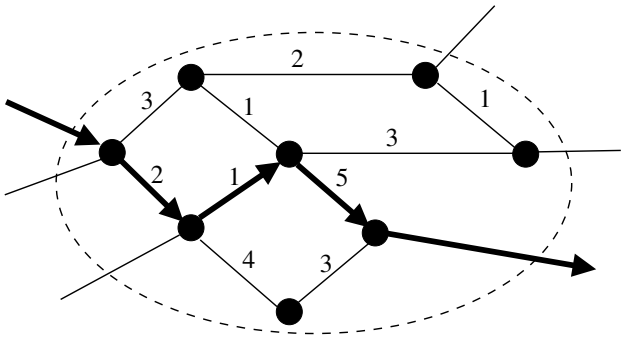

Figure 1: Shortest path routing based on integer link weights

loaded links, leading to packet loss that forces the end hosts to decrease their sending rates in response.

Early forays into delay-based routing protocols led to concerns about the stability of adaptive routing in packet-switched networks. Under load-sensitive routing, the forwarding of data packets would take place on the same small timescale as the delivery of information about the prevailing load on the individual links. Initial experiences in the ARPAnet suggested that acting on out-of-date information about link load causes routers to direct traffic to seemingly underutilized links, leading to congestion that ultimately forces the routers to switch to other paths. Despite some proposals that try to prevent these kinds of oscillation [4], the conventional IP routing protocols adapt only to changes in the network topology and router configuration. While research and standards work continued (and still continues) on more flexible routing protocols [5, 6], practitioners needed an effective way to influence the flow of traffic through their networks. Fortunately, the IP routing protocols have various tunable parameters that the network operators can adjust to change the paths the routers use to forward data packets. For example, the operator can tune the integer link weights the routers use to compute shortest paths for carrying the data traffic, as shown in Figure 1. However, identifying a good setting of these parameters in a large network is challenging for a human operator, setting the stage for the application of optimization techniques.

1.2 Optimization of IP Routing

As an alternative to having the distributed routing protocols adapt to the prevailing traffic, the network operators or automated management systems can modify the configuration of the "static" parameters that drive the operation of the routing protocols [7]. This leads to the "control loop" shown in Figure 2, where measurements of the operational network serve as inputs to a "what if" model that captures the effects of changes

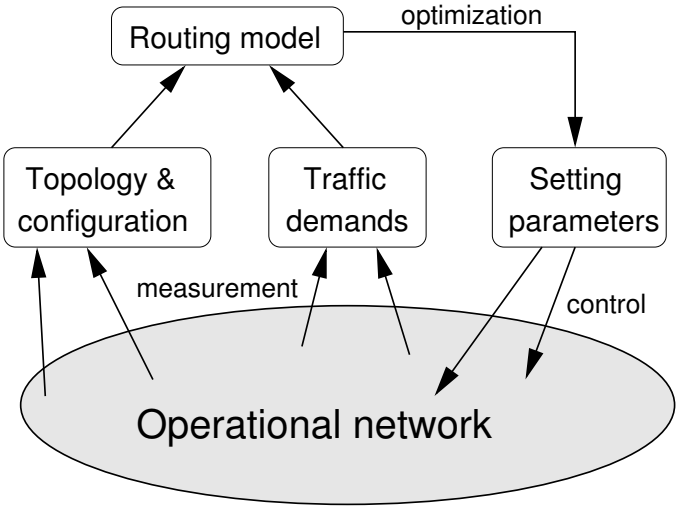

Figure 2: Key components of the route optimization framework

to the tunable parameters. The measurements capture the offered load on the network as well as the current topology, whereas the model predicts the outcome of the distributed path-selection process running on the routers for a given setting of the configurable parameters. This allows the use of optimization techniques for identifying parameter settings that satisfy the network's performance goals. Once good values have been identified, the human operator or automated system can contact the routers to change the configuration of the tunable parameters. Then, the routers collectively act on the new parameter values to compute new paths for forwarding data packets through the network, as predicted by the model.

The approach in Figure 2 has several advantages over extending the routing protocols to adapt automatically to the traffic. First, a network can continue to use the conventional IP routing protocols, rather than deploying or enabling new (potentially complex) protocols that adapt automatically to the prevailing traffic. Second, the network avoids the protocol oscillations that can arise when routers react to locally-constructed (potentially out-of-date) views of the prevailing traffic. Instead, routing changes are carefully planned based on an accurate, network-wide view of the traffic and topology. Third, the routers do not need to maintain statistics about the load on the individual links, disseminate this information throughout the network, or recompute paths as this information changes; avoiding these operations reduces the bandwidth and computational demands on the routers. Fourth, and perhaps most importantly, the selection of routing parameters can depend on a wide variety of performance and reliability constraints that may be difficult to incorporate in a distributed routing protocol. These constraints, and the optimization techniques for satisfying them, can

continue to evolve over time.

Despite these compelling advantages, the resulting optimization problem is difficult to solve because of the complexity of the underlying routing protocols and the diversity of the network objectives. The conventional IP routing protocols were not designed with optimization in mind, and the optimization of the tunable parameters is computationally difficult even for the simplest of metrics (e.g., such as minimizing the maximum link utilization) [8]. The parameter space is quite large, and small changes in the value of a parameter can lead to significant differences in the flow of traffic through the network. In the core the Internet, routing changes in a single AS may affect how traffic leaves that network and enters the next AS along the path to the destination [9]. This introduces additional complexity in modeling the effects of changes to the configurable parameters. Finally, the optimization problem needs to consider numerous objectives, such as link load, propagation delay, and hop count. More complex metrics come in to play as well [10], such as the desire to limit the total traffic directed to each neighboring AS or the frequency of configuration changes as the traffic fluctuates over time. The optimization process may also need to account for the effects of equipment failures and planned maintenance on the suitability of the parameter settings; in addition, multiple routers or links may fail or recover together if they have shared risks, such as a common power supply or optical amplifier.

