

Network Working Group  
Internet-Draft  
Intended status: Informational  
Expires: May 3, 2009

S. Bryant  
M. Shand  
Cisco Systems  
October 30, 2008

IPFRR in the Presence of Multiple Failures  
draft-bryant-shand-ipfrr-multi-01

Status of this Memo

By submitting this Internet-Draft, each author represents that any applicable patent or other IPR claims of which he or she is aware have been or will be disclosed, and any of which he or she becomes aware will be disclosed, in accordance with [Section 6 of BCP 79](#).

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

The list of current Internet-Drafts can be accessed at <http://www.ietf.org/ietf/lid-abstracts.txt>.

The list of Internet-Draft Shadow Directories can be accessed at <http://www.ietf.org/shadow.html>.

This Internet-Draft will expire on May 3, 2009.

Abstract

IP Fast Reroute (IPFRR) work in the IETF has focused on the single failure case, where the failure could be a link, a node or a shared risk link group. This draft describes possible extensions to not-via IPFRR that under some circumstances allow the repair of multiple simultaneous failures.

Requirements Language

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119](#) [1].

Internet-Draft

Multi-failure IPFRR

October 2008

## Table of Contents

<a href="#">1.</a>	Introduction . . . . .	<a href="#">3</a>
<a href="#">2.</a>	The Problem . . . . .	<a href="#">3</a>
<a href="#">3.</a>	Outline Solution . . . . .	<a href="#">4</a>
<a href="#">4.</a>	Looping Repairs . . . . .	<a href="#">5</a>
<a href="#">4.1.</a>	Dropping Looping Packets . . . . .	<a href="#">6</a>
<a href="#">4.2.</a>	Computing non-looping Repairs of Repairs . . . . .	<a href="#">6</a>
<a href="#">4.3.</a>	N-level Mutual Loops . . . . .	<a href="#">8</a>
<a href="#">5.</a>	Mixing LFAs and Not-via . . . . .	<a href="#">8</a>
<a href="#">6.</a>	Security Considerations . . . . .	<a href="#">9</a>
<a href="#">7.</a>	IANA Considerations . . . . .	<a href="#">9</a>
<a href="#">8.</a>	References . . . . .	<a href="#">10</a>
<a href="#">8.1.</a>	Normative References . . . . .	<a href="#">10</a>
<a href="#">8.2.</a>	Informative References . . . . .	<a href="#">10</a>
	Authors' Addresses . . . . .	<a href="#">10</a>
	Intellectual Property and Copyright Statements . . . . .	<a href="#">11</a>

## [1.](#) Introduction

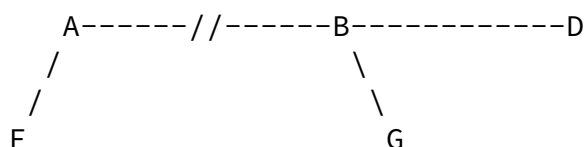
Work on IP fast reroute (IPFRR) in the IETF Framework [\[3\]](#), Basic [\[4\]](#) and not-via [\[2\]](#) has so far been restricted to the case of repair of a single failure. The single failure cases considered are a single link, a single node or a shared risk link group (SRLG). IPFRR repair of multiple simultaneous failures which are not members of a known SRLG have not been addressed because of concerns that the use of multiple concurrent repairs may result in looping repair paths. In order to prevent such loops, the current definition of IPFRR using not-via requires that packets addressed to a not-via address are not repaired but instead are dropped.

It is possible that a network may experience multiple simultaneous failures. This may be due to simple statistical effects, but the more likely cause is unanticipated SRLGs. When multiple failures which are not part of an anticipated group are detected, repairs are abandoned and the network reverts to normal convergence. Although safe, this approach is somewhat draconian, since there are many circumstances where multiple repairs do not induce loops.

This Internet draft explores the properties of multiple unrelated failures and proposes some methods that may be used to address this problem using not-via techniques.

## [2.](#) The Problem

Let us assume that the repair mechanism is based on not-via repairs. LFA or downstream routes may be incorporated, and will be dealt with later.



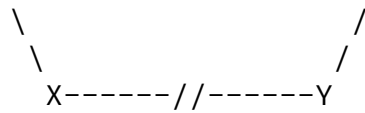


Figure 1: The General Case of Multiple failures

Note that depending on the repair case under consideration there may be other paths present in Figure 1, for example between A and B, and/or between X and Y. These paths are omitted for graphical

clarity.

