END-TO-END COMMUNICATION

Orestis Akribopoulos
Research Academic Computer Technology Institute
Patras, Greece
akribopo@cti.gr

Lorenzo Bergamini
Sapienza University of Rome, Department of Computer and Systems Sciences
Rome, Italy
bergamini@dis.uniroma1.it

Andreas Cord-Landwehr
Heinz-Nixdorf-Institute & Department of Computer Science,
University of Paderborn
Paderborn, Germany
cola@uni-paderborn.de

Christos Koninis
Research Academic Computer Technology Institute
Patras, Greece
koninis@cti.gr

Antoni Segura Puimedon
Universitat Politecnica de Catalunya, BarcelonaTech
Barcelona, Spain
antoni@celebdor.com

1. Introduction

One of the most important issues that must be addressed in a WSN in the ability for any pair of nodes inside the network to communicate with each other. If the nodes are within range of each other then routing is not necessary but in most cases intermediate nodes are needed to organize the network and act as routers carrying the data transmissions. The topology of a WSN varies over time and transmissions are subject to errors, so our the techniques employed by the routing algorithm must track the changes in the network topology and re-discover new routes when failures break the old ones. In this chapter we present a routing algorithm that takes advantage the self-organizing and adaptive components analyzed in the previous chapter in order to provide a resilient and reliable way to route messages across the network. The resulting algorithm has two main design goals, first to be able to scale to large societies and second to provides delay-tolerant "connectivity", i.e., guaranteed communication between any pair of system nodes.

2. Software Architecture

The End-to-end communication module (E2E) is a software component intended to hide the details of protocol hierarchies in the network. This abstraction allows the final user to call a unique *send (destination)* command when a message transmission is requested, without any knowledge requirements of the current location of the

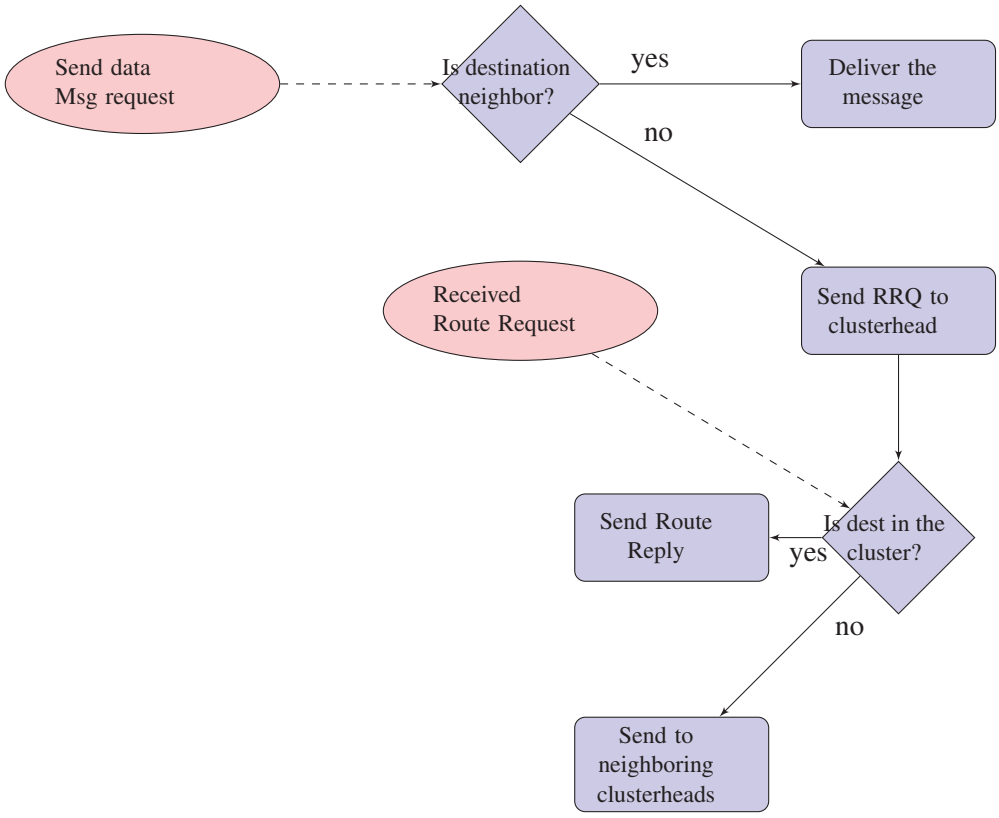


Figure 1: Flow Chart of E2E module operations

destination. E2E module can be seen as an implementation of a routing algorithm, where the message is sent to a destination, by choosing the best strategy according to the information provided by neighbor-discovery, clustering, and highways modules. It also takes advantage of the mobile elements of the network (e.g., roomba devices) to offer delay-tolerant and reliable connectivity), when message deliveries, towards nodes that are not a part of the current network are requested. To have a clear view of the operations performed by nodes in the E2E module, they are represented in flow-diagram 1.

When a new message is send through the E2E, the module first queries the ND module to examine whether the destination is an immediate neighbor, in which case the message is delivered directly. If the node is not an neighboring node the E2E will examine if a route to the destination exists in the route cache. In the case that a route to the destination is not known it will place the message in a buffer and send a route request message to the cluster-head. The cluster-head upon receiving a route request message it will examine if the destination is a member of the cluster. In the case that the destination is not in the cluster the cluster-head will forward the request to the heads of neighboring clusters and the process will continue for the entire network. Each node that forwards the request message adds its id to the message. If a cluster-head is found that it is the destination or the destination is a member of its cluster then a route reply message is send back to the originator(the node that send the original route request message) following the reverse path. When a route reply is received by the originator the route cache is updated with the new information, and the data message buffer is examined for pending messages to that destination. Each route in the cache and each pending data message have time-stamps, and separate timeout events are used to frequently clean stale entries.

