Loop-Free Convergence With Unordered Updates

Glenn Robertson, Nirupam Roy, Phani Krishna Penumarthi, Srihari Nelakuditi, and Jason M. O'Kane

***Abstract*—This paper studies the feasibility of minimizing convergence delay and forwarding disruption without carrying any additional bits in the IP header, to provide high availability despite link failures in traditional IP networks. Previously proposed mechanisms achieve two of these three objectives by trading off the other objective. For instance, the ordered forwarding information base updates approach may prolong the convergence delay, whereas the SafeGuard scheme requires carrying the path cost in the IP header. As a better alternative, we propose a scheme called fast convergence with fast reroute (FCFR), which combines the features of IP fast rerouting and interface-specific forwarding. We show that FCFR can achieve minimal convergence delay, while ensuring loop-free delivery during convergence, after a single non-partitioning failure in an IP network, without altering the IP header format, making it amenable for immediate deployment.**

***Index Terms*—Network failures, resilient routing, convergence delay, fast reroute, routing loops, order of updates.**

I. Introduction

STUDIES on the occurrence of failures in a backbone network have shown that failures of links and routers are common even in a well managed network [1]. On the other hand, an increasing number of users and services are relying on the Internet and expecting it to be always available. In order to ensure high availability in spite of failures, a routing scheme needs to quickly restore forwarding to affected destinations. Traditional routing schemes such as OSPF trigger link state advertisements in response to a change in topology, and cause network-wide recomputation of routing tables. Such a global rerouting incurs some delay before traffic forwarding can resume on alternate paths. During this *convergence delay*, routers may have inconsistent views of the network, resulting in forwarding loops and dropped packets [2].

Several IP fast reroute schemes such as NotVia [3], FIFR [4], MRC [5], LFA [6] and its variants like RLFA [7] have been proposed in the past to initiate local rerouting as soon as a failure is detected. In addition to the benefit of prompt forwarding resumption, local rerouting can also prevent unnecessary routing updates when the network outage is temporary. The downside of local rerouting is that the packets take longer detours to reach their destinations. In order to regain optimal routing if the failure lasts longer than a preset threshold duration, routing updates are triggered and a re-convergence of the network takes place. Thus, fast reroute techniques do not obviate the need for the eventual convergence process.

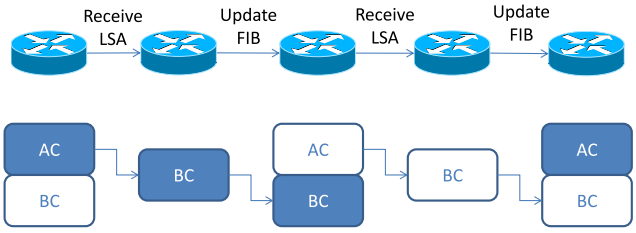


Fig. 1. Illustration of the actions of a router after a network event.

In order to prevent routing loops during this transitional period, other authors have proposed schemes such as ordered FIB updates [8]. This approach creates a stable transition from the outdated network topology to an updated view of the network. However, this process extends the convergence period of the network by waiting for acknowledgments that all routers have updated their information base in the proper order before proceeding. This process takes longer to converge than conventional OSPF and prolongs the time period before the network is prepared to adapt to another outage or change.

SafeGuard [9] was proposed to address the above concerns and achieve three interconnected objectives: 1) loop-free forwarding; 2) minimal forwarding disruption; and 3) minimal convergence delay. At no time can a forwarding loop happen with SafeGuard in the case of a single failure. SafeGuard also reduces the period when packets are dropped due to the lack of valid routes. Moreover, SafeGuard minimizes the convergence delay, i.e., packets are forwarded along optimal paths and the network is ready to absorb another change as soon as possible. The drawback of SafeGuard, however, is that it requires each packet to carry the cost of the remaining path to the destination, needing multiple bytes in the header. Our aim is to maintain these benefits, but without changing the IP header format, eliminating practical hurdles for deployment.

We propose a scheme called *fast convergence with fast reroute* (FCFR), which uses an existing fast reroute technique such as NotVia to create alternate routing during the convergence process. Under FCFR, each router maintains two forwarding tables (Fig. 1). The *before change* (bc) table is

Manuscript received January 23, 2016; revised September 14, 2016 and February 18, 2017; accepted February 21, 2017. Date of publication March 2, 2017; date of current version June 9, 2017. This work was supported in part by the National Science Foundation (NSF) under grants CNS-0448272 and CNS-0551650. The associate editor coordinating the review of this paper and approving it for publication was S. Schmid. *(Corresponding author: S. Nelakuditi.)*

G. Robertson is with the Department of Electrical Engineering and Computer Science, United States Military Academy, West Point, NY 10996 USA.

N. Roy is with the Department of Electrical and Computer Engineering, University of Illinois at Urbana–Champaign, Urbana, IL 61801 USA.

P. K. Penumarthi, S. Nelakuditi, and J. M. O'Kane are with the Department of Computer Science and Engineering, University of South Carolina, Columbia, SC 29208 USA (e-mail: srihari@cse.sc.edu).

Digital Object Identifier 10.1109/TNSM.2017.2675921

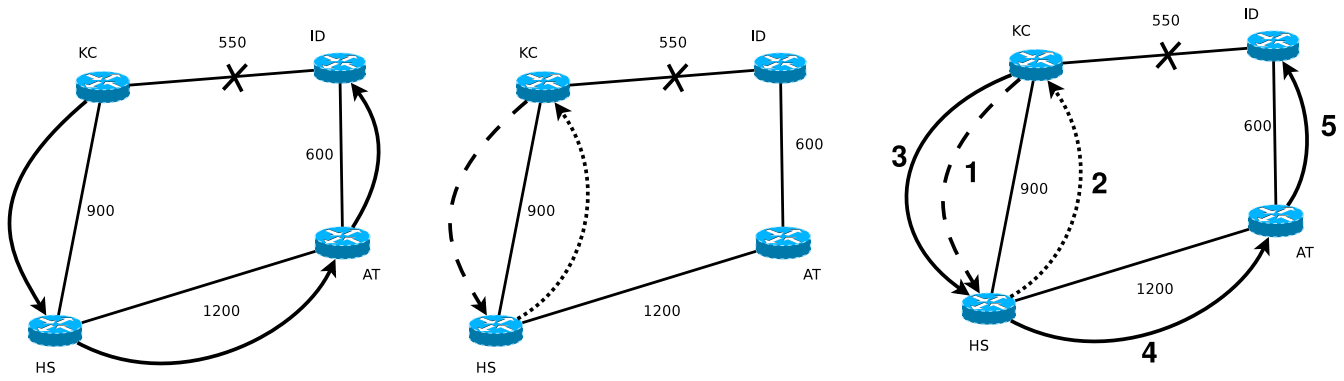


Fig. 2. (a) An example using NotVia on a part of the Abilene topology. (b) An illustration of transient loops caused by adjacent routers with inconsistent routing tables. (c) Using FCFR, KC forwards a packet using its ac routing table.

based on the outdated topology, and the *after convergence* (ac) table is computed once the router has the updated topology. Routers that have absorbed the new topology begin to forward packets with the ac table. But, if a packet reaches a router with only the bc table, it will use bc table for forwarding. Once a packet has been diverted using a bc forwarding table, the packet will continue to be forwarded along a path from the bc topology, with the aid of fast rerouting when necessary. This guarantees that packets which originate at an updated router will always get forwarded, either along an ac path, or an ac path followed by a bc path. Packets originating at not-yet-updated routers follow the bc path all the way to the destination.

To ensure that a packet does not alternate between bc and ac path segments and thus avoid forwarding loops during convergence, FCFR needs to be aware of the packet's forwarding mode. To that end, FCFR can either indicate the forwarding mode with a bit in the packet's header (called FCFR_1) or infer the forwarding mode based on the packet's incoming interface (called FCFR_0). We prove that FCFR_1 with an additional bit guarantees loop-free convergence regardless of the order of updates after a single non-partitioning failure in an IP network. Using FCFR_1 as a stepping stone to develop FCFR_0, we prove that FCFR_0 with interface-specific forwarding can guarantee the same for network topologies with symmetric link weights, without carrying any additional information in the packet.

The rest of this paper is organized as follows. Section II illustrates FCFR_1 that uses one bit in each packet to mark its current forwarding mode. Section III discusses how FCFR_0 can achieve the same outcome, without needing an additional bit in the packet, by utilizing interface specific forwarding. Section IV presents the results of our simulation. The limitations of FCFR are discussed in Section V. Section VI discusses the related work that motivated the design of FCFR. Finally, we conclude the paper in Section VII.

II. FCFR_1: Fast Convergence With One Bit Overhead

In this section, we illustrate a simple example, describe the intuition behind our approach, gradually develop the FCFR_1 scheme that requires an additional bit in the header for forwarding, and show how it provides fast loop-free convergence.

