Cluster Routing Protocol

Xinwen Cui

August, 2003

Supervised by professor Michel Barbeau

Table Of Contents

1 Protocol Overview 3
2 Protocol Properties 5
3 Protocol Assumptions 6
4 Basic Protocol Design Rules 7
4.1 Logical View of Cluster Head 7
4.2 Node Responding to Route Request 7
5 Example Network Diagram 8
6 Data Structures Maintained By Network Nodes 9
6.1 Tables Maintained by Cluster Member (Node E) 9
6.2 Tables Maintained by Cluster Head (Node 2) 10
7 Protocol Description 11
7.1 Route Discovery (Example -- Node E needs a route to node K) 11
7.1.1 Source Node 11
7.1.2 Source Cluster Head 12
7.1.3 Cluster Member 13
7.1.4 Intermediate Cluster Head 16
7.1.5 Destination Node 17
7.2 Route Set-up With C_RREP 18
7.2.1 Destination Cluster 19
7.2.2 Intermediate Cluster 20
7.2.3 Source Cluster 21
7.3 Route Set-up With C_RCACHED 22
7.3.1 Source Cluster 22
7.3.2 Intermediate And Destination Cluster 23
7.4 Route Maintenance 25
7.4.1 General Route Management 25
7.4.2 Route Repair 25
7.4.2.1 Route Repair Without Back Tracking (Cluster Member) 25
7.4.2.2 Route Repair With Back Tracking (Cluster Member) 27
7.4.2.3 Route Repair (Cluster head) 28
7.5 Route Error Processing 29
7.5.1 No Route For A Data Packet 29
7.5.2 Can Not Forward A C_RREP To Next Hop Cluster 30
7.5.3 Can Not Forward A C_RSETUP To Next Hop Cluster 32
8 Interaction With Cluster Formation Algorithm 34
8.1 A Node Joins Another Cluster 34
8.2 A Cluster Head Becomes A Cluster Member 34
9 References 35
A Protocol Design Testing 36
A.1 Testing Framework 36
A.2 Implementation Protocol Stack 37
A.3 Implementation Software 37
A.4 Testing Result 37
A.4.1 Node E 38
A.4.2 Node G 39
A.4.3 Node K 41

1 Protocol Overview

For Ad Hoc wireless network, route discovery and route maintenance are two main tasks of the routing protocol. Usually, if the routing protocol is reactive (on-demand), then broadcasting route request is used to find a network route. To control propagation of broadcasting messages in the network, some flooding control mechanisms are used to control the route request packet forwarding.

If the wireless network is not organized into a routing zone or routing hierarchy, then the reactive routing protocol is not scale because of route request message flooding and route maintenance. So, to design an Ad Hoc network routing protocol which scale to larger networks, the network has to be organized into routing areas. For example, ZRP (Zone Routing Protocol) [1] uses routing zone to control route request flooding.

Forming routing clusters is an effective way to contain the routing costs in larger Ad Hoc networks. Essentially, the network is organized into two hierarchies, the lower hierarchy is a cluster, then the network consists of interconnected clusters. Professor Michel Barbeau has proposed a new cluster formation algorithm [2]. To demonstrate the benefits of cluster in routing protocol design, we propose a reactive routing protocol which is based on routing clusters.

The network is organized into clusters by a cluster formation algorithm (described separately). For a cluster, there is a cluster head and several cluster members (possibly no cluster member). Based on the cluster formation algorithm, the cluster head is one hop away from each of its members, that is the cluster head is a neighbour of its members.

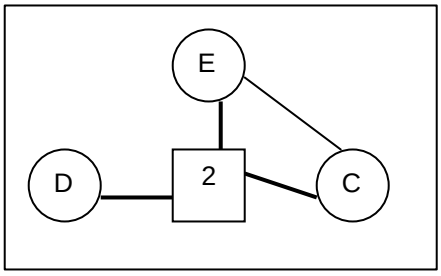


Figure 1 A Cluster

This routing protocol has the following requirements to the cluster formation algorithm (see next chapter for details):

1). Each node advertises its presence periodically, this message includes its address and its cluster head;

2). Each cluster member sends its neighbouring cluster heads, which it learned from its neighbours, to its belonging cluster head;

The cluster head combining the above information to set up a table which contains its cluster members and their connected neighbouring clusters. A cluster member which connected to another neighbouring cluster is called a cluster gateway.

The cluster head IP address acts as a network identity of its cluster (the cluster identity may be derived from other information gathered from its members).

Since a cluster head knows all its members and its cluster gateways to the neighbouring clusters, set-up a route inside a cluster or passing a cluster is trivial.

Route discovery has two phases, route request propagation and route set-up. Route request is propagated from cluster to cluster, and the route information is accumulated in the route request, the traversed cluster identity (assuming cluster head as a cluster identity) is appended into the route request.

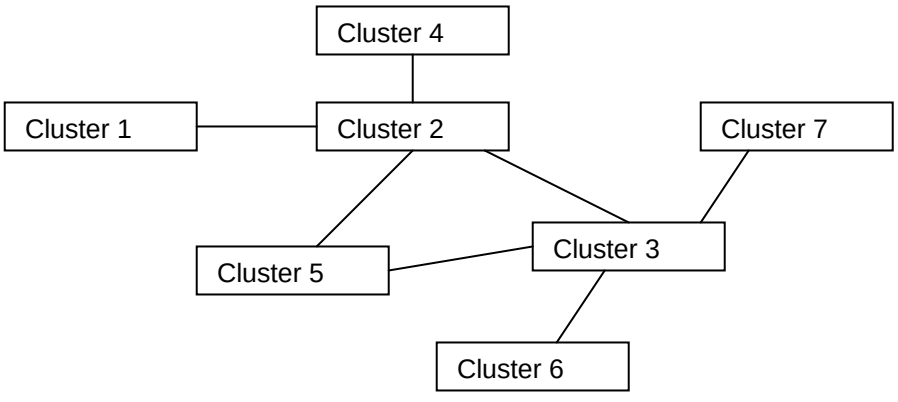


Figure 2 Abstracted View of A Network

Network forwarding route is table driven. Network route is set-up by following the route reply propagation from the route request destination to the route request source. As a node

forwards the route reply towards the source, its routing table is updated for both the source and destination routes.

2 Protocol Properties

This protocol has the following properties.

1). Route discovery and route maintenance operate entirely on demand.

2). Route information accumulated in the route request only includes the traversed clusters, so, the route request message size is increased proportionally to the traversed clusters, not to the traversed nodes. A larger network diameter can be supported by this protocol.

3). Route is set-up by following the route reply forwarding from the destination node to the source node, no route establishment during the route request propagation phase, so un-necessary route is not set-up and not maintained, this saves network node memory and CPU time.

4). Route request propagation is controlled by the traversed clusters appended into the route request, no complex graph calculation algorithm is involved.

3 Protocol Assumptions

For this protocol design, we make the following assumptions.

1). All links are bi-directional;

2). Each node belongs to a cluster, either as a member or cluster head;

3). Cluster has no overlapping, that is a node belongs to exactly one cluster;

4). Each node knows its neighbours;

5). Cluster formation is already done by the cluster formation algorithm;

6). Each cluster member knows its cluster head;

7). Each cluster head knows all its members, but cluster members maybe do not know each other, except two cluster members also are neighbours;

8). Cluster head are neighbours with each of its members;

9). The cluster head is as powerful as its members in terms of CPU power, memory capacity, battery life, etc;

10). Each node advertises its presence periodically, this message includes its address and cluster head;

11). Each cluster member sends its neighbouring cluster heads, which it learned from its neighbours, to its belonging cluster head;

If a node has not joined in a cluster, then this node does not participate in the routing protocol.

4 Basic Protocol Design Rules

4.1 Logical View of Cluster Head

A cluster head is internally separated into two logical parts, one is a logical cluster head, another one is a logical cluster member, it processes all inter-cluster and intra cluster routing messages corresponding to the following diagram,

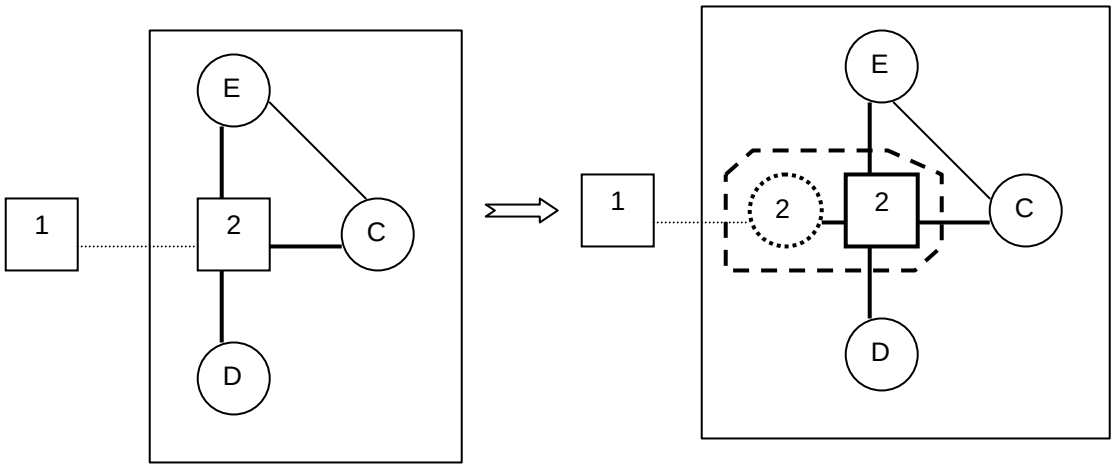


Figure 3 Cluster Head Logical View

The logical cluster member acts as a real cluster member when processing internal route request and route maintenance, also when processing inter-cluster routing messages.

4.2 Node Responding to Route Request

Only destination node can reply to a route request.

5 Example Network Diagram

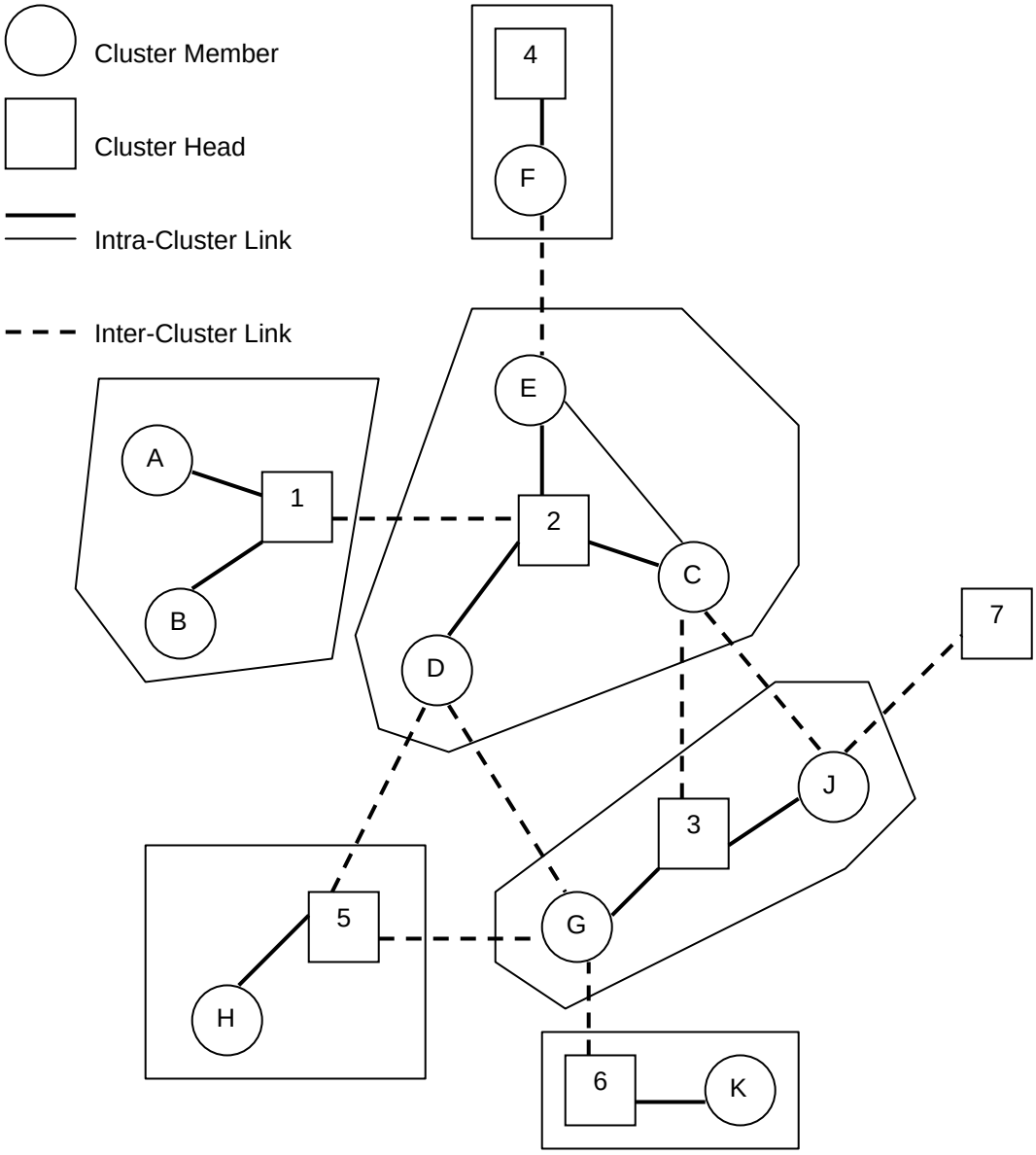


Figure 4 Example Network Diagram

6 Data Structures Maintained By Network Nodes

A node maintains the following data structures with information extracted from cluster formation, neighbour discovery and routing messages.

