Fault-Tolerant Dynamic Routing Based on Maximum Flow Evaluation

Jonatan Schroeder and Elias P. Duarte Jr.

Federal University of Paraná (UFPR)
Dept. of Informatics – P.O. Box 19018
81531-990 Curitiba - Brazil
{jonatan,elias}@inf.ufpr.br

Abstract. This work proposes a fault-tolerant dynamic routing algorithm that employs maximum flow evaluation for route selection, increasing the number of disjoint paths to the destination, enhancing the path redundancy, and so extending the possibility of using detours, or alternative paths if needed. Route distance is employed as a secondary criterion. Routes may be dynamically changed by intermediate routers, which usually have more recent information about topology changes. Formal proofs for correctness of the algorithm are also presented. The proposed algorithm was implemented in a simulation environment and experimental results are presented.

1 Introduction

Most routing algorithms employ the distance in number of hops as the main, if not the only, criterion for selecting routes. Nevertheless, for critical applications a robust route is not necessarily the shortest route.

This work proposes a novel approach for route selection based on route *robustness*. A robust route improves the probability that if faults occur along the route it is feasible and efficient to find another route, or *detour*, to the destination. A robust route improves the ability of finding detours to the destination in case of faults.

The proposed routing algorithm chooses an edge for the route of a given destination using maximum flow evaluation. This evaluation is employed in order to increase the number of disjoint paths to the destination, enhancing the path redundancy, and so improving the probability of quickly finding short detours, or alternative paths, after faults are detected. Route distance is employed as a secondary metric for route selection.

The proposed routing algorithm is dynamic in the sense that each node along the route only selects the next hop of the route, based on its current information. In other words, no route is pre-selected at the source, so that intermediate routers dynamically determine the route as they process the packet. This behavior exploits the fact that topology changes are quickly discovered by nodes that are closer to the change itself. This concept can be also extended to traffic information, i.e. information about congested or heavily/lightly used links.

The dynamicity of the algorithm has a potential impact on the behavior of routing during the *convergence latency* interval. Routing protocols present a convergence latency after the network topology changes [1] due to router or link failures. During the convergence latency interval all routing tables are updated in order to compute the new paths to be employed. For some protocols the convergence latency is quite large. For instance, the average latency for the Internet's BGP, (*Border Gateway Protocol*), is about 3 minutes, but intervals of up to 15 minutes [2] have also been reported. During the convergence latency interval, packets may be lost, and connections may be broken.

The proposed algorithm does not require that routers be initialized with the complete network topology, routers are initialized only with information about their neighbors. Nodes keep a local topology representation that is updated with periodic messages exchanged with neighbors. The algorithm is able to successfully route messages even when the local representation of the topology is out-of-date, or does not represent the complete network topology.

Figure 1 shows an example network topology and the route selected by the proposed algorithm to send a packet from s to t. Initially s chooses to send the packet through node a, from which there are two edge-disjoint paths to reach the destination t. As from node b there is only one available path, the edge to node a receives a better evaluation by the routing algorithm. Now, node a sends the packet to node e, because in comparison to node c both have the same number of edge-disjoint paths but the distance from e to t is shorter. However consider that the link from node e to node t has failed, and so far only node e has this information. Executing the algorithm, node e sends the packet back to node a, which then forwards the packet through c and d to the destination t. Details about the criteria employed and the formal specification of the algorithm are presented in section 2.

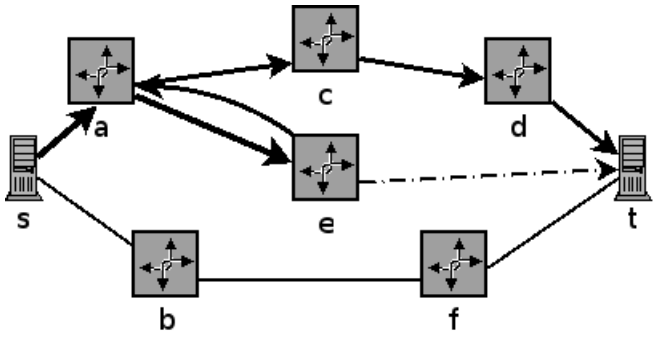


Fig. 1: An example topology and the route chosen by the algorithm.

The algorithm was implemented, and the implementation is available on the Internet, created with Java [3]. Implementation details, as well as experimental

results obtained with Internet-like networks on a simulation environment, are presented in section 4. A running applet is available at `http://www.inf.ufpr.br/jonatan/mfrp`.

This work is organized as follows. Section 2 presents the algorithm specification. Section 3 presents the proofs of correctness of the algorithm. Section 4 presents experimental results. Section 5 points to related work. Finally, section 6 concludes the paper.

2 The Proposed Algorithm

The proposed routing algorithm is executed for each packet that is sent from a given source to a given destination. The algorithm is run initially by the source node, which chooses the next node of the route, among its neighbors. When the packet arrives at the next node, this node runs the algorithm to choose the following node, and so on, until the packet reaches the destination.

The approach used by this algorithm is similar to that of Bellman-Ford algorithm [1], in the sense that each node does not need a previous knowledge of the topology. A node running the algorithm chooses the next node of the route, instead of the complete route. Furthermore, the algorithm works even if the most recent topology changes are not known by all nodes.

The proposed algorithm chooses the next node of the route through an evaluation of each adjacent edge. This evaluation is based on a trade-off between redundancy and the distance of the paths to the destination from the evaluated edge. After the evaluation, the best edge is chosen.

