Vicinity Routing in Large Scale Networks

*Israel Cidon and Atai Levy**

Department of Electrical Engineering
Technion - Israel Institute of Technology
Haifa 32000, Israel

May 24, 1998

Abstract

Efficient routing has been one of the most challenging and extensive researched topics in the communication networks field. During recent years, collecting the topology and network state information essential for the routing to every node has become a popular approach. The most common technique for solving this task termed a link state protocol (or topology update protocol). Unfortunately, using this technique in large networks introduces a problem of handling a large amount of data and information updates. Most of the solutions to this problem refer to hierarchical division of the network into smaller clusters.

Our solution takes a different direction. The algorithm defines a vicinity around each node that is to be updated with the change in the local node and link information. For the routing of information outside the vicinity an hierarchical yet flexible structure of borders node is defined. The route is calculated up to the nearest border and from this point a new calculation is made. This new architecture eliminates the need of dividing the network into clusters, in particular solving the inefficiency when such partition is done manually.

We have also investigate the ability to aggregate further information regarding remote parts of the network, hence, reducing the amount of storage and updates required.

1 Introduction

Efficient routing has been one of the most challenging and extensive researched topics in the communication networks field. Routing includes the problem of finding a way to pass a packet from one host to another through a datagram oriented networks (e.g. Internet) or defining the path for establishing connections through a circuit based networks (e.g. the telephone network, ATM or SNA[11]).

Two major tasks at all versions of routing solutions are the calculation of the path itself given a known network topology and state, and the related problem of gathering the geographically distributed information which is needed for performing such a calculation. Many times the two tasks are binded together and are quite inseparable. One of the most common example is the set of distance vector (also termed "Bellman Ford") solutions[4] that used to be popular in the past.

During the last years, the evolution has been toward the separation of the routing problem (at least within limited regions) to the above two different tasks. The route calculation task has been many times left to the implementor (as it requires no communication between nodes) or was given specific solution for a specific network (e.g. OSPF[6]). The task of collecting the topology and network state information has been mostly concentrated around a single solution termed a *link state protocol* (also termed topology update protocol). In the link state technique each station learns the topology of the entire network. The link state protocol became the most popular solution in the emerging standards namely IP and ATM. It is part of the relating network control platforms such as OSPF[6] and PNNI [1] and other "modern" network such as APPN[3].

In the link state protocol each node broadcasts to the entire network each change of its local edges. The broadcast is conducted through flooding which means that this information is forwarded over every link. Each node maintains a topology database in which the most recent updates regarding all links are stored. For routing calculations the node usually uses some version of shortest path algorithm using the network description contained in its local database. Unfortunately, using this technique in large networks routing introduces a new problem. Each routing station must receive and store a large amount of information updates since each station must maintain the entire network topology. Furthermore, the network and nodes might become loaded with frequent updates since each change must passed to all nodes over all links.

Most of the solutions to this latter problem[2, 5, 12] use an hierarchical division of the network into smaller clusters. Nodes use the full detailed network topology map to handle the routing within clusters. For example, in the IP standard, routing information within the cluster (or an Autonomous System) is forwarded using a link state algorithm (OSPF). Routing over the backbone network that connects the clusters is performed by a vector distance type algorithm (BGP). In the PNNI standard the routing information is split into multiple hierarchical levels. Cluster are larger as we go higher in the hierarchy. To confine the amount of information originated at higher level clusters an aggregation and compression of information must be used. In each level the nodes have a complete knowledge of its cluster topology. Within each cluster a leader is chosen to represent the network in the next higher level of the hierarchy. In this hierarchical solution of static division, all the nodes within a cluster have the same view of the network. Nodes that are at different clusters will have a different knowledge of the network. Note that even two adjacent nodes can be included in different clusters and hence must use high level clusters for the routing to each other. Similarly, any other routing across the boundaries of the clusters may be sub-optimal since the routing nodes do not have a complete information of the path. Thus, nodes that are expected to extensively communicate to each other should belong to the same cluster so a better routing will result. In addition, the (static) division of the network into clusters is done a priory while the network topology and traffic patterns may change dynamically without a related clustering modification. The changing traffic distribution in the network may change the notion of locality of the node information. Furthermore PNNI is based on leader election and each leader is used for the next level. Crashes of this leader or changes in the topology may result in a long recovery time.

In connection oriented networks such as ATM, establishing a connection is regularly done using source routing (PNNI) even for remote destinations. The use of source routing may results in sub optimal route since the source station is relying on outdated information as a result of the distance to the destination.

Jaffe [8] solves the problem of routing in large networks in different way. He shows an algorithm where two local databases are maintained. One is the inner cluster database, the other is the radius database. The radius database includes all the topology of the network that is at distance up to a certain radius. The maintenance of the radius database also requires more of the network resources as we will later describe. Routing is done using the best out of the two shortest paths of both databases. His solution still uses the fixed partition of the network into clusters. In particular, routing outside the cluster (and the radius) use only the fixed hierarchical division of the network and does not gain an advantage from the introduction of the radius database.

Our solution is based on the idea of including at each node a radius database but it takes a different direction. The algorithm defines a vicinity for each node that is to be updated with the change in the topology and other link information. This eliminates the need of dividing the network into clusters, in particular solving the inefficiency when such partition is done manually. For the case of circuit based routing, our solution calculates the routing path in increments or segments. The route is calculated up to a certain point and from this point a new calculation is made. When comparing to source routing, this solution may result in better route since the decision is made calculated using updated data. On the other hand it avoids the overhead incurred by calculating the route at every hop. Similar solution can be used in connectionless networks where routing outside the node vicinity is done toward nodes that report short distances to the final destination.

The rest of the paper is organized as follows: in section 2, an outline of our solution is described. In section 3, a detailed description of the inner algorithm is given and some practical improvements for implementation of this algorithm. Routing outside the node vicinity is described in section 4. We show some enhancements to decrease the network overhead in section 5.

2 Outline of the proposed solution

The routing algorithm is based on the classical assumption that most of the messages are destined locally. In order to exploit locality, our algorithm, similar to other hierarchical algorithms, divide the network into regions[9]. Most other algorithms use static division to clusters and splitting nodes to different cluster. This division may cause two adjacent stations to be in different clusters. Our algorithm avoids the use of fixed clusters by defining for each station a local environment. Each station knows all the topology information (links) that are at a fixed distance from it. This distance defines for each node an environment that is termed the *node's vicinity*. The distance can be measured using a variety of metric parameter (hops, delay, administrative weight, etc.).