In the current implementation of the E2E offers two strategies for facilitating the communication between the cluster-heads, the first is using the CLR module and the second the Highways module. Both offer the same functionality and can be used by the E2E in a transparent way.

The E2E module's can provide data message exchange between unconnected networks. This is done by interacting with the mobile elements of the society that are controlled by the Delay Tolerant System module. If a data message is in the buffer for more than 2 sec, is a consider a candidate for the DTS and will be delivered to the

next mobile element that will be in proximity. The movement is completely independent of the message delivery functionality. Each mobile element (i.e., roomba robots) stores received messages at its private round-buffer and on discovery of a new network node, it sends all saved messages to that node using the other components of the E2E module.

2.1 Experimental Results

In this section we present and discuss the experiments we made to evaluate the integration of the E2E module with Neighbor Discovery and Clustering modules. The metrics we want to measure are:

- *Packet Delivery Ratio*, a value measured at the sink node, indicating the percentage of correctly received packets over the number of totally sent packets
- *Latency*, indicating how much time in average a packet takes to reach its final destination (i.e. the sink).

The experiments were performed using 46 nodes in total installed in an office environment. In addition to the experiments of the E2E module, we also tested DSDV routing protocol [PB94], in order to compare the results obtained with the two different solutions. We decided to use DSDV protocol for our comparison because of several reasons: first of all, it is a proactive protocol. Even if in our implementation we do not store routing tables, we have a concept of hierarchy created by ND and CL modules, which is continuously updated and is used to send messages; DSDV is thus closer to our solution than any reactive protocol. Then, it is an extensively-tested solution, which has been shown to work fine on different settings and devices, and thus we can be reasonably sure that it provides a good benchmark. Last but not least, DSDV has been available on Wiselib for a long time, it has been tested a lot of times, and the current implementation works fine; with high probability the results we obtain are not biased by errors in the implementation. We also include a graph to count how many messages are send using HGW module, in figure 2.

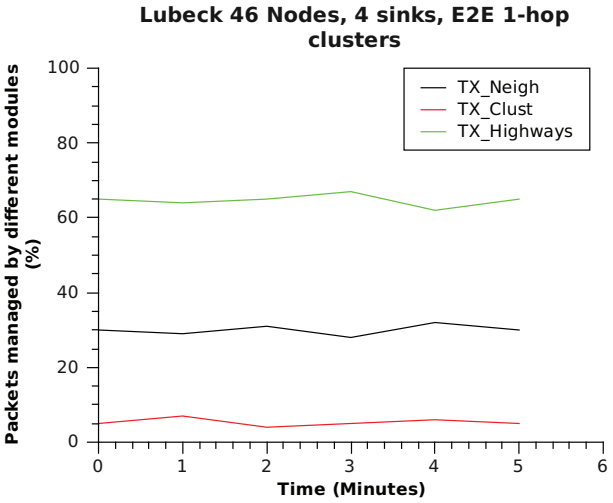
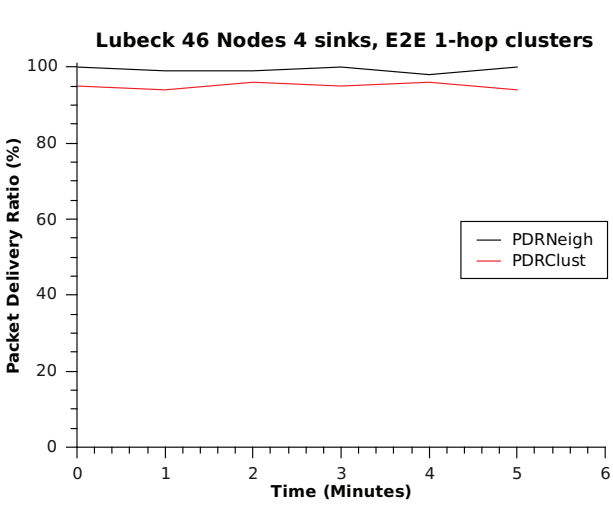


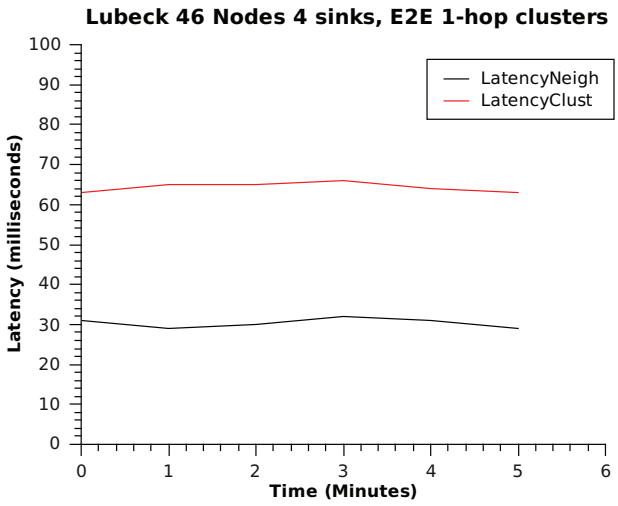
Figure 2: E2E Module usage for 46 nodes experiment

The first thing we can immediately notice from the experiments, is the amount of messages given to the HGW module is far lower than what we expected. This is due to the fact that the higher node density has the effect of more nodes being direct neighbors, and thus the messages are immediately delivered (this is confirmed by the fact that the number of messages directly delivered to the sink is around $\approx 30\%$). As for the CL module, we observe that the resulting clusters are composed of a few nodes, and in fact the messages sent to the cluster leader and then to the final destination are only $\approx 5 - 7\%$. Now we report the figures of PDR and latency.

