Decoupling Connectivity from Routing

Karthik Lakshminarayanan Thomas Anderson Scott Shenker Ion Stoica David Wetherall
University of California, Berkeley and University of Washington

Abstract

To provide routing flexibility, that is, to accommodate various performance and policy goals, routing protocols (such as OSPF and EIGRP) include many complex knobs. Owing to this complexity, protocols today do not adequately satisfy their main goal—to provide connectivity between nodes in the face of failures and misconfigured nodes. In this paper, we ask the question of how one can design routing protocols that are flexible, yet provide connectivity in the face of failures and misconfigurations. To this end, we propose a different routing paradigm that decouples the task of providing basic connectivity from sophisticated routing operations. We propose an underlying Basic Connectivity Routing Protocol (BCRP) that is robust to link failures and prevents misconfigured nodes from arbitrarily subverting traffic. Routing can then be made flexible by layering sophisticated route selection on top of BCRP; these protocols fall back to BCRP when failures are encountered.

1 Introduction

Arguably, the most important goal of routing protocols is to provide *connectivity*, that is, two nodes should be able to communicate as long as there is a path in the underlying network. To ensure connectivity, routing protocols need to rapidly detect and recover from failures. Fast failure recovery is only one aspect in ensuring connectivity; another important aspect is robustness to malicious or misconfigured nodes. Even a single misconfigured node can disrupt intra-domain connectivity to a large extent. Indeed, there are documented cases of a single misconfigured node causing disruption within an enterprise network [10].

In addition to connectivity, another property that both network operators and users require of routing protocols is *flexibility*, that is, the ability to accommodate a wide range of policy and performance goals, and allow these goals to seamlessly change over time. To allow flexibility, router vendors provide tens of complex knobs to configure routing protocols, such as OSPF and Cisco's EIGRP [1]. In fact, for OSPF, quite a few knobs need to be manipulated for just getting the basic protocol to work. This complexity not only makes these protocols prone to misconfigurations, but also complicates the failure recovery mechanisms. Since failure recovery is tied to the routing protocols, the recovery mechanisms are *constrained* by various policies and performance goals—for example, to prevent oscillations, link-state protocols are conservative in detecting link and node failures, thus increasing protocol convergence time after recovery. The problem is further compounded by the fact that today's routing protocols pay little attention to dealing with malicious and misconfigured routers.

In this paper, we ask the question of how one can design routing protocols that are flexible, yet robust in the presence of failures, misconfigurations, and malicious routers. To this end, we propose a different approach of designing routing protocols: instead of designing a single protocol that provides both flexibility and connectivity, we *separate* these two goals. We propose an underlying Basic Connectivity Routing Protocol (BCRP) that provides connectivity in the face of link failures, and prevents malicious (or misconfigured) nodes from arbitrarily subverting network traffic. Sophisticated routing protocols, tuned to various specific needs, can be layered on top of BCRP; these protocols fall back to BCRP when failures are encountered. In this way, designers of routing protocols can focus on the more complex demands while being assured that basic connectivity is still being provided by BCRP. In this paper, we restrict our exposition of BCRP to intra-domain routing. Extending BCRP to inter-domain routing is work-in-progress, but it appears that a similar approach can be used there too.

BCRP is based on the assumption, which we believe holds quite widely, that the installed set of links in a managed infrastructure (such as an ISP) is relatively stable. This information can be authenticated and reliably transmitted by a central administrative node to all the nodes periodically. Hence, BCRP can rely on each node having a recent snapshot of possibly available network links, which we term the *consistent map*. To communicate failures in BCRP, we don't rely on inter-router messages but instead carry the encountered link failures in the packet header. The route the packet takes is computed using the consistent map minus the failed links contained in the packet; hence, all nodes along the path of a packet compute routes based on consistent information. Since the list of failed links is monotonically non-decreasing, the packet is guaranteed to reach the destination as long as a path exists (modulo factors such as failure detection time).

In the next section, we present the BCRP algorithm and illustrate its benefits. In Section 3, we present several techniques that address efficiency and security issues in BCRP. In Section 4, we present simple experiments using Rocketfuel ISP topologies that suggest that BCRP is feasible. We conclude after listing some future directions in Section 6.