Routing information is disseminated using a link state algorithm (topology update algorithm) up to a limited distance. Within the vicinity, the algorithm is based on the topology database (link state) model and, outside the vicinity on a combination of the distance vector (Bellman Ford) algorithm and the local link state. This way, nodes receive complete information about nodes that are at a smaller distance from them. Unlike the link state algorithm, since each node has a different vicinity, the information stored at each node is different. Namely, the topology database at a node i will be composed of all nodes r that have i within their vicinity and the link emigrating from these nodes. The notion of the node vicinity also defines the node's *border nodes* as these nodes within the node vicinity whose distance is equal to the vicinity radius ($\mathcal{R}$). Each of the border node b is responsible for passing in its vicinity, the information that node r is at distance of $\mathcal{R}$ from itself. In particular, the nodes which are outside r's vicinity (but inside b's vicinity) learns about r through the link state updates of b. This guarantees that all nodes that are at distance $2 \cdot \mathcal{R}$ or less from r, learn about r. Moreover, the border nodes of b deliver the information about r to all the nodes that are at distance of $3 \cdot \mathcal{R}$ or less from r. This continues until the information of r reaches all the nodes in the graph. The routing outside the vicinity is done in several steps. The source chooses a route to the best border node inside its vicinity (best in the sense the overall distance from the destination through this node is smaller or equal to other choices). The selected border node continues this path to its best border node. This is done until destination is reached.

Throughout our discussion we will assume that all the routing algorithm uses minimum hop as their routing criteria. Later, we will show how our solution can be adjusted to other optimal routing criteria and the inclusion of additional QoS constraints.

3 Hop Limited Topology Update algorithm

In our description of the routing algorithm we refer to a routing station as a node. Nodes are connected by bidirectional links which are composed of a pair of opposite directional edges. A link is considered operational for the purpose of routing if both directional edges are operational. We define distance between two nodes as the minimum number of operational links between them. The basis for the algorithm is that every node has a full updated knowledge of its outgoing edges. The node is termed *responsible node* regarding its local outgoing edges. Each edge is described using a field termed *edge description* which contains some parameters that describe the edge such as capacity, throughput, delay, etc. It also contains the *edge description number* which is a sequence number that is assigned to distinguish a new data from an old one. The node stores this information in the *node database*. The node *vicinity* is termed as all the nodes whose distance from the responsible node is smaller than a system parameter termed $\mathcal{R}$. Each node is responsible to pass its edges descriptions to all the other nodes within its vicinity. For each edge entry the node maintains a *passed list* which contains the identity of the adjacent nodes that this description was passed to.

The message model described here is the classical dynamic model[10]. In this model, if a message is sent over an operational link it will correctly arrive within finite time or the link fails. An addition to the failure model is the assumption that a station may crash (becomes inoperative) due to shutdown or power failure. When this situation occurs all its links fail to operate and its memory may be lost and cannot be trusted. The station knows when it comes up and potentially recovers from a crash. The algorithm works in the following way:

Change in local edge description: For every change in the edge description, the node increases the edge description number by one. The node sends to all its neighbors[1] a new UPDATE message which contains the new description including the new description number. Also, each node performs a process termed *linked update* which will be described later. The whole operation of sending new UPDATE message with new edge description number will be termed *publication*. When an adjacent edge fails, the node deletes the neighbor connected via this edge, from all the passed lists of all the edges it maintains.

Arrival of UPDATE: Each UPDATE message contains an edge description which includes an edge description number. When a node receives this message, it compares it to its local view of the same edge. If the data in the UPDATE is newer (in terms of higher description number), it replaces its database information with the one from the UPDATE message and publishes it to all the neighbors that are inside the edge's responsible node vicinity, i.e. the distance between the responsible node and the neighbor according to the current database is smaller than $\mathcal{R}$. Else, it discards the message. This holds unless the UPDATE is regarding its own outgoing edges (it is the responsible node for this edge). In this case, if it receives a higher description information than the one in its database, it must publish a new UPDATE message with a higher description number.

In all UPDATE reception cases, the new received information may change the distance between certain responsible nodes and certain neighbors. In particular, parts of the database that were outside the neighbor vicinity may now be inside it. The node must check for each of its neighbors which edges description should be passed on to it. It sorts them in increasing order according to their distance from the neighbor node. The node sends to its neighbor these edge descriptions each in a new UPDATE message. It also adds the neighbor identity to the *passed list* of each such edge. This last operation is termed *linked update*.

[1]If we use a weight function other than hop distance we should pass the UPDATE message only to neighbors which are at distance less than $\mathcal{R}$

Local edge becomes operational: When an attached edge becomes operational, a new neighbor is added to the node. Since this neighbor is not in the *passed list* of any of the edges in the database, relevant parts of local database (according to the distance from an edge to the neighbor) are passed in separate UPDATE messages to the other side. The node updates the *passed list* accordingly. The node also performs a *linked update.*

Deleting of distant entries: To prevent keeping unnecessary information in the nodal database, the node may delete, in response to an UPDATE message, entries that are outside its vicinity. It is done by sending to all its neighbors a DELETE message that includes this edge description. After sending this description, the entry for this edge is deleted.

Receiving a DELETE message: The DELETE message includes an edge description and the sending node identity. The receiving node checks its database for the edge entry. If the responsible node for this edge is at distance larger than $\mathcal{R}$ from the sending node, the node deletes the sending node from the edge passed list. Else, it replies with an UPDATE message that includes the edge description and edge description number.

Jaffe [8] describes an algorithm based on the "radius database". First his algorithm passes topology information using a distance vector methodology up to a certain distance. This largely increases the number of updates that are transmitted by nodes. It suffers several drawbacks in comparison to our algorithm. Each UPDATE message also contains the distance to the source of the information and of the cluster it belongs to. This additional information is not needed in our solution as described above. His algorithm does not have a mechanism to delete entries that were in the past within the node radius. Hence, the size of the topology database can be very large. It also has larger number of messages. In the worst case a node may receive and change an entry for a particular edge and sequence number up to $\mathcal{R}$ times. since it may arrive in decreasing order. For each of these arrivals the node will publish a new message that may cause $\mathcal{R}$ times the number of messages in our algorithm.

3.1 Formal description of the UPDATE algorithm

Messages

- UPDATE :

    1. Edge identifier.

2. Description parameters - capacity, delay, throughput...
3. Description number - a sequence number to distinguish between older and newer description.

- DELETE :

1. Edge identifier.

Data Structure

Node database - Each entry includes:

1. Edge description.
2. Edge description number.
3. Passed list - A list of the node neighbors this edge description has been passed to.

Definitions

e - Identity number of edge e.

$\underset{msg}{\mathrm{num}}(e)$ - Description number of edge e in the message.

$\underset{msg}{\mathrm{des}}(e)$ - Description parameters of edge e in the message.

$\underset{db}{\mathrm{num}}(e)$ - Description number of edge e in the database.

$\underset{db}{\mathrm{des}}(e)$ - Description parameters of edge e in the database.

$Pass(e)$ - The group of all the neighbors nodes that the description of e was passed to.

$n(e)$ - The neighbor that is connected via edge e.

O - The set of all the outgoing edges from the node.

N - The set of all the neighbors of the node.

r_e - The responsible node of edge e.

Local_Node - The identity of the node which run this algorithm.

dis(n_1, n_2) - The shortest distance between node n_1 to node n_2.

dis(e, n_1) - The shortest distance between node r_e to n_1.

Messages :

UPDATE(e, $\underset{msg}{\mathrm{des}}(e)$, $\underset{msg}{\mathrm{num}}(e)$).

