IETF MANET Working Group
INTERNET-DRAFT
draft-ietf-manet-tora-spec-04.txt

V. Park
S. Corson
Flarion Technologies, Inc.
20 July 2001

# Temporally-Ordered Routing Algorithm (TORA) Version 1 Functional Specification

## Status of this Memo



## Abstract

This document provides both a functional description and a detailed specification of the Temporally-Ordered Routing Algorithm (TORA)--a distributed routing protocol for multihop networks. A key concept in the protocol's design is an attempt to de-couple the generation of far-reaching control message propagation from the dynamics of the network topology. The basic, underlying algorithm is neither distance-vector nor link-state; it is a member of a class referred to as link-reversal algorithms. The protocol builds a loop-free, multipath routing structure that is used as the basis for forwarding traffic to a given destination. The protocol can simultaneously support both source-initiated, on-demand routing for some destinations and destination-initiated, proactive routing for other destinations.

# 1 Introduction

The Temporally-Ordered Routing Algorithm (TORA) [1] is an adaptive routing protocol for multihop networks that possesses the following attributes:

* Distributed execution,
* Loop-free routing,
* Multipath routing,
* Reactive or proactive route establishment and maintenance, and
* Minimization of communication overhead via localization of algorithmic reaction to topological changes.

TORA is distributed, in that routers need only maintain information about adjacent routers (i.e., one-hop knowledge). Like a distance-vector routing approach, TORA maintains state on a per-destination basis. However, TORA does not continuously execute a shortest-path computation and thus the metric used to establish the routing structure does not represent a distance. The destination-oriented nature of the routing structure in TORA supports a mix of reactive and proactive routing on a per-destination basis. During reactive operation, sources initiate the establishment of routes to a given destination on-demand. This mode of operation may be advantageous in dynamic networks with relatively sparse traffic patterns, since it may not be necessary (nor desirable) to maintain routes between every source/destination pair at all times. At the same time, selected destinations can initiate proactive operation, resembling traditional table-driven routing approaches. This allows routes to be proactively maintained to destinations for which routing is consistently or frequently required (e.g., servers or gateways to hardwired infrastructure).

TORA is designed to minimize the communication overhead associated with adapting to network topological changes. The scope of TORA's control messaging is typically localized to a very small set of nodes near a topological change. A secondary mechanism, which is independent of network topology dynamics, is used as a means of route optimization and soft-state route verification. The design and flexability of TORA allow its operation to be biased towards high reactivity (i.e., low time complexity) and bandwidth conservation (i.e., low communication complexity) rather than routing optimality--making it potentially well-suited for use in dynamic wireless networks.

# 2 Terminology

MANET router or router:
A device--identified by a "unique Router ID" (RID)--that executes a MANET routing protocol and, under the direction of which, forwards IP packets. It may have multiple interfaces, each





identified by an IP address. Associated with each interface is a physical layer communication device. These devices may employ wireless or hardwired communications, and a router may simultaneously employ devices of differing technologies. For example, a MANET router may have four interfaces with differing communications technologies: two hardwired (Ethernet and FDDI) and two wireless (spread spectrum and impulse radio).

adjacency:
The name given to an "interface on a neighboring router".

medium:
A communication channel such as free space, cable or fiber through which connections are established.

communications technology:
The means employed by two devices to transfer information between them.

connection:
A physical-layer connection--which may be through a wired or wireless medium--between a device attached to an interface of one MANET router and a device utilizing the same communications technology attached to an interface on another MANET router. From the perspective of a given router, a connection is a (interface, adjacency) pair.

link:
A "logical connection" consisting of the logical *union* of one or more connections between two MANET routers. Thus, a link may consist of a heterogeneous combination of connections through differing media using different communications technologies.

neighbor:
From the perspective of a given MANET router, a "neighbor" is any other router to which it is connected by a link.

topology:
A network can be viewed abstractly as a "graph" whose "topology" at any point in time is defined by set of "points" connected by "edges." This term comes from the branch of mathematics bearing the same name that is concerned with those properties of geometric configurations (such as point sets) which are unaltered by elastic deformations (such as stretching) that are homeomorphisms.

physical-layer topology:
A topology consisting of connections (the edges) through the *same* communications medium between devices (the points)





communicating using the *same* communications technology.

network-layer topology:
A topology consisting of links (the edges) between MANET routers (the points) which is used as the basis for MANET routing. Since "links" are the logical union of physical-layer "connections," it follows that the "network-layer topology" is the logical union of the various "physical-layer topologies."

IP routing fabric:
The heterogeneous mixture of communications media and technologies through which IP packets are forwarded whose topology is defined by the network-layer topology.

## 3 Protocol Functional Description

This section is intended to provide an overview of the protocol and insight into its operation. The protocol specification provided in a subsequent section is intended to serve as the basis for protocol implementations. Thus, in the case of any discrepancies between the description in this section and the subsequent specification section, the specification section should be considered more athoritative.

TORA has been designed to work on top of lower layer mechanisms or protocols that provide the following basic services between neighboring routers:

* Link status sensing and neighbor discovery
* Reliable, in-order control packet delivery
* Link and network layer address resolution and mapping

* Security authentication

Events such as the reception of control messages and changes in connectivity with neighboring routers trigger TORA's algorithmic reactions.

