RTGWG
Internet-Draft
Intended status: Standards Track
Expires: July 26, 2015

P. Thubert, Ed.
Cisco
P. Bellagamba
Cisco Systems

January 22, 2015

# Available Routing Constructs
# draft-thubert-rtgwg-arc-03

## Abstract

This draft introduces the concept of ARC, a two-edged routing construct that forms its own fault isolation and recovery domain. The new paradigm can be leveraged to improve the network utilization and resiliency for unicast and multicast traffic in multiple environments, and is optimized to compute short reroute paths in case of breakages.

## Requirements Language

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119].

## Status of This Memo

## Copyright Notice

Table of Contents

## 1. Introduction

Traditional routing and forwarding uses the concept of path as the basic routing paradigm to get a packet from a source to a destination by following an ordered sequence of arrows between intermediate nodes. In this serial design, a path is broken as soon as a single arrow is, and getting around a breakage can require path re-computation, network re-convergence, and incur delays to till service is restored.

Multiple paths can be bound together for instance to form a Directed Acyclic Graph (DAG) to a destination, but that technique can be difficult to balance and cannot provide a full path redundancy even in the case of a biconnected graph. For instance, if the node that is closest to the DAG destination has only one link to that destination, then it does not have a alternate path to get to that destination.

It is also possible to compute an alternate routing topology for fast rerouting to a given destination, in which case some signaling, tagging or labeling can be put in place to indicate whether a packet follows the normal path or was rerouted over an alternate topology. Once a packet is rerouted, it is bound to the alternate topology so only one breakage can be handled with loop-free guarantees in most practical situations.

This draft introduces the concept of an Available Routing Construct (ARC) as a routing construct made of a bidirectional sequence of nodes and links with 2 outgoing edges, so that, upon a single breakage, each lively node in along ARC can still reach one of the outgoing edges.

The routing graph to reach a certain destination is expressed as a cascade of ARCs, each ARC providing its own independent domain of fault isolation and recovery. Unicast traffic may enter an ARC via any node but it may only leave the ARC through one of its two edges. One node along the ARC is designated as the Cursor. In normal unicast operations, the traffic inside an ARC flows away from the Cursor towards an edge. Upon a failure, packets may bounce on the breakage point and flow the other way along the ARC to take the other exit.

Aa a result an ARC is resilient to any single failure, and the recovery can be driven either from the data plane or the control plane. A second failure occurring within a same ARC will isolate an ARC segment. This can be further corrected from the control plane by reversing all the incoming Edges in a process that might recurse till

an exit is found. When ARC reversal is applied, an ARC topology is resilient to some cases of Shared Risk Link Group (SRLG) failures.

Properties of the Maximally Redundant Tree (MRT) and ARC are compared in [I-D.thubert-rtgwg-arc-vs-mrt]. The study shows that the reroute path that ARC derives is generally shorter than the alternate path that MRT computes. This property is largely due to the concept of cursor that delineates the shortest path on both sides of an ARC. Once a rerouted packet passes the cursor of the ARC in which it is rerouted, it should not cross a cursor again unless there is a second breakage later. It results that the packet follows the shortest path for the rest of the way, staying on the right side of each downstream ARC, when MRT would be following all subsequent eyes in the same direction.

This draft presents the concept and provides an intuition of how ARCs can simplify the operation and improve the network utilization and resiliency for all sorts of traffic in multiple environments, but defers to further documents to elaborate on the algorithms and optimizations in the different application domains. For instance, ARCs can also be used in datacenters for the purpose of fast-reroute, or within a service provider network to simplify load balancing operations or leverage optimally the ring topologies [RFC5921].

## 2. Terminology

The definition of the constituent parts of the "OAM" term is found in [RFC6291].

The draft uses the following terminology:

ARC: Available Routing Construct. An ARC is a loopless ordered set of nodes and links whereby traffic may enter via any node in the ARC but may only leave the ARC through either one of the ARC edges.

Comb: An ARC generalization: a Comb is a n-edged loopless set of nodes and links with n >= 2; traffic may enter via any node in the Comb but may only exit the Comb through one of its n edges. A Comb comes with a walk operation that enables to attempt to exit via every edge and to discover when all have been tried.

