

LINK REVERSAL ROUTING

Miia Vainio
P.O.Box 407,
00045 NOKIA GROUP
Miia.Vainio@nokia.com

Abstract

The routing protocols that are used in ad hoc networks must converge quickly, save power and be scalable and adaptive. Link reversal routing (LRR) protocols are one possible choice for ad hoc networks. LRR protocols try to localize the effect of topology change and react only when necessary. There exist three LRR algorithms: Gafni-Bertsekas (GB), Lightweight Mobile Routing (LMR) and Temporally-Ordered Routing Algorithm (TORA). The paper describes the principles of these three algorithms. GB is the oldest one but it has a convergence problem in case of partitions. LMR is more stable. It has two phases; The routes are established and maintained. TORA tries to combine the good qualities of GB and LMR. Finally the paper deals with the performance of the algorithms compared to each other and also to some other algorithms related to ad hoc network routing.

1 Introduction

People are getting use to transferring information also when moving from one place to another. Usually transmission requires the help of a network with some fixed network elements. However, it is not worthwhile to build this kind of infrastructured networks everywhere.

Ad hoc networking enables communication between mobile nodes without any fixed infrastructure. [1,2, 3] Ad hoc networks are meant to be formed temporarily by nodes close to each other whenever connections are needed. Fixed routers are not used because every node can act as a router. Thus, if a node has not got a direct connection to the destination, multihopping via other nodes is possible. The situation, where ad hoc networks are formed, can be for instance a group of people in a conference room or soldiers in a war.

The characteristics of ad hoc networks set some important requirements to the routing algorithms used. Because the nodes can move around all the time, the topology of the network may change often. Some nodes change place and get new neighbor nodes, some nodes move beyond the network and new nodes appear to the network. The routing protocol must converge quickly in order that the network has time to stabilize before new changes. If the time between changes is shorter than the time to converge towards a stable state, it may lead to problems, e.g. to loops. Good routing algorithm is loop-

free. In addition, the protocols should avoid unnecessary computation and control info transmissions. The ad hoc nodes are likely little wireless devices that have batteries. Transmissions and computation consume power. Also the limitations of the bandwidth resources must be taken into consideration when choosing a routing protocol. Because of all these limitations ad hoc networks are not the easiest environment for routing algorithms. Algorithms more suitable than conventional routing algorithms must be developed and used.

The ad hoc routing protocols can be divided into two groups. Proactive, i.e. table-driven, algorithms try all the time to keep track of the changes happening in the network. The nodes must be able to store tables containing information about the routes. All the nodes know the whole time how to send packets to every other node in the network. However, because of the rapid movement of the nodes, routing information becomes obsolete quickly. Whenever a change occurs, new routing information must be distributed throughout the whole network. Routes must be always defined although they would never be used. Thus, proactive routing algorithms waste bandwidth. The useless transmissions and computation of routes consume extra power. The increased control traffic can also lead to congestion. However, they also have one advantage. Because the routing information is always ready, packets can be sent immediately with minimal delay.

Reactive, i.e. on-demand driven, routing protocols search the route between nodes only when it is really needed. When a node has something to send, it initiates the procedure of finding the route to the destination node. Thereby, the amount of extra control information decreases compared to proactive algorithms and not so much memory space is needed for routing tables. However, the discovery of routes takes time. The delay before the beginning of the data transmission is longer than if the routes had been established beforehand.

Both the algorithm types have their advantages and disadvantages. Proactive algorithms suit better for networks with a great deal of traffic, strict delay requirements and not so much node movements while reactive algorithms are better when nodes move a lot but there is not very much traffic between nodes.

This paper concentrates on link reversal routing, which is one subtype of the ad hoc routing. Link reversal routing belongs to neither proactive nor reactive routing

types, because the algorithms can be used in both modes. The paper presents three algorithms. Chapter 2 concerns the features common to all of them. The algorithms are described from Chapter 3 onwards. Chapter 6 deals with the performance characteristics of the algorithms.

2 Main features of link reversal routing

Link Reversal Routing (LRR) is developed for networks whose topology is changing so fast that the conventional routing algorithms are not working properly anymore but still the change is not so fast that flooding would be the only possibility. [1] LRR protocols do not necessarily give the most optimal route from the source node to the destination but it does not matter because in this kind of situation, it is more important to have any correct route at all. The adaptivity and scalability of the LRR protocols make them very suitable for ad hoc networks.

LRR describes the network of nodes with Directed Acyclic Graph (DAG), which is a graph with directed arcs. Acyclic means that the graph has no loops. The graphs of LRR have exactly one node, which has only incoming links i.e. the node has only upstream neighbors. Other nodes have either incoming and outgoing links or just outgoing links. The node with only incoming links is the destination node. Every route of the DAG finally leads to the destination. Of course in order to these rules to be valid, the network must be in its stable state. Loops can exist momentarily during the convergence. Figure 1 depicts the concepts related to DAG.

