Technical Disclosure Commons

Defensive Publications Series

April 2022

Micro-loop free Delivery on the Post-Convergence Path after a Link change in the Topology

Anonymous

Follow this and additional works at: https://www.tdcommons.org/dpubs_series

Recommended Citation
Anonymous, "Micro-loop free Delivery on the Post-Convergence Path after a Link change in the Topology", Technical Disclosure Commons, (April 20, 2022)
https://www.tdcommons.org/dpubs_series/5074

This work is licensed under a Creative Commons Attribution 4.0 License.
This Article is brought to you for free and open access by Technical Disclosure Commons. It has been accepted for inclusion in Defensive Publications Series by an authorized administrator of Technical Disclosure Commons.

Micro-loop free Delivery on the Post-Convergence Path after a Link change in the Topology

ABSTRACT

This disclosure provides robust and generalized algorithms that allow prevention of micro-loops directly on the post-convergence path following a link failure (Link down) or upon re-convergence following a newly inserted link in the topology (Link up). Through this solution, if a remote link fails, traffic can leverage emulated local loop-free backup calculations on behalf of the PLR nodes without explicitly redirecting traffic to the PLR and instead directly tunnel traffic to reach the destination directly on the post convergence path (after the link churn) through a loop-free intermediary node (in a loop-free manner) during convergence, thus resulting in minimal to no traffic loss due to micro-loops on any link on the post convergence path.

DETAILED DESCRIPTION

Identification of Impacted Prefixes:

- The disclosure uses a modified Dijkstra's algorithm with vertex tagging to identify prefixes impacted by the failed or newly brought up link – this allows impacted prefixes to be identified unambiguously, without resorting to heuristics. This in turn allows this proposal to be applied to any topology of arbitrary complexity with no constraints on the number or type of links in the topology- for example, there is no requirement nor recommendation to use a ring topology for this proposal.
- For link down, the modified Dijkstra's algorithm identifies prefixes which were reachable via/behind the failed link, which now qualify for tunnel protection.
- For link-up scenarios, the modified Dijkstra's algorithm identifies destination prefixes that are behind both the near and far end PLR nodes as prefixes which now qualify for tunnel protection.
- The SPF based identification can protect prefixes in situations where the local next-hop or path for that prefix's route on the calculating router is not seen to change locally.
- Impacted prefixes identified in 3 and 4 are then sent using a directed tunnel to the PQ node or PQ segment identified in 1, which are loop free by derivation and can forward onward to the destination.

Other approaches for remote micro-loop avoidance, include the strategy of shipping to the near end Point of Local Repair (PLR), and then leveraging the switchover on the hardware forwarding entry at the PLR, which is less efficient than the scheme described in this proposal.

A framework or guideline is provided within which remote micro-loop avoidance solutions can be implemented on the post convergence path. It allows for remote micro-loop avoidance to be implemented, directly on the post convergence path using an intermediate node, which once reached is guaranteed to be loop-free, in the path onward

to the destination. This proposal provides clear, unambiguous algorithms to identify a loop free intermediate node on the post-convergence path for onward forwarding to the destination.

Remote link failures can also cause micro-loops as seen in Fig.2. Here, Node 1- Node 4 link has gone down, and Node 1 will kick off its Micro-loop solution which will help traffic running from Node 1 to Node 4. However, for traffic streams running from Node 3 to Node 4, this does not help, since micro-loops can occur round the ring topology based on convergence. Once again, this is because for routers Node 2, 3, 6 and 5, the pre-fault path to reach Node 4 is clockwise, and it takes time for all the routers to learn of and run SPF in response to the Node 1 – Node 4 link failure.