The formulae and equations for the computation of the metrics used in this work assert that each node has a local representation of the network topology. The topology is learnt and updated by each node through periodic messages exchanged with neighbors, as described below. This topology update messages are sets of tuples <*edge,state,timestamp*>, where *state* is either faulty or fault-free and *timestamp* is a counter of state changes. This timestamp allows nodes to determine whether the information is newer than the one it already has. The local representation, however, does not need to be complete and up-to-date. The topology is represented through a directed graph structure, with a set of vertices, corresponding to the network nodes, and a set of edges, corresponding to the network links.

This work considers crash faults, and the system is considered to be partially synchronous, i.e., there is a finite time limit, not necessarily known, for the delay on the communication between any pair of nodes.

2.1 Algorithm Specification

This section initially presents some preliminary definitions used in the algorithm specification.

A *directed graph* (or *digraph*) G is a pair (V, E) of sets, in which V is a set of nodes (or vertices) and E is a set of edges (or links). Each edge is a pair of exactly two different nodes.

Let $G = (V, E)$ be a digraph; let $c : E \to \Re$ be a function corresponding to the capacity of the digraph edges; let $u, v \in V$ be nodes of the digraph G. A *flow* between u and v is a function $f : E \to \Re$ where:

$$\forall e \in E, f(e) \leq c(e)$$

$$\forall t \in V - \{u, v\}, \sum_{e=(t,t') \in E} f(e) = \sum_{e=(t',t) \in E} f(e)$$

The size (or cardinality) of a flow f, represented as $|f|$, is defined as:

$$|f| = \sum_{e=(u,t) \in E} f(e) - \sum_{e=(t,u) \in E} f(e)$$

A flow f is said to be maximum if, for every flow f' between the same pair of nodes, $|f| \leq |f'|$.

Let $u, v \in V$ be nodes of the graph G. A *cut* between u and v is a set of edges C so that removing all edges in C from the graph G, u and v are not connected. The size (or cardinality) of a cut C, represented as $|C|$, is defined as the cardinality of the set C. A cut C is said to be minimum if, for every cut C' between the same pair of nodes, $|C| \leq |C'|$. For every pair of nodes of the network, the maximum flow and the minimum cut have the same cardinality, and are computed with the same algorithms [4]. So, this work will use both terms interchangeably.

Figure 2 shows the algorithm that is run when node n has to route a packet to a given destination. $\Gamma(G, e)$ is an evaluation function, specified below.

The evaluation of the edges to be used in the routing process is computed on a subgraph of the digraph that corresponds to the local representation of the network topology. This subgraph is obtained with the removal of the nodes already visited by the packet, and of the edges that are adjacent to these nodes. This strategy guarantees there will not be loops.

There is a specific case in which edges that lead to loops are used. Considering that topology information is not necessarily up-to-date in all nodes, a node can choose an edge based on a path that is not available anymore, or is faulty. When a packet reaches a node that already has up-to-date information, it is possible that the only available paths to the destination pass through nodes already visited. In this case, the packet needs to be delivered back to the node from which it came. However, the node that receives this packet probably has out-of-date information about the topology, since its selection led to a node without available route options. So, in order to make it possible for this node to have up-to-date information, the update message programmed to be sent to this node is anticipated, and is sent before the packet is sent back to the node. This way, the new node can take the decision for a new path based on more recently updated information about the topology, and so avoiding paths with no routing options.

In order to make the information about the availability of the routes reach all nodes in the network, a topology update process is run on each node. This process is run every α seconds, where α is a parameter of the algorithm. Each node sends

1. Add node n to the list of visited nodes of the message.
2. If there is an edge from node n to the destination, send the message through this edge and finish.
3. Create an auxiliary graph G', corresponding to the known topology, removing the visited nodes of the message.
4. Evaluate each adjacent edge of node n, using function $\Gamma(G, e)$ in graph G'. Edges reaching visited nodes are not evaluated, as well as edges without available paths to the destination.
5. If at least one edge was evaluated, send the message through edge e with the largest $\Gamma(G, e)$.
6. If no edge could be evaluated:
   (a) Remove n from the list of visited nodes.
   (b) If n is the source of the message, return an error.
   (c) If n is an intermediate node, send an update message, followed by the routed message, to the last node on the list of visited nodes.

Fig. 2: The algorithm for choosing the next edge.

for all its neighbors recently learnt information about topology changes. When a node receives a packet through an edge that was considered faulty, the node adds the edge to its local representation of the topology. If an update message is expected and is not received through an edge after a timeout, the edge is considered faulty and is removed from the local representation of the network topology. This timeout is called β ($\beta > \alpha$), and is also a parameter of the algorithm.

After receiving an update message, each node replies with an acknowledgement. After this acknowledgement is received, all information is marked so that the sender does not need to resend the information again in the next update message.

2.2 Edge Evaluation: Path Redundancy & Distance

The computation of the edge evaluation function $\Gamma(G, e)$ is based on a set of quantitative criteria related to the redundancy and to the size of the paths that pass through the evaluated edge. For each of these criteria, a weight is associated, so that the computation can be adapted to the priority given to each criterion. Each weight is a parameter of the algorithm.

The following formula corresponds to the computation of the edge evaluation:

$$\Gamma(G, e) = \sum_{c_n \in C} \omega_n \times c_n(e) \tag{1}$$

In this equation, $\Gamma(G, e)$ is the evaluation function, e is the edge being evaluated, C is the set of criteria (described below) and ω_n is the weight associated to criterion c_n.

The criteria used for edge evaluation are functions that receive a graph representing the topology as input and an edge, and return a numeric value. This work uses two criteria: the cardinality of the maximum flow (or the minimum cut) from the node adjacent to the evaluated edge to the destination node (c_1) and the length of the shortest path between these nodes (c_2).