DELETE(e, $n \in N$).

Algorithm

Edge e becomes operational

For each $e \in$ *database*:

send UPDATE(e, $\text{des}_{db}(e)$, $\text{num}_{db}(e)$) $\Longrightarrow n(e)$.

$Pass(e) \leftarrow Pass(e) + n(e)$.

Arrival of UPDATE(e, $\text{num}_{msg}(e)$, $\text{des}_{msg}(e)$)

If (($\text{num}_{msg}(e) > \text{num}_{db}(e)$) and ($e \notin O$))

$\text{num}_{db}(e) \leftarrow \text{num}_{msg}(e)$.

$\text{des}_{db}(e) \leftarrow \text{des}_{msg}(e)$.

$Pass(e) \leftarrow \emptyset$

call CHECK_UPDATE.

Else if (($\text{num}_{msg}(e) > \text{num}_{db}(e)$) and ($e \in O$))

$\text{num}_{db}(e) \leftarrow \text{num}_{msg}(e) + 1$.

$Pass(e) \leftarrow \emptyset$

$\forall n \in N$

send UPDATE(e, $\text{des}_{db}(e)$, $\text{num}_{db}(e)$) $\Longrightarrow n$.

$Pass(e) \leftarrow Pass(e) +$ n.

Else discard the message.

call DELETE_DISTANT.

Change in edge description of $e \in$

For each change:

$\text{num}_{db}(e) \leftarrow \text{num}_{db}(e) + 1$.

$Pass(e) \leftarrow \emptyset$

$\forall n \in N$

send UPDATE(e, $\text{des}_{db}(e)$, $\text{num}_{db}(e)$) $\Longrightarrow n$.

$Pass(e) \leftarrow Pass(e) +$ n.

call CHECK_UPDATE.

Receiving a DELETE(e, n)

If ($e \notin$ *database*) discard the message.

Else If ($dis(e, n) > \mathcal{R}$)

$Pass(e) \leftarrow Pass(e)$ - n.

Else ($dis(e, n) \leq \mathcal{R}$)

send UPDATE(e, $\text{des}_{db}(e)$, $\text{num}_{db}(e)$) $\Longrightarrow$ n.

CHECK_UPDATE

For each $n \in N$

$Not_Passed \leftarrow (\forall$e $\in$ database $|$ ($dis(e, n) < \mathcal{R}$) and ($n \notin$Pass(e)))

Sort Not_Passed according to dis(e, n) (smaller first).

For each $e \in Not\_Passed$

send UPDATE(e, $\text{des}_{db}(e)$, $\text{num}_{db}(e)$) $\Longrightarrow$ n.

$Pass(e) \leftarrow Pass(e) +$ n.

DELETE_DISTANT

For each $e \in database$

If ($dis(e, Local\_Node) > \mathcal{R}$)

For each $n \in N$

send DELETE(e, Local_Node) $\Longrightarrow$ n

delete the entry for e in the database.

3.2 Correctness proofs

Jaffe [7] has proven the correctness his version of the radius database. His algorithm is very different from ours including his correctness properties. Therefore, there is no way to derive our correctness from his and we presents the correctness of our algorithm.

In the following section we assume that a finite number of edge descriptions changes and a finite number of node crashes is encountered in the network.

Theorem 1 *After a finite time since the last edge changed or the last node crashed, no more messages are sent or accepted. Also, all nodes have an updated description of all the edges inside their vicinity.*

The proof of this theorem involves the definition and proof of several lemmas and it is given in Appendix A.1.

3.3 Practical considerations

Deleting of an edge entry

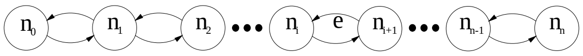


Figure 1:

A node may delete all the entries of edges that are outside its vicinity. If an edge is going up and down frequently, it will cause a large amount of routing information to be passed every time it is going up. This is worse when the graph become partly disconnected. For example, in figure 1, if all the nodes delete all the information for the other half of the graph each time that the edge e becomes inoperative then, it will cause all the information to be passed again when the node is operational again. It can be avoided if the nodes will delete these entries after a longer time.

Routing with other parameters than hops

The algorithm as described above is proven for the case of a hop distance based radius but the algorithm may use any weight function. It may include others parameters like delay, capacity, throughput, etc. The only restriction is that the same weight function will be used by all the node in the graph.

Configuring a vicinity

In some instances the flexibility of the vicinity boundaries may not be suitable. For example, it may happen if a certain weight function cause a vicinity to be too large. Another example is when the vicinity structured across certain administrative boundaries. Specific nodes may be forced to be the end of a vicinity by configuring its outgoing edge that crosses the end of the planned vicinity, with a weight equal to $\mathcal{R}$.

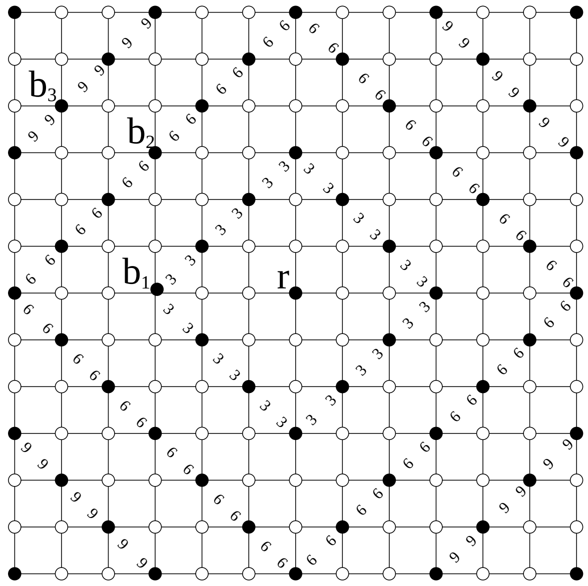


Figure 2: Which nodes become border nodes

4 Routing outside the Vicinity Size radius

Outside the node vicinity, the routing information is handled by length increments of $\mathcal{R}$. In the first increment, all[2] nodes that are at distance of $\mathcal{R}$ from a responsible node r, publish themselves as *border nodes of order 1 for r*. In the second increment, all nodes that have the closest first order border node exactly at distance of $\mathcal{R}$, and do not have r inside their vicinity publish themselves as *border nodes of order 2*. Clearly, these nodes are at a minimal distance of exactly twice $\mathcal{R}$ from r. In each increment, each node i marks in its database inside its vicinity the border nodes of the smallest order for r. If among all these marked nodes the closest one to i is at distance of $\mathcal{R}$, i becomes a border node. The order that i takes is one larger than the marked nodes. These increments continue until all the nodes in the graph finish to publish themselves as border nodes of any order for r. For example, in figure 2, we examine the responsible node r. Within its vicinity (marked with radius 3) all the nodes includes r in their database according to the inner protocol. The nodes at distance 3 from r (mark in bold circles) publish themselves as *border nodes of order 1*. The nodes that are at distance 3 from the first order border nodes (excluding r) publish themselves as *border nodes of order 2* for r.

[2] Later we will describe a way to decrease the number of border nodes in order to prevent message overload and memory consumption.

