Dynamic Route Computation Considered Harmful

Matthew Caesar
University of Illinois at Urbana-Champaign
caesar@cs.illinois.edu

Martín Casado
Nicira Networks Inc.
casado@nicira.com

Teemu Koponen
Nicira Networks Inc.
koponen@nicira.com

Jennifer Rexford
Princeton University
jrex@cs.princeton.edu

Scott Shenker
University of California at Berkeley
shenker@cs.berkeley.edu

ABSTRACT

This paper advocates a different approach to reduce routing convergence—side-stepping the problem by avoiding it in the first place! Rather than recomputing paths after temporary topology changes, we argue for a separation of timescale between *offline* computation of multiple diverse paths and *online* spreading of load over these paths. We believe decoupling failure recovery from path computation leads to networks that are inherently more efficient, more scalable, and easier to manage.

Categories and Subject Descriptors

C.2.1 [**Computer-Communication Networks**]: Network Architecture and Design

General Terms

Algorithms, Design

Keywords

Internet architecture, routing, convergence, protocols

1 Introduction

The traditional task assigned to routing is very clear: routing calculates paths based on the current view of the network topology. When the network changes, whether due to a transient failure[1] or a permanent change in the topology, routing recomputes these paths. In fact, the reactive nature of routing—dynamically computing paths in response to failures to ensure connectivity—is typically seen as crucial to the Internet's resilience. We disagree.

In fact, there are several problems with using routing to respond to failure. First, computing a new set of paths with a distributed routing algorithm can be slow, leading to noticeable outages. Attempts to speed up convergence often risk scalability (by increasing the frequency of message exchanges) or involve *ad hoc* modifications (such as route-flap damping and tuning MRAI timers). Moreover, this recomputation can be for naught, since equipment failures and planned maintenance can be extremely transient—often completing before routing has reconverged—leading to *another* recomputation once the link is restored. In addition, route recomputation involves a complicated distributed algorithm that: is hard to understand and debug; is often the source of bugs, vulnerabilities, and misconfigurations; can lead to "update storms" if not properly tuned; and can only scalably compute a limited range of routing options (e.g., it is difficult to compute various sets of disjoint paths with a scalable distributed algorithm). Lastly, in some cases (such as spanning tree and BGP), the recomputation of routes doesn't just fix broken paths, it also disrupts working ones.

In this paper, we argue that greater path diversity can, and should, reduce our reliance on failure-driven (i.e., "real-time") recomputation of routes. We are motivated by two trends in network and system design that improve an application's ability to tolerate failures:

- Multipath routing lets edge nodes (whether end hosts or ASes) circumvent failures by explicitly directing traffic over a different path.
- Data centers often replicate at the application layer, allowing a load balancer to direct flows to another server after a (host or network) failure.

We believe these trends should change the end-point's relationship with the network from "I'll accept whatever best-effort service you give me" to "I'll take action to improve my service". In the early days of the Internet, the emphasis was on end-points passively "adapting" to the current level

[§]This article is an editorial note submitted to CCR. It has NOT been peer reviewed. Authors take full responsibility for this article's technical content. Comments can be posted through CCR Online.

[1]The term "failure" is overly narrow, since planned link maintenance is a common source of short-term service interruption, but for convenience we will use the term failure for all temporary outages.

of network service (through congestion control and adaptive coding rates); now the emphasis is moving towards a more active response. In this paper, we take this trend to its logical extreme by arguing that the *entire* burden of responding to failures should rest on the endpoints, and that routing should respond only to long-term changes in the underlying topology.

This idea is not new with us. It is implicit in the way MPLS uses precomputed backup paths, in the recent literature on multipath routing, in proposals to compute routes at separate routing-control servers, and in datacenter replication techniques. Our goal is to articulate the idea independent of any particular proposal, and generalize it to all network contexts (interdomain, intradomain, enterprise, datacenter). This paper, then, is a polemic, not a detailed technical presentation. However, given the centrality of routing in network architecture, and the extent to which dynamic route computation is part of networking lore, perhaps this "wordy rather than nerdy" paper has a useful role to play.