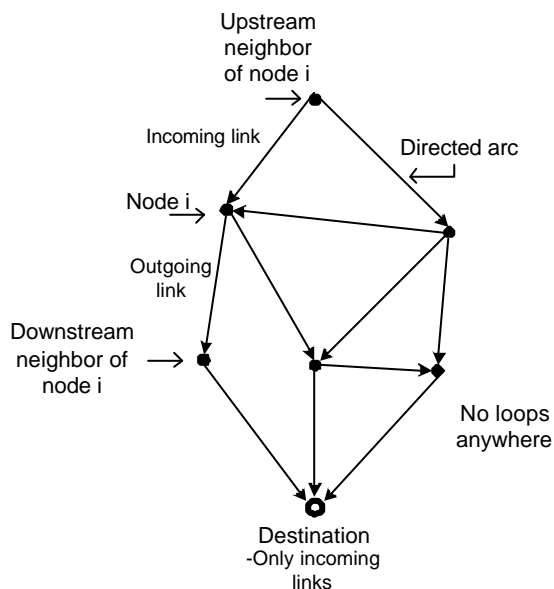


Figure 1. A directed acyclic graph

The main ideas of LRR are to localize the effect of topology change and to react only when it is necessary. If a link breaks, it does not automatically cause

reactions. Only if the broken link was the last outgoing link of the node, the node must start route maintenance. Even then, the information about the change is transferred only to nodes whose every path going to the destination included the broken link. Only the nodes, which have to check the direction of their links to stabilize the DAG again, notice that something changed. This kind of approach decreases the routing overhead. It is not even necessary for the node, which lost its last downstream link, to look for a new route to the destination if the route is not required for the node at that moment. The node must however inform others that routes through it cannot be used anymore. This is possible because of multiple routes. All routes are not always needed. Because every working route leads to the same destination, it does not matter if few of them break if there are others that can still be used.

However, because the information that the nodes have about the topology has been minimized, a node does not know its place related to other nodes in the network. It only knows its neighbors. The node does not know how far the destination is or what nodes are between it and the destination. This makes the route optimization difficult.

All the described algorithms are loop-free and deadlock free and establish multiple routes to one destination. If routes to multiple destinations are needed at the same time, the algorithm, which is used, can be executed to every destination separately, independently of the other executions.

3 The Gafni-Bertsekas algorithm

The Gafni-Bertsekas (GB) algorithm is the oldest one from these three algorithms. [1] It is quite simple. The main idea is to always keep a directed path from all the nodes to the destination. This is done by making sure that no node, other than the destination, exists in the DAG that have only incoming links. This checking is (or at least was originally) done proactively. Routes from other nodes to the destination are maintained all the time.

There are two methods to handle the nodes with no outgoing links. In case of full reversal method, those nodes reverse the direction of all their links. Thus, after that the nodes have only outgoing links. The other method, partial reversal method, is almost the same, but the nodes do not reverse the direction of all their links. Every node, except the destination, keeps a list of the reversed links between it and its neighbor. When a neighbor of a node reverses the direction of the link between the nodes, the node writes the name of the neighbor down. As the node loses its last outgoing link, all other links than the already reversed ones are reversed. After that the list is emptied. If all the links have been reversed i.e. the list contains all the neighbor

nodes, all the links are transformed from incoming to outgoing and the list is emptied. The reversals are informed by sending an update packet to the neighbors. If the update causes the neighbor to lose its last outgoing link, the reversing transaction continues with that node. The partial reversal method is usually more efficient than the full reversal method.

Figure 2 depicts the progress of full reversal GB algorithm. In a), a link between the destination and one of its neighbor breaks. The node that lost its link to the destination reverses all its links and broadcasts an update message (U) to its neighbors. The reversing of the link causes other links to be reversed. In e), no node loses its last downstream link anymore when receiving an update message.

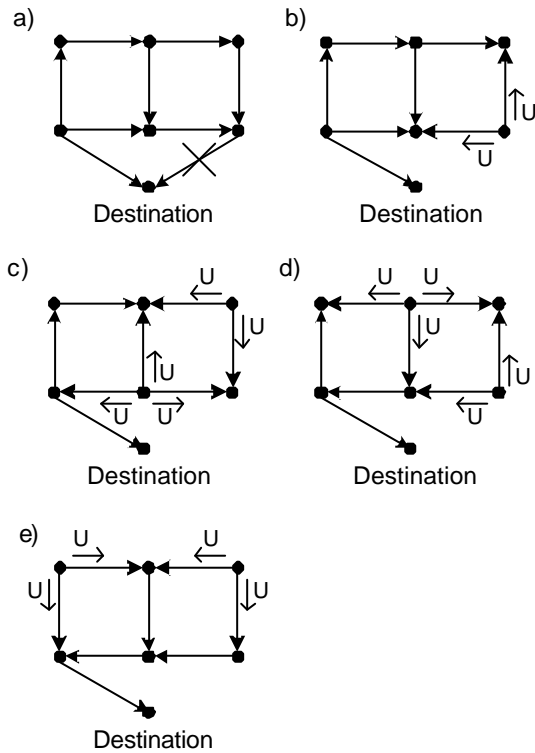


Figure 2. The full reversal GB algorithm

Figure 3 gives an example how the algorithm works in partial mode. When a link breaks in a), the neighbor node loses its last downstream link. It reverses its links and broadcast an update (U) message. Some neighbors lose their last downstream links because of the update. Those nodes reverse the links that are not in their list of reversed links and broadcast an update to their neighbors. For example in c) the node that points to the destination receives an update but because it still has downstream links after the update, it does not have to react to the change. Finally after e) the DAG is again in stable state. The only node that does not have downstream links is the destination.

