ISSN 2320 - 2602
Volume 2, No.11, November 2013
International Journal of Advances in Computer Science and Technology
Available Online at http://warse.org/pdfs/2013/ijacst022112013.pdf

Link Independent and Node Independent Algorithms for Efficient Multipath Routing

K.ASWANI, G.MURALI
JNTUACEP, Pulivendula, India, areddy815@gmail.com
Asst prof. JNTUACEP, Pulivendula, India, muralig521@gmail.com

ABSTRACT

Link-independent and node-independent DAGs satisfy the property that any path from a source to the root on one DAG is link-disjoint (node-disjoint) with any path from the source to the root on the other DAG. To achieve resilient multipath routing, first introduce the concept of independent directed acyclic graphs (IDAGs). For a given a network, we develop polynomial- time algorithms to compute link independent and node-independent DAGs. The algorithm developed in this paper is provides multipath routing, utilizes all possible edges, guarantees recovery from single link failure and achieves all these with at most one bit per packet as overhead when routing is based on destination address and incoming edge.

Keywords— Directed acyclic graphs (DAGs), failure recovery, independent trees, IP fast rerouting, multipath routing, network protection**.**

1. INTRODUCTION

The increasing use of streaming multimedia and voice-over-IP, precipitated by decreasing cost of handheld multimedia devices and net books, necessitates increased bandwidth provisioning and fast recovery from network failures. Thus, present-day IP networks employ several different strategies for improved end-to-end bandwidth and load balancing (using multipath routing) and fast recovery from link and node failures (using fast rerouting strategies). With the multipath routing, we can achieve bandwidth aggregation [1] by splitting data to the same destination into multiple streams, each routed through a different path, the effective bandwidth can be aggregated, congestion reduction [2], load balancing [3], security [4], and robustness [5] compared to the single shortest-path routing that is usually used in most networks.

Multipath routing in today's IP networks is merely limited to equal-cost multi paths Techniques developed for multipath routing are often based on employing multiple spanning trees or directed acyclic graphs (DAGs) [6]. When multiple routing tables are employed, a packet has to carry in its header the routing table to be used for forwarding. When the corresponding forwarding edge is not available, the packet needs to be dropped. This dropping is forced due to the potential looping of packets when transferred from one routing table to another. In the case of DAGs, computed by adding edges to the shortest-path tree, one cannot guarantee that a single-link failure will not disconnect one or more nodes from the destination.

Techniques developed for fast recovery from single-link failures provide more than one forwarding edge to route a packet to a destination. The techniques may be classified depending on the nature in which the backup edges are employed. The authors develop a method to augment any given tree rooted at a destination with "backup forwarding ports." We present two fast rerouting algorithms to achieve recovery from single-link and single-node failures, respectively. The idea is to calculated backup paths in advance. When a failure is detected, the affected packets are immediately forwarded through backup paths to shorten the service disruption. In [8], the authors present a framework for IP fast reroute detailing three candidate solutions for IP fast reroute that have all gained considerable attention. These are multiple routing configurations (MRCs) [7] to assure fast recovery from link and node failures in IP networks, failure insensitive routing (FIR) [9] to improve failure resiliency without jeopardizing routing stability , and tunneling using Not-via addresses (Not-via) [10]. The common feature of all these approaches is that they employ multiple routing tables. One approach to reduce the number of routing table entries for multipath forwarding is to construct two trees, namely red and blue, rooted at a destination node such that the paths from a source to the destination on the two trees are link/node- disjoint [11]. In this approach, two trees are constructed per destination node such that the paths from any node to the root on the two trees are disjoint. The trees may be constructed to obtain link-disjoint or node-disjoint paths if the network is two-edge or two-vertex connected, respectively. This approach is similar to those employing multiple routing tables, except that only two tables are required. Every packet may carry an extra bit in its header to indicate the tree to be used for routing. This overhead bit may be avoided by employing a routing based on the destination address and the incoming edge over which the packet was received, as every incoming edge will be present on exactly one of the trees. The colored tree approach allows every node to split its traffic between the two trees, thus offering disjoint multipath routing. In addition, when a forwarding link on a tree fails, the packet may be switched to the other tree. A packet may be transferred from one tree to another at most once as the colored tree approach is guaranteed to recover from only a single-link failure. The colored trees are

