3 WPAN - Part II

3.1 Setup

You need the following equipment:

- Different Zolertia® RE-MoteTM modules (2,4GHz).

And will use the following tools:

- A computer with VMWare Player and the Instant Contiki 3.0 virtual machine image.
- Cooja.

3.2 Objectives

The main objective of this lab is to implement a mechanism that is necessary to autoconfigure a network: the assigment of a unique addres for each node. Furthermore, during the implementation, students will gain knowledge about how a WPAN protocol is implemented in Contiki OS.

3.3 Introduction

3.3.1 Self-configuration

Sensor Wireless Networks are usually formed by a large number of networked sensing nodes. It is rather complex, or even unfeasible to configure *manually* each one of the devices, so it is necessary to implement mechanisms that are able to set up the network without human intervention. The major challenges are automatic address configuration, scalable routing and security [1].

The autoconfiguration protocols are classified into three categories: stateful approaches, stateless approaches and hybrid approaches [2].

- A stateful approach assumes the existence of a central entity to keep an address allocation table for whole network and assigns unique IP address for unconfigured node. i.e. the nodes know the network state.
- Stateless approaches do not need a central entity to maintain an address allocation table. Instead, each node selects an address by itself randomly and verifies its uniqueness with the so-called duplicate address detection (DAD) procedure. If duplication is detected, at least one of the nodes with duplicate addresses must change its address.
- Hybrid approaches combine both stateful and stateless mechanisms to improve the scalability and reliability of the autoconfiguration, but may result in higher complexity and higher protocol overhead.

It would be possible to use a dynamic host configuration protocol (DHCP) to acquire an IP address from a centralized server (like in wired networks). Nevertheless, when mobile nodes are involved, it is complex to maintain a centralized DHCP server. Many dynamic address configuration protocols have been proposed. However, most of the protocols rely on passive Duplicate Address Detection (DAD) mechanism to resolve the address conflict.

3.3.2 QDAD

One of the first stateless autoconfiguration protocol for Mobile Ad-hoc Networks (MANETs) was *Query-based DAD* (QDAD) [3]. In this proposal, when a node wants to obtain an address, as it may be difficult or impossible to contact any address allocation agent in the network, it should select a random address. This procedure does not guarantee uniqueness, so it is necessary to perform a Duplicate Address Detection (DAD) procedure[1].

The main idea behind this protocol is to ask the other nodes (using *broadcast* frames) if they are already using the same address. The algorithm is as follows:

- During the initialization of a node, it uses a random procedure to select a tentative address.
- Before making the selected address permanent, the node must be sure that address is unique within the network. It broadcasts the tentative address using an `ADDRESS_REQUEST (AREQ)` package. This package reaches all the neighbors, and the neighbors resend the package to the nodes in range until the `AREQ` package is received by all the nodes of the network.
- If any other node in the network observes that the tentative address matches its own address, it will respond with an `ADDRES_REPLY (AREP)` packet. This means that the tentative address is already in use and therefore the node that sent the `AREQ` package must choose a new tentative address. The `AREP` packet (unicast) follows the inverse route that followed the `AREQ` packet, so it is necessary for the intermediate nodes to store the address of the node from which the AREQ packet has been received.
- If the node that sent the `AREQ` packet does not receive a reply from any node after a certain time it will expire a timer associated with that packet. When the timer expires the packet is forwarded again another two more times. If after three times the timer expires and the node does not receive an `AREP` packet, it will assume that its address is unique and therefore will make it permanent.