2 BCRP Algorithm

We present how BCRP works by elaborating the main design ideas behind the protocol.

Decouple failure recovery from protocol convergence: The main reason behind slow failure recovery in traditional routing protocols is their monolithic design—a single protocol is responsible for ensuring protocol convergence when link costs change as well as recovering from link failures. By decoupling action on failure from normal routing behavior, we can ensure that failure recovery is faster.

Such a decoupling is particularly beneficial because typically hints about link/node failure are known long before failure actually is confirmed. For example, when higher-level HELLO messages are used, a lost or a delayed HELLO message could indicate failure, whereas a failure actually is confirmed only after multiple messages are lost (which constitutes a timeout). Routing protocols send updates about failure only after confirmation of failure to prevent oscillations. However, conservative action can be taken right after noticing the first hint of a failure, thus reducing the vulnerability window. Being conservative does not hurt us much since we do not inject this information into the routing protocol, but use it locally to avoid the link that is potentially failed.

Centralized dissemination of link costs: The set of installed links in a managed infrastructure (such as ISP or enterprise network) are typically stable over time periods of days. These links may go up and down from time to time, but the "map" that describes a superset of the links being used changes very slowly—only when physical links are added or deleted. Moreover, this map is known to some central administrative node, which can authenticate and reliably disseminate the network map, which we call the *consistent map*, once every time T (perhaps once an hour). Now, all nodes are guaranteed to have the same consistent map, a superset of the network graph. In practice, the central node might remove certain physical links when constructing the consistent map to account for links that might have long down-times (such as during periodic maintenance).

Failure information contained in packets: Packets contain information about the *failed links* that are encountered, *i.e.,* when a packet routed based on the consistent map encounters a link that is failed (or potentially failed, based on some detection algorithm), the link is added to the list of failed links contained in the packet. The route is computed based on the consistent map minus the failed links.

From these observations, it follows that all nodes make per-packet routing decisions consistently, since everyone has the same consistent map, and the remaining relevant information (the list of failed links) is contained in the packet. Since each packet is treated separately, the information contained in one packet does not affect the forwarding of other packets. The pseudocode for vanilla BCRP is presented in Figure 1.

```
Initialization: pkt.failed_links = NULL
Packet Forwarding:
  do
    path = ComputePath(M − pkt.failed_links)
    if (path == NULL)
      abort("No path to destination")
    else if (path.next_hop == FAILED)
      pkt.failed_links ∪= path.next_hop
    else
      Forward(pkt, path.next_hop)
  while (! PacketForwarded)
```

Figure 1: Vanilla basic connectivity routing protocol.

2.1 Properties of BCRP

BCRP has two main properties. Since the properties are intuitive, we only present informal arguments why they hold.

Property 1. Guaranteed reachability*: If link failures are detected instantaneously, and there is a spanning subgraph of the network graph that is connected for the entire time a packet is in transit, then BCRP will successfully route the packet.*

Consider a packet destined for node D. Let's say that BCRP takes the path through nodes N_1, N_2, N_3, hits a failed link f_α at N_3, then gets routed through N_4, N_5, N_6, hits a failed link f_β and so on. All nodes between each consecutive pair of failed links compute routes based on the same state. For example, nodes N_1, N_2 and N_3 would all route on the complete consistent map, whereas nodes N_4, N_5 and N_6 would route on the consistent map minus f_α and so on.

When the node where a failed link is encountered (such as N_3 and N_6) computes a route, there are two possibilities: either there is no path to the destination in which case the packet is dropped, or there is some path to the destination in which case the new graph on which nodes compute the path becomes smaller (*i.e.,* it does not include the failed edge).[1]

With every new failed link encountered, the graph over which the packet is routed monotonically becomes smaller. Hence, if the underlying graph has a spanning connected subgraph for the entire time the packet is in transit, the packet will reach the destination. In practice, packets would have a TTL (Time-to-Live) associated with them to prevent very long paths.