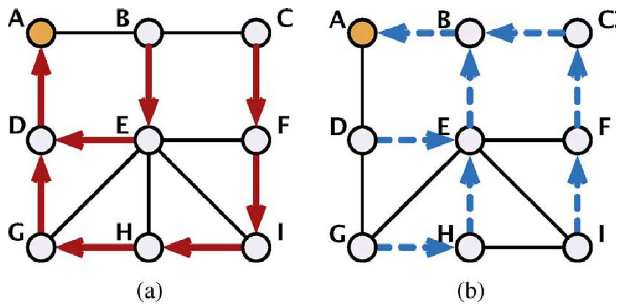

Figure 1: Illustration of node- independent trees for the example network. (a) Red tree. (b) Blue tree. Node A is the root (destination) node.

also referred to as "independent trees" in the literature [12].We will refer to the colored trees approach as the independent trees (ITrees) approach in the rest of this paper. Fig. 1 shows an example network where red and blue trees, rooted at node A, are constructed. This tree construction enables recovery from a single-link failure by switching from one tree to another. For example, consider a packet that is forwarded from node F to node A on the blue tree. When there are no failures, the packet would take the path F–C–B–A. If link C–B fails, then node C would reroute the packet on the red tree, thus the packet will follow the path F–C–F–I–H–G–D–A. Assume that a second link failure occurs on link I–H. As only two independent trees were constructed and recovery from arbitrary two link failures cannot be guaranteed, the packet will be dropped when the second link failure is encountered. One approach to enhance the robustness is to allow the packet to be switched multiple times between the trees. Such an approach will fail in the example considered above. The packet will be rerouted back and forth on the path I–F–C. We may analyze when switching back to a tree would guarantee not encountering a previous failure again by observing the properties of the independent tree construction process. However, the inherent limitation of the tree-based approach is that it utilizes only directed edges to route to a destination, where denotes the number of nodes in the network. The goal is therefore to utilize the additional links available in the network to improve robustness. To this end, we seek to construct independent directed acyclic graphs rooted at each node. Fig. 2(a) and (b) shows two independent directed acyclic graphs rooted at node A. Observe that node I has two red forwarding edges available. Thus, in the earlier example, if link I–H fails, the packet may be forwarded on link I–E to reach the destination.

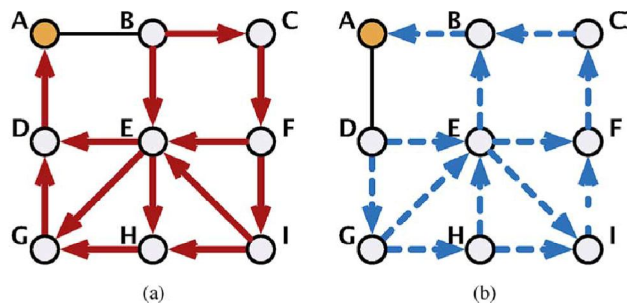

Figure 2: Illustration of node-independent DAGs in an example network where node A is the root (destination) node. (a) Red DAG. (b) Blue DAG.

2.INDEPENDENT DIRECTED ACYCLIC GRAPHS

Here consider a network with a set of nodes and links denoted by N and L , respectively. Here assume that links are bidirectional in nature, which may be realized using two unidirectional links. Here denote a bidirectional link between nodes i and j as, i to j while the directed link from i to j is denoted by i->j . When a link i->j fails, we assume that both directed edges i->j and j->i have failed. Here say that a DAG is rooted at d if d is the only node in the DAG that has no outgoing edges. Every other node has at least one outgoing edge. If we traverse a sequence of edges starting from any node, the path will terminate at node d and will be loop-free. Consider two directed acyclic graphs that are rooted at d. The two DAGs are said to be link-independent if for every node s any path from s to d to on one DAG is link disjoint with any path from s to d to on the other DAG. Similarly, the two DAGs are said to be node-independent if for every node s any path from s to d on one DAG is node-disjoint with any path from s to d on the other DAG.

2.1 Resilient Routing With IDAGs

The network is assumed to employ link-state protocol, hence every node has the view of the entire network topology. Every node computes two DAGs, namely red(R) and blue (B), for each destination and maintains one or more forwarding entries per destination per DAG. The DAGs may be used in two different ways to achieve resilient routing. In the first approach, referred to as Red DAG first (RDF), the packets are assumed to be forwarded on the red DAG first. When no forwarding edges are available on the red DAG, the packet is transferred to the blue DAG. When no blue forwarding edges are available, the packet is dropped. The DAG to be employed for routing is carried in an overhead bit (DAG bit) in every packet header. In the second approach, referred to as Any DAG first (ADF), a packet may be transmitted by the source on the red or blue DAG. In addition to the DAG bit, every packet also carries an additional bit that indicates whether the packet has been transferred from one DAG to another (Transfer bit). A packet is routed on the DAG indicated in its packet header. If no forwarding edges are available in that DAG and if the packet has not encountered a DAG transfer already, it is transferred to the other DAG. If no forwarding edges are available on the DAG indicated in the packet header and the packet has already encountered a DAG transfer, the packet is dropped. In both of the approaches described above, a node may forward the packet along any of the available forwarding edges in the DAG indicated in the packet header. Note that if the red and blue DAGs are (link- or node-) independent, then the network is guaranteed to recover from a single (-link or -node) failure when the packet is transferred from one DAG to the other. In addition, the network may tolerate multiple failures as some nodes may have many forwarding entries in each DAG.