The main criterion used in this work for the evaluation of the edges for routing is the maximum flow (or minimum cut) from the node adjacent to the evaluated edge to the destination. This criterion is called c_1. A classic algorithm for the computation of the maximum flow is the Ford and Fulkerson algorithm [4, 5]. This algorithm uses an edge-valued graph structure, in which, for a graph $G = (V, E)$ there is a function $c : E \to \Re$, that associates a value for each edge, corresponding to the capacity of the edge. Our algorithm assumes that $c(e) = 1$ for all edges. The complexity of this algorithm, when the capacity is an integer, is $O(NM)$, where N is the number of nodes and M is the number of edges of the graph [5]. For the weight associated with this criterion (ω_1) a positive value is used, since the evaluation of an edge is intended to be proportional to the result of this function.

In order to select routes that are not only robust but also short, one of the criterion used for the evaluation of an edge is the minimum distance from the node adjacent to the evaluated edge to the destination. This criterion is called c_2. As this work considers that all edge capacities or costs are equal to one, a breadth-first search on the graph, in which the number of rounds, or levels, passed through to find the destination node, starting at the evaluated node, is used as the result of the criterion. The breadth-first search is run in $O(M)$ steps, where M is the number of edges in the network [6, 5]. For the weight associated to this criterion (ω_2) a negative value is used, since a shorter distance is better for evaluation of the edge.

2.3 Example Executions

Figure 3 shows an example execution of the algorithm. In this figure, suppose node s has to send a packet to node t, The routing algorithm is run on s, evaluating all adjacent edges, i.e., (s, a), (s, b) and (s, c), and choosing the one that has the best evaluation and that will be used for the routing. Suppose the chosen edge is (s, b), the packet is then sent from s to b through this edge.

When node b receives the packet sent by s to t, it evaluates edges (b, d) and (b, e). Edge (b, s) is not evaluated, since node s has already been visited and, so, is removed from the graph used for the evaluation. Suppose the chosen edge is (b, e). The packet is then sent to node e. Similarly e evaluates its neighbor edges, discarding (e, b) and choosing (e, h). Finally, node h evaluates its adjacent edges. As it has an edge that goes directly to the final destination t, this edge is used. So, the packet is sent to node t through edge (h, t), achieving the final destination.

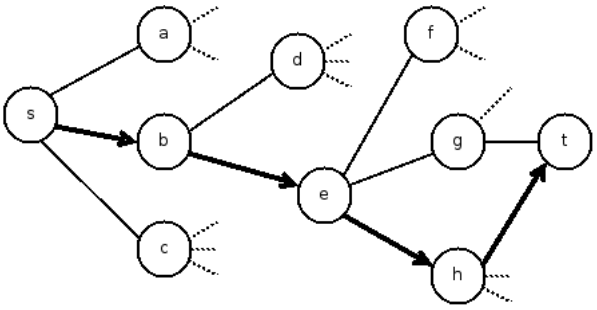


Fig. 3: An example execution.

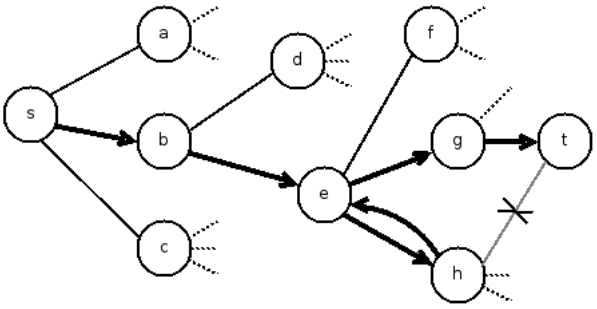


Fig. 4: Rerouting after an edge becomes faulty.

Consider another example. Suppose that, in the same network described above, a fault occurs on edge (h, t), that is used on the path between s and t. Figure 4 shows the resulting network after this fault. Suppose, again, that s has a packet to send to t, immediately after the fault occurs. Suppose that, when the packet is sent, only nodes h and t (adjacent to the faulty edge) have information about the occurrence of the fault. The algorithm is run by node s, which sends the packet to b, that sends the packet to node e, that sends the packet to node h. This procedure is executed in the same way it was in the previous example, since these nodes have not had their topology information changed.

When node h receives the packet to be sent to node t, it finds out that the only possible paths leading to t pass through nodes already visited by the packet, such as node e. Since there is no alternative route for sending the packet, node h sends a topology update message to node e. As node e receives this message, it will be able to take a decision based on more recent topology information. Finally, after sending the update message, node h sends the original packet back to e. When node e receives the original packet, it will take a new decision about the edge that should be used, now considering that the edge (h, t) is not employable. So, another edge is chosen, for example, edge (e, g), sending the packet to node g. This node sends the packet to node t, using the edge (g, t), and the packet achieves its destination.

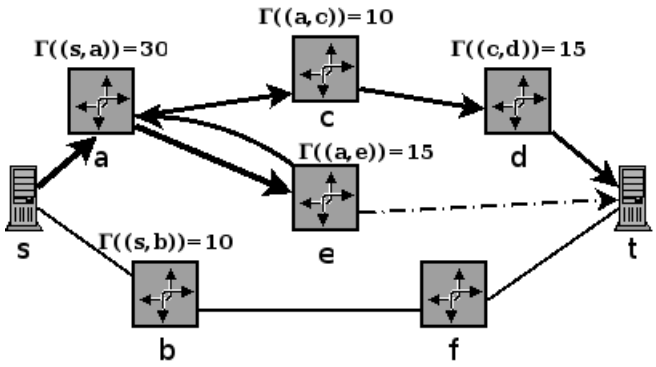


Fig. 5: An example execution with Γ evaluation.