6.1 Tables Maintained by Cluster Member (Node E)

Cluster Head Table

Cluster Head IP
2

When a node joins a cluster as a member, it updates this table with its new cluster head.

Node Neighbour Table

Neighbour IP	Neighbour Cluster Head IP
2	2
C	2
F	4

This table is updated with the information carried in the neighbour discovery message.

Node Routing Table

A node maintains a routing table for packet forwarding. The routing table is updated when the node receives a routing message or its neighbours are changed.

Destination IP	Next Hop Node IP (Neighbour)	Next Hop Cluster Head IP	Route [Traversed Cluster heads]	Timer	Route State
D	2	[-]	[2]	120	Valid
7	[-]	[-]	[-]	60	In-Request
K	2	3	[2, 3, 6]	300	Unreachable
J	C	3	[2, 3]	0	Refresh-Expired
4	F	4	[2, 4]	50	Need-Repair

A route entry can have the following state:

Valid	The route is valid for data transfer
In-Request	Route discovery in progress, that is a route request packet has been sent
Need-Repair	The node lost its next hop neighbour node towards the destination, the route needs to be repaired before data transfer (the route is only repaired when a data packet is forwarded)
Refresh-Expired	The route is not used for a long time.
Unreachable	The node could not find a route to the destination (the node will not send another route request until a time period is passed)

6.2 Tables Maintained by Cluster Head (Node 2)

Cluster Members Table

Cluster Member IP
C
D
E

This table is updated with the information from the cluster formation algorithm.

Node Neighbour Table

Neighbour IP	Neighbour Cluster Head IP
1	1
C	2
D	2
E	2

Neighbour Cluster Table

Neighbour Cluster Head IP	Cluster Gateway IP
1	2
3	C, D
4	E
5	D

This table is updated when a cluster member sends its neighbouring clusters to the cluster head.

Cached Route Table

Destination IP	Route [Traversed Cluster heads]	Cache Timer (Seconds)
K	[2, 3, 6]	300
J	[2, 3]	30
4	[2, 4]	120

This table is maintained for its members to look up a route which already learned when one of its members asks for the route.

Node Routing Table

Destination IP	Next Hop Node IP (Neighbour)	Next Hop Cluster Head IP	Route [Traversed Cluster heads]	Timer	Route State

7 Protocol Description

7.1 Route Discovery (Example -- Node E needs a route to node K)

7.1.1 Source Node

When a node sends a data packet to another node, it first checks to see whether the destination node is its neighbour or not. <1>, If the destination is its neighbour, no route discovery is necessary, the data packet is directly sent to the destination node;

```
Node K is not node E's neighbour.
```

otherwise, <2> the node checks for a route in its current routing table, if a route is found, the data packet is forwarded to the next hop node by following the found route;

```
Node E has not current route to node K.
```

<3> If no current route is found, the node asks its cluster head to discover a route, the routing packet sent to the cluster head is C_RREQ, Cluster Route REQuest.

The C_RREQ packet has the following format.

0…7	8…15	16…31
Type (C_RREQ)		Flags
Source Cluster Head IP (2)		
Source IP (E)		
C_RREQ Sequence ID (-)		
Destination Cluster Head IP (-)		
Destination IP (K)		
Traversed Cluster Hop Count (0)		
Traversed Cluster Head 1 (-)		

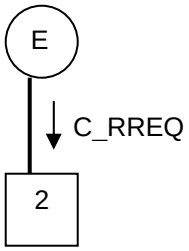


Figure 5 Source Node Sends C_RREQ To Its Cluster Head

7.1.2 Source Cluster Head

When the source cluster head receives a C_RREQ packet, it first checks to see whether the destination node is its member or not. <1> If the destination node is its member, it forwards the C_RREQ to the destination node.

Next, <2> If the source node is its member, the source cluster head searches its cached route table for a route to the destination, if a route is found, the source cluster head sends a C_RCACHED to the source node.

The C_RCACHED packet has the following format if the source cluster head has a cached route to the destination K.

0…7	8…15	16…31
Type (C_ RCACHED)		Flags
Source Cluster Head IP (2)		
Source IP (E)		
C_RREQ Sequence ID (-)		
Destination Cluster Head IP (6)		
Destination IP (K)		
Traversed Cluster Hop Count (1)		
Traversed Cluster Head 1 (3)		

When the source node receives a C_RCACHED packet, it will set-up a route to the destination (See Route Set-up with C_RCACHED for details).

Next, <3> if no route is found in the cached route table, the source cluster head broadcasts the C_RREQ locally after inserting a unique sequence number. The source node can send another C_RREQ for the same destination after a delay (fixed or exponential) if it does not receive a C_RREP.

The C_RREQ has the following format when the source cluster head broadcasting it.

0…7	8…15	16…31
Type (C_RREQ)		Flags
Source Cluster Head IP (2)		
Source IP (E)		
C_RREQ Sequence ID (123)		
Destination Cluster Head IP (-)		
Destination IP (K)		
Traversed Cluster Hop Count (0)		

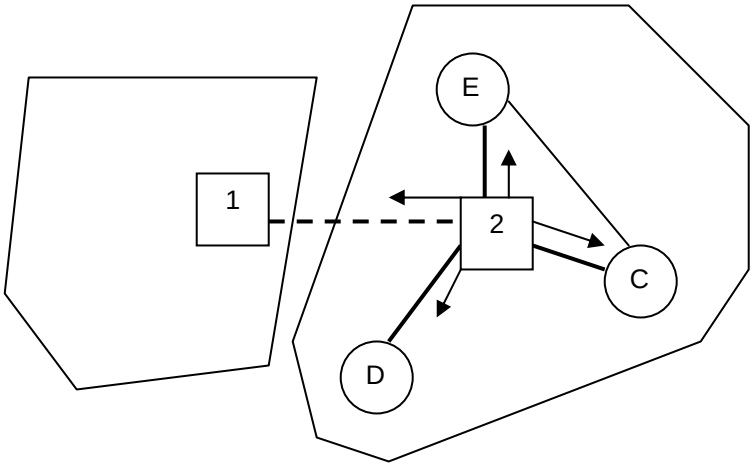


Figure 6 Cluster Head Broadcasts C_RREQ Locally

7.1.3 Cluster Member

when a node receives a C_RREQ, <1> If it is the destination of this received C_RREQ, it may reply to this C_RREQ (see Destination Node for details).

Otherwise, this node is not the destination of the received C_RREQ. <2> If the C_RREQ comes from its cluster head, it forwards this received C_RREQ to each its neighbour node which the neighbour's cluster head is not in the traversed cluster heads (including the source cluster head). For example, when node C receives the C_RREQ broadcasted from its cluster head 2, it will forwards this C_RREQ to its neighbour 3 (if a node has two neighbours belonging to the same neighbour cluster, it only forwards one copy to a neighbour, neighbour cluster head is preferred, otherwise, one neighbour cluster member is picked up randomly).

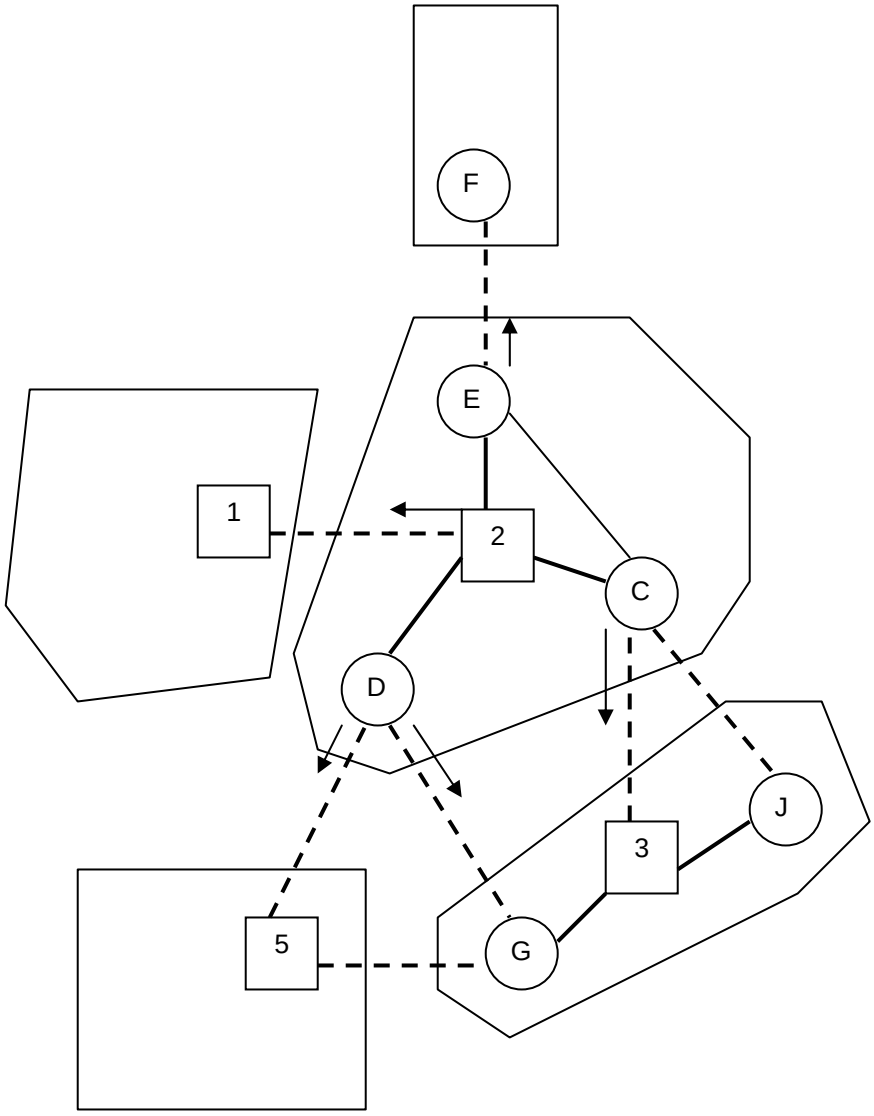
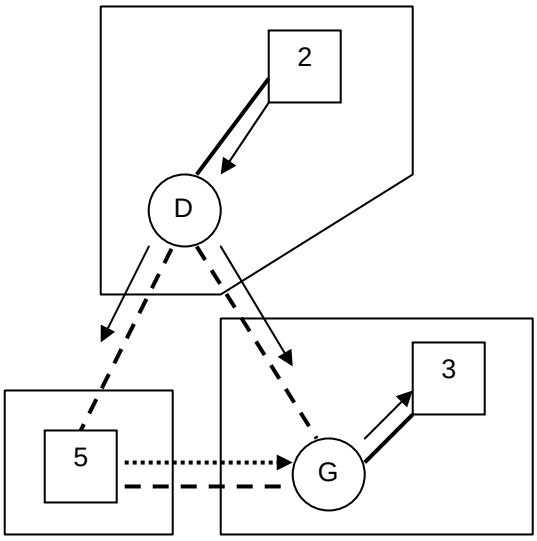


Figure 7 C_RREQ Propagation to Neighbour Clusters

<3> This node is not the destination of the received C_RREQ, also this C_RREQ is not sent by its cluster head, so the received C_RREQ must come from one of its neighbours. Next, It checks to see whether it has seen this C_RREQ recently by comparing the source cluster head IP and sequence number, if it has seen this C_RREQ recently, the received C_RREQ is discarded. This prevents the following scenario.



Suppose Node G has received the C_RREQ sent from node D and forwarded this C_RREQ to its cluster head 3, when node G receives the same C_RREQ from node 5, it will discards this C_RREQ since it has seen this C_RREQ recently.