III. CONSTRUCTING NODE-INDEPENDENT DAGS

Two-vertex-connectivity is the necessary and sufficient requirement for constructing two node-independent DAGs utilizing all the edges except those emanating from the given destination node. This necessary part of the requirement follows directly from the condition required for constructing two node-independent trees a special case of DAG.

Initialize the partial order for the nodes on the two DAGs. Compute the first cycle to be augmented. Compute successive paths to be augmented. The path starts and ends at distinct nodes that are already added to the DAGs, hence the paths from every node to the root of the DAG are node-disjoint. Note that the difference between the path augmentation employed for DAG construction here and that employed for tree construction.

Procedure NI-DAGs Construction

1) Initialize $\mathcal{R}$ and $\mathcal{B}$ to contain only the root node d. Initialize the partial order of the nodes on the red and blue DAGs to be the empty set.
2) Find a cycle $(d, v_1, ..., v_k, d)$. Let $v_k \to v_{k-1} \to ... \to v_1 \to d$ be the *red chain* and $v_1 \to v_2 \to ... \to v_k \to d$ be the *blue chain*. Add the blue chain to $\mathcal{B}$ and the red chain to $\mathcal{R}$. Update the precedence relation with $d \prec v_1 \prec v_2 \prec ... \prec v_k$ on the red DAG.
3) Find a path $(x, v_1, ..., v_k, y)$ that connects any two distinct nodes x and y on $\mathcal{R}$ and any k nodes not on $\mathcal{R}$, $k \geq 1$, such that $x \prec y$ on $\mathcal{R}$ or there does not exist an order between x and y on $\mathcal{R}$. Let $y \to v_k \to v_{k-1} \to ... \to v_1 \to x$ be the red chain and $x \to v_1 \to v_2 \to ... \to v_k \to y$ be the blue chain. Add the blue chain to $\mathcal{B}$ and the red chain to $\mathcal{R}$. Update the precedence relation with $x \prec v_1 \prec v_2 \prec ... \prec v_k \prec y$ on the red DAG. Note that: (i) if $y = d$, then $v_k \prec y$ is ignored; and (ii) if $y = d$ and/or $x = d$, $y \to v_k$ and/or $x \to v_1$ in the red and blue chains above, respectively.
4) If $\mathcal{B}$ does not span all the nodes in $\mathcal{G}$, goto step 3.
5) Compute a global order that is consistent with the partial on the red DAG. We denote x precedes y in the global order as $x \prec_g y$.
6) For every link i–j $(i \neq d, j \neq d)$ that is not on the DAGs:
   a) If $i \prec_g j$, then add $i \leftarrow j$ to the red DAG and $i \to j$ to the blue DAG.
   b) Otherwise, add $i \leftarrow j$ to the blue DAG and $i \to j$ to to the red DAG.
7) For every edge $i \to d$ that is not on the DAGs, add $i \to d$ either to the red or the blue DAG randomly.

Figure3: Procedure to construct two node-independent DAGs rooted at destination in a two-vertex-connected network.

Theorem: TheprocedureNIDAGsconstruction:1)assigns every edge other than the edges emanating from the destination to one of the two DAGs; and 2) any path from a source to the root in the red DAG is node-disjoint with any path from the source to the root in the blue DAG.

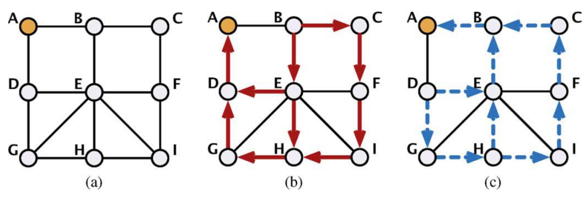

Figure 4: Illustration of the construction of node independent DAGs. (a) Example network. (b) Base red DAG. (c) Base blue DAG. Fig. 2(a) and (b) shows the final red and blue DAGs, respectively.