Cursor: A virtual point along an ARC that can be located on a node or on a link between 2 nodes. In normal operations, the traffic along the ARC flows away from its Cursor. If the Cursor is a node, then traffic can be distributed on both sides. The Cursor may be moved to change the way traffic is load balanced along an

ARC. It may also be placed at the location of a failure to direct traffic away from that point.

ARC Node: A Node that belongs to an ARC.

Edge ARC Node: An ARC Node at an edge of its ARC. An Edge ARC Node is a node via wich traffic can exit the ARC.

Edge Link: A directed link outgoing from an Edge ARC Node. Traffic can only exit from an ARC via an Edge Link. An Edge Link does not accept traffic into an ARC.

Intermediate ARC Node: A node that is not at an edge of an ARC. A Intermediate ARC Node node that can receive traffic and forward traffic between its adjacent nodes.

Intermediate Link: A link between two Intermediate ARC Nodes. An Intermediate Link is reversible, meaning that traffic is allowed in both directions though an individual packet is constrained in the way its direction is reversed. For stable links such as wired links, the typical constraint is that the direction of a packet may be reversed at most once along a given ARC.

Collapsed ARC: An ARC that is formed of a single node. This node is altogether the Cursor and both Edge Nodes. This implies that the node has at least 2 outgoing links to 2 different Safe Nodes.

```
                 |
                 |
                 V
               C+EAN
                /|\
               / | \
              |  V  |
              V     V

      E: Edge ARC Node          -|  collapsed in a single node
      C: Cursor                 -|
```

Figure 1: Collapsed ARC

Infrastructure ARC: An ARC that is formed of more than one node, which also means that the Edge Nodes are different nodes.

```
                 |    \  |                  |
                 |     \ |                  |      |
                 V       V                  V      |
             _->IAN<---->IAN<---->IAN<---->IAN<-_  |
           /                      + C            \ |
          /                                       \|
         V                                         V
        EAN                                       EAN
         |                                        /|\
         |                                       / | \
         |                                      |  V  |
         V                                      V     V

      IAN: Intermediate ARC Node  -|
      EAN: Edge ARC Node           |- All are Safe Nodes
      C: Cursor                   -|
```

Figure 2: Infrastructure ARC

DAG: Directed Acyclic Graph.

ARC Set (or Cascade): A DAG with ARCs as vertices. In the DAG, an edge between ARC A and ARC B corresponds to a link from an Edge ARC Node in ARC A and an arbitrary ARC Node in ARC B. Note that by definition, an ARC has at least 2 outgoing Edge Links, one per Edge Node, and maybe more if an Edge Node has multiple outgoing Edge Links. All vertices in the DAG have 2 forwarding solutions, even the ARC closest to the destination.

Omega: the abstract destination (== root) of an ARC Set. Omega is also referred as a complex destination in that it typically comprises more than one node and/or more than one link on a node. if Omega has a single node, then the plural interfaces on that node are considered as as many virtual node for the sake of the ARC computation algorithm.

ARC Height: An arbitrary distance from Omega that is associated to an ARC. The Height of an ARC must be more than the Height of any of the ARCs it terminates into. The order of ARC formation by a given algorithm can be used as a Height whereby an ARC is always strictly higher or lower than another.

Buttressing ARC: A split ARC that is merged into another ARC at one edge. An ARC and one or more Buttressing ARCs form a Comb construct that is resilient to additional breakages. A Buttressing ARC may be applied to an ARC or a Comb iff traffic

outgoing the Buttressing ARC Edge always reaches in an ARC that is lower than this ARC, or Omega.

```
                 |    \  |                  |
                 |     \ |                  |      |
                 V       V                  V      |
             _->IAN<---->IAN<---->IAN<---->IAN<-_----->IAN<-_
           /                      + C            \ |         \
          /                                       \|          \
         V                                         V           V
        EAN                                       EAN         EAN
         |                                        /|\          |
         |                                       / | \         |
         |                                      |  V  |        |
         V                                      V     V        V
```

Figure 3: Comb with Buttressing ARC

Safe Node: A node is Safe if there is no single point of failure - apart from the node itself - on its way to Omega. From this definition, a node is Safe if it has at least two non-congruent paths to two different other Safe Nodes. It results that a Safe node that is not Omega has at least two completely disjunct paths to Omega. When an ARC has been successfully constructed, all its nodes become safe with respect to the Omega for which the ARC was constructed. By extension for a collapsed path Omega is deemed to be Safe, that is any node that pertains in Omega is a Safe Node.

