Link-Layer Discovery Protocol (LLDP) Agent

Introduction

In this project your goal is to create a simple LLDP agent.

The finished LLDP agent should be able to:

- Announce itself by sending LLDP messages to the network
- Parse LLDP messages received from the network and print them on `stdout`.

Along with this task sheet, you have received a framework for the LLDP agent, which you will have to complete. You will only have to fill in the parts relevant to LLDP frame processing.

Usage

> All components that open raw sockets need **root priviliges**. This includes the LLDP agent itself as well as the provided test suite.

You can start the provided LLDP agent by running the following command from the project root, where `<interface>` represents the network interface the agent uses to send and receive LLDPDUs.

```
sudo python main.py -i <interface>
```

In addition we provide you with an automated test suite which you can use to verfify all the parts of your implementation.

The tests can be executed by running the command `sudo nosetests` in the project root folder.

Passing

The following requirements have to be fulfilled to pass the project:

- When receiving packets from the network, only LLDP frames should be parsed.
- Your agent should only handle frames which are directed to one of the LLDP multicast addresses.
- You don't need parse TLVs which are not supported by your agent.
  Make sure that frames which include TLVs which are not supported by the agent don't crash the agent.
- Ensure that your agent does not parse and output LLDP frames it sent itself.
- Generated frames should include all *mandatory* TLVs in the *correct order*.

> You only need to send your messages to the nearest-bridge MAC address (`01:80:c2:00:00:0e`).

> Make sure you raise appropriate exceptions as detailed in the Framework section.

To pass this project **all tests we provide have to be passed**. They check most of the conditions above. There are **additional tests not visible to you**, which check whether you correctly implemented the specification.

We encourage you to write your own tests while developing. You can use the tests provided to you as a reference.

Submission

We **require** you to hand in the code as a `.zip` archive, using the **Course Management System**. The structure should be **the same we provided**. You may include additional files (such as additional tests), however this is not required to pass the project.

We will **deduce points** for not adhering to the folder structure, so do not risk failing the project by not providing the files in the correct structure.

Framework

We provide you with a framework for writing your agent. You have to fill in all code sections marked with "`#TODO: Implement`".

`lldpa/lldpAgent.py` contains the main application logic. You need to implement:

- `run_receive`: Implements the process of receiving LLDP frames from the network.
- `run_announce`: Implements the process of sending LLDP frames to the network.
- `parse_lldp_frame`: Implements the parsing of raw data received from the network into an `LLDPMessage` object.
- `generate_lldpdu`: Generates the raw byte data representing the LLDP data unit from the python classes.

`lldpa/tlvs` contains one class for each TLV your agent has to support. You will have to implement of their methods.

See the comments in the class files for detailed information about specific methods.

Exceptions

To sensibly handle errors in the LLDP agent, the following Exceptions are available.

- `LLDPParseException`: Raised whenever parsing fails (incomplete data, etc.).
- `ImproperDestinationMACException`: Raised when the destination MAC address is not one of the expected LLDP destination addresses.
- `EoLLLDPDUNotEmptyException`: Raised when there is something wrong with the EoLLDPDU TLV.
- `OptionalTLVTypeOutOfRangeException`: Raised when an unknown TLV is encountered.
- `ImproperTLVOrderException`: Raised when the order of TLVs is not correct.

Receiving Frames

When the LLDP agent has received and parsed an LLDP frame, it should print a message detailing the received information to *stdout*.

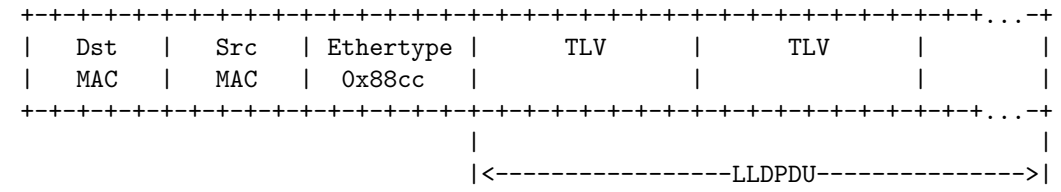
The message **MUST** have the following format.

```
LLDPMessage(src_mac=ab:cd:ef:01:23:45,chassis_id=ab:cd:ef:01:23:45,port_id=eth0,ttl=120)
```

Protocol Specification

This section includes a quick reference to the LLDP wire format. For more details on LLDP refer to *U11 "Link Layer (MAC)"* (or *IEEE 802.1ab*).

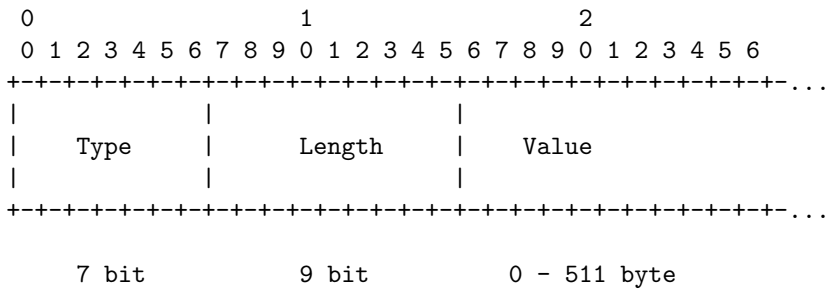
LLDP Ethernet Frame Structure



Required TLVs in LLDPDU:

- Chassis ID TLV (Type: 1)
- Port ID TLV (Type: 2)
- Time to live TLV (Type: 3)
- End of LLDPDU TLV (Type: 0)

TLV Structure



Multicast Addresses

- `01:80:c2:00:00:0e` (Nearest Bridge)
- `01:80:c2:00:00:03` (Nearest Non-TPMR Bridge)
- `01:80:c2:00:00:00` (Customer Bridge)

Supported Subtypes

For Chassis ID, we require you to parse and build subtype 4, which is used to encode a MAC.

For Port ID, we require you to parse and build:

- subtype 3, which is used to encode a MAC.
- subtype 7, which is used to encode a port number.