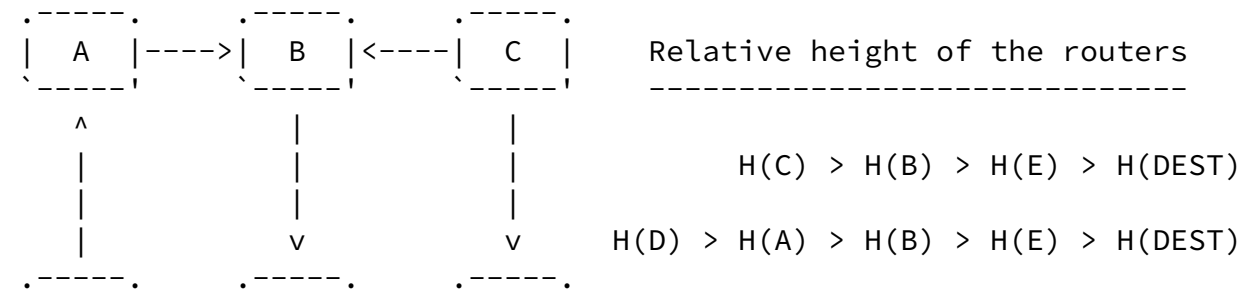
A logically separate version of TORA is run for each "destination" to which routing is required. The following discussion focuses on a single version of TORA running for a given destination. The term destination is used herein to refer to a traditional IP routing destination, which is identified by an IP address and mask (or prefix). Thus, the route to a destination may correspond to the individual address of an interface on a specific machine (i.e., a host route) or an aggregation of addresses (i.e., a network route).

TORA assigns directions to the links between routers to form a routing structure that is used to forward datagrams to the destination. A router assigns a direction ("upstream" or "downstream") to the link with a neighboring router based on the relative values of a metric associated with each router. The metric





maintained by a router can conceptually be thought of as the router's "height" (i.e., links are directed from the higher router to the lower router). The significance of the heights and the link directional assignments is that a router may only forward datagrams downstream. Links from a router to any neighboring routers with an unknown or undefined height are considered undirected and cannot be used for forwarding. Collectively, the heights of the routers and the link directional assignments form a loop-free, multipath routing structure in which all directed paths lead downstream to the destination, see Figure 1. Note that in this example, C is closer to the destination than B in terms of number of hops, but the height metric of C is greater than that of B.



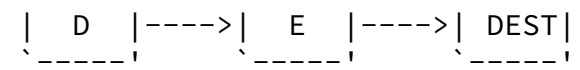



Figure 1: Conceptual representation of the directed acyclic graph defined by the relative height of network routers.

TORA can be separated into four basic functions: creating routes, maintaining routes, erasing routes, and optimizing routes. Creating routes corresponds to the selection of heights to form a directed sequence of links leading to the destination in a previously undirected network or portion of the network. Maintaining routes refers to the adapting the routing structure in response to network topological changes. For example, following the loss of some router's last downstream link, some directed paths may temporarily no longer lead to the destination. This event triggers a sequence of directed link reversals (caused by the re-selection of router heights), which re-orients the routing structure such that all directed paths again lead to the destination. In cases where the network becomes partitioned, links in the portion of the network that has become partitioned from the destination must be marked as undirected to erase invalid routes. During this erasing routes process, routers set their heights to null and their adjacent links become undirected. Finally, TORA includes a secondary mechanism for optimizing routes, in which routers re-select their heights in order to improve the routing structure. TORA accomplishes these four functions through the use of four distinct control packets: query (QRY), update (UPD),





clear (CLR), and optimization (OPT).

## 3.1 Creating Routes

Creating routes can be initiated on-demand by a source or proactively by a destination. In either case, routers select heights with respect to the given destination and assign directions to the links between neighboring routers.

In the on-demand mode, creating routes is accomplished via a query-reply mechanism using QRY and UPD packets. A source initiates the process by sending a QRY packet to its neighbors that identifies the destination for which a route is requested. The QRY packet is propagated out from the source (i.e., processed and forwarded by neighboring routers) until it is received by one or more routers that

have a trusted route to the destination. As the QRY packet is forwarded, routers set a route-requested flag and discard any subsequent QRY packets received for the same destination. The route-requested flag supresses redundant route requests and reduces the need for subsequent route requests when a destination is temporarily unreachable. Routers that have a trusted route to the destination repsond to the QRY packet by sending an UPD packet to their neighbors that identifies the relevant destination and the height of the router sending the UPD packet. Routers also maintain the time at which an UPD packet was last sent to its neighbors and the time at which links to neighboring routers became active, to reduce redundant replies to a given route request. When a router with the route-requested flag set receives an UPD packet, it sets its height and sends an UPD packet to its neighbors that identifies the relevant destination and the new height of the router sending the UPD packet. Thus, routers in the network select heights for the requested desination, learn of their neighbors heights for the destination and assign link directions based on those heights. To ensure that a route request continues to propagate when the destination was initially unreachable, routers with the route-requested flag set must resend a QRY packet upon activation of a new link (i.e., discovery of a new neighbor).

In the proactive mode, the destination initiates the creating routes process by sending an OPT packet that is processed and forwarded by neighboring routers. The OPT packet identifies the destination, the mode of operation for the destination and the height of the router sending the OPT packet. The OPT packet also contains a sequence number used to uniquely identify the packet and ensure that each router processes and forwards a given OPT packet from a destination at most once. As the OPT packet is forwarded, routers set their mode of operation correspondingly, reselect their heights, and send an OPT packet to their neighbors that identifies the relevant destination





and the new height of the router sending the UPD packet.

## 3.2 Maintaining Routes

Maintaining routes is only performed for nodes that have a height other than NULL. Furthermore, any neighbor's height that is NULL is not used for the computations. A node i is said to have no downstream links if HEIGHT < HT_NEIGH[k] for all non-NULL neighbors k. This will

result in one of five possible reactions depending on the state of the node and the preceding event. Each node (other than the destination) that has no downstream links modifies its height, HEIGHT = (tau[i], oid[i], r[i], delta[i], i), as follows:

Case 1 (Generate):

Node i has no downstream links (due to a link failure).

(tau[i], oid[i], r[i])=(t, i, 0), where t is the time of the failure.

(delta[i],i)=(0, i)

In essence, node i defines a new reference level. The above assumes node i has at least one upstream neighbor. If node i has no upstream neighbors it simply sets its height to NULL.