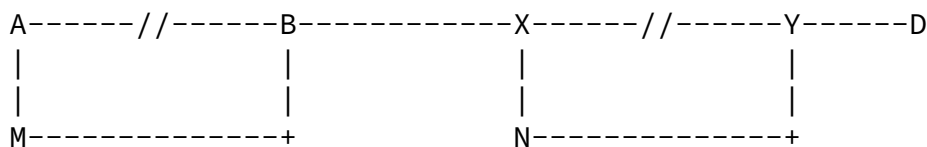


Figure 2: Concatenated Repairs

There are three cases to consider:

- 1) Consider the general case of a pair of protected links A-B and X-Y as shown in the network fragment shown Figure 1. If the repair path for A-B does not traverse X-Y and the repair path for X-Y does not traverse A-B, this case is completely safe and will not cause looping or packet loss.

A more common variation of this case is shown in Figure 2, which shows two failures in different parts of the network in which a packet from A to D traverses two concatenated repairs.

- 2) In Figure 1, the repair for A-B traverses X-Y, but the repair for X-Y does not traverse A-B. This case occurs when the not-via path from A to B traverses link X-Y, but the not-via path from X to Y traverses some path not shown in Figure 1. In standard not-via the repaired packet for A-B would be dropped when it reached X-Y, since the repair of repaired packets is currently forbidden. However, if this packet were allowed to be repaired, the path to D

would be complete and no harm would be done, although two levels of encapsulation would be required.

3) The repair for A-B traverses X-Y AND the repair for X-Y traverses A-B. In this case unrestricted repair would result in looping packets and increasing levels of encapsulation.

The challenge in applying IPFRR to a network that is undergoing multiple failures is, therefore, to identify which of these cases exist in the network and react accordingly.

### 3. Outline Solution

When A is computing the not-via repair path for A-B (i.e. the path for packets addressed to Ba, read as "B not-via A") it is aware of the list of nodes which this path traverses. This can be recorded by a simple addition to the SPF process, and the not-via addresses associated with each forward link can be determined. If the path

were A, F, X, Y, G, B, (Figure 1) the list of not-via addresses would be: Fa, Xf, Yx, Gy, Bg. Under standard not-via operation, A would populate its FIB such that all normal addresses normally reachable via A-B would be encapsulated to Ba when A-B fails, but traffic addressed to any not-via address arriving at A would be dropped. The new procedure modifies this such that any traffic for a not-via address normally reachable over A-B is also encapsulated to Ba unless the not-via address is one of those previously identified as being on the path to Ba, for example Yx, in which case the packet is dropped.

The above procedure allows cases 1 and 2 above to be repaired, while preventing the loop which would result from case 3.

Note that this is accomplished by pre-computing the required FIB entries, and does not require any detailed packet inspection. The same result could be achieved by checking for multiple levels of encapsulation and dropping any attempt to triple encapsulate. However, this would require more detailed inspection of the packet, and causes difficulties when more than 2 "simultaneous" failures are contemplated.

So far we have permitted benign repairs to coexist, albeit sometimes

requiring multiple encapsulation. Note that in many cases there will be no performance impact since unless both failures are on the same node, the two encapsulations or two decapsulations will be performed at different nodes. There is however the issue of the MTU impact of multiple encapsulations.

In the following section we consider the various strategies that may be applied to case 3 - mutual repairs that would loop.

#### [4.](#) Looping Repairs

In case 3, the simplest approach is to simply not install repairs for repair paths that might loop. In this case, although the potentially looping traffic is dropped, the traffic is not repaired. If we assume that a hold-down is applied before reconvergence in case the link failure was just a short glitch, and if a loop free convergence mechanism further delays convergence, then the traffic will be dropped for an extended period. In these circumstances it would be better to "abandon all hope" (AAH) [[draft-bryant-francois-shand-ipfrr-aah-00.txt](#)] and immediately invoke normal re-convergence.

Note that it is not sufficient to expedite the issuance of an LSP reporting the failure, since this may be treated as a permitted simultaneous failure by the oFIB algorithm. It is therefore

necessary to explicitly trigger an oFIB AAH.

##### [4.1.](#) Dropping Looping Packets

One approach to case 3 is to allow the repair, and to experimentally discover the incompatibility of the repairs if and when they occur. With this method we permit the repair in case 3 and trigger AAH when a packet drop count on the not-via address has been incremented. Alternatively, it is possible to wait until the LSP describing the change is issued normally (i.e. when X announces the failure of X-Y). When the repairing node A, which has precomputed that X-Y failures are mutually incompatible with its own repairs receives this LSP it can then issue the AAH. This has the disadvantage that it doesn't overcome the hold-down delay, but it requires no "data-driven" operation, and it still has the required effect of abandoning the

oFIB which is probably the longer of the delays (although with signalled oFIB this should be sub-second).

Whilst both of the experimental approaches described above are feasible, they tend to induce AAH in the presence of otherwise feasible repairs, and they are contrary to the philosophy of repair pre-determination that has been applied to existing IPFRR solutions.