Property 2. Security property*: As long as the "correct" path of a packet does not traverse any malicious node, BCRP will successfully route the packet. By correct path, we refer to the path that would be taken if no node is malicious.*

[1] If routers cannot hold on to the packet due to resource limitations, packets might be dropped during the recomputation period.

In today's protocols, malicious routers subvert a network to cause more routes to flow through them. In BCRP, no link information is exchanged between the nodes. Furthermore, each packet is treated independently of other packets—only failures that the packet encounters are taken into account for computing the paths. Hence, a node not in the path that the packet takes using BCRP cannot affect the fate of the packet. While BCRP does not provide security guarantees that are as strict as some earlier theoretical works (such as [3, 9, 13]), it provides isolation: it restricts the damage that a malicious node can cause to only the traffic it forwards. We contrast BCRP with earlier work in more detail in Section 5.

2.2 Usage of BCRP

We briefly discuss how BCRP interacts with higher level protocols. All packets contain a 1-bit flag that indicates whether the packet should be routed with BCRP. The protocols layered on top of BCRP explicitly invoke BCRP by setting the flag. For simplicity, we assume that once the BCRP flag is set, the packet will be routed using BCRP till the destination. Each packet routed using BCRP carries BCRP-specific information (such as the list of failed links) in a special header called the BCRP header. The BCRP header can be implemented as a layer-2.5 header, similar to MPLS.

BCRP can be invoked in many scenarios. When a source wants the security property, it sets the flag on every packet since the security property applies only to packets routed with BCRP. For handling failures, if the higher layer protocol does not have a route to the destination, it can hand over the packet to BCRP. Alternatively, it can send a packet to BCRP and request BCRP routing only if the outbound link is detected as "possibly failed" (as we noted earlier, BCRP can be less conservative in detecting failures). In addition, the BCRP header provides useful diagnostic information to the network operators when failures occur.

3 Detailed Design of BCRP

We present techniques to reduce the overhead of the vanilla BCRP algorithm. We also address two resource exhaustion attacks on BCRP. We note that the techniques presented in this section do not affect the two properties of BCRP. Due to lack of space, we don't present a detailed reasoning.

3.1 Reducing computational overhead

Vanilla BCRP requires path computation for every packet that encounters a failure at every node that the packet traverses. Supporting high link speeds with per-packet, per-node recomputation is infeasible.

3.1.1 Eliminating per-node route computation

To avoid computation at *every* node in the packet's path, the nodes where failures are encountered insert a source route to the destination. Source routing between failures implies that only nodes where the packets encounter failures need to perform any computation; all other nodes merely forward packets based on the source route. (Note that this mechanism does not sacrifice the security property.)

Since source routes can potentially be long, we discuss possible ways of reducing the overhead. Intuitively, after routing a packet a few hops away from the failed link (based on consistent map minus the failed links), default routing based on the consistent map *alone* would likely avoid the failed link. Hence, nodes can possibly add the source route only till the point from where the default route will take the packet to the destination. Since the failed link is still added to the packet, persistent loops don't occur, and we don't sacrifice the guaranteed reachability property.

3.1.2 Eliminating per-packet route computation

To eliminate per-packet recomputation at nodes where failures are encountered, nodes perform some simple precomputation. Each node, in addition to maintaining the default forwarding table (based on the consistent map), precomputes a table for each of its outbound links that should be used when that outbound link is failed. In other words, for every outbound link l, the node computes the forwarding table using the consistent map minus the link l. However, in terms of the actual forwarding state, such a precomputation only doubles the memory requirement: for each destination, in addition to the default path P computed using the consistent map, we need to store the precomputed path computed using consistent map minus l_P, where l_P is first hop in P.

When a packet encounters a failure at link l at this node, the precomputed path P_l to the destination (using the consistent map minus the link l) would be the desired path as long as P_l does not contain a link l' that belongs to the set of failed links that the packet carries. If this constraint is not satisfied, recomputation using the consistent map minus the failed links (including l) is needed. When the fraction of failed links is small, intuitively, the chance that a recomputation is triggered is small. We later present some simulations using the Rocketfuel ISP topologies that support this intuition.

