Chapter 1

BUBBLE Forwarding

1.1 Introduction

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

1.2 BUBBLE forwarding strategy

BUBBLE is a delegation forwarding strategy which combines both LABEL and RANK. Here we assume that each node has two metrics, i.e. LABEL and RANK. The BUBBLE forwarding strategy observes the community structure with LABEL and the centrality of nodes with RANK and then decides the best forwarding node. The one simple BUBBLE algorithm can be described in Algorithm 1.

```
for each_EncounteredNode do
    if LabelOf(EncounteredNode) == LabelOf(DestinationNode) AND
    RANKOf(EncounteredNode) > RANKOf(CurrentNode) then
        SetAsDelegates(EncounteredNode);
    end
end
```

Algorithm 1: BUBBLE forwarding algorithm

1.3 Implementation

Our BUBBLE implementation has been developed specifically for the Linux testbed. Haggle framework has been chosen as the framework for our BUBBLE[1] forwarding strategy. 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[2] as default. We

[1]The source code has been released via the Google code repository in http://code.google.com/p/haggle-cambridge/.

[2]However, there is currently no dynamic module loading in Haggle. ForwarderBubble module is made default by replacing the ForwarderProphet.

aimed at replacing the Prophet forwarding module with our BUBBLE module. The file structure of our implementation is shown in 1.1.

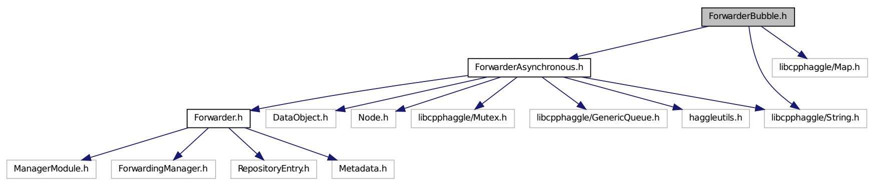

Figure 1.1: File structure for ForwarderBubble

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

In BUBBLE the routing table is a mapping between node_id and the metrics, i.e., LABEL and RANK. Each node will maintain a routing table. As long as the nodes exchange the routing information, It will exchange the metrics with each other. When receiving new routing information, the nodes will add a new node_id and the metrics to its new routing table. The implementation of BUBBLE 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 ForwarderBubble class:

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

```
bool ForwarderBubble::newRoutingInformation()
{

    if (this is new node)
        add node_id to id_to_string table;

    while (RoutingInformation) {

        RoutingInformation->getParameter(RANK);

        RoutingInformation->getParameter(LABEL);

        save the node_id, LABEL and RANK parameters to the metric
             table;

        go to next RoutingInformation;

    }

    return true;
}
```

Listing 1.1: newRoutingInformation()

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

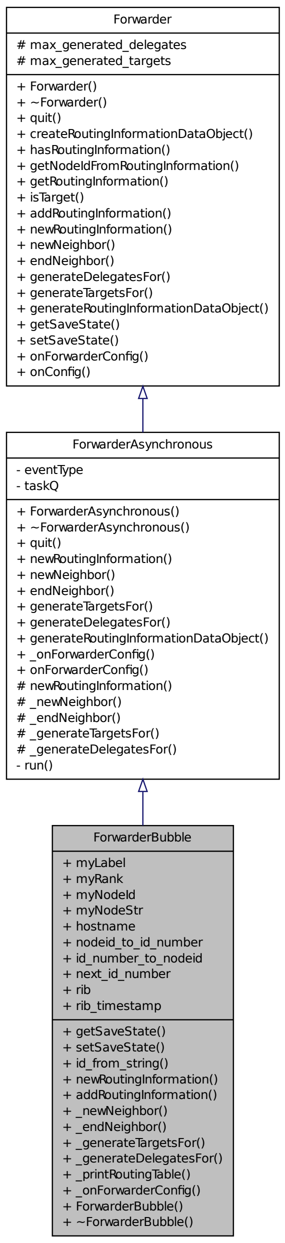

Figure 1.2: Inheritance diagram for ForwarderBubble

```
bool ForwarderBubble::addRoutingInformation()
{

   Add node_id to RoutingInformation;

   Add myRANK to RoutingInformation;

   Add myLABEL 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 ForwarderBubble::_generateDelegatesFor()
{

   create sorted_delegate_list;

   get the currentRANK;

   get the delegateRANK;

   get the neighborLabel;

   get the targetLabel;

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

         if (neighborLabel==targetLabel && 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

ForwarderBubble Class

2.1 ForwarderBubble Class Reference

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 _printRoutingTable (void)
- void _onForwarderConfig (const Metadata &m)
- ForwarderBubble (ForwardingManager ∗m=NULL, const EventType type=-1)
- ∼ForwarderBubble ()

Public Attributes

- LABEL_T myLabel
- RANK_T myRank
- bubble_node_id_t myNodeId
- string myNodeStr

- char hostname [HOSTNAME_LEN]
- 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

2.1.1 Detailed Description

Forwarding module based on LABEL and RANK algorithms
Definition at line 51 of file ForwarderBubble.h.

2.1.2 Constructor & Destructor Documentation

ForwarderBubble::ForwarderBubble (ForwardingManager ∗ *m* = `NULL`, const EventType *type* = `-1`)

ForwarderBubble::∼ForwarderBubble ( )

2.1.3 Member Function Documentation

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

The _endNeighbor called when a node just ended being a neighbor.
Reimplemented from ForwarderAsynchronous.

void ForwarderBubble::_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 ForwarderBubble::_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 ForwarderBubble::_newNeighbor (const NodeRef & *neighbor*) `[virtual]`

The _newNeighbor is called when a neighbor node is discovered.
Reimplemented from ForwarderAsynchronous.

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

The _onForwarderConfig function reads the configuration from config.xml file.
Reimplemented from ForwarderAsynchronous.

void ForwarderBubble::_printRoutingTable (void)

The _printRoutingTable function prints the routing table in debug mode.

bool ForwarderBubble::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 ForwarderBubble::getSaveState (RepositoryEntryList & *rel*) `[virtual]`

getSaveState function gets saved forwarding metrics table.
Reimplemented from Forwarder.

bubble_node_id_t ForwarderBubble::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 ForwarderBubble::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 ForwarderBubble::setSaveState (RepositoryEntryRef & *e*) `[virtual]`

getSaveState function saves forwarding metrics table.
Reimplemented from Forwarder.

2.1.4 Member Data Documentation

char ForwarderBubble::hostname[HOSTNAME_LEN]

MyHostname is the hostname for this node.

Map<bubble_node_id_t, string> ForwarderBubble::id_number_to_nodeid

id_number_to_nodeid is used to convert table from the NodeId to the NodeStringId.

LABEL_T ForwarderBubble::myLabel

MyLabel is the LABEL for this node.

bubble_node_id_t ForwarderBubble::myNodeId

MyNodeId is the bubble_node_id_t for this node.

string ForwarderBubble::myNodeStr

MyNodeStringId is the string_id for this node.

RANK_T ForwarderBubble::myRank

MyRank is the RANK for this node.

bubble_node_id_t ForwarderBubble::next_id_number

next_id_number is free to be used.

Map<string, bubble_node_id_t> ForwarderBubble::nodeid_to_id_number

HaggleKernel is the kernel handler. / HaggleKernel ∗kernel;

/∗∗ nodeid_to_id_number is used to convert table from the NodeStringId to the NodeId.

bubble_rib_t ForwarderBubble::rib

This is the local forwarding metrics table.

Timeval ForwarderBubble::rib_timestamp

The timestamp for local forwarding metrics.

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

- ForwarderBubble.h

Index