The graph relative to PDR shows almost all the packets sent from direct neighbors are correctly delivered to the final destination, and the same is for packets sent through cluster leaders (where the PDR is around $\approx 95 - 98\%$. The recorded latency is significantly lower for both neighbor and cluster delivery; our analysis shows that this is due to the fact that the nodes are deployed over a larger area, thus reducing interferences between concurrent



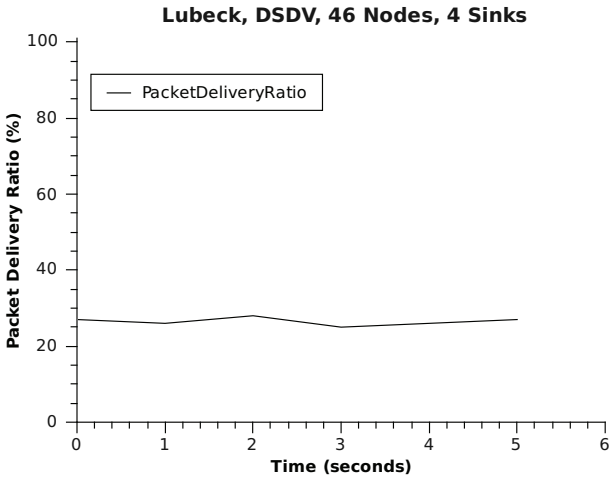
(a) Packet Delivery Ratio



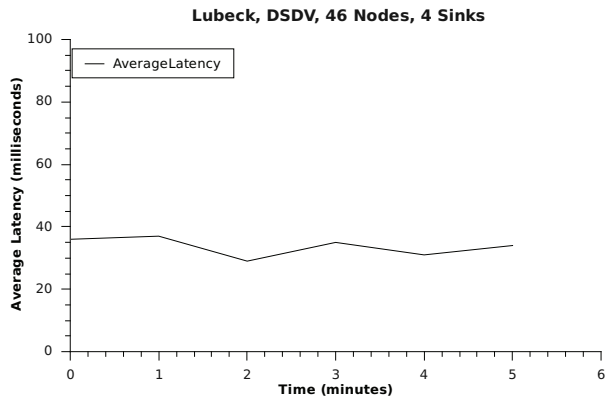
(b) Average Latency

Figure 3: E2E metrics for 46 nodes experiment

communications. As for DSDV algorithm, its results are reported in figures 4(a) and 4(b). We can immediately notice that the size of the network has a negative impact on the performance of DSDV algorithm, as the average PDR is lower than the one recorded in the other testbeds ($\approx 30\%$ versus $\approx 45 - 50\%$). The latency, on the other hand, remains more or less the same.



(a) Packet Delivery Ratio



(b) Average Latency

Figure 4: Metrics for 46 nodes DSDV experiment

3. Highways Algorithm

The Highway algorithm consists in devising a series of mechanisms to allow higher level treatment of clusters as single entities, i.e., virtual nodes. This change in the clusters treatment means that the cluster leader becomes the logical leading entity of the virtual node, and the leader, but mainly the rest of the cluster nodes, become the communication ports of the virtual node with their neighboring virtual nodes. The list of nodes that consti-

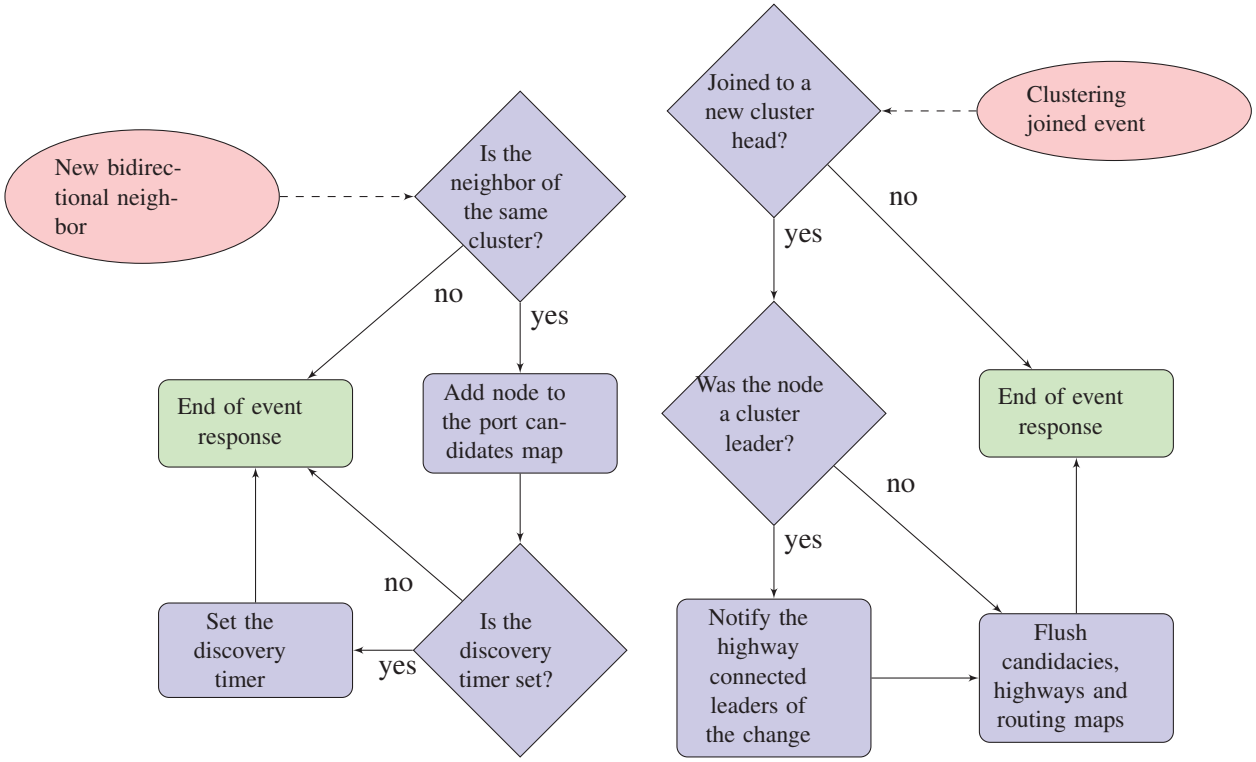
tute communication paths between leaders and are chosen by both leaders to perform this task is what we call highways.