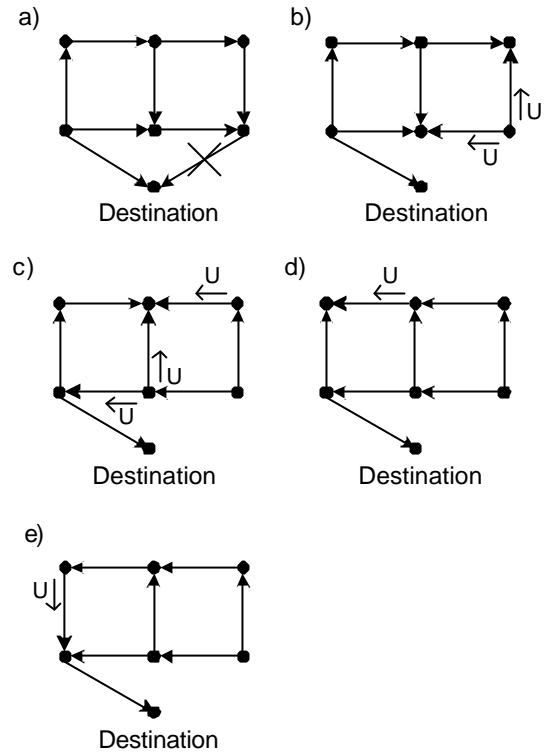


Figure 3. The partial reversal GB algorithm

The partial reversal method described above is list-based. Partial reversal method has also height-based form. In height-based form every node i has a triplet (α, β, i) . α and β are integers. α represents the reference level and β and i the relative height. The links are directed according to the relative heights of the neighbors. The set of triplets are ordered lexicographically. Lexicographically means that for instance if there are two pairs (a,b) and (c,d), then (a,b) > (c,d), if $a > c$ or if $a = c$ and $b > d$.

In the beginning the reference level α is 0 for all nodes and for all nodes i and j , it is known that $(\alpha_i, \beta_i, i) > (\alpha_j, \beta_j, j)$ if and only if the link between them is directed from i to j . The values of the triplet are defined again when a node notices that it does not have downstream links anymore. In that case, the node has become a local minimum and it tries to correct the situation by choosing a bigger value for α than what is the minimum value of its neighbors. That way, the node reverses the direction of a link. Let j be the neighbor of i and that $(\alpha_i, \beta_i, i) > (\alpha_j, \beta_j, j)$. If a node i (other than the destination) loses its last downstream link, then the values got (with the k th iteration) for α and β of the node i are defined:

$$\alpha_i^{k+1} = \min\{\alpha_j^k \mid j\} + 1$$

$$\beta_i^{k+1} = \begin{cases} \min\{\beta_j^k \mid j, \alpha_i^{k+1} = \alpha_j^k\} - 1 & \text{If there exists a neighbor } j \text{ with } \alpha_i^{k+1} = \alpha_j^k \\ \text{else} & \end{cases}$$

$$\beta_i^{k+1} = \beta_i^k.$$

Other nodes do not change their values of α and β .

The height-base form is described in Figure 4. The little arrows depict the update messages transmitted to inform that the direction of the link and the height value of the sending node have been changed. For instance the first changed height value has been counted by taking the minimum α value of the neighbors (here all has 0) and incrementing it by one. The value of β does not change because all the α values of the neighbors differ from the new value of α_i . In c), the rule $\alpha_i^{k+1} = \alpha_j^k$ is fulfilled and the new value (=0) of β is the minimum value of its neighbors' β value minus 1.

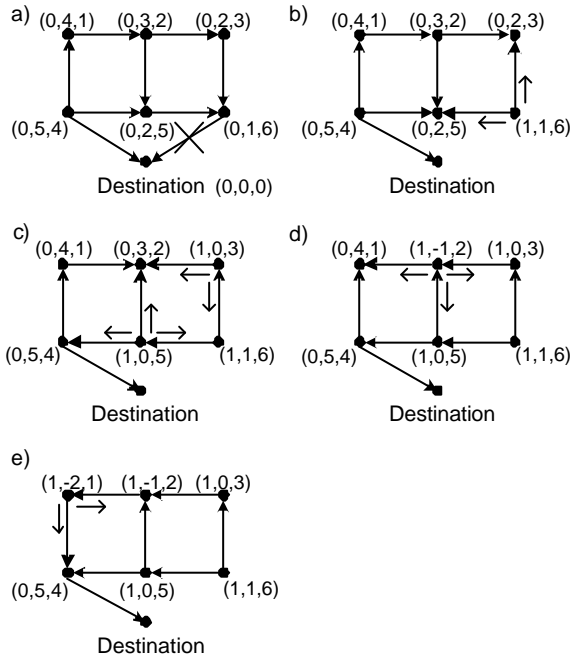


Figure 4. The height-based form of the partial reversal GB

The GB algorithms are deadlock free and loop-free and multiple routes can be maintained due to the use of DAGs. However, if some part of the network is disconnected from the destination, the GB algorithms prove to be unstable. The nodes will send control packets and message packets until the part is reconnected. This unstable behavior is a significant disadvantage because in case of ad hoc networks this kind of partitioning is probably quite common. If no partitioning occurs, the GB algorithms convergence after a finite time. [1,4]

4 Lightweight mobile routing

Because of the instability of the GB algorithm, better algorithms must be developed. The Lightweight Mobile Routing (LMR) is one candidate. [5] It is a reactive algorithm i.e. routes are established to the destination

only when necessary. Like GB, LMR is deadlock free and loop-free, but LMR is more stable.