A. Motivation

Fig. 2a shows an example using a part of the Abilene topology [10]. Suppose the link between Kansas City (KC) and Indianapolis (ID) fails. The routers adjacent to the failure would initiate an IP Fast Reroute scheme such as NotVia to temporarily reroute traffic. In order to reroute the traffic around the failed link, the NotVia mechanism would encapsulate each packet with the destination of ID, using a special address which indicates that the link KC−ID has failed. This special address is precomputed and known to each router along the alternate route. Therefore, the temporary reroute scheme would send packets along the path KC−HS−AT−ID.

After detecting the failure, the KC router would also send out an LSA[1] to advertise the failure throughout the network. Each router in the network would recompute its forwarding table according to the updated topology. Once KC completed its computation, it would update its Forwarding Information Base (FIB) as soon as possible and begin to forward using the updated tables. Thenceforth, packets at KC bound for ID would be routed toward Houston (HS) as the next hop.

During the transient period when all routers have not completed updating their FIB, there is an inconsistent view of the network topology. The routers adjacent to the failure (KC and ID) may be the first to update their FIBs. However, their immediate neighbors, HS and AT, might update their FIBs a little later due to the delay in propagation and processing of LSAs across the network. Therefore, for a short time, packets at HS bound for ID would be re-routed back toward KC.

Fig. 2b shows an illustration of what are termed transient loops, or micro-loops, which occur during the convergence period. These loops are caused by adjacent routers that have differing views of the network topology. In this example, KC has begun to forward based on the updated forwarding table, indicated by the dashed line. Note that rerouting with NotVia is not triggered at KC since the new next hop is not ID

[1]Though we are discussing an LSA corresponding to a link going down, the same arguments about loop-free convergence apply to LSAs about a link coming up or an increase or decrease in its weight.

because KC has already updated its forwarding table. HS is re-computing its topology and is still using the old forwarding table, as indicated by the dotted line. This conflicting view of the network is the cause of these temporary loops.

Transient loops can cause dropped packets due to TTL expiration, and additional load on the affected links, which competes with legitimate traffic for bandwidth on those links. These packets are not delivered due to forwarding inconsistencies in the network, and can cause the loss of critical data, such as VoIP calls and real-time collaboration sessions.

The previously proposed approach using ordered updates [8] can eliminate these transient loops. With ordered updates, KC would update its forwarding table only after HS has updated its table. Therefore, until KC updates its table, packets to ID that arrive at KC get rerouted using NotVia address of ID. This NotVia path would be the same in both new and old forwarding tables since the link KC−ID is excluded from the computation. Once KC updates its table, packets get forwarded along the new path KC−HS−AT−ID without any loops.

The goal of FCFR is to minimize convergence delay and minimize network disruption during the convergence process. The drawback of ordered updates is that loops are prevented at the expense of increased convergence delay. This may not be a serious concern since packets are delivered use fast rerouting during this period. However, it is desirable to reduce the convergence delay for two reasons. First, packets take longer detours during convergence and it is preferable to restore the optimal routing as soon as possible. Second, and perhaps more importantly, once the network converges after a failure, it is ready to handle another failure. Thus overlapping multiple failures can be treated as sequential single failures which are easier to handle. These are the reasons that motivated the design of SafeGuard [9]. Our aim is to achieve the same benefits as SafeGuard, without requiring 4 bytes of information in the header of each packet. In the following, we present the intuition behind FCFR_1 that achieves both fast convergence and fast rerouting with only one additional bit of overhead.

B. Intuition

Suppose a link's state changes at time t_0. Let us refer to the forwarding table as being in the *before change* (`bc`) era if it was computed before t_0 and therefore, does not reflect the change. Similarly, a recomputed forwarding table that accounts for the change is said to be in the *after convergence* (`ac`) era. Based on this naming convention, a network is said to be converged at time t_c, when all the routers are in the `ac` era. During the time between t_0 and t_c, which is the convergence delay, some routers' forwarding tables are in the `bc` era while others are in the `ac` era. Due to this inconsistency, packets may get caught in transient forwarding loops.

Clearly, when all the routers forward using the `ac` era table, packets will not loop. Similarly, no loops can exist if all the routers use the `bc` era forwarding tables. Using the `bc` table, packets could arrive at a router adjacent to the failure and potentially get dropped because the next-hop is not reachable. However, fast reroute mechanisms such as NotVia are employed to handle this scenario by having adjacent routers perform local rerouting along an alternate loop-free path. In other words, with `ac` era forwarding by all routers, no packet encounters a forwarding loop or routing failure. The same scenario occurs by using `bc` era forwarding at all routers in combination with NotVia. The only difference is that `ac` era forwarding is optimal. Therefore, we would like to have all routers start using `ac` tables as early as possible.

Now, imagine a hypothetical scheme where all routers use the `bc` table for forwarding along an alternate route using NotVia during the time between t_0 and t_c. Then, at time t_c, all routers would switch to using the `ac` table for forwarding. Such a hypothetical scheme would not have any forwarding loops or forwarding failures. Obviously, this hypothetical scheme is not feasible. First of all, the time to recompute $(t_c - t_0)$ is not known in advance. Second, and perhaps more importantly, it is not feasible for all the routers to update their FIBs simultaneously at time t_c. Therefore, we need a practical scheme that behaves approximately like this hypothetical scheme.

Our approach is based on the following intuition. Consider the path traversed by a packet. Suppose we segment the packet's path into `ac` and `bc` segments (with one or more hops) such that all forwarding within a segment belong to the same era. The possible combinations of segments in a path could be:

1) `bc`
2) `ac`
3) `ac−bc`
4) `bc−ac`
5) `bc−ac−bc−···`
6) `ac−bc−ac−···`

A forwarding loop is only possible when a path is allowed to alternate between `bc` and `ac` segments as in cases of (5) and (6). Therefore, by constraining a packet to traverse between `bc` and `ac` segments only once at the most, we can ensure that the packet does not loop. Our approach is based on this intuition — it allows only the first 3 cases to happen. In other words, a packet gets forwarded using `ac` tables until it encounters a router which has only the `bc` table. Thereafter, it gets forwarded by `bc` tables until it reaches the destination.

C. Scheme

We now describe how FCFR_1 achieves fast loop-free convergence using NotVia to provide fast rerouting. Under NotVia, when a link l fails, the adjacent router encapsulates the original datagram inside another packet with the destination address set to the next-hop's not-via address. Because of the meaning of the not-via address, it gets routed consistently by all routers along an alternate path that does not include l. Without notifying others about the failure, NotVia can guarantee delivery to all destinations in the case of a single failure. But routing would be suboptimal and another concurrent failure could cause loops. Therefore, a link state update is triggered even while performing local rerouting with NotVia.

The basic idea behind FCFR_1 is to use one bit, which represents the `era`, in the packet to convey how it should be forwarded. This bit represents either `bc` or `ac`. As illustrated

in Fig. 1, during the time between receiving LSA and updating FIB, a router uses only the bc table. Between the time of FIB update and network convergence, a router may reference either the bc or ac table based on packet's era field. When the network is converged to a stable state, each router has both bc and ac tables but only the ac table is used for forwarding.

When a packet has the era bit set to bc, all routers consistently forward according to their bc tables as per the fast reroute mechanism. If the packet has the era bit set to ac, routers forward it according to the ac table if their FIB update is complete. If a router is still in the bc era, it resets the era bit in the packet to bc. From then on, the packet only gets forwarded along the bc path. The packet era is initialized to bc or ac based on the state of the originating router.

The mapping of bc/ac to 0/1 is not static. It changes after every link up/down event. However, it is consistently interpreted by different routers provided the following two conditions are satisfied. 1) A router does not install a new FIB before its neighbor receives the corresponding LSA. This is a reasonable assumption considering that propagating the LSA over a link takes much less time than the time needed to recompute new forwarding table and update the FIB. 2) At most one failure event is being propagated in the network, which is observed to be the most common scenario [1]. Under these conditions, we can prove that if the convergence delay for OSPF is $(t_c - t_0)$, then FCFR$_1$ converges within $(t_c - t_0)$.

The actions taken by a router in response to different events under FCFR$_1$ are specified in Algorithm 1 (see Table I for the notation). These operations can be summarized as follows.

- Each packet p has a 1-bit field in the header called era.
- Each router R maintains its current era.
- A packet p's era is set to R's era if p originates at R.
- R forwards p using the table for p.era.
- If R does not have a table for p.era, then it toggles p.era and forwards it according to that table.
- A packet in the bc era stays in that era until it reaches its destination using fast reroute.
- A packet in the ac era switches to the bc era if it encounters a router in the bc era enroute to the destination.

Now let us revisit the Abilene example shown in Fig. 2 and consider what would happen under FCFR$_1$ for the same failure. The routers at KC and ID would be the first to recompute the updated topology and begin forwarding using the updated ac forwarding table. However, the routers at HS and AT incur a notification delay due to the LSA propagation process before they can compute the new table. Therefore, during the transition period, we have neighboring routers in the network with inconsistent forwarding tables. Fig. 2c shows how FCFR$_1$ prevents loops from forming in this situation.