?-S: A node N is deemed dependent on a node S or S-dependent (denoted as ?-S) if S is the last single point of failure along N's shortest path to Omega.

## 3. ARC Set representations

An ARC Set can be represented in a number of fashions:

Graph View:

```
                     H2<==>H<==>H1<---I--->J1<==>J--->K1<===>K
                     |           |         |          |      |
                     |           |         |          |      |
                     V           V         V          V      V
               D1<==>D<==>D3    E1<==>E   F1<==>F<==>F2      G
               |           |     |    |    |         |      / \
               |           |     |    |    |         |     /   \
               V           V     V    V    V         V    V     V
               B1<==>B2<==>B3<==>B--->A<==>A1<------C2<==>C<==>C4
               |                      |    |                    |
               |                      |    |                    |
               |                      V    V                    |
               +--------------------> Omega <-------------------+
```

Figure 4: Routing Graph View

This representation is similar to a classical routing graph with the peculiarity that some Links are marked reversible. An ARC is represented as a sequence of reversible links. The node that holds the Cursor is also indicated somehow.

ARC View:

```
                          +========I========+
                          |                 |
                          |              +====J====+
                          |              |         |
                   +====H====+           |     +=====K=====+
                   |         |           |     |           |
            +====D====+   +====E====+  +====F====+  +====G====+
            |         |   |         |  |         |  |         |
            +=========B=========+   |  |   +=========C=========+
            |                   |   |  |   |                   |
            |                 +======A=======+                 |
            |                 |              |                 |
 ------------------------------------------------------------------Omega
```

Figure 5: ARC Representation

This ARC representation abstracts a whole ARC as a single vertex. An ARC ends in one or more other ARCs, but it has to be noted that even if both edges of an ARC end in a same other ARC, it ends in

fact in 2 different nodes, or Omega.  This is turn can be represented as a DAG as described in Paragraph 3.

Collapsed DAG view:

```
          +====+           +====+           +====+           +====+
          |  H |   <---    |  I |    --->   |  J |    --->   |  K |
             |     \__                         |        ___/    |
             |        \                        |       /        |
             V         _|                      V     |_         V
          +====+           +====+           +====+           +====+
          |  D |           |  E |           |  F |  <---     |  G |
             \  \         __/  \__         __/  \__        /  /
              \  \       /         \      /        \      /  /
               _| _|   |_           _|  |_          _|  |_ |_
                  +====+            +====+           +====+
                  |  B |    --->    |  A |   <---    |  C |
                     |                | |               |
                     V                V V               V
 ------------------------------------------------------------------Omega
```

Figure 6: ARC DAG

In the DAG representation, an ARC is abstracted as a vertex and links between ARCs are shown as directed edges.  This way, the reversible links are omitted and the graph is simplified.  It can be noted that even the vertex closest to Omega has 2 non-congruent forwarding solutions, that is Heir Links to Omega.

## 4.  Lowest ARC First

The open Lowest ARC First(oLAF) algorithm is presented below in such a way as to help the reader figure how an ARC Set can be obtained but not in a computer-optimized fashion that is left to be determined. oLAF is based on Dijkstra's algorithm for Shortest Path First (SPF) computation, and is designed in such a fashion that the reverse SPF tree towards a destination is conserved and preferred for forwarding along the resulting ARC Set.

We make the computation on behalf of Omega, that is an abstraction, but could represent the node or the set of nodes that we want to reach with an ARC Set.  If Omega is instantiated as an actual destination node, then that node may be a fine location for an ARC Computing Engine.