<4> This C_RREQ is not a duplicate one, it checks to see whether its cluster head is in the traversed cluster heads (including the source cluster head) or not. If its cluster head is already in the traversed cluster heads (including the source cluster head), its cluster has been traversed, it discards the C_RREQ; otherwise, it forwards the received C_RREQ to its cluster head. For example, when node C receives the broadcasted C_RREQ from node 3, since node C's cluster 2 has been traversed (the source cluster), it will discards the received C_RREQ.

7.1.4 Intermediate Cluster Head

When a cluster head receives a C_RREQ, it checks to see whether the destination is its member or not. <1> If the destination is not its cluster member, it appends its cluster head IP into the C_RREQ, then broadcasts the C_RREQ locally. For example, when cluster head 3 receives the C_RREQ from cluster 2, since the destination K is not its member, it will broadcast the C_RREQ locally after appending its cluster head IP into the received C_RREQ.

0…7	8…15	16…31
Type (C_RREQ)		Flags
Source Cluster Head IP (2)		
Source IP (E)		
C_RREQ Sequence ID (123)		
Destination Cluster Head IP (-)		
Destination IP (K)		
Traversed Cluster Hop Count (1)		
Traversed Cluster Head 1 (3)		

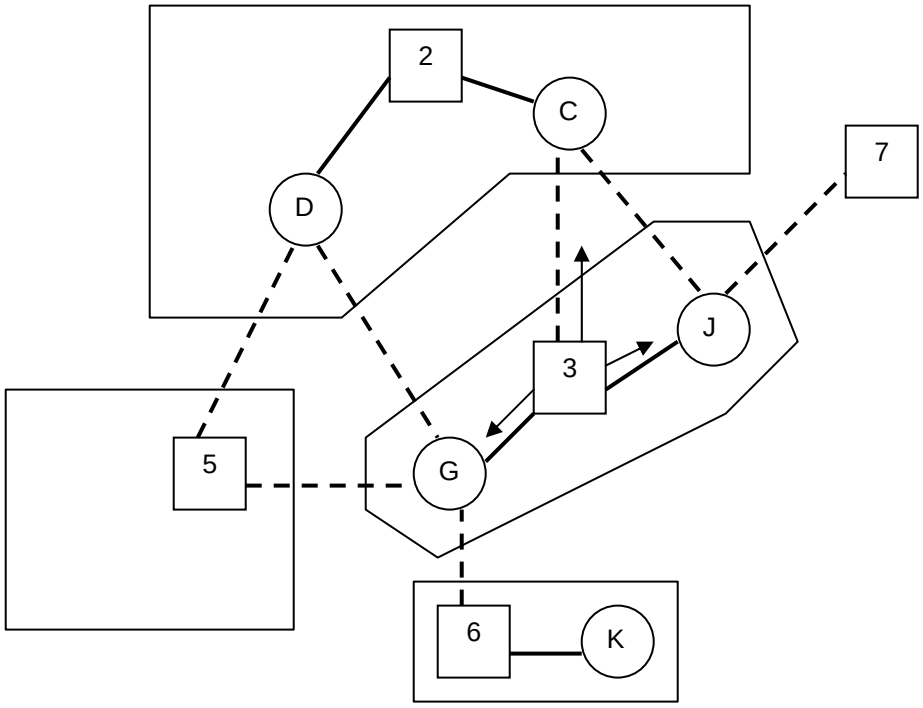


Figure 8 C_RREQ Propagation By Intermediate Cluster Head

<2> If the destination is its cluster member, it forwards the C_RREQ to the destination.

7.1.5 Destination Node

When the destination node receives a C_RREQ, if it has recently replied to this C_RREQ, it discards the received C_RREQ. Otherwise, it sends a C_RREP to the source node by following the revered cluster heads route in the C_RREQ. For example, if the destination node K receives a C_RREQ with the following format.

0…7	8…15	16…31
Type (C_RREQ)		Flags
Source Cluster Head IP (2)		
Source IP (E)		
C_RREQ Sequence ID (123)		
Destination Cluster Head IP (-)		
Destination IP (K)		
Traversed Cluster Hop Count (1)		
Traversed Cluster Head 1 (3)		

The C_RREP has the following format.

0…7	8…15	16…31
Type (C_RREP)		Flags
Source Cluster Head IP (2)		
Source IP (E)		
C_RREQ Sequence ID (123)		
Destination Cluster Head IP (6)		
Destination IP (K)		
Traversed Cluster Hop Count (1)		
Traversed Cluster Head 2 (3)		

7.2 Route Set-up With C_RREP

Route is established by following the forwarding of C_RREP to the source node. For example, the C_RREP has the following format.

0…7	8…15	16…31
Type (C_RREP)		Flags
Source Cluster Head IP (2)		
Source IP (E)		
C_RREQ Sequence ID (123)		
Destination Cluster Head IP (6)		
Destination IP (K)		
Traversed Cluster Hop Count (1)		
Traversed Cluster Head 2 (3)		

7.2.1 Destination Cluster

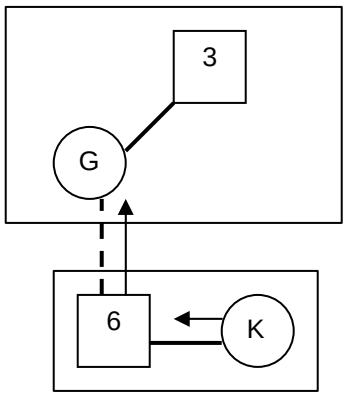


Figure 9 Route Set-up In Destination Cluster

When the destination node K receives the C_RREQ, it first adds a route entry into its routing table for the source node E if no valid route entry for the source node E is existing. Next, the destination node K sends a C_RREP to the source node E. Since node K does not have a neighbour with the next hop cluster 3, it will send the C_RREP to its cluster head 6 (The cluster head knows all its cluster member gateways to the neighbouring clusters).

Node K Routing Table

Destination IP	Next Hop Node IP (Neighbour)	Next Hop Cluster Head IP	Route [Traversed Cluster heads]	Timer	Route State
E	6	3	[6, 3, 2]		Valid

When node 6 (cluster head) receives the C_RREP, since the destination is its cluster member (also its neighbour), so, node 6 does not create a route entry for the destination K. Node 6 searches its neighbour cluster table for a gateway to the next hop cluster 3, since itself has a neighbour belonging to cluster 3, so, node 6 creates a route entry to the source node E, then forwards the C_RREP to its neighbour G.

Node 6 Routing Table

Destination IP	Next Hop Node IP (Neighbour)	Next Hop Cluster Head IP	Route [Traversed Cluster heads]	Timer	Route State
E	G	3	[6, 3, 2]		Valid

7.2.2 Intermediate Cluster

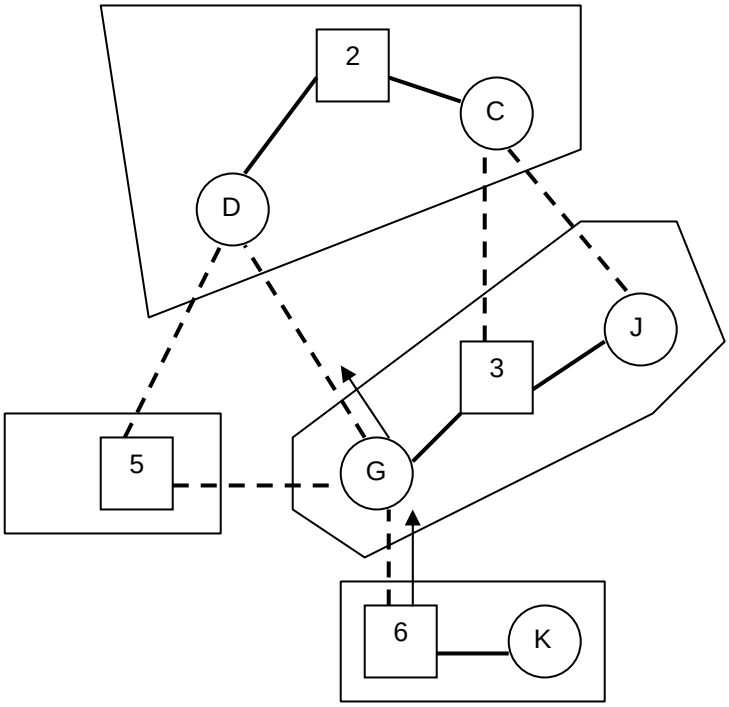


Figure 10 Route Set-up In Intermediate Cluster

Node G Routing Table

Destination IP	Next Hop Node IP (Neighbour)	Next Hop Cluster Head IP	Route [Traversed Cluster heads]	Timer	Route State
E	D	2	[3, 2]		Valid
K	6	6	[3, 6]		Valid

When node G receives the C_RREP from node 6, it will add a route entry to the destination K with the next hop node as the node it received the C_RREP.

Since node G has a neighbour node D to the next hop cluster 2 in the C_RREP, it adds a route to the source node E into its routing table, then forwards the C_RREP to its neighbour D.

7.2.3 Source Cluster

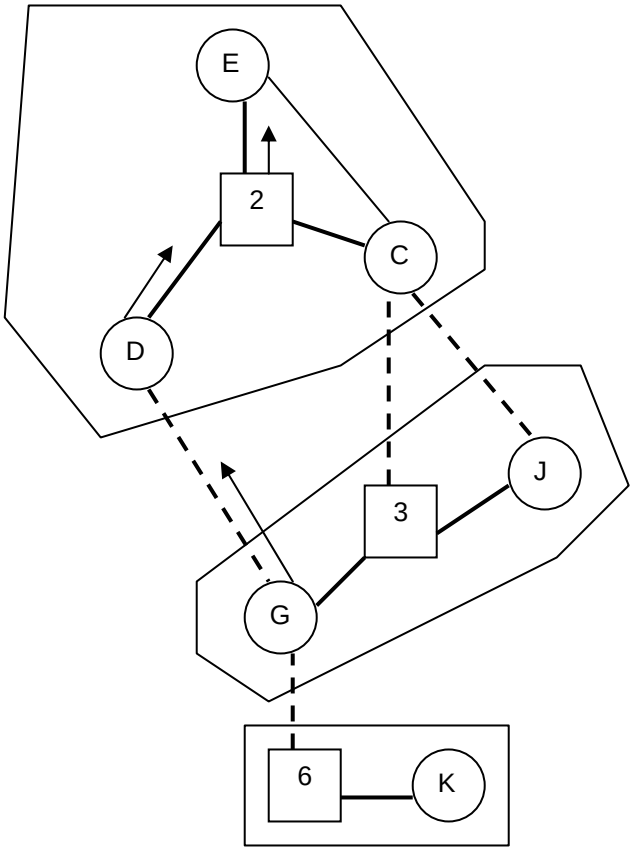


Figure 11 Route Set-up In Source Cluster

Node D Routing Table

Destination IP	Next Hop Node IP (Neighbour)	Next Hop Cluster Head IP	Route [Traversed Cluster heads]	Timer	Route State
E	2	[-]	[2]		Valid
K	G	3	[2, 3, 6]		Valid

When the source cluster head receives a C_RREP to its member, it will cache the discovered route to the destination in the C_RREP for other members to look up.

Node 2 Routing Table

Destination IP	Next Hop Node IP (Neighbour)	Next Hop Cluster Head IP	Route [Traversed Cluster heads]	Timer	Route State
K	D	3	[2, 3, 6]		Valid

Cluster Head 2 Cached Route Table

Destination IP	Route [Traversed Cluster heads]	Cache Timer (Seconds)
K	[2, 3, 6]	300

Node E Routing Table

Destination IP	Next Hop Node IP (Neighbour)	Next Hop Cluster Head IP	Route [Traversed Cluster heads]	Timer	Route State
K	2	3	[2, 3, 6]		Valid

When the source node receives a C_RREP, if this C_RREP is not forwarded to it by its cluster head, it sends a copy of this C_RREP to its cluster head for discovered route caching.

7.3 Route Set-up With C_RCACHED

Suppose cluster head 2 has a cached route as following.

Cluster Head 2 Cached Route Table

Destination IP	Route [Traversed Cluster heads]	Cache Timer (Seconds)
K	[2, 3, 6]	300

7.3.1 Source Cluster

Node C needs a route to the destination K for packet forwarding. Since node C has no current route to node K, it will send a C_RREQ to its cluster head 2. When cluster head 2 receives the C_RREQ from its member C, it searches its cached route for a route to destination K; cluster head 2 has a cached route to node K, it sends a C_RCACHED to node C.

0…7	8…15	16…31
Type (C_ RCACHED)		Flags
Source Cluster Head IP (2)		
Source IP (E)		
C_RREQ Sequence ID (-)		
Destination Cluster Head IP (6)		
Destination IP (K)		
Traversed Cluster Hop Count (1)		
Traversed Cluster Head 1 (3)		

When node C receives the C_RCACHED packet from its cluster head, it will add a route entry to the destination K as following.