The LMR protocol can be divided into three separate phases. First the required routes must be built. That is called a construction phase. As changes happen in the topology, some routes must be reestablished (maintenance phase). Finally the routes are not needed anymore and the route destruction phase begins. However, because it can be assumed that the topology of ad hoc networks change quite frequently, which causes the invalid routes to be removed, separate destruction phase is not really needed. The maintenance phase takes care of the deletion of invalid routes. Thus, the two important phases of the LMR protocol are the construction and the maintenance phases.

4.1 Construction phase

Initially the network consists of nodes and undirected links between the nodes. The nodes know their neighbors, but they do not know anything else about the topology of the network. A node can transmit three kinds of control packets to its neighbors to enable routing: query (QRY), reply (RPY) and failure-query (FQ). The control packets are sent on a control channel that is different from the channel, which is used to carry the actual data packets. When a control packet is transmitted, it is put to a single packet transmission queue. If a new packet should be set to the queue even though there is still an old packet in the queue, the old packet is discarded and the new packet is sent instead of it.

The construction phase begins when one node has something to send to some other node. All the links, besides the links connected to the destination, are undirected. Figure 5a describes the initial state. The source node does not know where the destination is located or which way it should send a message. The routes must be built before real data transmission. The source node (node i) sends a QRY packet (Q in the figure). This packet is flooded through the network. When a node receives the QRY packet, it broadcasts the packet ahead to its neighbors. However, every packet is only sent once by every node. The node remembers the packet IDs of the packets it has received or sent

If the destination can be reached, the QRY packet is at some point broadcast to the neighbors of the destination. In the Figure 5 the neighbor is node k . Because the neighbors already know the route to the destination, they inform it by broadcasting an RPY packet (marked with R in the figure) back to the link the QRY packet came from. The unsigned links, through which the RPY packet passes, are converted to directed links. The direction is from the receiving node to the node that sent the RPY. Hence, as the response reaches back to the source node,

the routes from the source to the destination are ready and the construction phase is over (h.) in the figure). Nevertheless, if a node does not receive the response within a reasonable time, a timer expires and the node can send a new QRY. This can be repeated until the node finally gets a response or it does not need the route anymore.

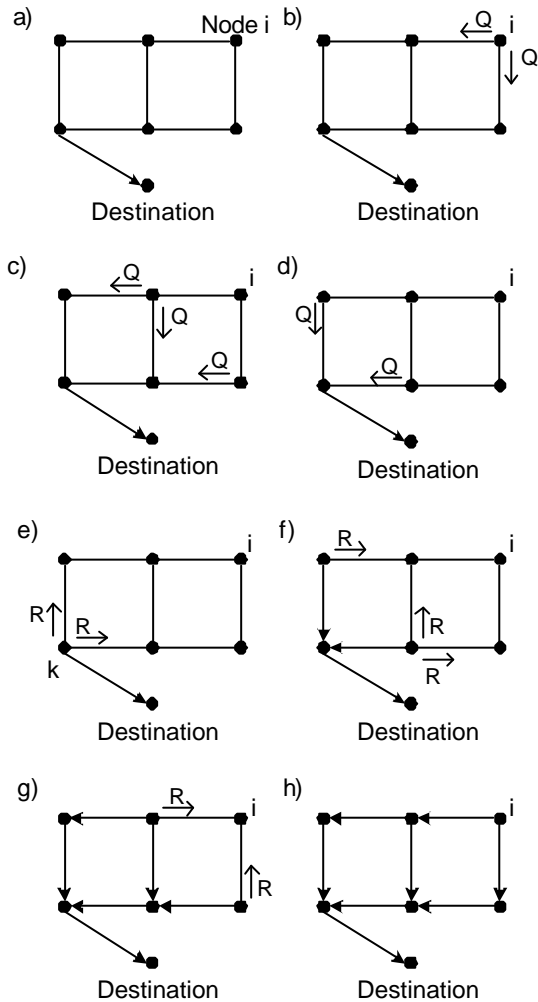


Figure 5. Route initialization in case of LMR

4.2 Maintenance phase

After the initialization the network can start data transmission. Maintenance is not needed as long as all the nodes have at least one route to the destination. The loss of the last route triggers a maintenance reaction. If the node that lost the route has an upstream neighbor, the node must broadcast an FQ packet in order to make sure that the invalid route is deleted. The packet informs to the neighbors that no valid route exists anymore through the node to the destination and that the node requires a new route. The invalid route is eliminated by changing the link between the nodes from a directed link to an

undirected link. The neighbor checks if it still has an alternative route after the erase of the invalid route. If there are other routes, it transmits an RPY packet back to the node; otherwise it forwards the FQ packet to its upstream neighbors. The sending of an FQ packet upstream continues until some node responds that it has a route to the destination or all the invalid routes have been erased and still no route has been found. No route can be found if the part of the network has been separated from the destination. Then, all the links of the partition becomes undirected and the node desiring the route broadcasts QRY from time to time to check if the part has been connected back to the rest of the network.

If the node receiving a FQ message or losing the last downstream link, does not have upstream neighbors at all, the node can choose to send a QRY requiring a route itself or if it does not need a route, it can do nothing.