These practical challenges make it difficult (and frequently impossible) to derive analytical solutions to the optimization problem. Instead, solving the optimization problem depends on having efficient models of the routing protocols and effective ways to explore large parts of the parameter space. To reduce the computational overhead, the "what if" models should not *simulate* the detailed operation of the routing protocols over time as the routers exchange hypothetical control messages. Instead, the models should determine the *outcome* of the routing protocols—the paths the routers would ultimately select for carrying the traffic. In addition, the optimization process should not require recomputing these outcomes from scratch for each candidate setting of the tunable parameters; instead, *incremental algorithms* should be used to minimize the computational overhead for exploring the search space. Finally, experimental results on real input data should be used to drive heuristics that *limit the search space*, in terms of the number of parameters and the range of their values.

In the next section, we explore these principles in the context of optimizing routing inside a single AS, based on conventional shortest-path routing protocols with tunable edge weights. This problem has been studied widely over the past several years and the solutions have been incorporated in management tools

used in many operational IP networks [9, 11, 12]. Next, we describe how to extend the routing model to accurately capture the operation of large backbone networks that have multiple egress points for directing traffic toward each destination. Then, we explore the role of optimization in tuning the interdomain routing policies that control how traffic flows between ASes. In the conclusion, we briefly discuss recent research work on designing distributed routing protocols that are easier to tune as well as centralized approaches for directly computing routes on behalf of the operational routers.

2 Shortest-Path Routing in a Single Network

Many Autonomous Systems (ASes) run routing protocols such as Open Shortest Path First (OSPF) [13] or Intermediate System-Intermediate System (IS-IS) [14] that compute shortest paths based on configurable link weights. In this section, we formulate an optimization problem for selecting the link weights based on the network topology, traffic matrix, and an objective function. On the surface, tuning a single integer weight on each link may not seem flexible enough to satisfy diverse performance objectives. Yet, experimental results suggest that it is possible to achieve near-optimal routing for real topologies and traffic matrices. Although the optimization problem is NP-hard, local search techniques are surprisingly effective in finding good solutions and easily support diverse performance objectives and operational constraints.

2.1 Formulating the Optimization Problem

In conventional intradomain routing protocols, each router is configured with a static integer weight on each of its outgoing links, as shown in Figure 1. The routers flood the link weights throughout the network and compute shortest paths as the sum of the weights. Each router uses this information to construct a table that drives the forwarding of each IP packet to the next hop in its path to the destination. These protocols view the network inside an AS as a graph $G(R, L)$ where each router is a node $r \in R$ and each directed edge is a link $\ell \in L$ between two routers. Each unidirectional link has a fixed capacity c_ℓ, as well as a configurable weight w_ℓ. The outcome of the shortest-path computation can be represented as the proportion $P_{i,j,\ell}$ of the traffic from router i to router j that traverses the link ℓ. In the simplest case, the network has a single shortest path from i to j, resulting in $P_{i,j,\ell} = 1$ for all links ℓ along the path, and $P_{i,j,\ell} = 0$ for the remaining links. More generally, the network may split traffic along multiple shortest paths, resulting in $P_{i,j,\ell} \in [0, 1]$. For example, an OSPF or IS-IS router typically splits traffic evenly along one or more outgoing links along

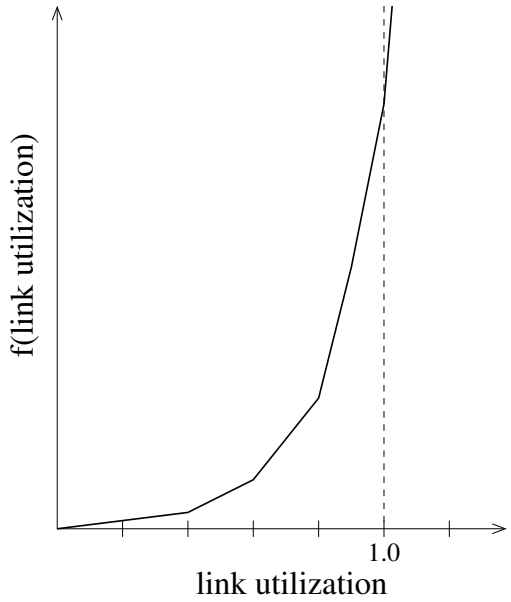

Figure 3: Convex function $f()$ of link utilization u_ℓ/c_ℓ

shortest paths to the destination.

The selection of the link weights $\{w_\ell\}$ should depend on the offered traffic, as captured by a matrix $M_{i,j}$ that represents the rate of traffic entering at router i that is destined to router j. The traffic matrix can be computed based on traffic measurements [15–17] or may represent explicit subscriptions or reservations from users. Given the traffic matrix $M_{i,j}$ and link weights w_ℓ, the volume of traffic on each link ℓ is $u_\ell = \sum_{i,j} M_{i,j} P_{i,j,\ell}$—the proportion of traffic that traverses link ℓ summed over all pairs of routers. An *objective function* can quantify the "goodness" of a particular setting of the link weights. For example, minimizing the simple objective function $\max_\ell(u_\ell/c_\ell)$ would result in a set of link weights that minimizes the worst-case utilization over all links in the network. Although minimizing the maximum utilization is a natural and intuitive objective, this metric is sensitive to individual bottleneck links that may be difficult to avoid under any routing solution. Instead, the optimization could consider a network-wide objective of minimizing $\sum_\ell f(u_\ell/c_\ell)$ for a convex function $f()$ that penalizes solutions that have heavily-loaded links, as shown in Figure 3.

Even for these simple objective functions, the problem of optimizing the link weights $\{w_\ell\}$ is NP-hard [18–20]. The computational challenge arises because of the limited control over the splitting of traffic over multiple shortest path routes. If a router could split traffic over multiple outgoing links in *arbitrary* proportions, the optimization problem would be much simpler—allowing the use of multicommodity-flow

solutions [21]. However, the routers themselves do not compute paths based on any information about the traffic matrix or the link loads, leaving them no basis for determining the appropriate "splits" of traffic over the shortest paths. In addition to the NP-hard optimization problem, the limitations on path splitting make it difficult for an optimal setting of the link weights (if known) to approximate the efficiency of a multicommodity-flow solution with the same topology and traffic matrix. In fact, the performance gap, as measured by the objective function, between an optimal setting of the link weights and a multicommodity-flow solution can be arbitrarily high in the worst case [8, 18]. Fortunately, real network topologies and traffic matrices have much smaller performance gaps between the two kinds of routing [8]; in practice, the performance differences are nearly indistinguishable.

