Achieving Convergence-Free Routing using Failure-Carrying Packets

Karthik Lakshminarayanan
University of California, Berkeley

Matthew Caesar
University of California, Berkeley

Murali Rangan
University of California, Berkeley

Tom Anderson
University of Washington

Scott Shenker
University of California, Berkeley

Ion Stoica
University of California, Berkeley

ABSTRACT

Current distributed routing paradigms (such as link-state, distance-vector, and path-vector) involve a convergence process consisting of an iterative exploration of intermediate routes triggered by certain events such as link failures. The convergence process increases router load, introduces outages and transient loops, and slows reaction to failures. We propose a new routing paradigm where the goal is not to reduce the convergence times but rather to eliminate the convergence process completely. To this end, we propose a technique called Failure-Carrying Packets (FCP) that allows data packets to autonomously discover a working path without requiring completely up-to-date state in routers. Our simulations, performed using real-world failure traces and Rocketfuel topologies, show that: (a) the overhead of FCP is very low, (b) unlike traditional link-state routing (such as OSPF), FCP can provide both low loss-rate as well as low control overhead, (c) compared to prior work in backup path precomputations, FCP provides better routing guarantees under failures despite maintaining lesser state at the routers.

Categories Subject Descriptors

C.2.2 [**Computer-Communication Networks**]: Network protocols–*Routing protocols*; C.2.6 [**Computer-Communication Networks**]: Internetworking

General Terms

Algorithms, Design, Performance.

Keywords

Internet routing, convergence, protocols.

1. INTRODUCTION

Recent large-scale deployments of delay and loss-sensitive applications have led to stringent demands on routing. Lost or delayed packets in applications such as Voice over IP (VoIP), streaming media, gaming, and telecommuting/video conferencing applications can result in significant performance degradation. ISPs hence have strong incentives to reduce delay and loss on their networks, as these are often key metrics used when negotiating Service-Level Agreements (SLAs) associated with such applications. Routing convergence is one of the key impediments to meeting strict SLAs.

Traditional routing paradigms—distance-vector, path-vector, and link-state—differ substantially in the nature of the state maintained by and exchanged between routers. However, all these paradigms rely on protocol messages to alert routers about changes in the network topology. It is only after the news of a topology change has reached all routers, directly in the case of link-state and indirectly in the case distance-vector and path-vector, that the protocol can ensure that the forwarding tables define consistent routes between all pairs of nodes. Thus, all such routing protocols experience a *convergence* period—after the change has been detected and before all routers learn about the change—during which the routing state might be inconsistent.

While the convergence process is invoked whenever link costs change, link and router failures are the events that cause the most serious problems. They can cause losses [3] and, in some cases, trigger *LSA storms*, resulting in high CPU and memory utilization in routers and increased network instability [10]. Though the convergence period fundamentally depends on network properties such as the diameter of the network, it is exacerbated in practice due to system-level issues such as protocol timers.

The attempts to solve this problem in the literature can be roughly classified into three categories: (a) designing loop-free convergence protocols, (b) reducing the convergence period of protocols, and (c) using precomputed backup paths to route around failures. The first category of proposals involves protocol changes (such as ordering of LSAs [15]) to ensure that the convergence process does not cause transient loops. The second category involves reducing convergence period by tweaking protocol parameters (such as LSA propagation timers and periodicity of HELLO messages) [3]. These mechanisms often achieve lower convergence times but at the expense of additional overhead, and lower stability (as we show in our experiments). The third category deals specifically with link failures by precomputing backup paths for links, which can be used when the link in question fails [18, 19, 28]. Recently, R-BGP [20] proposes using precomputation-based backups for fast-failover during BGP convergence; R-BGP also provides provable guarantees such as loop-prevention. These backup mechanisms typically deal with the failure of single links gracefully; however, in order to provide guarantees for simultaneous failures of multiple arbitrary links, the number of precomputed paths needed is extremely high.

Using the state-of-the-art techniques, the convergence period can be eliminated for single failures, and more generally the duration and impact of convergence can be reduced. While these changes are quantitatively beneficial, they do not change the qualitative fact

SIGCOMM'07, August 27–31, 2007, Kyoto, Japan.

that these protocols could (due to multiple failures) endure a convergence period during when it is hard to provide routing guarantees.

In this paper, we propose a different routing paradigm, called Failure-Carrying Packets (FCP) that *eliminates* the convergence period altogether. Once a failure is detected locally, packets are guaranteed to be routed to their destination as long as a path to the destination exists in the network.

FCP takes advantage of the fact that network topology in the Internet does not undergo arbitrary changes. In intradomain ISP networks and in the AS-level Internet graph there is a well-defined set of *potential* links (*i.e.,* those that are supposed to be operational) that does not change very often. The set of these potential links that are actually functioning at any particular time can fluctuate (depending of link failures and repairs), but the set of potential links is governed by much slower processes (*i.e.,* decommissioning a link, installing a link, negotiating a peering relationship). Thus, one can use fairly standard techniques to give all routers a consistent view of the potential set of links, which we will call the Network Map. FCP hence adopts a link-state approach, in that every router has a consistent network map.

Since all routers have the same network map, all that needs to be carried by the packets is information about which of these links have failed at the current instant. This *failure-carrying packets* approach ensures that when a packet arrives at a router, that router knows about any relevant failures on the packet's previous path. This eliminates the need for the routing protocol to immediately propagate failure information to all routers, yet allows packets to be routed around failed links in a consistent loop-free manner. We also present a variant called Source-Routing FCP (SR-FCP) that provides similar properties even if the network maps are inconsistent, at the expense of additional overhead in packet headers.

Though we present FCP to introduce a new routing paradigm that is qualitatively different from previous approaches, we show through simulation that it has the potential to provide quantitative benefits as well. Using real-world ISP topologies and failure data, we show that the overhead of using FCP — in terms of computation, overhead in packet headers, and stretch incurred — is very small. We also compare FCP with OSPF and show that, unlike OSPF, FCP can simultaneously achieve both low loss and low overhead. Finally, we show that compared to prior work in backup path precomputations, FCP provides much lower loss-rates while maintaining less state at the routers.

Though we present FCP as a link-state protocol, an approach which applies directly to intradomain and enterprise routing, we believe that the same idea can be used for interdomain routing as well. To this end, we outline a strawman proposal for applying FCP to interdomain routing in Section 7. We leave a complete study of applying FCP to interdomain routing for future work.

2. FAILURE-CARRYING PACKETS

We now introduce the FCP algorithm and its properties using a simple network model where routers use link-state routing.

In FCP, all nodes (we will use the terms router and node interchangeably) in the network maintain a consistent *Network Map*, which represents the link-state of the network; we relax the map consistency assumption in Section 2.2. In the absence of failures, FCP reduces to a link-state protocol; when failures occur, FCP behaves quite differently, as we now explain. For the purpose of our discussion, we assume that all nodes know the network map, and, unless otherwise specified, that this map does not change. We discuss how the network map is disseminated and updated in Section 4.

```
Initialization: pkt.failed_links = NULL
Packet Forwarding:
  while (TRUE)
    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)
      return
```

Figure 1: **Basic FCP protocol.**

2.1 Basic FCP design

The main intuition behind FCP is that it is enough for a router to know the list of failed links in the network, in addition to the network map, to compute the path to a destination. FCP uses the packet header to gather and carry the list of failed links required for routing *that* packet. As we show later, the packet needs to carry only the failed links that the packet has so far encountered along its path, not all failed links in the network, in order for this to work. Thus, the number of failures carried in any packet header is typically very small.

Figure 1 shows the pseudocode of the basic FCP protocol. When a packet arrives at a router, its next-hop is computed using the network map minus the failed links in the header. If this next-hop would send the packet out an interface that has a failed link, then the router: (1) inserts the failed link into the packet header, (2) recomputes the route using this new failure information, and (3) returns to step one if the new next-hop also incurs a failure and, if not, forwards the packet to its next-hop. Note that each packet is treated separately; *the failure information contained in a packet is not incorporated into the routing tables*.