All these *border nodes of order 2* forms a logic circle of radius twice $\mathcal{R}$ around r (marked with radius 6). This is done until every node r in the graph is at distance less than $\mathcal{R}$ from at least one border node for every node in the graph.

When a node establishes a connection to a destination that is outside its vicinity, it establishes first a path to one of the closest[3] border nodes of minimum order in its vicinity. This node will establish a path to the next border node of smaller order, etc. This is done until the destination is reached. Note, that this procedure results in a minimum hop distance route.

4.1 The distributed Algorithm

The following actions are all taken as a reaction to a Change in the node vicinity. The changes in the node vicinity are due to the local edges changes or to the arrival of UPDATE messages that affect the database information. We add to the first protocol a new type of topology item which describes a border node. In this topology item each border node includes the destination node identity, its identity as a border node, its order, a sequence number, and some metric that describes its path to the destination node (to be later discussed). This topology item is passed using a regular UPDATE message as part of the *inner vicinity protocol.* Special tag is added to the message to indicate if the border node ceases to function as a border node for a certain node r.

After each change, node i will perform the next procedure: For each node r, if according to i's local view i is at a distance of $\mathcal{R}$ from r, i becomes a *border node of order 1* for r. For each node r that is outside i's vicinity i will examine all the border nodes for r that are within its vicinity. Among all these nodes i chooses the borders with the smallest order. If the closest one is at distance of exactly $\mathcal{R}$ from i, i becomes a border node of order one larger than the ones it chose and sends an UPDATE message appropriately.

If i was a border node for a certain node, and now its distance is smaller than $\mathcal{R}$ from the smallest and closest border node, it ceases to function as a border node, informing all the nodes inside its vicinity through an UPDATE message.

4.2 Formal description

Definitions

[3]Cases where routing includes QoS constraints will be discussed later.

r - Identity number of responsible node r.

ord(b) - The order of border b.

$path_r$ - The description of a path or paths from Local_Nodeto node r.

Local_Node - The identity of the node which run this algorithm.

Algorithm

Change in the node vicinity
call CHECK_BORDER.

Arrival of UPDATE
call CHECK_BORDER.

CHECK_BORDER
For each r $\in$node vicinity:
  If $(dis(Local\_Node, r) = \mathcal{R})$
    call BE_BORDER(r, 1).
  If Local_Node is a border node for r and
    $(dis(Local\_Node, r) \neq \mathcal{R})$
      call CEASE_BORDER(r).
  If r is a border node of order k for node n
    and $(dis(Local\_Node, r) = \mathcal{R})$ and
    $\nexists m \in database|$(m is a border node of
    order smaller than k for node n) or
    ((m is a border node of order k for node n)
    and$(dis(Local_Node, m) \neq \mathcal{R})$)
      call BE_BORDER(r, k).
  Else if not exists such r in vicinity
    call CEASE_BORDER(r).

CEASE_BORDER
For each $n \in N$
  send UPDATE(r, No_Border,
  Local_Node) $\Longrightarrow$ n.

BE_BORDER(,)
choose one border node m for r with the smallest order.
$path_r$ $\leftarrow$ The combination of the path to m and the path from m to r.
For each $n \in N$
  send UPDATE(r, $path_r$,
  Local_Node, ord(m)+1) $\Longrightarrow$ n.

4.3 Correctness Proof

In the following we assume that after some time t the network stabilizes in the sense that no more edges changes happen and the network is connected.

Theorem 2 *After a finite time since the last edge topological change or the last node crash, no more messages are sent or accepted and every node has inside its vicinity at least one border node for every node in the graph.*

The proof of this theorem involves The definition and proof of several lemmas and it is given in Appendix A.2.

5 Practical Enhancements

5.1 Reducing the number of border nodes

In our description of the outer routing algorithm, each node at distance of $\mathcal{R}$ from a responsible node must become border node, and must publish itself as a border node within its vicinity. While this approach solves the problem of having too many nodes in the nodes database, it may result in a large number of messages and high memory consumption due to a large number of border nodes for every node in the graph each advertises the distance to that responsible node within its vicinity. In the sequel, we describe a modification to the algorithm that reduces the total number of border nodes for a responsible node r. A reduction in the number of border nodes results in a sub optimal routing. Let's term the length of the optimal path between nodes i and j to be *Opt(i,j)*. We define a *reduction factor* $0 < \alpha < 1$ and we will present an algorithm that produces routing paths with length bounded by $(1 + \alpha) \cdot Opt(i, j)$. In other words, the algorithm introduces a trade-off between the reduction in the number of border nodes for some node r and the increase in the length of the routes that lead to r.

The modified algorithm changes the decision process whether or not a node n should become a border node of order k for r. In addition to its distance to the closest border node of order *k-1* for r, this node also checks if its distance from all the known (known refers to this node database) border nodes of order k is larger than $\frac{\alpha}{2} {*} \mathcal{R}$. If its distance is larger, then the node must become a border node of order k for r.[4]

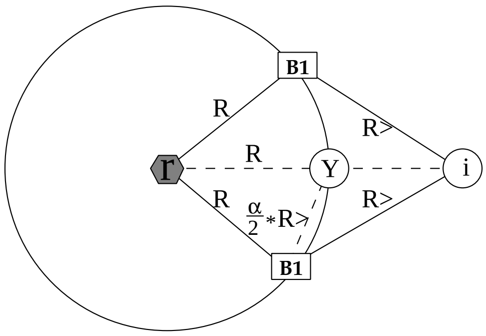


Figure 3: A node should become a border node

[4]Note that, in order to improve the quality of paths between certain nodes in terms of routing metrics other than hops, a node may also be required to become a border node even if its distance is smaller than $\frac{\alpha}{2} {*} \mathcal{R}$ from the closer border node of the same order.

For example, node Y at figure 3 will not publish itself as a border node since its distance from the other border node is less than $\frac{\alpha}{2} * \mathcal{R}$.

In certain timing scenarios, this modification alone does not prevent many or all nodes at distance of $\mathcal{R}$ from r from becoming border nodes. This may happen since the local knowledge of distances may be known before they receive the publication of other border nodes of the same order within their vicinity. Therefore, additional modifications are needed to indicate when a node should cease or become a border node for r. We may suggest that a node i may cease to function as a border node if, according to i's local view, all nodes in i's vicinity have at least one border node (except for i) with order k for r at distance smaller than $\frac{\alpha}{2} * \mathcal{R}$. However, this condition alone may raise a new problem, where a group of nodes exchange the role of being a border node for the same r indefinitely. To prevent this problem we permit a node to cease being a border node only if it has a border node with higher identity at distance smaller than $\frac{\alpha}{2} * \mathcal{R}$.