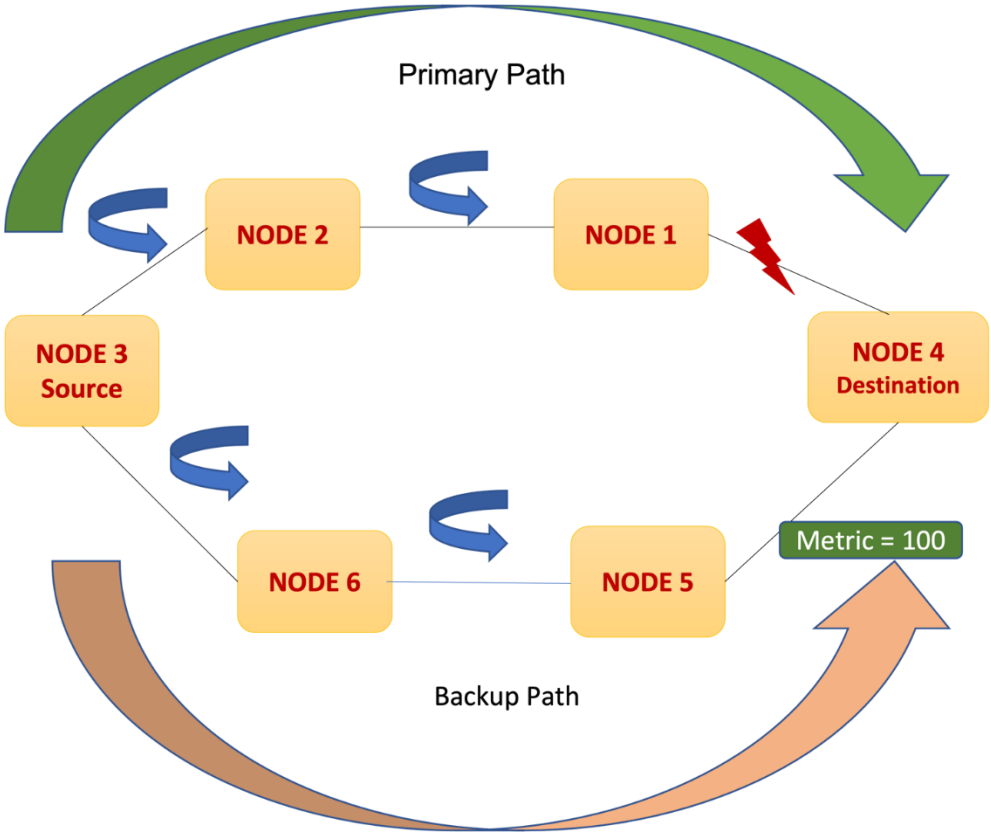

Figure 1

The Ship-To-PLR solution does provide a reasonably good solution to remote micro-loop avoidance, but when compared to this proposal, has additional overhead in the sense that it is less efficient because it first directs the traffic eligible for protection to the PLR, which then leverages its TiLFA backup to forward to its PQ node or P+Q segment to hair-pin the traffic through the source again. This is depicted in Fig. 2 below. Additionally, the specific version of Ship-To-PLR implemented in SAOS releases 10.6.0 and 10.7.0 uses a heuristic to identify impacted prefixes.

By contrast, this proposal does not re-direct traffic to the PLR and does not hair-pin traffic through the source. Instead, effectively, it calculates a TiLFA backup on behalf of the PLR, (which is guaranteed to be on the post-convergence path after the link down) and sends to the PQ node or P+Q segment identified by the backup path. This allows traffic to be directly sent on the post convergence path in a loop free manner. Additionally, the modified Dijkstra's algorithm used in this proposal is not a heuristic and can unambiguously identify impacted prefixes, (whose routes changed by the link up or link down) on any arbitrary topology with any number of links and any manner of interconnection.