3.1.3 Reducing computation time

For performing recomputation, we borrow from the literature on incremental recomputation [4,7]. Prior research has shown that incremental recomputation can be performed within the order of few milliseconds even for graphs with a thousand nodes [2]. Performing recomputation within a few milliseconds is very reasonable since failure detection itself could take that much time, and hence recomputation does not substantially worsen the vulnerability period.

Furthermore, since many of the incremental algorithms construct shortest-path trees, the recomputation step yields paths to *all* destinations. Hence, by saving this information, the node can avoid recomputation for all future packets with the same set of failed links irrespective of the destination.

3.2 Addressing security issues

Since BCRP introduces additional overhead on the routers (such as route computation), we now discuss router resource exhaustion attacks on BCRP. First of all, we note that a network can prevent hosts outside the network from abusing BCRP by having the edge (ingress) routers strip off any BCRP headers when the packet enters the network.

Within a single network, in BCRP, two nodes do not exchange any protocol messages. The only information that is exchanged between nodes is what is contained in the packets: a source route, and a set of failed links. The only operations that BCRP nodes perform are forwarding packets, and recomputing routes. By manipulating both the pieces of information contained in the packet, a malicious node can make the benign nodes waste its resources, in terms of forwarding operations as well as recomputations.

3.2.1 Protection from source route manipulation

We consider attacks that a malicious router can perform by manipulating source routes only. A node can inject packets with long source routes, thus wasting network resources. Even with a TTL restriction (of, say, n), a router can make each packet it generates consume n units of network resources. A simple solution to detect this problem is statistical checking of source routes in packets. Each node can check, with a probability p, if the source route in the packet indeed corresponds to what one would obtain using the consistent map and the failure information in the packet.

Assume that a malicious source route is L hops long. The probability that *some* node checks the route is $(1-(1-p)^L)$. Since checking cannot be done at line speeds, $p \ll 1$, and hence the probability $\approx (1-(1-Lp)) = Lp$. Hence, the expected number of malicious packets that would go through before detection is $1/Lp$, and the expected damage is $L \times 1/Lp = 1/p$. With $L = 20$ and $p = 0.001$, an attack would be detected within 50 packets and would cause 1000 unnecessary forwarding operations in expectation.

When a node has detected that *some* node in the path is malicious, there are many ways of reacting. In addition to dropping such malicious packets, the node can simply raise an alarm so that the network operator can debug the problem further. Alternatively, the node can possibly send a message to the node that forwarded the malicious packet requesting it to check its packets more frequently.

3.2.2 Protection from failure header manipulation

By manipulating failure header, a malicious node can not only construct long source routes, but also make a benign router perform unnecessary recomputation. For example, in Figure 2, node N1 forwards a packet destined to D to node N2 with failure list containing the edge N4−D. Since the default next hop N2−N3 is failed and the precomputed path N2−N4−D has a failed link N4−D, N2 needs to perform a recomputation. Recomputation is triggered at a node N only

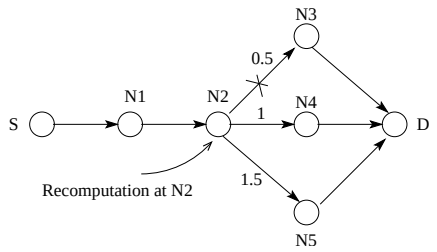

Figure 2: Node N2 gets packet from N1 with a failure list containing the edge (N4,D). All unlabeled edges have unit weight.

if (a) the next-hop at N based on the packet's source route is failed, *and* (b) the precomputed path to destination that does not use the failed link contains a link that belongs to the set of failed links carried in the packet. Due to these constraints the recomputation attack is not easy to launch.

The node N2 can now verify that it is possible to reach N2 from the source of the packet when the failures that the packet enumerates are encountered. In the example, by emulating BCRP, N2 can detect that the packet could not have encountered the link N4−D. Once a malicious attempt is detected, reaction can be similar to that described above. Since this detection mechanism could involve a few recomputations, the check can be done only probabilistically to share the recomputation resources with the route recomputation mechanisms involved in packet forwarding.