The hierarchy construction is performed in collaboration with the neighborhood discovery and the clustering algorithms. Both algorithms play the crucial job to generate the events to which the Highway algorithm reacts, as the nature of the algorithm presented here is purely event driven. However, whereas the neighborhood discovery generates the connectivity events, altering the likelihood of a node becoming a port; the clustering generates what we could call resetting events, as they are basically used to detect when a node switches clusters or it simply becomes a mono-node cluster.

The nodes in a cluster that are not elected as leaders by the clustering algorithm are, from the perspective of the Highway algorithm, mainly deemed as potentially communicating nodes that notify about other clusters and, if chosen to form part of a highway, pass the highway level messages around. Leaders, on top of the communicating capabilities, are entitled to initiate negotiations with other leaders to establish highways between them, disable and/or change them.

3.1 Implementation details

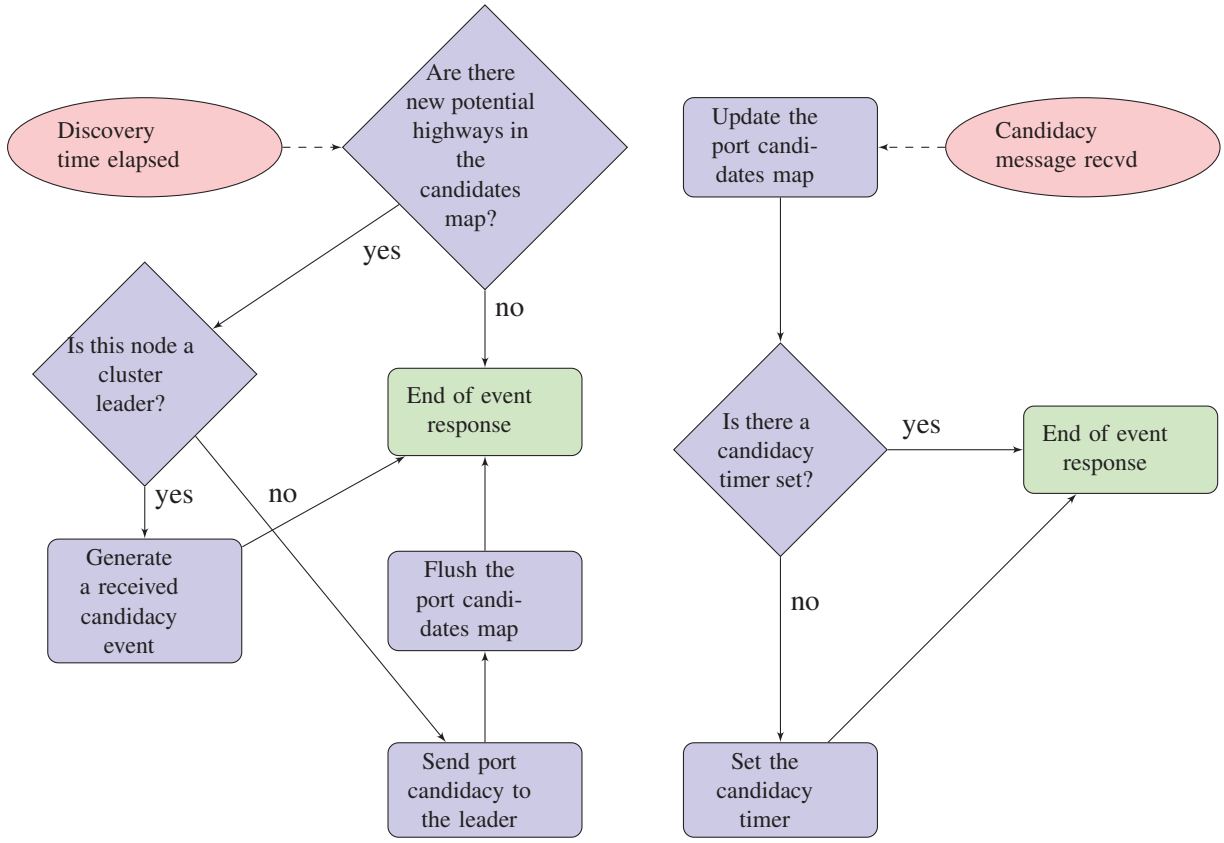
In this subsection, the reader will be presented with a set of diagrams and explanations of the algorithm. Due to the event driven nature of the Highway algorithm, the kind of diagrams chosen are flowchart, which allow for a great case per case visualization, reducing the complexity that a comparable traditional monolithic algorithmic view would show in treating the events. The first diagrams, 5(a) and 5(b) account for the interfaces with the underlying algorithms, the neighborhood discovery and the clustering, respectively. This external events, are successfully received by the Highway Algorithm thanks to the Wiselib's callback registering capabilities implemented by both of those modules. The registering takes place in the enabling routine of the algorithm that, due to it's triviality, will not be diagrammed.



(a) Highway algorithm response to a neighborhood discovery new bidirectional neighbor event

(b) Highway algorithm response to a clustering joined event

Figure 5: Algorithm diagram of the response to the clustering and neighborhood discovery related events.



(a) Highway algorithm response to a discovery timer event

(b) Highway algorithm response to a candidacy request

Figure 6: Highway algorithm response to discovery and candidacy message events.

As a result of the previous flowcharts, granted that there are multiple nodes and clusters, the 5(a) decision path will fill the port candidates map and set up a timer to allow for a discovery time window which, on its end, might result in several candidacy requests, as seen in the diagram 6(a), depending on the amount of changes that happened to the network since the last discovery time window. The candidacy requests, as depicted in the same diagram, will then be sent to the leader or, if the current node is a cluster leader, it will be processed as if the request was sent to itself. Faking the message reception on the leader avoids some extra handling that, otherwise, would be needed to deal with this particularity. The handling of the candidacy messages consists in merely adding them to the candidacies map, and in an homologous way of the 5(a) diagram, sets a message receiving window, as displayed in the 6(b) figure.