Example*:* Consider the nine-node network shown in Fig. 4(a), where we seek to compute IDAGs rooted at node A. The base DAGs are computed by considering the cycle A–D–E–B followed by paths D–G–H–E and H–I–F–C–B for augmentation and are shown in Figure 4(b) and (c).

4. CONSTRUCTING LINK INDEPENDENT DAGS

Two-edge connectivity is a necessary and sufficient condition for constructing two link-independent DAGs. Similar to the requirement of node-independent DAGs, the necessary part of the requirement follows from the independent tree construction. We show the sufficiency part of the requirement by constructing the desired DAGs. Fig. 5 shows the procedure to construct two link independent DAGs.

Procedure LI-DAGs Construction

1) Divide the network into two-vertex-connected (2V) components.
2) In each 2V-component, identify the unique articulation node through which every path from any node in that component must traverse to reach d. We refer to this articulation node as the root node of the component. In the component that contains node d, we assume that the root node of the component is node d itself.
3) In each 2V-component, construct two node-independent DAGs to the root node of that component.
4) Merge all the node-independent DAGs to obtain the link-independent DAGs.

Figure 5: Procedure to construct two link-independent DAGs rooted at destination in a two-edge-connected network.

5. ALTERNATIVE APPROACH TO IDAG CONSTRUCTION USING GRAPH EXPANSION

Now present an alternative algorithm to construct IDAGs such that Steps 5–7 of the NI-DAGs construction may be completely avoided. Observe that Steps 5–7 in the NI-DAGs construction are required because not all links are considered by the base IDAG construction (Steps 1–4). If, however, we can modify the graph such that the base IDAG construction would consider all links, then Steps 5–7 can be eliminated. To this end, we develop a graph expansion technique that results in an expanded graph G with N+L nodes and 2L links, where *N* and *L* denote the number of nodes and links in the original graph.

Procedure to Construct Colored DAGs through Gr
Expansion

1) Initialize the red and blue DAG, $\mathcal{R}$ and $\mathcal{B}$, to con
   only the root node d.
2) In the original network $\mathcal{G} = (\mathcal{N}, \mathcal{L})$ replace every lir
   connecting nodes x and y with a node v_ℓ and two li
   (v_ℓ, x) and (v_ℓ, y). Let the resultant expanded netw
   be denoted by $\mathcal{G}'(\mathcal{N}', \mathcal{L}')$.
3) In the expanded graph $\mathcal{G}'$, construct two colored t
   DAGs (using [16]).
4) Contract each pair of links v_ℓ–x, v_ℓ–y in the expan
   graph back to the original graph with red and blue cc
   assigned in each direction appropriately.

Figure6: Procedure to construct two node-independent DAGs rooted at destination using virtual graph expansion.

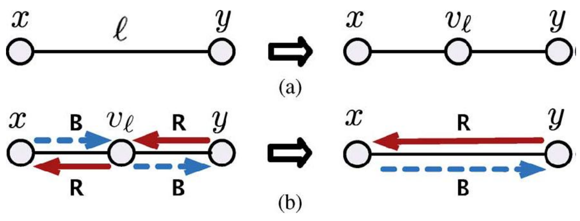

Figure 7: Illustration of the construction of node-independent DAGs using a virtual node. (a) Expansion with virtual node. (b) Contraction to remove virtual node

Figure 6 shows the procedure to construct two node independent DAGs rooted at destination using graph expansion. Consider a link *l € L* that connects nodes x and y in the original graph. We replace this link with v*l* a node and two links v*l* –x and v*l* - *y*. Fig. 7(a) shows the expansion of link . Since every link in in the expanded graph is the original graph results in a new node, the total number of nodes L+N . Since every link in the original graph results in two links in the expanded graph, the total number of links in the expanded graph is 2L .Observe that all the nodes that correspond to a link in the original graph have exactly two outgoing links. Now, consider the base IDAG construction on the expanded the graph. Since Steps 1–4 must account for all nodes being added, all the nodes corresponding to the links in the original graph must be added in steps 1–4, thus all links in the original graph are considered.

Procedure for Multiple Colored Tree Pairs Construction

1) Initialize the link usage frequencies to zero.
2) Consider network links in the descending order of their usage frequency. For each link:
   a) If the graph remains two-vertex connected without it then remove it from the graph.
3) Construct two independent trees on the two-vertex connected graph of step 2.
4) For each link used in the independent trees increment its link usage frequency.
5) If the number of colored tree pairs is less than M go to step 2.