2 Jumping to (Logical) Conclusions

Separating failure recovery from path computation takes several recent research trends to their logical conclusion.

Difficulty of improving routing convergence. Slow convergence is a perennial problem with today's dynamic protocols for both interdomain [17] and intradomain [7] routing. Beyond imposing a heavy load on the routers, routing convergence is responsible for most serious performance disruptions for interactive applications [16]. Despite significant progress on reducing convergence time, current solutions—ranging from careful tuning of timers to propagating additional routing information—increase router overhead and protocol complexity. We view the difficulty of significantly reducing convergence time as a warning that we should *consider alternative approaches that avoid the convergence process entirely.*

MultiProtocol Label Switching (MPLS). MPLS enables the establishment of label-switched paths, using signaling protocols like RSVP and LDP. Once paths are established, routers simply forward packets based on a label. MPLS has fast-reroute mechanisms that compute and install backup paths that are used when a primary path (or an individual link on that path) fails. MPLS also enables splitting of traffic over multiple paths, and establishment of label-switched paths that span multiple ASes [5]. In fact, MPLS could be used as a building block for "offline multipath routing," by *establishing multiple primary label-switched paths in advance (based on the designed topology) and keeping these paths in the forwarding tables (even when failures occur).*

Separating routing from routers. Several recent proposals move route computation from the routers to separate in-network servers [3, 4, 10]. These architectures enable better routing decisions (as well as better access-control policies) by applying network-wide policies to a network-wide view. The routers simply forward packets and collect measurement data at the behest of the servers. The main practical challenge they face is maintaining an accurate, timely view of the network topology, to adapt quickly to changes. We argue that these proposals should be taken one step further by allowing these in-network servers to *compute routes in an offline fashion, based on the designed topology rather than the current up/down status of the links.*

Load balancing over multiple paths. Support for multipath routing, coupled with flexible load balancing by edge nodes, is a recurring theme in a wide variety of recent research. Intelligent route control allows stub ASes to split traffic over multiple upstream providers [1]. Deflecting packet through intermediate nodes—whether end hosts [2], routers [24], or ASes [23]—enables multipath routing on top of today's less flexible routing system. Multipath routing with flexible traffic splitting reduces traffic engineering from an NP-hard non-convex problem to a tractable convex one [11]. Multipath transport leads to more effective (and stable) dynamic load-balancing schemes [11, 13]; in fact, these papers implicitly assume the existence of multiple pre-pinned paths. We argue that multipath routing not only improves performance and reliability, but can *obviate the need to dynamically recompute routes in response to failures.*

Theoretical innovations. Numerous theoretical studies have shed light on how to route more effectively and efficiently. For example, work in *compact routing* and *game theory* give path assignments that minimize routing state, or maximize economic objectives. However, these schemes focus on *what* routing tables should contain rather than *how* this state should be computed and populated in a distributed fashion. In fact, the algorithms underlying these schemes are likely too computationally demanding to run in real time, when links and nodes fail. By decoupling route computation from failure recovery, we *enable the use of more sophisticated algorithms for computing routes.*

Difficulty of making routing protocols secure. Today's interdomain routing system is notoriously vulnerable to configuration mistakes, malicious attacks, and economically-driven manipulations. Moving to a secure version of BGP (e.g., S-BGP [14]) has proven difficult and expensive. Perhaps more disturbingly, even a ubiquitous deployment of S-BGP would remain vulnerable to strategic attacks, such as announcing one path while forwarding data traffic on another [8]. As an alternative to verifying BGP updates in the control plane, multipath routing allows each AS to select a working path based on end-to-end monitoring of integrity and path performance [21]. We take this argument one step further by *computing these multiple paths offline, while still relying on online monitoring to select a working path.*