To understand FCP better, consider the example in Figure 2, a network with unit link weights. Assume N_1 sends a message to N_d, and that links $N_3{-}N_d$ and $N_5{-}N_7$ are down. Since only nodes adjacent to the failed links know about the failure, the packet is forwarded along the shortest path in the original graph, (N_1, N_2, N_3, N_d), until it reaches the failed link $N_3{-}N_d$. At this point, N_3 computes a new shortest path to N_d based on the map minus link $N_3{-}N_d$, and includes the failed link $N_3{-}N_d$ in the header. Let us assume that this path is (N_4, N_5, N_7, N_d). When the packet reaches N_5, N_5 adds the failed link $N_5{-}N_7$ to the header, and computes a new shortest path that does not include the two failed links. Eventually the packet reaches the destination, N_d, along this path.

In general, there are two possibilities when a packet hits a failed link: 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 graph on which routers compute the path becomes smaller (*i.e.,* because it does not include the failed link).[1] With every new failed link inserted in the packet header, the graph over which the packet is routed becomes monotonically smaller.

2.2 Source-Routing FCP (SR-FCP)

Basic FCP assumes that all nodes have the same map. We now relax this assumption, by presenting an alternate design that employs source-routing. With source-routing FCP (SR-FCP), the first

[1] There is a third possibility that arises due to congestion: if the router cannot hold on to the packet due to resource limitations, packets might be dropped.

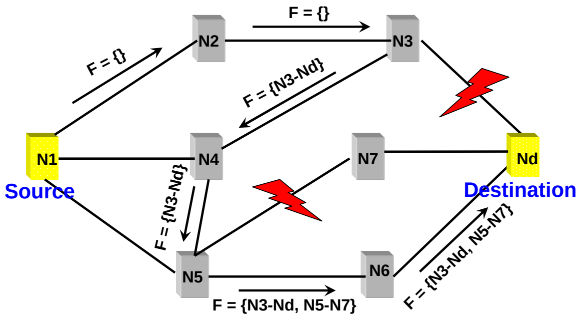


Figure 2: **An example illustrating FCP routing.**

router on the packet path inserts the entire route to the destination in the packet header. Subsequent routers simply forward the packet based on the source route in the packet header until the packet either reaches the destination or encounters a failed link. In the latter case, the node adds the failed link to the packet header (exactly like basic FCP), and replaces the source route in the header with a newly computed route, if any exists, to the destination.

The main advantage of SR-FCP over FCP is that it works correctly even when not all nodes have the same network map. Thus, SR-FCP does not require that all nodes have the same map. A second advantage is that, unless there is a link failure, packet forwarding does not require a lookup operation, and thus can be implemented much faster in practice.

On the downside, SR-FCP increases the packet overhead, by requiring each packet to carry the source route. Furthermore, the inconsistency across maps can significantly increase the list of failed links, as any link that does not appear in all maps can be potentially marked as a failed link.

For the sake of simplicity, most of our discussion will focus on basic FCP, and then we will discuss briefly the properties of this alternate SR-FCP approach.

2.3 Properties

We present two key properties of FCP: *guaranteed reachability* and *path isolation*. Informally, the reachability property says that as long as the network is connected and there are no packet losses due to congestion, every packet is guaranteed to reach its destination despite any link failures. The path isolation property says that a malicious node cannot impact the path followed by a packet unless it is already on that path. Finally, we show that SR-FCP can provide these properties even when the node maps are inconsistent.

Since a failed node can be represented as a node whose entire set of links has failed, in the remainder of this section we consider only link failures. Furthermore, unless otherwise specified, we consider only fail-stop failures[2], and assume that FCP employs shortest path routing. To state FCP's properties more precisely, we start with the following definition.

DEFINITION 1. *Let M be the network graph (map). Define the liveness graph $LG(t_1, t_2)$ as the maximal graph consisting of only nodes and links of M that are alive at all times during the closed time interval $[t_1, t_2]$.*

Note that once a link goes down during $[t_1, t_2]$, it is removed from $LG(t_1, t_2)$ irrespective of whether the link comes back again or not. This aspect captures the fact that, in FCP, once a failed link is added to the packet header, it is never removed from the header.[3]

Next, we give sufficient conditions that guarantee packet delivery in FCP.

LEMMA 1. **Guaranteed Reachability:** *Consider packet p entering network M at time t_1. Assume link failures are detected instantaneously, there are no packet losses due to network congestion, and the propagation delay over any link is one time unit. Let $d(G)$ denote the diameter of graph G.*

Then, FCP guarantees that p will be delivered to the destination by time t_2, where t_2 is the smallest time, if any, such that the following two conditions hold:

1. *there are at most f failures during $[t_1, t_2]$, where $f \leq (t_2 - t_1)/d(LG(t_1, t_2)) - 1$*
2. *$LG(t_1, t_2)$ is connected and spans (all nodes of) M*

PROOF. The main part of the proof is to show that packet p is delivered to its destination by *some* time t_2 that satisfies conditions (1) and (2). From here it follows trivially that packet p will be delivered by the smallest value of such t_2.

The proof is by contradiction. Assume p is *not* delivered to the destination by a time t_2 that satisfies both conditions (1) and (2).

We start with *two* observations. The first observation is that a packet will not encounter the same failed link twice. This follows from the fact that once a packet encounters a failed link l, this link is carried in the packet header, and each subsequent path computation will avoid l.

The second observation is that any packet forwarded during $[t_1, t_2]$ will take at most $d(LG(t_1, t_2))$ time to either reach the destination, or encounter a (new) failed link. This is because, unless a new failure is encountered, every node uses the same map and failed link list (*i.e.,* the one carried by the packet) to forward the packet along the shortest path. Furthermore, at any time $t \in [t_1, t_2)$, the shortest path is at most $d(LG(t_1, t))$. This is because, from condition (2) and definition 1, both $LG(t_1, t_2)$ and $LG(t_1, t)$ span M, and $LG(t_1, t)$ includes all links of $LG(t_1, t_2)$, which yields $d(LG(t_1, t_2)) \geq d(LG(t_1, t)), \forall t \in (t_1, t_2)$.

Let k be the number of link failures encountered by packet p during interval $[t_1, t_2]$, where $k \leq f$ by hypothesis. After encountering the k^{th} failure, the packet is routed along the shortest path to destination. Since there are no packet losses, the only reason p may not reach the destination is either because (a) p encounters another link failure, or because (b) some node A, tries to forward p but does not have a route to D in the network map minus the list of failed links. However, (a) cannot be true, since by the second observation, p would encounter the $(k+1)^{th}$ failure by time $t_1 + (k+1) \times d(LG(t_1, t_2)) \leq t_1 + (f+1) \times d(LG(t_1, t_2)) \leq t_2$, which violates condition (1). Similarly, (b) cannot be true as it implies that the liveness graph is disconnected at some point during the interval $[t_1, t_2]$, which violates condition (2). This completes the proof. □

Note that FCP can fail even if there is a viable path in $LG(t_1, t_2)$, but this can only occur if the $LG(t_1, t_2)$ is disconnected and the packet-in-flight gets stranded on a disconnected component. As an example, consider Figure 1. As before, N_1 initially sends the packet to N_2. However, at this instant, let the links $N_1 - N_2$, $N_3 - N_d$ and $N_3 - N_4$ all go down. Since N_2 and N_3 are disconnected from the destination they cannot route the packet to N_d despite the fact that

[2] Under the fail-stop assumption, processes fail by halting and failures are easily detectable.

[3] FCP does not reuse failed links to avoid cycles.

the path $N_1 - N_5 - N_6 - N_7$ was always active. Note that condition (2) in the above Lemma filters out this scenario, as it requires $LG(t_1, t_2)$ to span the entire graph.

In today's protocols, malicious routers can send fake route updates, and hence subvert a network to cause more packets to flow through them [17, 31]. In FCP, once the map is uploaded to each node, there are no dynamic link updates that nodes exchange to modify this map. 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 which is not on the packet's path, as computed by FCP, cannot affect the fate of the packet. The next lemma states this property.

LEMMA 2. **Path isolation**: *Assuming the map distribution is secure, malicious nodes cannot perform off-path attacks.*

PROOF. The proof follows directly from the fact that an off-path node has no way to contaminate the routing state of the nodes along a packet's path. This is because nodes along the packet's path compute the route *solely* based on the disseminated map and the list of failed links in the packet header. □

The main assumption we make here is that the map dissemination is much less frequent than route updates in today's routing protocols, and thus we can afford to improve the security of the map dissemination operation, even at the expense of an increased overhead.