Node C Routing Table

Destination IP	Next Hop Node IP (Neighbour)	Next Hop Cluster Head IP	Route [Traversed Cluster heads]	Timer	Route State
K	3	3	[2, 3, 6]		Valid

Then Node C sends a C_RSETUP packet to its next hop node 3 on the route to the destination K.

The C_RSETUP packet has the following format.

0…7	8…15	16…31
Type (C_ RSETUP)		Flags
Source Cluster Head IP (2)		
Source IP (C)		
C_RREQ Sequence ID (-)		
Destination Cluster Head IP (6)		
Destination IP (K)		
Traversed Cluster Hop Count (1)		
Traversed Cluster Head 1 (3)		

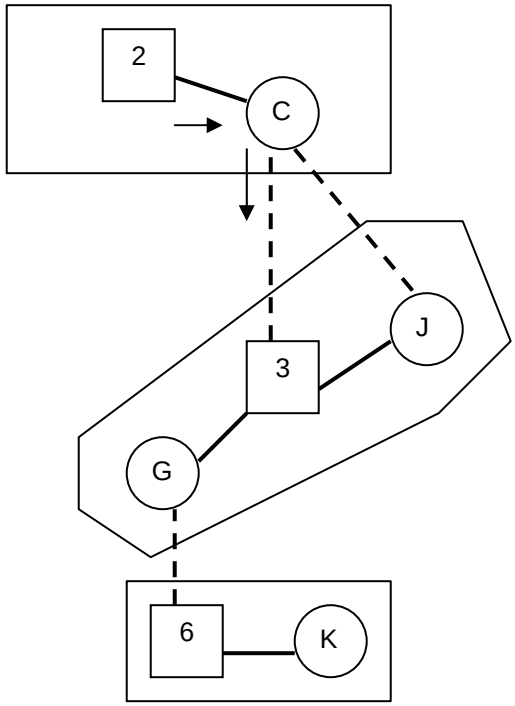


Figure 12 C_RSETUP Propagation From Source Cluster

7.3.2 Intermediate And Destination Cluster

When a node receives a C_RSETUP packet from its neighbour, it will set-up a route back to the source node if no route existing, then set up a route to the destination with the route in the C_RSETUP packet. A node propagates the received C_RSETUP packet to the next hop node towards the destination.

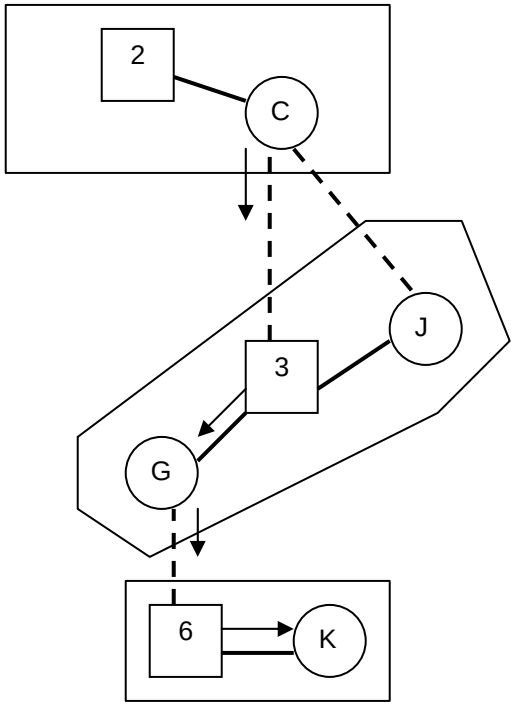


Figure 13 C_RSETUP Propagation In Intermediate Cluster

Node 3 Routing Table

Destination IP	Next Hop Node IP (Neighbour)	Next Hop Cluster Head IP	Route [Traversed Cluster heads]	Timer	Route State
K	G	6	[3, 6]		Valid
C	C	2	[3, 2]		Valid

Node G Routing Table

Destination IP	Next Hop Node IP (Neighbour)	Next Hop Cluster Head IP	Route [Traversed Cluster heads]	Timer	Route State
K	G	6	[3, 6]		Valid
C	3	2	[3, 2]		Valid

Note: The route entry to the destination K maybe already existing (set-up by C_RREP)

Node 6 Routing Table

Destination IP	Next Hop Node IP (Neighbour)	Next Hop Cluster Head IP	Route [Traversed Cluster heads]	Timer	Route State
C	G	3	[6, 3, 2]		Valid

Node K Routing Table

Destination IP	Next Hop Node IP (Neighbour)	Next Hop Cluster Head IP	Route [Traversed Cluster heads]	Timer	Route State
C	6	3	[6, 3, 2]		Valid

7.4 Route Maintenance

7.4.1 General Route Management

1). Un-used route is deleted after a timing period.

2). If a node lost a neighbour node, it marks all routes, in its routing table with the next hop node as the lost neighbour, as need repair.

7.4.2 Route Repair

7.4.2.1 Route Repair Without Back Tracking (Cluster Member)

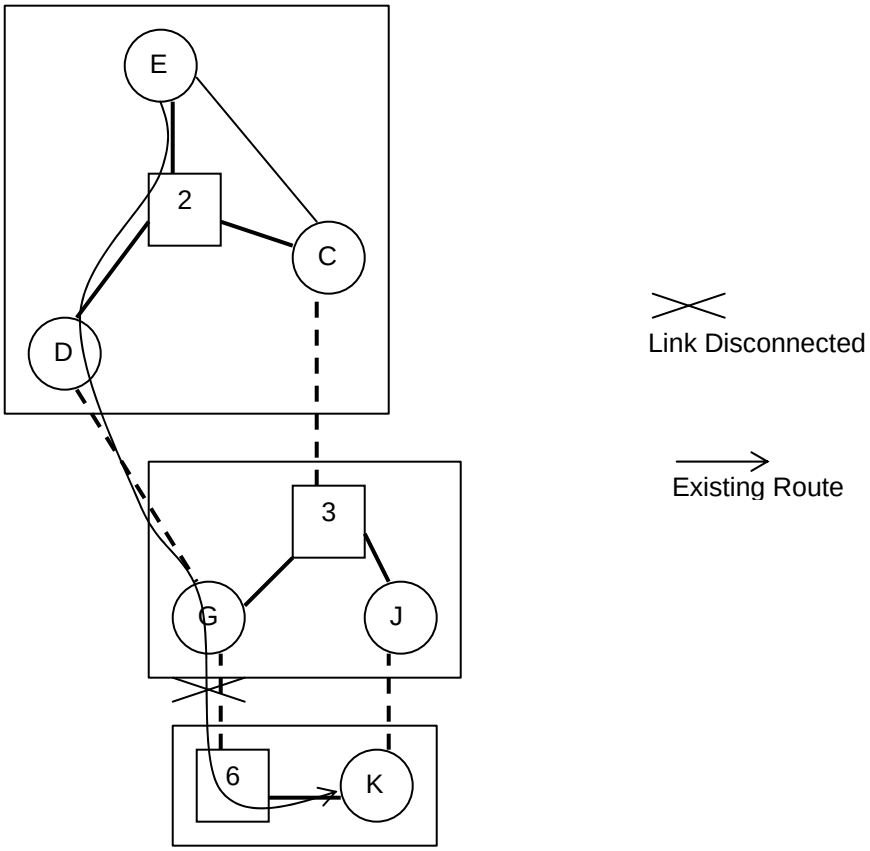


Figure 14 Route Repair

When node G detected that it lost a neighbour node 6, which it currently has a route as the next hop node, then node G marks that route as need-repair.

Node G Routing Table

Destination IP	Next Hop Node IP (Neighbour)	Next Hop Cluster Head IP	Route [Traversed Cluster heads]	Timer	Route State
E	D	2	[3, 2]		Valid
K	**6**	**6**	**[3, 6]**		**Need-Repair**

When node G receives a data packet with the destination K, it checks its routing table for a route to the destination K, a route is existing but the route to node K is marked as Need-Repair. <1> Node G checks whether it has another neighbour node belongs to the next hop cluster for the route entry K; Node G has no neighbour node belonging to the next hop cluster 6. <2> If the data packet is forwarded to it from its cluster head, it sends a C_RREPAIR_BT (Cluster Route Repair Back Tracking) packet to its cluster head (see next section for details). <3> The data packet is not from its cluster head, so it sends a C_RREPAIR packet to its cluster head, and updates its route entry for the destination with the next hop node as its cluster head, then forwards the data packet to its cluster head.

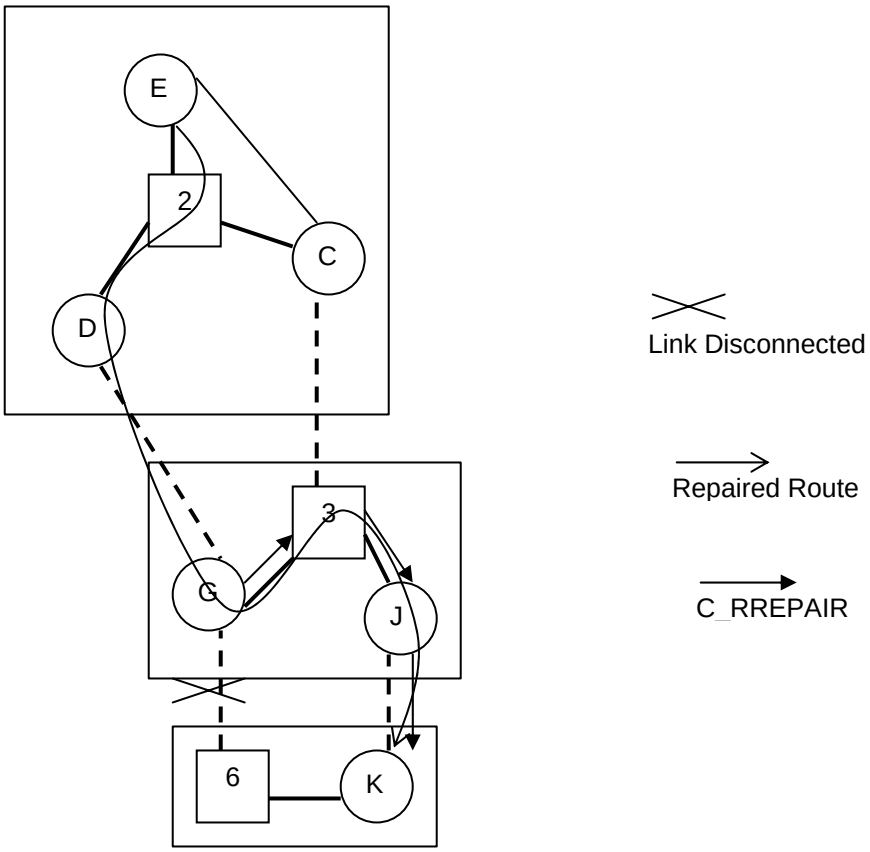


Figure 15 Repair Route Without Back Tracking

When a node receives C_RREPAIR packet, if it could not set-up a route to the destination, it will return a C_RUNREACH packet to the node which sent the C_RREPAIR packet to it.

The C_RREPAIR has the following format.

0…7	8…15	16…31
Type (C_RREPAIR)		Flags
Source Cluster Head IP (2)		
Source IP (E)		
C_RREQ Sequence ID (-)		
Destination Cluster Head IP (6)		
Destination IP (K)		
Traversed Cluster Hop Count (1)		
Traversed Cluster Head 1 (3)		

When a node forwards a C_RREPAIR packet to one of its neighbour towards the next hop cluster, it updates its route entry to the destination with that neighbour as the next hop node.

7.4.2.2 Route Repair With Back Tracking (Cluster Member)

When a node receives a data packet from its cluster head, and the data packet has a route entry marked as Need-Repair, and it could not find another neighbour which belongs to the next hop cluster on the route to the destination, the node will send a C_RREPAIR_BT packet to its cluster head. The node also updates the route with the next hop to its cluster head.

The C_RREPAIR_BT has the following format.

0…7	8…15	16…31
Type (C_RREPAIR_BT)		Flags
Source Cluster Head IP (2)		
Source IP (E)		
C_RREQ Sequence ID (-)		
Destination Cluster Head IP (6)		
Destination IP (K)		
Traversed Cluster Hop Count (1)		
Traversed Cluster Head 1 (3)		

When a cluster head receives C_RREPAIR_BT from one of its members, it updates its neighbour cluster table, then searches for another gateway member towards the next hop cluster. <1> If the cluster head could not find another gateway, it deletes its route entry to the destination, then sends a C_RUNREACH packet to the cluster member which sent it the C_RREPAIR_BT, also, it sends a C_RUNREACH towards the source of the C_RREPAIR_BT. <2> If the cluster head has another gateway to the next hop cluster, it sends a C_RREPAIR packet towards the destination, also updates its route entry for the destination.