2.2 Using Search Algorithms

The computational complexity of the optimization problem and the desire to support diverse objective functions limit the options for how to solve the optimization problem. Local-search techniques [22] are an attractive way to find a good setting of the link weights subject to an arbitrary objective function. Each iteration of the search evaluates a candidate solution (i.e., an assignment of the weights w_ℓ) and sets the stage for exploring a neighborhood of solutions by changing one, or a few, link weights. Although the search could focus exclusively on neighbors that reduce the value of the objective function, this approach runs the risk of falling in to a local minimum that is much worse than the optimal solution. Instead, the search should consider steps that move away from better solutions, while taking care to avoid repeated evaluation of the same setting of the link weights. For example, tabu search and simulated annealing can generate non-improving moves, and hashing the solutions as they are evaluated can prevent the algorithm for considering the same scenario twice [19].

Each iteration of the local search requires computing the all-pairs shortest paths—the shortest path(s) between each pair of routers—for a single setting of the link weights. In the worst case, this requires running Dijkstra's shortest-path algorithm on the weighted graph to determine the routes $P_{i,j,\ell}$, computing the total traffic $\sum_{i,j} M_{i,j} P_{i,j,\ell}$ on each link ℓ, and evaluating the objective function. For a large network with $|R|$ nodes and $|L|$ edges, the computational overhead can be quite high. Fortunately, the local-search algorithm does not need to compute these quantities from scratch in each iteration. For example, if each iteration of the local search considers a change to a single link weight, incremental graph algorithms can be used to quickly determine the *changes* in the paths, link loads, and objective function. The search can often start

with a reasonable setting of the link weights taken from the existing network; often, changing one or two link weights is sufficient to achieve near-optimal performance. In addition, selecting link weights from a small number of distinct values is an effective way to substantially reduce the search space without much reduction in the quality of the solution.

Studies of various search techniques have shown that it is possible to find near-optimal settings of the link weights on graphs with hundreds of nodes in a reasonable amount of time [8, 23, 24]. In addition, a good setting of the link weights performs almost as well as the theoretical upper bound from solving the multi-commodity flow problem, which assumes greater flexibility in routing the traffic than the shortest-path routing algorithms permit. In practice, the performance gap between a good weight setting and the multi-commodity flow solution is especially small for real network topologies, which typically have a relatively low diameter and a narrow range of link capacities. Although performing the local search is computationally inexpensive for small graphs, identifying good weight settings in larger graphs may argue for an alternate approach. A promising alternative is a two-phase algorithm where the first phase allocates each traffic matrix element to a single path and the second phase tries to find a setting of the link weights that can achieve these paths [25].

2.3 Incorporating Practical Constraints

Using local search to optimize the link weights provides enormous flexibility in addressing the many practical constraints on setting the link weights in operational networks. In particular, minimizing the number and frequency of weight changes is very important to avoid disrupting the network. When a link weight changes, the new information is flooded throughout the network and the routers recompute their shortest paths. During this convergence period, the routers in the network do not have a consistent view of the shortest-path routes for some destinations; in the meantime, packets may be lost or experience long delays. Although recent work has brought the convergence times down considerably [26], delays of several seconds are not uncommon. As such, operators do not change the link weights unless the current routing configuration is causing performance problems.

Robustness to variations in the traffic matrix: The setting of the link weights should not be sensitive to small variations in the traffic matrix M. Existing techniques for measuring or inferring the traffic matrix [15–17] have limitations on their accuracy, and the traffic matrix itself fluctuates over time. The simplest way to allow for uncertainty would be to increase the elements $M_{i,j}$ of the traffic matrix by some target amount.

More generally, the local search algorithm could evaluate a candidate setting of the link weights over a *set* of traffic matrices, favoring a solution that performs well for each traffic matrix over one that performs best for one at the expense of the others. This approach is extremely effective for selecting link weights that accommodate diurnal cycles in the traffic, allowing network operators to have a single assignment of link weights for both daytime and nighttime traffic [10]. In fact, recent work [27] shows that it is possible to find routing solutions that perform well *independently* of the traffic matrix, or across an extremely wide range of traffic demands.

Surviving equipment failures and planned maintenance: Ideally, the assignment of the link weights would be robust to common network disruptions, such as single-link failures. When a link fails, the information is flooded throughout the network and the routers compute new shortest paths over the remaining edges. Preventing congestion after a failure could require the local search algorithm to evaluate a candidate setting of the link weights under each failure scenario [28]. However, evaluating the weights under all failures is computationally prohibitive in large network settings. Fortunately, it is possible to identify and evaluate a much smaller set of critical scenarios [29]. In practice, link weights that perform well on the original network topology continue to perform well after most single-link failures. However, for a few failure scenarios, the link weights may need to change to avoid congestion. Fortunately, changing one or two link weights is typically sufficient to alleviate the congestion. As a proactive measure, the necessary weight changes could be computed in advance of any link failure and stored by the network management system. These same precomputed weight changes are extremely useful when network operators must intentionally "fail" a link in order to fix or upgrade the equipment; in this case, the weight changes can be made in advance, before disabling the equipment for maintenance.

Supporting diverse constraints on path preferences: When there are multiple shortest paths between two routers, the routers along the paths split the traffic over multiple outgoing links—a practice known as equal-cost multi-path routing. Rather than alternating between these links at the packet level, routers typically attempt to forward packets for the same source-destination pair along a single path; this reduces the likelihood that packets from the same TCP connection arrive out-of-order at the receiver. Load-balancing is typically achieved by performing a hash function on the source and destination IP addresses of each packet; the value of the hash function determines which outgoing link should carry the packet. As a result, ties introduce uncertainty in the exact distribution of the traffic over the links; with two outgoing links, the traffic does not necessarily divide exactly in half. The local search can easily account for the effects of uncertainty

by penalizing solutions that have ties (e.g., by setting $P_{i,j,\ell}$ to 0.6 rather than 0.5, implicitly assuming that *both* links must carry 60% of the original traffic load). On the other hand, having multiple shortest paths is advantageous in some settings. In particular, if one of the two links fails, the other shortest path still remains—allowing the network to converge more quickly, causing fewer packet losses and delays [30]. The local search algorithm can easily bias toward solutions with a larger number of ties by including the number of ties in the objective function.

