2019

SOFTENG 364: Assignment 2: Version 2

Craig Sutherland

Contents

Important Dates

Submission due date: 4 June, 2019 – 5pm on Canvas

Getting started

Download the associated zip file from Canvas. This file contains the following items:

- checksum.py
- crc.py
- ping.py
- routing.py

Extract the code to a folder in your home drive and work from it directly. It is advisable to use a git repository to track your changes.

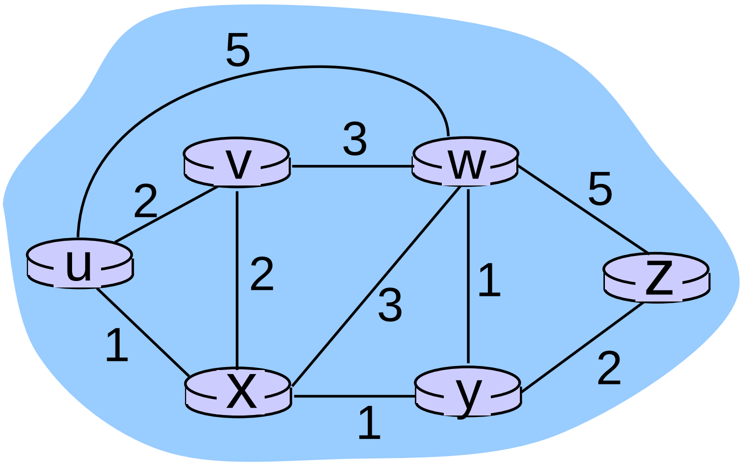
Part 1. Link-state routing

In this part of the assignment, you will implement parts of the Link-state routing algorithm.

Task 1.1. Predecessor lists

Modify `dijkstra_generalized()` (in routing.py) so as to return predecessors (in addition to distances).

Using your new function, **visualize** the least-cost path tree for node u for the following network, as we did in Lab 1:

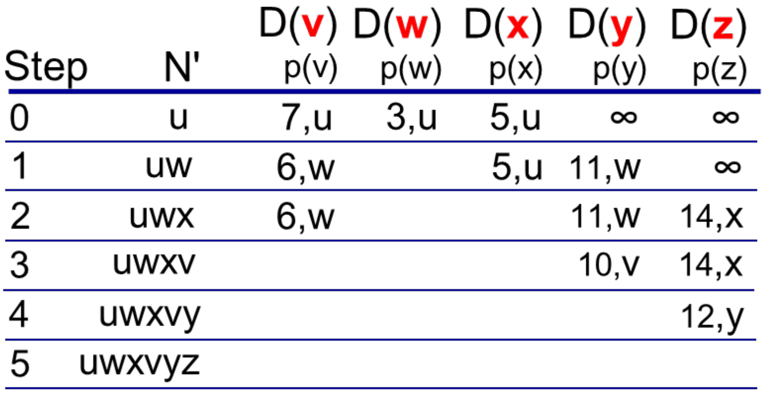


- We need to initialize the predecessor map appropriately and update it each time the distance map is update. Only a few lines of code are required.

Task 1.2. Forwarding tables

Write a function `forwarding` that produces the forwarding table associated with a predecessor map.

- Verify that the output is consistent with the example on Slide 19 of Lecture 1 (see below):



		D(v)	D(w)	D(x)	D(y)	D(z)
Step	N'	p(v)	p(w)	p(x)	p(y)	p(z)
0	u	7,u	3,u	5,u	∞	∞
1	uw	6,w		5,u	11,w	∞
2	uwx	6,w			11,w	14,x
3	uwxv				10,v	14,x
4	uwxvy					12,y
5	uwxvyz					

- Nodes may be visited more than once.

Part 2. Error detection

In this part of the assignment, you will implement error detection.

Task 2.1. One's complement in Python

Write a function `hextet_complement(x)` to compute the one's complement of a Python `int` regarded as a fixed-width hextet (16 bits, two bytes/octets, four nibbles).

- Use the the **invert** operator ~ and a suitable mask, as discussed in the lab.
- Don't worry about handling the case where the argument itself occupies more than one hextet.

Task 2.2. The Internet Checksum

Implement the Internet Checksum (https://en.wikipedia.org/wiki/IPv4_header_checksum) in Python.

- Use hextet_complement() to compute the one's complement.
- Your function should work for any Python sequence whose elements are bytes.
- Ensure that you can reproduce the calculation given in Section 3 of IETF 1071 (https://tools.ietf.org/html/rfc1071).

