P3 - Path Finder

May 1, 2022

Contents

The code is similar to code given in book, but converted to C++ to make the code more managable in objects and IPv4 ICMP is implemented.

Problems: Due to some reasons, signal handler is not being invoked by SIGALRM so the serial version is given. To convert the code to parallel implementation, simply uncomment the last line (Receive(this)) of function ICMPService::sendnextTTLRequests() with pthread code with following logic:

```
pthread_t thread;
pthread_create(..., Receiver, this);
pthread_join(..)
```

The decision of only using one thread at any point of time instead of many threads per TTL value is taken because the sequence number for large number of URLs (10000 as given in pdf) will exhaust the available distinct seq numbers in 3 TTL requests. If this limit is low, we just need to replace send_req(i, ttl, i == 0); with send_req(i, ttl, false); inside sendnextTTLRequests() to not to reset sequence number and change some data strutctures to use TTL as another dimension.

1 Structs/classes/namespaces

1.1 ICMPInfo

1. *u_short rec_seq:* Sequence number to send for destination. This will uniquely identify the packet and is used to match the sent packet to calculate RTT.

2. u_*short rec_ttl:* Specifies the ttl value to send.

3. *timeval rec_tv:* Used to calculate RTT.

1.2 Random IP Generator

This namespace contain simple random generation algo to generate IP addresses.

1.3 DNSResolver

This class simply take the given url and converts it to corresponding sockaddr_in. Returns -1 on error or 0 on success.

1.4 CommunicationInfo

1. ICMPInfo* sentInfo: The info that was sent

2. ICMPInfo* recvInfo: The info that was received, this is set by checking the sequence number

3. sockaddr recv_sock_addr: sock_addr data structure that was returned in reply for printing the IP address of router
4. int receive_status: Type of ICMP packet. Same semantics as given in the book.

1.5 ICMPService

1. ttl: TTL to send for every IP in current iteration.
2. sendfd: file descripter of where to send data
3. recvfd: RAW socket for reading
4. SA_Send: socket address that were sent. array index corresponds to sequence number sent - (36767 + ...)
5. sentSeqs: for each IP, store the sequence numbers that were sent. Usng this, we can directly call SA_Send for sequence number to retrive the info.
6. requests: Requests that were sent on the channel and received. Complete record.
7. finishedIP: index of IP address that were completes in less than max_ttl.
8. send_req(): send the requests to IP stored at index *index,* with TTL=*ttl* and a parameter *resetSeq,* when true, will reset the seq to 0 (32768+... is added later)
9. Other functions are doing trivial work.

2 Working of code

Usage: ./a.out <num_ip_to_generate>

First, random IP addresses are generated. Then, urls are taken from urls.txt, converted to corresponding IPs using DNSResolver and then the process continues by sending all packets for TTL=1, then 2, and so on.