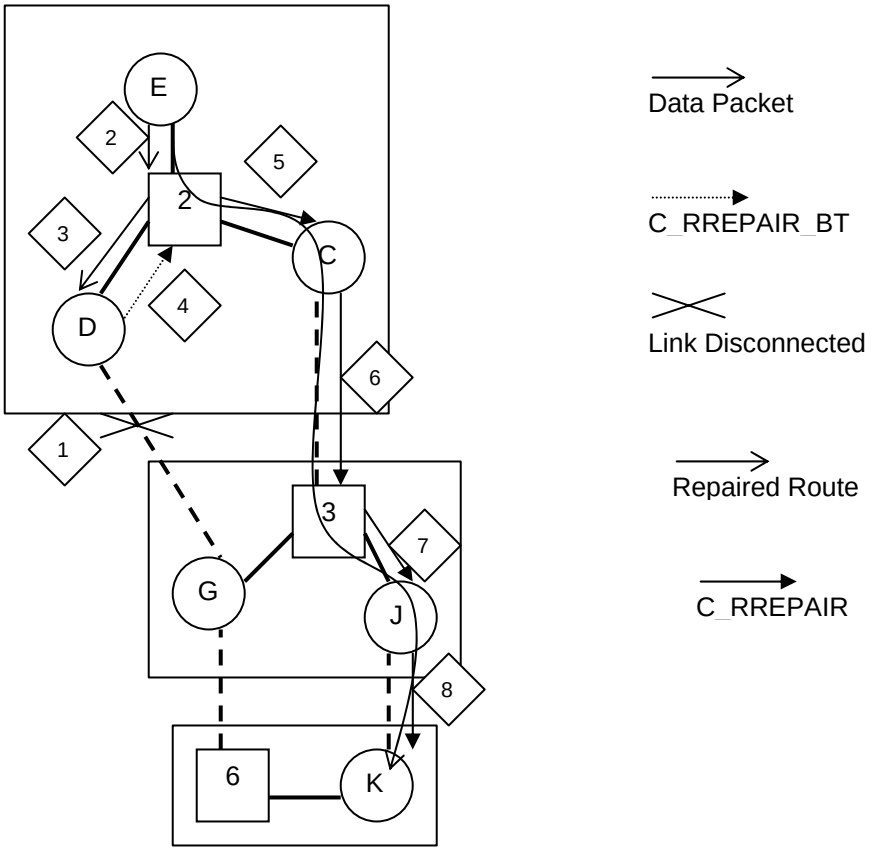


Figure 16 Route Repair With Back Tracking

7.4.2.3 Route Repair (Cluster head)

When a cluster head receives a data packet which it has a route marked as need-repair, it searches its neighbour cluster table for a gateway to the next hop cluster, if a gateway is found, then the cluster head sends a C_RREPAIR message to the gateway, and updates its route entry with the next hop node to the gateway node; otherwise, it deletes the route entry, and sends a C_RUNREACH message to the node which sent the data packet to it.

7.5 Route Error Processing

7.5.1 No Route For A Data Packet

If a node receives a data packet, which it has no route to the destination, it returns a route unreachable error message to the neighbour node which forwarded the data packet.

C_RUNREACH message format:

0…7	8…15	16…31
Type (C_RUNREACH)		Flags
Source IP		
Destination IP		

If a node receives a C_ RUNREACH message from its neighbour node, it deletes the route to the destination in its routing table. If the node has a route in its routing table to the source node of this received C_ RUNREACH, this C_ RUNREACH is forwarded to the next hop node towards the source node. In this way, the unreachable route is deleted hop-by-hop following the route to the source node (if the route is not symmetric, the C_RUNREACH may be propagated until the source node is reached or a node has no route back to the source of the data packet).

When the source node receives a C_ RUNREACH, it deletes its current route to the destination, then re-discovers a new route.

7.5.2 Can Not Forward A C_RREP To Next Hop Cluster

When a node receives a C_RREP from its neighbour node, it searches its neighbours for a gateway to the next hop cluster in the received C_RREP. <1> If the node could not find a neighbour to the next hop cluster, it forwards the C_RREP to its cluster head after setting-up a route to the source with its cluster head as the next hop node; <2> If the C_RREP is sent by its cluster head, it sends C_RREP_BT (Cluster Route Reply Back Tracking, see following paragraphs for details) to its cluster head.

When a cluster head receives a C_RREP from its neighbour node (cluster member or not), it searches its cluster members (including itself) for a gateway to the next hop cluster in the received C_RREP, if the cluster head could not find a gateway to the next hop cluster, it sends a C_RUNREACH to the destination in the C_RREP. The route to the source of the C_RREP is deleted when a node forwarding the C_ RUNREACH to the destination.

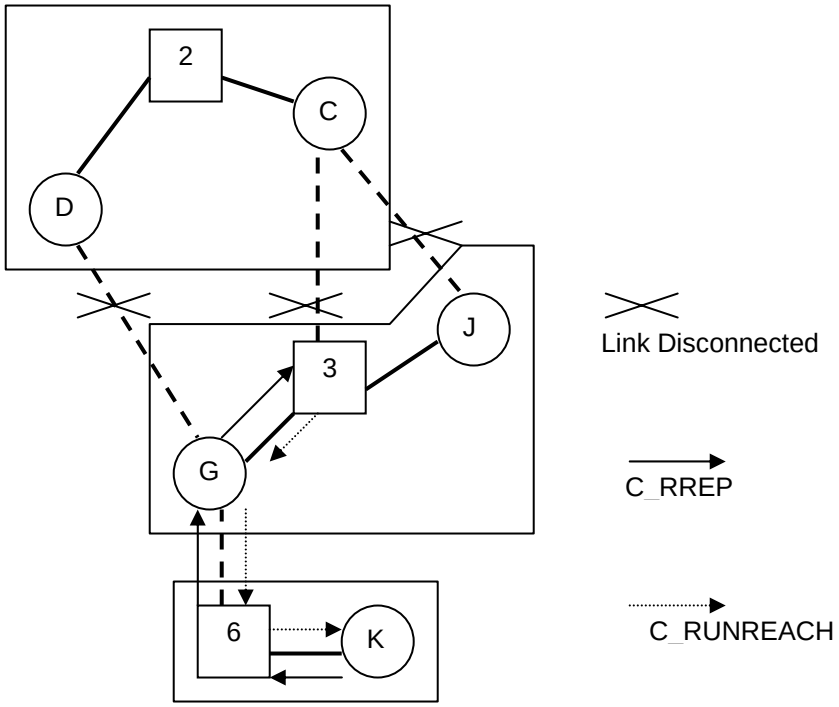


Figure 17 Route Error When Forwarding C_RREP

When a cluster head receives a C_RREP_BT packet from one of its members, it first updates its neighbour cluster table, then if the cluster head has another gateway member (

including itself) towards the next hop cluster, it will update the route to the source in the C_RREP_BT, then forwards a C_RREP to the newly picked up gateway member (C_RREP packet is formed from the received C_RREP_BT packet).

The C_RREP_BT packet format is as following.

0…7	8…15	16…31
Type (C_RREP_BT)		Flags
Source Cluster Head IP (2)		
Source IP (E)		
C_RREQ Sequence ID (123)		
Destination Cluster Head IP (6)		
Destination IP (K)		
Traversed Cluster Hop Count (1)		
Traversed Cluster Head 1 (3)		

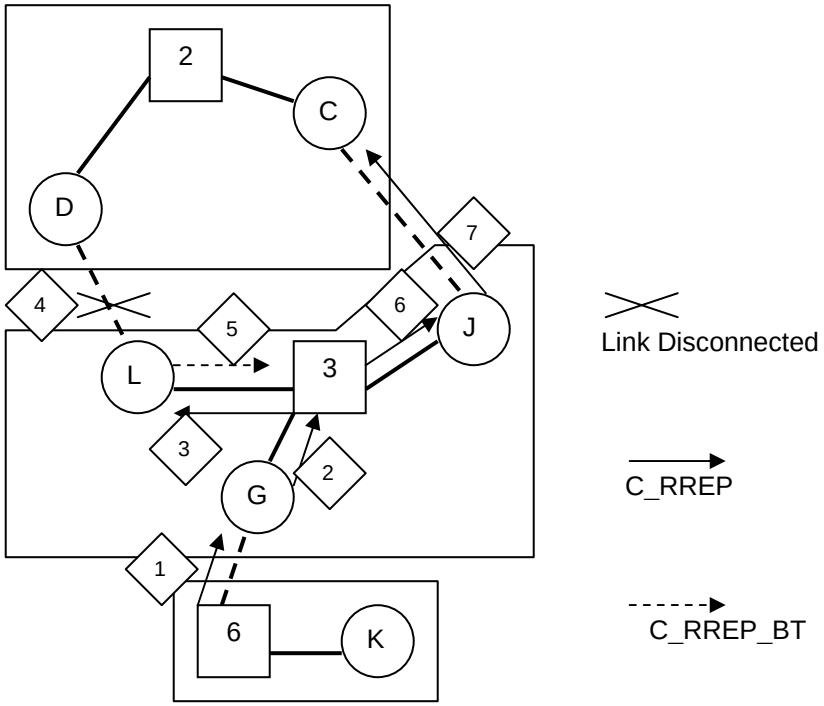


Figure 18 Route Reply Back Tracking

7.5.3 Can Not Forward A C_RSETUP To Next Hop Cluster

When a node receives a C_RSETUP from its neighbour, it will set up a route back to the source, then if it has a neighbour to the next hop cluster, it will forward the C_RSETUP to that neighbour; otherwise it forwards the received C_RSETUP to its cluster head; If the C_RSETUP is already forwarded to it from its cluster head, it will send a C_RSETUP_BT (Cluster Route Set-up Back Tracking, see following paragraphs for details) back to its cluster head.

When a cluster head receives a C_RSETUP packet from its neighbour, if it could not find a gateway to the next hop cluster in the C_RSETUP packet, it will send a C_RUNREACH for the destination in the C_RSETUP packet by following the route back to the source. The route to the destination is deleted when a node forwards the C_RUNREACH back to the source.

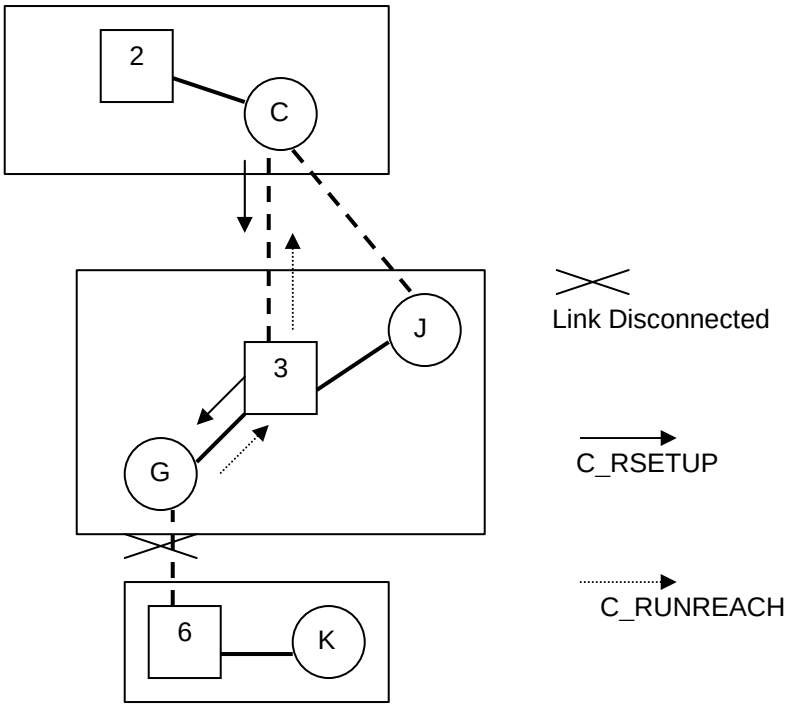


Figure 19 Route Error When Forwarding C_RUNREACH

When a cluster head receives a C_RSETUP_BT packet from one of its members, it first updates its neighbour cluster table, then if the cluster head has another gateway member (including itself) towards the next hop cluster, it will update the route to the destination in the

C_RSETUP_BT, then forwards a C_RSETUP to the newly picked up gateway member (C_RSETUP packet is formed from the received C_RSETUP_BT packet).

The C_RSETUP_BT packet format is as following.