First, the router at KC begins to forward packets using its updated (ac) forwarding table. KC sends each packet with the era bit set to ac, as indicated by the dashed line. The router at HS would receive the packets with ac era and recognize that it does not have the updated forwarding table. Therefore, HS would reset the era bit to bc in these packets and forward them using the outdated (bc) forwarding table back to KC. This return path is indicated by the original dotted line. Once a packet arrives back at KC with era set to bc, it must look up its previous (bc) forwarding table. Since the link from KC to ID is down, the router uses NotVia encapsulation to route the packet via the path KC−HS−AT−ID, as before. Once HS computes its ac table, packets follow the new optimal path. Thus, as soon as the routers along the shortest path update their FIBs, FCFR$_1$ resumes optimal forwarding.

TABLE I
NOTATION

p.era	era for forwarding of packet p
$\overline{p.\texttt{era}}$	complement of p.era
p.dst	destination of packet p
R.era	current era of the router R
$\mathcal{F}_R$[era]	forwarding table at R corresponding to era
$\mathcal{F}_R$[era](dst)	next-hop to dst from R as per era
$\mathcal{N}(R)$	Set of neighboring routers of R
$\mathcal{ISF}_R$[prv](dst)	next-hop to dst at R for packets from prv

Algorithm 1 Operations Under FCFR$_1$

1: **if** Router R receives a packet p
2: **if** $\mathcal{F}_R[p.\texttt{era}]$ does not exist, **then** $p.\texttt{era} \leftarrow R.\texttt{era}$
3: forward p to $\mathcal{F}_R[p.\texttt{era}](p.\texttt{dst})$
4:
5: **if** Router R receives a new LSA
6: purge $\mathcal{F}_R[\overline{R.\texttt{era}}]$
7:
8: **if** Router R has recomputed new FIB
9: $R.\texttt{era} \leftarrow \overline{R.\texttt{era}}$
10: $\mathcal{F}_R[R.\texttt{era}] \leftarrow$ new FIB
11:
12: **if** Packet p originates at router R
13: $p.\texttt{era} \leftarrow R.\texttt{era}$
14: forward p to $\mathcal{F}_R[p.\texttt{era}](p.\texttt{dst})$
15:
16: **if** Router R initializes its state
17: $R.\texttt{era} \leftarrow 0$
18: $\mathcal{F}_R[1] \leftarrow \mathcal{F}_R[0]$

Next, we discuss FCFR$_1$'s ability to handle multiple failures and interoperate with fast reroute schemes other than NotVia, and address potential concerns with its implementation.

Handling multiple failures: Thus far, we have described FCFR$_1$ assuming a single link failure[2] is being propagated in the network. However, FCFR$_1$ can handle multiple correlated link failures, for instance due to a router failure, by treating them as a single event with a single FIB update. In case of a router failure, another router R would receive multiple LSAs, one from each of the neighbors of the failed router, in quick succession. The router R can then treat them all as corresponding to a single failure event: purge the bc table, toggle the era, and update the new FIB or ac table, all only once. This requires a change in the line 5 of the Algorithm 1. Once the router R receives a new LSA and the bc table is purged, no further action is taken upon receiving the subsequent LSAs that arrive before the computation and updating

[2]Unless otherwise explicitly stated as a node failure, in this paper, by default, a failure event refers to a single link failure.

of new FIB. Further, the FIB computation has to take into account the information from all those LSAs, such that new FIB reflects the new topology. In addition, FCFR$_1$ can successfully protect against any two unrelated failures if they are spaced apart by a time interval of at least $(t_c - t_0)$.

Working with other fast reroute schemes: The above discussion of FCFR$_1$ assumes that NotVia is being employed for fast rerouting. We chose NotVia as it provides 100% coverage against single failures, besides helping us make the description of FCFR$_1$ concise and clear. Here, we argue that FCFR$_1$ also works well with other fast reroute schemes such as Loop Free Alternates (LFA) [6] and Remote LFA (RLFA) [7].

LFA essentially routes around a failure by identifying a neighbor that would not route back to it. For instance, in Fig. 3, given a link outage between ID and CH, the router at ID could forward packets destined for NY to AT. The router AT is considered a loop-free alternate because it is closer to the destination NY than the path going back through the forwarding router ID. In this case, the cost from AT-WA-NY is 1100, and the cost from AT-ID-CH-NY is 1550. While LFA has very low overhead, it does not offer full protection against some failures depending on the topology. In the above failure scenario, for destination CH, router ID can not find a loop-free alternate. RLFA is an extension of LFA that improves coverage against failures with the aid of tunneling. The core idea behind RLFA is to pick an intermediate node, between the router adjacent to the failure and the destination such that both the shortest path segments do not include the failed link. In the above example, under RLFA, ID would tunnel packets destined for CH to NY along the shortest path ID-AT-WA-NY. The router NY will then forward the original packet to the destination CH, along the shortest path NY-CH.

Note that the shortest paths from ID to NY and NY to CH do not include the ID-CH link. Consequently, a link state update indicating the failure of ID-CH does not affect the forwarding between them, as their shortest paths will be the same before and after convergence. Essentially, packets destined to CH arriving at ID in *bc* mode would get encapsulated with destination address set to NY. These packets get routed consistently by all routers along the shortest path from ID to NY, as it does not include ID-CH link. The same is true for the onward routing from NY to the destination CH. Thus, the operation of FCFR$_1$ with RLFA would be very similar to that with NotVia.

Deploying in real world: FCFR$_1$ requires that each packet carry its `era`, a 1-bit field in the header. A way to implement this in current networks is to use the reserved *flag* bit in the IPv4 header [11]. Then, while forwarding a packet, a router has to lookup the corresponding forwarding table based on this flag bit and set that bit as per Algorithm 1. Effectively, using one bit in each packet and an additional forwarding table at each router, FCFR$_1$ can guarantee loop-free fast convergence and fast rerouting, the proof of which is given below.

D. Proof of Loop-Free Convergence With FCFR$_1$

We now present a formal proof of the loop-free convergence property of the FCFR$_1$ scheme, assuming the following

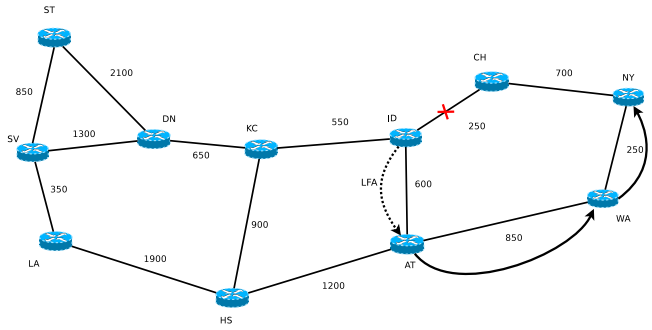


Fig. 3. Illustration of FCFR$_1$ with LFA and RLFA: When the link ID-CH is down, using LFA, router ID forwards packets bound for NY to AT, which is downstream, as the next hop. But there is no such LFA from ID to CH. So, RLFA tunnels packets bound for CH to intermediate destination NY, which in turn sends them to their original destination CH. Since the shortest paths of these two segments remain the same before and after convergence, FCFR$_1$ can coexist with RLFA to provide loop-free fast reroute and fast convergence.

constraints: 1) Only one network event propagates throughout the network until the convergence for that event is complete; 2) A router R does not install a new FIB and change its $R.\texttt{era}$ until its direct neighbors are notified of the corresponding LSA.

Invariant 1: $\forall R_n \in N(R), R.\texttt{era} = R_n.\texttt{era}$ iff $\mathcal{F}_R[R.\texttt{era}]$ and $\mathcal{F}_{R_n}[R_n.\texttt{era}]$ correspond to the same topology.

According to the FCFR$_1$ scheme, before a network event, all routers in the network have the same `era` value and operate with forwarding tables that use the same topology. Therefore this condition always holds true before a failure event. After an event, a router changes its `era` value when installing the updated forwarding table which represents the topology after the event. This table update happens only once during the convergence period given that only one network event propagates throughout the network at a time. Therefore, if two routers have same the `era` value, then their forwarding tables will represent the same topology and vice versa.

Invariant 2: $\forall R_n \in N(R), R.\texttt{era} \neq R_n.\texttt{era}$ iff exactly one of $\mathcal{F}_R[\overline{R.\texttt{era}}]$ and $\mathcal{F}_{R_n}[\overline{R_n.\texttt{era}}]$ does not exist.

According to the FCFR$_1$ scheme, a router R changes its `era` value only when it has completed a FIB update. Therefore, the value of the `era` bit for R differs from its neighbor R_n only if one of them, say R_n, has installed a new FIB and the other hasn't. Then, according to our assumptions, the neighbor R of the updated router R_n should have received the LSA and purged its forwarding table $\mathcal{F}_R[\overline{R.\texttt{era}}]$ immediately after receiving it. Therefore, $\mathcal{F}_R[\overline{R.\texttt{era}}]$ will not exist until R completes a FIB update. Conversely, if $\mathcal{F}_{R_n}[\overline{R_n.\texttt{era}}]$ does not exist, that implies R_n is computing an update and would have the same `era` value as R until its FIB update is complete.