More generally, the use of local search allows the objective function itself to change over time as different metrics reign in importance. For example, for best-effort Internet traffic, minimizing the maximum link utilization (or the sum of $f()$ over all links) is a very effective way to maximize TCP throughput. Yet, the situation is completely different when interactive applications such as Voice-over-IP (VoIP) and video games enter the picture. For these applications, keeping propagation delay low (e.g., below 100 msec) is crucial, and disruptions during routing protocol convergence must be avoided by reducing the frequency of routing changes and by picking solutions that have multiple shortest paths. The underlying machinery of local search is extremely flexible for incorporating complex metrics and changing their importance in evaluating candidate settings to the link weights.

3 Early-Exit Routing in Large Backbone Networks

The vast majority of Internet traffic must traverse multiple Autonomous Systems (ASes) on the way to the destination. Whereas the previous section focused on a single network in isolation, this section considers the additional challenges faced by transit ASes (such as large Internet Service Providers) that connect to other networks in the Internet. Optimizing the link weights in these networks requires a more complex model that captures the effects of *early-exit* routing, as well as optimization metrics that favor solutions that avoid shifting traffic from one exit point to another.

3.1 Destinations With Multiple Egress Points

Although many networks are largely self-contained, transit providers connect to numerous other networks at multiple geographic locations. For example, in Figure 4, AS A allows a customer network (such as a company or university campus) connected to router i to reach destinations (such as Web servers) in ASes B and C. Large transit ASes often *peer* with each other in multiple locations to improve reliability and

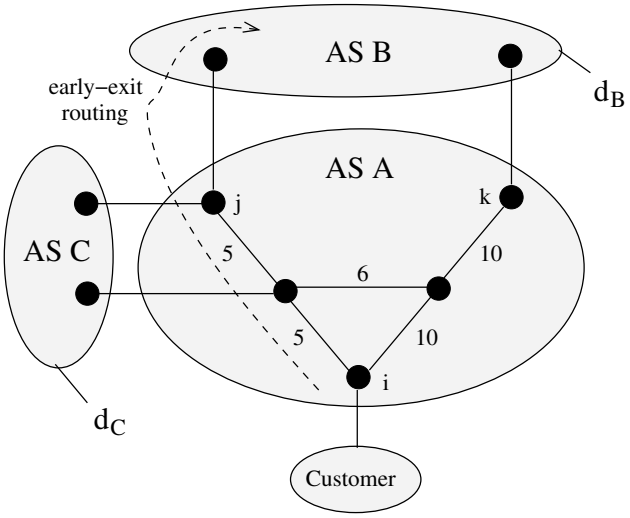

Figure 4: AS A with connections to AS B and AS C

performance. Increasingly, customers often connect to their providers at multiple locations for improved reliability. Each connection consists of two routers—one in each domain—that exchange reachability information using the Border Gateway Protocol (BGP). In essence, BGP is the glue that holds the disparate parts of the Internet together. For example, a router in AS B would advertise a path for destination d_B to the adjacent router in AS A. Since the two networks peer in two locations, the routers in AS A have two possible *egress points* (the links to AS B at routers j and k) for directing traffic toward destination d_B.

At a high level, the two interdomain paths are largely equivalent—for example, both paths traverse the same number of ASes (and, in fact, the same AS) en route to the destination. When a router has two "equally good" interdomain routes to a destination, the BGP routing decision depends on the cost of the *intra*domain path to each egress router. For example, router i would direct traffic destined to d_B via router j, since the intradomain path of cost 10 to router j is shorter than the path of cost 20 to router k. The common practice of *early-exit* or *hot-potato* routing tries to minimize the use of internal network resources by shuttling the traffic to a neighboring AS as early as possible. In fact, AS B would perform early-exit routing in the reverse direction, for the traffic from d_B to the customer in AS A's network, leading to the asymmetric routing where traffic from the customer leaves AS A via router j and traffic to the customer enters via router k. Given the many destinations spread throughout the Internet, a transit AS would have to choose between multiple egress points for a large portion of the destinations, making it crucial to capture the effects of early-exit routing on the flow of traffic.

3.2 Modeling the Effects of Early-Exit Routing

Optimizing the intradomain link weights requires several extensions to the model presented earlier in Section 2 [9, 31]:

- **Egress links connecting to neighboring domains:** The graph $G(R, L)$ needs to include the external links that connect to routers in neighboring ASes. The objective function should consider the traffic load u_ℓ on these links when evaluating candidate settings of the weights on the internal links.

- **Set of egress links for reaching each destination d:** The graph can include a logical node to represent the set of egress links E_d available for reaching each destination d. For example, the logical node representing d_B would have logical links to the two border routers in AS B. Since these logical links are not real, the objective function does not need to consider the traffic load imparted on these links.

- **Offered traffic from ingress points to destinations:** Since the setting of the link weights affects how a router selects an egress point, the traffic matrix is not an appropriate representation of the offered traffic. Instead, the traffic demand $v_{i,d}$ of load from ingress point i to destination d is the correct abstraction [15].

- **Selection of the closest egress point:** Evaluating a candidate setting of the link weights requires comparing the intradomain path costs from ingress i to each egress point in E_d. This comparison is necessary to identify the "closest" egress point $o_{i,d}$ for ingress point i; the traffic ultimately flows over the shortest path to the closest egress point.

In the extended model, the load u_ℓ on each external link ℓ is the sum over all traffic demands $v_{i,d}$ that have a closest egress point $o_{i,d} = \ell$. For the internal links ℓ, the traffic load comes from the traffic $v_{i,d}P_{i,j,\ell}$ such that the closest egress point $o_{i,d}$ is incident to router j.