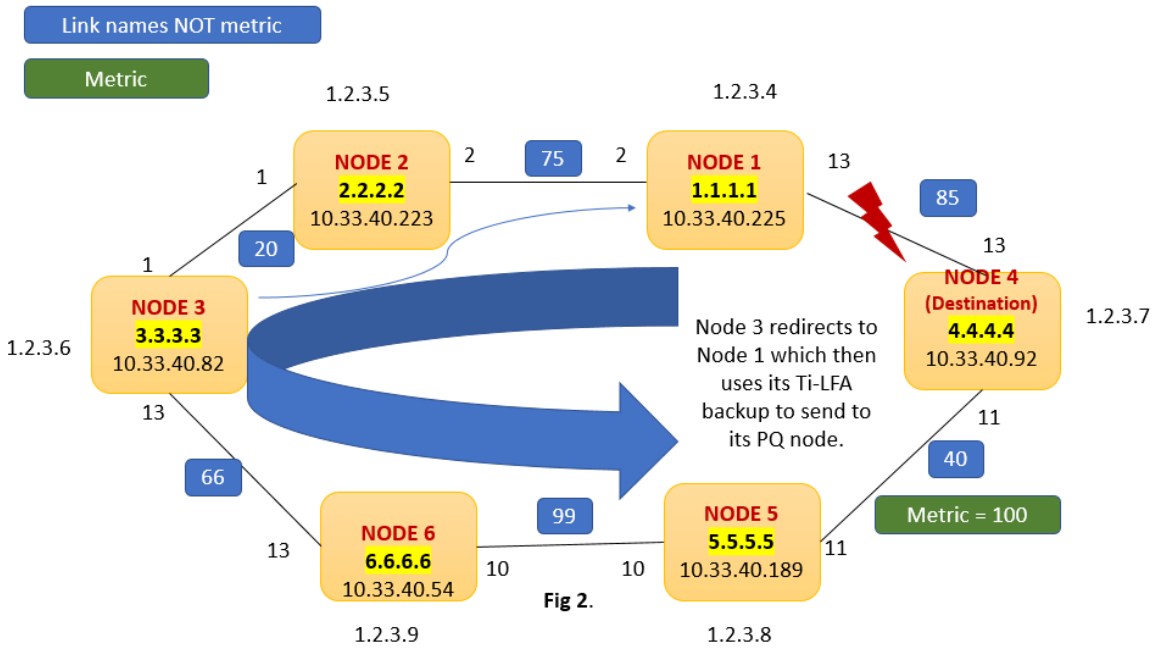

Figure 2

This proposal assumes symmetric link costs, meaning that for every link in the topology, the link cost of a link A-B at node A is the same as the link cost for the same link at node B (i.e., the link cost of the link in either direction is the same).

Ideally, the prefixes that need to be protected are the ones that are not in the P space of the failed link with regard to the SPF tree rooted at the source.

Assuming symmetric link costs means that the topology is a connected undirected graph, and every vertex in the topology will either be in the P space, or in the Q space – that is, between the P space and the Q space, every vertex in the topology will be covered – there cannot be vertices that are neither in the P space nor in the Q space.

This in turn allows us to use the Q space to identify all the prefixes that need protection, and the algorithm used here for the link down case, tracks all prefixes behind the PQ node or P+Q segment and marks them eligible for protection.

In the present solution, the following stages occur:

1. Identification of Link Down and Detection of Nearer End:
   - A router detects that a remote link has gone down and runs logic to identify the closer end of the link and the farther end of the link by measuring the distance to the nearer and farther end of the link
   - In Figure 3 below, Node 3 will detect the Node1-Node4 Link down (via IGP flooding) and will deduce that Node 1 is the closer endpoint to itself. Node 1 is identified as the near-end PLR and Node 4 is identified as the far-end PLR.
   - In the event that both ends of the link are seen to be equidistant from this router, no action is taken by this proposal.