Figure 8: Procedure to construct multiple pairs of colored trees rooted at node utilizing the maximum number of links.

6. PROCEDURE FOR CONSTRUCTION OF MULTIPLE RED–BLUE TREES

Our key goal in this paper is to use the maximum possible number of network edges for data transmission. As mentioned earlier, one possible solution is to construct multiple pairs of colored trees. We would like to compare our IDAGs approach to the multiple pairs of colored trees approach. In this section, we introduce the procedure for constructing multiple pairs of colored trees. In order to use the maximum number of edges, we construct multiple colored tree pairs that share as few edges as possible. Note that for a given pair, the red and blue trees are independent, however trees from different pairs are not necessarily independent. Let the number of pairs of colored trees needed be. It is necessary and sufficient for a network to be two-vertex (edge) connected to compute atleast one pair of vertex(edge)independent trees . Given a network we construct a pair of independent trees using the procedure described record the link usage frequency of all links and sort links in the descending order of usage frequency. We then consider links in the given order, removing a link while the network remains two-vertex (edge) connected and compute a pair of independent trees. We repeat this procedure until the desired number of colored tree pairs is obtained.4 Fig. 8 shows the procedure to construct multiple pairs of colored trees rooted at node. We provide several simulation results of the multiple pairs of colored trees technique in Section.

7. CONCLUSION

The concept of independent directed acyclic graphs (IDAGs) is introduced and developed a methodology for resilient multipath routing using two IDAGs.The polynomial time algorithms to construct node-independent and link-independent DAGs using all possible edges in the network. This algorithm provides effective multipath routing and also recovers single link failures. In addition, the trees based on the shortest paths on the IDAGs have better performance than that of the ITrees approach since the average shortest path length on the IDAGs is shorter than the average path length on the ITrees.

REFERENCES

1. J. Tsai and T. Moors, "**A review of multipath routing protocol : From wireless ad hoc to mesh networks,"** in *Proc. ACoRN Early Career Res. Workshop Wireless Multihop Netw.*, Jul. 17–18, 2006, pp. 17–22

2. S. Murthy and J. Garcia-Luna-Aceves, **"Congestion-oriented shortest multipath routing**, " in *Proc. IEEE INFOCOM*, Mar. 1996, vol. 3, pp.1028–1036

3. P. P. Pham and S. Perreau, "**Performance analysis of reactive shortest path and multi-path routing mechanism with load balance**," in *Proc. IEEE INFOCOM*, 2003, pp. 251–259.

4. W. Lou, W. Liu, and Y. Fang, "**A simulation study of security performance using multipath routing in ad hoc networks**," in *Proc. IEEE Veh. Technol. Conf.*, Oct. 2003, vol. 3, pp. 2142–2146

5. Z. Ye, S. V. Krishnamurthy, and S. K. Tripathi, "**A framework for reliable routing in mobile ad hoc networks**," in *Proc. IEEE INFOCOM*, Apr. 2003, pp. 270–280

6. G. Lee and J. Choi, "**A survey of multipath routing for traffic engineering**," 2002 [Online]. Available: http://academic.research. microsoft.com/Publication/10842993/a-survey-of-multipath- routing-for-traffic-engineering

7. A. Kvalbein, A. F. Hansen, T. Ĉiĉić, S. Gjessing, and O. Lysne, "**Fast IP network recovery using multiple routing configurations**," in *Proc. IEEE INFOCOM*, Apr. 2006, pp. 1–11.

8. M. Shand and S. Bryant, "**IP fast reroute framework**" IETF Internet Draft draft-ietf-rtgwg-ipfrr-framework-08.txt, Feb. 2008

9. S. Lee, Y. Yu, S. Nelakuditi, Z.-L. Zhang, and C.-N. Chuah, "**Proactive vs. reactive approaches to failure resilient routing**," in *Proc. IEEE INFOCOM*, Mar. 2004, vol. 1, pp. 176–186.

10. S. Bryant, M. Shand, and S. Previdi, "**IP fast reroute using not-via addresses,**" Internet Draft draft-ietf-rtgwg-ipfrr-notvia-addresses-02.txt, Feb. 2008.

11. G. Jayavelu, S. Ramasubramanian, and O. Younis, "**Maintaining colored trees for disjoint multipath routing under node failures**," *IEEE/ACM Trans. Netw.*, vol. 17, no. 1, pp. 346–359, Feb. 2008.

12. A. Huck, "**Independent trees in graphs**," *Graphs Combin.*, vol. 10, pp.