Once the candidacies timer elapses its alloted time, it generates an event that is processed in the way depicted in figure 7(a). As the reader can see, it basically checks if any of the highway candidates present real opportunities to create working highways and decides to act accordingly, by sending a port requested in cases of map entries with potential. After that, the map is flushed for the next candidacies window to have a clean slate. The port requests are sent to the matching target leader through the highway to be path of nodes and then the algorithm moves on, without blocking nor waiting for any kind of response. The lack of waiting for any kind of response simplifies the solution, allows the devices to operate without any hindrance and, in any case, doesn't bar from treating a response if it is received.

The responses to the port requests can be acknowledgments, non acknowledgments and no response at all if, due to malfunctions of medium phenomena, the message is lost. The decision process that determines which kind of message will be returned can be observed in the figure 7(b). The Highway algorithm basically accepts the request unless there is a working highway different to the proposed one. To determine if it is working or not, the

algorithm gives and takes points from the highways on the highway map based on successful transmissions. The point give an take is done in an asymmetrical way, taking 3 points per sent message and awarding 4 per received message, this way, long term reliable highways present always a better balance than the clean balance (0).

When a node receives an acknowledgment (figure 7(c)), it automatically places the acknowledged highway into its working highways map. In contrast, when a non-acknowledgment is received (figure 7(d)), it is removed from the highways map and the process of the figure 7(a) is invoked without the usual candidacy time window. This is done to try to renegotiate ports on failure, and also because non-acknowledgments are additionally used to signal the end of life of a working highway, i.e., if a leader ceases to function as such, like in the diagram 5(b), it tries to notify the receiving ends of its now void highways of the invalidity of the paths.

3.2 Experimental results

This subsection presents a small subset of the results of the real hardware experimentation of the Highway algorithm. These results, just like any real life experimentation, are determined in great measure by the physical conditions of the test environment, in our case, the topology and connectivity. The influence of these conditions not only affects the algorithm in a direct way, but it also impacts indirectly the performance through the variations it can cause to the event generating algorithms, neighborhood discovery and clustering. In regards to the physical phenomena, the algorithm was tested in different conditions, of which two can be seen in figures 8(a) and 8(b), which correspond to the testbeds of the Research Academic Computer Technology Institute (CTI) and the University of Geneva (UNIGE) respectively.

As seen in the picture of figure 8(a), the network is unconnected and it matches well the topology of the testbed, as the two sub graphs sit in different wings of the building and the connectivity between these two parts is variable and of lesser quality. In the Geneva testbed (figure 8(b)), due to the proximity of the nodes, the degree of connectivity is high and thus, the clustering, neighborhood discovery and highway algorithms cooperate to a successful fully connected graph. It is important to note, that these figures are just snapshots of an instant, as the topology is naturally varying during the experiment due to noise, moving obstacles and other changing conditions.

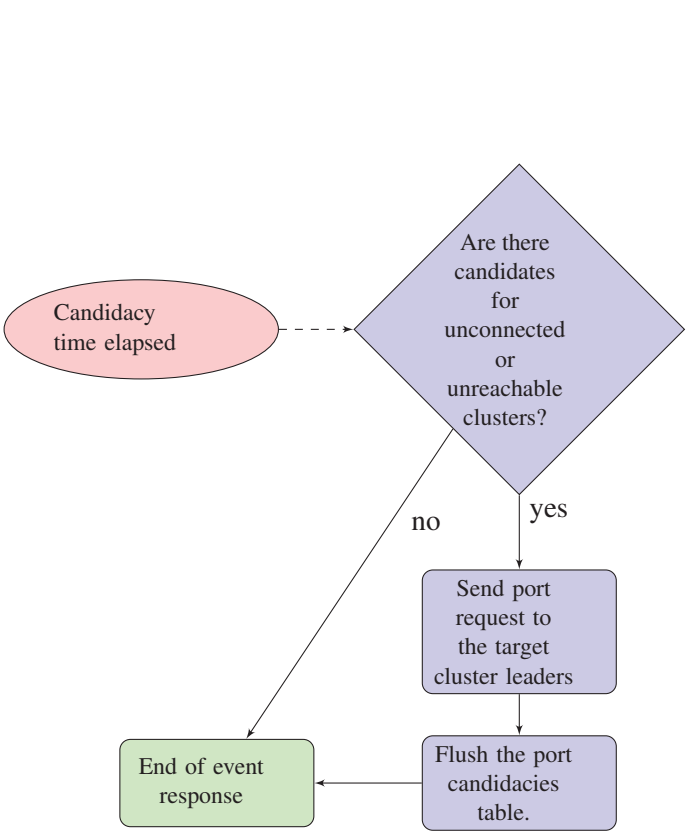
The other experiment, with which this subsection concludes, seen in figures 9(a) and 9(b), is about the amount of message passing that is generated by the Highway algorithm operation only. This measurement can be a good indicator of features such as energy consumption, medium usage, overhead and delivery rate. In this experiment, we generated highway traffic after waiting $40sec$ initially, to allow the underlying modules to stabilize, and then every cluster leader sent a message to each of the reported highways attached to it. After a continuous cycle of a random $10 - 30s$ back off and data transmission.

In figure 9(a) the network was stable enough so that the amount of SEND and ACK messages were well above the highway construction and maintenance messages. This trend could be easily reverted by introducing more instability to the medium. After these two kinds of messages, which just propagate around the generated traffic, there is the CAND and REQ messages. The graph shows how each step of the algorithm (in messages) filters the amount of occurrences, in concordance with the diagrams of the previous subsection, although party due to message loss.