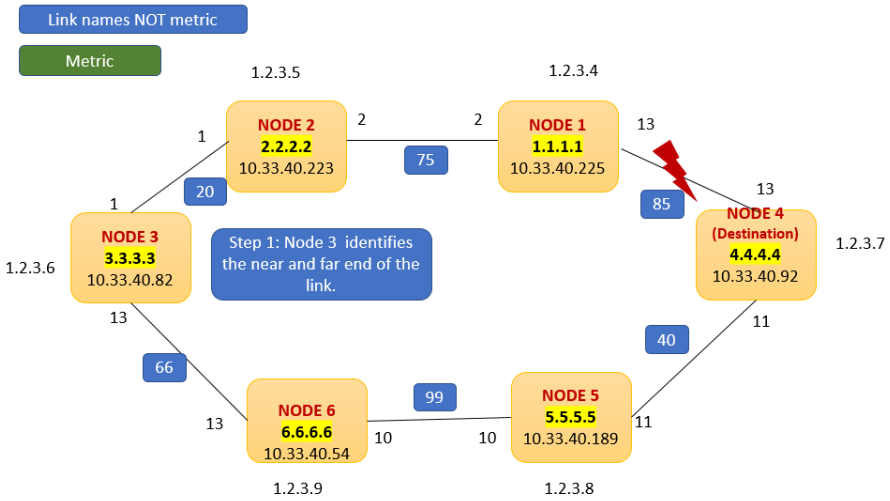

Figure 3

2. Emulation of PQ space calculation as if at PLR:
   - Each node sourcing traffic, upon receipt of LSPs identifying the downed link, computes an emulated SPF (reverse SPF) rooted at the near end node to identify the P space with respect to the failed link, on the SPF tree rooted at the node. P space is computed by excising the sub-tree hanging below the downed link from the SPF tree calculated above and removing all nodes in the excised sub-tree from the P space.
   - Each node sourcing traffic, upon receipt of LSPs identifying the downed link, computes an emulated SPF (reverse SPF) rooted at the far end of the link, identifying the Q space of the link. Q space is computed by excising the sub-tree hanging below the downed link from the SPF tree calculated above and removing all nodes in the excised sub-tree from the Q space.
   - The P space and the Q space are then intersected to identify either a single PQ node, or disjoint P and Q sets. This node or set of nodes is then preserved for the next step in 3 below.

- Labels to the intermediate nodes are generated after consulting the segment routing database. These labels are saved for use in the modified Dijkstra's run below.

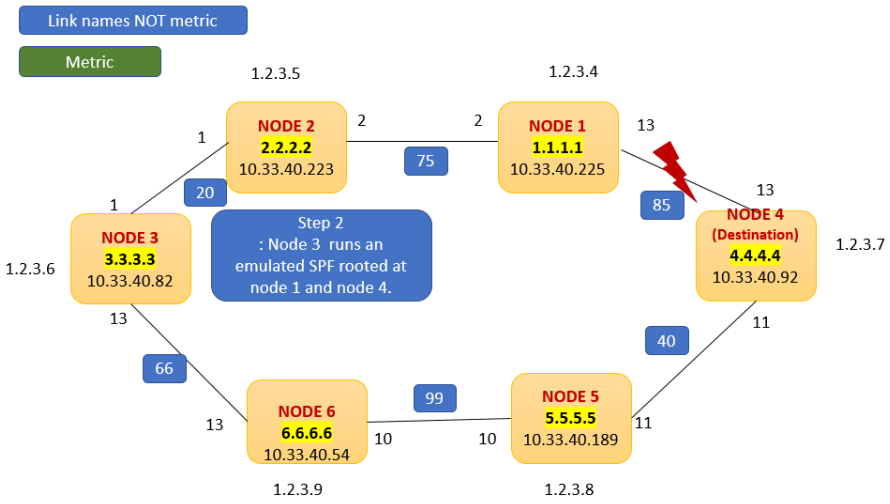

Figure 4

3. [Link Down] Modified Dijkstra's Run to Indentify Impacted and Protectable Destinations:
   - At each node in the network sourcing traffic, other than the PLR nodes, when pulling in optimal reachability candidate nodes from the TENT BHeap, each node is checked to see if it matches the PQ node computed in the previous step (in the case of single segment TiLFA) or the P Node or the Q Node in the case of disjoint P and Q spaces (double segment TiLFA). Additionally, its parent node is tracked and some flags on the parent node are inherited on each child node, as described below.
   - The SPF implementation needs to track, for every vertex pulled into the TENT BHeap, the parent vertex which pulled this vertex into TENT, while the vertex is in TENT and the cost at which the parent pulled the vertex into TENT. Note that this information may need to be updated as different parent vertices pull the same vertex into TENT. Most Dijkstra implementations will track this data already.
   - When pulling the most optimally reachable (candidate) node out of the TENT BHeap, if there is a single PQ node (P and Q spaces intersect on a single node), and if the candidate Node matches the PQ Node, then a MATCH_Q_NODE flag is set on the matching vertex along with a flag of CHILD_NODE_Q.
   - Else in the case of disjoint P and Q sets, when pulling the most optimally reachable (candidate) node out of the TENT BHeap, if the P and Q sets are disjoint, and if the candidate Node matches a Q Node (a node in the Q set), and if the parent which pulled in the Q Node into the TENT BHeap at its most