Note that the path isolation property does not provide security guarantees against arbitrary attacks. For example, a malicious node can still mount denial of service attacks by sending spurious packets with large lists of fake failed links in the hope of overloading the CPUs of its neighbors. This attack is similar to a malicious node sending a large number of fake routing updates to its neighbors.

The next result shows that SR-FCP is able to provide these properties, even in the presence of inconsistent maps. In this case, the properties apply to the graph defined by the intersection of all maps in the system. Intuitively, this is because SR-FCP potentially treats any link that is not in all maps as a failed link. In particular, if a link l in a packet's source route is not in the map of a node A that forwards the packet, A simply adds l to the list of failed links.

LEMMA 3. *Consider a network where the maps maintained by nodes are not necessarily consistent. Redefine the notion of link failure to include every link that does not belong to all node maps.*

Using the new definition of link failure, SR-FCP achieves both the guaranteed reachability and the path isolation properties, as stated by Lemmas 1 and 2, respectively.

PROOF. The proof for guaranteed reachability is similar to the proof of Lemma 1. The only difference is that, in this case, nodes may have different maps. However, by using source routing, we ensure that the two observations in Lemma 1 are still true. Let A be the node that has computed and inserted the source route in a given packet p. Since, in the route computation step, A eliminates the failed links encountered by p so far, p will not encounter the same failed link twice. Furthermore, every subsequent node along p's path uses the source route inserted by A to route packet p until either p reaches its destination or encounters another failed link. Since the map used by A to compute the source route of p is a superset of $LG(t_1, t_2)$ (where times t_1 and t_2 are as defined in Lemma 1)[4], it follows that it takes p at most $d(LG(t_1, t_2))$ to reach the destination or the next failed link.

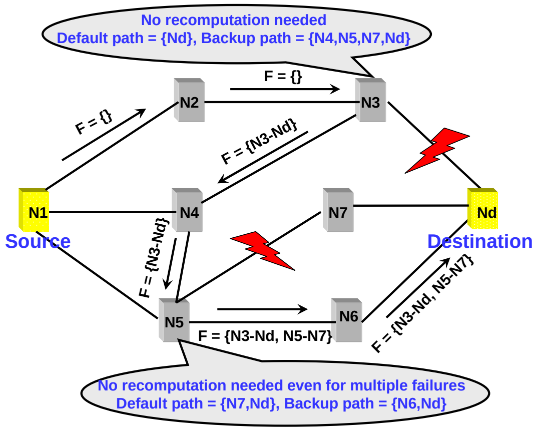

No recomputation needed
Default path = {Nd}, Backup path = {N4,N5,N7,Nd}
F = {}
F = {}
F = {N3-Nd}
N1
N2
N3
N4
N5
N6
N7
Nd
Source
Destination
F = {N3-Nd}
F = {N3-Nd, N5-N7}
F = {N3-Nd, N5-N7}
No recomputation needed even for multiple failures
Default path = {N7,Nd}, Backup path = {N6,Nd}

Figure 3: An example in which a packet experiences multiple link failures but recomputation is not necessary.

The proof of the path isolation property follows again from the fact that an off-path node has no way of contaminating the routing state of the nodes along packet's path. □

2.4 Challenges

We have described the basic algorithm and its fundamental properties. In the rest of the paper, we address the main challenges of realizing FCP.

- Computational overhead (Section 3): Whenever a packet carrying failure information arrives at a router, the router needs to compute new routes. We present mechanisms to reduce the computation overhead significantly.
- Map dissemination and updates (Section 4): FCP relies on all routers having a consistent view of the network map, which requires a map dissemination and update protocol.
- Quantitative performance (Section 5): While FCP's correctness properties might be theoretically appealing, to have practical relevance, FCP must be compared quantitatively to OSPF performance, as well as backup path techniques that are commonly used in operational networks today.
- Deployment (Section 6): For FCP to have practical implications, the mechanisms should be deployable with minimal changes to the infrastructure. We discuss how we can leverage currently deployed mechanisms (such as MPLS), and several earlier proposals (such as RCP) to achieve our goals.
- FCP extensions (Section 7): Since much of the paper discusses FCP as a link-state routing protocol, it is directly applicable only in the intradomain context. We discuss how FCP can deal with incomplete maps and with policy constraints needed for interdomain routing.

3. REDUCING OVERHEAD OF FCP

Basic FCP requires computation for every packet that encounters a failure at every node that the packet traverses. We present several mechanisms that reduce the overhead significantly by adding only a small overhead to router state.

3.1 Reducing per-packet route computation

To reduce per-packet computation at nodes where failures are encountered, nodes perform some precomputation. Each node (in addition to the default forwarding table), for every adjacent link

[4] By the definition of link failure in Lemma 3, $LG(t_1, t_2)$ does not contain any link unless the link is present in all maps, including the A's map.

l, computes the forwarding table using the consistent map minus l; this table is used when l is failed. However, in terms of actual forwarding state, such a precomputation *only doubles* the memory requirement: for each destination, in addition to the default path P computed using the map, we need to store the precomputed path computed using map minus l_P, where l_P is first hop in P.

LEMMA 4. *If a packet p encounters a failed link l at node N, and the precomputed path P_l to the destination (using the consistent map minus l) does not contain a link that belongs to the set of failed links that p carries, then P_l can be used to route p to the destination, and no recomputation is necessary.*

PROOF. Proof follows from the fact that a shortest path is unaffected by removing a link not contained in that path. □

Figure 3 illustrates the intuition behind the above technique. When the packet reaches node N_5, multiple failures are encountered. But since the backup path at N_5 for the failed link N_5-N_7 does not traverse N_3-N_d, recomputation is not necessary. In other words, recomputation is needed only when failures happen on both the primary and backup path. Hence, when the fraction of failed links is small, the chance that a recomputation is triggered is low.

3.2 Reducing recomputation time

Each node maintains a cache of the paths that it computes based on failures seen in packets. For each combination of failures, a node performs computation to find shortest paths only once. This is because performing a shortest path computation on $M\backslash F$, where M is the map, and F is the list of failed links, yields shortest paths to all destinations.

For performing recomputation, we borrow from the literature on incremental recomputation [13, 25]. Prior research has shown that incremental recomputation can be performed within the order of a few milliseconds even for graphs with a thousand nodes [3]. Performing recomputation within a few milliseconds is very reasonable; since failure detection itself could take that much time, 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.

3.3 Reducing packet overhead

We now present a mechanism to reduce packet overhead further at the expense of local mapping state at the nodes. Consider a node N_1 sending a failure header that includes a set of failed links F to node N_2. With the failure header F, the node N_1 associates a label l_f, and includes the mapping $l_f \rightarrow F$ when it sends the packet to N_2. After N_1 receives an acknowledgment from N_2 about the mapping, N_1 includes only the label l_f rather than the entire failure header F. Labels don't have global meaning but are specific to a pair of adjacent nodes. Since labels are allocated on-demand for each new combination of failures that is encountered, the number of labels needed is atmost the number of different failure combinations a router encounters, which is small in practice. For robustness, a time-to-live value T can be associated with a label; if no packet with a particular label is seen for period T, the label is removed.

4. DISSEMINATION OF NETWORK MAPS

We now turn to the problem of disseminating the global network map to all nodes periodically. The purpose of the map is to provide all routers with a loosely-synchronized but globally consistent view of network state.

4.1 Network map information

To reduce overhead as well as react quickly to changes, the network map does not include transient changes to the network. For instance, if a link fails temporarily for a short duration, it is not removed from the map. Rather, only long-term updates such as planned outages and newly provisioned links are published in the map (short-term updates are handled by the FCP protocol described in previous sections). In order to reduce bandwidth consumption, only the difference from the previous version of the map can be disseminated.

4.2 Basic map dissemination

The map is disseminated by an RCP-like *coordinator* [9] using reliable flooding. The coordinator sends the map via TCP to a set of nodes, which in turn sends the map to their neighbors along the outgoing links in the map, and so on. To avoid receiving the map multiple times form its neighbors, a node can ask a neighbor to cancel the map transmission once the node gets the map from another neighbor. If a node is down during the map distribution, the node will get the map from its neighbors once it comes up.