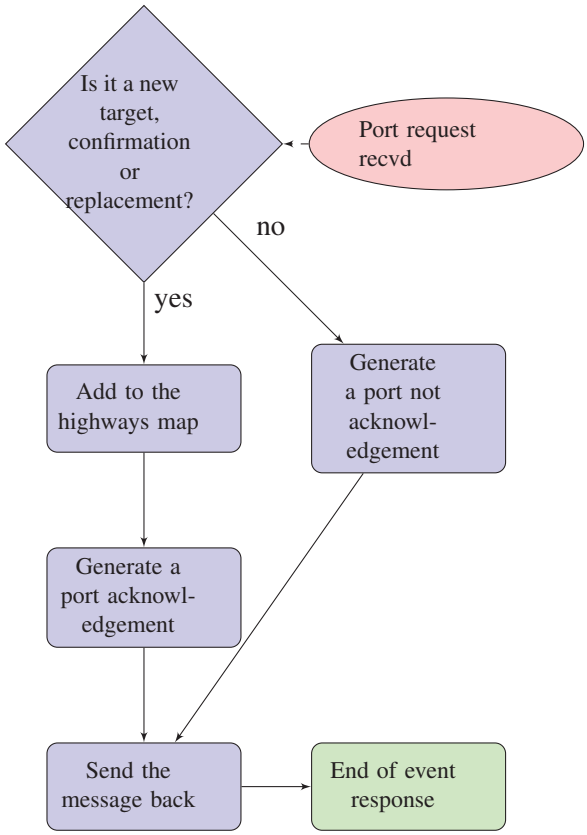
In Figure 9(b), only the sent and received messages at the endpoints are displayed. From the graph one can see that the initial highway operation attains a high delivery rate, but as the clustering shifts (and thus the formed highways have to either be dropped at one or both ends, or reach the maximum amount of allowed unreturned ACK messages to be dropped), a breach appears between the sent and the received. The gap stabilizes along the experiment just shy of 70%. This delivery rate has been found to be more influenced by the clustering changes than by the direct highway connectivity. Even though, what obviously makes the clustering algorithm change at an accelerated pace is, in fact, a reduction on the connectivity. In several runs of the experiment the delivery rate was found to be typically in the range of $50 - 70\%$. These data are obtained in the same experiment as that in the previously commented figure, and the data was gathered by parsing of a sizable number of debug messages.

4. Delay Tolerant System – Robots

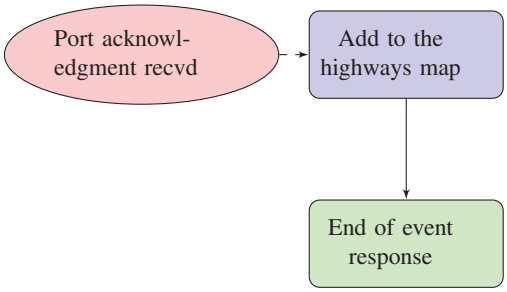
The connectivity of networks is an important point in end-to-end communication to ensure successful transmissions of messages to their targets. In FRONTS we could weaken the assumption to have only networks that



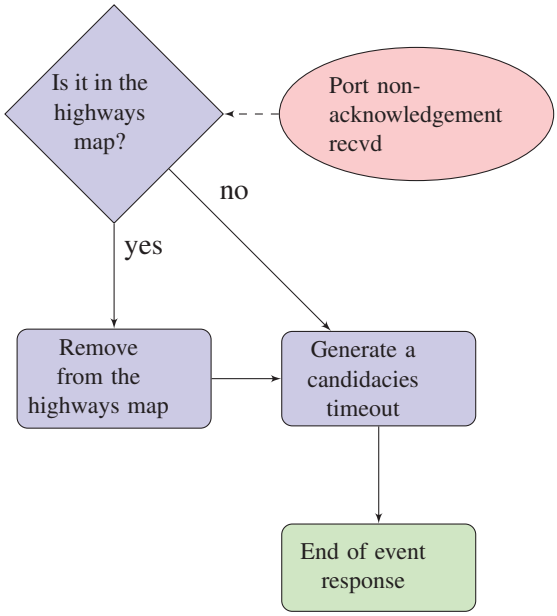
(a) Highway algorithm response to a candidacy timer event on a leader