Invariant 3: During the convergence period, the forwarding table $\mathcal{F}_R[\texttt{era}_{old}]$ exists in all routers, where $\texttt{era}_{old}$ is the `era` value in all routers before the network event.

A router R contains both the tables $\mathcal{F}_R[\texttt{era}_{old}]$ and $\mathcal{F}_R[\overline{\texttt{era}_{old}}]$ before the event (after initialization and possible convergence from previous events). After receiving an LSA for a network event, it removes the table $\mathcal{F}_R[\overline{\texttt{era}_{old}}]$. This can not happen prior to receiving the LSA, so $R.\texttt{era}$ still remains as $\texttt{era}_{old}$ after removing the old table. It changes $R.\texttt{era}$ bit

only after computing the new table. Therefore, a router always removes the table $\mathcal{F}_R[\overline{\texttt{era}_{old}}]$, and $\mathcal{F}_R[\texttt{era}_{old}]$ table exists in all routers at least until the end of the convergence period.

Invariant 4: A packet p can change its p.era value at most once during its flight.

Initially, a packet p's era value is the same as the era value of the router that originates it. According to FCFR_1 p.era changes only when the packet visits a router R where $\mathcal{F}_R[p.\texttt{era}]$ does not exist. Since an updated router has both the tables, p.era can only change its value at a non-updated router. All non-updated routers have $\texttt{era}_{old}$ as their R.era value, so p.era always gets the value $\texttt{era}_{old}$ after the change. Once p.era value becomes $\texttt{era}_{old}$, it never changes because by Invariant 3, $\mathcal{F}_R[\texttt{era}_{old}]$ exists at all routers.

In order to prove the loop freedom property of FCFR_1, it is necessary and sufficient to establish that "a packet never visits a router twice with same p.era value".

We prove this property of FCFR_1 by contradiction. Assume that a packet has visited router R twice with same p.era value through a non-empty sequence of routers S starting and ending with R. Invariants 1 and 2 prove that the interpretation of era values is consistent among all routers in the network. Based on Invariant 4, p.era could not have changed more than once in flight. If p.era is $\texttt{era}_{old}$, then at all the routers in S the packet would have been routed using the forwarding table corresponding to the old topology. If p.era is $\overline{\texttt{era}_{old}}$, then all routers in S would have installed new tables and the packet would have been routed using only the tables that correspond to the updated topology. In both cases, in a single topology (whether updated or not) the shortest path from R to the destination can not include R twice, a contradiction. Therefore, using FCFR_1 a packet will never visit a router twice with same p.era value, hence FCFR_1 is loop free.

III. FCFR_0: Fast Convergence Without Any Additional Bits

While FCFR_1 achieves the desired objectives of minimal forwarding disruption and convergence delay as mentioned in the introduction, with just one additional bit per packet and thus much less than the four bytes needed by SafeGuard, it still requires a change in the IP header and the forwarding process at a router. To eliminate this requirement and facilitate practical deployment, we explore the possibility of achieving the same without any changes to the packet format and the forwarding paradigm. In this section, we present such a scheme, FCFR_0, that behaves like FCFR_1 in networks with symmetric link weights, using interface-specific forwarding.

Interface-specific forwarding implies that a packet's next-hop is determined based on both its destination and incoming interface. Let $\mathcal{ISF}_R[\texttt{prv}]$ be the forwarding table corresponding to the incoming interface $\texttt{prv}\rightarrow R$. Then, a packet p arriving at R from the previous hop prv gets forwarded to the next-hop $\mathcal{ISF}_R[\texttt{prv}](\texttt{p.dst})$. Note that interface-specific forwarding is quite feasible with the current router architectures, as they already maintain a forwarding table at each line card of an interface for performing lookup at line speed. The only deviation is that unlike in current routers which have a copy of the same forwarding table at each interface, under FCFR_0, some of the entries in these tables could be different.

Apart from interface-specific forwarding, FCFR_0 assumes that all the links in the network are bidirectional with equal weight in both directions, which is generally true for backbone networks. With symmetric link weights, a forwarding loop happens only when a packet traverses two neighboring routers that are in different eras and each router is the other router's next-hop for the packet's destination according to their view of the network topology. Therefore, a router R in the ac era can infer that a packet's previous hop Q is in the bc era, if in the ac era the next hop from R to destination is Q. In that case, R can forward the packet according to its bc table and avoid a potential forwarding loop. When R has only the bc table, a packet is forwarded under FCFR_0 like in FCFR_1, based on its destination alone regardless of the incoming interface.

Consider the topology in Fig. 4, where each edge is labelled with its weight. The routers marked with S and D are the source and destination of the packet respectively. Let us consider the situation where the link between routers R_3 and D is broken and the network has not yet converged. The darker blue routers are ac routers and the lighter yellow routers are in the bc era. A packet destined for D and originated at the router S gets forwarded to the router R_1 according to the bc topology. R_1 uses R_2 as its next hop in both the ac and bc topology, so the packet reaches R_2 from R_1. Upon receiving the packet, R_2 uses the ac table to forward it to R_4 which in turn sends it to R_5. R_5 is a bc router, so it has only the bc forwarding table. According to that table, R_5 forwards the packet back to R_4. This time, R_4 observes that the packet arrives along the incoming interface $R_5\rightarrow R_4$, which indicates that R_5 is in the bc era. Therefore, R_4 now uses the bc table to forward the packet to R_2, which sends the packet to the router R_3, applying the same inferencing mechanism. Now, R_3 being aware of the adjacent link failure, reroutes the packet to the NotVia address of D, avoiding the failed link R_3-D. Since the path to this NotVia address of D, $R_3\rightarrow R_2\rightarrow R_4\rightarrow R_5\rightarrow$D, is the same in both ac and bc tables, the packet gets forwarded consistently without any loops to destination D. Thus, FCFR_0 achieves the same effect as FCFR_1, without an era bit in the packet, in networks with symmetric link weights.

FCFR_0 operations are formally specified in Algorithm 2. These operations are similar to those in Algorithm 1 for FCFR_1, except for the usage of interface-specific forwarding. When a packet p arrives at R from the previous hop prv, it is simply forwarded to $\mathcal{ISF}_R[\texttt{prv}](p.\texttt{dst})$. Upon computing a new FIB, it is pushed to all the interfaces as usual, except for the interfaces corresponding to the next-hops (could be multiple with ECMP) of each destination. Only those entries will be set to the next-hops corresponding to the bc table. When the router receives an LSA, these entries are reset to the usual next-hops, and thus effectively purging the bc table.

Note that only the $\mathcal{ISF}$ table needs to be in the forwarding plane, and the rest reside in the control plane. As mentioned earlier, current router architectures already maintain a forwarding table at each line card of an interface for performing lookup at line speed. Therefore, FCFR_0 can be realized by changing only the forwarding table computation

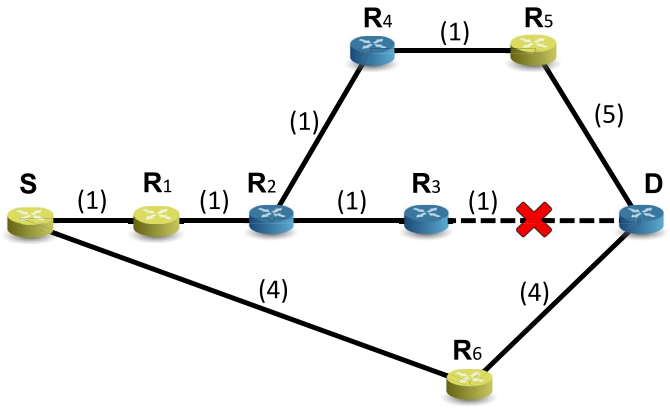


Fig. 4. Illustration of forwarding during convergence under FCFR$_0$, when the link between R_3 and D goes down. Routers R_2 and R_4 have updated their forwarding tables and entered the `ac` era, while others are in the `bc` era.

Algorithm 2 Operations Under FCFR$_0$

```
1:  if Router R receives a packet p from previous hop prv
2:      forward p to IS𝓕_R[prv](p.dst)
3:
4:  if Router R receives a new LSA
5:      for each destination dst
6:          IS𝓕_R[𝓕_R[R.era](dst)](dst) ← 𝓕_R[R.era](dst)
7:      purge 𝓕_R[R̄.era]
8:
9:  if Router R has recomputed a new FIB
10:     R.era ← R̄.era
11:     𝓕_R[R.era] ← new FIB
12:     for each neighbor prv of router R
13:         IS𝓕_R[prv] ← 𝓕_R[R.era]
14:     for each destination dst
15:         IS𝓕_R[𝓕_R[R.era](dst)](dst) ← 𝓕_R[R̄.era](dst)
16:
17: if Packet p originates at router R
18:     forward p to 𝓕_R[R.era](p.dst)
19:
20: if Router R initializes its state
21:     R.era ← 0
22:     𝓕_R[1] ← 𝓕_R[0]
```

algorithm in the control plane such that some of the entries are interface-dependent. Thus, it is possible to deploy FCFR$_0$ without altering the packet format or the forwarding plane of current networks. However, FCFR$_0$ has an obvious limitation. It is applicable only in networks with symmetric link weights. This is a fundamental constraint and we do not foresee any possibility of extending FCFR$_0$ to networks with asymmetric link weights. Note that FCFR$_1$ can however be deployed in such topologies, as it is agnostic to link weight symmetry.