## 4.1. Init

So we start with an proverbial Initial Set of Nodes that are interconnected by Links, and Omega that is the destination that we want to reach with an ARC Set.

If there is no Heir, we're done.  If there is a single Heir then the graph is mono-connected, so we restart the computation taking that Heir off the Set of Nodes and making it Omega.

Else, if Omega is a single Node, or if Omega is composed of multiple nodes but we are willing to accept that both ends of an ARC terminate in a same node in Omega, then we create virtual Omega nodes, a minimum of two and at most one per Heir, and we make them the new Omega.  Note: we need at least two destinations because both ends of an ARC cannot terminate in a same node.

Now we can start building an ARC Set towards the resulting Omega.

In this process, we create so-called Dependent Sets of nodes, each owned by a Safe Node S, DSet(S).  DSet(S) contains nodes that are not determined to be Safe at the current stage of the computation and for which S, the owner Safe Node, is the last single point of failure on the shortest path tree to Omega.  It results that a given node can be at most in one DSet, and that a Safe Node belongs to its own DSet.

For each node S in Omega we create a DSet(S) in which we place S.

## 4.2. Growing Trees

And then the process goes like this:

We select the node in the Set of Nodes that is closest to Omega using the cost towards Omega as if we were building a traditional reverse SPF tree and we place the selected node in the same Dependent Set as its parent in the reverse SPF tree.  Note that for a Heir, the parent might be a real node in Omega, or a virtual Omega node.

If we kept it at that, we would be building subtrees that are hanging off a Safe Node and together would represent the reverse shortest path tree towards Omega, each subtree being grown separately inside DSet(S) where S is the (virtual) Safe node that is the root of the subtree.

### 4.3. Being Safe

But once we have placed the selected node in a DSet, we consider its neighbors one by one. If at least one of the neighbors is already in a different DSet than this node, we select the neighbor that provides the shortest alternate path to Omega for the selected node.

Doing so, we have isolated two paths:

- o one along its own shortest path that is contained within its own Dependent Set and that leads to the owner Safe Node of this set.

- o and one via the selected neighbor, along its own shortest path within the selected neighbor's Dependent Set and that leads to the owner Safe Node of that other set.

Because the two sets are different and have no intersection, these paths are non-congruent. And because the two non-congruent paths lead to two different Safe Nodes, this node is Safe.

It might happen that:

- o the selected node's parent is already a Safe Node, in which case the selected node is the Edge AN on its shortest path side.

- o It might also happen that the selected neighbor is already a Safe Node, in which case selected node is the Edge AN on its alternate side.

If both conditions are met for a same AN, then that AN forms a collapsed ARC by itself.

### 4.4. Bending An ARC

Now we form an ARC as follows:

- o A height is attributed to this ARC that must be strictly more than that of the ARCs it terminates into, if any. The order in which the ARCs are built may be used in some cases.

- o The ARC terminates in the two Safe Nodes that are the owners of the two DSets. The normal behaviour is to make a Edge Link the link to the Safe Node.

- o If the Safe Node at one end forms a collapsed ARC by itself, it may be absorbed in the ARC in order to build a multi-edged ARC.

- If one of the two Safe Nodes pertains in a ARC or a Comb construct that is higher than the other end, then this ARC may be merged at the Safe Node with its original ARC, in order to form a Comb construct whereby this ARC is a Buttressing ARC of the Comb. The resulting Comb conserves the height on the original ARC or Comb that it extends.

- The ARC is built by adjoining the two non-congruent paths that we isolated for the selected node.

- The selected node is the node farthest from Omega in the resulting ARC, so we make it the Cursor.

- The link between the selected node and the selected neighbor would not have been used in a classical reverse SPF tree. Here, we have determined that this link is in fact critical to connect 2 zones of the network (the DSets) that can act as a back up for one another in case of the failure of their respective single points of failure (the Safe Nodes).

- Because the ARC can be used in both directions, each AN along the ARC has two non-congruent paths to the Safe Nodes that the ARC terminates into. So it is a Safe Node. We create individual DSets for each AN and we move the AN to its own DSet.