(b) Highway algorithm response to a port request

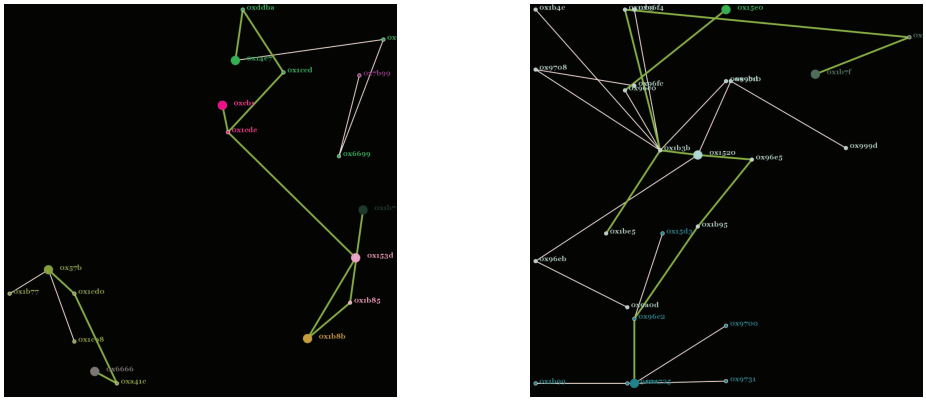


(c) Highway algorithm response to a port acknowl-edgement



(d) Highway algorithm response to a port non-acknowledgement

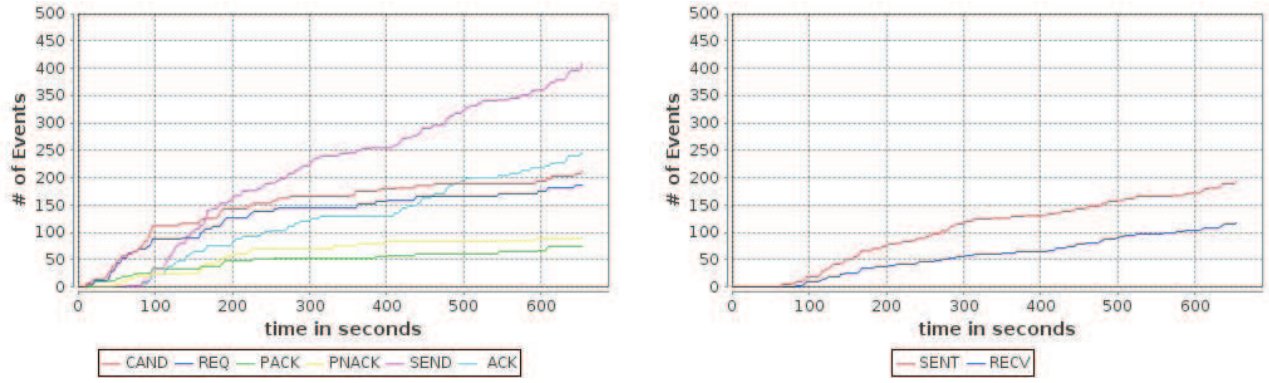
Figure 7: Highway algorithm port negotiation.



(a) Divided network (CTI)

(b) Connected network (UNIGE)

Figure 8: Snapshots of the output of the algorithm for highways' formation on different testbeds. Each color identifies a cluster, where the bigger radius node is the leader. Highways are emphasized in green.



(a) Total number of messages to build highways by type

(b) Aggregated number of messages sent and received through highways

Figure 9: Aggregated messages per each type sent over the network

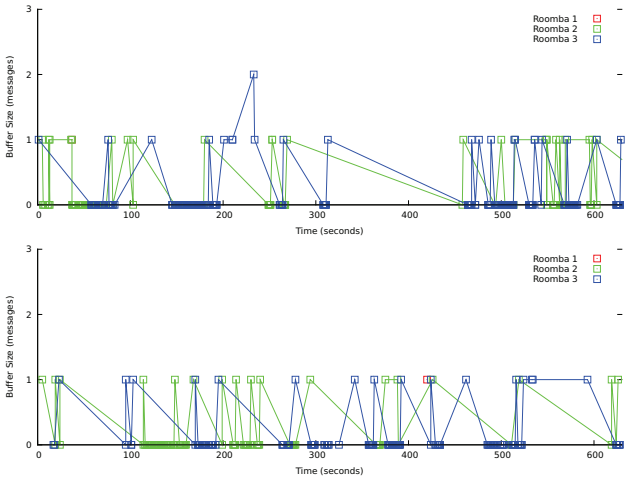
are not directly but use robots to carry unroutable messages between them. The core module for this delay tolerant connection is given by the E2E module: This is done by interacting with the mobile elements of the society that are controlled by the robot motion module. The roombas and the moways that execute the robot motion module collect the messages that cannot be delivered (since the destination is not located within the same connected component of the network), carry the messages to the disconnected network and deliver the messages to their final destination.

4.1 Roombas

For the roomba modules we decided to make a clear distinction between the E2E module and the robot motion module. The reasoning is based on the sensor modules that were used for our experiments. For instance, with the current sensors it was not possible to tell the robot to move to the best signal-strength or even record paths and re-walk them as the movement accuracy was to low. With this in mind the functionality of moving and delay-tolerant sending and receiving is accurately devided into the two modules.

On the motion side, we have the robot motion module that executes one of the motion patterns *straight line* or *random walk* (cf. Robot Motion Chapter). On the communication side, we have the E2E module that decides

(a) Experiment Setting with Three Roombas



(b) Transmitted Messages with Three Roombas

Figure 10: The figures present results for three roombas at random walk, whereas (a) visualizes the experiment setting and (b) gives the used buffer sizes and transmitted messages for each roomba.

whether the target of a message is routable (then it will be forwarded in the network) or unroutable (then it will be transmitted to a roomba device if such a roomba accidentally meets this network node). In the case of transmitting the message to a roomba device, the roomba collects this message and writes it to the first empty position inside its buffer. If the buffer is full, the roomba overwrites the oldest stored message it has stored.

Internally the roomba stores messages in a round-buffer of fixed size. If a message is successfully transmitted by roomba device (using the E2E module) to a subnetwork, the specific entry in the round-buffer is invalidated. On discovery of a new network the roomba successively sends all stored messages to the discovered node. Analogue, a network node that discovers a roomba (by the network discovery module) sends all unroutable messages to the roomba in range. To improve transmission rates, the roombas also merge their round buffers if they accidentally meet. This is, each roomba dumps all its valid stored messages and sends them to the other roomba, which then handles them like messages received from network nodes.

The experiments provide strong evidence that our implementation reliably supports a delay tolerant communication between unconnected networks, if we assume that the roomba device, which is executing the robot motion module, detects the unconnected networks at its walks. A working communication with several roombas at the system, with multiple messages carried by a single roomba, and with inter-roomba communication could be tested successfully. However, much improvement is possible in connecting the robot motion module directly to the delay tolerant end-to-end communication. This can be done, for instance, by using more accurate hardware for the roomba device such that recorded walks could actually be re-walked. Or by adding further sensor capabilities like globally unique positions or estimates of directions for maximal signal strenght. By this, regular checks for new messages at already known networks would become possible.

The above described E2E communication functionality of the roomba devices is encapsulated in the `RoombaMessageDelivery` class. This functionality currently can only be used with the PC `OsModel`; however, code changes to allow execution at more low-cost hardware are straight-forward.