To ensure path isolation property (see lemma 2), we use public key cryptography, where the coordinator signs each map with its private key. The coordinator's public key is distributed to all nodes in the network either out-of-band (*e.g.,* manually) or via a public-key infrastructure (PKI), if available. Since map dissemination is a relatively rare event, we believe that the overhead imposed by the signature operations will be acceptable. Furthermore, since the overhead on the coordinator is relatively independent on the network size, we expect the coordinator to scale to large network sizes.

While we have described how maps are disseminated, the following challenges remain, which we address next:

- How do nodes decide when to switch to a new version of the map? How does routing happen during the window of switching maps when not all nodes route on the same version?
- How do we make the coordinator resilient to failures?

4.3 Transitioning to new map

We first present a protocol that tries to ensure that all nodes route using the same map. Then, we describe a mechanism that ensures correct routing even when the protocol fails to transition all nodes to the new map simultaneously.

When each node receives a new map, it sets a local timer that expires after a period T_{lp}, where T_{lp} is a conservative estimate of the diameter of the network. A node switches to a new map either when: (a) the timer expires, or (b) it receives a packet that is routed using a new map. Intuitively, when the timer at the first node that receives the map expires, all nodes would have received the new network map completely. Hence, when it starts routing using the new map, all nodes that route packets would switch to the new map. This cascading process quickly resulting in the entire network switching to the new map depending on the amount of traffic flowing in the network. In the worst-case, all nodes would switch to the new map within T_{lp} of each other.

In practice, for an intradomain topology spanning an entire continent, we expect that the longest shortest-path in the network would be no larger than a few hundred milliseconds (for comparison, one-way delay within the continental United States is roughly 50ms).

Hence, T_{lp} can be conservatively chosen to be a few seconds to account for processing delays at each node as well.

4.3.1 Packet forwarding during map transitions

We now present a practical packet routing method using the map update process that works even in the case when the map-dissemination exceeds T_{lp}. Here, we assume that every node maintains not only the latest map, but the previous version of the map as well. The basic idea is to downgrade a packet to using the previous version of the map if the new map is not completely disseminated.

Each forwarded packet contains a sequence number that indicates the sequence number of the map used to route that packet. When a node starts routing a packet, its current active network map (which is either the most recent map it has received or the previous version). Let a node n receive a packet from n'. Let the sequence number contained in the packet be s', the sequence number of the active map at the current node be s, and the largest received sequence number received at the current node be s_{max} ($s_{max} \geq s$).

> *Case (a)* $s' < s$: Forward the packet using $M_{s'}$.
>
> *Case (b)* $s \leq s'$: Set $s{=}min(s', s_{max})$, and forward the packet using M_s.

Using this protocol, the originator starts forwarding the packet p using its active map. In the worst case, p is demoted to using the previous map if the latest map is not fully disseminated. Map demotion would happen only when the timer expires before all nodes get the map, and should not happen with conservatively chosen timeouts. Even if a packet is demoted, the protocol's basic correctness is not affected. During map transition, all nodes would not switch version numbers exactly at the same time. However, eventually all nodes would update to the new map, and eventually routing consistency is guaranteed. As noted earlier, if a node n receives a packet with a new sequence number s_{max} before its timer for s_{max} expires, then n updates s to s_{max}.

4.4 Coordinator fault-tolerance

We now describe a mechanism for replicating the coordinator to improve fault tolerance. The main idea is to use replicated coordinators (in a similar manner to the RCP design) with replicas having a replica number that denotes the ordering of the replica.[5] For example, replica 1 takes precedence over replica 2. Each replica independently chooses a map M (all of them execute the same algorithm, and hence should pick the same map, though they may not be synchronized), and disseminates M skewed in time such that replica 1 sends the map update at time $t{=}0$, replica 2 sends the update at time $t{=}T/n$, replica 3 at time $t{=}2T/n$ and so on, where T is the periodicity of updates that each replica sends, and n is the number of replicas. Each node floods only the map of the replica with the lowest sequence number such that the map is no older than time $T{+}c$, where $c > 0$ is the maximum time for the map to be disseminated across the network. The node suppresses the other maps it receives to save bandwidth resources.

For routing, each node in the system would use the map from the lowest numbered replica with the highest sequence number such that the map is no older than time $T{+}c$, where c is defined as above. Even if there are network partitions, within all connected components, eventual routing consistency would be reached. Partitions heal at the network layer during subsequent updates.

4.5 Periodicity of map updates

The map update period would depend on how frequently links are decommissioned or added, planned network outages, and frequency at which link costs change (say, for traffic engineering). In the extreme case, one could just disseminate the map as frequently as the default OSPF LSA generation period (which is typically 30 seconds). Unlike OSPF, the coordinator can send only the difference from the previous network map to save bandwidth resources. The only limitation is the overhead to sign the map.

5. EVALUATION OF FCP

We first present our experimental methodology, the data sets we used, and protocol configurations we used for comparison purposes. We present results in two parts. In the first part, we present results that show that the overhead of FCP is very small. In the second part, we compare FCP with OSPF as well as backup computation techniques. Finally, we present the overhead involved in using SR-FCP as a function of degree of inconsistency in the maps at the routers. To summarize our results:

- The overhead of FCP is very low both in terms of computation overhead and packet header overhead.
- Unlike traditional link-state routing (such as OSPF), FCP can provide both low loss-rate as well as low control overhead,
- Compared to prior work in backup path precomputations, FCP provides better routing guarantees under failures despite maintaining lesser state at the routers.

Table 1: Protocol configuration parameters

Parameter	OSPFD	FCP/Backup computations	OSPFD default
hello-interval	400 ms	50 ms	1 sec
dead-interval	2 sec	250 ms	5 sec
retransmit-interval	2 sec	n/a	5 sec
throttle-interval	2 sec	n/a	5 sec

5.1 Methodology

Protocols: We compared FCP with two alternate strategies: the *OSPF* [23] link-state protocol, and an MPLS-like protocol that precomputes *backup-paths*. To compare with OSPF, we leveraged the OSPFD software router developed by John Moy [24], which completely implements the OSPF protocol as specified by RFCs 2328 and 1765 [22, 23]. We configured OSPFD following the millisecond-convergence recommendations given in [3], including the incremental Dijkstra's algorithm described in [25].

OSPFD also contains a network emulation toolkit for evaluating deployments, which we extended to support FCP and backup-paths implementations. To compare with backup-path precomputation, in our results, we use the sample selection algorithm used by Juniper Networks [19]. Although algorithms exist to compute optimal backup paths we found that such algorithms involved substantial computation time, which led to poor results when applied to the dynamically changing networks we consider here.

Configuration: To configure OSPFD's timers, we conducted a simulation study to determine settings that performed well on our workloads. By default, we configured OSPFD to send one probe every 400ms, and considered a link to be down if no probes are received for 2 seconds. To reduce sensitivity to flapping links, we

[5] In practice, since the coordinator performs tasks only at coarse timescales, having a small number of replicas for the coordinator might suffice.

configured OSPFD to "treat bad news differently from good news" by propagating link failures immediately but delaying propagation of link arrivals for five seconds. Given that FCP and the backup-path scheme do not propagate failure information globally, we configured them with faster probing times (one HELLO every 50ms). Each router sends pings to all other routers, with one ping every 15 seconds[6]. Finally, to fully stress FCP routing, the network maps are never updated in the experiments we present.

Data: Link failures and arrivals were driven by ISIS traces collected on the Abilene Internet2 backbone [1]. These traces contain timestamped Link State Advertisements (LSAs). We modified the network emulator to drive link failures by playing back LSAs based on their timestamps. To investigate sensitivity to failure rate, we artificially vary the rate at which LSAs are played back against our implementation. To evaluate larger networks and a wider range of parameters, we also used Rocketfuel [29] topologies and used a shifted Pareto distribution to drive the time-to-failure distribution for each link. We vary the mean failure interarrival time, while holding the mean failure duration fixed at 40 seconds (we found that varying the failure duration gave similar results to varying the interarrival time). Since Rocketfuel traces do not have link weights for router-level topologies, we assign each link a weight of 1. Though we had six AS topologies from the Rocketfuel data, we report results from AS 1239 (Sprint) Rocketfuel topology as a representative sample. The Sprint AS topology has 283 nodes and 1882 links.

5.2 Overhead of FCP

FCP introduces extra overhead only for packets that encounter a failed link. The overhead can be classified into: (a) network overhead, *i.e.,* stretch of routing the packet, (b) packet header overhead, and (c) recomputation overhead.