Case 2 (Propagate):

Node i has no downstream links (due to a link reversal following reception of an UPD packet) and the ordered sets (tau[k], oid[k], r[k]) are not equal for all neighbors k.

(tau[i], oid[i], r[i])=max{(t[k], oid[k], r[k]) of all neighbors k}

(delta[i],i)=(delta[m]-1, i), where m is the lowest neighbor with the maximum reference level defined above.

In essence, node i propagates the reference level of its highest neighbor and selects a height that is lower than all neighbors with that reference level.

Case 3 (Reflect):

Node i has no downstream links (due to a link reversal following reception of an UPD packet) and the ordered sets (tau[k], oid[k], r[k]) are equal with r[k] = 0 for all neighbors k.





(tau[i], oid[i], r[i])=(tau[k], oid[k], 1)

(delta[i],i)=(0, i)

In essence, the same level (which has not been "reflected") has propagated to node i from all of its neighbors. Node i "reflects" back a higher sub-level by setting the bit r.

Case 4 (Detect):

Node i has no downstream links (due to a link reversal following reception of an UPD packet), the ordered sets (tau[k], oid[k], r[k]) are equal with r[k] = 1 for all neighbors k, and oid[k] = i (i.e., node i defined the level).

(tau[i], oid[i], r[i])=(-, -, -)

(delta[i],i)=(-, i)

In essence, the last reference level defined by node i has been reflected and propagated back as a higher sub-level from all of its neighbors. This corresponds to detection of a partition. Node i must initiate the process of erasing invalid routes as discussed in the next section.

Case 5 (Generate):

Node i has no downstream links (due to a link reversal following reception of an UPD packet), the ordered sets (tau[k], oid[k], r[k]) are equal with r[k] = 1 for all neighbors k, and oid[k] != i (i.e., node i did not define the level).

(tau[i], oid[i], r[i])=(t, i, 0), where t is the time of the failure

(delta[i],i)=(0, i)

In essence, node i experienced a link failure (which did not require reaction) between the time it propagated a reference level and the reflected higher sub-level returned from all neighbors. This is not necessarily an indication of a partition. Node i defines a new reference level.

Following determination of its new height in cases 1, 2, 3, and 5, node i updates all the entries in its link-status table; and broadcasts an UPD packet to all neighbors k. The UPD packet consists of the destination-ID, j, and the new height of the node i that is

broadcasting the packet, HEIGHT. When a node i receives an UPD packet from a neighbor k, node i reacts as described in the creating routes section and in accordance with the cases outlined above. In the event of the failure a link (i, k) that is not its last downstream link, node i simply removes the entries HT_NEIGH[k] and LNK_STAT[k] in its height and link-status tables.

### 3.3 Erasing Routes

Following detection of a partition (case 4), node i sets its height and the height entry for each neighbor k to NULL (unless the destination j is a neighbor, in which case the corresponding height entry is set to ZERO), updates all the entries in its link-status table, and broadcast a CLR packet. The CLR packet consists of the destination-ID, j, and the reflected reference level of node i, (tau[i], oid[i], 1). In actuality the value r[i] = 1 need not be included since it is always 1 for a reflected reference level. When a node i receives a CLR packet from a neighbor k it reacts as follows:

a) If the reference level in the CLR packet matches the reference level of node i; it sets its height and the height entry for each neighbor k to NULL (unless the destination j is a neighbor, in which case the corresponding height entry is set to ZERO), updates all the entries in its link-status table and broadcasts a CLR packet.

b) If the reference level in the CLR packet does not match the reference level of node i; it sets the height entry for each neighbor k (with the same reference level as the CLR packet) to NULL and updates the corresponding link-status table entries. Thus, the height of each node in the portion of the network that was partitioned is set to NULL and all invalid routes are erased. If (b) causes node i to lose its last downstream link, it reacts as in case 1 of maintaining routes.

### 3.4 Optimizing Routes

TBD.

## 4 Protocol Specification

The subsequent specification is intended to be of sufficient detail to serve as a template for implementations.

### 4.1 Configuration

A router is configured with a router ID (RID), which must be unique among the set of routers collectively running TORA within a routing





domain. This value may correspond to one of the router's IP addresses.

For each interface "i" of a router, the following parameters must be configured.

| | |
|---|---|
| IP_ADDR[i] | IP address of interface. |
| ADDR_MASK[i] | Address mask of interface. |
| PRO_MODE[i] | Indicates reactive/proactive mode of operation. |
| OPT_MODE[i] | Indicates optimization mode of operation. |
| OPT_PERIOD[i] | Period for optimization mechanism. |

For each interface, a network route corresponding to the address and mask of the interface may be added to the routing table. Additionally, TORA may respond to requests (i.e., QRY packets) for routes to destination addresses that match the set of addresses identified by the interface configurations. PRO_MODE[i] (0=OFF, 1=ON) indicates if routes to the destination identified by the corresponding interface address and mask should be created proactively. OPT_MODE[i] (00=OFF, 01=PARTIAL, 10=FULL, 11=reserved for future use) indicates the type (if any) of optimizations that should be used for the destination identified by the corresponding interface address and mask, while the OPT_PERIOD[i] sets the frequency at which the optimizations will occur.

## 4.2 State Variables

A router maintains the state of the configuration parameters outlined above. In addition, for each interface a router maintains a sequence number that is incremented upon changes to the interface mode of operation.

| | |
|---|---|
| MODE_SEQ[i] | Sequence number associated with mode of interface "i". |

For each destination "j", a router maintains the following state variables.

| | |
|---|---|
| HEIGHT[j] | This router's height metric for routing to "j". |
| MODE_SEQ[j] | Sequence number of most recent mode for "j". |

PRO_MODE[j]  Indicates reactive/proactive mode of operation for "j".
OPT_MODE[j]  Indicates optimization mode of operation for "j".
OPT_PERIOD[j]  Indicates optimization period for "j".
RT_REQ[j]  Indicates whether a route request to "j" is pending.
TIME_UPD[j]  Time last UPD packet regarding "j" sent by this router.