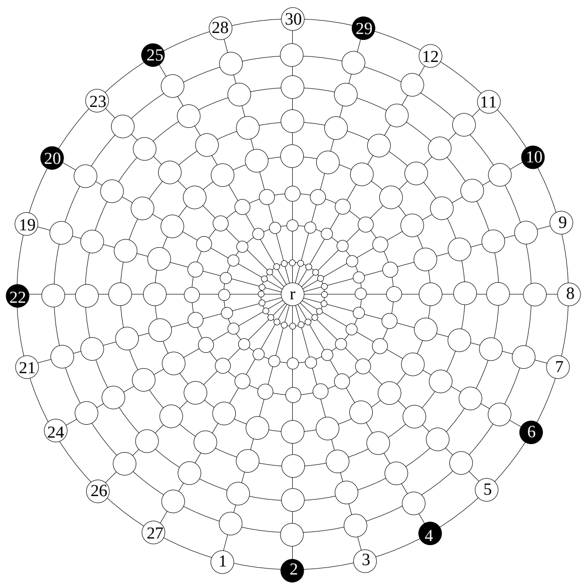


Figure 4: Reducing the number of border nodes

For example Figure 4 shows a network with $\mathcal{R} = 8$ and $\alpha = \frac{3}{4}$. The current border nodes are marked as bold circles. Node 20 will cease to function as a border node since it knows a border node with a higher identity (25) at distance smaller than $\frac{\alpha}{2} * \mathcal{R}$=3. Nodes 2, 4 will also cease being a border nodes since border 2 knows border 4 and border 4 knows border 6. But after node 4 ceases being a border node, node 2 will notice that it has no border node at distance of $\frac{\alpha}{2} * \mathcal{R}$ and it will become a border node again.

This algorithm reduces the number of border nodes. Finally, every two adjacent border nodes will be at a distance of at least $\frac{\alpha}{2} * \mathcal{R}$ of each other and the maximal distance between two adjacent border node will be $\alpha * \mathcal{R}$. Note that for $\alpha \leq \frac{2}{\mathcal{R}}$ the modified algorithm is equal to the unmodified one, and for higher value of α the number of border nodes decreases. We cannot increase α beyond 1 since the node learns its distance from the inner algorithm and it only provides information up to a $\mathcal{R}$ radius. The number of border nodes depends on the graph topology. In the case where the graph corresponds to a circle of n border nodes around a responsible node (as in figure 4), the number of border nodes is reduced to less than $n \cdot \frac{2}{\alpha \cdot \mathcal{R}}$.

Theorem 3 *A finite time after the last topology change, there will be no more changes in the border nodes.*

Proof Let's look at an arbitrary node r. We prove this theorem by applying induction on the order of the border node for r. The base of the induction is true for order 0 as the responsible node does not change. The step of the induction assumes that all the border nodes of order n-1 for r have published themselves and will not change any more. Let S be the set of nodes that are at distance $\mathcal{R}$ from a border node of order (n-1) which become border nodes of order n for r and publish themselves during the algorithm. Let's look at node $x \in S$ that has the highest identity in S. This node will not cease to be a border node of order n for r since it will not know any other border node of order n for r that has a higher identity. According to the algorithm, all nodes that are at distance smaller than $\frac{\alpha}{2} * \mathcal{R}$ from x will cease to function as a border node since they know a higher identity *border node of order n for r* at distance smaller than $\frac{\alpha}{2} * \mathcal{R}$. Let's remove all these nodes including node x from S, and look for the higher identity again. We continue this procedure until there are no more nodes in S. Thus, the final role of node in S is obtained. Therefore the set of border nodes of order n for r is finite and will not change any more. □

Lemma 1 *A node that is at distance of $\mathcal{R}$ from the closest border node* B *of order n for node* r *and does not have border node of order n-1 for* r *within its vicinity, is at distance of at least $\frac{(n+1) \cdot \mathcal{R}}{(1+\alpha/2)}$ hops from* r.

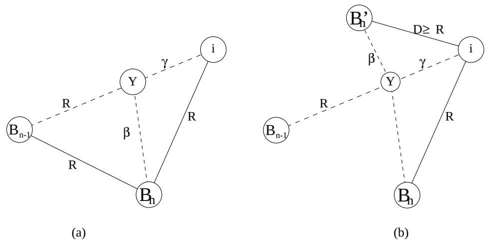


Figure 5:

Proof The lemma states the property for every node that has potential to become a border node of order n+1 for r Including, of course, the final border nodes of that order. Let's look at an arbitrary responsible node r. We will prove this lemma by induction over the order of the border nodes for r. The base of the induction clearly holds since border nodes of order 1 for r are at distance of $\mathcal{R} > \frac{1\cdot\mathcal{R}}{(1+\alpha/2)}$. We assume that every border node of order ℓ ($\ell \leq n$) is at least at distance $\frac{l\cdot\mathcal{R}}{(1+\alpha/2)}$[5]. Let's look at any node i that is at a distance of exactly $\mathcal{R}$ from its closest border node B_n of order n for r and does not have a border node of order n-1 within its vicinity. The shortest path between i and its closest border node of order n-1 (which we term B_{n-1}) must pass through a node Y that is at distance of $\mathcal{R}$ from r (see figure 5(a)or(b)). If $Y = B_n$ the lemma clearly holds, hence we assume $Y \neq B_n$.

We examine two cases:

1. B_n is the closest border node of order n to Y (figure 5(a)). We mark the distance between Y and i with γ, and the distance between Y and B_n with β. According to the algorithm, $\beta \leq \frac{\alpha}{2}\cdot\mathcal{R}$ otherwise Y would become border node of order n for r. We obtain, $\gamma+\beta \geq \mathcal{R}$ from the triangle inequality. Therefore $\gamma \geq \mathcal{R}-\beta \geq \mathcal{R}-\frac{\alpha}{2}\cdot\mathcal{R}$.

2. B_n is the not the closest border node to Y and there is another border node B'_n (figure 5(b)). We mark the distance between Y and i with D, and the distance between Y and B'_n with β. This case is even easier since $D \geq \mathcal{R}$ and $\beta \leq \frac{\alpha}{2}\cdot\mathcal{R}$ as before. Thus, We obtain as in the first case $\gamma \geq D-\beta \geq \mathcal{R}-\beta \geq \mathcal{R}-\frac{\alpha}{2}\cdot\mathcal{R}$.

In both cases, the distance between Y and i is at least $(1-\frac{\alpha}{2})\cdot\mathcal{R}$. From the induction

[5]It is not necessary for the proof to also assume that all potential border nodes follow the same condition

assumption we obtain that B_{n-1} is at distance of at least $\frac{(n-1)\cdot\mathcal{R}}{1+\alpha/2}$ from r. The distance from node i to B_{n-1} is $(2-\frac{\alpha}{2})\cdot\mathcal{R}$.

For $0<\alpha<1$, the distance from i to r is at least

$$\frac{(n-1)\cdot\mathcal{R}}{1+\alpha/2}+(2-\frac{\alpha}{2})\cdot\mathcal{R}>\frac{(n+1)\cdot\mathcal{R}}{1+\alpha/2}$$

□

Theorem 4 *If the routing procedure is for any node* i *to reach any node* r *is by* i *reaching first the closest border node of the smallest order for* r *and from there to follow closest border nodes in decreasing order. Then, the path length between nodes* i *and* r *is bounded above by* $(1+\alpha)\cdot Opt(i,j)$.