The end-to-end argument. The end-to-end argument states (roughly) that, whenever possible, protocol operations should occur at the endpoints of the communication rather than within the network. While routing is often cited as an operation that must be done within the network, the

recent developments cited above suggest otherwise. Moreover, certain routing-related operations can be done only at end hosts, such as dealing with failures that networks cannot efficiently detect (e.g., application-specific performance issues, buggy or malicious routers). Today's single-path routing protocols leave endpoints vulnerable to such faults, as the network has no way of knowing it should switch to a different (working) route. Consistent with the end-to-end argument, we argue that *the data plane (rather than the control plane) is the right place to detect and react to reachability problems.*

3 Offline Routing, Online Load Balancing

In this section, we present our proposal for separating route computation (on the deployed topology) from failure handling (when link status changes). We next present some simple analysis to motivate our design choices, and then discuss several implementation approaches.

3.1 Computing Paths on Deployed Topology

Although our idea is conceptually simple, we describe it here in more detail to clarify our underlying assumptions and the mechanisms needed to make it a reality. To avoid confusion, we adopt the following terminology:

- *Topology* refers to the *deployed* links, whether physical links or virtual links representing tunnels or aggregated links.
- *Status* refers to the *current state* of the links, whether they are up or down.

Armed with this terminology, our proposal is simply to let routing compute paths based solely on the deployed topology. That is, routing should ignore all status changes and only recompute paths when the topology changes. These topology changes are rare (see below) and often known well in advance. Thus, the computation of routes can be a lengthy process, and can be done in a centralized fashion [3] or by third-parties if desired. This flexibility, in both the length and location of the computation, allows a wide variety of routes to be computed (such as k disjoint paths, or computing paths according to various criteria).

An endpoint A communicating with a remote endpoint B behind a failed link now has two options. First, if B is functionally replicated, then A can ignore network status changes and simply direct packets to substitute endpoints providing the same service. Modern service implementations typically replicate their internal state to recover from hardware failures, allowing the service to continue after an endpoint fails. Second, if B isn't replicated, or if maintaining connectivity to the same endpoint is preferable, A can simply direct traffic to one of the paths to B that is not affected by the failure.

3.2 Underlying Assumptions About Failures

Our approach relies on two underlying assumptions, which may not hold in all settings. First, we assume topology changes are far less frequent than status changes. If this were not the case, then our approach would not be significantly decreasing route recomputation, and our point would be moot. However, this assumption is supported by a study [19],[2] where failures with time-to-repair longer than 24 hours were 3.7% of all failures. Thus, only one in 25 routing events reflects a long-term topological change, and therefore our approach would decrease the frequency of route computation by more than an order of magnitude.

Second, we assume the number of alternatives offered (either endpoint replication or multiple paths to the same endpoint) is enough to mask link failures. When we say "mask failures" we mean only to the extent that routing reconvergence could alleviate the link failures. For instance, multipath can only mask failures if the topology remains connected; but that restriction applies to traditional route recomputation as well. Consider a simple Poisson model where paths fail at rate f and links are repaired at rate r. In this case, the probability that all k alternative path options are unavailable is $(1+\frac{r}{f})^{-k}$. If the route recomputation takes time τ, then the average downtime of the route reconvergence is $f\tau$. Thus, in this trivial model the comparison of the downtimes of these two approaches reduces to a comparison of the quantities $(1+\frac{r}{f})^{-k}$ and $f\tau$. As long as the repair rate is an order of magnitude greater than the failure rate, we think it is easier to achieve high reliability by increasing k (which is just adding another endpoint replica or exposing an additional path) or increasing r (which involves improving network management) than decreasing τ, which involves tuning a complicated distributed route selection process so that it converges faster while still remaining scalable and correct. Note that improving the failure rate doesn't favor route recomputation, since decreases in f favor our approach over reconvergence as long as $k > 1$.