For each destination "j", a router maintains a separate instance of the following state variables for each neighbor "k".





HT_NEIGH[j][k]  The height metric of neighbor "k."
LNK_STAT[j][k]  The assigned status of the link to neighbor "k."
TIME_ACT[j][k]  Time the link to neighbor "k" became active.

## 4.3 Auxiliary Variables

For each destination "j" to which routing is required, a router may maintain the following auxiliary variables. Although each of the variables can be computed based on the entries in the LNK_STAT table, maintaining the values continuously may facilitate implementation of the protocol. Whether these variables are maintained continuously or computed when needed is implementation specific.

NUM_ACTIVE[j]  Number of neighbors (i.e., active links).
NUM_DOWN[j]  Number of links marked DN in the LNK_STAT table.
NUM_UP[j]  Number of links marked UP in the LNK_STAT table.

## 4.4 Height Data Structure

Each HEIGHT[j] and HT_NEIGH[j][k] entry requires a data structure that comprises five components. The first three components of the Height data structure represent the reference level of the height entry, while the last two components represent an offset with respect to the reference level. The five components of the Height data structure are as follows.

HEIGHT.tau  Time the reference level was created.
HEIGHT.oid  Unique id of the router that created the reference level.
HEIGHT.r  Flag indicating if it is a reflected reference level.
HEIGHT.delta  Value used in propagation of a reference level.
HEIGHT.id  Unique id of the router to which the height metric refers.

To simplify notation in this specification, a height may be written as an ordered quintuple--e.g., HEIGHT[j]=(tau,oid,r,delta,id). The following two predefined values for a height are used throughout the specification of the protocol.

NULL=(-,-,-,-,id) An unknown or undefined height. Conceptually, this can be thought of as an infinite height.

ZERO=(0,0,0,0,id) The assumed height of a given destination. Note that here "id" is the unique id of the given destination.

## 4.5 Determination of Link Status

Each entry in the LNK_STAT table is maintained in accordance with the following rule.

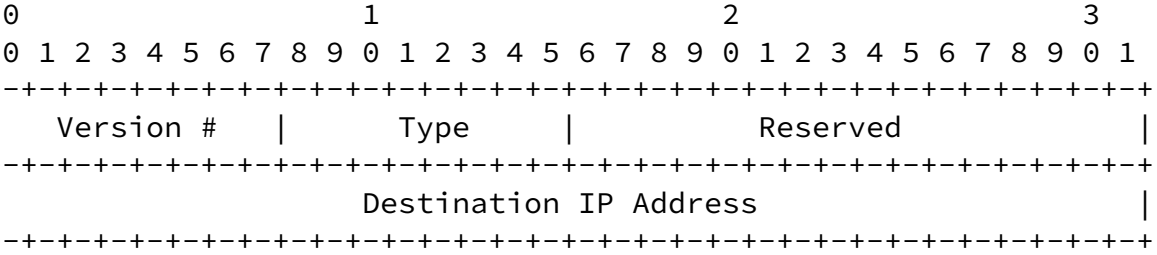




```
if        HT_NEIGH[k]==NULL    then   LNK_STAT[k]=UN;
else if   HEIGHT==NULL         then   LNK_STAT[k]=DN;
else if   HT_NEIGH[k]<HEIGHT   then   LNK_STAT[k]=DN;
else if   HT_NEIGH[k]>HEIGHT   then   LNK_STAT[k]=UP;
```

## 4.6 TORA Packet Formats

### 4.6.1 Query (QRY) Packet Format



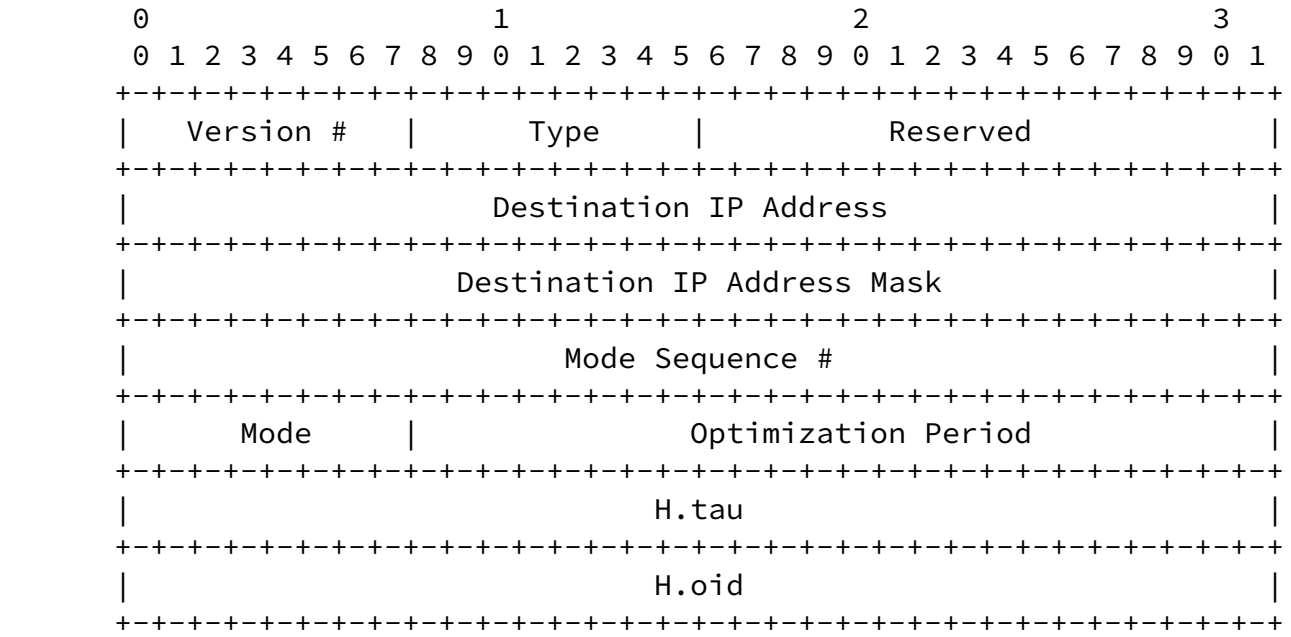

Version #
The TORA version number. This specification documents version 1.