Another example execution is shown in figure 5. The network shown in this figure is the same as that shown in figure 1. In this example we show the evaluation function $\Gamma(G, e)$ computed for the selection of the edges. Suppose the weights for the criteria described above, ω_1 and ω_2, are respectively 20 and -5. Suppose node s has a packet to send to node t, and edge (e, t) is faulty, but only edges e and t have this information at this time.

Initially, node s evaluates all its adjacent edges, i.e. edges (s, a) and (s, b). Both evaluations are made in a subgraph G', that is equivalent to the original topology representation in node s, except for node s itself. Note that edge (e, t), even faulty, is still in graph G', since node s does not have the information that it is faulty. In graph G', the maximum flow from node a to node t is equal to two, and from node b to node t is equal to one, so $c_1((s, a)) = 2$ and $c_1((s, b)) = 1$. The minimum path from node a to node t is $a - e - t$, with distance 2. The minimum path from node b to node t is $b - f - t$, also with distance 2. So, $c_2((s, a)) = c_2((s, b)) = 2$. Applying these values to the $\Gamma(e)$ equation, we have: $\Gamma((s, a)) = 30$ and $\Gamma((s, b)) = 10$, and thus edge (s, a) is selected.

When the packet arrives at node a, it evaluates edges (a, c) and (a, e). Edge (a, s) is ignored, since node s was already visited. Now the evaluation is made on a subgraph G'', similar to the original topology representation in node a, but removing nodes s and a, already visited. Note that edge $e - t$, even faulty, is still in subgraph G'', since node a does not have the information it is faulty. In subgraph G'', the maximum flow from both nodes c and e to the destination t is equal to 1, so $c_1((a, c)) = c_1((a, e)) = 1$. The minimum path from node c to node t is $c - d - t$, with distance 2. The minimum path from node e to node t is $e - t$, with distance 1. So, $c_2((a, c)) = 2$ and $c_2((a, e)) = 1$. Applying these values to the $\Gamma(e)$ equation, we have: $\Gamma((a, c)) = 10$ and $\Gamma((a, e)) = 15$, and thus edge (a, e) is selected.

Node e, receiving the packet, has no alternatives for routing, as its only non-faulty adjacent edge leads to node a, already visited. So, a topology update message is sent to node a, containing the information about edge (e, t), followed

by the packet itself. When node a receives both the message and the packet, it re-evaluates all its adjacent edges. Edge (a, e) is now ignored, as it does not have any available path to the destination. Edge (a, s) is ignored, and $\Gamma((a, c)) = 10$, as described above, and (a, c) is select.

Now node c receives the packet, and it evaluates the only possible edge (c, d), with $c_1((c, d)) = 1$, $c_2((c, d)) = 2$ and $\Gamma((c, d)) = 15$. The packet is delivered to node d, that has a direct available link to node t, and sends the packet to the destination through this link.

3 Proofs

In this section we present proofs of correctness of the algorithm, as well as proofs for the number and size of the update messages, the edge selection complexity and the latency of the algorithm.

3.1 Correctness

The first proof corresponds to the correctness of the algorithm, i.e., if there is a route between two nodes, the algorithm will route a packet successfully between these two nodes. This proof assumes the following hypotheses. The first hypothesis states that nodes that are adjacent to an edge (or to another node) have correct information about the state of this edge (or node). The second hypothesis states that the source node has at least one non-faulty path to the destination in its local representation, i.e., in the local representation of the topology in the source node at least one available path between source and destination must be fully working; please note that this is not necessarily the selected route. Another hypothesis states there is no topology change from the time the packet leaves the source and the time it arrives at the final destination, or until the source learns that there is no available path to the destination.

Initially a lemma is proved stating that if a packet is sent through all edges with paths in the source's local representation of the network and returns to the source in every path, the source will learn there is no route to the destination and the routing will correctly fail. After that, another lemma proves that, if there is a non-faulty path to the destination through the selected edge, this edge is used in the route. The next lemma proves that if there is one or more paths in the local representation of the network in the source that are not faulty, then an edge that is part of one of these paths is chosen to the route. Finally, using the proved lemmas, the theorem stating the correctness of the algorithm is proven.

Lemma 1. *Consider graph $G = (V, E)$, and two non-faulty nodes $s, t \in V$ of this graph. Consider all nodes have the correct state information of their adjacent edges in their local representations of the topology. If s sends a packet with destination t using the proposed algorithm, and no edge adjacent to s is selected, or if after the message is sent in several tries through all adjacent edges and all return an update message to the source, then s learns there is no available route to the destination.*

Proof. Assume the set of adjacent non-faulty edges of s is the set $\{e_1, e_2, \dots e_n\}$. Edges that do not have available paths leading to the destination are ignored by the algorithm, and so are not included in this set. Proving by induction, assume initially that this set is empty, i.e., s does not have any fault-free adjacent edge. The proof in this case is trivial, since s learns that there is no available route to the destination.

Assume now the lemma is true for set $\{e_1, e_2, \dots e_{n-1}\}$. Assume that the set is $\{e_1, e_2, \dots e_{n-1}, e_n\}$. Assume, without loss of generality, that the chosen edge for routing is edge e_n. So, the packet is sent through edge e_n to the next node, say node u. As there is no path to the destination, the packet has to return to the node from where it came from. One of the hypotheses assumes the packet is sent back through the same edge, e_n, but this happens only after u sends s a message with all topology updates learned by u, considering u has no available route to the destination t. After s receives this update message, it updates its local representation of the topology. When the original packet arrives, a new evaluation will start, and this time edge e_n is ignored, since the received topology information points to the nonexistence of paths to the destination t through edge e_n. So the set of available edges is changed to $\{e_1, e_2, \dots e_{n-1}\}$, for which the lemma is true. So, by induction, the lemma is true, and s learns there is no available route to the destination.