A. Proof of Loop-Free Convergence With FCFR$_0$

We now prove that FCFR$_0$ is loop-free, under the following constraints: 1) Only one network event propagates throughout the network until the convergence for that event is complete;

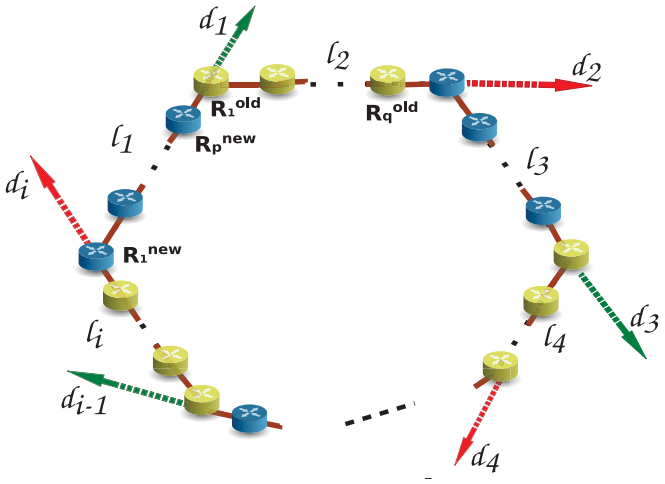


Fig. 5. A generic circular loop scenario in a symmetric link network.

2) A router R does not install a new FIB and change its R.`era` until its direct neighbors are notified of the corresponding LSA. 3) Links are bidirectional with symmetric weights. We show that under FCFR$_0$, a packet does not traverse the same link in the same direction more than once.

We first analyze the loop freedom property of a shortest path routing protocol. The only property assumed in this generic routing protocol is that each router delivers a packet to the next hop in the shortest path to the destination according to its view of network. We show that shortest path routing can avoid a certain type of routing loop. Once loop freedom is established for the shortest path routing, we extend the argument for FCFR$_0$ to prove its loop freedom in general.

Property 1: The generic shortest path routing protocol is free from circular loops in symmetric link networks.

We use the term *circular routing loop* to refer to a scenario where a packet reaches the initial router (can be any of the participating routers that is considered as the starting point of the looping packet) without traversing any edge twice in the same direction, Consider an example scenario shown in Fig. 5. In this loop, the updated and non-updated routers are arbitrarily chosen. The darker nodes depict the updated routers that use the new topology and lighter nodes are for non-updated/uninformed ones that use the old topology to forward a packet. Uninformed nodes can not be neighbors of any updated node as per our assumptions, however they would use the same topology as the non-updated nodes. Since the path used by uninformed nodes does not differ from a non-updated neighbor, we can consider uninformed nodes as non-updated ones in this proof without loss of generality.

The circular loop consists of segments of consecutive updated/non-updated nodes that follow the same path to the destination. These segments can contain one or more possible nodes. For example, the segment consisting of routers R_1^{new} through R_p^{new} follows the new topology path to the destination. The last router in this segment R_p^{new} forwards the packet to R_1^{old}, the first node of the next segment. This node uses the old topology shown by the dotted lines from router R_1^{old} to router R_q^{old}. We define the collective edge weight of the links

from R_1^{new} to R_1^{old} as l_1 and that of the links along the old path from R_1^{old} to the destination as d_1. Similarly, the values for next segment are l_2, d_2 respectively and so on. We consider i such segments in the circular loop. To show that the shortest path routing will not complete any circular loop in a symmetric link network, it will be sufficient to show that the loops are not possible with shortest path routing.

We first assume that a loop exists and then we prove the loop free property by showing a contradiction. In a circular loop as shown in Figure 5, the router R_1^{new} can forward the packet from $R_1^{new} \rightsquigarrow R_1^{old}$ and then follow the new path shown with the dotted line from router R_1^{old}. Alternately, R_1^{new} can forward the packet from $R_1^{new} \rightsquigarrow R_k^{old}$ and then follow the path shown with the dotted line from the router R_k^{old}. R_1^{new} chooses the first path as the shortest path to the destination, therefore the total weight of the first path must be less than the second one as captured in the following inequality:

$$l_1 + d_1 < l_i + d_{i-1}$$

Following the similar argument for all of the i segments, we can obtain the following inequalities.

$$\begin{aligned} l_2 + d_2 &< l_1 + d_i \\ l_3 + d_3 &< l_2 + d_1 \\ &\vdots \\ l_{i-1} + d_{i-1} &< l_{i-2} + d_{i-3} \\ l_i + d_i &< l_{i-1} + d_{i-2} \end{aligned}$$

Now adding the left and right hand sides yields a contradiction,

$$\sum_{n=1}^{i}(l_n + d_n) < \sum_{n=1}^{i}(l_n + d_n)$$

Therefore, in symmetric link networks, no generic shortest path routing protocol will cause circular loops.

Property 2: $FCFR_0$ is free from circular routing loops.

In $FCFR_0$ an updated router does not follow the old or new topology independently. The previous hop decides the choice of topology in updated routers. However, a non-updated router, which is a neighbor of any updated one, has only the old table and therefore no choice of topology. It always forwards with the old topology view. As the generic routing algorithm proves freedom from circular loops without any restriction on the choice of topology in updated nodes and $FCFR_0$ does not contradict any assumption made therein, it is evident that $FCFR_0$ is also free from a circular loop.

Next, we consider what we refer to as the *linear routing loop*, where a packet reaches the initial router by traversing all the edges of the loop exactly once in each direction, as in Figure 6. We prove that $FCFR_0$ does not cause a packet to go around a linear routing loop more than once. Towards that end, we establish some invariants under $FCFR_0$.

Invariant 5: In $FCFR_0$ an updated node can not return a packet to the router that last forwarded the packet to it.

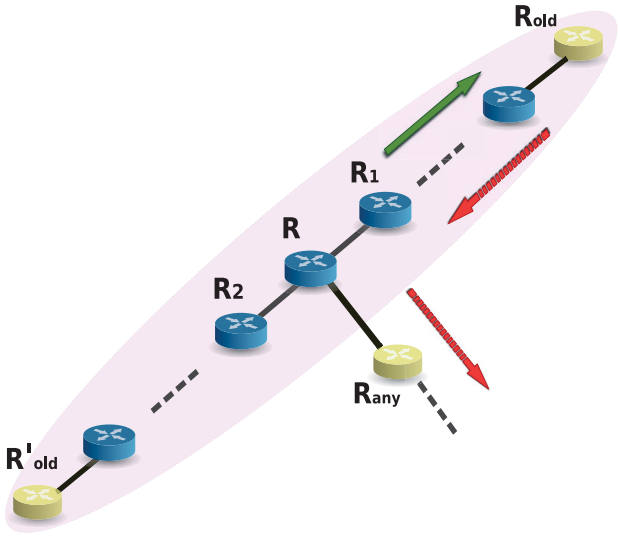


Fig. 6. A generic linear loop scenario in a symmetric link network.

Suppose an updated node R gets a packet from a neighboring node Q. For R to return the packet to Q, in the new topology Q must be its next hop, which implies that Q must be using the old topology. In this case, R will forward this packet using the old topology. When both Q and R use the same old topology for forwarding, R can not return the packet to Q.

Invariant 6: Though an uninformed node has two forwarding tables, it always uses the old topology to forward packets.

$FCFR_0$ assumes that there can not be any uninformed node as a neighbor of an updated node. Therefore an uninformed node can only get a packet from another uninformed node or a non-updated node. Moreover, at any time there can be at most a single event which an uninformed node is unaware about. Therefore, when an uninformed node A forwards a packet to another uninformed node B, it always uses its $\mathcal{F}_A[A.\texttt{era}]$ table and it in turn leads B to use its $\mathcal{F}_B[B.\texttt{era}]$, both new tables in their view. But, $\mathcal{F}_A[A.\texttt{era}]$ and $\mathcal{F}_B[B.\texttt{era}]$, in uninformed nodes A and B respectively, actually correspond to the old topology. Now, in case a non-updated node C forwards a packet to B, C uses its only table $\mathcal{F}_C[\overline{C.\texttt{era}}]$ which represents the topology before the failure according to the $FCFR_0$ actions. Therefore, from B's view the packet comes following a new topology and it uses its $\mathcal{F}_B[B.\texttt{era}]$ to forward this packet. This shows that under all circumstances an uninformed node uses the old topology to forward a packet.

Invariant 7: In $FCFR_0$, an uninformed node does not get a packet returned from the node it forwarded the packet to.

We use the similar argument as in Invariant 2 to claim that an uninformed node can only pass a packet to another uninformed node or a non-updated node. Now, according to the Invariant 2, in both of the cases the recipient node uses the old topology, which is also used by the uninformed sender node. As both the sender and receiver nodes use the same topology to forward the packet, the receiver will not return it to the sender.

