Internet of Things

Laboratory 4

mDNS & DNS-SD
(2024-2025)

”Gheorghe Asachi” Technical University of Iaşi
Faculty of Automatic Control and Computer Engineering
Computer Engineering Department

Contents

List of Tables

List of Figures

1. Objectives

Understanding the concept of a decentralized DNS system and implementation of an application facilitating communication with other development platforms available in the laboratory.

2. mDNS

Multicast DNS (mDNS) is a communication protocol primarily designed for resolving IP addresses associated with queried domain names. The resource types transmitted are identical to those used in standard DNS infrastructure; however, communication occurs in a decentralized manner via the multicast group 224.0.0.251 (IPv6: ff02::fb).

Specifically, a device on a local network joins the aforementioned multicast group and initiates an mDNS server (resolver) on which it configures a specific name. Subsequently, the server responds to queries received on the multicast group for the resource `name.local` (`.local` being the default domain used by mDNS, conventionally limited in scope to the local network). Responses are typically sent to the multicast group by default, although unicast transmission is also possible. The choice between multicast and unicast depends on the system characteristics: multicast is employed when the systems involved have caching capabilities, whereas unicast is used to reduce network traffic when caching is unavailable.

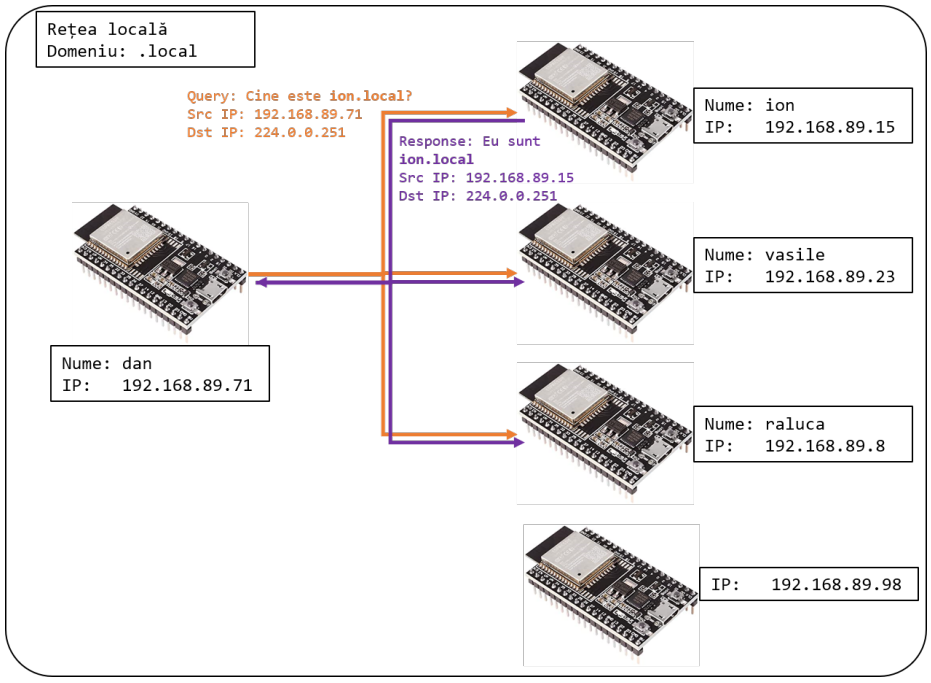

Fig. 1. Sample mDNS interaction

3. DNS-SD

The mDNS implementation can also be utilized to discover services exposed on a local network. For example, a network printer may respond to queries for *printer* services, providing details such as its configured name/IP address, default settings, and access path (e.g., web interface).