Lemma 2. *Consider $G = (V, E)$ is a graph, and $s, t \in V$ are two non-faulty nodes of this graph. Consider all nodes have the correct state information of their adjacent edges in their local representations of the topology. Consider there is a path from s to t, and that (s, u) is the first edge of this path (u and t may be the same node). If s sends a packet with destination t through the edge (s, u) using the proposed algorithm, the packet will either reach its destination or return to the source s.*

Proof. We prove by induction. First assume graph G has only two nodes. These nodes have to be s and t and, by definition, they are not the same node. The only edge that can be evaluated by node s is edge (s, t), since there is no other possible edge in the graph. So, $t = u$. If this edge is faulty, s has the information about this fault (hypothesis) and so (s, u) cannot be evaluated. However, if the edge is not faulty, then the packet is sent by node s directly to node t through the edge (s, u), arriving node t. When $t = u$ a similar proof can be given.

Assume now that the lemma is true for a graph with $(n - 1)$ nodes. Suppose that G has n nodes. When s sends a packet through the edge (s, u), the hypotheses assert s has the information that this edge is not faulty. Node u, on receiving the packet and after evaluating its adjacent edges, removes node s from the graph used for the evaluation, and so continues the execution on a subgraph containing $(n - 1)$ nodes, for which the lemma is true. So, by induction, the lemma is proven true for every number of nodes.

Lemma 3. *Consider graph $G = (V, E)$, and two non-faulty nodes $s, t \in V$. Consider all nodes have the correct state information of their adjacent edges in their local representations of the topology. Consider s has a available path leading*

to t in its local representation. If s has to send a packet to t, using the proposed algorithm, an edge that belongs to a fully available route is chosen.

Proof. Proving by induction, consider s has only one neighbor, called u. As there is an available path leading to t, this path has to pass through u (which can be t itself). When the edges are evaluated, edge (s, u) is chosen because s has a non-faulty route passing through this edge in its local representation, and it is the only available edge. So, the lemma is true for one neighbor.

Now consider the lemma is true for $(n-1)$ neighbors. Suppose s has n neighbors, called u_1, u_2, $\ldots u_{n-1}$, u_n. Using the proposed algorithm, some edges are going to be discarded, as they do not belong to available paths to the destination, as proven by the previous lemma. Suppose, without loss of generality, (s, u_n) is discarded. So, the algorithm continues with $(n-1)$ neighbors, for which case the lemma is true.

Consider now that no edge is discarded. Without loss of generality, consider the edge evaluated as the best is (s, u_n). The algorithm sends a packet through this edge. If the edge belongs to a available path leading to t, the lemma is proven. If, however, there is no available path leading to t through (s, u_n), the packet returns to node u_n, as proven by lemma 2, and a routing message is sent back to node s with more recent information about the network topology. So, edge (s, u_n) is discarded, as it does not belong to an available to node t. The algorithm continues with $(n-1)$ neighbors, for which it is true. Thus the lemma is proven for any number of neighbors.

Theorem 1. *Consider graph $G = (V, E)$, and two non-faulty nodes $s, t \in V$. Consider all nodes have the correct state information of their adjacent edges in their local representations of the topology. Consider s has at least one available non-faulty path leading to t in its local representation. If s sends a packet to t using the proposed algorithm, the packet is delivered to t.*

Proof. Initially consider there is a non-faulty edge linking s to t. In this case, as s is adjacent to the edge and, so, has the information about its state, the packet is sent from s to t through the edge (s, t), arriving at t, as proposed.

Consider now there is no direct link between s and t, or it exists, but is faulty. According to lemma 3, as s has a non-faulty path leading to destination t in its local representation of the topology, an edge that belongs to a non-faulty route is chosen. According to lemma 2, as the chosen edge leads to a available path, the packet arrives its destination. So, the packet arrives the destination t.

3.2 Number and Size of Update Messages

Theorem 2. *Consider a network that employs the proposed algorithm for routing. The number of topology update messages sent by all nodes is $O(M)$ every α seconds, and each message has $O(M)$ entries.*

Proof. The proposed algorithm employs periodic topology update messages. According to the algorithm specification, each node sends a message every α seconds

for each of its neighbors. If g_v is the degree of node v, i.e. the number of adjacent edges of node v, each node sends g_v messages every α seconds. Thus $\sum_{v \in V} g_v = 2M$ $2M$ messages are sent by all nodes every α seconds, and the number of messages sent is $O(M)$ every α seconds.

The worst case of the message size is when it has information about all links in the network. As each edge is counted twice (once for each adjacent node), the largest possible message has $2M$ entries, so the message size, in the worst case, is $O(M)$ entries.

3.3 Edge Selection Complexity

Theorem 3. *Consider a network that employs the proposed algorithm for routing. The complexity for evaluating and selecting an edge for routing is $O(M^2)$.*

Proof. For the selection of an edge, the proposed algorithm computes $\Gamma(G, e)$ for each of its neighbor edges, i.e. for g_v edges. The $\Gamma(G, e)$ computation includes the computation of two criteria: the criterion c_1, corresponding to the maximum flow, that has a complexity of $O(NM)$ [5]; and the criterion c_2, corresponding to the breadth-first search, with complexity of $O(M)$ [5]. Thus, the complexity for the computation of $\Gamma(e)$ is $O(NM + M) = O(NM)$. As this function is computed for each neighbor edge, the selection of an edge is done in $O(NM\overline{g_v})$ steps. As $N\overline{g_v} = 2M$, the selection is done in $O(M^2)$ steps.

3.4 Latency

In this subsection, we evaluate the time required after a topology change for a message to be properly routed.

