AA. 2017/2018

Andrea Milanta

878403

IoT Project Report

Ad-hoc On-demand Distance Vector routing protocol

Goal

The goal of the project is to provide an implementation of a routing protocol similar to the Ad-Hoc On-Demand Distance Vector (AODV).
The provided implementation should be tested in a simulated environment on the provided topology.

Approach

The development of the project followed the classic approach of software development. The first phase was the clear identification of the requirements. This involved the study both of the original AODV and the modified version proposed to clearly define the specifications of the protocol. Once the protocol had been clearly understood and uniquely defined, the focus was shifted on the actual implementation with the definition of the developing and testing environment. The next step was the definition of the software architecture, with the identification of the key structures and processes. Finally, the code has been written and tested in the simulation environment. During this process the log file structure was also defined so as to be as clear as possible.

Protocol Specifications

The protocol provided is a modified version of the AODV protocol. The following requirements and constraints were identified:

1. Each node must be able to generate random data every 30s.
2. Each node must be able to save the next hop of a route towards a given destination.
3. Each node must be able to forward a DATA package towards its required destination if the required route is available.
4. Each node must be able to broadcast a ROUTE_REQUEST and initialize the route discovery process if there is no route in the routing table for the required destination.
5. Each node must send a DATA package it had previously delayed once the route to its destination becomes available.
6. Each node must be able to discard ROUTE_REQUEST it has already received.
7. Each node must be able to broadcast forward new ROUTE_REQUEST where it is not the final destination.
8. Each node must be able to improve the route towards a destination if a shorter (less hops) one is found.
9. Each node must be able to make routes older than 90s invalid.
10. Each node must be able to forward a ROUTE_REPLY to all neighbours which sent the corresponding ROUTE_REQUEST less than 1s before.
11. Each node must be able to send back a ROUTE_REPLY if it receives a ROUTE_REQUEST with itself as destination.

Operating System

The project was implemented using Contiki 2.7. This was chosen for its simplicity, its effective concurrency and its user-friendly dedicated simulation environment (Cooja).

Main components

As part of the architectural design, we identified the main logic components to be developed and described their software implementation.

Packets

Data is shared over the network in packets of bytes (chars). To avoid working with character array, as soon as a packet is received, it is converted to a corresponding data structure which is then used for the rest of the computation. Each packet type should run on its own channel, however as an extra layer of protection, the translation software checks the custom header before choosing the corresponding structure.

- **DATA_PACKET:** data packet. Contains fixed size payload and id of destination.
- **RREQ_PACKET:** packet representing a ROUTE_REQUEST. Contains the original source, the desired destination and an incremental ID.
- **RREP_PACKET:** packet representing a ROUTE_REPLY. Contains the original source, the desired destination and the ID of the corresponding request plus a number representing the hops, which is increased every time it is forwarded.

Communication

Communication happens across three channels, each dedicated to a packet type.

- **Data channel:** Unicast channel dedicated to DATA_PACKET packets.
- **RREQ channel:** Broadcast channel dedicated to RREQ_PACKET packets.
- **RREP channel:** Unicast channel dedicated to RREP_PACKET packets.

3 channels were preferred over one to take advantage of Contiki callback system, which allows to specify a function which is executed every time a packet is received on the associated channel. 3 channels allow for 3 different callback functions instead of a single one with multiple if inside.

Tables

Tables were used to store information beyond the time scope of a single process.

- **routingTable:** Holds the best route for each destination. The route is described by the next hop and the total number of hops to the final destination.
- **discoveryTable:** Holds all ROUTE_REQUEST received in the last second and which are waiting for a ROUTE_REPLY.
- **waitingTable:** Holds all DATA packages waiting to be sent. They will be sent once the route to their destination becomes available.

In terms of implementation, each table is a fixed size array of structs (see *AODV.h*) including a validity flag and age.

Processes

Process are independent "programs" which run concurrently and share the same data resources. Thus, they correspond to the main tasks required by each node

- **Initialization process:** Starts all connections and initializes tables.
- **Data process:** Generates a data packet every 30s, tries to send any data packet (generated and received) to their destination and puts them in the waiting table if the route is not available.
- **RREQ process:** Handles the generation and forwarding of ROUTE_REQUESTs.
- **Aging process:** Routinely (once per second) scans all tables to update the age of every entry and make invalid those which become too old. While scanning the waiting table, it also checks if a route has become available for any destination and if so proceeds to send the corresponding data.

Extra features

Following testing, I decided to implement some extra features and add some modifications to improve the clarity and significance of the result.

- **Debugging button:** A debugging button has been added that allows to follow what happens during the simulation much more in details than the default log.
- **LEDs:** They were added to each mode to give a quick and immediate feedback of the main events.
  - **Green:** Data process is active. A data packet is created or received and attempted to be forwarded.
  - **Yellow:** RREQ process is active. a ROUTE_REQUEST is being broadcasted.
  - **Red:** A route entry has been invalidated due to age.
- **Random Initial Delay:** To reduce conflicts a random initial delay ([0, 30s]) has been introduced before sending the first data packet.

Customizability

The implementation offers some degree of customizability, accessible as #define in the code of *main.c* and *AOVD.h*.

The customization mostly regards payload size, time constraints, channels and number of nodes.

Additional Info

The full code is available at: https://github.com/AndreaMilanta/AODV