#### 4.2. Computing non-looping Repairs of Repairs

An alternative approach to simply dropping the looping packets, or to detecting the loop after it has occurred, is to use secondary SRLGs. With a link state routing protocol it is possible to precompute the incompatibility of the repairs in advance and to compute an alternative SRLG repair path. Although this does considerably increase the computational complexity it may be possible to compute repair paths that avoid the need to simply drop the offending packets.

This approach requires us to identify the mutually incompatible failures, and advertise them as "secondary SRLGs". When computing the repair paths for the affected not-via addresses these links are simultaneously failed. Note that the assumed simultaneous failure and resulting repair path only applies to the repair path computed for the conflicting not-via addresses, and is not used for normal addresses. Note that this implies that although there will be a longer repair path when there is more than one failure, if there is a single failure the repair path length will be "normal".

Ideally we would wish to only invoke secondary SRLG computation when we are sure that the repair paths are mutually incompatible. Consider the case of node A in figure 1. A first identifies that the repair path for A-B is via F-X-Y-G-B. It then explores this path

determining the repair path for each link in the path. Thus, for example, it performs a check at X by running an SPF rooted at X with the X-Y link removed to determine whether A-B is indeed on X's repair path for packets addressed to Yx.

Some optimizations are possible in this calculation, which appears at first sight to be order  $hk$  (where  $h$  is the average hop length of repair paths and  $k$  is the average number of neighbours of a router).

When A is computing its set of repair paths, it does so for all its k neighbours. In each case it identifies a list of node pairs traversed by each repair. These lists may often have one or more node pairs in common, so the actual number of link failures which require investigation is the union of these sets. It is then necessary to run an SPF rooted at the first node of each pair (the first node because the pairings are ordered representing the direction of the path), with the link to the second node removed. This SPF, while not an incremental, can be terminated as soon as the not-via address is reached. For example, when running the SPF rooted at X, with the link X-Y removed, the SPF can be terminated when Yx is reached. Once the path has been found, the path is checked to determine if it traverses any of A's links in the direction away from A. Note that, because the node pair XY may exist in the list for more than one of A's links (i.e. it lies on more than one repair path), it is necessary to identify the correct list, and hence link which has a mutually looping repair path. That link of A is then advertised by A as a secondary SRLG paired with the link X-Y. Also note that X will be running this algorithm as well, and will identify that XY is paired with A-B and so advertise it. This could perhaps be used as a further check.

The ordering of the pairs in the lists is important. i.e. X-Y and Y-X are dealt with separately. If and only if the repairs are mutually incompatible, we need to advertise the pair of links as a secondary SRLG, and then ALL nodes compute repair paths around both failures using an additional not-via address with the semantics not-via A-B AND not-via X-Y.

A further possibility is that because we are going to the trouble of advertising these SRLG sets, we could also advertise the new repair path and only get the nodes on that path to perform the necessary computation. Note also that once we have reached Q space with respect to the two failures we need no longer continue the computation, so we only need to notify the nodes on the path that are not in Q-space.

One cause of mutually looping repair paths is the existence of nodes with only two links, or sections of the network which are only bi-connected. In these cases, repair is clearly impossible - the

failure of both links partitions the network. It would be



advantageous to be able to identify these cases, and inhibit the fruitless advertisement of the secondary SRLG information. This could be achieved by the node detecting the requirement for a secondary SRLG, first running the not-via computation with both links removed. If this does not result in a path, it is clear that the network would be partitioned by such a failure, and so no advertisement is required.

#### [4.3.](#) N-level Mutual Loops

It is tempting to conclude that the mechanism described above can be applied to the general case of N failures. If we use the approach of assuming that repairs are not mutual, and catching the loops and executing AAH when they occur, then we can attempt repairs in the case of N failures.

If we use the approach of avoiding potentially mutual repairs and creating secondary SRLG, then we have to explore N levels of repair, where N is the number of simultaneous failures we wish to repair.

#### [5.](#) Mixing LFAs and Not-via

So far in this draft we have assumed that all repairs use not-via tunnels. However, in practise we may wish to use loop free alternates (LFAs) or downstream routes where available. This complicates the issue, because their use results in packets which are being repaired, but NOT addressed to not-via addresses. If BOTH links are using downstream routes there is no possibility of looping, since it is impossible to have a pair of nodes which are both downstream of each other Basic [\[4\]](#).

Loops can however occur when LFAs are used. An obvious example is the well known node repair problem with LFAs Basic [\[4\]](#). If one link is using a downstream route, while the other is using a not-via tunnel, the potential mechanism described above would work provided it were possible to determine the nodes on the path of the downstream route. Some methods of computing downstream routes do not provide this path information. If the path information is however available, the link using a downstream route will have a discard FIB entry for the not-via address of the other link. The consequence is that potentially looping packets will be discarded when they attempt to cross this link.