However, one can construct graphs in which malicious recomputation attempts are legitimately possible too—the probabilistic checking will not detect these. In such cases, if the node does not have recomputation resources, it would just send on the precomputed path (using M − Failed_Link_At_N) and add Failed_Link_At_N to the packet header. If this packet reaches the attacker again before hitting the link that is faked as failed, then it is equivalent to the attacker generating a fresh packet. If not, the node whose link was added by attacker will detect the attack and would raise an alarm.

4 Feasibility of BCRP

In this section, using ISP topologies from the Rocketfuel study [11], we present preliminary results that suggest that BCRP is feasible for intra-domain topologies.[2] Experiments were performed with five ISP backbone topologies; however, we report only a representative sample (we include the AS numbers from the Rocketfuel data). For the link weights, we used the inferred link weights from the Rocketfuel study [6]. The topologies had a few hundred routers, with a network diameter in the range $9-16$, and average path length in the range $4-6$. In all the experiments, we failed between 1% and 10% of the links uniformly at random, and studied the different metrics of interest. Since we didn't have access to the

[2]Since we do not (yet) have detailed ISP failure data, we couldn't perform a more detailed evaluation.

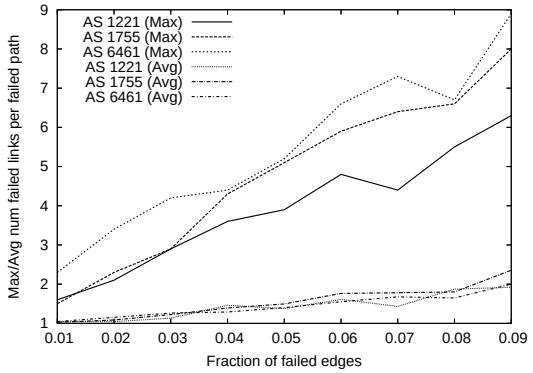

Figure 3: Average/Maximum number of failed links encountered by a packet, restricted to packets that encounter at least one failure, as a fraction of failed links.

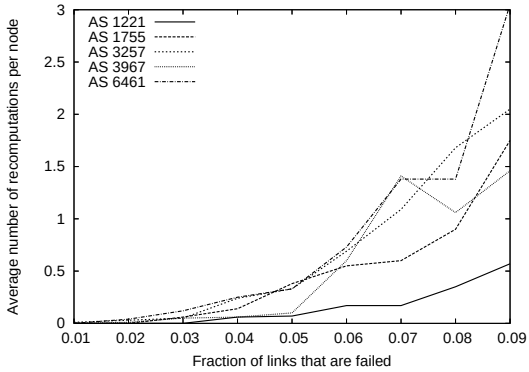

Figure 4: Mean number of recomputations needed per node (for all possible paths in the graph) as a function of fraction of failed links.

actual traffic matrices, all experiments were performed for *all pairs communication*, and results averaged over 20 runs.

BCRP introduces extra overhead only for packets that encounter a failed link. The overhead can be classified into: (a) packet header overhead, since failed links as well as source routes are carried, (b) recomputation overhead, and (c) network overhead, *i.e.,* extra cost of routing the packet.

Packet overhead. In BCRP, the failed links that the packet encounters are carried in the packet header. To evaluate the extra space needed, we plotted the average and maximum number of failed links (for packets that encounter at least one failure) as a function of the fraction of failed edges (see Figure 3). The average number of failed links per path is only around two even when 10% of the links are failed. The maximum number of failed links is understandably higher, but less than 10 for the same failure rate. Assuming that two bytes are needed to represent a node ID (this would allow a network of 65536 nodes), the average and maximum per-packet overhead are 16 and 40 bytes, respectively, even for 10% link failures. Note that 10% link failures is an extremely high failure rate, and we expect failures in managed ISP networks to be much smaller in practice.

In addition to failed links, packets also carry source routes to reduce computation overhead. Since the average path length in the topologies we used is about 6, and the diameter is about 15, the average and maximum overhead for packets that encounter failures are 12 and 30 bytes, respectively.