optimal cost matches a P Node (a node in the P set), then a MATCH_Q_NODE flag is set on the matching vertex along with a flag of CHILD_NODE_Q.

- Else, when pulling the most optimally reachable node out of the TENT BHeap, if the parent node has CHILD_NODE_Q, then the CHILD_NODE_Q flag is set on this vertex as well, by inheritance.
- At the end of the SPF run, vertices which have the CHILD_NODE_Q flag set qualify for protection, which will also apply to all prefixes attached to the vertex.
- Using the modified Dijkstra SPF listed in the steps above to identify prefixes previously reachable behind the failed link, allows the protection to begin as close to the traffic source as possible, and can protect prefixes in some topologies where the link down event may not result in a changed next-hop or path metric for the prefix's route. As an optimization, we could additionally stipulate that the next-hop or the (cumulative) path metric for any prefix marked eligible for protection in the previous step must have changed across convergence – but this is strictly optional.
- Note that these modifications are during the regular convergence SPF, which must be run for convergence purposes in all cases – there is no additional SPF required by this proposal other than the emulated SPFs at the near and far end as identified in step 2.

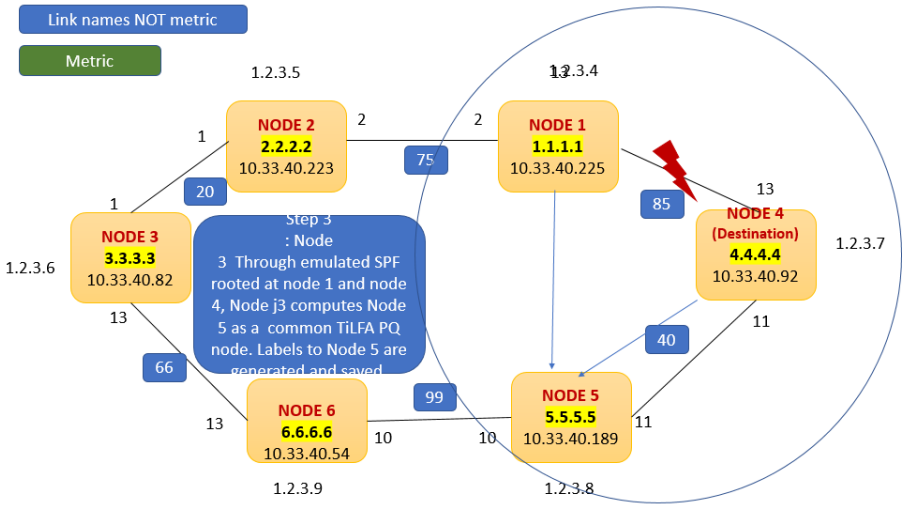

Figure 5

4. Link-down: Installation of Tunnels for Protected Destinations:
   - When generating labels for the destination prefix, labels to loop-free intermediate PQ node or P+Q segment generated in section 3 above are prepended to the destination label, adjusting for PHP cases.