Property 3: $FCFR_0$ does not cause a packet to go through a linear routing loop more than once.

We consider a generic linear loop repetition scenario in a symmetric link network as shown in Fig. 6 and argue that the scenario is not possible with $FCFR_0$. According to the Invariant 5, such a loop must have more than two nodes. Moreover, the nodes at the extreme ends of the loop must return the packet to the router that last forwarded it. Therefore, these two nodes must be non-updated as indicated by Invariant 5. The intermediate nodes have to forward a packet to two different neighbors and it is possible if the nodes contain two tables for old and new topologies. In $FCFR_0$, a router can have two tables if it is an uninformed or updated node. Note that an uninformed node can not be a neighbor of an updated node. So, the intermediate nodes of the repeatable linear loop must be either all uninformed or all updated. According to Invariant 7, the node adjacent to the end nodes must be updated, therefore all intermediate nodes are updated. The two non-updated nodes at the extreme ends of the loop forward a packet with old topology paths in opposite directions in the loop. To make this possible, there must be an old topology path from the router R_{old} to the destination that deviates from the loop at an intermediate node, R, as shown in the figure.

The linear loop as shown in Fig. 6 must involve forwarding a packet from R'_{old} to R_{old} and back twice to be an incident of loop repetition. Lets consider a packet being forwarded from R_1 to R. If $R \rightarrow R_1$ exists in the new shortest path tree, then R uses the old path and forwards it to R_{any} and thus breaks the loop. Otherwise, the packet will follow the new shortest path from R to R_2, and then through the consecutive updated nodes to R'_{old}. Now, suppose packet comes back to the router R from R_2. Since we have considered $R \rightarrow R_2$ as being part of the new shortest path, when the packet goes from R to R'_{old}, $R_2 \rightarrow R$ can not be part of the new path. So, R forwards the packet to R_{any} and thus breaks the loop again. Thus, in networks with symmetric links, forwarding with $FCFR_0$ is loop-free.

IV. Performance Evaluation

To evaluate the performance of FCFR, we used SSFNet [12]. Since both variants of FCFR are similar except for the bit in the header, we compared FCFR's performance against Safeguard, ordered FIB updates (oFIB), and standard OSPF. Implementations of all schemes except FCFR were obtained from the authors of SafeGuard. oFIB prevents micro-loops during convergence by enforcing a strict order of routing updates across different routers. As in [9], we simulate the fast mode of oFIB which uses signaling messages to impose the update order. Further, oFIB is employed in combination with NotVia, to ensure that packets are fast rerouted through a backup path when they encounter a failed component during the convergence period. We used two well-known topologies Abilene and Exodus for the simulation. In addition, we used one randomly generated topology for comparison.

The schemes are evaluated using the following metrics: average convergence delay, average packet loss rate, average path stretch during convergence, and loop duration. This last metric captures the fraction of links in the network whose status change to up or down can cause loops and the duration they last. This study validates that OSPF causes loops during convergence whereas Safeguard, ordered updates, and FCFR effectively preclude loops from forming during convergence.

For each topology, we simulated four different failure scenarios: link down, node down, link up and node up. For example, in the first scenario, we randomly generate a single link failure in the network, and repeat the simulation for 100 different runs. For each failure scenario, we record data for the four metrics listed, and then report the average metric or cumulative distribution function for each type of failure.

A. Convergence Delay

The first metric evaluated during the simulation was the *convergence delay* of each protocol. As illustrated in Figure 7, the convergence delay of OSPF, SafeGuard, and FCFR is the same for all four topologies. This is because under these protocols, each node updates its FIB immediately upon receipt of an LSA. Therefore, there is no additional delay required due to ordered updates or obtaining some kind of routing consensus prior to convergence. This feature of our scheme satisfies the first part of our goal of fast convergence with fast rerouting. The ordered updates approach takes longer to converge, particularly for node failures. One effect of this delay is that the network is performing sub-optimal routing for a longer period of time by relying on the underlying fast reroute mechanism. The major impact, however, is that the network is not prepared to deal with another failure because it is still compensating for a previous event.

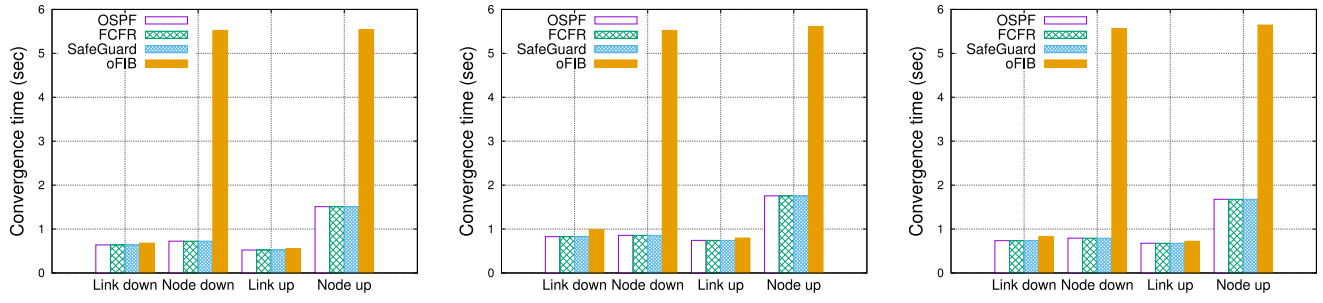
With our approach, we can achieve both fast rerouting and fast convergence. During the time the network is recomputing new routes, a fast reroute scheme like NotVia can redirect packets to their destinations, albeit over a sub-optimal path. This in itself is no better or worse than any other fast reroute scheme. However, our key advantage is that during convergence, we can achieve loop-freedom without incurring any additional cost in terms of convergence delay. Therefore, the network is prepared as soon as possible to deal with another failure.

B. Packet Loss

We also compared the performance of these schemes in terms of their rate of *packet loss* during the process of convergence. Figures 8 and 9 show the packet loss incurred by all the schemes for single link and single node failures respectively. It is evident that all other schemes have significantly lower packet loss rate than OSPF. As expected, FCFR restores forwarding just as well as SafeGuard. Other measurements in the remaining topologies confirm this result as well.

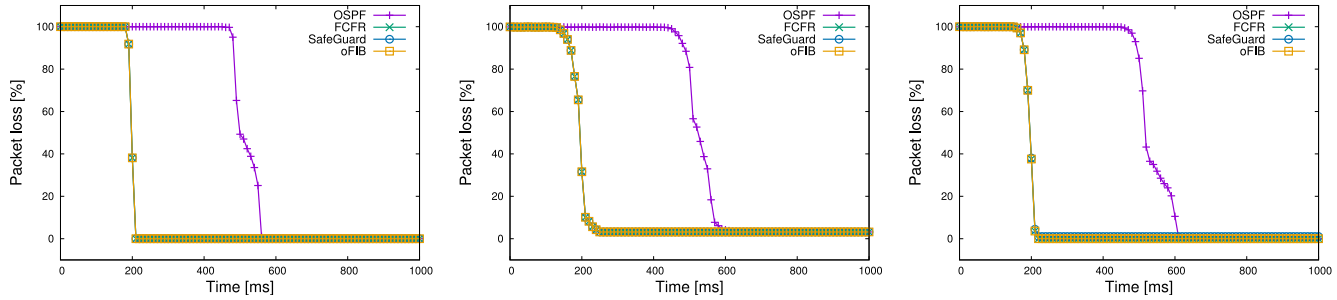
C. Path Stretch

The *path stretch* metric measures the optimality of routes during the convergence process. It is the ratio of the actual path length to the optimal path length. Any path stretch greater than 1 indicates a sub-optimal path; the higher the value, the longer



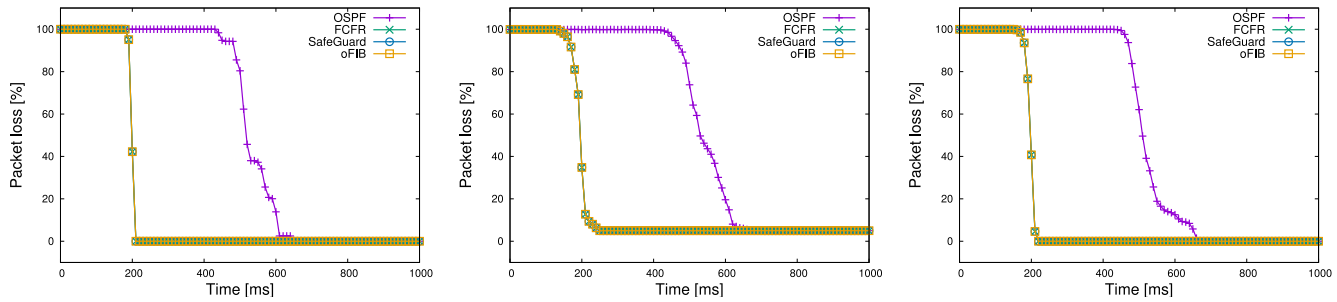
(a) Abilene topology (b) Exodus topology (c) Random topology

Fig. 7. OSPF, SafeGuard, and FCFR have minimal convergence delay, while ordered updates incurs higher delay, particularly for a node up or down event.