Recomputation overhead. Figure 4 plots the maximum number of recomputations that are needed averaged over all the nodes in the network. (Note that some nodes perform more recomputations than others.) Since all possible paths are considered, this number presents an upper-bound on the actual number of recomputations that would be needed given a certain traffic demand. The results show that for up to 5% failure rate, recomputations are almost never needed (the average number of recomputations per node is under 0.5 over all pairs communication). Recomputations are needed only rarely because: (a) precomputed paths provide the route to the destination in most cases, and (b) precomputed state is saved for future packets. The number increases super-linearly with the failure rate since the different possible combinations of failed links grows combinatorially.

Routing overhead. Since BCRP learns link failures as the packet is routed, the routes that packets routed with BCRP take would be sub-optimal compared to those computed by an oracle that knows all failure information. Figure 5 (top) plots the CDF of the stretch that BCRP introduces relative to this oracle. We only consider the paths in which failure was encountered; in other cases, the stretch is trivially 1. For about 90% of the BCRP routed paths, the stretch is lower than 1.5, and the maximum value is under 4. The scatterplot of the stretch plotted against the oracle-computed distance in Figure 5(bottom) shows that most of the large values of stretch are caused only by short, low-cost paths. This result is promising since the actual additional cost imposed by such short paths is anyway low.

5 Related Work

We separate related work into two parts: those that address failure recovery in routing protocols, and those that provide guarantees on secure routing.

There have been several efforts to provide fast fail-over. In fact, the idea of decoupling failure recovery from protocol convergence is not new. There are several works—such as IP restoration [5], MPLS Fast-Reroute [8], etc.—that propose having precomputed backup routes which can be used when primary paths in the network fail. However, backup routes are practical only for single failure scenarios; to achieve the guaranteed reachability property of BCRP with multiple failures, many backup paths would be needed. In contrast, BCRP provides a graceful way of handling multiple link failures; indeed from our experiments, we see that even with a failure rate of 1%, multiple failures can occur in real networks.

Some efforts have addressed failure recovery directly at the level of routing protocol. For instance, Alaettinoglu *et.*

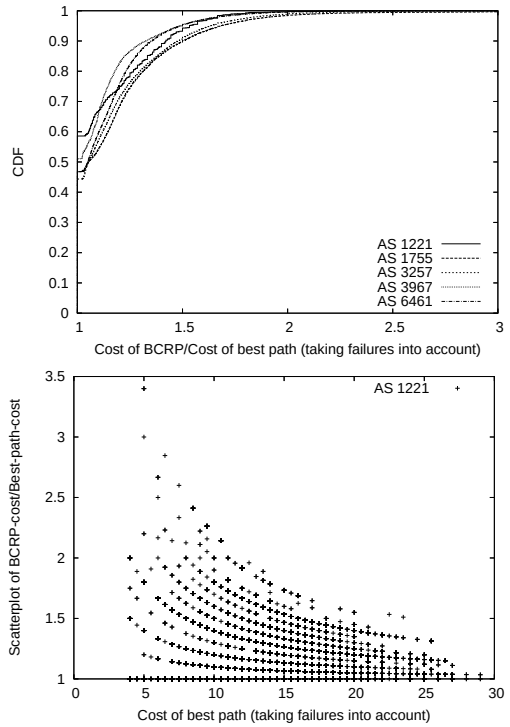

Figure 5: Top: CDF of the BCRP stretch (ratio of cost of BCRP to the cost of the best possible path in the graph with failed links removed). Bottom: Scatterplot of the same ratio, with cost of best possible path in the x-axis. This plot shows that the ratio is high only for very short, low-cost, paths, and longer paths incur low stretch.

al. [2] propose how IGP implementations can be cranked up to reduce convergence time to a few milliseconds even when links fail. However, such tweaks are restricted by protocol constraints—for example, arbitrarily reducing the timer values for detecting change in link status could potentially make routes oscillate due to false positives in detecting failed links. Furthermore, adjusting link weights in OSPF can temporarily destabilize the network, even with fast convergence, because often multiple weights need to be adjusted simultaneously.