Figure 6 depicts a link failure and how the maintenance phase of LMR corrects the loss of a route. Node j loses its routes to the destination in a). The node transmits a FQ packet to its neighbor. The neighbor k has another route to the destination. Thus, k sends a RPY (i.e. R) packet back to the node j . The direction of the link between the nodes is changed and after that j has a route to the destination again.

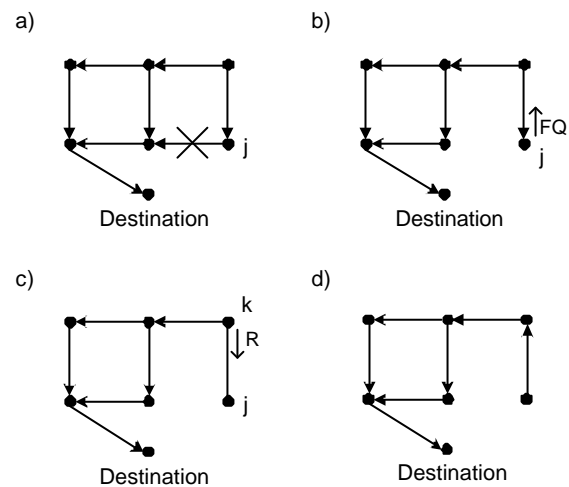


Figure 6. An example of a link failure and LMR maintenance phase

In order to ensure stable reaction in all cases, there are some special rules for some isolated situation. In these special cases, the links can also get other states (downstream-blocked, unassigned-waiting and awaiting-broadcast) than unassigned (i.e. undirected), downstream and upstream. In case of e.g. dynamic topology, a downstream blocking rule must be introduced to avoid loops. There is also a specific rule to prevent deadlocks. These rules are more carefully explained in [5].

The basic LMR algorithm does not make it possible to optimize the route. One or more routes are established but the source cannot know which one of them is the shortest path to the destination. However, by adding a field, which is increased by every hop, into the RPY packets the length of the routes can be estimated. The nodes can store this value and check it when they have something to transmit. Nevertheless, because the path length estimations are only transmitted inside RPYs and no other maintenance is available for the values, they do not always tell what really is the shortest route.

The LMR protocol also has some bad features in case of partitioned networks. “False reply” propagation can sometimes create temporarily invalid routes. The invalid routes will be removed, but a strict time limit to that cannot be given. [4]

5 Temporally-ordered routing algorithm

Temporally-Ordered Routing Algorithm (TORA) is based on both the GB algorithm and the LMR algorithm. [4, 6] TORA utilizes a height metrics such as the GB protocol. The request process and three control packets (query (QRY), update (UPD) and clear (CLR)) are similar to the LMR protocol. However, TORA has a separate erasing phase. The goal has been to correct the problems occurring with the GB and LMR protocols. TORA reacts to topology changes and partitions more efficiently and erases the invalid routes within a finite time.

The height value of a node consists of five components, (t, oid, r, d, i) . The first three are related to the reference level used. t is the time when the reference level was created. The nodes are expected to be synchronized. This can be achieved e.g. with GPS (Global Positioning System). All nodes are assumed to have a unique id. Oid is the id of the node, which set the new reference level. The reference level can be divided into two sublevels by using r , which is only one bit value. The remaining two components describe the height values related to the reference level. d describes the height value respect to the reference level. Finally i makes sure that the nodes can always be ordered lexicographically. i is a unique identifier of the node. In addition to its own value, the node also keeps a list of the height values of its neighbors. The direction of a directed link is from the higher node to the lower. The height of the destination is always zero $(0, 0, 0, 0, dest_id)$. In the beginning the height of all the other nodes is NULL $(-, -, -, -, id)$, which describes the value of infinity, and the links are undirected.

5.1 Construction phase

A node with undirected links wants to get a route to another node (Node i in Figure 7b). It broadcasts a QRY packet to its neighbors. All nodes have a route-required flag, which is set to 1 when sending a QRY. If the receiving node does not have a route to the destination and if its route-required flag is not set, the node forwards the query and sets its flag. (In Figure 7 the nodes with route-required bit set are marked with white node.) If the node has no route but the flag is set, it discards the packet. In case the receiving node has a route (i.e. it has at least one downstream node) and its height is NULL, it sets its reference height according to the reference height value of its downstream neighbor, which has the minimum non-NULL height value, but increases the d value by one compared to the value of the neighbor (Look the picture e, the neighbor node of the destination). The node also broadcasts a UPD packet (marked as U in the figure). If its own height is not NULL, the node compares the transmission time of the last UPD and the time when the link, through which the QRY was transmitted, became active. If the time of the UPD packet transmission is later, the QRY packet is discarded; otherwise new update is sent.

When a node gets a UPD packet, it first updates the list containing its neighbors' heights. Then if the node has a set route-required flag, it updates its height according to the minimum non-NULL value of its neighbors (just as described earlier), updates the states of its links, unsets the flag, and broadcasts UPD. If the flag is not set, only the states of the links are updated.