On the surface, the extended model seems much more complicated than the model in Section 2. For example, the routers in large backbone networks typically have routes for more than 150,000 distinct destinations (blocks of contiguous IP addresses). In addition, a network with n edge routers could conceivably have 2^n unique sets of egress routers, and even more sets of egress links. In practice, though, many destinations have the same set of egress routers, and the number of distinct egress sets is much smaller than 2^n [32]. For example, a neighboring AS typically advertises comparable BGP routing information over all

of its peering links, resulting in a single egress set for of these destinations. In essence, the optimization process can treat the group of related destinations as a single destination d with one larger traffic demand $v_{i,d}$ per ingress point. This reduces the complexity of evaluating a candidate setting of the link weights. In addition, as in Section 2, the size of the search space can be reduced substantially by exploring incremental changes to the link weights, a small set of unique weight values, and so forth.

3.3 Avoiding Changes in the Chosen Egress Point

In a transit network with early-exit routing, a change in the link weights can affect where traffic leaves the network. In Figure 4, ingress router i initially directs traffic via egress point j with the smaller intradomain path cost of 10. However, a link failure could increase the cost to reach j from 10 to 21, making egress point k the closer egress point with a path cost of 20. Hence, when the failure occurs, router i would shift all traffic destined to d from egress point j to egress point k. In practice, small internal changes, due to a link failure or a weight change, can cause a router to shift the routes for tens of thousands of destinations at once [33]. This, in turn, can dramatically affect the flow of traffic in the network, and the offered load on the neighboring domain, resulting in serious congestion. To avoid these undesirable disruptions, these effects should be taken into account when designing the topology or planning changes to the network.

For example, in Figure 4 the router i selects the egress point j (with a path cost of 10) over egress point k (with a path cost of 20) to direct traffic toward d_B. However, if the left link from router i needs to be disabled temporarily for maintenance purposes, the path cost to j would increase to 21 ($= 10 + 6 + 5$), making k the closer egress point. This would cause i to change its routing decision and direct traffic via router k. The routing change can be avoided by changing the weight of the middle link from 6 to 4 before the maintenance activity; this ensures that the path to i has cost 19—smaller than the cost to k. The goal of avoiding unnecessary early-exit routing changes leads to a new optimization problem to identify changes to the existing link weights that would allow a link (or router) to be removed from the network with the minimal disruption to the existing traffic. More generally, network operators can use models of early-exit routing to identify particular links and routers which would cause large disruptions when they fail [31]. Optimizing the link weights to minimize the likelihood of such disruptions across a set of failure scenarios is useful for avoiding the performance problems that arise when unexpected early-exit routing changes occur.

4 Interdomain Routing Policies

The flow of traffic through a transit network depends on the BGP routes advertised by neighboring ASes, as well as the local policies configured on the individual routers. In this section, we consider how tuning the BGP policies influences the forwarding of data traffic. Next, we describe how to extend the routing model from the previous section to capture the role of routing policies. Then, we discuss approaches to exploring the very large search space of BGP policy configurations, as well as fundamental limitations on the ability of models to predict the load on the links in an AS.

4.1 BGP Policies Influencing the Path-Selection Process

In the simplest case, the routers in an AS select BGP routes with the shortest AS path, breaking ties based on the proximity of the egress points. More generally, a router can be configured to apply a policy that assigns a *local preference* to a route, to select one route over another with a shorter AS path or closer egress point. Today's routers provide an extremely flexible "programming language" for specifying rules for assigning the local-preference attribute. For example, the policies may differentiate between routes learned from different neighboring ASes, based on the commercial relationship. In transit networks, the common practice is to assign a higher preference to BGP routes learned from customers than to routes learned from upstream providers, to ensure that data traffic traverses neighbors that are paying customers, even if the path through the provider is shorter [34, 35], as shown in Figure 5. Similarly, a network may assign a lower local preference to BGP routes learned over low-bandwidth links that exist only to provide backup service [36], to ensure that data traffic traverses the high-bandwidth primary links except when they have failed.

In addition to assigning preferences based on the relationship with the neighboring AS, operators configure policies to influence the load on the network links [32, 37, 38]. For example, suppose an AS learns BGP routes to a destination from two upstream providers. By assigning a lower local preference to one route, the AS decides to direct traffic via the route learned from the other provider. Careful assignment of local preference over the range of destinations is helpful in balancing the load on the two links. In some cases, one provider might charge a higher price for sending traffic, or generally have poorer performance. To reduce financial cost or improve performance, the AS may tend to prefer routes through the other provider, up to the point where the link becomes too heavily loaded. More generally, an AS may connect to a single AS in multiple geographic locations. If one of the links to the neighboring AS becomes congested, the

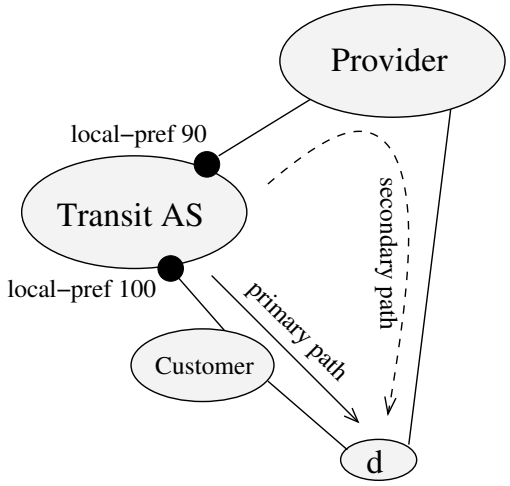

Figure 5: Transit AS assigning higher local preference to the route through a customer

operators may reconfigure the adjacent router to assign a lower local preference value to some of the BGP routes learned at this location. This ensures that the routers in the AS direct the traffic for these destinations through the other, lightly-loaded links to the same AS.