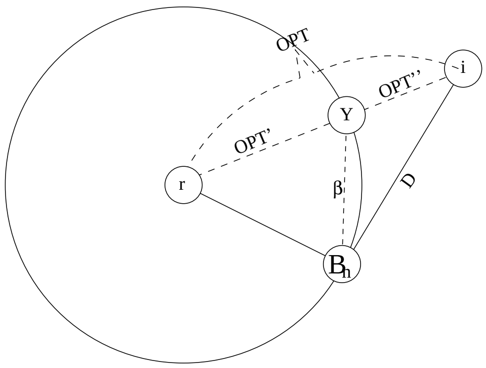


Figure 6:

Proof From lemma 1, we obtain that every node i at distance of exactly $\mathcal{R}$ from the closest border node for r satisfies this condition. A node that is at different distance will establish a connection also to the closest border node B_n (see figure 6). Node Y is the potential border node of order n which i should have used for optimal routing. We marked the distance from Y to i with OPT'' and the distance from Y to r with OPT'. From lemma 1 we obtain $OPT'\geq\frac{n\cdot\mathcal{R}}{1+\frac{\alpha}{2}}$, from the triangle inequality $D\leq OPT''+\beta\leq OPT''+\frac{\alpha}{2}$. The optimal route to r is $OPT=OPT''+OPT'$. The route length that i will establish is $D+n\cdot\mathcal{R}$ and :

$$(1+\alpha)OPT=(1+\alpha)(OPT'+OPT'')\geq(1+\alpha)\frac{n\cdot\mathcal{R}}{1+\frac{\alpha}{2}}+OPT'$$

$$\geq (1+\alpha)n \cdot \mathcal{R} + \frac{\alpha}{2} \cdot \mathcal{R} + OPT'' \geq n \cdot \mathcal{R} + D$$

□

5.2 Aggregating information

As in other distance vector based algorithms our outer routing algorithm might require large routing tables and long messages in order to maintain reachability to all nodes. Each node is a potential destination and all the other nodes in the graph must learn a way to reach it. Similar to all these other routing algorithms aggregation procedures should be devised in order to solve this problem. Unfortunately, such aggregation is not guaranteed to be efficient in the general case and require some 'aggregability' assumption regarding the distances and the naming of nodes that should be aggregated together. We will describe here a solution to this problem, under the same aggregation assumption, by combining border reports for a number of nodes within one border node.

An information can be aggregated by a border node which collapses several responsible nodes into one distance entry. Let's assume that a node m knows some border nodes $b_1 \ldots b_k$ that have close enough ('aggregable') description about their responsible nodes $r_1 \ldots r_k$. m may become a border node for $r_1 \ldots r_k$ even if its distance to some of the border nodes is smaller[6] than $\mathcal{R}$. In the UPDATE message, the node will publish a list of this group of nodes and an aggregation of the paths to them.

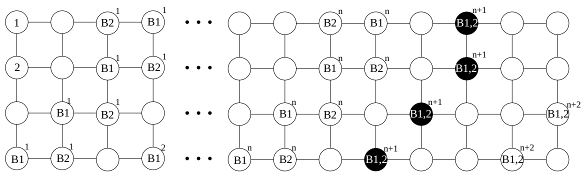


Figure 7: Aggregating border nodes.

The order that m will publish depend on $b_1 \ldots b_k$. If $b_1 \ldots b_k$ has the same order n, then m takes an order higher by one, n+1. For example, in figure 7 the vicinity radius is 3 hops. The number in the top right of every border is the order of the border. Nodes *1, 2* are aggregated together by the nodes marked with bold circles. If we look at the top bold node we see that its distance from border node of order n for 2 is equal to $\mathcal{R}$=3, but its distance from border node of order n for 1 is only 2 hops. This node

[6]In our solution, it can not be larger.

becomes a border node since it realizes that the distance between these two border nodes is small in relation to the distance to the responsible nodes (the order of the nodes times $\mathcal{R}$).

If $b_1 \dots b_k$ have different orders, then m publishes the minimum and the maximum orders plus one. For example if the smallest order of $b_1 \dots b_k$ is ℓ and the highest is k, m will publish the order as $\ell + 1$ to k+1. The lower number will be used by other nodes as the order of the node for the algorithm, and the maximum value will be used for routing as an estimation of the distance between the border node to the responsible nodes. If the order of the aggregation at the border node is ℓ to k the nodes in the vicinity that are closer than $\mathcal{R}$ will not become a border node if they do not know a border node with order smaller than ℓ at distance of exactly $\mathcal{R}$. The distance to all the responsible nodes that this aggregation node presents is less or equal than $k * \mathcal{R}$. The nodes that are at distance of $\mathcal{R}$ from this node and do not have another border node of order smaller than ℓ for these responsible nodes will become a border node of order $\ell + 1$ to k+1.

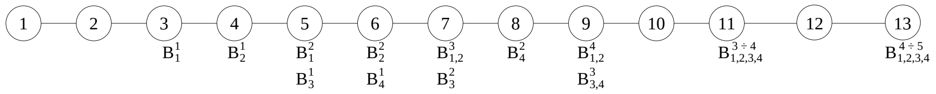


Figure 8: Aggregating different orders of border nodes together.

For example in figure 8 the vicinity size equals to two hops. The number in the top right is the order and the number in the bottom right is the identity of the aggregated nodes. Border node 7 is aggregating border nodes (5,6) of nodes 1, 2 with order 3. Border node 9 is aggregating border nodes (7,8) of nodes 3, 4 with order 4. Border node 11 is aggregating border node (9) of nodes 1, 2, 3, 4 with order 3÷4. Border node 13 is the next border nodes of nodes 1, 2, 3, 4 and it takes the order 4÷5.

6 Summary

We have developed a novel technique that performs a hierarchical routing without dividing the network into rigid clusters. Similar to other hierarchical architecture, this algorithm is based on the assumption of communication locality. Our system algorithm is divided into two parts namely the outer and the inner algorithms. In the first algorithm, each routing station has a complete knowledge of its local surrounding using a distance limited link state protocol. We also described a way to handle the outer routing using the update mechanism of the inner routing. Our outer algorithm present a way to communicate between remote station. It use the border node of the inner algorithm as the way to pass the routing information. Each border node presents to the outer surrounding itself as a way to route to the node. To prevent the problem of a large number of messages in the outer part, thus we devised a way to control the number of border nodes and to aggregate similar routing information.

In case of circuit based routing, our algorithm calls for the use of segmented routing. We intend to compare this routing to the complete source routing as it done today in the PNNI standard. Further work may also be done concerning the aggregation of names and information.

References

[1] *P-NNI Draft Specifications*, 1994. Revision 4.

[2] A. E. Baratz and J. M. Jaffe. Establishing virtual circuits in large computer networks. In *INFOCOM*, pages 311–318. IEEE, 1983.

[3] A.E. Baratz, J.P. Gray, J.M. Jaffe, and D.P. Pozefsky. SNA networks of small systems. *IEEE Journal of Selected Areas in Communication*, 3(3):416–426, May 1985.