We next use simulation to evaluate performance under the particular failure modes and topologies that are commonly found in networks. Figure 3.2 shows the degree to which network outages are masked, in terms of the number of flows affected by one or more failure events. Here, we assume a failure model based on that of a large tier-1 ISP network, where links fail with mean time between failures of two days, mean time to repair of one minute [19]. We then compare a simple single-path routing scheme (upon failure, routing discovers a new working path after a convergence delay) with a multipath scheme (the endpoint utilizes a fixed set of k disjoint-as-possible available paths) and measure the fraction of time the path is unavailable. In the network we studied, the route recomputation averages 1.2 seconds, which is less

[2]This data is from 2002, but unfortunately we couldn't find more recent data. However, we do not expect that the rate of topology changes has increased over time.

than the average failure duration. Nonetheless, even for a relatively low value of k ($k = 2$ or 3), the multipath approach outperforms the single-path routing approach. This is because the likelihood of multiple paths failing simultaneously is far less than the probability of undergoing a convergence event. While these numbers indicate that our approach works well, it is important to note that there are environments where our assumptions do not hold. For example, wireless systems may have high failure rates and unpredictable outage durations. However, for many of the "standard" networking environments (such as interdomain, ISP, enterprise, and datacenter networking) these assumptions do hold.

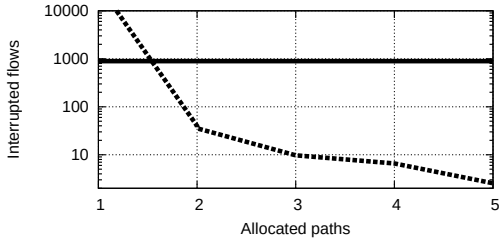

Figure 1: Comparison of single-path dynamic routing (solid line) and multipath static routing (dashed line).

3.3 Implementation and Deployment Scenarios

Conventional IP routing architectures have two components: a *control plane* which is responsible for disseminating routes and selecting which routes to propagate further, and a *data plane* responsible for high-speed packet forwarding. Our approach continues to rely on conventional mechanisms to perform packet forwarding, but decouples the route selection process from routers. In order to make this design practical, we require endpoints to implement several new (yet tenable) mechanisms, which we discuss next.

Application-level failure recovery. It has traditionally been the network's responsibility to ensure a single end-to-end path remained available for the duration of the communication session. Removing dynamic route recomputation from the network and placing the responsibility of route selection on the endpoints has implications for applications.

Many datacenter deployments already include service interruption in their end-to-end failure model, handling it through application-level load balancing. Large webserver deployments, for example, must be able to withstand server failure on the back-end (which we suspect happens more often then network link failures). Network failures can be handled with the same recovery characteristics. In both cases, ongoing sessions may be interrupted, but subsequent connections will be sent to reachable servers via the load balancer.

However, this approach assumes the application is tolerant of service disruption during network failure, which may not always be the case. For example, in the absence of a user hitting refresh, today's HTTP implementations typically do not issue a new request due to a dropped TCP connection since that would be necessary only to recover from server crashes. With our proposal, they should, as TCP cannot recover from endpoint changes. While this may seem like introducing a major application requirement, we note increasingly popular but inherently intermittent wireless network connectivity has already caused many applications to become more robust against transient network outages.

Endpoint-based failure detection. We have been implicitly assuming that endpoints decide which of their available paths to use to forward data (or which remote replica to contact). To do this, endpoints independently monitor the availability and quality of paths, and use this to inform their choice. While path monitoring can be tricky (due to, for instance, asymmetric routes), we nonetheless feel that current failure detection protocols such as Bidirectional Forwarding Detection are a promising starting point. In addition, path-quality monitoring techniques can be made robust in the presence of adversaries that selectively add, drop, or modify packet [9], in contrast to the simple (and easily-gamed) failure-detection mechanisms used in most routing protocols. End-to-end techniques, particularly by end hosts, can also monitor *integrity*, such as authenticating the destination by using IPsec or SSL [21]—something routers in the middle of the Internet are ill-equipped to do.