Theorem 4. *Consider a network that employs the proposed algorithm for routing. Consider that the source node has an available route to the destination in its local representation of the topology. Consider that the state of an edge toggles. The time elapsed from the occurrence of this event until a packet can be successfully delivered to the destination is β seconds.*

Proof. Assume that there is a route to the destination in the source's local representation of the topology, and that this route does not pass through the edge that toggled state. This can be assumed as the route was assumed to be available. In this case, only the nodes adjacent to the edge have information about the state change. If the edge becomes faulty, each neighbor will learn the state after β seconds without receiving any message through the faulty edge. If the state change is the recovery of a faulty edge, or a new edge that is added to the network, then the neighbors will learn the state change after at most α seconds, when the next topology update message is sent. As $\beta > \alpha$, in this case the latency is β seconds.

Theorem 5. *Consider a network that employs the proposed algorithm for routing. Consider, now, there is no available route to the destination in the source's local representation of the network. Consider that the state of an edge toggles. The time elapsed after the occurrence of this event until a packet can be successfully delivered to the destination is $O(D(G)\alpha)$ seconds, where $D(G)$ is the diameter of the graph representing the network topology.*

Proof. In this case the source node does not have any available path to the destination in its local representation of the network. However, there is a path leading to the destination passing through a faulty link that becomes available. The neighbors of this link learn the state change in at most α seconds. This information has to be sent to the neighbor's neighbors, and so on until the information arrives at the node that has a packet to send. In each step the information takes at most α seconds to arrive at the next node. As each node sends the message to all its neighbors, the message will arrive at the node that needs the information in a number of steps equivalent to the minimum distance between this node and the edge that has its state changed. As the maximum shortest distance of two nodes in a graph is the diameter of the graph, or $D(G)$, the latency of the proposed algorithm for this case is $O(D(G)\alpha)$.

4 Implementation and Experimental Results

A simulator for the proposed routing algorithm was implemented in Java [3], version J2SE (*Java 2 Standard Edition*) 1.4.2. A running applet is available at `http://www.inf.ufpr.br/jonatan/mfrp`.

The implementation was divided in three modules. The first module is the main module, which contains internal procedures, such as those employed for the communication between nodes and edges, the edge evaluation for routing, and the procedures for sending and receiving the topology update messages. The second module is the graphical interface module, used for setting parameters and visualizing the execution of the algorithm; the user can draw the topology, observe the flow of routing information, determine the state of links and nodes, and check selected routes. The third module is the simulation module, employed to obtain experimental results on random graphs with random events.

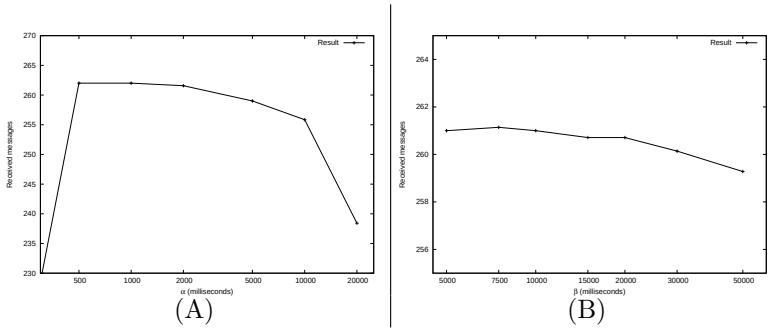
The random graphs were generated using the *Power Law* distribution model [7]. This model is proven to generate topologies that are very similar to those of real networks, such as the Internet [8–10]. The simulation module, mentioned before, implements the algorithm by Bu and Towsley [11] for creating random Power Law graphs.

The experimental results presented in this paper take into account the number of messages that successfully arrive at the destination. For each simulation, several graphs of 100 nodes each were generated. We could not employ larger graphs due to lack of resources for the simulation. For each graph, a set of events was generated. An event corresponds to either the change of the state of an edge, the change of the state of a node state or the creation of a packet for routing. Each event has an associated timestamp.

The graph generator employed the Bu and Towsley algorithm with the following parameters: m_0 (initial backbone size) corresponding to 5 nodes, p (related to the number of edges) equal to 0.6, and β (related to the graph sparsity) equal to 0.2. In each graph random events were scheduled for about 10 minutes. About every 120 seconds a randomly chosen node toggled its state; a random edge state change was scheduled for about every 60 seconds; a packet was generated about every two seconds. In all tests, the simulation was run several times for each graph, using a representative result as the final result for each simulation.

Since for all simulations the same set of events was used, packets were ignored in case they did not arrive at the destination because (1) the source or destination failed or (2) there is no available route between source and destination. Each simulation generated a total of 286 messages for each graph in average, and about 25 messages were discarded for one of the reasons presented above.

In the first simulation experiment a varying time interval between topology update messages, known as α, was employed. The following values were used for α: 10 milliseconds, 0.5 seconds, 1 second, 2 seconds, 5 seconds, 10 seconds and 20 seconds. In all simulations, β was defined as 2α, the *delay* for a message transmission through an edge was 200 milliseconds and the values 20 and -5 were used respectively for the weights of the criteria c_1 and c_2 (ω_1 and ω_2). The results are presented in figure 6, in (A). In this figure, the average number of messages arriving at the destination in all graphs is used as result.



(A) (B)

Fig. 6: Simulation results for a varying α and β.

We can observe through the results of experiment 1 that the algorithm performance is worse for higher values of α. The same occurs for very low values, as 10 milliseconds. In the latter case, only five messages were successfully routed to their destinations. This happens because, for a very low α, the number of messages in the network is higher, causing congestion at some points and the topology update messages get delayed. With a higher value of α, there is a reduction in the number of messages and most topology update messages can be received in time. However, as α increases, the time for a node to receive topology update messages also increases, and the local representation of the topology at

the nodes get out-of-date for longer periods. Thus, for this experiment we can conclude that the best range for α is between 500 milliseconds and two seconds.