4.2 Sunspot/Moway

Moways are the second hardware platform that provide delay-tolerant and reliable connectivity between unconnected networks. A SunSPOT has been mounted on top of each robot, as described on Section 8.4. The neighbor discovery module, the delay tolerant service (DTS) and the robot motion are implemented on the SunSPOT. The module which is responsible for the motion of the Moway is described in Section 8.4 and provides two different motion modes: the *Remote Control* and the *Random Walk*.

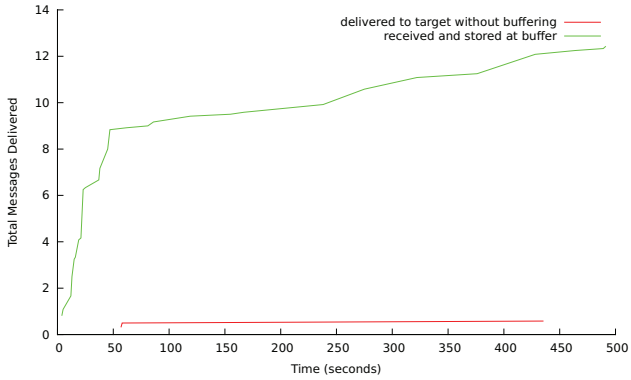


Figure 11: Average total transmitted messages for an experiment with one roomba moving on a straight-line movement and connecting two disconnected that produce messages every 5 sec. aimed to the other network.

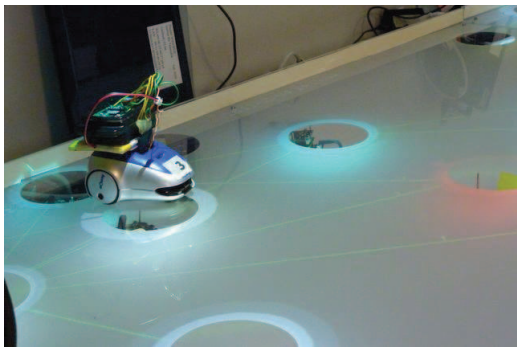


Figure 12: Moway delivering delay-tolerant messages (see Section 10.2

E2E module decides whether the target of a message is routable or unroutable. In the case of an unroutable target, the messages will be transmitted to a Moway robot. The Moway, which is executing a Random Walk, collects the messages and stores them on a internal buffer.

In particular, when DTS receives a message from a network node, checks the target of the specific message and stores it to a bounded queue. DTS maintains different thread-safe queues per destination. All of the queues have fixed capacity and follow the FIFO principle. If the queue is full, the oldest stored message is removed. As soon as the neighbor discorvery protocol discovers a bidirectional neighbor, the delay tolerant service is informed. If there are undelivered messages for the specific neighbor, DTS tries to send all stored messages. DTS follows the `Observable/Observer` design pattern, observing the neighbor discovery module to avoid polling continuously for changes regarding neighbor's connectivity status. During the delivery of the stored messages to a specific target, DTS notifies robot motion module and stops Moway's motion. After the completion of the message transmission, DTS re-notifies the robot motion module in order to start the Moway's motion. All of the transmitted messages, are stored in a fixed capacity list.

Moreover, when a Moway discovers another Moway, they merge their queues with the undelivered messages. In order to ensure that each undelivered stored message will be delivered only one time, Moways also merge their lists with the transmitted messages. In particular, neighbor discovery module notifies the DTS module, when another Moway is bidirectional neighbor. DTS tries to send both undelivered and delivered messages, which are stored in the buffers, and the other Moway responds with the intersection of the stored and received messages. As soon as the merging procedure is completed, Moways will have in their queues the same messges.

5. Interfaces

5.1 E2E Interface

The E2E Module follows the general Wiselib interface. It also implements the Routing Concept and provides a send method and a reg_recv_callback that can be used for register callback functions in order to get notifications upon reception of data messages.

```
template<...> class E2E {
 int enable(void);
 int disable(void);

 init(RadioT, ClockT, TimerT, DebugT, ClusterT, CLusterRadioT);
 init(RadioT, ClockT, TimerT, DebugT, ClusterT, HighwaysT);

 send(node_id_t dest, size_t len, block_data_t *data );

 int reg_recv_callback(T *obj_pnt);
 int unreg_recv_callback(int idx);
}
```

Interfaces with other components:

Neighborhood Discovery From ND module, E2E obtains the list of current neighbors of the selected node; this information is used to check if the destination specified in the send method is currently a neighbor, and then if the message can be immediately delivered.

Clustering From CL module, E2E obtains information on the leader of the cluster it belongs to, and (only for the cluster leader), information of which nodes are currently part of the cluster.

Highways/Cluster Radio E2E obtains a special send method which is used by cluster leaders only for inter-cluster communication (i.e. for sending messages from one cluster to the other).

Group-key Establishment E2E module provides GKE the instance of the routing needed by the module to establish the private key between the requested set of neighbors (for details on GKE module, we refer to its description).

5.2 Delay Tolerant System Interface

Delay Tolerant System follows the Radio Concept of Wiselib. Thus implements the `provides enable_radio()` and `disable_radio()` functions to switch the radio on and off. In order to send a message there is a `send` function that is implemented exactly as the send function of Radio. Finally it provides a receive callback, to be able to receive radio messages from the DTS.

```
template<...> class DTS {
 int enable_radio(void);
 int disable_radio(void);

 void send(node_id_t receiver, size_t len, block_data_t *data);
 node_id_t id();

 int reg_recv_callback(T *obj_pnt);
 int unreg_recv_callback(int idx);
}
```

Interfaces with other components:

Neighborhood Discovery From ND module, DTS obtains the list of current neighbors. This information is used to check if the destination specified in the send method is currently a neighbor, and then if the message can be immediately delivered. ND module also provides information about the presence or not of a Roomba robot (i.e. a mobile element) in the current neighborhood.

Robot Motion Planning E2E module uses the robot motion planning module to control the mobile elements so that messages are transfered to disconnected parts of the society.