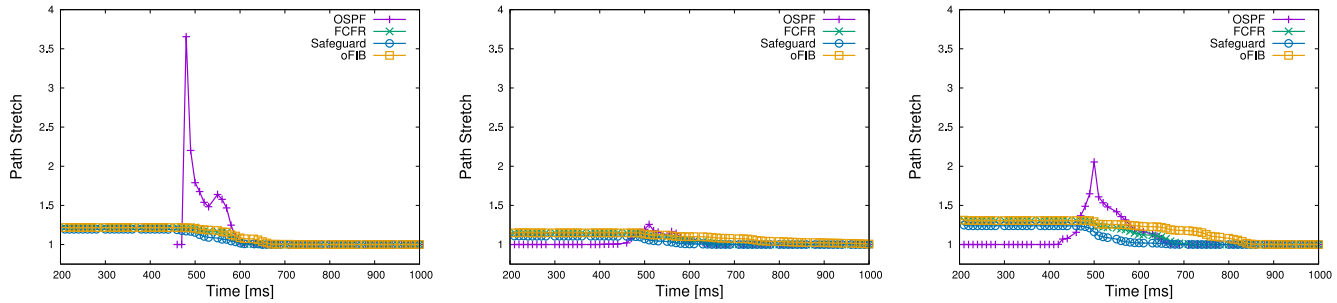
(a) Abilene (b) Exodus (c) Random

Fig. 8. Packet loss rates for a single link failure. Fast rerouting algorithms (i.e., FCFR, SafeGuard, and ordered updates) begin delivering packets immediately after the failure is detected, whereas OSPF continues to drop packets until forwarding tables are updated.



(a) Abilene (b) Exodus (c) Random

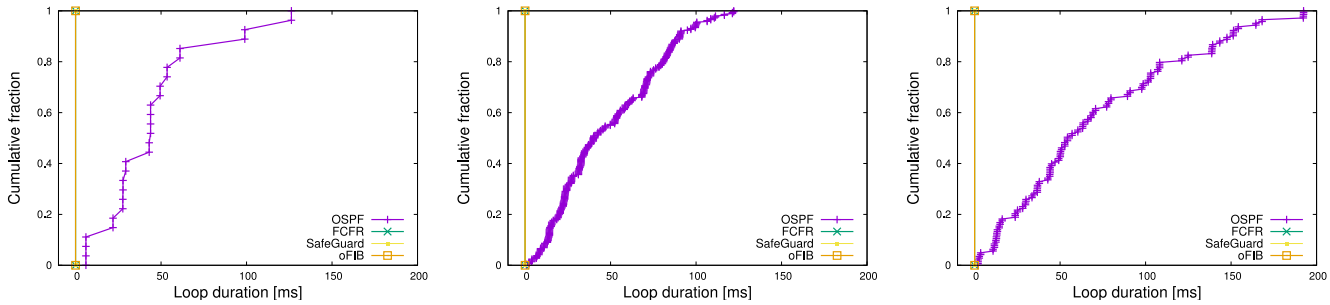
Fig. 9. Packet loss rates for a single node failure. These results are similar to those for a single link failure.



(a) Abilene (b) Exodus (c) Random

Fig. 10. Path stretch for a single link failure remains low initially for OSPF as packets get dropped at the failed link, then increases as routing inconsistencies start to occur. Stretch is higher initially with fast reroute schemes as they deliver packets through longer routes, then decreases as the network converges.

the route is compared to the optimal path. Fig. 10 illustrates the path stretch plotted over time for different topologies used in our simulation with single link failures. Similar behavior has been observed for single node failures too. The path stretch for OSPF is quite high during convergence mainly because some packets loop for a while and then escape the loop to reach



(a) Abilene (b) Exodus (c) Random

Fig. 11. The cumulative fraction of links vs. loop duration for a single link failure. SafeGuard, ordered updates, and FCFR guarantee loop prevention during the convergence period and therefore have a loop duration of zero for all links. OSPF enables routing loops to form during convergence, and thus a significant fraction of the links in the network have loops lasting about 100 ms.

the destination. Safeguard generates the shortest paths and converges the quickest back to the unit path length. Ordered updates also has lower path stretch than OSPF but takes longer to converge due to its strict updating requirements and signaling. We find that FCFR converges as quickly as Safeguard, but with the cost of at most one bit of overhead and no additional signaling, such as that required by ordered updates.

D. Flow Amplification

The final metric we measured is *loop duration*. This metric shows the duration of time a probe packet is caught in a transient loop before it is dropped. Figure 11 shows the CDF of the fraction of links whose status update caused loops vs. the loop duration for single link failures. OSPF clearly has a significant fraction of links that cause loops during convergence. Safeguard, ordered updates, and FCFR all have their own mechanism which prevents loops from forming. This behavior is confirmed by the simulation, which shows no forwarding loops for any of the above three schemes.

These results affirm that FCFR performs better than each of the competing approaches. It can achieve loop-free convergence unlike OSPF, with lower convergence delay than ordered FIB updates and lower header overhead than SafeGuard.

V. Limitations and Discussion

FCFR is critically dependent on the consistent interpretation of a packet's `era` field by different routers along its path. FCFR assumes single or correlated failures and does not guarantee loop-freedom in the case of multiple, independent failures. Another scenario in which the `era` field could be misinterpreted is when a failure partitions the network temporarily until the failure is repaired. The problem of synchronizing the `era` value between two segmented partitions of the network can be solved by utilizing OSPF's mechanism for synchronizing router databases. When two routers establish a new adjacency, they establish which router is the master and slave based on which has the higher router ID. The master node then controls the exchange of LSA information between the two routers. If one router has a more recent LSA than the other, the newer one is promulgated so that both routers have consistent databases.

In order to solve the problem of `era` synchronization, we can utilize this mechanism to resynchronize the era value when a network partition is repaired. In the case of identical eras on both sides of the partition, no action is needed. If both sides of the partition are in a different era, then the master router will impose its era on the slave. The slave router will then flood this new value out to all other nodes in that former partition. Once a network partition is repaired, all nodes will have a consistent view of the network topology.

VI. Related Work

The idea for FCFR was developed from other schemes which provide transient loop prevention, but at a greater cost in packet header information or signaling overhead across the network. Extensive work has been done in the area of loop-free convergence. The previously cited papers on Safeguard [9] and ordered updates [8] are good examples of this work. Compared with these two schemes, we showed that FCFR provides the same level of robustness without the cost of carrying remaining path length in all packets or the delay of waiting for all routers in the network to update in the proper order.

Another work in this area is the incremental update mechanism proposed in [13] and [14], which also alleviates the transient loop problem. However, this method is aimed largely at ISPs who need to conduct planned maintenance on a link and have a network management tool to implement the required metric changes. In contrast, FCFR could be applied in any network with little operator intervention and can react to any changes in the network, whether they are planned or not.

Failure carrying packets [15] tries to eliminate the need for convergence. However, the cost of this scheme is that it must maintain a list of failed links in the network, which is a significant modification to the routing protocol and the packet header, while FCFR works without requiring such changes.

Path Splicing proposed in [16] perturbs link weights to produce multiple routing trees and allows traffic to switch trees at any hop en route to the destination. While path splicing can sustain connectivity despite multiple failures, the number of splicing header bits needed is proportional to the number of hops and there is a small probability of forwarding loops.

The concept of Packet Recycling [17] takes advantage of cellular graph embeddings to reroute packets that would otherwise be dropped in case of failures. It needs $\log_2(d)$ bits in the header to cover all non-disconnecting failure combinations, where d is the diameter of the network. While the low header overhead is remarkable, packets under PR take long detours.

Loop Free Alternates [6] have been implemented in production networks. This approach works well depending on the network topology but cannot guarantee protection against multiple failures nor even any single link failure without modifying the network specifically to work with LFA's.

Keep forwarding [18] uses an approach to protect against k-link failures in the network by using interface-specific forwarding. However, KF also does not guarantee delivery even under single link failures. Another similar scheme [19] guarantees packet delivery under multiple link failures using rooted arc-disjoint spanning trees. However, this approach is not meant to eliminate routing loops caused during the convergence process.

Another relevant work is Slick Packets [20] which uses source controlled routing to avoid failures. However, SCR requires multiple round trips in some cases to determine an alternate route, which often exceeds the 50 ms standard for fast rerouting. In order to overcome this constraint, Slick Packets employs a directed acyclic graph called a *forwarding subgraph* (FS), which is encoded in each packet header. This FS specifies a set of paths that intermediate routers can use in case of failures. One drawback of SP is that it requires a large amount of overhead in each packet in order to encode the FS.