Instead of dedicating resources to the selection of a path (and detecting a failed path) endpoints can implement an alternative, more elegant strategy as suggested in [22]: once they use multiple paths *simultaneously* they can remain ignorant about path failures and yet have optimal performance and reaction time to failures. While we don't explore this option in detail here, we note multipath extensions to existing transport protocols are already on their way (for details see, *e.g.*, [6]).

Fixing failures. Obviously, one needs a mechanism to fix failures in the infrastructure, and to do so on a reasonable time scale since this determines the probability of experiencing more than k simultaneously failed paths (or replicas). The balance between the urgency of infrastructure repairs and the number of alternative paths exposed to endpoints involves site-specific factors, and service providers will likely adopt different tradeoffs in this regard.

Optimizations. The high-level design discussed above is a pure embodiment of our argument in its independence from the network-based failure recovery. However, even though endpoints shouldn't assume that the network will recover from link failures, the network can still improve the quality of service it provides for the endpoints by several means. As discussed above, depending on their customers, it may be in their interests to do so.

We don't prevent local link availability improvements, as long as they scale, and don't interfere with operating paths. For instance, one can still use MPLS fast failover mechanisms to improve availability by masking network element and link failures. In a similar manner, physical links can be aggregated

together into a single virtual link to improve availability and performance.

We also don't prevent network-assisted improvements to the endpoints' failure reaction time. If most endpoints don't use multiple paths simultaneously, but instead prefer timely recovery from path failures by selecting an alternative path, service providers may *explicitly* inform endpoints about link failures to help guide them in their choice of alternate paths. However, it is nontrivial to design an efficient and secure failure dissemination protocol, which is all the more reason to promote the simultaneous use of multiple paths.

4 Implications

Removing routing recomputation from the network separates routing into two simple steps that have few restrictions and interdependencies in their implementation: topology discovery and path computation. Each can be done in a variety of ways and, because they are not done in real time, do not pose scalability problems. In the following we briefly discuss some obvious examples of how to take advantage of this freedom from implementation and scaling constraints.

Improved scalability of routing. Scalability of Internet routing is largely limited by the overhead of routing state exchanges (*churn*). The continuing growth of the Internet, coupled with the need for multi-homing and fine-grained traffic-engineering have rapidly increased routing table size, which in turn has increased churn. Our proposal, by making route recomputation far less frequent, greatly reduces churn. As an example of the resulting possibilities, significantly reduced churn would make the research community's various "flat" and name-based routing proposals slightly less unrealistic. These proposals call for even greater routing table size requirements than we have today (see, *e.g.*, [15]). With enormous routing tables, churn-induced router CPU load becomes a limiting factor, unless the rate of route updates can be significantly curtailed, as in our proposal.

Unification of mechanism. Failure detection and recovery is already an integral part of many systems, such as datacenters supporting online services. The addition of dynamic route recomputation to these environments introduces redundant functionality with unnecessary complexity. In degenerate cases, it is possible for multiple dynamically adapting systems (such as latency aware load-balancing and route selection) to create feedback loops with continuous instability.

Novel path computation procedures. Since we no longer require that these steps be performed quickly after a failure, endpoints (or the routing service they use) may use more advanced algorithms to perform these steps. For example, offloading route computation to servers (*e.g.*, Ethane, RCP, RAS [3, 4, 18]) requires network state to be quickly propagated to the server for processing, increasing load and sensitivity to delay. Our approach allows this to be done without concern for timeliness of interaction between network elements and the server. Also, in BGP, routing updates propagate in a single direction (from destinations back towards sources). Computing routes on longer timescales may enable use of protocols that require multiple rounds of message exchanges amongst participants. For example, adjacent ISPs may negotiate to determine routes that achieve more mutually beneficial economic goals. Finally, recent research indicates that performance and flexibility of route selection may be significantly improved with some measure of endpoint participation. By allowing endpoints to participate in route selection, application-specific performance requirements can be better accommodated and network-wide performance can be improved through better congestion control and traffic engineering [11, 13].