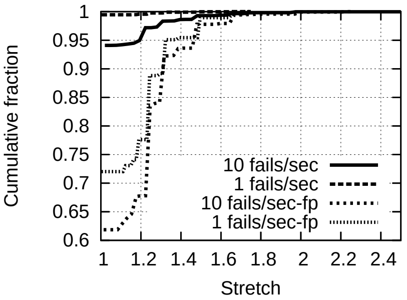


Figure 4: **Stretch for varying mean failure inter-arrival times (in seconds). FCP incurs a stretch penalty, but this penalty is small even at high link failure rates.**

Stretch: After failure, FCP does not necessarily discover the next-shortest path, incurring a stretch penalty. We define *stretch* to be the ratio of the number of hops traversed by the packet divided by the number of hops along the shortest working path between the source and destination. Figure 4 shows the CDF of stretch over all pairs of sources and destinations for increasing failure rates (where failure rate is measured by the number of links that fail per second in the network). The figure shows two pairs of CDFs, one pair for paths only affected by failures (marked '-fp') and the other pair for all paths. With 1 failure per second, 2% of paths were affected by failures, the average stretch was 1.07, and worst-case stretch was under 1.7. Even at very high failure rates (10 failures per second), the average stretch was 1.1 and the worst-case stretch was under 2; in this case, 11% of the paths were affected by failures.

Stretch increases with the rate of failures because the number of failed links a packet encounters increases. Hence, FCP is forced to reroute packets multiple times which reduces the chances that the optimal end-to-end path is taken. However, as shown later, we found this stretch was comparable to the backup-path selection strategy that we compared against.

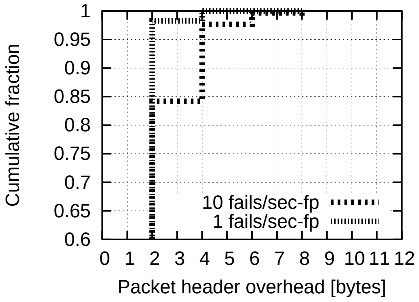


Figure 5: **Packet overhead of FCP.**

Packet header overhead: Figure 5 shows the CDF of minimum packet overhead incurred by the failure header with FCP, using the OC48 trace from CAIDA [2] to generate realistic traffic workloads. For the Abilene failure trace [1], FCP inflates the average packet size by a negligible amount. The maximum header size during the run is 8 bytes per packet, assuming each failure header takes 2 bytes. Although header sizes can potentially be larger in networks with more simultaneous failures[7], we can reduce the packet overhead by using the label optimization described in Section 3.3. It is important to note that headers are only added on link failure.

Recomputation overhead: Figure 6(a) shows a CDF of the number of recomputations per packet in a network with 1 link failure per second. Roughly 2% of packets require recomputation to be performed under the vanilla FCP implementation. Figure 6(b) shows the time to recompute paths after link failure as measured on a 3GHz Intel Pentium processor with 2GB of RAM. Recomputation time is below one millisecond on all topologies.

The number of on-demand recomputations performed depends on the failure rates: about 2% and 13% of the failure-affected paths require on-demand computation for failure rates of 1 and 10 link failures per second respectively. Note that the backup precomputation that we perform is specific to FCP and has much lower overhead than traditional backup precomputation strategies. This is because we only compute backup paths for adjacent link failures at each node as opposed to computing backups to protect against failures of combinations of links.

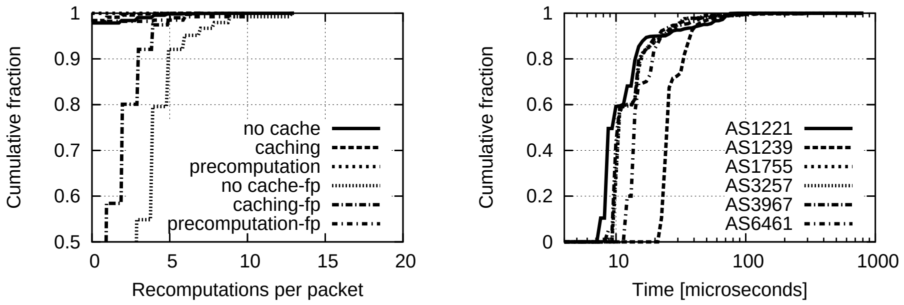
5.3 Benefits of FCP

5.3.1 Comparison with OSPF

In Figure 7(a), we vary the rate at which OSPFD sends HELLO packets to neighbors, and measure the effect on control overhead and the fraction of data packets delivered. We tuned FCP with a fast probing rate (one probe every 50ms) since faster probing does not incur a penalty in control overhead in terms of LSAs disseminated because none of the link failures detected are announced.

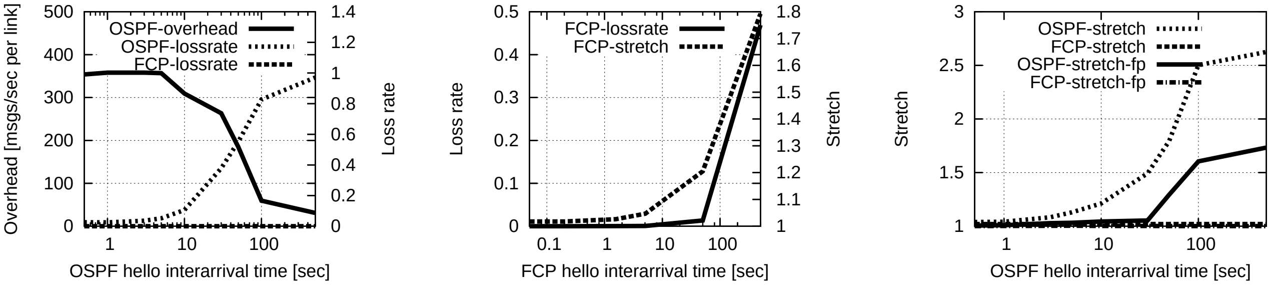
[6] We found that sending pings at a faster rate could overload the OSPFD implementation.

[7] The expected number of failures encounted by a packet is proportional to the diameter of the network.



(a) (b)

Figure 6: ***Recomputation costs of FCP:*** **(a) Number of recomputations. (b) Recomputation times.**



(a) (b) (c)

Figure 7: ***Comparison with OSPF:*** **(a) Unlike FCP, OSPF cannot simultaneously provide low control overhead and high availability, (b) Reducing FCP's HELLO timer reduces stretch and loss without increasing control overhead, (c) OSPF's map becomes inconsistent with the topology at low probing rates, resulting in a stretch penalty.**

As the HELLO probing rate is increased, the number of data packets lost decreases, since we react to failures more quickly. However, probing at a faster rate also causes more short-term failures to be detected and propagated, increasing control overhead. Note that when measuring control overhead, we factor out HELLO messages for both FCP and OSPF.

On the other hand, FCP undergoes only a very small (yet non-zero) loss-rate of less than 0.1%; the loss of FCP in our experiments is non-zero since link failure detection takes a finite time and during this period, all packets trying to use that link will be dropped. Although FCP exhibits similar behavior to OSPF in terms of stretch and loss rate while varying the probing rate (Figure 7(b)), its control overhead is not a function of link failure rate, and hence its probing rate can be increased without inflating control overhead. We found it was possible to tune OSPF to achieve this loss-rate, but only at the expense of a increasing its control traffic to above 300 messages per second per link.

The average stretch shows a similar result; as the probing rate decreases it takes longer to detect link repairs, and hence a larger fraction of working paths are not discovered by a packet. This is shown by the first two lines in Figure 7(c). Since we can keep the FCP probing rate high without compromising overhead, FCP has a much lower stretch than OSPF. The low stretch is due to FCP's ability to discover efficient secondary routes, rather than the fact that most paths are not affected by failure. To illustrate this point, we plot the results only over packets whose primary paths are failed (see the bottom two lines in Figure 7(c) that are marked '-fp').