## 4.5. Orienting Links

For each ARC Node along the ARC:

- any link (there can be zero for a collapsed ARC, one for an Edge AN or two of them for a Intermediate AN) between this AN and a next AN along this ARC is made an Intermediate Link, that is, reversible. The normal direction, away from the Cursor, preserves the shortest path.

- If this AN is an Edge AN for this ARC, than all links off this node that terminate in a Safe Node are made Edge Links, that is, outgoing but not reversible.

- All the other links left undetermined.

The nodes left in the Dependent Sets but the owner Safe Node are still not Safe. They are moved back to the original Set of Nodes to enable forming additional ARCs which might depend on this ARC in the ARC Set.

### 4.6. Looping or recursing

We are done processing the particular node we had picked in the original Set of Nodes. If the Set of Nodes as it stands now is not empty, we continue from Section 4.2.

If the Set of Nodes went empty, we are done with this pass and we consider the Dependent Sets that we have put together. In a biconnected graph, there should be one set per node and one node per set, denoting that every node is a Safe Node.

If some portion of the graph is mono-connected, then each mono-connected portion forms the Dependent Set of the Safe Node that is its single point of failure. In order to be maximally redundant, we need to form the ARCs again, within the Dependent Set.

To do so, we remove the Safe Node from the Dependent set and make it Omega. We make the resulting DSet our Set of Nodes and run the algorithm again.

This may recurse a number of times if the graph has mono-connected zones within others.

## 5. Forwarding Along An ARC Set

Under normal conditions, the traffic flows away from the Cursor of the current ARC and cascades into the next ARC on that side of the Cursor, with the Height of the current ARC decreasing monotonically from ARC to ARC till Omega is reached.

The same goes for a generic Comb construct. When Buttressing ARCs are applied on a main ARC or other Buttressing ARCs, the final construct assumes the shape of a tree. The tree may be walked in different manners but the shortest path requires to start going down the current ARC or Buttressing ARC to its Edge.

In case of Label forwarding, the same recursive technique is applied and a multiple ARC label path is constructed. Each ARC has is own set of label path per Omega, each ARC Set label path being merged into the lower ARC label set, thus at the interconnection point. At minimum, ARC label path should be built from the Cursor toward each edge, but this would require label path recompilation upon Cursor move, the proposed approach is then to build for the normal flow to an Omega one pair of label path from edge to edge.

As this label construct maps the ARC topology with local significant label, the Label Distribution Protocol (LDP) could be reused to announce label association to neighbors on the ARC.

Upon a breakage inside an ARC, until a corrective action takes place, some traffic will be lost. The corrective action might be either operated at the control plane or the data plane, if immediate action and near-zero packet loss is required.

## 5.1. Control Plane Recovery

Upon a first breakage in an ARC, the Cursor is moved to the breakage point, either a node or a link, so that traffic flows away from the Cursor again.

Upon a second breakage within a same ARC, a segment of the ARC is now isolated. Both breakage points become sinks till an additional corrective action, such as modifying the ARC Set, takes place. All incoming links in the isolated segment are blocked , causing the traffic to exit at the other end of the incoming ARCs.

Blocking an Edge Link in the incoming ARC may create an isolated segment in the incoming ARC as well if it is a second breakage there too, or if both edges of the incoming ARC terminate in the broken segment. In that case the process recurses and the broken zone can be determined as the collection of the isolated segments.

If a segment of an ARC is getting isolated by a dual failure but that ARC segment has incoming Edges then the ARC can be reversed. This reversal is done by reversing of all the incoming Edges, which become outgoing. The segment that was isolated now benefits from multiple exits in a loop free fashion. This process might in turn isolate a segment of an ARC that was incoming and the process recurses and some links flap. If a real exit exits the process will stabilize, but a count to infinity must be put in place to avoid a permanent flapping when a whole ARC Subset is physically isolated. One may consider that this process is in fact the classical link reversal technique, as applied to the DAG of ARCs.