0…7	8…15	16…31
Type (C_RSETUP_BT)		Flags
Source Cluster Head IP (2)		
Source IP (C)		
C_RREQ Sequence ID (-)		
Destination Cluster Head IP (6)		
Destination IP (K)		
Traversed Cluster Hop Count (1)		
Traversed Cluster Head 1 (3)		

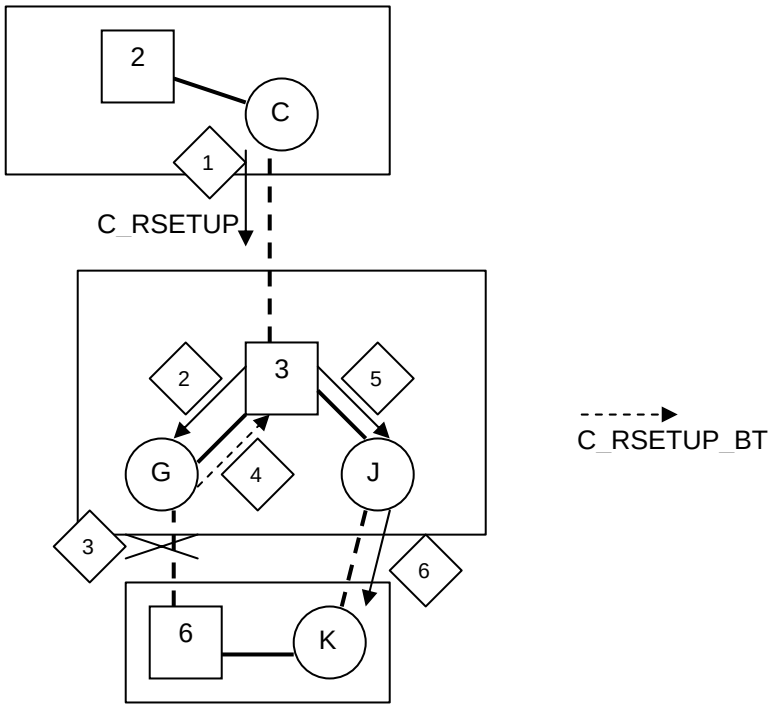


Figure 20 C_RSETUP Back Tracking

8 Interaction With Cluster Formation Algorithm

8.1 A Node Joins Another Cluster

If a node joins another cluster, it will flush its routing table.

8.2 A Cluster Head Becomes A Cluster Member

If a node switches its cluster role from a cluster head to a cluster member, it will flush its routing table. Also, the neighbour cluster table and the cached route table are deleted since these tables are not maintained by a cluster member.

9 References

[1] Charles E. Perkins, Ad Hoc Networking, Chapter 7 ZRP by Zygmunt J. Haas & Marc R. Pearlman, Cornell University.

[2] Cluster formation algorithm by professor Michel Barbeau.

A Protocol Design Testing

To test this cluster routing protocol design, I implement the route discovery part of this routing protocol under Cygwin environment (a Linux-like environment for Windows).

A.1 Testing Framework

In this testing, a network node is implemented as a process, and the link between two network nodes is a Unix domain socket. When a node sends a network packet to another node, it sends this message to the Unix domain socket which connects these two nodes; if a node broadcasts a network packet, it sends this message to all Unix domain sockets which connect this node to its neighbour nodes.

The testing network looks like the following diagram.

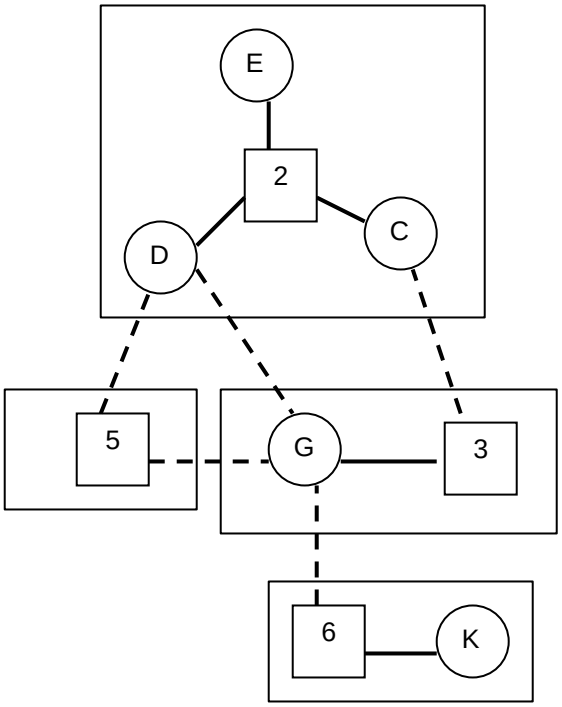


Figure 21 Testing Network

Each network node has an IP version 4 address.

A.2 Implementation Protocol Stack

The following is the protocol stack for this testing.

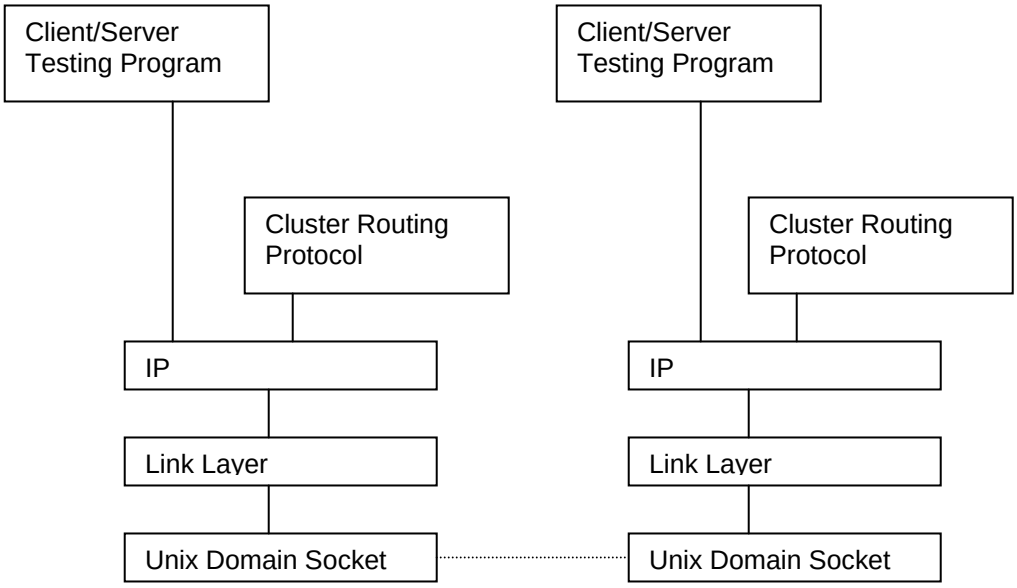


Figure 22 Testing Protocol Stack

A.3 Implementation Software

Each protocol stack is implemented as a C++ class. Please refer to the source code files for detailed implementation.

A.4 Testing Result

In this testing, Node E is a testing client, and all other nodes runs as a server. Testing client and server have no difference when running cluster routing protocol, the only difference is that the testing client sends a message to the server periodically, and the server echoes the received client message back to the client.

The following is an extraction of the testing output when node E sends messages to node K. When reading the testing tracing, the network node addresses are configured as

```
Node 2 129.1.3.25 Node E 129.1.3.5  Node C 129.1.3.70  Node D 129.2.3.7
Node 3 47.0.1.5   Node G 47.0.2.10
Node 5 53.11.2.8
Node 6 50.2.2.3   Node K 51.3.3.4
```

A.4.1 Node E

```
Initialize log
Log initialized successfully
Wed Jul 30 22:28:48 2003 [Thread 10289184] Client started
Wed Jul 30 22:28:48 2003 [Thread 10289184] read_config: Begin reading configuration file
Wed Jul 30 22:28:48 2003 [Thread 10297248] IPProtocol : event handler thread started
Wed Jul 30 22:28:48 2003 [Thread 10289184] LoProtocol : local unix socket
<LO_129.1.3.5.sock>
Wed Jul 30 22:28:48 2003 [Thread 10289184] LoProtocol : add neighbour (129.1.3.25)
Wed Jul 30 22:28:48 2003 [Thread 10289184] CRoutingImpl : add_neighbour success
Wed Jul 30 22:28:48 2003 [Thread 10289184]   neighbour 129.1.3.25 -> cluster head
129.1.3.25
Wed Jul 30 22:28:48 2003 [Thread 10289184] End reading configuration file
Wed Jul 30 22:28:48 2003 [Thread 10289184] Client starting testing loop
Wed Jul 30 22:28:48 2003 [Thread 10298064] LoProtocol : Receiving loop started
Wed Jul 30 22:28:48 2003 [Thread 10298064] LoProtocol : waiting for incoming message
Wed Jul 30 22:28:48 2003 [Thread 10297248]
Wed Jul 30 22:28:48 2003 [Thread 10297248] ------CRoutingImpl------
Wed Jul 30 22:28:48 2003 [Thread 10289184] Client tester SendLoop() : xPush() succeeded
Wed Jul 30 22:28:48 2003 [Thread 10297248] ->Sent routing message
Wed Jul 30 22:28:48 2003 [Thread 10297248] Cluster Route Request
Wed Jul 30 22:28:48 2003 [Thread 10297248] Type 1 Flags 0
Wed Jul 30 22:28:48 2003 [Thread 10297248] Src Cluster Head 129.1.3.25
Wed Jul 30 22:28:48 2003 [Thread 10297248] Src Node         129.1.3.5
Wed Jul 30 22:28:48 2003 [Thread 10297248] Seq Id           0
Wed Jul 30 22:28:48 2003 [Thread 10297248] Dst Cluster Head 0.0.0.0
Wed Jul 30 22:28:48 2003 [Thread 10297248] Dst Node         51.3.3.4
Wed Jul 30 22:28:48 2003 [Thread 10297248] Traversed Cluster Count 0
Wed Jul 30 22:28:48 2003 [Thread 10297248]
Wed Jul 30 22:28:48 2003 [Thread 10297248] IPProtocol : xPush(msg) dst (51.3.3.4) packet
queued.
Wed Jul 30 22:28:48 2003 [Thread 10297248] IPProtocol : xPush(msg) dst (129.1.3.25) route
found.
Wed Jul 30 22:28:48 2003 [Thread 10297248]                    next hop (129.1.3.25).
Wed Jul 30 22:28:48 2003 [Thread 10297248] LoProcotol : send success (LO_129.1.3.25.sock)
Wed Jul 30 22:28:49 2003 [Thread 10297248] IPProtocol xDemux : received packet in host
byte order
Wed Jul 30 22:28:49 2003 [Thread 10298064] LoProtocol : waiting for incoming message
Wed Jul 30 22:28:49 2003 [Thread 10297248] IP Header:
Wed Jul 30 22:28:49 2003 [Thread 10297248]  Ver 4 Hl 5 TOS 0
Wed Jul 30 22:28:49 2003 [Thread 10297248]  Len 48 Ident 1 offset 0
Wed Jul 30 22:28:49 2003 [Thread 10297248]  TTL 1 Pro 50 Sum 0
Wed Jul 30 22:28:49 2003 [Thread 10297248]  Src 129.1.3.25
Wed Jul 30 22:28:49 2003 [Thread 10297248]  Dst 255.255.255.255
Wed Jul 30 22:28:49 2003 [Thread 10297248]
Wed Jul 30 22:28:49 2003 [Thread 10297248]
Wed Jul 30 22:28:49 2003 [Thread 10297248] ------CRoutingImpl------
Wed Jul 30 22:28:49 2003 [Thread 10297248] ->Received routing message from 129.1.3.25
Wed Jul 30 22:28:49 2003 [Thread 10297248] Cluster Route Request
Wed Jul 30 22:28:49 2003 [Thread 10297248] Type 1 Flags 0
Wed Jul 30 22:28:49 2003 [Thread 10297248] Src Cluster Head 129.1.3.25
Wed Jul 30 22:28:49 2003 [Thread 10297248] Src Node         129.1.3.5
Wed Jul 30 22:28:49 2003 [Thread 10297248] Seq Id           1
Wed Jul 30 22:28:49 2003 [Thread 10297248] Dst Cluster Head 0.0.0.0
Wed Jul 30 22:28:49 2003 [Thread 10297248] Dst Node         51.3.3.4
Wed Jul 30 22:28:49 2003 [Thread 10297248] Traversed Cluster Count 0
Wed Jul 30 22:28:49 2003 [Thread 10297248]
Wed Jul 30 22:28:52 2003 [Thread 10297248] IPProtocol xDemux : received packet in host
byte order
Wed Jul 30 22:28:52 2003 [Thread 10297248] IP Header:
Wed Jul 30 22:28:52 2003 [Thread 10298064] LoProtocol : waiting for incoming message
Wed Jul 30 22:28:52 2003 [Thread 10297248]  Ver 4 Hl 5 TOS 0
Wed Jul 30 22:28:52 2003 [Thread 10297248]  Len 52 Ident 2 offset 0
Wed Jul 30 22:28:52 2003 [Thread 10297248]  TTL 16 Pro 50 Sum 0
Wed Jul 30 22:28:52 2003 [Thread 10297248]  Src 129.1.3.25
Wed Jul 30 22:28:52 2003 [Thread 10297248]  Dst 129.1.3.5
Wed Jul 30 22:28:52 2003 [Thread 10297248]
Wed Jul 30 22:28:52 2003 [Thread 10297248]
Wed Jul 30 22:28:52 2003 [Thread 10297248] ------CRoutingImpl------
```

```
Wed Jul 30 22:28:52 2003 [Thread 10297248] ->Received routing message from 129.1.3.25
Wed Jul 30 22:28:52 2003 [Thread 10297248] Cluster Route Reply
Wed Jul 30 22:28:52 2003 [Thread 10297248] Type 2 Flags 0
Wed Jul 30 22:28:52 2003 [Thread 10297248] Src Cluster Head 129.1.3.25
Wed Jul 30 22:28:52 2003 [Thread 10297248] Src Node         129.1.3.5
Wed Jul 30 22:28:52 2003 [Thread 10297248] Seq Id           1
Wed Jul 30 22:28:52 2003 [Thread 10297248] Dst Cluster Head 50.2.2.3
Wed Jul 30 22:28:52 2003 [Thread 10297248] Dst Node         51.3.3.4
Wed Jul 30 22:28:52 2003 [Thread 10297248] Traversed Cluster Count 1
Wed Jul 30 22:28:52 2003 [Thread 10297248]   Cluster 47.0.1.5
Wed Jul 30 22:28:52 2003 [Thread 10297248]
Wed Jul 30 22:28:52 2003 [Thread 10297248] CRoutingImpl : add route entry
Wed Jul 30 22:28:52 2003 [Thread 10297248]     dst 51.3.3.4 -> next hop node 129.1.3.25
Wed Jul 30 22:28:52 2003 [Thread 10297248] IPProtocol : xPush(msg) dst (51.3.3.4) route
found.
Wed Jul 30 22:28:52 2003 [Thread 10297248]                    next hop (129.1.3.25).
Wed Jul 30 22:28:52 2003 [Thread 10297248] LoProcotol : send success (LO_129.1.3.25.sock)
Wed Jul 30 22:28:53 2003 [Thread 10297248] IPProtocol xDemux : received packet in host
byte order
Wed Jul 30 22:28:53 2003 [Thread 10297248] IP Header:
Wed Jul 30 22:28:53 2003 [Thread 10298064] LoProtocol : waiting for incoming message
Wed Jul 30 22:28:53 2003 [Thread 10297248]  Ver 4 Hl 5 TOS 0
Wed Jul 30 22:28:53 2003 [Thread 10297248]  Len 56 Ident 2 offset 0
Wed Jul 30 22:28:53 2003 [Thread 10297248]  TTL 12 Pro 17 Sum 0
Wed Jul 30 22:28:53 2003 [Thread 10297248]  Src 51.3.3.4
Wed Jul 30 22:28:53 2003 [Thread 10297248]  Dst 129.1.3.5
Wed Jul 30 22:28:53 2003 [Thread 10297248]
Wed Jul 30 22:28:53 2003 [Thread 10297248] ClientTester xDemux : echo: <Client 10289184 :
sequence 1> from 51.3.3.4
Wed Jul 30 22:28:53 2003 [Thread 10297248] IPProtocol : xPush(msg) dst (51.3.3.4) route
found.
```