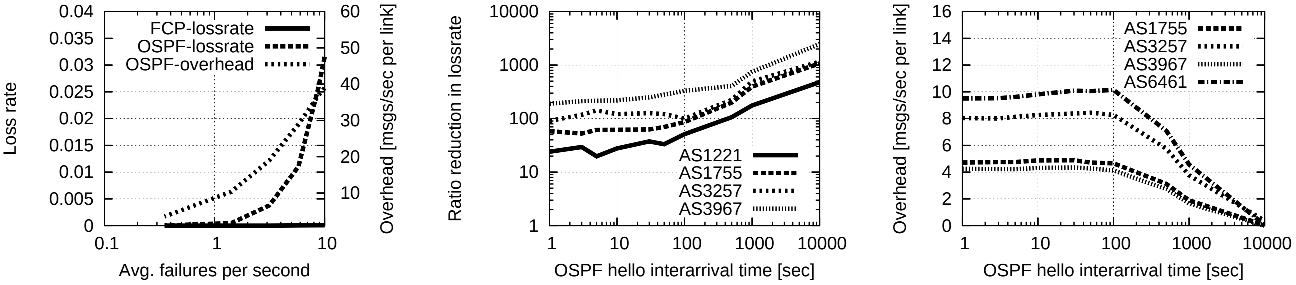
5.3.2 *Effect of varying parameters*

In Figure 8(a), we vary the mean interarrival time for link failures, but fix OSPF's probing interval at 400ms, and plot the loss rate and overhead. For a wide variety of failure rates, FCP outperforms OSPF by an order of magnitude, while simultaneously maintaining a lower control overhead. Note that OSPF's overhead begins decreasing as the failure rate increases past the ability of the probing protocol to keep up with link events.

In Figure 8(b), we vary the probing rate and plot the fraction increase in loss rate of OSPF over the loss rate of FCP for various topologies. Although the amount of improvement varies across topologies, FCP provides more than one order of magnitude lower loss rate than OSPF. As shown in Figure 8(c), FCP also reduces control overhead. In general, we found that denser topologies (e.g. AS 1221, with an average degree of 6.2) had less benefit from FCP than sparser topologies (e.g. AS 3257, with an average degree of 3.7). This happens because in denser topologies OSPF has a larger number of paths to choose from, and is hence more likely to discover a working path.

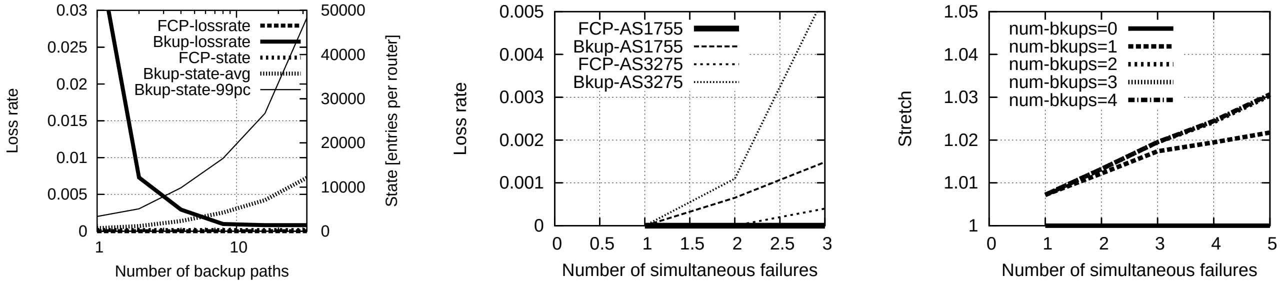
5.3.3 *Comparison with backup-path selection*

Unlike OSPF, the backup-path strategy we used can attain very fast failover times without a significant increase in control overhead. However, to minimize loss rates, the backup-path strategy needs to account for every failure contingency, and hence requires a substantial number of backup paths. Precomputing a large number of backup paths to account for different combinations of multiple link failures increases state per router. This tradeoff is shown in Figure 9(a). For example, with 8 backup paths per link, the backup path strategy requires 4210 entries per router, and experiences a loss-rate of 0.05%. However on the same workload, FCP requires only 255 entries yet attains a loss rate of less than 0.002%. Moreover, unlike FCP, the distribution in state across routers is not uniform, and hence the top 1% of routers require more than $20,285$ entries. However, for switching between maps, FCP must temporarily maintain a second copy of its routing state. Although this state is only main-



(a) (b) (c)

Figure 8: ***Effect of varying parameters:*** **(a) failure rate on control overhead and data packet loss rate. (b) topology on loss rate. (c) topology on control overhead.**



(a) (b) (c)

Figure 9: ***Comparison with Backup-paths:*** **(a) Unlike FCP, Backup-paths cannot simultaneously provide low state and low loss-rate. (b) FCP maintains low loss for a variety of failure rates. (c) Backup-paths stretch penalty for varying numbers of backup paths.**

tained for a short period, and can be stored as deltas (differences from the current map), in the worst case this could double FCP's state requirements (to 510 in this example).

Figure 9(b) shows the performance in the presence of simultaneous failures on two representative topologies. We fix the number of backup paths to two, and vary the number of randomly selected links to simultaneously fail. On both topologies, the backup strategy and FCP have roughly equal loss-rates during single failures. However, when more than one failure occurs, FCP has significantly lower losses. (FCP has non-zero loss since the failure detection is not instantaneous.) This happens because as the failure rate increases, it becomes more likely the backup-path strategy will encounter a set of failures not covered by the set of backup paths. Finally, Figure 9(c) shows that the backup-paths strategy incurs a stretch penalty during link failures. This happens because link-disjoint paths as used as backups, which tend to be longer than the path FCP finds around the failure.

5.3.4 Effect of inconsistent maps

So far, we have assumed that all nodes have a consistent state of the network map. Here, we investigate the overhead incurred by SR-FCP—in terms of average routing stretch and per-packet overhead—as a function of map inconsistency factor (see Figure 10). Specifically, for a chosen map inconsistency factor d, we instantiate the network map M_n at each node n by picking links randomly from the actual network map M, such that the intersection of maps at all nodes forms a spanning and connected subgraph of M, which contains only a fraction $(1 - d)$ of links in M (with high probability). The x-axis is capped at 0.3 since that is the largest fraction of inconsistency for which the intersection of maps at all nodes forms a spanning subgraph.

The plot shows that the stretch and packet overhead are small even when maps are highly inconsistent. Even when the inconsistency factor is 0.3, the average stretch is less than 1.03, and the average size of a packet header is under 10 bytes (assuming 2 bytes per node for source routes and failures). The reason SR-FCP performs so well is because SR-FCP pays a penalty only if the source node performing a route computation misses some links that could have resulted in significantly shorter paths; an intermediate node just forwards a packet based on the packet's source route irrespective of whether downstream links in that source route (not adjacent to the intermediate node) are present in the node's map or not.

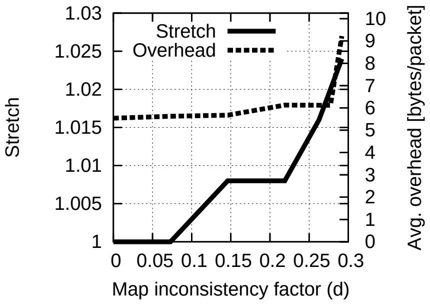


Figure 10: Effect of inconsistency in network maps on the overhead incurred by SR-FCP.

6. DEPLOYMENT ISSUES

FCP represents a substantial departure from traditional routing mechanisms, and hence unsurprisingly requires several modifications to router design even for deployment at the intradomain level. Although these changes are by no means trivial, in this section we outline how they may be implemented as extensions to existing protocols and designs.

Map dissemination: The role of the coordinator is akin to the centralized node in the case of RCP [9]. Such a design is amenable in case of ISP networks where the centralized administrative node can act as the map coordinator and periodically disseminates the network map. In addition, to better handle packet forwarding during map transitions (see Section 4.3.1), routers must store both the current and the previous map, instead of only the current map as most of the existing protocols do.

FIB state: If dynamic failure-based path computations are not cached, the FIB state is doubled since at the minimum, the next hop information should be maintained for current map and previous map. Even if path computations are cached, the average additional state required is not high. With the precomputation optimization for each outgoing link (described in Section 3.1), the FIB state is again only doubled overall, which we believe is a modest requirement.

Forwarding: In the optimized version of FCP, routers add a label corresponding to a list of failed links to packet headers, and perform forwarding based on the label. Appending and forwarding based on labels is addressed by Multi-Protocol Label Switching (MPLS) [12]. FCP also needs to invoke recomputation for new failures encountered, by invoking special processing via the slow path.