We'll use this function to check IP packets in Part 3.

> **Note:** (from the lab worksheet) that an implementation in C of the Internet Checksum is provided in RFC 1071, Section 4.

```
/*
The following "C" code algorithm computes the checksum with an inner loop that sums 16-
bits at a time in a 32-bit accumulator.
*/
{
    /*
    Compute Internet Checksum for "count" bytes beginning at location "addr".
    */
    register long sum = 0;

    while( count > 1 )  {
    /*  This is the inner loop */
        sum += * (unsigned short) addr++;
        count -= 2;
    }

    /*  Add left-over byte, if any */
    if( count > 0 )
        sum += * (unsigned char *) addr;

    /*  Fold 32-bit sum to 16 bits */
    while (sum>>16)
        sum = (sum & 0xffff) + (sum >> 16);

    checksum = ~sum;
}
```

- With respect to the C code above:
    - A literal translation into Python of the C code `sum += * (unsigned short) addr++;` is not possible. Rather, we will need to treat the two adjacent bytes separately and shift the bits in the "larger" one.

  - Warning: Recall from the worksheet that Python's operator ~ does **not** work in the same was as C's version.
- If you like, it is possible to index every second element of a sequence data as follows:

```
>>> data = b'abcdefg'
>>> data[0::2]
b'aceg'
>>> data[1::2]
b'bdf'
>>>
```

- If you like, Python does have a built-in sum function.

Task 2.3. Cyclic Redundancy Checks

Implement a function to perform CRC checks with a given generator on an arbitrary sequence of bytes.

- Verify that your function reproduces the calculation of slide 6-15.
- There is no need to store the quotient.
- Use the template file crc.py, which contains several test cases.
- The sample code provided in the lab worksheet performs one step of the long division; please ensure that you are happy with this. Hence, we need iterate through a sequence of such steps: Very little new code is required because the polynomial coefficients are either 0 or 1.

Part 3. ICMP and socket programming

In this part of the assignment, you will use your modules icmp and checksum to re-implement ping in Python.

- To send ICMP messages, we'll need to use a so-called raw socket, as returned by the following snippet:

```
import socket
socket.socket(family=socket.AF_INET,
              type=socket.SOCK_RAW,
              proto=socket.getprotobyname("icmp"))
```

Your system might require Administrator permissions to open raw sockets: In particular, you may not be able to complete this task on the Faculty's lab PCs.

- Use Python's argparse module to process the following command-line options:

Name	Abbreviation	Type	Default	Help
`--timeout`	`-w`	`int`	`1000`	Timeout to wait for each reply (milliseconds)
`--count`	`-c`	`int`	`4`	Number of echo requests to send
`hosts`		`str`		URL or IPv4 address of target host(s)

A demonstration of the required line interface is shown in the examples below.

- Use Python's struct and collections.namedtuple libraries to to de/serialize ICMP messages to/from byte sequences.
- Use Python's with statement to guarantee that your socket is closed gracefully. socket module's documentation provides several examples (https://docs.python.org/3/library/socket.html#example).
- Use a suitable function from the time module to estimate round-trip time (elapsed time).
- Each ICMP echo request should carry the time instant at which it was created/sent. In reality, this would not be necessary, but it relates to one of the challenge problems.

- Use exceptions to signal checksum errors and timeouts. All exceptions should be handled in verbose_ping().
- Use built-in functions to calculate the minimum, maximum, and mean of the calculated round-trip times, as demonstrated in the example below.
- Complete details about the ICMP messages used for ping implementations are detailed on wikipedia.org (https://en.wikipedia.org/wiki/Ping_(networking_utility)).
- A detailed template is available to help with Part 3. Study the template and replace each "TODO" with suitable code.
- The type specifiers passed to struct.pack and struct.unpack must be consistent with the ICMP protocol's packet format:

Field	Integer Type	Size in bytes
type	unsigned char	1
code	unsigned char	1
checksum	unsigned short	2
identifier	unsigned short	2
sequence_number	unsigned short	2

- The type of our payload will be `float`:

```
>>> import time
>>> type(time.process_time())
<class 'float'>
>>> type(time.perf_counter())
<class 'float'>
>>> type(time.clock())
<class 'float'>
```

You can execute `ping.py` from the IPython console inside Anaconda using `%run` e.g.

```
%run ping --help
%run ping --count 5 www.google.com
```