A.4.2 Node G

```
Initialize log
Log initialized successfully
Wed Jul 30 22:27:59 2003 [Thread 10289184] Server started
Wed Jul 30 22:27:59 2003 [Thread 10297248] IPProtocol : event handler thread started
Wed Jul 30 22:27:59 2003 [Thread 10289184] read_config: Begin reading configuration file
Wed Jul 30 22:27:59 2003 [Thread 10289184] LoProtocol : local unix socket
<LO_47.0.2.10.sock>
Wed Jul 30 22:27:59 2003 [Thread 10289184] LoProtocol : add neighbour (47.0.1.5)
Wed Jul 30 22:27:59 2003 [Thread 10289184] CRoutingImpl : add_neighbour success
Wed Jul 30 22:27:59 2003 [Thread 10289184]   neighbour 47.0.1.5 -> cluster head 47.0.1.5
Wed Jul 30 22:27:59 2003 [Thread 10289184] LoProtocol : add neighbour (129.2.3.7)
Wed Jul 30 22:27:59 2003 [Thread 10289184] CRoutingImpl : add_neighbour success
Wed Jul 30 22:27:59 2003 [Thread 10289184]   neighbour 129.2.3.7 -> cluster head
129.1.3.25
Wed Jul 30 22:27:59 2003 [Thread 10289184] LoProtocol : add neighbour (53.11.2.8)
Wed Jul 30 22:27:59 2003 [Thread 10289184] CRoutingImpl : add_neighbour success
Wed Jul 30 22:27:59 2003 [Thread 10289184]   neighbour 53.11.2.8 -> cluster head
53.11.2.8
Wed Jul 30 22:27:59 2003 [Thread 10289184] LoProtocol : add neighbour (50.2.2.3)
Wed Jul 30 22:27:59 2003 [Thread 10289184] CRoutingImpl : add_neighbour success
Wed Jul 30 22:27:59 2003 [Thread 10289184]   neighbour 50.2.2.3 -> cluster head 50.2.2.3
Wed Jul 30 22:27:59 2003 [Thread 10289184] End reading configuration file
Wed Jul 30 22:28:00 2003 [Thread 10298056] LoProtocol : Receiving loop started
Wed Jul 30 22:28:00 2003 [Thread 10298056] LoProtocol : waiting for incoming message
Wed Jul 30 22:28:49 2003 [Thread 10298056] LoProtocol : waiting for incoming message
Wed Jul 30 22:28:49 2003 [Thread 10297248] IPProtocol xDemux : received packet in host
byte order
Wed Jul 30 22:28:49 2003 [Thread 10297248] IP Header:
Wed Jul 30 22:28:49 2003 [Thread 10297248]  Ver 4 Hl 5 TOS 0
Wed Jul 30 22:28:49 2003 [Thread 10297248]  Len 48 Ident 1 offset 0
Wed Jul 30 22:28:49 2003 [Thread 10297248]  TTL 16 Pro 50 Sum 0
Wed Jul 30 22:28:49 2003 [Thread 10297248]  Src 129.2.3.7
Wed Jul 30 22:28:49 2003 [Thread 10297248]  Dst 47.0.2.10
Wed Jul 30 22:28:49 2003 [Thread 10297248]
Wed Jul 30 22:28:49 2003 [Thread 10297248]
Wed Jul 30 22:28:49 2003 [Thread 10297248] ------CRoutingImpl------
```

```
Wed Jul 30 22:28:49 2003 [Thread 10297248] ->Received routing message from 129.2.3.7
Wed Jul 30 22:28:49 2003 [Thread 10297248] Cluster Route Request
Wed Jul 30 22:28:49 2003 [Thread 10297248] Type 1 Flags 0
Wed Jul 30 22:28:49 2003 [Thread 10297248] Src Cluster Head 129.1.3.25
Wed Jul 30 22:28:49 2003 [Thread 10297248] Src Node         129.1.3.5
Wed Jul 30 22:28:49 2003 [Thread 10297248] Seq Id           1
Wed Jul 30 22:28:49 2003 [Thread 10297248] Dst Cluster Head 0.0.0.0
Wed Jul 30 22:28:49 2003 [Thread 10297248] Dst Node         51.3.3.4
Wed Jul 30 22:28:49 2003 [Thread 10297248] Traversed Cluster Count 0
Wed Jul 30 22:28:49 2003 [Thread 10298056] LoProtocol : waiting for incoming message
Wed Jul 30 22:28:49 2003 [Thread 10297248]
Wed Jul 30 22:28:49 2003 [Thread 10297248] IPProtocol : xPush(msg) dst (47.0.1.5) route
found.
Wed Jul 30 22:28:49 2003 [Thread 10297248]                    next hop (47.0.1.5).
Wed Jul 30 22:28:49 2003 [Thread 10297248] LoProcotol : send success (LO_47.0.1.5.sock)
Wed Jul 30 22:28:50 2003 [Thread 10297248] IPProtocol xDemux : received packet in host
byte order
Wed Jul 30 22:28:50 2003 [Thread 10297248] IP Header:
Wed Jul 30 22:28:50 2003 [Thread 10297248]  Ver 4 Hl 5 TOS 0
Wed Jul 30 22:28:50 2003 [Thread 10297248]  Len 52 Ident 1 offset 0
Wed Jul 30 22:28:50 2003 [Thread 10298056] LoProtocol : waiting for incoming message
Wed Jul 30 22:28:50 2003 [Thread 10297248]  TTL 1 Pro 50 Sum 0
Wed Jul 30 22:28:50 2003 [Thread 10297248]  Src 53.11.2.8
Wed Jul 30 22:28:50 2003 [Thread 10297248]  Dst 255.255.255.255
Wed Jul 30 22:28:50 2003 [Thread 10297248]
Wed Jul 30 22:28:50 2003 [Thread 10297248]
Wed Jul 30 22:28:50 2003 [Thread 10297248] ------CRoutingImpl------
Wed Jul 30 22:28:50 2003 [Thread 10297248] ->Received routing message from 53.11.2.8
Wed Jul 30 22:28:50 2003 [Thread 10297248] Cluster Route Request
Wed Jul 30 22:28:50 2003 [Thread 10297248] Type 1 Flags 0
Wed Jul 30 22:28:50 2003 [Thread 10297248] Src Cluster Head 129.1.3.25
Wed Jul 30 22:28:50 2003 [Thread 10297248] Src Node         129.1.3.5
Wed Jul 30 22:28:50 2003 [Thread 10297248] Seq Id           1
Wed Jul 30 22:28:50 2003 [Thread 10297248] Dst Cluster Head 0.0.0.0
Wed Jul 30 22:28:50 2003 [Thread 10297248] Dst Node         51.3.3.4
Wed Jul 30 22:28:50 2003 [Thread 10297248] Traversed Cluster Count 1
Wed Jul 30 22:28:50 2003 [Thread 10297248]   Cluster 53.11.2.8
Wed Jul 30 22:28:50 2003 [Thread 10297248]
Wed Jul 30 22:28:50 2003 [Thread 10297248] IPProtocol xDemux : received packet in host
byte order
Wed Jul 30 22:28:50 2003 [Thread 10297248] IP Header:
Wed Jul 30 22:28:50 2003 [Thread 10297248]  Ver 4 Hl 5 TOS 0
Wed Jul 30 22:28:50 2003 [Thread 10297248]  Len 52 Ident 1 offset 0
Wed Jul 30 22:28:50 2003 [Thread 10297248]  TTL 1 Pro 50 Sum 0
Wed Jul 30 22:28:50 2003 [Thread 10297248]  Src 47.0.1.5
Wed Jul 30 22:28:50 2003 [Thread 10297248]  Dst 255.255.255.255
Wed Jul 30 22:28:50 2003 [Thread 10297248]
Wed Jul 30 22:28:50 2003 [Thread 10297248]
Wed Jul 30 22:28:50 2003 [Thread 10297248] ------CRoutingImpl------
Wed Jul 30 22:28:50 2003 [Thread 10297248] ->Received routing message from 47.0.1.5
Wed Jul 30 22:28:50 2003 [Thread 10297248] Cluster Route Request
Wed Jul 30 22:28:50 2003 [Thread 10297248] Type 1 Flags 0
Wed Jul 30 22:28:50 2003 [Thread 10297248] Src Cluster Head 129.1.3.25
Wed Jul 30 22:28:50 2003 [Thread 10297248] Src Node         129.1.3.5
Wed Jul 30 22:28:50 2003 [Thread 10297248] Seq Id           1
Wed Jul 30 22:28:50 2003 [Thread 10297248] Dst Cluster Head 0.0.0.0
Wed Jul 30 22:28:50 2003 [Thread 10297248] Dst Node         51.3.3.4
Wed Jul 30 22:28:50 2003 [Thread 10297248] Traversed Cluster Count 1
Wed Jul 30 22:28:50 2003 [Thread 10297248]   Cluster 47.0.1.5
Wed Jul 30 22:28:50 2003 [Thread 10297248]
Wed Jul 30 22:28:50 2003 [Thread 10297248] IPProtocol : xPush(msg) dst (50.2.2.3) route
found.
Wed Jul 30 22:28:50 2003 [Thread 10297248]                    next hop (50.2.2.3).
Wed Jul 30 22:28:50 2003 [Thread 10297248] LoProcotol : send success (LO_50.2.2.3.sock)
Wed Jul 30 22:28:50 2003 [Thread 10297248] IPProtocol : xPush(msg) dst (53.11.2.8) route
found.
Wed Jul 30 22:28:50 2003 [Thread 10297248]                    next hop (53.11.2.8).
Wed Jul 30 22:28:50 2003 [Thread 10297248] LoProcotol : send success (LO_53.11.2.8.sock)
Wed Jul 30 22:28:51 2003 [Thread 10298056] LoProtocol : waiting for incoming message
Wed Jul 30 22:28:51 2003 [Thread 10297248] IPProtocol xDemux : received packet in host
byte order
```

```
Wed Jul 30 22:28:51 2003 [Thread 10297248] IP Header:
Wed Jul 30 22:28:51 2003 [Thread 10297248]  Ver 4 Hl 5 TOS 0
Wed Jul 30 22:28:51 2003 [Thread 10297248]  Len 52 Ident 2 offset 0
Wed Jul 30 22:28:51 2003 [Thread 10297248]  TTL 16 Pro 50 Sum 0
Wed Jul 30 22:28:51 2003 [Thread 10297248]  Src 50.2.2.3
Wed Jul 30 22:28:51 2003 [Thread 10297248]  Dst 47.0.2.10
Wed Jul 30 22:28:51 2003 [Thread 10297248]
Wed Jul 30 22:28:51 2003 [Thread 10297248]
Wed Jul 30 22:28:51 2003 [Thread 10297248] ------CRoutingImpl------
Wed Jul 30 22:28:51 2003 [Thread 10297248] ->Received routing message from 50.2.2.3
Wed Jul 30 22:28:51 2003 [Thread 10297248] Cluster Route Reply
Wed Jul 30 22:28:51 2003 [Thread 10297248] Type 2 Flags 0
Wed Jul 30 22:28:51 2003 [Thread 10297248] Src Cluster Head 129.1.3.25
Wed Jul 30 22:28:51 2003 [Thread 10297248] Src Node         129.1.3.5
Wed Jul 30 22:28:51 2003 [Thread 10297248] Seq Id           1
Wed Jul 30 22:28:51 2003 [Thread 10297248] Dst Cluster Head 50.2.2.3
Wed Jul 30 22:28:51 2003 [Thread 10297248] Dst Node         51.3.3.4
Wed Jul 30 22:28:51 2003 [Thread 10297248] Traversed Cluster Count 1
Wed Jul 30 22:28:51 2003 [Thread 10297248]   Cluster 47.0.1.5
Wed Jul 30 22:28:51 2003 [Thread 10297248]
Wed Jul 30 22:28:51 2003 [Thread 10297248] CRoutingImpl : add route entry
Wed Jul 30 22:28:51 2003 [Thread 10297248]     dst 51.3.3.4 -> next hop node 50.2.2.3
Wed Jul 30 22:28:51 2003 [Thread 10297248]
CRoutingImpl : forward route reply to next hop cluster 129.1.3.25
Wed Jul 30 22:28:51 2003 [Thread 10297248] CRoutingImpl : add route entry
Wed Jul 30 22:28:51 2003 [Thread 10297248]     dst 129.1.3.5 -> next hop node 129.2.3.7
Wed Jul 30 22:28:51 2003 [Thread 10297248]
CRoutingImpl : forward route reply to next hop node 129.2.3.7
Wed Jul 30 22:28:51 2003 [Thread 10297248] IPProtocol : xPush(msg) dst (129.2.3.7) route
found.
Wed Jul 30 22:28:51 2003 [Thread 10297248]                    next hop (129.2.3.7).
Wed Jul 30 22:28:51 2003 [Thread 10297248] LoProcotol : send success (LO_129.2.3.7.sock)
Wed Jul 30 22:28:52 2003 [Thread 10297248] IPProtocol xDemux : received packet in host
byte order
Wed Jul 30 22:28:52 2003 [Thread 10297248] IP Header:
Wed Jul 30 22:28:52 2003 [Thread 10297248]  Ver 4 Hl 5 TOS 0
Wed Jul 30 22:28:52 2003 [Thread 10298056] LoProtocol : waiting for incoming message
Wed Jul 30 22:28:52 2003 [Thread 10297248]  Len 50 Ident 1 offset 0
Wed Jul 30 22:28:52 2003 [Thread 10297248]  TTL 14 Pro 17 Sum 0
Wed Jul 30 22:28:52 2003 [Thread 10297248]  Src 129.1.3.5
Wed Jul 30 22:28:52 2003 [Thread 10297248]  Dst 51.3.3.4
Wed Jul 30 22:28:52 2003 [Thread 10297248]
Wed Jul 30 22:28:52 2003 [Thread 10297248] IPProtocol : xPush(msg) dst (51.3.3.4) route
found.
Wed Jul 30 22:28:52 2003 [Thread 10297248]                    next hop (50.2.2.3).
Wed Jul 30 22:28:52 2003 [Thread 10297248] LoProcotol : send success (LO_50.2.2.3.sock)
Wed Jul 30 22:28:52 2003 [Thread 10297248] IPProtocol xDemux : received packet in host
byte order
Wed Jul 30 22:28:52 2003 [Thread 10297248] IP Header:
Wed Jul 30 22:28:52 2003 [Thread 10297248]  Ver 4 Hl 5 TOS 0
Wed Jul 30 22:28:52 2003 [Thread 10298056] LoProtocol : waiting for incoming message
Wed Jul 30 22:28:52 2003 [Thread 10297248]  Len 56 Ident 2 offset 0
Wed Jul 30 22:28:52 2003 [Thread 10297248]  TTL 15 Pro 17 Sum 0
Wed Jul 30 22:28:52 2003 [Thread 10297248]  Src 51.3.3.4
Wed Jul 30 22:28:52 2003 [Thread 10297248]  Dst 129.1.3.5
Wed Jul 30 22:28:52 2003 [Thread 10297248]
Wed Jul 30 22:28:52 2003 [Thread 10297248] IPProtocol : xPush(msg) dst (129.1.3.5) route
found.
Wed Jul 30 22:28:52 2003 [Thread 10297248]                    next hop (129.2.3.7).
Wed Jul 30 22:28:52 2003 [Thread 10297248] LoProcotol : send success (LO_129.2.3.7.sock)
```