## 5.2. Data Plane Recovery

Upon a breakage inside an ARC, it is possible in the data plane to reverse the direction of -to turn- a given packet once along the ARC so the packets exits over the other Edge Link. But in order to avoid loops, it is undesirable to reverse the direction of a given packet a second time.

Note that once a given packet leaves an ARC to enter the next, it is free to bounce again in the next ARC. In other Words, the domain that is impacted by a turn is limited to the current ARC itself; the ARC forms the event horizon wherein the notion that a turn happened may cause a loop.

So a local strategy must be put in place inside an ARC to allow a given packet to bounce once upon a breakage, and get dropped upon a second breakage.

In the case of IP packet forwarding, a packet can be tagged when it bounces inside an ARC, or when it passes the Cursor, for instance by reserving a TOS bit for that purpose. When the packet bounces, the bit is set and when the packet leaves the ARC, the bit is reset and may be used again in the next ARC. In the generic case of a Comb, a strategy must be put in place to walk the structure and drop a packet that tries all the Edges. it attempts to pass the Cursor twice in a same direction, meaning that more than a full walk was already accomplished.

### 5.2.1. Label Switched ARCs

In the case of MultiProtocol Label Switching (MPLS) forwarding, the same result can be achieved with Label Switched ARCs (LSARCs), that are composed of either 3 or 4 Labels Switched Paths (LSPs) along the ARC.

3-Labels method: In this case we lay a primary LSP from the cursor to the Edge in each direction, and a backup LSP Edge to Edge in each direction. So a node along the way has three labels, one primary and two backup, one in each direction. Should the primary path fail, the packet can be placed along the backup LSP in the other direction. We'll note that this method constraints the location of the Cursor. Should the Cursor move, The primary LSPs have to be recomputed, at a minimum between the old and the new location of the Cursor where the direction is reversed.

4-Labels method: In this case we have a primary and a backup LSPs in each direction all of them Edge to Edge, 4 labels total. The labels are independent of the location of the Cursor, so the Cursor can be moved from a node to the next in control plane with no impact on labels. This method consumes an additional label but is more amenable to load balancing techniques and allows each node that inject a packet inside an ARC to make its own decision of the exit edge for a given packet or flow.

### 5.2.2. Segment Routed ARCs

In the case of an infrastructure that is capable of Segment Routing (SR) [I-D.ietf-spring-segment-routing], the tag in the packet is in essence a Routing Header (RH) via the cursor. The RH forces routing to the destination all the way back up the broken ARC and then down on the other end. via the cursor of a broken ARC.

Upon a breakage, the node detecting the failure reroutes the packet towards the other edge, which means going backwards up to the cursor, and following normal routing from there.

The Routing Header may indicate, as consumed, an entry that points on the broken edge, if that is necessary for the cursor to figure out which is the broken edge so as to route towards the other edge.

### 5.3. Flooding

ARCs probably apply to both unicast and multicast traffic, as illustrated by [I-D.thubert-rtgwg-arc-bicast]. In particular, ARCs enable a redundant flooding of a packet. The flooded packet is injected at all edges ending in Omega, and from there swims upriver along the reverse ARC direction. The packet is then forwarded from the incoming edge of the ARC to the other edge where it is absorbed. On the way along the ARC, the packet is copied into all the ARCs that terminate in this ARC where the process recommences.

Since a packet is finally injected from both edges of any ARC, it should get to all nodes in an ARC even if there is one breakage in that ARC. in normal conditions, at least two copies of the packet circulate in an ARC, one in each direction, and a mechanism should be put in place to make sure that only one copy is injected in an incoming edge.

## 6. ARC Signaling

### 6.1. Serial ARC Representation

A single ARC can be serialized as the sets of endpoints at both edges and the ordered list of nodes in the ARC between the edges. Since the endpoints are effectively nodes in downstream ARCs, the set of all serialized ARCs provides a full description of the topology.

### 6.2. Centralized vs. Distributed computation

An ARC set can be computed with a slightly altered Shortest Path First algorithm, as further explained in Section 4. It results that any node, or all nodes participating to a Link State protocol, may learn the topology and compute an ARC Set. If all nodes compute the topology on their own and asynchronously, micro-loops will follow till the network converges.