4.2 Modeling the Effects on the Flow of Traffic

Capturing the effects of routing policies requires extensions to the model presented earlier in Section 3 [39]. By changing how the local-preference attribute is set, tuning the BGP policies affects how a router at the periphery of the network selects a best route. This, in turn, affects the egress set for each destination address block (or *prefix*). Modeling the influence of BGP policies on the flow of traffic requires:

- **BGP-learned routes from neighbor ASes:** The input data for the computation is the set of BGP-learned routes from neighboring ASes. In practice, the BGP-learned routes can be captured by "dumping" the BGP routing table at each router, or monitoring the BGP routes as they are advertised by the neighbors.

- **Specification of routing policies:** A BGP routing policy is a sequence of clauses, where each clause specifies a set of routes (e.g., based on the destination prefix or the elements in the AS path) and assigns a local-preference value to the matching routes. For example, a policy might assign a local preference of 100 to all routes with AS "1234" as the second hop, and 90 to all others.

- **Model of BGP path selection:** Computing the egress set requires applying the policies to the BGP routes learned from neighboring demands, and then selecting the "best" of the modified paths. In particular, the model should select routes with the highest local preference and, among those, the routes with the shortest AS path length. Ultimately, each router would select the route with the "closest" egress point.

- **Offered traffic from ingress points to destination:** As in Section 3, estimating the flow of traffic requires overlaying the traffic demands $v_{i,d}$ on top of the paths through the AS to reach the chosen egress point for each ingress point and destination prefix.

With an accurate view of the BGP-learned routes and the traffic demands, an optimization tool can conduct a local search over possible changes to the routing policies and evaluate the influence on the flow of traffic through the network.

4.3 Limiting the Search Space

The significant flexibility in specifying BGP routing policies, coupled with the large number of destination prefixes, makes the search space extremely large—far too big to explore exhaustively. Fortunately, heuristics are helpful for limiting the overhead of the optimization process [32]:

- **Exploring incremental policy changes:** As in Section 2, the local search can explore incremental changes to the router configuration, rather than considering all possible policies from scratch. For example, the optimization could, on noting that one edge link is congested, focus on modifying the policy in this one location to assign a smaller local preference to the BGP routes for some destinations. This approach not only limits the search space but also reduces the overhead of evaluating the effects of the configuration change, since the optimization tool need only consider the traffic demands that *move* to new egress points.

- **Focusing only on popular destination prefixes:** A typical IP backbone network has BGP routes for more than 150,000 destination address blocks. In practice, the vast majority of these destinations contribute receive only a small fraction of the traffic [40]. Changing the local-preference values for a small handful of popular destination prefixes would move a significant amount of traffic from one egress point to the other, while minimizing the number of routing changes and reducing the local-search overhead. In addition, these popular destinations tend to have more stable candidate BGP

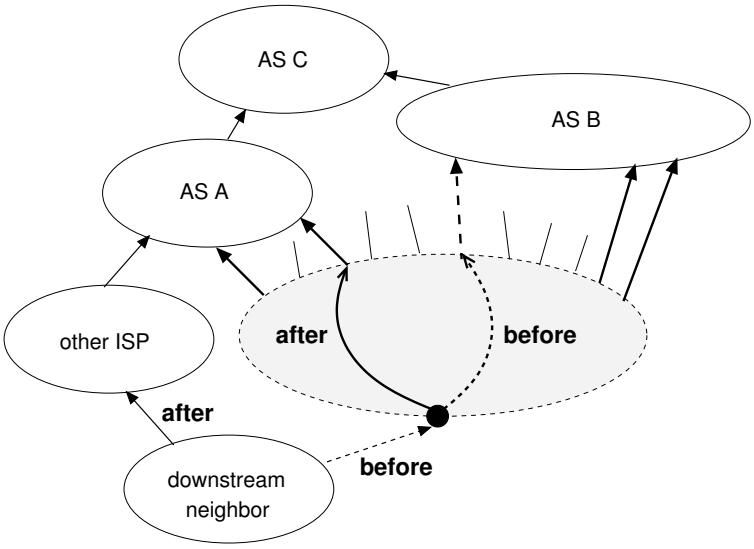

Figure 6: Change in neighbor's behavior upon receiving a new BGP route

routes [41], making it likely that the new BGP policy configuration continues to be effective in the future.

- **Focusing on groups of destination address blocks:** In practice, a network has the same (or similar) BGP routing options for multiple destination prefixes. For example, a large customer may have a dozen blocks of addresses that are advertised exactly the same way, at the same locations in the network. Specifying routing policies that match on the *group* of destinations (say, by matching on common aspects of their AS paths) reduces the search space, and also makes the routing in the network less sensitive to small changes in other route attributes. In addition, groups of destinations tends to have a more stable, aggregate traffic volume, making the predictions of the traffic load in the network more accurate.

A combination of these three techniques substantially reduces the overhead of exploring the search space and tends to favor robust solutions.

However, in some cases, the impact of a BGP policy change may be hard to predict because of side effects on neighboring domains [32]. In the example in Figure 6, ASes A and B both advertise paths to destinations in AS C. Initially, there are five "best" routes—two via AS A and three via AS B. Routers on the west coast route via AS A and routers on the east coast route via AS B. Suppose that the leftmost link to AS B is congested (as illustrated by the dashed line), and the BGP policy for this egress point is modified

to assign a lower local preference to routes originating from AS C. After this change, some routers might switch from a route via the leftmost link to AS B to a route via the rightmost link to AS A. These routers would advertise the new best path to downstream neighbors. Depending on the neighbor's routing policies, the new advertisement might cause the neighbor to select a different next-hop AS (i.e., another ISP) for reaching this prefix. This could result in an unpredictable decrease in the volume of traffic entering the domain at this router. Similarly, the routing change could trigger an increase in traffic if other neighbors preferred the (A, C) route over the (B, C) route. These kinds of side effects make it difficult to predict how the change in routing policy would affect the volume of traffic on the network links.