A.4.3 Node K

```
Initialize log
Log initialized successfully
Wed Jul 30 22:28:22 2003 [Thread 10289184] Server started
Wed Jul 30 22:28:22 2003 [Thread 10289184] read_config: Begin reading configuration file
Wed Jul 30 22:28:22 2003 [Thread 10289184] LoProtocol : local unix socket
<LO_51.3.3.4.sock>
```

```
Wed Jul 30 22:28:22 2003 [Thread 10289184] LoProtocol : add neighbour (50.2.2.3)
Wed Jul 30 22:28:22 2003 [Thread 10289184] CRoutingImpl : add_neighbour success
Wed Jul 30 22:28:22 2003 [Thread 10297248] IPProtocol : event handler thread started
Wed Jul 30 22:28:22 2003 [Thread 10289184]   neighbour 50.2.2.3 -> cluster head 50.2.2.3
Wed Jul 30 22:28:22 2003 [Thread 10289184] End reading configuration file
Wed Jul 30 22:28:22 2003 [Thread 10298056] LoProtocol : Receiving loop started
Wed Jul 30 22:28:22 2003 [Thread 10298056] LoProtocol : waiting for incoming message
Wed Jul 30 22:28:51 2003 [Thread 10297248] IPProtocol xDemux : received packet in host
byte order
Wed Jul 30 22:28:51 2003 [Thread 10297248] IP Header:
Wed Jul 30 22:28:51 2003 [Thread 10297248]  Ver 4 Hl 5 TOS 0
Wed Jul 30 22:28:51 2003 [Thread 10298056] LoProtocol : waiting for incoming message
Wed Jul 30 22:28:51 2003 [Thread 10297248]  Len 52 Ident 1 offset 0
Wed Jul 30 22:28:51 2003 [Thread 10297248]  TTL 16 Pro 50 Sum 0
Wed Jul 30 22:28:51 2003 [Thread 10297248]  Src 50.2.2.3
Wed Jul 30 22:28:51 2003 [Thread 10297248]  Dst 51.3.3.4
Wed Jul 30 22:28:51 2003 [Thread 10297248]
Wed Jul 30 22:28:51 2003 [Thread 10297248]
Wed Jul 30 22:28:51 2003 [Thread 10297248] ------CRoutingImpl------
Wed Jul 30 22:28:51 2003 [Thread 10297248] ->Received routing message from 50.2.2.3
Wed Jul 30 22:28:51 2003 [Thread 10297248] Cluster Route Request
Wed Jul 30 22:28:51 2003 [Thread 10297248] Type 1 Flags 0
Wed Jul 30 22:28:51 2003 [Thread 10297248] Src Cluster Head 129.1.3.25
Wed Jul 30 22:28:51 2003 [Thread 10297248] Src Node         129.1.3.5
Wed Jul 30 22:28:51 2003 [Thread 10297248] Seq Id           1
Wed Jul 30 22:28:51 2003 [Thread 10297248] Dst Cluster Head 0.0.0.0
Wed Jul 30 22:28:51 2003 [Thread 10297248] Dst Node         51.3.3.4
Wed Jul 30 22:28:51 2003 [Thread 10297248] Traversed Cluster Count 1
Wed Jul 30 22:28:51 2003 [Thread 10297248]   Cluster 47.0.1.5
Wed Jul 30 22:28:51 2003 [Thread 10297248]
Wed Jul 30 22:28:51 2003 [Thread 10297248]
CRoutingImpl : sending route reply message
Wed Jul 30 22:28:51 2003 [Thread 10297248]
Wed Jul 30 22:28:51 2003 [Thread 10297248] ------CRoutingImpl------
Wed Jul 30 22:28:51 2003 [Thread 10297248] ->Sent routing message
Wed Jul 30 22:28:51 2003 [Thread 10297248] Cluster Route Reply
Wed Jul 30 22:28:51 2003 [Thread 10297248] Type 2 Flags 0
Wed Jul 30 22:28:51 2003 [Thread 10297248] Src Cluster Head 129.1.3.25
Wed Jul 30 22:28:51 2003 [Thread 10297248] Src Node         129.1.3.5
Wed Jul 30 22:28:51 2003 [Thread 10297248] Seq Id           1
Wed Jul 30 22:28:51 2003 [Thread 10297248] Dst Cluster Head 50.2.2.3
Wed Jul 30 22:28:51 2003 [Thread 10297248] Dst Node         51.3.3.4
Wed Jul 30 22:28:51 2003 [Thread 10297248] Traversed Cluster Count 1
Wed Jul 30 22:28:51 2003 [Thread 10297248]   Cluster 47.0.1.5
Wed Jul 30 22:28:51 2003 [Thread 10297248]
Wed Jul 30 22:28:51 2003 [Thread 10297248] IPProtocol : xPush(msg) dst (50.2.2.3) route
found.
Wed Jul 30 22:28:51 2003 [Thread 10297248]                    next hop (50.2.2.3).
Wed Jul 30 22:28:51 2003 [Thread 10297248] LoProcotol : send success (LO_50.2.2.3.sock)
Wed Jul 30 22:28:52 2003 [Thread 10297248] IPProtocol xDemux : received packet in host
byte order
Wed Jul 30 22:28:52 2003 [Thread 10297248] IP Header:
Wed Jul 30 22:28:52 2003 [Thread 10298056] LoProtocol : waiting for incoming message
Wed Jul 30 22:28:52 2003 [Thread 10297248]  Ver 4 Hl 5 TOS 0
Wed Jul 30 22:28:52 2003 [Thread 10297248]  Len 50 Ident 1 offset 0
Wed Jul 30 22:28:52 2003 [Thread 10297248]  TTL 12 Pro 17 Sum 0
Wed Jul 30 22:28:52 2003 [Thread 10297248]  Src 129.1.3.5
Wed Jul 30 22:28:52 2003 [Thread 10297248]  Dst 51.3.3.4
Wed Jul 30 22:28:52 2003 [Thread 10297248]
Wed Jul 30 22:28:52 2003 [Thread 10297248] ServerTester xDemux : <Client 10289184 :
sequence 1> from 129.1.3.5 to 51.3.3.4
Wed Jul 30 22:28:52 2003 [Thread 10297248] IPProtocol : xPush(msg) dst (129.1.3.5) route
found.
Wed Jul 30 22:28:52 2003 [Thread 10297248]                    next hop (50.2.2.3).
Wed Jul 30 22:28:52 2003 [Thread 10297248] LoProcotol : send success (LO_50.2.2.3.sock)
```

From above testing output, we can see that the network route from source node E to destination node K is correctly discovered by the cluster routing protocol.