It makes more sense to limit the computation of an ARC Set to specific nodes, typically a Path Computation Element (PCE), a Network Management Entity (NME), or nodes in Omega. This is typically what happens in a Software Define Networking (SDN) environment.

### 6.3. ARC Topology Injection

Regardless of the central entity that computes the ARC set, the new or updated ARC Set is serialized in a control message and flooded over itself from Omega as described in Section 5.3.

The new ARC Set can be used as soon as it is received, in the direction from which it is received, since a path along nodes in that direction exists already, through the nodes that forwarded the control messages. The full ARC redundancy is only available when a control message has been received along an ARC in both directions. In that model, there is no micro-loop.

### 6.4. ARC Operations, Administration, and Maintenance

Operations, Administration, and Maintenance (OAM) frames are used within an ARC and flow periodically or asynchronously from an edge to the other. Such frame may carry indications such as a breakage or a congestion, and may be used to control the load balancing, or link reversal operations.

## 7. Other ARC Operations

### 7.1. Node-Local vs. ARC-Wide reaction

ARCs enable forwarding plane reactions to breakages. In the simple case of a single breakage in an ARC, the reaction can be immediate to the discovery of the breakage and consists in rerouting the packet towards the other edge across the cursor, as explained in Section 5.2.

More complex situations require the coordination of all the nodes along an ARC. For instance, load balancing requires the knowledge of the congestion level at multiple points along the ARC, whereas the solution to the Shared Risk Link Group (SRLG) problem discussed in Section 7.3 requires all incoming edges in an isolated ARC segment to be blocked before they can be returned. For such ARC-Wide coordinated reactions, OAM frames are necessary to enable forwarding plane rapid reactions.

### 7.2. Load Balancing

In normal conditions, only the Cursor may distribute its traffic between the two Edge Nodes. If an Edge Node is still congested after the Cursor forwards all its traffic towards the other Edge Node, then the Cursor can be moved towards the congested Edge in order to derive even more traffic towards the other Edge. If both Edges are congested, then a back-pressure can be applied on the incoming ARCs

so that they move their own traffic towards their own alternate Edge. The process may recurse.

It is expected that control frames similar to those defined for MPLS Fault Management Operations, Administration, and Maintenance (OAM) [RFC6291] will echo from Edge Node to Edge Node provide information such as liveliness and load. In order to establish a control loop between the Edge Nodes and the Cursor, the OAM frame would carry at least a logical information whether:

The Edge Node is capable of forwarding data down to the next ARC

the load may be increased (e.g. rate below threshold including hysteresis)

the load should be decreased (e.g. congestion observed as increased latency or buffer bloat)

If the load should be decreased towards of congested Edge Node and the load may be increased towards the other then the Cursor may adjust its balancing of the load, or move Cursor ownership towards the congested Edge if it is already redirecting all the traffic towards the non-congested Edge.

If the Cursor is balancing traffic away from the default position due to a past congestion notification and the Edge that was congested now reports that the load may be increased, then the reverse operation can happen and the Cursor may balance the load back to the original position taking the reverse steps as above.

If the OAM can not be forwarded due to a link or a node failure, then the last node towards the broken Edge becomes Cursor and echoes the OAM frames advertising that it is an Edge node that is blocked, not capable of forwarding data down to the next ARC.

If both Edges are experiencing a congestion then the condition should be reported to the Edge Nodes of all incoming ARCs. Same goes when both Edges are blocked.

## 7.3. Shared Risk Link Group

Essentially, the Shared Risk Link Group (SRLG) problem is that a physical breakage may end up breaking more than one apparently unrelated IP links. such a breakage may end up breaking an ARC in more than one place, effectively creating isolated segments.

The basic approach to solve that problem is the classical link reversal technique. Since ARCs form a DAG as illustrated in

Figure 6, it is possible to return all the incoming edges in an isolated ARC segment so that traffic that circulates inside the segment is actually fed back in incoming ARCs. The incoming ARCs are considered broken on that edge so all the traffic is fed into the other edge. If this causes the incoming ARC to be doubly broken, the process recurses. in that incoming ARC. Over a number of iterations, if there is an exit, it will be found and the traffic will be funneled that way. If there is none, after a certain number of iterations, the process counts to infinity and stops.