Type
The TORA packet type. For QRY packet this field is set to 1.

Reserved
Field reserved for future use.

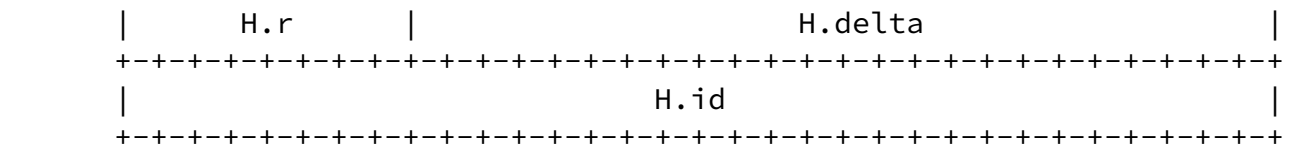
Destination IP Address
The IP address for which a route is being requested.

### 4.6.2 Update (UPD) Packet Format








Version #
The TORA version number. This specification documents version 1.

Type
The TORA packet type. For UPD packet this field is set to 2.

Reserved
Field reserved for future use.

Destination IP Address
The IP address for which a route is being requested.

Destination IP Address Mask

The network mask associated with the destination IP address.

Mode Sequence #
Sequence number associated with the subsequent mode and optimization period fields. Used for propagation of most recent mode state and to ensure each router processes mode information at most once.

Mode
The mode of operation associated with the destination IP address and mask. This field is used to indicate reactive/proactive routing and also the type (if any) of optimizations used for the destination.

Optimization Period
The period for optimization packets originated by the destination.

H.tau
The H.tau value, associated with the destination IP address and mask, of the router sending the UPD.

H.oid
The H.oid value, associated with the destination IP address and mask, of the router sending the UPD.

H.r
The H.r value, associated with the destination IP address and mask, of the router sending the UPD.

H.delta

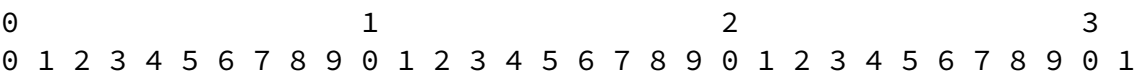


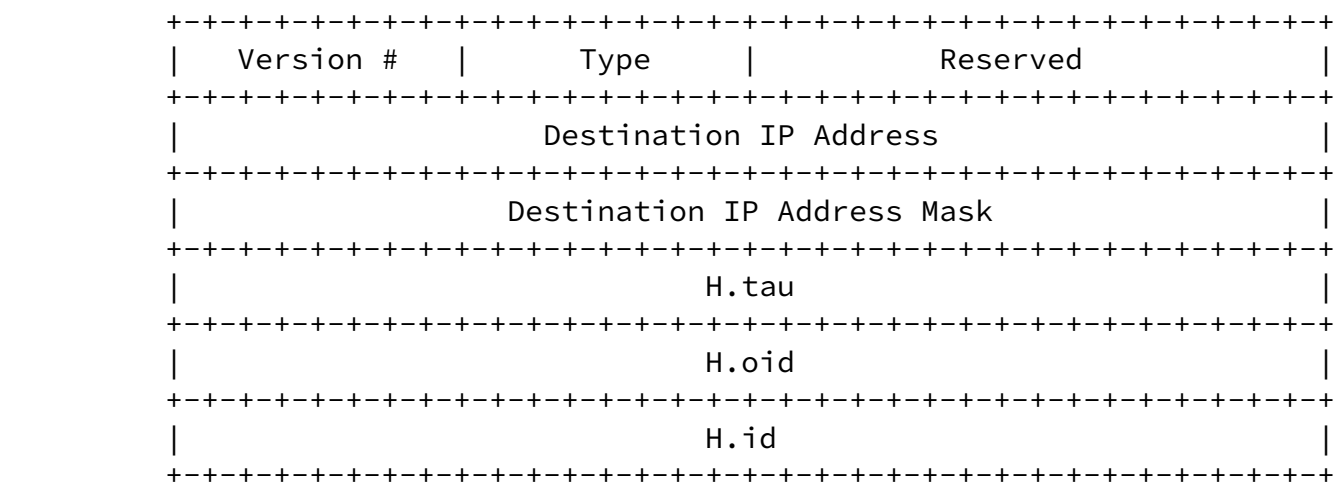


The H.delta value, associated with the destination IP address and mask, of the router sending the UPD.

H.id
The H.id value, associated with the destination IP address and mask, of the router sending the UPD (i.e., unique router ID).

### 4.6.3 Clear (CLR) Packet Format






Version #
The TORA version number. This specification documents version 1.

Type
The TORA packet type. For CLR packet this field is set to 3.

Reserved
Field reserved for future use.

Destination IP Address
The IP address for which a route is being requested.

Destination IP Address Mask
The network mask associated with the destination IP address.

H.tau
The H.tau value, associated with the destination IP address and mask, of the router sending the UPD.