To prevent the effects of routing changes from propagating to neighboring domains, the optimization tool should, whenever possible, achieve the load-balancing goals by adjusting routing policies for destinations for which every potential best route has the *same AS path*. In practice, an AS would have many such destinations. For example, a destination residing in AS B would be reachable through three egress points to AS B that offer the same one-hop AS path. Changing the routing policy for one (or more) of those destination prefixes would move traffic away from the dotted link to one of the other links to AS B, without changing the AS path seen by the downstream neighbor. Depending on the BGP implementation, the downstream AS might not even receive a new BGP advertisement, since the AS path has not changed. This allows the network to alleviate congestion on a link by moving traffic to other, lightly-loaded links, without changing the BGP routing information sent to neighboring domains.

5 Conclusion

IP routing protocols have tunable parameters that operators can set to control the flow of traffic through their networks. Optimization based on local-search techniques plays an important role in adapting these parameters to the prevailing network conditions. In this chapter, we considered three variants of the optimization problem, with increasing complexity:

1. When each destination connects to the network at a single location, the optimization problem consists of setting the link weights that drive how the routers direct traffic on shortest paths. The inputs to the optimization problem are the traffic matrix and the capacitated network topology.

2. When some destinations are reachable via multiple egress points, the optimization problem becomes more complicated. Instead of the traffic matrix, the offered load is represented as a set of traffic

demands—the volume of traffic entering at a certain router and traveling to a particular destination. The optimization problem must also consider the set of egress points for each destination prefix.

3. In addition to selecting the link weights, the optimization problem can consider changes to the BGP policies that determine the set of egress points for each destination prefix. To determine the egress set, the optimization must consider how the configurable routing policies assign the local-preference attribute of the BGP routes learned from neighboring domains.

For each optimization problem, local-search techniques are effective in finding good solutions that support diverse performance objectives and operational constraints.

Still, the optimization problems are computationally challenging, often requiring heuristics for limiting the size of the search space. Arguably, the underlying routing protocols were not designed with optimization in mind. Just as an understanding of the routing protocols leads to interesting optimization problems, a deeper understanding of optimization techniques could lead to better routing architectures. Initial forays into redesigning IP routing with optimization in mind include:

- **Routing protocols with simpler optimization problems:** Optimizing the link weights in a shortest-path routing protocol is an NP-hard problem, as discussed in Section 2. The work in [42] considers a simple variant of OSPF and IS-IS routing, where the routers forward traffic on paths in *inverse proportion* to the sum of the weights, rather than sending all traffic on shortest paths. This small change leads to an optimization problem that can be solved in polynomial time for simple objective functions, as well as a protocol that has smaller reactions to small changes in the path costs.

- **Explicit negotiation between neighboring domains:** When a configuration change causes a router to direct traffic to a different egress point, the neighboring domain starts receiving traffic at a different ingress point. Rather than make configuration changes independently, neighboring ASes could coordinate their activities to find mutually beneficial ways to direct the traffic that traverses both domains [43, 44]. However, this approach depends on having an effective way to satisfy the objectives of both ASes, while ensuring that neither domain has an incentive to provide misleading information to the other.

- **Logically-centralized control over path selection:** The routing protocols in today's IP networks were designed, first and foremost, to be implemented in a distributed fashion. More recently, the in-

creasing capabilities of computers makes it possible to select the paths for a large collection of routers in a separate platform with a network-wide view of the topology and traffic [45–47]. Rather than emulating today's distributed protocols, these platforms could define new frameworks for computing paths in a logically-centralized fashion to satisfy network engineering goals directly.

Designing new IP routing architectures that are more amenable to optimization is a promising avenue for future research.

References

[1] B. M. Leiner, V. G. Cerf, D. D. Clark, R. E. Kahn, L. Kleinrock, D. C. Lynch, J. Postel, L. G. Roberts, and S. S. Wolff, "The Past and Future History of the Internet," *Communications of the ACM*, vol. 40, pp. 102–108, February 1997.

[2] D. D. Clark, "The design philosophy of the DARPA Internet protocols," in *Proc. ACM SIGCOMM*, pp. 106–114, August 1988.

[3] W. R. Stevens, *TCP/IP Illustrated, Volume 1: The Protocols*. Addison-Wesley, January 1994. ISBN 0201633469.

[4] A. Khanna and J. Zinky, "The revised ARPANET routing metric," in *Proc. ACM SIGCOMM*, pp. 45–56, September 1989.

[5] S. Chen and K. Nahrstedt, "An overview of quality of service routing for next-generation high-speed networks: Problems and solutions," *IEEE Network Magazine*, pp. 64–79, November/December 1998.

[6] E. Crawley, R. Nair, B. Rajagopalan, and H. Sandick, "A framework for QoS-based routing in the Internet," Request For Comments 2386, IETF, August 1998.

[7] B. Fortz, J. Rexford, and M. Thorup, "Traffic engineering with traditional IP routing protocols," *IEEE Communication Magazine*, October 2002.

[8] B. Fortz and M. Thorup, "Internet traffic engineering by optimizing OSPF weights," in *Proc. IEEE INFOCOM*, March 2000.

[9] A. Feldmann, A. Greenberg, C. Lund, N. Reingold, and J. Rexford, "NetScope: Traffic engineering for IP networks," *IEEE Network Magazine*, pp. 11–19, March 2000.

[10] B. Fortz and M. Thorup, "Optimizing OSPF/IS-IS weights in a changing world," *IEEE Journal on Selected Areas in Communications*, vol. 20, pp. 756–767, May 2002.

[11] Cariden MATE Framework. `http://www.cariden.com/products/`.

[12] OpNet SP Guru. `http://www.opnet.com/products/spguru/home.html`.

[13] J. Moy, "OSPF Version 2," Request For Comments 2328, IETF, April 1998.

[14] R. Callon, "Use of OSI IS-IS for Routing in TCP/IP and Dual Environments," Request For Comments 1195, IETF, December 1990.

[15] A. Feldmann, A. Greenberg, C. Lund, N. Reingold, J. Rexford, and F. True, "Deriving traffic demands for operational IP networks: Methodology and experience," *IEEE/ACM Trans. Networking*, vol. 9, June 2001.