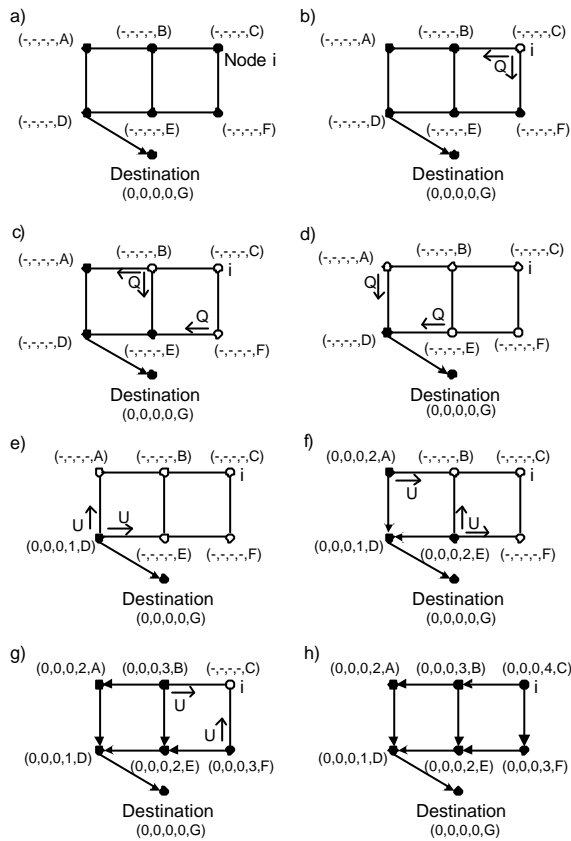


Figure 7. The initialization of routes in TORA

5.2 Maintenance phase

Nodes that have height NULL are not considered when defining the height values and no height maintenance is performed for them either. There are five reactions to the situation when a node i does not have a downstream link. In the first case the node does not have downstream links because of a link failure:

1. The node defines a new reference level and becomes a global maximum. It has been assumed here that the node has upstream neighbors. If that is not the case, the height is just set to NULL.

$$(\mathbf{t}_i, oid_i, r_i) = (t, i, 0), \text{ where } t \text{ is the time of the failure}$$

$$(\mathbf{d}_i) = (0, i)$$

If the node loses a link that is not its last downstream link, information related to the link and the neighbor is just removed from the node's lists.

Other cases occur when the node loses its last downstream link when it receives a UPD packet and reverses its links. The UPD packet causes the states of the links and the heights of the neighbors to be updated. The neighbor of the node i is here j . The four possible reactions are:

2. The reference levels of all the neighbors are not the same. The node chooses the maximum reference level and select a height lower than all the other heights of that reference level.

$$(\mathbf{t}_i, oid_i, r_i) = \max\{(\mathbf{t}_j, oid_j, r_j) \text{ of all neighbors}\}$$

$$(\mathbf{d}_i) = (\mathbf{d}_m - 1, i), \text{ where } \mathbf{d}_m \text{ is the minimum height value related to the reference value.}$$

3. The reference level is same for all neighbors so that the value of r is 0. r divides the reference levels into two sublevels: the original reference level and the higher reflected reference level associated to the original reference level. The node "reflects" back the higher reference level (higher because r is 1, not 0) towards the node that created the reference level. If the reflected level is propagated back to its origin from all of its neighbors, no route to the destination can be found.

$$(\mathbf{t}_i, oid_i, r_i) = (\mathbf{t}_j, oid_j, 1)$$

$$(\mathbf{d}_i) = (0, i)$$

4. The reference levels of the neighbors are the same and $r = 1$. Also the reference level is defined by the node i itself. Thus, the reflected level has reach the origin and no route could be found. The node must initiate the route erasing.

$$(\mathbf{t}_i, oid_i, r_i) = (-, -, -)$$

$$(\mathbf{d}_i) = (-, i)$$

5. The reference levels are the same with $r = 1$ and the originator of the reference level is not i . This does not yet necessarily mean that partitioning has happened. However, the node has experienced a link error (no reaction was required for it) at time t . The node defines a new reference level.

$$(\mathbf{t}_i, oid_i, r_i) = (t, i, 0)$$

$$(\mathbf{d}_i) = (0, i)$$

In cases 1, 2, 3 and 5 the node updates the states of its links and sends a UPD packet to its neighbors.

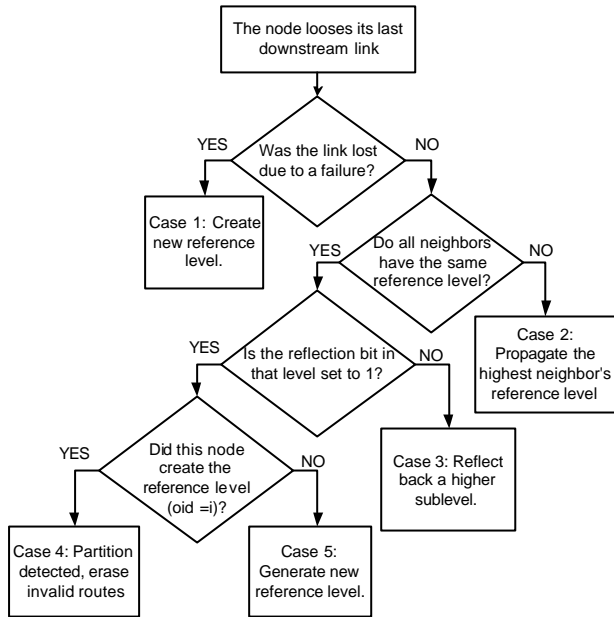


Figure 8. A flowchart describing the TORA maintenance phase. [4]

5.3 Route destruction

When the reflected reference level propagates back to its originator (case 4) and thus the originator knows that the networks has been partitioned, invalid routes must be destroyed. The node sets its own and its neighbors' height to NULL, updates the link states and broadcasts a CLR packet. The CLR packet includes information about the reference level. When a node receives the packet, it checks if the receiving node uses the same reference level than the node that sent the CLR packet. If the level is the same, the node clears its own and its neighbors' height values, updates link states and forwards the CLR packet. If the reference level in the CLR packet is different from the level of the receiving node, the node sets the heights of all the neighbors with the same reference level than the packet to NULL and updates the link states. If this makes the node to lose its last downstream link, it reacts as in case 1 of the maintenance phase.

