CMPU4021 Distributed Systems – Labs

Week 1 – Introduction to Networking in Python

Learning Outcomes:

- Printing your machine's name and IPv4 address
- Retrieving a remote machine's IP address
- Converting an IPv4 address to different formats
- Finding a service name, given the port and protocol
- Converting integers to and from host to network byte order

Note 1: You may need to setup the environment for smooth running analog to setting up for Java.

Python version 3.12.4 is installed in Labs.

T1. Printing your machine's name and IPv4 address

Sometimes, you need to quickly discover some information about your machine, for example, the hostname, IP address, number of network interfaces, and so on.

Examine `PT1\1_1_local_machine_info.py` and run

- `python 1_1_local_machine_info.py`

T2. Retrieving a remote machine's IP address

Often you may need to translate a machine's hostname into its corresponding IP address, for example, a quick domain name lookup. This recipe introduces a simple function to do that.

Examine and run `PT2\1_2_remote_machine_info.py`

If you run it, it gives the following output:

```
> python 1_2_remote_machine_info.py

IP address of www.python.org: 151.101.36.223
```

Change the `remote_host` variable value. See the output.

T3. Converting an IPv4 address to different formats

When you would like to deal with low-level network functions, often the usual string notation of IP addresses are not very useful. They need to be converted to the packed 32-bit binary formats.

Examine and run `PT3\1_3_ip4_address_conversion.py`. You should see the following output:

- `python 1_3_ip4_address_conversion.py`

```
IP Address: 127.0.0.1 => Packed: 7f000001, Unpacked:
127.0.0.1
IP Address: 192.168.0.1 => Packed: c0a80001, Unpacked: 192.168.0.1
```

In this recipe, the two IP addresses have been converted from a string to a 32-bit packed format using a for-in statement. Additionally, the Python `hexlify` function is called from the `binascii` module. This helps to represent the binary data in a hexadecimal format. Change the `ip_addr` value. See the output.

T4. Finding a service name, given the port and protocol

If you would like to discover network services, it may be helpful to determine what network services run on which ports using either the TCP or UDP protocol.

Examine and run PT4\1_4_finding_service_name.py. If you run this script, you will see the following output:

- `python 1_4_finding_service_name.py`

```
Port: 80 => service name: http
Port: 25 => service name: smtp
Port: 53 => service name: domain
```

This indicates that http, smtp, and domain services are running on the ports 80, 25, and 53 respectively. In this recipe, the for-in statement is used to iterate over a sequence of variables. So for each iteration, we use one IP address to convert them in their packed and unpacked format.

Check the output for different ports.

T5. Converting integers to and from host to network byte order

If you ever need to write a low-level network application, it may be necessary to handle the low-level data transmission over the wire between two machines. This operation requires some sort of conversion of data from the native host operating system to the network format and vice versa. This is because each one has its own specific representation of data.

Python's socket library has utilities for converting from a network byte order to host byte order and vice versa. You may want to become familiar with them, for example, `ntohl()/htonl()`.

Examine and run `PT5\python 1_5_integer_conversion.py`. If you run this recipe, you will see the following output:

- `python 1_5_integer_conversion.py`

```
Original: 1234 => Long host byte order: 3523477504,

Network byte order: 3523477504

Original: 1234 => Short host byte order: 53764,

Network byte order: 53764
```

In this example, we take an integer and show how to convert it between network and host byte orders. The `ntohl()` socket class function converts from the network byte order to host byte order in a long format. Here, `n` represents network and `h` represents host; `l` represents long and `s` represents short, that is, 16-bit.

Change the `data` value. Check the output.

References

- Chapter 1, Kathiravelu P., Sarker F., Python Network Programming Cookbook, Second Edition, Packt Publishing Ltd, 2017
- https://www.python.org/
- https://docs.python.org/3/library/ipc.html