There has been lot of theoretical research on achieving reliable communication in the presence of byzantine faults (such as [3, 9, 13]). However, these protocols have high overhead, and are not practical at high link speeds. Much of the recent research on secure routing has been directed at BGP in particular. In contrast to these works, BCRP leverages a practical assumption—that a centralized node can transmit the consistent map to all nodes—to achieve its security property. BCRP's data path involves no cryptographic operations; the reliable transmission of the consistent map requires some cryptographic operations, but the process is infrequent. While the security property of BCRP is less strict than those offered by some prior theoretical works, we believe that BCRP is useful in practice, as it restricts the damage caused by a malicious node to only the traffic that the node itself forwards.

6 Conclusion

In this paper, we proposed the separation of the two important goals of routing—connectivity and flexibility. The central idea is that an underlying protocol (Basic Connectivity Routing Protocol or BCRP) provides connectivity in the face of failures as well as misconfigured (or malicious) nodes. Sophisticated routing protocols (such as OSPF, EIGRP, etc.) can be layered on top of BCRP.

We present some experimental results using Rocketfuel data that suggest that BCRP is practical. However, there are many more dimensions to evaluate and understand BCRP. We are currently working to get detailed failure data from an ISP in order to study how useful BCRP is in practice. Theoretically understanding how much BCRP benefits based on the properties of the graphs would also be useful.

In this paper, we discussed the applicability of BCRP to intra-domain routing. We are currently exploring how to extend BCRP to inter-domain routing. The fact that the key observations from intra-domain case have analogues in the inter-domain case—that the AS graph changes infrequently, that AS-relationships can be published globally since they can be inferred anyway [12][3]—gives one hope.

References

[1] NANOG Mailing List Archives. `http://www.merit.edu/mail.archives/nanog/`.

[2] C. Alaettinoglu, V. Jacobson, and H. Yu. Towards Millisecond IGP Convergence. IETF Draft, 2000.

[3] D. Dolev, C. Dwork, O. Waarts, and M. Yung. Perfectly Secure Message Transmission. *Journal of the ACM*, 1993.

[4] P. Franciosa, D. Frigioni, and R. Giaccio. Semi-dynamic shortest paths and breath-first search in digraphs. In *STACS*, 1997.

[5] G. Iannaccone, C. Chuah, S. Bhattacharyya, and C. Diot. Feasibility of IP Restoration in a Tier-1 Backbone. *IEEE Networks, Special Issue*, March 2004.

[6] R. Mahajan, N. Spring, D. Wetherall, and T. Anderson. Inferring Link Weights using End-to-End Measurements. In *Proc. IMW*, 2002.

[7] P. Narvaez, K.-Y. Siu, and H.-Y. Tzeng. New Dynamic Algorithms for Shortest Path Tree Computation. *IEEE/ACM Transactions on Networking*, 2000.

[8] P. Pan, G. Swallow, and A. Atlas. Fast Reroute Extensions to RSVP-TE for LSP Tunnels. RFC 4090, May 2005.

[9] R. Perlman. *Network Layer Protocols with Byzantine Robustness*. PhD thesis, MIT, 1988.

[10] A. Shaikh, C. Isett, A. Greenberg, M. Roughan, and J. Gottlieb. A Case Study of OSPF Behavior in a Large Enterprise Network. In *Proc. IMW*, 2002.

[11] N. Spring, R. Mahajan, and D. Wetherall. Measuring ISP Topologies with Rocketfuel. In *Proc. of SIGCOMM*, 2002.

[12] L. Subramanian, M. Caesar, C.-T. Ee, M. Handley, M. Mao, S. Shenker, and I. Stoica. HLP: A Next-generation Interdomain Routing Protocol. In *Proc. SIGCOMM*, 2005.

[13] L. Subramanian, R. H. Katz, V. Roth, S. Shenker, and I. Stoica. Reliable Broadcast in Unknown Fixed Identity Networks. In *Proc. PODC*, 2005.

[3] There is incentive for ASes to publish peering information, since otherwise, those links would not be used by BCRP.