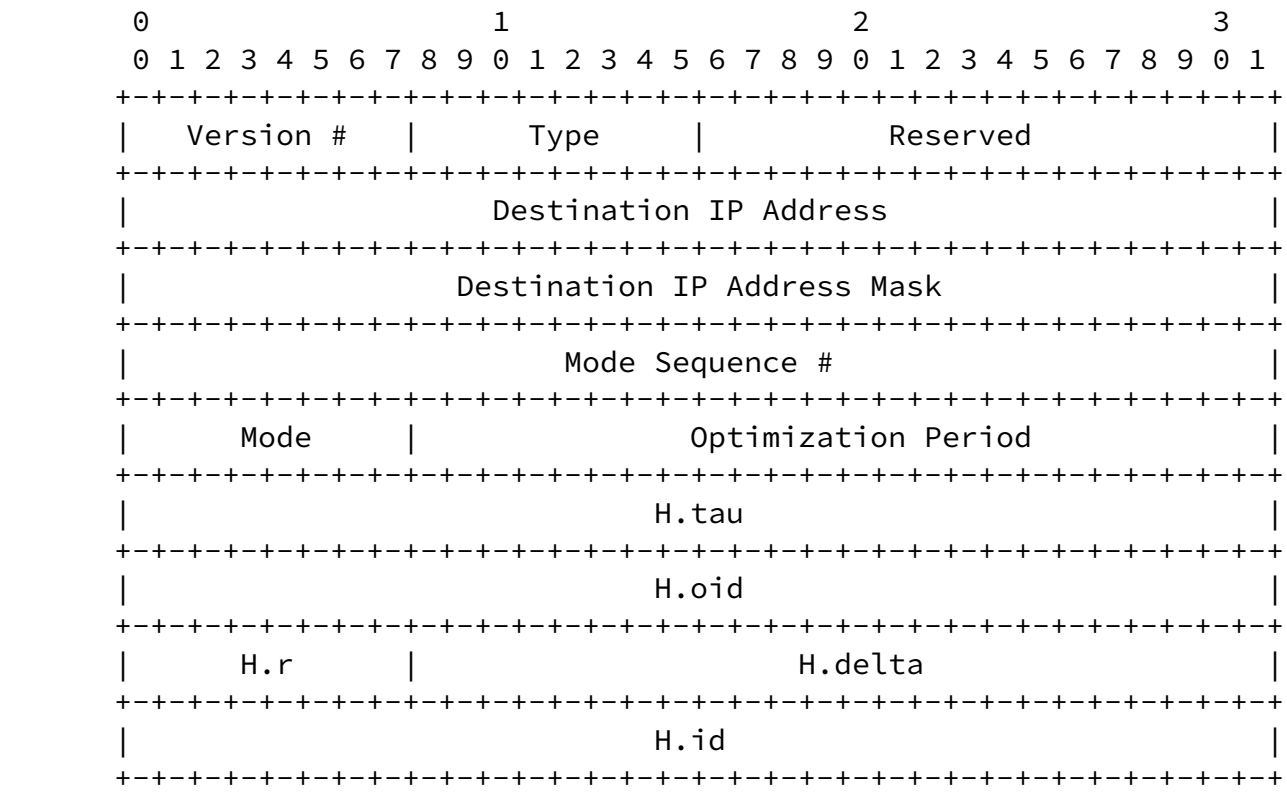
H.oid
The H.oid value, associated with the destination IP address and mask, of the router sending the UPD.





H.id
The H.id value, associated with the destination IP address and mask, of the router sending the UPD (i.e., unique router ID).

### 4.6.4 Optimization (OPT) Packet Format




Version #
The TORA version number. This specification documents version 1.

Type
The TORA packet type. For OPT packet this field is set to 4.

Reserved
Field reserved for future use.

Destination IP Address
The IP address for which a route is being requested.

Destination IP Address Mask
The network mask associated with the destination IP address.

Mode Sequence #
Sequence number associated with the subsequent mode and optimization period fields. Used for propagation of most recent mode state and to ensure each router processes mode information at most once.

Mode
The mode of operation associated with the destination IP address and mask. This field is used to indicate reactive/proactive routing and also the type (if any) of optimizations used for the destination.

Optimization Period
The period for optimization packets originated by the destination.

H.tau
The H.tau value, associated with the destination IP address and mask, of the router sending the UPD.

H.oid
The H.oid value, associated with the destination IP address and mask, of the router sending the UPD.

H.r
The H.r value, associated with the destination IP address and mask, of the router sending the UPD.

H.delta
The H.delta value, associated with the destination IP address and mask, of the router sending the UPD.

H.id
The H.id value, associated with the destination IP address and mask, of the router sending the UPD (i.e., unique router ID).

## 4.7 Event Processing

### 4.7.1 Initialization

TBD

### 4.7.2 Connection Status Change

The TORA process receives notification of link status changes from lower layer mechanisms or protocols. It is anticipated that the TORA process will have access to all the information about the connections. Thus, upon notification, TORA will have sufficient information to determine if any new links have been established or any existing links have been severed. If either is the case, then TORA must proceed as outlined in appropriate subsequent section (4.7.3 or 4.7.4). In addition, since a link is potientially composed of multiple connections, it is also possible for a connection that was used in the routing table to be severed without resulting in the

corresponding link being severed. In this case TORA must modify the





appropriate routing table entries.

### 4.7.3 Link with a New Neighbor "k" Established

For each destination "j":

Set TIME_ACT[j][k] to the current time and increment NUM_ACTIVE[j].

If the neighbor "k" is the destination "j", then set HT_NEIGH[j][k]=ZERO, LNK_STAT[j][k]=DN and increment NUM_DOWN[j], else set HT_NEIGH[j][k]=NULL and LNK_STAT[j][k]=UN.

If the RT_REQ[j] flag is set && neighbor "k" is the destination "j" then I) else II).

I) Set HEIGHT[j]=HT_NEIGH[j][k]. Increment HEIGHT[j].delta. Set HEIGHT[j].id to the unique id of this node. Update LNK_STAT[j][n] for all n. Unset the RT_REQ[j] flag. Set TIME_UPD[j] to the current time. Create an UPD packet and place it in the queue to be sent to all neighbors. Event Processing Complete.