New remedies against failures. While our approach makes the network less responsible for failures, it simplifies reaction to correlated failures. Correlated failures occur when, for example, multiple IP adjacencies traverse the same optical fiber. Such events can do extreme damage to the network, unless great care has been taken to ensure these failures do not take out a set of replicated links. These "shared risks" are typically not visible to layer-three routers, making it difficult (if not impossible) to rely on dynamic routing protocols to selects paths based on this information. In addition, computing multiple paths that do not share correlation is computationally expensive. Removing the need to perform this computation quickly allows us to use these computationally intense algorithms to select routes less prone to failure. The fact that our approach does not need to conflate failure diagnosis (detection) and failure recovery anymore (as often happens now) could have cost-reducing implications for network management. Once failure detection and recovery are decoupled (in routing), failure diagnosis will be simpler.

5 Discussion

One can read this paper in the spirit of "clean-slate" design, and ignore all issues of deployment. This would be an understandable escape, because retooling the entire Internet to remove dynamic routing protocols represents a massive undertaking. However, deployment of our design may be accelerated by several factors. First, our design is agnostic to any particular underlying protocol, and hence may be naturally implemented atop a wide variety of existing protocols and platforms. Systems and protocols to provision multiple paths across networks have been explored in previous works (for a small sampling, see [20, 23, 24]), and these techniques may be directly adapted to our design. Second, our design is enabled by the increased use of load-balancing devices and application-layer replication techniques. Finally, and most importantly, deployment of multipath may happen naturally – as larger portions of end hosts become capable of selecting amongst multiple paths, routing protocols like

BGP may become slowly less and less reactive to transient failures (due to ISPs' preferences to reduce churn).

We advocate replacing dynamic computation of routes with two mechanisms, multipath routing and application-layer replication. Keeping in mind that good network designs place minimal constraints on the algorithms that implement them and the applications that operate atop them, we observe that these mechanisms can be used and implemented in a variety of ways. Multiple paths may be used to avoid malicious intermediaries, to select routes according to application-specific goals, to improve throughput, or may be used as backups to improve resilience. End hosts may directly specify which paths to use, they may be decided by network policy, or some combination of the two. In addition, topology or path information can be disseminated in several ways, ranging from periodically updating network maps to routing path requests to a separate network service responsible for knowing the network topology.

While our proposal refutes longstanding networking practice, we are not saying that the past 35 years (or more) of dynamic route computation has been a mistake. When networks were in their infancy, the focus wasn't on high availability but on extreme recoverability; short outages were fine, but the ability to recover from serious and widespread failures was paramount. For this goal, dynamic computation of routes is necessary, since replication can't deal with severe failure scenarios. Now that networks have become finely-tuned and carefully managed components of our critical infrastructure, the priorities have changed. High availability is now the paramount goal, and this cannot be achieved through dynamic computation of routes. Distributed routing algorithms can't respond fast enough, nor provide flexible enough selection of paths, to preserve service quality. Instead, the infrastructure should provide replicated resources, something that network and systems infrastructures are well suited for. Applications and endpoints should then leverage this replication to ensure their own high availability. This separation of concerns between the infrastructure and endpoints is a familiar theme in distributed systems, and (similar to [12]) our proposal is merely a restatement of these longstanding insights.

We are by no means eliminating route computation. We still need it to deal with the (hopefully) rare case of widespread failures which fundamentally change the network topology. Thus, route computation remains an important component of our routing infrastructure. Our proposal does nothing more than limit route computation to the task for which it is best suited.

6 References

[1] A. Akella, J. Pang, B. Maggs, S. Seshan, and A. Shaikh. A comparison of overlay routing and multihoming route control. In *Proc. SIGCOMM*, 2004.

[2] D. Andersen, H. Balakrishnan, F. Kaashoek, and R. Morris. Resilient overlay networks. In *Proc. SOSP*, 2001.

[3] M. Caesar, D. Caldwell, N. Feamster, J. Rexford, A. Shaikh, and K. van der Merwe. Design and implementation of a routing control platform. In *Proc. NSDI*, 2005.