Service names are stored through PTR records. The key follows the format `_service_type._protocol`, where `_service_type` is either a proprietary service name or one registered by IANA (the complete list is available here), and `_protocol` represents the transport layer protocol used (`_tcp` or `_udp`). Additionally, the PTR record includes an instance name alongside the service type and protocol, enabling subsequent queries for other service-specific records.

```
#<name>      <ttl> <cls> <type> <rdata>
 _http._tcp  3600  IN    PTR    ion_s_web_server._http._tcp
```

Listing 1. Sample PTR record

Additional information for a service can be obtained through the following types of records:

- SRV - contains information regarding the location of the service (hostname and port number).

```
#<serv.proto.dom> <ttl> <cls> <type> <prio> <pond> <port> <hostname>
 _http._tcp.local 3600  IN    SRV    0      0      80     esp32_board.local
```

Listing 2. Sample SRV record

- TXT - contains additional details necessary for accessing the service (e.g., default path, buffer sizes, timeout values).

Info

To discover all available services within the local domain, a query for the PTR record with the key `_services._dns-sd._udp.local` can be executed. All mDNS servers exposing services will respond to this query.

4. Tasks

1. Refer to the Espressif documentation to familiarize yourself with mDNS.

2. Create a new project, importing the code used in previous labs for connecting to the lab's AP. Configure the device hostname (`mdns_hostname_set()`) using the format `esp32-familyname.local`.

3. Use the `mdns_query_a()` function to discover and list the IP addresses of other available platforms within the laboratory network.

> **Info**
>
> In order to use the mDNS library, copy the *mDNS* folder from the attached archive to the project root folder. Add the directive `list(APPEND EXTRA_COMPONENT_DIRS mdns)` to the `CMakeLists.txt` file from the root folder, just before the last line (`project(...)`).

4. Modify the previous application in order to discover all the available services in the local network.

5. Create a new project and import the code used in Lab 2 (task 2) for controlling a LED on another ESP32 platform via UDP datagrams.

   In the case of this application, a limitation was the prior knowledge of the IP address of the platform to which datagrams were sent. Therefore, the code will be extended in the following way:

   Reception configuration:

   (a) Assign a unique hostname to your platform in the format `esp32-familyname`. Verify the assignment by running `ping esp32-familyname.local` from the lab PC connected to the *lab-iot* AP.

   (b) Register a service of type `_control_led._udp` associated with a freely chosen port number. Integrate this port number into the existing code (within the bind function).

   Astfel, nu mai este necesară cunoaşterea adresei IP a plăcii şi a portului folosit la nivelul socket-ului pentru controlul LED-ului la distanţă.

   Transmission configuration:

   (a) Upon pressing the button, perform a query for services of type `_control_led._udp`. Await responses for 5 seconds.

   (b) If multiple responses are received, randomly select one response and extract its IP address and port number.

   (c) Send a UDP datagram to the selected IP-port pair using the format specified in Lab 2.

Support documents

- ESP32 datasheet and manual
- esp-idf API reference
- FreeRTOS API reference

Errata

Problem Description: The platform rarely responds to mDNS queries, making it difficult to discover its name or exposed services.
Problem Analysis: Debugging revealed that multicast frames are rarely received. While it is normal not to receive all multicast/broadcast frames in Wi-Fi networks, the observed scale is unusual. This behavior occurs because:

- These frames are transmitted with acknowledgment from a station to the Access Point (AP), but are sent unacknowledged from the AP to other stations.
- When stations connected to the AP use power management mechanisms (IEEE 802.11 Power Management Protocol), the AP buffers multicast/broadcast frames during the DTIM period and delivers them to stations upon request at the end of this period.

Correlating the above information with the fact that by default, the Wi-Fi interface of the platform is configured to use power-saving mode (PS), the following experimental observations were made:

- With PS mode enabled, broadcast frames are correctly received (tested using the ARP protocol).
- With PS mode enabled, multicast frames are very rarely received (tested using UDP sockets).
- With PS mode disabled, both multicast and broadcast frames are correctly received.

It is concluded that there might be an implementation deficiency either in the Wi-Fi driver or within a component of the lwIP stack. Tests were performed using esp-idf versions 4.3 and 4.3.2. This issue was not observed previously in version 4.2.
Solution: When configuring the Wi-Fi interface, after the call to `esp_wifi_start()` and before establishing a connection (prior to the call to `esp_wifi_connect()`), disable the power management mode using the command `esp_wifi_set_ps(WIFI_PS_NONE)`.