Chapter 1

Rank Forwarding

1.1 Introduction

In this report, we demonstrate the implementation of RANK forwarding algorithm in Haggle framework.

1.2 RANK forwarding strategy

RANK is a delegation forwarding strategy in opportunistic network. Here we assume that each node has metric named RANK, which is used to represent the quality of the node as a relay node. The RANK forwarding strategy always tries to forward the message to the highest RANK. The internal algorithm in each node can be described in Algorithm 1.

```
1 for each_EncounteredNode do
2     if RANKOf(EncounteredNode) > RANKOf(CurrentNode) then
3         SetAsDelegates(EncounteredNode);
4     end
5 end
```

Algorithm 1: RANK forwarding algorithm

1.3 Implementation

Haggle framework has been chosen as the base architecture for our RANK forwarding strategy. Our RANK implementation has been developed specifically for the Linux testbed. The source code has been released via the Google code repository in http://code.google.com/p/haggle-cambridge/.

The key implementation relies in Haggle kernel, which is in charge of managing the forwarding, network, connectivity, contacts, neighbours and more. At the moment, the Haggle kernel only supports Prophet forwarding module[1] as default. We aimed at replacing the Prophet forwarding module with our RANK module. The file structure of Rank implementation is shown in 1.1.

[1]However, there is currently no dynamic module loading in Haggle. ForwarderRank module is made

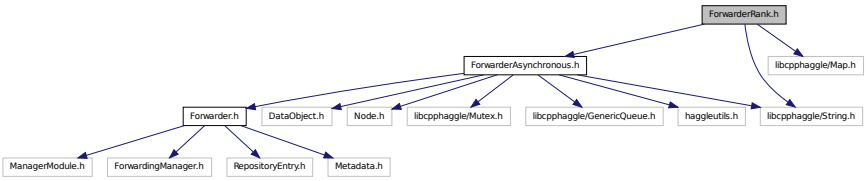

Figure 1.1: File structure for ForwarderRank

In line with the Haggle framework, the ForwarderRank class is inherited from the ForwarderAsynchronous class, and implements some functions as indicated in the Forwarder class (see Figure 1.2).

In RANK the routing table is a mapping between node_id and RANK. Each node will maintain a routing table. As long as the nodes exchange the routing information, It will exchange the RANK with each other. When receiving new routing information, the nodes will add a new node_id and RANK to its new routing table. The implementation of RANK algorithm has been built on delegation forwarding. Finally the nodes which are created as relay node are generated as the delegates.

The following functions are implemented in ForwarderRank class:

- newRoutingInformation() parses metadata containing metrics received from neighbor.

```
1  bool ForwarderRank::newRoutingInformation()
2  {
3
4      if (this is new node)
5          add node_id to id_to_string table;
6
7      while (RoutingInformation) {
8
9          RoutingInformation->getParameter(RANK);
10
11         save the node_id and RANK parameter to routing table;
12
13         go to next RoutingInformation;
14
15     }
16
17     return true;
18 }
```

Listing 1.1: newRoutingInformation()

- addRoutingInformation() adds new routing information that you want to give to neighbor.

```
1  bool ForwarderRank::addRoutingInformation()
2  {
3
```

default by replacing the ForwarderProphet.

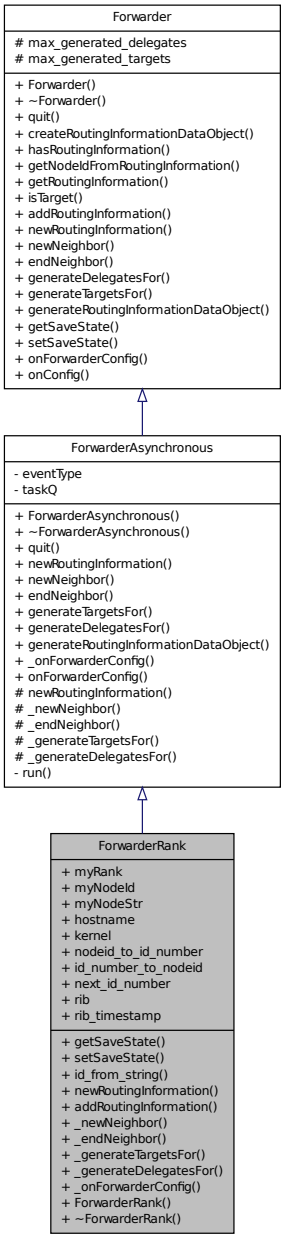

Figure 1.2: Inheritance diagram for ForwarderRank

```
    Add node_id to RoutingInformation;

    Add myRANK to RoutingInformation;

    return true;

}
```

Listing 1.2: addRoutingInformation()

- generateDelegatesFor() generates a list of neighbors that are good forwarders for the target. The other_targets parameter is a list of already found targets. It returns delegates in an event.

```
void ForwarderRank::_generateDelegatesFor()
{

    create sorted_delegate_list;

    get the currentRANK;

    get the delegateRANK;

    for (each node in routing table)
    {
        if (node is not target  && node is not current_node ) {

            if (delegateRANK>currentRANK)
                {
                    insert node into sorted_delegate_list;
                }
            }
        }
    }

    sort(sorted_delegate_list);

    kernel->addEvent(EVENT_TYPE_DELEGATE_NODES);

}
```

Listing 1.3: generateDelegatesFor()

- generateTargetsFor() generates a list of targets that a neighbor is a good delegate for. Return targets in an event.