Applicability of intradomain routing controls: Since the notion of having a link-state graph is retained, the key semantics of intradomain routing maps remain unaffected. FCP continues to provide cost-based shortest-path routing in the absence of link failures. Hence, assignment of addresses and access controls, traffic engineering, and other aspects of configuration/maintenance remain unchanged. Specifically for traffic engineering, long-term planning changes to the link costs can be introduced by the central coordinator. For short-term, reactive cost changes introduced by the routers themselves, there would be a short delay since the updates are not installed instantaneously, but have to go through the coordinator before the TE link-cost changes become active. Since the link-cost changes go through the coordinator, it can be ratified before it is incorporated into the network map in order to preserve the path isolation property.

7. EXTENSIONS TO FCP

We described FCP as a link-state routing protocol, and hence is directly applicable to intradomain networks. Here, we present extensions to FCP to broaden the scope of applicability. Specifically, we turn to how FCP can be used to improve interdomain routing, both in terms of iBGP and eBGP routing stability.

7.1 Improving iBGP stability under link failures

Hot potato routing is commonly used by ISPs to select the closest exit point amongst multiple equally-good interdomain routes. Failure of links within the network, failures of next-hop links going out of the network from the border routers, as well as small perturbations in intradomain costs can lead to *hot-potato disruptions*, where large amounts of traffic oscillate between egress points. Such disruptions can lead to routing loops, router overload, and externally visible BGP routing changes [33, 34]. Hence a scheme that can prevent such instability during change of egress points is desired [32, 33].

We present a simple modification to FCP to allow it to operate over iBGP routes within a single domain for the case of failure of links. We augment the link-state network map maintained by internal routers to treat an egress route to a particular prefix as a *virtual link* directly connected to the destination prefix. FCP is agnostic to the the notion of virtual links, and it treats virtual and actual links identically; we use the term virtual link only for convenience. When either a normal link or a virtual link fails, a router can use FCP to forward the packet to an alternate egress connected to the same next-hop AS. This ensures that routing to external routes remains consistent even if failures are not immediately propagated.

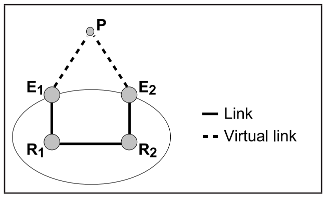


Figure 11: Mitigating iBGP disruptions using FCP.

A simple illustrative example is shown in Figure 11. The network map consists of actual links $E_1 - R_1$, $E_2 - R_2$ and $R_1 - R_2$, and virtual links $E_1 - P$ and $E_2 - P$ for prefix P. Initially, let both routers R_1 and R_2 use egress E_1 to reach the destination prefix P. When the link (R_1, E_1) (or the BGP next hop from E_1 towards P) fails, R_1 appends the (R_1, E_1) (or the virtual link (E_1, P)) to packet headers causing R_2 to forward the packets via E_2. Traditional iBGP/OSPF routing would undergo a routing loop between R_1 and R_2 lasting until R_2's scan process, i.e. visiting the BGP routing decision for each prefix, completes.

7.2 Interdomain policy routing

Interdomain routing today suffers from long outages arising from a slow convergence process that occurs after certain routing events [21]. In this section, we discuss how we can leverage FCP to avoid failures during the convergence process of BGP. In our proposal, we only consider changes on the data plane; we do not modify BGP's route announcement and propagation protocol. Next, we discuss two of the key challenges faced by our proposal: (a) how are the network maps defined and distributed? (b) how are policies respected when FCP is used?

7.2.1 FCP network map

Unlike the case of intradomain routing, there is no natural centralized authority to act as a coordinator for distributing AS-level network maps. Hence, we assume that nodes work with inconsistent maps and use SR-FCP (Section 2.2).

All routers in the network run BGP protocol for exchanging routes as they do today. Each router defines the FCP map using the latest set of BGP updates it has received from all its neighbors.

7.2.2 Using SR-FCP with policy routing

Naively implementing SR-FCP would have adverse policy implications. This is because, by using AS-level source routes, an AS can force downstream ASes to forward traffic at the expense of violating their own policies. We next present a solution to this challenge.

The main idea behind our solution is to treat any policy violation as a link failure. We assume that ASes only implement policies that are a function of the neighbor from whom they received the advertisement and local policy considerations (as opposed to, for example, policies dependent on the presence of an non-neighbor AS in the AS-path). Almost all of today's BGP policies fall into this category [35].

Consider a packet p routed from source AS S to destination AS D. When a router in the source AS S starts routing a packet, it adds the AS-level path (source route) in the packet header, just as proposed by SR-FCP. Let a router R belonging to an intermediate AS I receives the packet p with an AS-level source route $ASR(p)$. Let $ASR(R, D)$ represent the AS-level route computed by R to the destination AS D (based on its BGP route selection criteria). Finally, let $NextHop(Route)$ represent the AS-level next-hop for a route. The following cases are possible:

1. $N{=}NextHop(ASR(p)){=}NextHop(ASR(R, D))$ and next-hop to N is alive. In this case, R simply forwards the packet to N.

2. $N{=}NextHop(ASR(p)){=}NextHop(ASR(R, D))$ and that next-hop to N is dead. In this case, R invokes SR-FCP by adding the AS-path $I - N$ (recall that I is the intermediate AS that R belongs to) to the failure header. R then forwards the packet along the best AS-path that it knows which does not have any failures.

3. $NextHop(ASR(p)){\neq}NextHop(ASR(R, D))$. We discuss this case next.

If the AS decides that the source route present in the packet is *not compatible* with its choice of routes, it does the following operations for preventing transient loops. First, it adds $NextHop(ASR(p)$ to the failure header of the packet. Second, it uses the best route it has to the destination that does not have any failed links, and adds that source route to the packet. We leave the degree to which an AS allows routes that are not the most-preferred to be used in the source route as a meta-policy decision of that AS. This meta-policy represents the tradeoff between the degree to which the AS allows FCP to recover from failures and the extent to which policies are obeyed. For instance, an AS can decide that it allows only the top two preferred routes.

7.2.3 Discussion

Since BGP propagates only routes that can be potentially used downstream, routers will not obtain the entire link-state of the network. However, after BGP converges, all the nodes should have consistent route selection information, *i.e.,* all nodes will pick the same AS-level path for each destination prefix. In other words, at all nodes, the AS-level source route in the packet header will be identical to its most preferred route to the destination (Case (1) above).

During the convergence process, routing using the above modified SR-FCP protocol ensures that packets will not enter into transient loops. This follows from the fact that packets are routed using SR-FCP, hence at every stage the packet either makes progress based on the source route or that a link is added to the list of failed links in the packet header. Eventually, the packet will reach the destination or will discover that there are no more paths (based on links in the failure header), and will be dropped. Also, as we mentioned above, the extent to which FCP will help route around failures depends on the flexibility of the policy at the ASes to use source routes that are not the most preferred.

A practical challenge that our scheme faces is deployability, as it requires one to insert the AS source route and the list of failed links in the IP header. While one can use IP options for this purpose, finding a comprehensive solution is a subject of future work.

8. RELATED WORK

Prior work to address the problem of routing convergence in the literature at the protocol level can be roughly classified into three categories: (a) designing loop-free convergence protocols, (b) reducing the convergence period of protocols, and (c) using precomputed backup paths to route around failures. We don't discuss attempts at addressing such issues at the higher layers, for instance using overlays to get around underlying routing problems.

The idea of carrying information to route a packet in the header of the packet itself is inspired by Stoica's work on dynamic packet state [30]. FCP is similar in spirit to LOLS [26] used for routing in wireless adhoc networks. However, FCP separates network map information and transient failures by not introducing transient failures into the map, and hence can provide better routing guarantees.

Loop-free convergence: Several approaches ensure that convergence takes place in a way that it obeys certain correctness constraints. For example, link-state vector routing [5] advertises a subset of links, and uses a termination-detection algorithm to break loops. Diffusing computations [16] achieves theoretical loop-free routing convergence using a distance-vector paradigm. However, as the authors of that paper note, the performance after node failures and network partitions is a concern because all network nodes have to be involved in the same diffusing computation.