[16] A. Medina, N. Taft, K. Salamatian, S. Bhattacharyya, and C. Diot, "Traffic matrix estimation: Existing techniques compared and new directions," in *Proc. ACM SIGCOMM*, August 2002.

[17] Y. Zhang, M. Roughan, N. Duffield, and A. Greenberg, "Fast, accurate computation of large-scale IP traffic matrices from link loads," in *Proc. ACM SIGMETRICS*, June 2003.

[18] D. H. Lorenz, A. Orda, D. Raz, and Y. Shavitt, "How good can IP routing be?," Tech. Rep. 2001-17, DIMACS, May 2001.

[19] B. Fortz and M. Thorup, "Increasing Internet capacity using local search," *Computational Optimization and Applications*, vol. 29, no. 1, pp. 13–48, 2004.

[20] A. Juttner, A. Szentesi, J. Harmatos, M. Pioro, and P. Gajowniczek, "On solvability of an OSPF routing problem," in *Proc. Nordic Teletraffic Seminar*, 2000.

[21] Z. Wang, Y. Wang, and L. Zhang, "Internet traffic engineering without full mesh overlaying," in *Proc. IEEE INFOCOM*, April 2001.

[22] E. Aarts and J. Lenstra, *Local Search in Combinatorial Optimization*. Wiley-Interscience, 1997.

[23] M. Ericsson, M. Resende, and P. Pardalos, "A genetic algorithm for the weight setting problem in OSPF routing," *J. Combinatorial Optimization*, vol. 6, no. 3, pp. 299–333, 2002.

[24] L. S. Buriol, M. G. C. Resende, C. C. Ribeiro, and M. Thorup, "A memetic algorithm for OSPF routing," in *Proc. INFORMS Telecom*, pp. 187–188, 2002.

[25] M. Pioro, A. Szentesi, J. Harmatos, A. Juttner, P. Gajowniczek, and S. Kozdrowski, "On OSPF related network optimisation problems," in *Proc. IFIP ATM IP*, July 2000.

[26] C. Alaettinoglu, V. Jacobson, and H. Yu, "Towards milli-second IGP convergence." Expired Internet Draft, draft-alaettinoglu-isis-convergence-00.txt. `http://www.packetdesign.com/Documents/convergence.pdf`.

[27] D. Applegate and E. Cohen, "Making intra-domain routing robust to changing and uncertain traffic demands: Understanding fundamental tradeoffs," in *Proc. ACM SIGCOMM*, August 2003.

[28] A. Nucci, B. Schroeder, S. Bhattacharyya, N. Taft, and C. Diot, "IGP link weight assignment for transient link failures," in *Proc. International Teletraffic Congress*, August 2003.

[29] B. Fortz and M. Thorup, "Robust optimization of OSPF/IS-IS weights," in *Proc. International Network Optimization Conference*, pp. 225–230, October 2003.

[30] A. Sridharan, S. B. Moon, and C. Diot, "On the correlation between route dynamics and routing loops," in *Proc. Internet Measurement Conference*, October 2003.

[31] R. Teixeira, A. Shaikh, T. Griffin, and G. Voelker, "Network sensitivity to hot-potato disruptions," in *Proc. ACM SIGCOMM*, September 2004.

[32] N. Feamster, J. Borkenhagen, and J. Rexford, "Guidelines for interdomain traffic engineering," *ACM SIGCOMM Computer Communication Review*, vol. 33, October 2003.

[33] R. Teixeira, A. Shaikh, T. Griffin, and J. Rexford, "Dynamics of hot-potato routing in IP networks," in *Proc. ACM SIGMETRICS*, June 2004.

[34] G. Huston, "Interconnection, peering, and settlements," in *Proc. INET*, June 1999.

[35] L. Gao and J. Rexford, "Stable Internet routing without global coordination," *IEEE/ACM Trans. Networking*, vol. 9, pp. 681–692, December 2001.

[36] L. Gao, T. G. Griffin, and J. Rexford, "Inherently safe backup routing with BGP," in *Proc. IEEE INFOCOM*, April 2001.

[37] B. Quoitin, S. Uhlig, C. Pelsser, L. Swinnen, and O. Bonaventure, "Interdomain traffic engineering with BGP," *IEEE Communication Magazine*, May 2004.

[38] T. Ye and S. Kalyanaraman, "A recursive random search algorithm for large-scale network parameter configuration," in *Proc. ACM SIGMETRICS*, June 2003.

[39] N. Feamster, J. Winick, and J. Rexford, "A model of BGP routing for network engineering," in *Proc. ACM SIGMETRICS*, June 2004.

[40] W. Fang and L. Peterson, "Inter-AS traffic patterns and their implications," in *Proc. IEEE Global Internet*, December 1999.

[41] J. Rexford, J. Wang, Z. Xiao, and Y. Zhang, "BGP routing stability of popular destinations," in *Proc. Internet Measurement Workshop*, 2002.

[42] J. Fong, A. Gilbert, S. Kannan, and M. Strauss, "Better alternatives to OSPF routing," in *Proc. Workshop on Approximation and Randomized Algorithms in Communication Networks*, 2001.

[43] R. Mahajan, D. Wetherall, and T. Anderson, "Negotiation-based routing between neighboring ISPs," in *Proc. USENIX Symposium on Networked Systems Design and Implementation*, May 2005.

[44] J. Winick, S. Jamin, and J. Rexford, "Traffic engineering between neighboring domains." `http://www.cs.princeton.edu/~jrex/papers/interAS.pdf`, July 2002.

[45] N. Feamster, H. Balakrishnan, J. Rexford, A. Shaikh, and J. van der Merwe, "The case for separating routing from routers," in *Proc. ACM SIGCOMM Workshop on Future Directions in Network Architecture*, August 2004.

[46] O. Bonaventure, S. Uhlig, and B. Quoitin, "The Case for More Versatile BGP Route Reflectors." Internet Draft draft-bonaventure-bgp-route-reflectors-00.txt, July 2004.

[47] A. Farrel, J.-P. Vasseur, and J. Ash, "Path computation element (PCE) architecture." Internet Draft draft-ietf-pce-architecture-00.txt, March 2005.