[4] M. Casado, M. J. Freedman, J. Pettit, J. Luo, N. McKeown, and S. Shenker. Ethane: Taking control of the enterprise. In *Proc. SIGCOMM*, 2007.

[5] A. Farrel, J.-P. Vasseur, and J. Ash. A Path Computation Element (PCE)-Based Architecture, August 2006. RFC 4655.

[6] A. Ford, C. Raiciu, and M. Handley. TCP Extensions for Multipath Operation with Multiple Addresses. Internet Draft. draft-ford-mptcp-multiaddressed-02.txt, Oct 2009.

[7] P. Francois, C. Filsfils, J. Evans, and O. Bonaventure. Achieving sub-second IGP convergence in large IP networks. *ACM SIGCOMM CCR*, July 2005.

[8] S. Goldberg, S. Halevi, A. Jaggard, V. Ramachandran, and R. Wright. Rationality and traffic attraction: Incentives for honestly announcing paths in BGP. In *Proc. SIGCOMM*, 2008.

[9] S. Goldberg, D. Xiao, E. Tromer, B. Barak, and J. Rexford. Path-quality monitoring in the presence of adversaries. In *Proc. ACM SIGMETRICS*, 2008.

[10] N. Gude, T. Koponen, J. Pettit, B. Pfaff, M. Casado, N. McKeown, and S. Shenker. NOX: Towards an Operating System for Networks. In *ACM SIGCOMM CCR*, July 2008.

[11] J. He, M. Suchara, M. Bresler, J. Rexford, and M. Chiang. Rethinking Internet traffic management: From multiple decompositions to a practical protocol. In *Proc. CoNEXT*, 2007.

[12] J. P. John, E. Katz-Bassett, A. Krishnamurthy, T. Anderson, and A. Venkataramani. Consensus Routing: The Internet as a Distributed System. In *Proc. NSDI*, 2008.

[13] S. Kandula, D. Katabi, B. Davie, and A. Charny. Walking the tightrope: Responsive yet stable traffic engineering. In *Proc. SIGCOMM*, 2005.

[14] S. Kent, C. Lynn, and K. Seo. Secure Border Gateway Protocol (S-BGP). *IEEE J. Selected Areas in Communications*, 2000.

[15] T. Koponen, M. Chawla, B.-G. Chun, A. Ermolinskiy, K. H. Kim, S. Shenker, and I. Stoica. A Data-Oriented (and Beyond) Network Architecture. In *Proc. SIGCOMM*, 2007.

[16] N. Kushman, S. Kandula, and D. Katabi. Can you hear me now?! it must be BGP. *ACM SIGCOMM CCR*, April 2007.

[17] C. Labovitz, A. Ahuja, A. Bose, and F. Jahanian. Delayed Internet routing convergence. *IEEE/ACM Trans. Networking*, June 2001.

[18] K. Lakshminarayanan, I. Stoica, and S. Shenker. Routing as a service. Technical Report CSD-04-1327, UC Berkeley, 2004.

[19] A. Markopoulou, G. Iannaconne, S. Bhattacharrya, C.-N. Chuah, and C. Diot. Characterization of failures in an IP backbone. In *IEEE/ACM Trans. Networking*, October 2008.

[20] M. Motiwala, M. Elmore, N. Feamster, and S. Vempala. Path splicing. In *Proc. SIGCOMM*, 2008.

[21] D. Wendlandt, I. Avramopoulos, D. Andersen, and J. Rexford. Don't secure routing protocols, secure data delivery. In *Proc. HotNets*, 2006.

[22] D. Wischik, M. Handley, and M. B. Braun. The Resource Pooling Principle. *ACM SIGCOMM CCR*, October 2008.

[23] W. Xu and J. Rexford. MIRO: Multi-path Interdomain ROuting. In *Proc. SIGCOMM*, 2006.

[24] X. Yang and D. Wetherall. Source selectable path diversity via routing deflections. In *Proc. SIGCOMM*, 2006.