In the case where the mutual repairs are both using not-via repairs, the loop will be broken when the packet arrives at the second failure. However packets are unconditionally repaired at downstream

routes, and thus when the mutual pair consists of a downstream route and a not-via repair, the looping packet will only be dropped when it gets back to the first failure. i.e. it will execute a single turn of the loop before being dropped.

There is a further complication with downstream routes, since although the path may be computed to the far side of the failure, the packet may "peel off" to its destination before reaching the far side of the failure. In this case it may traverse some other link which has failed and was not accounted for on the computed path. If the A-B repair (Figure 1) is a downstream route and the X-Y repair is a not-via repair, we can have the situation where the X-Y repair packets encapsulated to Yx follow a path which attempts to traverse A-B. If the A-B repair path for "normal" addresses is a downstream route, it cannot be assumed that the repair path for packets addressed to Yx can be sent to the same neighbour. This is because the validity of a downstream route must be ascertained in the topology represented by Yx, i.e. that with the link X-Y failed. This is not the same topology that was used for the normal downstream calculation, and use of the normal downstream route for the encapsulated packets may result in an undetected loop. If it is computationally feasible to check the downstream route in this topology (i.e. for any not-via address Qp which traverses A-B we must perform the downstream calculation for that not-via address in the topology with link Q-P failed.), then the downstream repair for Yx can safely be used. These packets cannot re-visit X-Y, since by definition they will avoid that link. Alternatively, the packet could be always repaired in a not-via tunnel. i.e. even though the normal repair for traffic traversing A-B would be to use a downstream route, we could insist that such traffic addressed to a not-via address MUST use a tunnel to Ba. Such a tunnel would only be installed for an address Qp if it were established that it did not traverse Q-P (using the rules described above).

## [6.](#) Security Considerations

Security considerations described in Framework [\[3\]](#), Basic [\[4\]](#) and not-via [\[2\]](#) apply to this work. Any additional security considerations will be provided in a future revision of this draft

## [7.](#) IANA Considerations

There are no IANA actions required by this draft.

## [8.](#) References

Bryant & Shand

Expires May 3, 2009

[Page 9]

---

Internet-Draft

Multi-failure IPFRR

October 2008

### [8.1.](#) Normative References

- [1] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", [BCP 14](#), [RFC 2119](#), March 1997.
- [2] Shand, M., Bryant, S., and S. Previdi, "IP Fast Reroute Using Not-via Addresses", [draft-ietf-rtgwg-ipfrr-notvia-addresses-02](#) (work in progress), February 2008.

### [8.2.](#) Informative References

- [3] Shand, M. and S. Bryant, "IP Fast Reroute Framework", [draft-ietf-rtgwg-ipfrr-framework-08](#) (work in progress), February 2008.
- [4] Atlas, A. and A. Zinin, "Basic Specification for IP Fast Reroute: Loop-Free Alternates", [RFC 5286](#), September 2008.

## Authors' Addresses

Stewart Bryant  
Cisco Systems  
250, Longwater Ave, Green Park,  
Reading RG2 6GB  
UK

Email: [stbryant@cisco.com](mailto:stbryant@cisco.com)

Mike Shand  
Cisco Systems  
250, Longwater Ave, Green Park,  
Reading RG2 6GB  
UK

Email: [mshand@cisco.com](mailto:mshand@cisco.com)

## Full Copyright Statement

Copyright (C) The IETF Trust (2008).

This document is subject to the rights, licenses and restrictions contained in [BCP 78](#), and except as set forth therein, the authors retain all their rights.

This document and the information contained herein are provided on an "AS IS" basis and THE CONTRIBUTOR, THE ORGANIZATION HE/SHE REPRESENTS OR IS SPONSORED BY (IF ANY), THE INTERNET SOCIETY, THE IETF TRUST AND THE INTERNET ENGINEERING TASK FORCE DISCLAIM ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY WARRANTY THAT THE USE OF THE INFORMATION HEREIN WILL NOT INFRINGE ANY RIGHTS OR ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

## Intellectual Property

The IETF takes no position regarding the validity or scope of any Intellectual Property Rights or other rights that might be claimed to pertain to the implementation or use of the technology described in this document or the extent to which any license under such rights might or might not be available; nor does it represent that it has made any independent effort to identify any such rights. Information on the procedures with respect to rights in RFC documents can be found in [BCP 78](#) and [BCP 79](#).

Copies of IPR disclosures made to the IETF Secretariat and any assurances of licenses to be made available, or the result of an attempt made to obtain a general license or permission for the use of such proprietary rights by implementers or users of this

specification can be obtained from the IETF on-line IPR repository at <http://www.ietf.org/ipr>.

The IETF invites any interested party to bring to its attention any copyrights, patents or patent applications, or other proprietary rights that may cover technology that may be required to implement this standard. Please address the information to the IETF at [ietf-ipr@ietf.org](mailto:ietf-ipr@ietf.org).