Example: Command line interface

```
> python ping.py --help
usage: ping.py [-h] [-w milliseconds] [-c num] host [host ...]

Test a host.

positional arguments:
  host                  URL or IPv4 address of target host(s).

optional arguments:
  -h, --help            show this help message and exit
  -w milliseconds, --timeout milliseconds
                        Timeout to wait for each reply (milliseconds).
  -c num, --count num   Number of echo requests to send.
```

Example: Successful invocation

```
>python ping.py www.python.org --count 3
Contacting www.python.org with 36 bytes of data
Reply from 151.101.0.223 in 5ms: ICMPMessage(type=0, code=0, checksum=48791,
identifier=33540, sequence_number=0)
Reply from 151.101.0.223 in 12ms: ICMPMessage(type=0, code=0, checksum=35850,
identifier=33540, sequence_number=1)
Reply from 151.101.0.223 in 6ms: ICMPMessage(type=0, code=0, checksum=61385,
identifier=33540, sequence_number=2)
Ping statistics for 151.101.0.223:
    Packets: Sent = 3, Received = 3, Lost = 0 (0% loss)
Approximate round trip times in milli-seconds:
    Minimum = 5ms, Maximum = 12ms, Average = 7ms
```

Example: Timeout

Auckland University's web-server doesn't respond to ICMP echo requests:

```
> python ping.py www.auckland.ac.nz --count 3 --timeout 1500
Contacting www.auckland.ac.nz with 36 bytes of data
Request timed out after 1500ms
Request timed out after 1500ms
Request timed out after 1500ms
Ping statistics for 130.216.159.127:
Packets: Sent = 3, Received = 0, Lost = 3 (100.0% loss)
```

Marking Criteria (Total 8% of course):

Criteria	**Ratings**	
Task 1.1. Predecessor map **[3 marks]**	3 marks:	Sensible, return type, correct output
	2 marks:	Sensible & correct but incompatible interface (return type)
	2 marks:	Sensible but incorrect output
	1 mark:	Correct return type only
	0 marks:	Multiple errors or omissions
1.2. Forwarding table from predecessor list **[3 marks]**	3 marks:	Sensible, return type, correct output
	2 marks:	Sensible & correct but incompatible interface / return type
	2 marks:	Sensible but incorrect output
	1 marks:	Correct interface only
	0 marks:	Multiple errors or omissions
2.2. Internet Checksum **[4 marks]**	4 marks:	Sensible sum (even & odd), fold, complement; correct output
	3 marks:	Doesn't handle odd-length sequence
	3 marks:	Sensible but incorrect output
	1 mark:	Byte pairs aren't summed correctly
	0 marks:	Multiple errors or omissions
2.3. CRC checks **[3 marks]**	3 marks:	Sensible loop, condition, update; correct output
	2 marks:	Sensible but incorrect output
	0 marks:	Multiple errors or omissions
Task 3. Command line arguments **[2 marks]**	2 marks:	All three command line options provided
	1 mark:	One missing option
	0 marks:	Multiple errors or omissions
Task 3. ICMP packet: struct.unpack **[1 mark]**	1 mark:	struct.unpack() called correctly
	0 marks:	Multiple errors or omissions
Task 3. Socket programming: with, sendto/recvfrom, settimeout **[3 marks]**	3 marks:	with, sendto/recvfrom, settimeout used correctly
	2 marks:	No. settimeout
	2 marks:	No. with contruct
	0 marks:	Three errors/omissions
Task 3. Packet statistics **[1 mark]**	1 mark:	{Sent, Received, Lost} calculated and displayed
	0 marks:	Error or omission
Task 3. RTT statistics **[1 mark]**	1 mark:	{Min, Max, Avg} calculated and displayed
	0 marks:	Error or omission
Task 3. Define & throw ChecksumError; handle both errors **[2 marks]**	3 marks:	Define and throw ChecksumError; handle ChecksumError and a time-out error
	2 marks:	One omission
	1 mark:	Two omissions
	0 marks:	Multiple errors or omissions
Overall: Best Practice **[1 mark]**	1 mark:	Awareness of consistency, clarity, efficiency, generality
	0 marks:	One or more issues
Total:	**24 Marks**	

Files to submit:

Code submission (Single zip file)

a. One submission **per person** through Canvas under "Code"

b. Zip file should be named Assignment_**UPI**.zip, replace **UPI** with your own UPI.

 i. For example, Assignment_JDOE123.zip