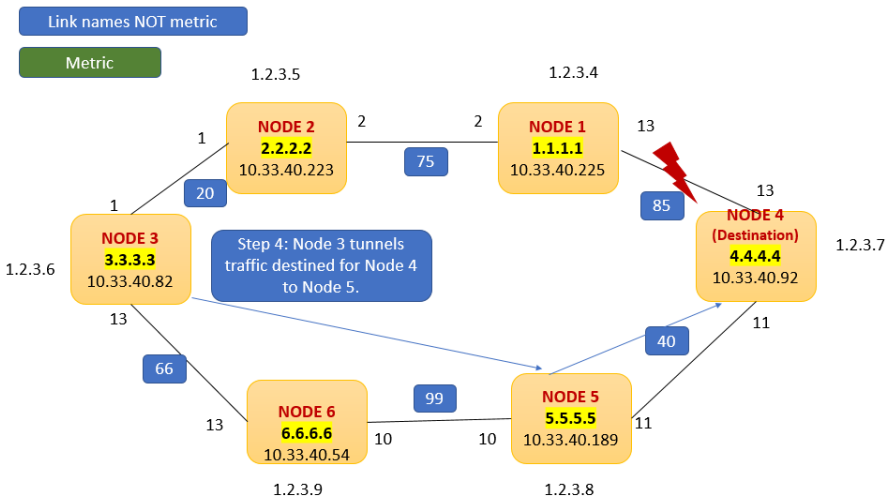

Figure 6

Description of Proposed Solution for Link Up Including Novel Algorithm:

Identification of Link-Up and Detection of Nearer End:

- A router detects that a remote link has now come up and runs logic to identify the closer end of the link and the farther end of the link by measuring the distance to the nearer and farther end of the link
- Node 3 will detect the Node1-Node4 Link up (via IGP flooding) and will deduce that Node 1 is the closer endpoint to itself. Node 1 is identified as the near-end PLR and Node 4 is identified as the far-end PLR.
- In the event that both ends of the link are seen to be equidistant from this router, no action is taken by this proposal.

○ [Link Up] Modified Dijkstra's Run to compute routes with tunnels through the Link that came up, for prefixes behind it.

- When pulling in optimal reachability candidate nodes from the TENT BHeap, each node is checked to see if it matches the near-end PLR node (in the case of single segment TiLFA) or the far-end PLR node. Additionally, its parent node is tracked and any flags on the parent node are tracked.
- The SPF implementation needs to track, for every vertex pulled into the TENT BHeap, the parent vertex which pulled this vertex into TENT, while the vertex is in TENT and the cost at which the parent pulled the vertex into TENT. Note that this information may need to be updated as different parent vertices pull the same vertex into TENT. Most Dijkstra implementations will do this already.
- When pulling the most optimally reachable node out of the TENT BHeap, if a Node matches the Far-End Node, and if the parent vertices pulling in the Far-End Node into the TENT BHeap is the Near-End node, a flag MATCH_FAR_NODE is set on the matching vertex along with a flag of CHILD_NODE_FAR.

- Else, when pulling the most optimally reachable node out of the TENT BHeap, if the parent node has CHILD_NODE_FAR, then the CHILD_NODE_FAR flag is set on this vertex as well by inheritance.
- At the end of the SPF run, vertices which have the CHILD_NODE_FAR flag qualify for protection, which will also apply to all prefixes attached to the vertex.
- Using the modified Dijkstra SPF to identify prefixes reachable behind the new link, allows the protection to begin as close to the traffic source as possible, and can protect prefixes in some topologies where the link up event may not result in a changed next-hop or path metric for the prefix's route.
- Note that these modifications are during the regular convergence SPF, which must be run for convergence purposes in all cases – there is no additional SPF required by this proposal for the link up scenario.