Recently, software defined networking (SDN), in which the control plane is centralized, is gaining in popularity [21]. This trend has increased the granularity of control over the forwarding plane, however at the cost of relying on a single point to distribute routing updates the entire network. To combine the advantages of SDN and traditional routing, Vissicchio *et al.* [22] describe a hybrid approach called Fibbing, which introduces fake nodes in the network to induce the desired forwarding tables. Inherently, link failures will necessitate changes in the forwarding plane and fake nodes, which requires recomputation by the control plane of each router in the network. Therefore, loop-free convergence with FCFR without any signalling overhead would aid Fibbing too.

An approach most related to this work was proposed in [23]. It also uses interface-specific forwarding to mitigate transient loops during convergence. But it prevents loops by discarding packets that arrive through unusual interfaces. In contrast, FCFR ensures loop-free convergence without any packet loss.

An earlier version of the FCFR approach with one bit in the header, i.e., FCFR_1, appeared in [24]. Compared to that version, apart from significant revisions including pseudocode and formal proof, this paper presents FCFR_0 that does not require any changes to the IP header or forwarding plane.

VII. Conclusion

In this paper, we proposed the FCFR scheme to prevent forwarding loops during the convergence period and restore the network to an optimal routing state as soon as possible. We have shown that FCFR with NotVia can achieve fast convergence with fast rerouting using only one bit per packet and an additional forwarding table per router, without any signaling overhead. Further, we have proved that, it is possible to achieve the same outcome using interface-specific forwarding in networks with symmetric link weights, without any changes to the packet format or the forwarding plane. We evaluated the performance of FCFR using SSFNet simulator and demonstrated that it has lower packet loss than OSPF, shorter convergence delay than ordered updates, and overall similar performance as SafeGuard with lower overhead. One drawback of FCFR, however, is that it can not gracefully handle multiple independent failure events that occur in succession, before each event has been completely absorbed by the network, addressing which is our immediate future work.

Acknowledgment

The authors would like to thank Xiaowei Wang and Ang Li for making their SafeGuard implementation in SSFNet available to them.

References

[1] A. Markopoulu, G. Iannaccone, S. Bhattacharyya, C.-N. Chuah, and C. Diot "Characterization of failures in an IP backbone," in *Proc. IEEE INFOCOM*, Hong Kong, Mar. 2004, pp. 2307–2317.

[2] U. Hengartner, S. B. Moon, R. Mortier, and C. Diot, "Detection and analysis of routing loops in packet traces," in *Proc. ACM IMW*, Marseilles, France, Nov. 2002, pp. 107–112.

[3] S. Bryant, S. Previdi, and M. Shand, "A framework for IP and MPLS fast reroute using not-via addresses," Internet Eng. Task Force, Fremont, CA, USA, RFC 6981, Aug. 2013.

[4] J. Wang and S. Nelakuditi, "IP fast reroute with failure inferencing," in *Proc. INM*, Kyoto, Japan, Aug. 2007, pp. 268–273.

[5] A. Kvalbein, A. F. Hansen, T. Cicic, S. Gjessing, and O. Lysne, "Fast IP network recovery using multiple routing configurations," in *Proc. IEEE Infocom*, Barcelona, Spain, Apr. 2006, pp. 1–11.

[6] G. Rétvári, J. Tapolcai, G. Enyedi, and A. Császár, "Ip fast ReRoute: Loop free alternates revisited," in *Proc. INFOCOM*, Shanghai, China, Apr. 2011, pp. 2948–2956.

[7] S. Bryant, C. Filsfils, S. Previdi, M. Shand, and N. So, "Remote loop-free alternate (LFA) fast reroute (FRR)," Internet Eng. Task Force, Fremont, CA, USA, RFC 7490, Apr. 2015.

[8] P. Francois and O. Bonaventure, "Avoiding transient loops during IGP convergence in IP networks," *ACM Trans. Netw.*, vol. 15, no. 6, pp. 1280–1292, Dec. 2007.

[9] A. Li, X. Yang, and D. Wetherall, "SafeGuard: Safe forwarding during routing changes," in *Proc. ACM CoNEXT*, Rome, Italy, 2009, pp. 301–312

[10] *Abilene Network*. Accessed on Feb. 28, 2017. [Online]. Available: https://en.wikipedia.org/wiki/Abilene_Network

[11] J. Postel *et al.*, "Internet protocol," Internet Eng. Task Force, Fremont, CA, USA, RFC 791, 1981.

[12] *Scalable Simulation Framework*. Accessed on Feb. 28, 2017. [Online]. Available: http://www.ssfnet.org

[13] F. Clad, P. Mérindol, J.-J. Pansiot, P. Francois, and O. Bonaventure, "Graceful convergence in link-state ip networks: A lightweight algorithm ensuring minimal operational impact," *IEEE/ACM Trans. Netw.*, vol. 22, no. 1, pp. 300–312, Feb. 2014.

[14] F. Clad, S. Vissicchio, P. Mérindol, P. Francois, and J.-J. Pansiot, "Computing minimal update sequences for graceful router-wide reconfigurations," *IEEE/ACM Trans. Netw.*, vol. 23, no. 5, pp. 1373–1386, Oct. 2015.

[15] K. Lakshminarayanan *et al.*, "Achieving convergence-free routing using failure-carrying packets," in *Proc. ACM SIGCOMM*, Kyoto, Japan, Aug. 2007, pp. 241–252.

[16] M. Motiwala, M. Elmore, N. Feamster, and S. Vempala, "Path splicing," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 4, pp. 27–38, 2008.

[17] S. S. Lor, R. Landa, and M. Rio, "Packet re-cycling: Eliminating packet losses due to network failures," in *Proc. ACM HotNets*, Monterey, CA, USA, Oct. 2010, Art. no. 2.
[18] B. Yang, J. Liu, S. Shenker, J. Li, and K. Zheng, "Keep forwarding: Towards k-link failure resilient routing," in *Proc. IEEE INFOCOM*, Toronto, ON, Canada, Apr. 2014, pp. 1617–1625.
[19] T. Elhourani, A. Gopalan, and S. Ramasubramanian, "IP fast rerouting for multi-link failures," in *Proc. IEEE INFOCOM*, Toronto, ON, Canada, Apr. 2014, pp. 2148–2156.
[20] G. T. K. Nguyen *et al.*, "Slick packets," *ACM SIGMETRICS Perform. Eval. Rev.*, vol. 39, no. 1, pp. 205–216, 2011.
[21] N. Feamster, J. Rexford, and E. Zegura, "The road to SDN," *Queue*, vol. 11, no. 12, p. 20, 2013.
[22] S. Vissicchio, O. Tilmans, L. Vanbever, and J. Rexford, "Central control over distributed routing," in *Proc. ACM SIGCOMM*, London, U.K., Aug. 2015, pp. 43–56.
[23] S. Nelakuditi, Z. Zhong, J. Wang, R. Keralapura, and C.-N. Chuah, "Mitigating transient loops through interface-specific forwarding," *Comput. Netw.*, vol. 52, no. 3, pp. 593–609, 2008.
[24] G. Robertson, J. Bedenbaugh, and S. Nelakuditi, "Fast convergence with fast reroute in IP networks," in *Proc. IEEE HPSR*, Richardson, TX, USA, Jun. 2010, pp. 100–106.

Glenn Robertson received the B.S. degree in electrical engineering from United States Military Academy, West Point, in 1996, the M.S. degree in computer engineering from Colorado Technical University in 2000, and the Ph.D. degree in computer science and engineering from the University of South Carolina, Columbia, in 2012. He is currently an Assistant Professor with the United States Military Academy. His current research interests include network reliability and cybersecurity education.

Nirupam Roy is currently pursuing the Ph.D. degree in electrical and computer engineering with the University of Illinois at Urbana–Champaign. His research interests are in the broad areas of networking, wireless systems, and mobile computing. He was a recipient of the Best Student Project Award for his bachelor's thesis from IIEST, Shibpur, India, the Outstanding Thesis Award for his master's thesis from the University of South Carolina, Columbia, and the M.E. Van Valkenburg Graduate Research Award from the University of Illinois at Urbana–Champaign in 2015.

Phani Krishna Penumarthi received the M.S. degree in computer science and engineering from the Indian Institute of Technology Madras, in 2012. He is currently pursuing the Ph.D. degree with the Department of Computer Science and Engineering, University of South Carolina. His research interests lie in the intersection of wireless networks and software defined networking.

Srihari Nelakuditi received the B.E. degree from Andhra University, Visakhapatnam, the M.Tech. degree from the Indian Institute of Technology Madras, and the Ph.D. degree in computer science and engineering from the University of Minnesota, Minneapolis. He is currently a Professor with the Department of Computer Science and Engineering, University of South Carolina, Columbia. His research interests are in resilient routing, wireless networking, and mobile computing. He was a recipient of the NSF CAREER Award in 2005 and the Google Faculty Research Award in 2013.

Jason M. O'Kane received the B.S. degree from Taylor University, in 2001, and the M.S. and Ph.D. degrees from the University of Illinois at Urbana–Champaign, in 2005 and 2007, respectively, all in computer science. He is an Associate Professor of Computer Science and Engineering and the Director of the Center for Computational Robotics, University of South Carolina. His research spans algorithmic robotics, planning under uncertainty, and computational geometry. He was a recipient of the NSF CAREER Award in 2010.