If the iterations are performed too quickly, the process may cause micro-loops. OAM frames circulating within the broken segment can solve that issue. On the way in, the OAM frame should block all incoming ARCs, which effectively causes the edges to appear broken in incoming ARCs. On the way back, the OAM frame returns the incoming edges to be used the other way.

## 7.4. Olympic Rings

By Olympic Ring problem we mean how to optimally reuse the multiple path opportunities that interconnecting 2 rings enable. ARCs can simply be deployed inside a ring A to reach a connected ring B by installing an ARC between adjacent interconnections on the rings. If the rings only connect at one point, there is a single ARC going all the way around the ring, with the cursor at the far side. If there are more than one interconnection, then you always end up with as many ARCs as there are interconnections.

A packet being forwarded inside a ring picks the side of the ring that is away from the cursor, taking effectively the shortest path to the next ring. If a hop is broken, then the packet is returned to the other edge of the ARC, which is the adjacent interconnection between the ARCs.

Note: There is no need in that model to artificially disable one hop in the ring and re-enable it in case of breakage.

## 7.5. Routing Hierarchies

The ARC methods may be used to build and/or leverage routing hierarchies, allowing high availability at multiple hierarchical levels. In one hand, the view of an ARC Set can be simplified by abstracting an ARC as a node in a DAG. The view of the routing topology is thus simplified, as illustrated in Figure 6.

In the case of connected rings,abstracting a full ring as a node, ARCs can be applied to a graph of rings, providing another level of redundancy and an abstract end-to-end path computation, ring to ring

to ring. ARCs may be used to make that computation resilient as well.

## 8. Manageability

This specification describes a generic model. Protocols and management will come later

## 9. IANA Considerations

This specification does not require IANA action.

## 10. Security Considerations

This specification is not found to introduce new security threat.

## 11. Acknowledgments

The authors wishes to thank Dirk Anteunis, Stewart Bryant, IJsbrand Wijnands, George Swallow, Eric Osborne, Clarence Filsfils and Eric Levy-Abegnoli for their participation and continuous support to the work presented here.

## 12. References

### 12.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

### 12.2. Informative References

[I-D.ietf-spring-segment-routing]
Filsfils, C., Previdi, S., Bashandy, A., Decraene, B., Litkowski, S., Horneffer, M., Shakir, R., Tantsura, J., and E. Crabbe, "Segment Routing Architecture", draft-ietf-spring-segment-routing-00 (work in progress), December 2014.

[I-D.thubert-rtgwg-arc-bicast]
Thubert, P. and I. Wijnands, "Applying Available Routing Constructs to bicasting", draft-thubert-rtgwg-arc-bicast-01 (work in progress), October 2013.

[I-D.thubert-rtgwg-arc-vs-mrt]
Thubert, P., Enyedi, G., and S. Ramasubramanian, "Available Routing Constructs", draft-thubert-rtgwg-arc-vs-mrt-01 (work in progress), January 2014.

[RFC5921] Bocci, M., Bryant, S., Frost, D., Levrau, L., and L. Berger, "A Framework for MPLS in Transport Networks", RFC 5921, July 2010.

[RFC6291] Andersson, L., van Helvoort, H., Bonica, R., Romascanu, D., and S. Mansfield, "Guidelines for the Use of the "OAM" Acronym in the IETF", BCP 161, RFC 6291, June 2011.

## Authors' Addresses

Pascal Thubert (editor)
Cisco Systems, Inc
Building D
45 Allee des Ormes - BP1200
MOUGINS - Sophia Antipolis  06254
FRANCE

Phone: +33 497 23 26 34
Email: pthubert@cisco.com

Patrice Bellagamba
Cisco Systems
214 Avenue des fleurs
Saint-Raphael  83700
FRANCE

Phone: +33.6.1998.4346
Email: pbellaga@cisco.com