Another experiment varied the *timeout* interval for receiving information from a given neighbor, called β. The simulation was run having α equal to 5 seconds, and using the following values for β: 5, 7.5, 10, 15, 20, 30 and 50 seconds. In all simulations, the *delay* for a message to pass through an edge is 200 milliseconds, and the values 20 and -5 were used respectively for the criterion weights ω_1 and ω_2. The result is shown in figure 6, in (B). In this figure, as above, results correspond to the average of the number of packets arriving at the destination when routed with the proposed algorithm.

The results obtained show that there is a very low variation for the value of β (note that the scales are different than the ones of the previous figure), however the result is better for shorter values of β, closer to α. For higher values, the algorithm's latency is increased, and the time for a node to acknowledge the failure of an edge is also increased, and so rising the probability of sending a message through a faulty edge.

5 Related Work

The dependability requirements for modern networks is increasingly high. The convergence latency of current Internet routing protocols, especially BGP is a problem [2, 12]. During this interval packets are lost and connections are broken. This situation can persist even when the physical network is redundant, offering alternative physical routes for communication.

Several approaches have been proposed in order to decrease BGP's convergence latency and/or to avoid its consequences. Recently, Sahoo et al. [13] presented a strategy for adjusting BGP parameters, specially MRAI (*Minimum Route Advertisement Interval*) and for reducing the processing overhead of routers, allowing the reduction of the convergence latency after large-scale failures. A set of tools for monitoring BGP routers, as well as determining relevant events that may result in routing anomalies are presented in [14]. In [15] an alternative BGP version is proposed, with messages for reporting faults that allow new information to be distinguished from old information. The authors show that their strategy avoids part of BGP instability problems.

Another strategy is FRTR (*Fast Routing Table Recovery*), introduced in [16], which was proposed to detect and correct inconsistencies in neighbor routers tables; the paper shows that the original BGP neither detects nor solves several types of inconsistencies. An evaluation of the packet loss rate during the convergence latency is presented in [17]. The authors conclude that increasing the network connectivity causes a decrease of the packet loss rate, and that the ability of the protocol of quickly propagating network state information is also important to reduce the packet loss rate. Two papers [18, 19] propose strategies for path dependence analysis, in order to reduce the number of paths considered during the convergence, and so reducing the convergence latency. None of these papers proposes a solution that solves completely the problem.

An approach for finding robust paths, proposed in [20], is based on the identification of nodes that belong to highly connected components on the network, in order to find paths that pass through these nodes. Two connectivity criteria are defined, named #C(v) and MCC(v), or the connectivity number and the maximum connectivity component of a vertex, respectively. Polynomial algorithms based on Gomory and Hu's cut tree [21] are proposed for computing these criteria. However, this approach is intended to be used for internal routing only, as nodes are required to to maintain a complete and up-to-date topology representation of the network.

Another related work refers to QoS routing, in which besides distance other criteria are taken into account such as bandwidth, jitter and delay. In this case, even if there are no link or node failures, and communication is not totally interrupted, the previously agreed quality of service (QoS) level has to be maintained. When a QoS violation is predicted or detected, *rerouting* is required [22]. The MPLS (*Multi Protocol Label Switching*) protocol [23] is usually employed in this setting. Using MPLS, it is possible to establish virtual circuits that carry flows with specific QoS requirements. When a MPLS router has to change the virtual circuit used for transmitting a flow, rerouting occurs for *QoS restoration* [24].

Several strategies for MPLS rerouting have been proposed. In [25], an architecture is presented based on mobile agents for monitoring virtual circuits and rerouting after a QoS failure tendency is detected. This kind of approach is said to be proactive, as opposed to reactive approaches which cause rerouting only after detecting that the agreed parameters have been broken. In the proactive approach periods when the network does not offer the required QoS levels are avoided; on the other hand, rerouting is some times executed without being necessary, as a QoS fault tendency is not always confirmed. Most proactive QoS restoration techniques are based on back-up routes that are reserved for the flow from the time it is established to the time it is released [26], even when they are not necessary. A comparison between proactive and reactive strategies based on traffic engineering is presented in [27]. Tanaka et al. [28] take into account the physical network technology to evaluate rerouting strategies, considering IP routers and optical devices, such as PXC's (*Photonic Cross Connects*) and DWDM (*Dense Wavelength Division Multiplexing*).

6 Conclusion

This work introduced a new fault-tolerant dynamic routing algorithm. Routes are dynamically selected with maximum flow evaluation and distance. The proposed algorithm does not require that routers be initialized with the complete network topology. Intermediate routers are able to switch the path employed, and this path itself is selected based on robustness, i.e. the number of edge-disjoint routes it offers. The proposed routing approach was formally specified. The correctness of the algorithm was proven in as well as the complexity, latency, and the number and sizes of messages. Experimental results obtained through

simulation in Internet-like topologies were presented, allowing an evaluation of choices for the algorithm parameters.

Future work includes extending the proposed algorithm to deal with QoS (Quality of Service) routing, allowing the selection of paths based on delay, cost and bandwidth. The development of a path cache, for which the first message for a source-destination pair establishes a path for others to follow, and so reducing overflow, is under study. Employing additional criteria for evaluating edges, such as the number of paths leading to the destination and the average distance of such paths, is also under consideration, as well as simulation tests comparing the algorithm with other well-known routing algorithms, such as Dijkstra and Bellman-Ford. Implementations on larger networks and on real networks are also planned for the future. Finally, a protocol using the proposed algorithm is being developed. In order to be practical when deployed in large networks, this protocol must employ techniques for enhancing the performance of the algorithm, such as off-line route evaluation, and routing flows instead of single packets.