II) If PRO_MODE==1 and HEIGHT[j]!=NULL then A) else B).

A) Set TIME_UPD[j] to the current time. Create an UPD packet and place it in the queue to be sent to all neighbors. If the RT_REQ[j] flag is set, create a QRY packet and place it in the queue to be sent to all neighbors. Event Processing Complete.

B) If the RT_REQ[j] flag is set, create a QRY packet and place it in the queue to be sent to all neighbors. Event Processing Complete.

### 4.7.4 Link with Prior Neighbor "k" Severed

For each destination "j":

Decrement NUM_ACTIVE[j]. If LNK_STAT[j][k]==DN, decrement NUM_DOWN[j]. If LNK_STAT[j][k]==UP, decrement NUM_UP[j].

If NUM_DOWN[j]==0 then I) else II).

I) If NUM_ACTIVE[j]==0 then A) else B).

A) Set HEIGHT[j]=NULL. Unset the RT_REQ[j] flag. Event Processing Complete.

B) If NUM_UP==0 then 1) else 2).





1) If HEIGHT[j]==NULL then a) else b).

a) Event Processing Complete.

b) Set HEIGHT[j]=NULL. Set TIME_UPD[j] to the current time. Create an UPD packet and place it in the queue to be sent to all neighbors. Event Processing Complete.

2) Set HEIGHT[j].tau to the current time. Set HEIGHT[j].oid to the unique id of this node. Set HEIGHT[j].r=0. Set HEIGHT[j].delta=0. Set HEIGHT[j].id to the unique id of this node. Update LNK_STAT[j][n] for all n. Unset the RT_REQ[j] flag. Set TIME_UPD[j] to the current time. Create an UPD packet and place it in the queue to be sent to all neighbors. Event Processing Complete.

II) Event Processing Complete.

### 4.7.5 QRY Packet Regarding Destination "j" Received from Neighbor "k"

If the RT_REQ[j] flag is set then I) else II).

I) Event Processing Complete.

II) If HEIGHT[j].r==0 then A) else B).

A) If TIME_ACT[j][k]>TIME_UPD[j] then 1) else 2).

1) Set TIME_UPD[j] to the current time. Create an UPD packet and place it in the queue to be sent to all neighbors. Event Processing Complete.

2) Event Processing Complete.

B) If HT_NEIGH[j][n].r==0 for any n then 1) else 2).

1) Find m such that HT_NEIGH[j][m] is the minimum of all height entries with HT_NEIGH[j][n].r==0. Set HEIGHT[j]=HT_NEIGH[j][m]. Increment HEIGHT.delta. Set HEIGHT[j].id to the unique id of this node. Update LNK_STAT[j][n] for all n. Set TIME_UPD[j] to the current time. Create an UPD packet and place it in the queue to be sent to all neighbors. Event Processing Complete.

2) Set the RT_REQ[j] flag. If NUM_ACTIVE[j]>1 then a) else b).

a) Create a QRY packet and place it in the queue to be





sent to all neighbors. Event Processing Complete.

b) Event Processing Complete.

### 4.7.6 UPD Packet Regarding Destination "j" Received from Neighbor "k"

If MODE_SEQ field of received packet is greater than MODE_SEQ[j], update entries PRO_MODE[j], OPT_MODE[j], and MODE_SEQ[j].

Update the entries HT_NEIGH[j][k], and LNK_STAT[j][k]. If the RT_REQ[j] flag is set and HT_NEIGH[j][k].r==0 then I) else II).

I) Set HEIGHT[j]=HT_NEIGH[j][k]. Increment HEIGHT.delta. Set HEIGHT[j].id to the unique id of this node. Update LNK_STAT[j][n] for all n. Unset the RT_REQ[j] flag. Set TIME_UPD[j] to the current time. Create an UPD packet and place it in the queue to be sent to all neighbors. Event Processing Complete.

II) If NUM_DOWN[j]==0 then A) else B).

A) If NUM_UP[j]==0 then 1) else 2).

1) If HEIGHT[j]==NULL then a) else b).

a) Event Processing Complete.

b) Set HEIGHT[j]=NULL. Set TIME_UPD[j] to the current time. Create an UPD packet and place it in the queue to be sent to all neighbors. Event Processing Complete.

2) If all HT_NEIGH[j][n], for all n such that HT_NEIGH[j][n] is non-NULL, have the same reference level then a) else b).

a) If HT_NEIGH[j][n].r==0, for any n such that HT_NEIGH[j][n] is non-NULL, then i) else ii).

i) Set HEIGHT[j]=HT_NEIGH[j][n], where n is such that HT_NEIGH[j][n] is non-NULL. Set HEIGHT[j].r=1. Set HEIGHT[j].delta=0. Set HEIGHT[j].id to the unique id of this node. Update LNK_STAT[j][n] for all n. Set TIME_UPD[j] to the current time. Create an UPD packet and place it in the queue to be sent to all neighbors. Event Processing Complete.

ii) If HT_NEIGH[j][n].oid==id, where n is such that HT_NEIGH[j][n] is non-NULL and id is the unique id of this node, then x) else y).





x) Save the current values of HEIGHT[j].tau and HEIGHT[j].oid in temporary variables. Set HEIGHT[j]=NULL. Set NUM_DOWN[j]=0. Set NUM_UP[j]=0. For every active link n, if the neighbor connected via link n is the destination j, set HT_NEIGH[j][n]=ZERO and LNK_STAT[j][n]=DN else set HT_NEIGH[j][n]=NULL and LNK_STAT[j][n]=UN. Create a CLR packet, with the previously saved values of tau and oid, and place it in the queue to be sent to all neighbors. Event Processing Complete.