[4] D. Bertsekas and R. Gallager. *Data Networks.* Prentice Hall, second edition, 1992.

[5] G. Huang and Shan Zhu. A new HAD algorithm for optimal routing of hierarchically structured data networks. Boston, Massachusetts, April 1995.

[6] C. Huitema. *Routing in The Internet.* Prentice Hall, 1995.

[7] J. M. Jaffe. Hierarchical clustering with topology databases. Technical report 88.155, IBM Israel Scientific Center, April 1985.

[8] J. M. Jaffe. Hierarchical clustering with topology databases. *Computer Networks and ISDN Systems*, 15:329–340, October 1988.

[9] L. Kleinrock and F. Kamoun. Hierarchical routing for large networks. *Computer Networks*, 1(3):155–174, 1977.

[10] Adrian Segall. Distributed network protocols. *IEEE Transaction on Information Theory*, IT-29(1):23 – 35, January 1983.

[11] A. S. Tanenbaum. *Computer Networks.* Prentice-Hall, second edition, 1988.

[12] Paul F. Tsuchiya. The landmark hierarchy: a new hierarchy for routing in very large networks. pages 35–42, Stanford, California, August 1988. ACM. also in *Computer Communication Review* 18 (4), Aug. 1988.

Appendix

A Correctness Proofs

A.1 Inner algorithm

In the following section we assume that a finite number of edge descriptions changes and a finite number of node crashes is encountered in the network.

Lemma 2 *Assume that for edge* e *the highest edge description number known at time* t_0 *is* $\max_e(t_0)$*. Also assume that between time* t_0 *to time* t *there are* $x_e(t_0)$ *changes in the state of* e *that were reported by its responsible node r(e), and r(e) has crashed no more than* $Cr_e(t_0)$ *since* t_0*.*

If $t > t_0$ *then* $\max_e(t) \leq \max_e(t_0) + x_e(t_0) + Cr_e(t_0) + 1$

Proof According to the protocol, only the responsible node of an edge may create a new edge description (with a different number). Let us observe, without loss of generality, an edge e with a responsible node r(e). All the edges description of e are originated in r(e). The edge description number in r(e) at time t_0 can be $\max_e(t_0)$ or smaller. According to the protocol, there are three cases that will cause r(e) to send an edge description:

i. Change in the edge (including the report of edge going down).

ii. Node recovers from a crash.

iii. Report from a neighbor with higher edge description number.

The first case results in the edge description number being increased by one. The second case causes the number to start again from zero. The third case causes the number to be one higher than the number in the report. According to the lemma assumption, from time t_0 until time t the first case happen $x_e(t_0)$ times. Each time it causes the number to increase by one.

The other possible event that triggers an increase in the description number is the third case. This can only happen after a node crash and recovery. The highest number that r can receive is the highest number before the crash. This will cause r to send a number that is higher by one than the one it received. Assuming there are only $Cr_e(t_0)$ node crashes since t_0, then, the increment can only happen $Cr_e(t_0)$ times.

If at time t_0 the number in r(e) is smaller than $\max_e(t_0)$, there may be a time in which $\max_e(t_0)$ arrive to r(e) and from that time it start to count the changes and crashes. But in this case $\max_e(t)$ will be even smaller.

Finally if we sum the three cases together we get

$$\max_e(t) \leq \max_e(t_0) + x_e(t_0) + Cr_e(t_0) + 1$$

□

Lemma 3 *Assume there are no DELETE messages sent from any of the nodes. After a finite time since the last edge change and the last node crash, no more messages are sent or accepted.*

Proof The time t is defined to be the last time a change in an edge or node crash happened.

Let's look, without loss of generality, at one edge e. Since the last time this edge was operating, the following event classes caused a transmission over the edge e:

1. Outgoing edge becomes operational.
   (a) If e becomes operational, The two connecting nodes exchange their database.
   (b) Another outgoing edge except e becomes operational. The exchange of the information on the other node cause an arrival of new information. Therefore, some linked information that has not been passed before may be passed on edge e.
2. Arrival of edge description UPDATE
   (a) If the UPDATE message is new and the node is not responsible for the described edge, it will be sent to the other side. Also linked information that have not been passed before, may be passed.

(b) If the UPDATE message is newer than the database record and the node is responsible for the edge in the UPDATE message, a new UPDATE message with higher number is sent.

3. Change in outgoing edge description

   (a) The node sends the new change over the edge.

   (b) Due to the change, parts of the database that have not been passed before, may be passed.

After time t, events of class 1 and class 3 do not happen since according to the assumption, no more changes in the edge occur. Events of class 2 happen only a limited number of times, since after each such event an UPDATE message is passed on the edge e and both nodes update their database for the higher number. Since the node does not crash after time t and no delete operation is done at the node, it will not change its description for any smaller number. This means that for each $e \in E$ all different descriptions can be passed over the edge and only once. This is bounded by the highest number for each edge that exists in the graph at time t, according to lemma 2. □

Lemma 4 *Assume there are no more changes in the network after time* t. *Also assume that after* t, *none of the responsible nodes receives an UPDATE messages which describe one of their outgoing edges and has a higher description number than the one in their database. Then, there is a time after which no more messages are sent or accepted.*

Proof Let t_k be the later time between t and the last time any node received a new UPDATE message (with higher description number) which describes edges at distance k or less, from it. At this time, the node knows the state of all the edges that are outgoing from distance smaller or equal to k. ($t_0 = t$)

Note that for the simple case that there are no DELETE messages, the property of this lemma was proven by Lemma 3. It will be proven that there exists t_k for any k and therefore there is a time after which no UPDATE messages are sent. After this time no more DELETE message are sent since DELETE messages are only sent as a response of the reception of an UPDATE message. Therefore, there is a time after which no more messages are sent. The existence of such t_k will be proven by induction over the distance from all responsible nodes.

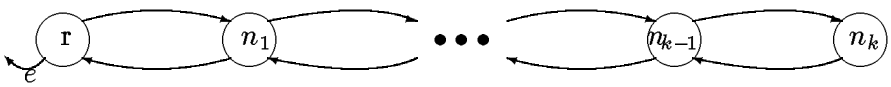
Let us examine the base of the induction. After time t_0, all nodes at distance 0 (the responsible node itself) from an arbitrary responsible node r will not send any message that includes information of an edge description originated in r and will not delete these entries.

Let us look, without loss of generality, at a responsible node r and at any of its outgoing operational edge e. The responsible node r sends a message with information about e under the following events.

- Change in the state of e.
- Arrival of an UPDATE message for e to r.
- Arrival of an UPDATE message for other edges to r which due to change in the local topology database will cause sending DELETE message regarding e.

If this happen after time t_0 then the first two cases are in contradiction to the basic assumption of the lemma. The third case can not happen since a responsible node will never erase an entry of an operating outgoing edge.

Let us examine the step of the induction. The induction assumption is that there exists a time t_{k-1} where all nodes at distance $k-1$ from a responsible node r will not send any message that includes information about r outgoing edges and will not delete the entries of these edges. We need to prove that there exist time t_k for which all nodes at distance k from r will not send any message that includes information for r outgoing edges and will not delete these entries.