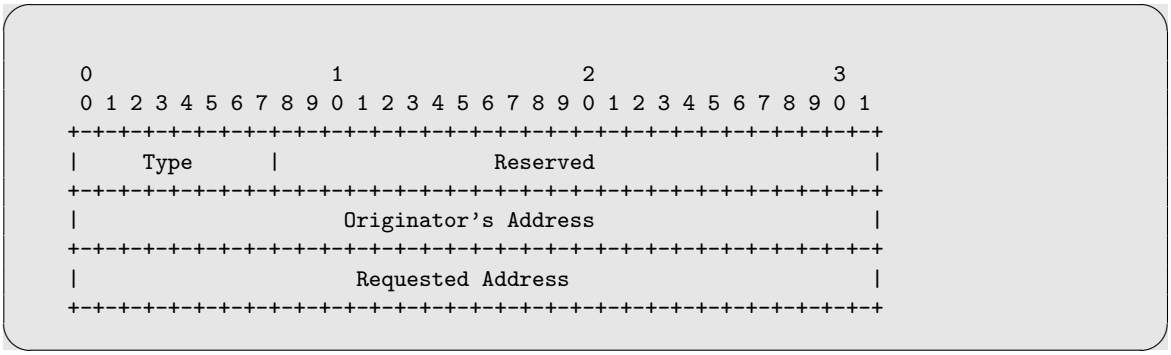
Each node can be in one of the three following states:

- `NO_ADDRESS_STATE`: In this state the node does not have a network address assigned, therefore, it must not process or send route control messages generated by other nodes, nor participate in sending data. A node changes from the `NO_ADDRESS_STATE` state to the `ADVERTISING_STATE` state when a timer (t_{START}) expires.

[1] This procedure should be repeated when a new node joins the network, or when a part of the network that was disconnected *heals*

- ADVERTISING_STATE: When a node generates its tentative address, it changes to the ADVERTISING_STATE state. The node remains in this state while the process to check if the address is unique in the network is running. If the address is considered unique, the node changes to the NORMAL_STATE state. In case there is another node with the same address, it returns to the state NO_ADDRESS_STATE. When a node is in this state, it should not send route control messages generated by other nodes and or participate in sending data.
- NORMAL_STATE: When the timer of the third AREQ packet expires (there are no AREP packets answering to the AREQ packet), the node changes to NORMAL_STATE state and considers that the tentative address is valid. Now the node can send and receive data normally. Nevertheless, if a node detects an address conflict with other nodes in the network, it must return to the NO_ADDRESS_STATE state.

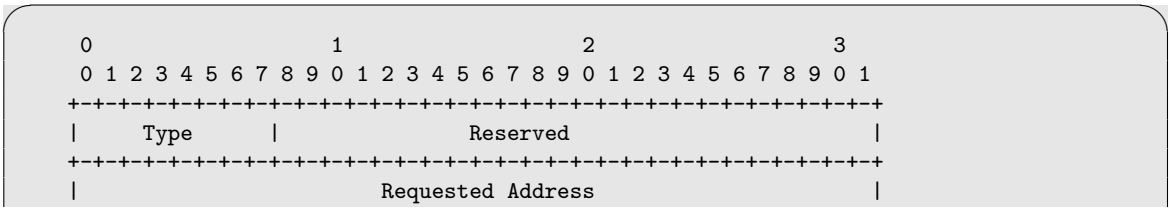
AREQ Packet format



The format of the AREQ message proposed by Perkins et al. [3] is illustrated above, and contains the following fields:

- Type: 1
- Reserved: Sent as 0; ignored upon reception.
- Originator's address: The randomly selected temporary address used by the originator of the flooded AREQ message. Is used as a temporary identification to receive the answers to the AREQ message.
- Requested Address: The randomly selected address that is being requested by the node issuing the address request.