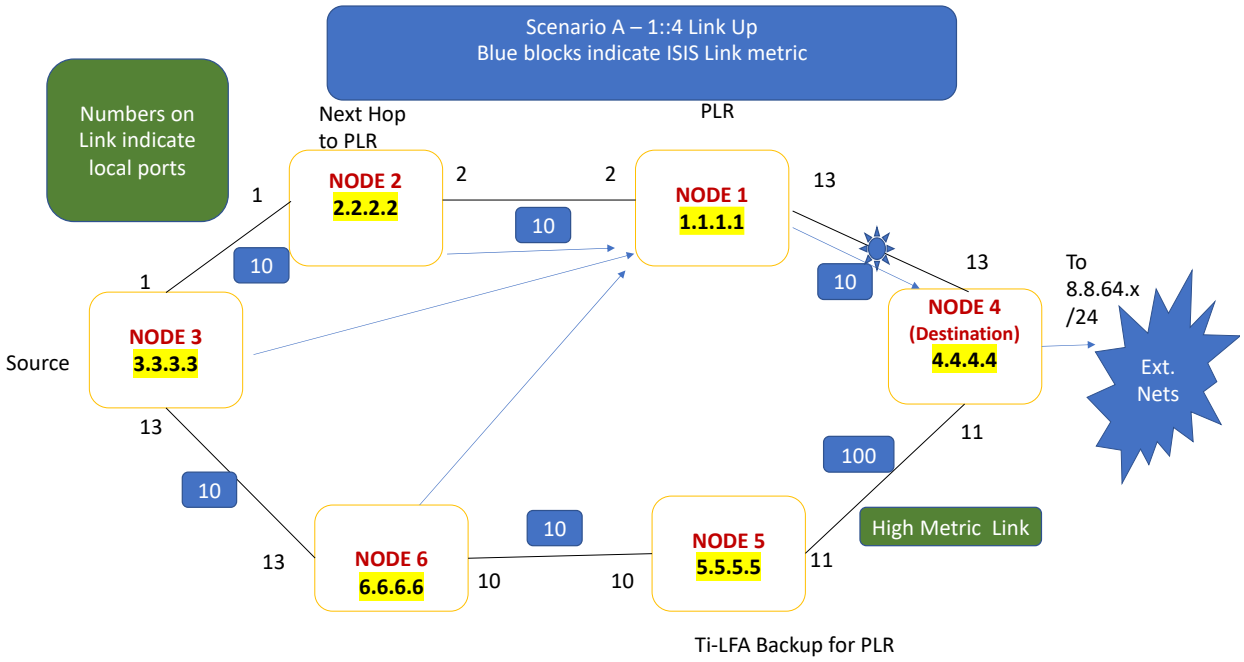

Figure 7

Link-up Route Handling at the PLR:

When installing routes to the impacted destinations as identified above, the labels to the near-end node and the adjacency sid of the link are pre-pended, adjusting for PHP cases.

It will be appreciated that some embodiments described herein may include one or more generic or specialized processors ("one or more processors") such as microprocessors, digital signal processors, customized processors, and Field-Programmable Gate Arrays (FPGAs) and unique stored program instructions (including both software and firmware) that control the one or more processors to implement, in conjunction with certain non-processor circuits, some, most, or all of the functions of the methods and/or systems described herein. Alternatively, some or all functions may be implemented by a state

machine that has no stored program instructions, or in one or more Application-Specific Integrated Circuits (ASICs), in which each function or some combinations of certain of the functions are implemented as custom logic. Of course, a combination of the aforementioned approaches may be used. Moreover, some embodiments may be implemented as a non-transitory computer-readable storage medium having computer-readable code stored thereon for programming a computer, server, appliance, device, etc. each of which may include a processor to perform methods as described and claimed herein. Examples of such computer-readable storage mediums include, but are not limited to, a hard disk, an optical storage device, a magnetic storage device, a ROM (Read Only Memory), a PROM (Programmable Read-Only Memory), an EPROM (Erasable Programmable Read-Only Memory), an EEPROM (Electrically Erasable Programmable Read-Only Memory), Flash memory, and the like. When stored in the non-transitory computer-readable medium, the software can include instructions executable by a processor that, in response to such execution, cause a processor or any other circuitry to perform a set of operations, steps, methods, processes, algorithms, etc.

Although the present disclosure has been illustrated and described herein with reference to preferred embodiments and specific examples thereof, it will be readily apparent to those of ordinary skill in the art that other embodiments and examples may perform similar functions and/or achieve like results. All such equivalent embodiments and examples are within the spirit and scope of the present disclosure.