First we prove for $k \leq \mathcal{R}$. Let us observe, without loss of generality, at a node n_k which is at distance k from r, at a shortest path $n_k, n_{k-1}, \ldots, n_1, r$ and at the edge e that is outgoing from r. The lemma assumption is that after t, the responsible nodes are not getting UPDATE messages with higher description numbers for their outgoing edges. At Time t_{k-1} all nodes have final information about nodes that are at distance of *k-1*. At the latest, at time t_{k-1}, this final information is sent from such n_k to all the nodes at distance k. Let's mark the time when all this information arrives as t_k, where $t_k \geq t_{k-1}$. Note, that at time t_k, the information about e at n_k is at the same or higher description number. After t_k, no new information regarding e can arrive at n_k, since arriving of such information to n_k will cause sending it to n_{k-1} (which is at distance *k-1* to e) and it will imply that n_{k-1} will get new information after t_{k-1}. From the above and the induction hypothesis we also know that after time t_k, n_k has updated information of all the edges along the path $r, n_1, \ldots, n_k$ and of edge e (which are closer to it). Therefore, n_k will not delete these entries, the entry for e, nor send a DELETE message through one of its outgoing edges. Note, that at time t_k, if a node is at distance greater than k it can not appear in node local database as at distance of k or less. Since, looking at the closest node of all the nodes that hold this characteristic n_l $(l < k)$ and at the n_{l-1} that is one closer to the responsible node r, it can be seen that n_{l-1} will have a correct description of r outgoing edges and it will pass it to n_l. This implies that n_l has an updated description of all the path to r.

For $k > \mathcal{R}$. At time $t_{\mathcal{R}}$ any node knows all the edges at distance $\mathcal{R}$. As shown above, after this time all the nodes will not send an UPDATE message for nodes that are at distance smaller or equal to $\mathcal{R}$. According to the protocol, a node can not send an UPDATE message for responsible nodes that are at distance larger than $\mathcal{R}$. Therefore, no more UPDATE messages will be sent after $t_{\mathcal{R}}$. A node may receive an UPDATE message that was sent before $t_{\mathcal{R}}$ and it may send, in response, a DELETE message for all the entries that are at distance larger than $\mathcal{R}$. This DELETE message is for entries that are outside of $\mathcal{R}$, since all nodes have full information about all the nodes inside the vicinity. These DELETE messages will not be replied with any message since this means that after which is a time $t_{last} \geq t_{\mathcal{R}}$ where no more UPDATE or DELETE messages are sent or accepted. $\square$

Proof (theorem 1) From lemma 4, we can see that if none of the responsible nodes receives an UPDATE message for its own outgoing edges then, there is a time where no more messages are sent or accepted. These UPDATE messages must arrive from any other node in the graph. But this description was sent in the past from the responsible node. If we use lemma 2 we can see that this description number is limited[7]. This means that an UPDATE message will arrive to a responsible node only limited number of times. If no new UPDATE message arrive until t_{last}, then lemma 4 will hold. Else, during the time between t and t_{last} an UPDATE message will arrive and there will be a new t_0 and therefore a new t_{last}. After the final time when an UPDATE message for any of the edges in all the responsible nodes arrives, there are no more messages sent

[7] If we use lemma 2 for $t = 0, \max_e(0) = 0$ we obtain since the first establishment of the edge and after $x_e(0)$ changes and $Cr_e(0)$ crashes, the highest number can be $x_e(0) + Cr_e(0)$.

or accepted over any of the edges. Also, according to lemma 4, all the nodes have an updated description of all the edges inside their vicinity. □

A.2 Outer algorithm

Lemma 5 *Let t_0 be the time after which no topological changes occur. A finite time after t_0, every node at distance $\mathcal{R}$ hops from an arbitrary responsible node* r*, will publish itself as border node of order 1 for* r*, and no more messages regarding border node of order 1 for* r *will be sent or accepted.*

Proof: According to theorem 1 of the inner protocol, there is a time ($t_1 > t_0$) after which all nodes within the vicinity of an arbitrary node r, know their distance from r. This includes all the nodes at distance of exactly $\mathcal{R}$ hops. Therefore, according to line 3 in CHECK_BORDER, node B that is at distance of $\mathcal{R}$ from r, will publish itself as a *border node of order 1 for r.* After a finite time, according to theorem 1, all the nodes within B's vicinity will receive the information that B is a border node of order 1 for r. Similarly, at t_1 every node which published itself as a border node of order 1 for r, and it is not at distance of $\mathcal{R}$ will publish to all the nodes within its vicinity that it is no longer a border node of order 1 for r. Therefore, after a finite time from t_1, nodes which now are at distance of exactly $\mathcal{R}$ hops from r will finish publishing themselves and no more messages regarding border nodes of order 1 for r will be sent or accepted. □

Lemma 6 *Let t_0 be the time after which no topological changes occur. A finite time after t_0, every node at distance of $\ell \cdot \mathcal{R}$ hops from an arbitrary responsible node* r*, will publish itself as border node of order ℓ for* r*, and no more messages regarding border node of order ℓ for* r *will be sent or accepted.*

Proof: Let us look, without loss of generality at one responsible node r. Let's prove the lemma by applying induction on the distance from r. The base of the induction is proven at lemma 5. According to lemma 5 all the nodes, in radius less or equal to $2{\cdot}\mathcal{R}$ will know r since all the border nodes of order 1 for r have published themselves.

The step of the induction assumes that a node at distance less or equal to $(n-1){\cdot}\mathcal{R}$ knows at least one *border node of order n-1 or less for r* and no more messages regarding *border node of order (n-1) for r* are sent or accepted. Let's look at an arbitrary node n at distance of $n{\cdot}\mathcal{R}$. Within n vicinity there is at least one node v that is at distance of exactly $(n-1)\cdot\mathcal{R}$ from r since n vicinity radius is $1\cdot\mathcal{R}$. v knows its distance from r according to the assumption. Therefore, v must have published itself as a border node of order (n-1) for r and since n is inside v's vicinity, n must know that v is a border node of order (n-1) for r. n must publish itself as a border node of order n for r to all the nodes within its vicinity. Since the number of such nodes is finite, after a finite time, all these nodes will publish themselves as border node of order n for r and no more messages regarding border node of order n for r will be sent or accepted. $\square$

Theorem 5 *After a finite time since the last edge topological change or the last node crash, no more messages are sent or accepted and every node has inside its vicinity at least one border node for every node in the graph.*

Proof According to lemma 6, every node at distance of $\ell\cdot\mathcal{R}$ hops from an arbitrary responsible node r, will publish itself as border node of order ℓ for r. This is true for all the nodes in the graph, therefore, if an arbitrary node k is at distance $n\cdot\mathcal{R}$ from r, it will become a border node of order n for r. After a finite time, all the border nodes in the graph will finish to publish themselves and all the nodes in the graph will have at least one border node of any order inside their vicinity. $\square$