TORA enables a highly adaptive loop-free routing, but as in case of other LRR protocols, optimization is the problem. In the beginning, when the reference level is 0 for all the nodes, by comparing the values of d the node can determine the shortest path. However, after the creation of new reference levels, this is not the case anymore. The problem can be eased by periodically sending refresh packets from the destination. Thus, the destination starts this process. The refresh packet sets the reference levels of all the nodes to 0 and sets d to its valid value. This transmission should be carefully controlled and it should occur at a very low rate. This

approach is the proactive mode of TORA, while the approach presented earlier was the reactive form.

Another problem can follow from the need of synchronized clocks. However, clocks do not necessarily mean physical clocks. It is sufficient to use logical clocks e.g. counters in order to be able to define the order of the events. In addition, in some situations oscillatory behavior can occur with TORA [1].

TORA has been under development in the IETF (Internet Engineering Task Force) MANET (Mobile Ad hoc Networks) working group [6].

6 Performance comparisons

There are not so many simulations available that compare the link reversal algorithms to each other or to other ad hoc algorithms. However, some comparisons can be made. More accurate results, simulation models and parameters can be read from the reference material.

6.1 Lightweight mobile routing vs. Gafni-Bertsekas

Simulation results presented in [5] show the difference in performance between LMR, GB and flooding. For LMR the simulations were made for both the basic algorithm and for the algorithm with the possibility to use also distance information carried in the packets. The obtained results were that for heavy traffic the LMR algorithm is always better than GB. At heavy traffic, the traffic channels are very congested and the algorithm that routes the messages most efficiently is the best. For light traffic conditions the order is from best to the worst: LMR (with distance metric), GB and LMR (basic) if the rate of change is low. Whilst in case of high rate of change and light traffic, GB outperforms both LMR modes. As the traffic is light, the most important thing affecting the performance is, how fast the protocol can find the routes and transmit the data messages. There is no congestion. If the topology changes slowly, it is not very important how fast the algorithm can react to the changes. Finding the shortest route is more important. Thus LMR with distance metric is the best algorithm. When the change rate increases, the algorithm used should react quickly. GB outperforms LMR because of its lower complexity. All the protocols perform better than flooding until the rate of change is so high that only flooding can be used. [5]

6.2 Performance of temporally-ordered routing algorithm

Worst-case protocol complexities for link reversal algorithms and some other routing algorithms are compared in [4], which claims that TORA has better worst-case complexity than many other algorithms close

to it. In addition TORA, GB and LMR do not necessarily have to react to link additions, which decreases the complexity compared e.g. to Ideal Link-State (ILS) routing algorithm even more. However, worst-case comparison does not give very clear and reliable results about the features of the algorithms. Thus, not too much weight should be set to these results. Table 1 gathers some of the results together. The Time Complexity (TC) is the number of steps required to perform a protocol operation and Communication Complexity (CC) means the number of messages exchanged in performing the operation. The GB mode is here full reversal. [4]

Table 1. Worst case protocols complexities.

$|L|$ =the number of network links, d =the network diameter, x =the number of nodes affected by a topological change, l =the length of the longest directed path in the affected network segment, D =the maximum nodal degree

Protocol	TC	CC
ILS	$O(d)$	$O(2 L)$
GB(connected, postfailure)	$O(2l)$	$O(lDx)$
GB(disconnected, postfailure)	∞	∞
LMR(connected, postfailure)	$O(2l)$	$O(2Dx)$
LMR(disconnected, postfailure)	$< \infty$	$< \infty$
TORA(connected, postfailure)	$O(2l)$	$O(2Dx)$
TORA(disconnected, postfailure)	$O(3l)$	$O(3Dx)$

No simulation were found, where TORA would have been compared to LMR or GB. But simulations comparing the performance between TORA and some other ad hoc routing protocols are available. Corson and Park [7] executed simulations to TORA and ILS routing. The performance was measured based on bandwidth utilization efficiency, mean message packet delay and message packet throughput. The effects of network size, topology change rate and average network connectivity were studied. The network connectivity does not affect very much the performance of the protocols. If the bandwidth is kept the same, when the size of the network or the rate of changes increase, the performance of TORA eventually becomes better than the performance of ILS. Especially, the control overhead of ILS increases more quickly causing congestion and queuing delay for message traffic. Finally, more bandwidth is used for control overhead than for data. ILS is a shortest path algorithm. TORA is not. Thus, under some conditions TORA may outperform a shortest path algorithm. Actually, it was claimed in [7] that because of the control overhead that is needed for all shortest path algorithms, it could be assumed according to the simulations, that under some conditions TORA performs better than any shortest path algorithms. However, no proofs were given. [7]

If TORA is compared to Dynamic Source Routing (DSR) and Ad hoc On-demand Distance Vector