Chapter 2

ForwarderRank Class

2.1 ForwarderRank Class Reference

2.1.1 Detailed Description

Forwarding module based on RANK algorithms

2.1.2 Member Data Documentation

char ForwarderRank::hostname[HOSTNAME_LEN]

MyHostname is the hostname of current node

Map<bubble_node_id_t, string> ForwarderRank::id_number_to_nodeid

converting table: convert from the NodeId to the NodeStringId

HaggleKernel* ForwarderRank::kernel

HaggleKernel

bubble_node_id_t ForwarderRank::myNodeId

MyNodeId is the NodeId of current node

string ForwarderRank::myNodeStr

MyNodeStringId is the NodeStringId of current node

RANK_T ForwarderRank::myRank

MyRank is the RANK of current node

bubble_node_id_t ForwarderRank::next_id_number

The next id number which is free to be used

Map<string, bubble_node_id_t> ForwarderRank::nodeid_to_id_number

converting table: convert from the NodeStringId to the NodeId

bubble_rib_t ForwarderRank::rib

This is the local forwarding metrics table

Timeval ForwarderRank::rib_timestamp

This is the timestamp.

Public Member Functions

- size_t getSaveState (RepositoryEntryList &rel)
- bool setSaveState (RepositoryEntryRef &e)
- bubble_node_id_t id_from_string (const string &nodeid)
- bool newRoutingInformation (const Metadata *m)
- bool addRoutingInformation (DataObjectRef &dObj, Metadata *parent)
- void _newNeighbor (const NodeRef &neighbor)
- void _endNeighbor (const NodeRef &neighbor)
- void _generateTargetsFor (const NodeRef &neighbor)
- void _generateDelegatesFor (const DataObjectRef &dObj, const NodeRef &target, const NodeRefList *other_targets)
- void _onForwarderConfig (const Metadata &m)
- ForwarderRank (ForwardingManager *m=NULL, const EventType type=-1)
- ~ForwarderRank ()

Public Attributes

- RANK_T myRank
- bubble_node_id_t myNodeId
- string myNodeStr
- char hostname [HOSTNAME_LEN]
- HaggleKernel * kernel
- Map< string, bubble_node_id_t > nodeid_to_id_number
- Map< bubble_node_id_t, string > id_number_to_nodeid
- bubble_node_id_t next_id_number
- bubble_rib_t rib
- Timeval rib_timestamp

2.1.3 Constructor & Destructor Documentation

ForwarderRank::ForwarderRank (ForwardingManager * *m* = *NULL*, const EventType *type* = *-1*)

ForwarderRank::~ForwarderRank ( )

2.1.4 Member Function Documentation

void ForwarderRank::_endNeighbor (const NodeRef & *neighbor*) [virtual]

The _endNeighbor called when a node just ended being a neighbor.

Reimplemented from ForwarderAsynchronous.

void ForwarderRank::_generateDelegatesFor (const DataObjectRef & *dObj,* const NodeRef & *target,* const NodeRefList * *other_targets*) [virtual]

The _generateDelegatesFor function generates an EVENT_TYPE_DELEGATE_NODES event to provide all the nodes that are good delegate forwarders for the given node.

This function is given a target to which to send a data object, and answers the question: To which delegate forwarders can I send the given data object, so that it will reach the given target?

If no nodes are found, no event should be created.

Reimplemented from ForwarderAsynchronous.

void ForwarderRank::_generateTargetsFor (const NodeRef & *neighbor*) [virtual]

The _generateTargetsFor function generates an EVENT_TYPE_TARGET_NODES event to provide all the target nodes that the given node is a good delegate forwarder for.

This function is given a current neighbor, and answers the question: For which nodes is the given node a good delegate forwarder?

If no nodes are found, no event should be created.

Reimplemented from ForwarderAsynchronous.

void ForwarderRank::_newNeighbor (const NodeRef & *neighbor*) [virtual]

The _newNeighbor is called when a neighbor node is discovered.

Reimplemented from ForwarderAsynchronous.

void ForwarderRank::_onForwarderConfig (const Metadata & *m*) [virtual]

The _onForwarderConfig function reads the configuration from config.xml file. The forwarding module will get the RANK configuration from this function.

Reimplemented from ForwarderAsynchronous.

bool ForwarderRank::addRoutingInformation (DataObjectRef & *dObj,* Metadata * *parent*) [virtual]

The addRoutingInformation function is used to generate the Metadata containing routing information which is specific for that forwarding module.

Reimplemented from Forwarder.

size_t ForwarderRank::getSaveState (RepositoryEntryList & *rel*) [virtual]

getSaveState function gets saved forwarding metrics table

Reimplemented from Forwarder.

bubble_node_id_t ForwarderRank::id_from_string (const string & *nodeid*)

id_from_string function get the bubble_node_id_t from the NodeStringId. If the bubble_node_id_t wasn't in the map to begin with, it is inserted, along with a new id number.

bool ForwarderRank::newRoutingInformation (const Metadata * *m*) [virtual]

The newRoutingInformation function is used when a data object has come in that has a "Routing" attribute.

Also called for each such data object that is in the data store on startup.

Since the format of the data in such a data object is unknown to the forwarding manager, it is up to the forwarder to make sure the data is in the correct format.

Also, the given metric data object may have been sent before, due to limitations in the forwarding manager.

Reimplemented from ForwarderAsynchronous.

bool ForwarderRank::setSaveState (RepositoryEntryRef & *e*) [virtual]

getSaveState function saves forwarding metrics table

Reimplemented from Forwarder.

The documentation for this class was generated from the following file:

- ForwarderRank.h

Index