y) Set HEIGHT[j].tau to the current time. Set HEIGHT[j].oid to the unique id of this node. Set HEIGHT[j].r=0. Set HEIGHT[j].delta=0. Set HEIGHT[j].id to the unique id of this node. Update LNK_STAT[j][n] for all n. Unset the RT_REQ[j] flag. Set TIME_UPD[j] to the current time. Create an UPD packet and place it in the queue to be sent

to all neighbors. Event Processing Complete.

b) Find n such that HT_NEIGH[j][n] is the maximum of all non-NULL height entries. Find m such that HT_NEIGH[j][m] is the minimum of the non-NULL height entries with the same reference level as HT_NEIGH[j][n]. Set HEIGHT[j]=HT_NEIGH[j][m]. Decrement HEIGHT.delta. Set HEIGHT[j].id to the unique id of this node. Update LNK_STAT[j][n] for all n. Set TIME_UPD[j] to the current time. Create an UPD packet and place it in the queue to be sent to all neighbors. Event Processing Complete.

B) IF PRO_MODE changed from OFF to ON as a result of this UPD packet reception and HEIGHT[j]==NULL then 1) else 2)

1) Find m such that HT_NEIGH[j][m] is the minimum of all non-NULL height entries. Set HEIGHT[j]=HT_NEIGH[j][m]. Increment HEIGHT[j].delta. Set HEIGHT[j].id to the unique id of this node. Update LNK_STAT[j][n] for all n. Set TIME_UPD[j] to the current time. Create an UPD packet and place it in the queue to be sent to all neighbors. Event Processing Complete.

2) Event Processing Complete.

### 4.7.7 CLR Packet Regarding Destination "j" Received from Neighbor "k"

If HEIGHT[j].tau and HEIGHT[j].oid match the values of tau and oid from the CLR packet and HEIGHT[j].r==1 then I) else II).





I) Save the current values of HEIGHT[j].tau and HEIGHT[j].oid in temporary variables. Set Height[j]=NULL. Set NUM_DOWN[j]=0. Set NUM_UP[j]=0. For every active link n, if the neighbor connected via link n is the destination j, set HT_NEIGH[j][n]=ZERO and LNK_STAT[j][n]=DN else set HT_NEIGH[j][n]=NULL and LNK_STAT[j][n]=UN. If NUM_ACTIVE[j]>1 then A) else B).

A) Create a CLR packet, with the previously saved values of tau and oid, and place it in the queue to be sent to all neighbors. Event Processing Complete.

B) Event Processing Complete.

II) Set HT_NEIGH[j][k]=NULL and LNK_STAT[j][k]=UN. For all n such that HT_NEIGH[j][n].tau and HT_NEIGH[j][n].oid match the values of tau and oid from the CLR packet and HT_NEIGH[j][n].r==1, set HT_NEIGH[j][n]=NULL and LNK_STAT[j][n]=UN. If NUM_DOWN[j]==0 then A) else B).

A) If NUM_UP==0 then 1) else 2).

1) If HEIGHT[j]==NULL then a) else b).

a) Event Processing Complete.

b) Set HEIGHT[j]=NULL. Set TIME_UPD[j] to the current time. Create an UPD packet and place it in the queue to be sent to all neighbors. Event Processing Complete.

2) Set HEIGHT[j].tau to the current time. Set HEIGHT[j].oid to the unique id of this node. Set HEIGHT[j].r=0. Set HEIGHT[j].delta=0. Set HEIGHT[j].id to the unique id of this node. Update LNK_STAT[j][n] for all n. Unset the RT_REQ[j] flag. Set TIME_UPD[j] to the current time. Create an UPD packet and place it in the queue to be sent to all neighbors. Event Processing Complete.

B) Event Processing Complete.

### 4.7.8 OPT Packet Regarding Destination "j" Received from Neighbor "k"

If MODE_SEQ field of received packet is greater than MODE_SEQ[j] then I) else II).

I) Update entries PRO_MODE[j], OPT_MODE[j], and MODE_SEQ[j]. If PRO_MODE[j] changed as a result of this OPT packet reception || (OPT_MODE[j]==PARTIAL && HEIGHT[j]!=NULL) || OPT_MODE[j]==FULL then A) else B).





A) Set HEIGHT[j]=ZERO. Set HEIGHT[j].delta to the value of the DELTA field in the received OPT packet + 1. Set HEIGHT[j].id to the unique id of this node. Update LNK_STAT[j][n] for all n. Unset the RT_REQ[j] flag. Set TIME_UPD[j] to the current time. Create an OPT packet and place it in the queue to be

sent to all neighbors. Event Processing Complete.

B) Event Processing Complete.

II) Event Processing Complete.

4.7.9 Mode Configuration Change or Optimization Timer Event for local interface "i"

Increment MODE_SEQ[i]. Create an OPT packet and place it in the queue to be sent to all neighbors. If OPT_MODE[i]==PARTIAL || OPT_MODE[i]==FULL, schedule a local optimization timer event for interface "i" to occur at a time randomly selected between 0.5*OPT_PERIOD[i] and 1.5*OPT_PERIOD[i] seconds based on a uniform distribution. Event Processing Complete.

## 5 Security Considerations

TBD.

## 6 Intellectual Property Rights Notice

Both the University of Maryland and the U.S. Naval Research Laboratory have applied for patents relating to the technology described in this internet draft.

## References

[1] V. Park and M. S. Corson, A Highly Adaptive Distributed Routing Algorithm for Mobile Wireless Networks, Proc. IEEE INFOCOM '97, Kobe, Japan (1997).

[4] M.S. Corson and V. Park, An Internet MANET Encapsulation Protocol (IMEP), draft-ietf-

## Author's Addresses

Vincent D. Park
park@flarion.com
(908) 947-7084

M. Scott Corson
corson@flarion.com
(908) 947-7033

Flarion Technologies, Inc.
Bedminster One
135 Route 202/206 South
Bedminster, NJ 07921