Reordering LSAs during propagation has been proposed to ensure that transient loops are avoided, for the specific cases of protected and planned link failures and cost changes [15] alone. Not-via addresses [8] uses a mechanism very similar to the precomputation optimization presented in Section 3.1. Consider a router R that performs the following computation: For dealing with node failures, by iterating over each other router R', R precomputes backup paths to all destinations assuming that R' is down. For dealing with link failures, it performs a similar precomputation by iterating over neighboring links. However, the draft states that they do not aim to handle multiple simultaneous link/router failures. Nearside tunneling [7] dynamically constructs tunnels to the closest router adjacent to the failure, and forward traffic via the tunnel during the convergence process. A simpler scheme is to simply forward via a loop-free alternate path in the presence of failure, or to forward to a U-turn alternate when no loop-free alternate exists [4]. The location of link failures may be inferred from the interface on which the packet arrives [27]. While these approaches improve properties of the convergence process, they still require routing updates at the control plane, and are hence subject to the control overhead versus availability tradeoff we discussed in our results.

Reducing convergence times: Some efforts have addressed failure recovery directly at the level of the routing protocol. For instance, Alaettinoglu *et al.* [3] propose to modify IGP implementations to reduce convergence time to a few milliseconds even when links fail, by modifying timers and improving run-time of the route computation algorithm. However, reducing timers can increase the control overhead and worsen network stability, as shown by our experimental results. In general, there has been substantial debate over what parameters to use, and it is not clear that there is a single correct choice of these timers or if they can be eliminated completely.

Such protocol 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 because often multiple weights need to be adjusted simultaneously.

Using precomputed backup-paths: Several works have proposed using precomputed backup routes when primary paths in the network fail. Examples include IP restoration [18], MPLS Fast-

Reroute [28], and others [4,6,11,19]. A short evaluation of the fast reroute techniques is presented in [14]. More recently, R-BGP [20] proposes using a simple precomputation-based backup method for fast-failover during BGP convergence process that has some provable guarantees such as loop-prevention and valley-free routing.

Backup routes are practical only when there are small numbers of simultaneous failures; to achieve the guaranteed reachability property of FCP with multiple failures, several backup paths would be needed. In fact, from our experiments, we see that even with low failures rates, multiple failures can simultaneously occur in real networks. In contrast to precomputed backup paths, FCP not only provides correctness guarantees in the face of multiple link failures, but does so by requiring much lesser state at the routers than backup path strategies typically do.

9. CONCLUSION

We proposed Failure-Carrying Packets (FCP), a new routing technique which eliminates the convergence period endured by traditional routing protocols. The basic idea of FCP is simple: once all routers have a loosely-synchronized, consistent view of the network, it is enough for a router to know the list of failed links to correctly compute the path to a destination.

Though we primarily present FCP to introduce a new routing paradigm that is qualitatively different from previous approaches, we present optimizations that make FCP practical. Using real-world ISP topologies and failure traces, we show that both the computational as well as packet overhead incurred by FCP is small. We also present comparisons with both OSPF as well as a commercially-used backup path technique. In the former case, we show that unlike OSPF, FCP can provide both low loss-rate and low control overhead. In the latter case, we shows that FCP provides better routing guarantees under failures despite maintaining lesser state at the routers. Though the basic model of FCP as a link-state routing paradigm is directly applicable only to intradomain networks, we discuss how FCP can be applied to interdomain policy routing as well. Studying the applicability of FCP in different routing networks (such as interdomain routing, wireless networks, sensor networks) more deeply is a topic of future work.

Acknowledgements

We would like to thank our colleages at Berkeley for their feedback and discussions on earlier drafts of the paper. We also thank our shepherd, Dina Papagiannaki, and the anonymous reviewers of SIGCOMM 2007 for valuable suggestions that helped improve the quality of presentation. This research was sponsored by NSF under grant numbers 0205519 and 0225660.

10. REFERENCES

[1] Abilene observatory data collections. 2006. `http://abilene.internet2.edu/observatory/`.

[2] Anonymized OC48 traces, CAIDA. 2006. `http://data.caida.org/`.

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

[4] A. Atlas. U-turn Alternates for IP/LDP Fast-Reroute. Internet Draft draft-atlas-ip-local-protect-uturn-03.txt, February 2006.

[5] J. Behrens and J. J. Garcia-Luna-Aceves. Distributed, scalable routing based on link-state vectors. In *Proc. ACM SIGCOMM*, 1994.

[6] S. Bryant, C. Filsls, S. Previdi, and M. Shand. IP Fast Reroute using Tunnels. Internet draft draft-bryant-ipfrr-tunnels-01.txt, Oct 2004.

[7] S. Bryant and M. Shand. A Framework for Loop-free Convergence. Internet Draft draft-bryant-shand-lf-conv-frmwk-03, October 2006.

[8] S. Bryant, M. Shand, and S. Previdi. IP Fast Reroute Using Not-via Addresses. Internet Draft draft-bryant-shand-ipfrr-notvia-addresses-03, 2006.

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

[10] G. Choudhury. Prioritized Treatment of Specific OSPF Version 2 Packets and Congestion Avoidance. RFC 4222, October 2005.

[11] G. Choudhury, A. Atlas, R. Torvi, C. Martin, B. Imhoff, and D. Fedyk. Basic Specification for IP Fast-Reroute: Loop-free Alternates. IETF draft, 2005.

[12] B. Davie and Y. Rekhter. MPLS: Technology and Applications. In *Morgan Kaufmann*, 2000.

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

[14] P. Francois and O. Bonaventure. An evaluation of IP-based Fast Reroute Techniques. In *Proc. CoNEXT*, 2005.

[15] P. Francois and O. Bonaventure. Avoiding transient loops during IGP convergence in IP networks. In *Proc. INFOCOM*, 2005.

[16] J. J. Garcia-Luna-Aceves. Loop-Free Routing Using Diffusing Computations. *IEEE/ACM Transactions on Networking*, 1993.

[17] Y. Hu, A. Perrig, and M. Sirbu. SPV: Secure Path Vector Routing for Securing BGP. In *Proc. ACM SIGCOMM*, 2004.

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

[19] Juniper-Networks. Configure alternate backup paths using fate-sharing. 2005. `http://www.juniper.net/techpubs/software/junos/junos53/swconfig53-mpls-apps/html/mpls-signaled-config37.html`.

[20] N. Kushman, S. Kandula, D. Katabi, and B. Maggs. R-BGP: Staying Connected in a Connected World. In *Proc. NSDI*, 2007.

[21] Z. Mao, R. Govindan, G. Varghese, and R. Katz. Route Flap Damping Exacerbates Internet Routing Convergence. In *Proc. SIGCOMM*, 2002.

[22] J. T. Moy. OSPF Database Overflow. RFC 1765, March 1995.

[23] J. T. Moy. OSPF Version 2. RFC 2328, April 1998.

[24] J. T. Moy. OSPF complete implementation. In *Addison-Wesley, New York*, 2001.

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

[26] S. Nelakuditi, S. Lee, Y. Yu, J. Wang, Z. Zhong, G.-H. Lu, and Z.-L. Zhang. Blacklist-Aided Forwarding in Static Multihop Wireless Networks. In *Proc. of SECON*, 2005.

[27] S. Nelakuditi, S. Lee, Y. Yu, Z.-L. Zhang, and C.-N. Chuah. Fast Local Rerouting for Handling Transient Link Failures. *IEEE/ACM Transactions on Networking*, 2007.

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

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

[30] I. Stoica. *Stateless Core: A Scalable Approach for Quality of Service in the Internet*. PhD thesis, Carnegie Mellon University, Dec. 2000.

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

[32] R. Teixeira, N. Duffield, J. Rexford, and M. Roughan. Traffic Matrix Reloaded: Impact of Routing Changes. In *Proc. of PAM*, 2005.

[33] R. Teixeira, T. Griffin, A. Shaikh, and G. Voelker. Network Sensitivity to Hot-Potato Disruptions. In *Proc. of SIGCOMM*, 2004.

[34] R. Teixeira, A. Shaikh, T. Griffin, and J. Rexford. Dynamics of Hot-Potato Routing in IP Networks. In *Proc. of ACM SIGMETRICS*, 2004.

[35] F. Wang and L. Gao. Inferring and Characterizing Internet Routing Policies. In *Proc. IMC*, 2003.