(AODV) in a situation when they are implemented over 802.11-based link layers, TORA performs poorly. This is simulated in [8]. The performance was compared according to packet delivery ratio, routing overhead and path optimality. In terms of routing packet overhead, the performance of TORA was the worst. With 10 and 20 sources, the simulation showed that still over 90 percent of the packets were delivered but with 30 sources, lots of packets were dropped because the traffic could not be handled anymore. DSR and AODV performed well with all the simulated mobility rates and movement speeds. [8] The reason for the bad performance is that TORA requires its routing information to be broadcast reliably to its adjacent neighbors. DSR and AODV require only reliable unicast transmissions. The reliability for TORA is achieved by implementing IMEP (Internet MANET Encapsulation Protocol) layer under TORA. IMEP offers reliable in-order control message routing delivery between the neighbors. IMEP has an effect on the results of the simulation [9]. TORA is not designed to be used above any MAC (Medium Access Control) layer technologies, but probably for e.g. TDMA (Time Division Multiple Access) based networks [1].

Energy consumption is an important feature for ad hoc network routing protocols. The terminals have limited battery sources. Thus, if the protocol requires constant control packet transmissions, the power source of the terminal exhausts too quickly. Another problem is to prevent the heating of the terminal. Cano and Manzoni simulated the energy consumption of four ad hoc routing protocols [9]. The studied protocols were DSR, AODV, TORA and DSDV (Destination-Sequenced Distance-Vector). The simulation environment was the same as in [8] with 802.11 MAC layer and also IMEP layer was used below the TORA. TORA had the highest energy consumption. However, in addition to the TORA control packets, also IMEP packets affected the result. IMEP sends HELLO messages periodically. TORA also proved to be unscalable. The energy consumption increased 518% when the number of nodes increased from 25 to 50.

The last simulation study dealt with here, examined several routing protocols (TORA, DSDV, AODV, DSR etc). [10] This time no link layer details, e.g. MAC protocol, were modeled. The performance was evaluated based on the fraction of packets delivered (number of packets delivered to the destination vs. number of packets sent), end-to-end delay and routing load (bytes of routing packets vs. bytes of data packets transmitted). Again, TORA performed badly. It gave the lowest fraction of packets delivered even though its multipath capability. The reason can be e.g. that the initial route discovery of TORA takes too long. The overhead of finding and maintaining multiple routes seemed to affect more than the benefits of having multiple routes. TORA also had the worst average end-to-end delay and routing

load characteristics. The absence of the distance information affected the delays, and the load was high because invalid routes caused by network partitions always has to be erased in TORA although they would not be used. [10]

TORA does not seem to give very good results in simulations (except in [7]). All the presented simulation studies have however limitations that must be taken into account. For instance, in case of the study described in [7] the model did not have true node mobility and the study in [10] had no transmission errors etc. [10] More simulations are needed.

7 Conclusions

This paper dealt with the link reversal routing. Three algorithms were introduced: GB, LMR and TORA. The goal of the LRR algorithms is to localize the effect of topology change and to avoid unnecessary reactions. All the algorithms use DAGs to describe the topology of the network. The algorithms are loop-free and deadlock free but do not enable route optimization in their basic forms.

GB can be full reversal or partial reversal. The partial reversal algorithm can be list-based or height-based. GB has problems with convergence when a part of the network separates from the rest of the network.

LMR has two phases: the construction phase and the maintenance phase. Basic LMR can be enhanced by adding a distance parameter to response packets. The parameter can be used for route optimization. LMR is more stable than GB in case of partitions, but false replies can temporarily create invalid routes.

LMR and GB were compared in a simulation. In case of heavy traffic, LMR was always better but if the traffic was light and topology rate of change was high, GB performed better than LMR.

TORA has taken features from both the GB and the LMR algorithms. It has three phases: construction phase, maintenance phase and destruction phase. The links are directed according to the node heights. Height values consist of 5 components, which define a reference level and a height related to the level.

The simulations have given slightly inconsistent results. One simulation claimed that eventually TORA outperforms ILS and maybe even all the shortest path algorithms. On the other hand, other simulations present that TORA performs poorly and is unscalable. However, all the simulations had restrictions. No definite conclusions can be done. More simulations must be executed.

References

- [1] Perkins, C. E.: Ad hoc networking, USA, 2000, ISBN 0-201-30976-9
- [2] Royer, E.; Toh C.-K.: A review of current routing protocols for ad hoc mobile wireless networks, April 1999
- [3] Tseng, C.-C.: Introduction to ad hoc mobile wireless networks, Jan 2000
- [4] Park, V.; Corson, M. S.: A highly adaptive distributed routing algorithm for mobile wireless networks, USA, 1997
- [5] Corson, M.S.; Ephremides, A.: A distributed routing algorithm for mobile wireless networks, USA, 1994
- [6] Park, V.; Corson, M. S: Internet draft; Temporally-ordered Routing Algorithm (TORA), version 1, Functional Specification, 2001
- [7] Park, V.; Corson, M. S: A performance comparison of TORA and Ideal Link-State routing, USA, 1998
- [8] Broch J, et al.: A performance comparison of multi-hop wireless ad hoc network routing protocols, USA, 1998
- [9] Cano, J-H; Manzoni P: A performance comparison of energy consumption for mobile ad hoc network routing protocols, Spain, 2000
- [10] Das S. R; Castañeda R; Yan J: Simulation based performance evaluation of mobile, ad hoc network routing protocols, USA, 1998