AREP packet format



```
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

The format of the AREP message proposed by Perkins et al. [3] is illustrated above, and contains the following fields:

- Type: 2
- Reserved: Sent as 0; ignored upon reception.
- Requested Address: The randomly selected address that is being requested by the node issuing the AREQ.

Detailed procedure [3]

Address Request (AREQ) The node selects a random ID that will serve as a source address for the short period while the node performs the address discovery. Then, the node issues an Address Request (AREQ) for that randomly selected address. The node places its randomly selected source address in the Originator's Address field. The node then broadcasts this request to its neighbors. It then sets a timer for ADDRESS_DISCOVERY.

Address Request Processing When a node receives an AREQ message, the node first notes the Requested_Address and Originator's Address. It checks its buffered list of AREQ message identifiers to determine whether it has seen this request before. If it has already seen this request, it discards the packet. Otherwise, the node enters the values of these fields into a temporary buffer. These two values serve to uniquely identify the request. If the node receives this request again as the packet is rebroadcast by its neighbors, it will note that it has already received the request, and hence will not reprocess the packet.

Next, the node creates a reverse route entry for the node indicated by the Originator's Address field. It enters this destination address in its route table, and uses the node from which it received the AREQ as the next hop towards the source node. The node enters a lifetime for this route of REVERSE_ROUTE_LIFETIME. In this way, if the node later receives an AREP, the node will have a current route to the source node. Hence it will be able to forward the AREP towards the source node.

The node then checks whether its own address matches the requested address in the AREQ. If the node's address does not match the requested address, it rebroadcasts the packet to its neighbors.

On the other hand, if the node has the same address as that in the AREQ, the node MUST reply to the packet. To do so, it creates an Address Reply (AREP) packet. It copies the Requested_Address from the AREQ message, and places them in the AREP. It then unicasts this packet to the source node, as indicated by the source address in the header of the received AREQ message. The reverse route that was created by the AREQ broadcast is used to route the AREP back to the source node.

Address Reply Processing When a node originates an AREQ, it sets a timer for ADDRESS_DISCOVERY. During that time, it waits for the reception of an AREP. If no AREP is returned for the selected address within a timeout period, the node retries the AREQ up to

AREQ_RETRIES times. If, after all retries, no AREP is still received, the node assumes that the address is not already in use, and that the address can safely be taken for its own.

On the other hand, if the node does receive an AREP within the discovery period, and if the Requested_Address match its recorded values, then this indicates that another node within the ad hoc network is currently using that Requested_Address. In this case, the node randomly picks another address and begins the ad hoc DAD procedure again.

Default Values [3]

```
Parameter Name         Value
---------------------- -----
ADDRESS_DISCOVERY      3 * NODE_TRAVERSAL_TIME * NET_DIAMETER / 2
REVERSE_ROUTE_LIFETIME ADDRESS_DISCOVERY * 2
ADDRESS_RETRIES        3
NET_DIAMETER           10
NODE_TRAVERSAL_TIME    40
UID_TIMEOUT            2 * ADDRESS_DISCOVERY
```

3.4 Implementation

3.4.1 Contiki OS

We are going to implement QDAD using the Rime stack. This stack will provide many functionalities that we don't need, but in other case (for example, working directly over the MAC layer) the implementation would be more complex.

To get familiar with the environment, launch the *Instant Contiki* Virtual Machine, open a terminal an run the following commands:

```
user@instant-contiki:~$ cd /home/user/contiki/examples/rime
user@instant-contiki:~/contiki/examples/rime$ make TARGET=cooja example-multihop.cooja
```

This command will compile the *Multihop example* for RIME and launch the simulation environment. Add 40 motes to a 200 × 200 m scenario, and in the *Network* window change the parameters as shown in figure 3.1

Try to understand the code in example-multihop.c (Ask in case you have any doubt!!!), and launch the simulation[2].

Implement the QDAD protocol presented above, using the files example-multihop.c and example-broadcast.c as an starting point. You will also need other functions described in the official documentation about the *Rime communication stack* (http://www.eistec.se/docs/contiki/a02302.html).

[2] In order to ask a node to send a packet, you have to *press* the button of the node (click with the right button of your mouse on the desired node and choose *Click button on Contiki XX*)

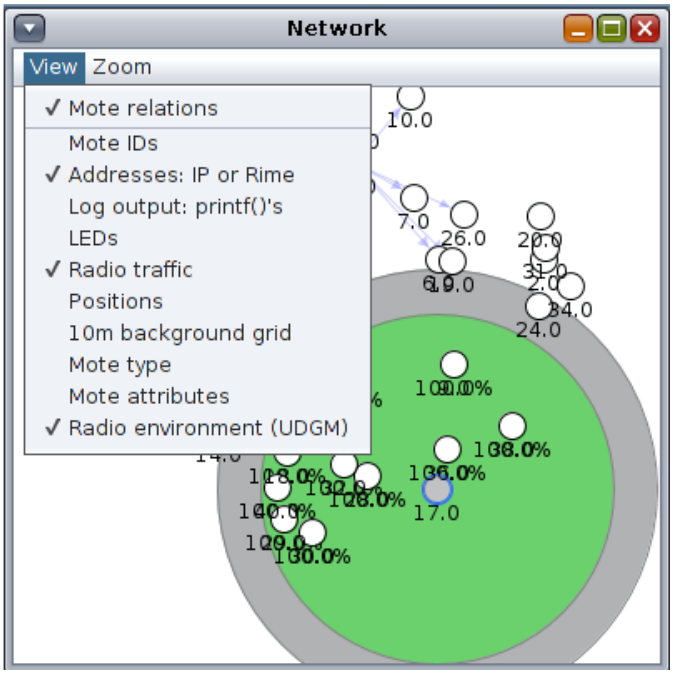


Figure 3.1: Visualization parameters for the simulation

`Cooja` assigns sequential addresses to your nodes. In order to test your implementation, you will have to change the addresses calling `linkaddr_set_node_addr` with a random number. Also, you can use the simulation interface to assign the same address to two nodes[3], and check if they are able to discover that they addresses are not unique.

3.4.2 Ideas for the implementation

In order to implement a *proof-of-concept* of the protocol, you can follow the next ideas:

- You can create a simplified version of the `AREQ` packet by using `packet buffers`[4] and sending the following information in the payload of the Rime packet:

```
struct message {
  linkaddr_t originators_address;
  linkaddr_t requested_address;
  uint8_t hops;
};
```

- For the `AREP` you can just use a multihop packet without any payload (when a node receives a multihop packet, it would assume that its address is duplicated). Obviously, in a *real* implementation we can't follow this approach, because multihop

[3] `Right click→Mote tools for Contiki XX→Variable watcher`, and then select the variable `linkaddr_node_addr`

[4] `http://anrg.usc.edu/contiki/index.php/Packetbuffer_Basics#void_packetbuf_clear.28void.29`

packets will be used to exchange information (and not only to notify duplicated addresses).

- Create a function that will handle broadcast messages. This function should discard packets that we have already processed, or packets which have been re-broadcasted too many times (use the `hops` field). The function should also keep a list of addresses and the next hop to reach such address (in order to forward packets towards that address). Finally, this function should re-broadcast `AREQ` packets to the neighbors of the node.
- Create a function that forwards any received multihop packet. It should check in a list the next hop to reach the destination of the packet.
- Create a function that handles any multihop packet that is addressed to this node (in this simplified implementation, a multihop packet addressed to this node means that the temporary address is duplicated and that the procedure to find an address should start again).
- In the main process, create an infinite loop that implements an state machine with three states: `no_address`, `advertising`, and `normal`. In the `advertising` state, the node broadcasts an `AREQ` message, waits until a timer expires, and repeats the procedure two more times. Wen the third timer expires, the node can use the selected address as permanent.

- *1.- Describe the relevant parts of your code, the tests performed and the results.*
- *2.- Provide a copy of your code (compressed in a `.tar.gz` file).*
- *3.- Check the protocol in the Zolertia nodes. Provide a copy of the code and the binary (compressed in a `.tar.gz` file).*

3.5 References

1 Sahadevaiah, K., Ramakrishnaiah, N., and Reddy, P. P. (2015). "IPv6 Address Auto-Configuration Protocol for Mobile Ad Hoc Networks". Procedia Computer Science, 57, 907-914.

2 Kilian Weniger and Martina Zitterbart, "Address Autoconfiguration in Mobile Ad Hoc Networks: Current Approaches and Future Directions". IEEE Network, Page No. 6-11, July/August 2004.

3 Charles E. Perkins, Jari T. Malinen, and Ryuji Wakikawa. "IP Address Autoconfiguration for Ad Hoc Networks". Mobile Ad Hoc Networking Working Group. IETF Internet Draft. 2001.