References

1. Huitema, C.: Routing in the Internet. 2^{nd} edn. Prentice Hall, Upper Saddle River (1999)
2. Labovitz, C., Ahuja, A., Bose, A., Jahanian, F.: Delayed internet routing convergence. In: SIGCOMM. (2000) 175–187
3. : Java Technology () `http://java.sun.com`.
4. Ford Jr., L.R., Fulkerson, D.R.: Flows in networks. Princeton University Press (1962)
5. Cormen, T.H., Leiserson, C.E., Rivest, R.L.: Introduction to Algorithms. Second edn. McGraw-Hill (1990)
6. Dijkstra, E.W.: A note on two problems in connexion with graphs. Numerische Mathematik **1** (1959) 269–271
7. Faloutsos, M., Faloutsos, P., Faloutsos, C.: On Power-Law Relationships of the Internet Topology. In: Proceedings of the ACM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication (SIGCOMM'99), Cambridge, Massachusetts, USA, ACM Press (1999) 251–262
8. Medina, A., Matta, I., Byers, J.: On the Origin of Power Laws in Internet Topologies. SIGCOMM Computer Communication Review **30**(2) (2000) 18–28
9. Chen, Q., Chang, H., Govindan, R., Jamin, S., Shenker, S., Willinger, W.: The Origin of Power-Laws in Internet Topologies Revisited. In: Proceedings of the 21st Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM'2002). (2002)
10. Tangmunarunkit, H., Govindan, R., Jamin, S., Shenker, S., Willinger, W.: Network Topology Generators: Degree-Based vs. Structural. In: Proceedings of the ACM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication (SIGCOMM'2002). (2002) 147–159
11. Bu, T., Towsley, D.F.: On Distinguishing between Internet Power Law Topology Generators. In: Proceedings of the 21st Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM'2002). (2002)
12. Pei, D., Zhang, B., Massey, D., Zhang, L.: An analysis of convergence delay in path vector routing protocols. In: Computer Networks. Volume 50:3. (2006)

13. Sahoo, A., Kant, K., Mohapatra, P.: "improving bgp convergence delay for large-scale failures". In: The 7th IEEE/IPIP International Conference on Dependable Systems and Networks (DSN'06), Philadelphia, U.S.A. (2006)
14. Wong, T., Jacobson, V., Alaettinoglu, C.: Internet Routing Anomaly Detection and Visualization. In: The 6th IEEE/IPIP International Conference on Dependable Systems and Networks (DSN'05), Yokohama, Japan (2005)
15. Zhang, H., Arora, A., Liu, Z.: A Stability-Oriented Approach to Improving BGP Convergence. In: The 23rd IEEE International Symposium on Reliable Distributed Systems (SRDS'04), Florianópolis, Brazil (2004)
16. Wang, L., Massey, D., Patel, K., Zhang, L.: FRTR: A scalable mechanism for global routing table consistency. In: Proceedings of the IEEE/IFIP International Conference on Dependable Systems and Networks (DSN'2004), Florence, Italy (2004) 465–474
17. Pei, D., Wang, L., Massey, D., Wu, S.F., Zhang, L.: A Study of Packet Delivery Performance During Routing Convergence. In: The 4th IEEE/IPIP International Conference on Dependable Systems and Networks (DSN'03), San Francisco, U.S.A. (2003)
18. Chandrashekar, J., Duan, Z., Zhang, Z.L., Krasky, J.: Limiting Path Exploration in BGP. In: The 24th IEEE INFOCOM (INFOCOM'04), Miami, U.S.A. (2005)
19. Pei, D., Zhao, X., Wang, L., Massey, D., Mankin, A., Wu, S., Zhang, L.: Improving BGP Convergence through Consistency Assertions. In: The 21st IEEE INFOCOM (INFOCOM'02), New York, U.S.A. (2002)
20. Duarte Jr., E.P., Santini, R., Cohen, J.: Delivering packets during the routing convergence latency interval through highly connected detours. In: Proceedings of the IEEE/IFIP International Conference on Dependable Systems and Networks (DSN'2004), Florence, Italy (2004) 495–504
21. Gomory, R.E., Hu, T.C.: Multi-terminal network flows. SIAM Journal on Applied Mathematics **9** (1961) 551–556
22. Funagalli, A., Valcarenghi, L.: Restauration vs. WDM Protection: Is There an Optimal Choice? In: IEEE Network. (2000)
23. Rosen, E., Viswanathan, A., Callon, R.: RFC 3031: Multi-Protocol Label Switchin. (2001)
24. Hellstrand, F., Sharma, V.: RFC 3469: Framework for MPLS-based Recovery. (2004)
25. Correia, R.B., Pirmez, L., et al.: Rerroteamento Parcial Pró-Ativo em Redes Baseadas em Circuito Virtual no Suporte ao Gerenciamento de Desempenho Pró-Ativo. In: XXIII Simpósio Brasileiro de Redes de Computadores (SBRC'2005), Fortaleza, Brazil (2005)
26. Medhi, D.: A Perspective on Network Restoration. Handbook of Optimization in Telecommunications (2005)
27. Puype, B., Yan, Q., Colle, D., et al.: Multi-Layer Traffic Engineering in Data Centric Optical Networks. In: COST266-IST OPTIMIST Workshop on Optical Networks, Budapest, Hungary (2003)
28. Tanaka, S., et al.: Field Test of GMPLS All Optical Path Rerouting. IEEE Photonics Technology Letters **17**(3) (2005)