RTT

IS F462: Network Programming

Assignment 2

Bir Anmol Singh
Guntaas Singh
K Vignesh

1. Objective

The given problem wanted to use I/O multiplexing in order to resolve RTT (round trip time value) for IPv4 and IPv6 addresses as quickly as possible. For each host, 3 RTT values were to be computed and finally printed as output.

The program aimed at achieving maximum possible throughput (ips/time).

2. Design

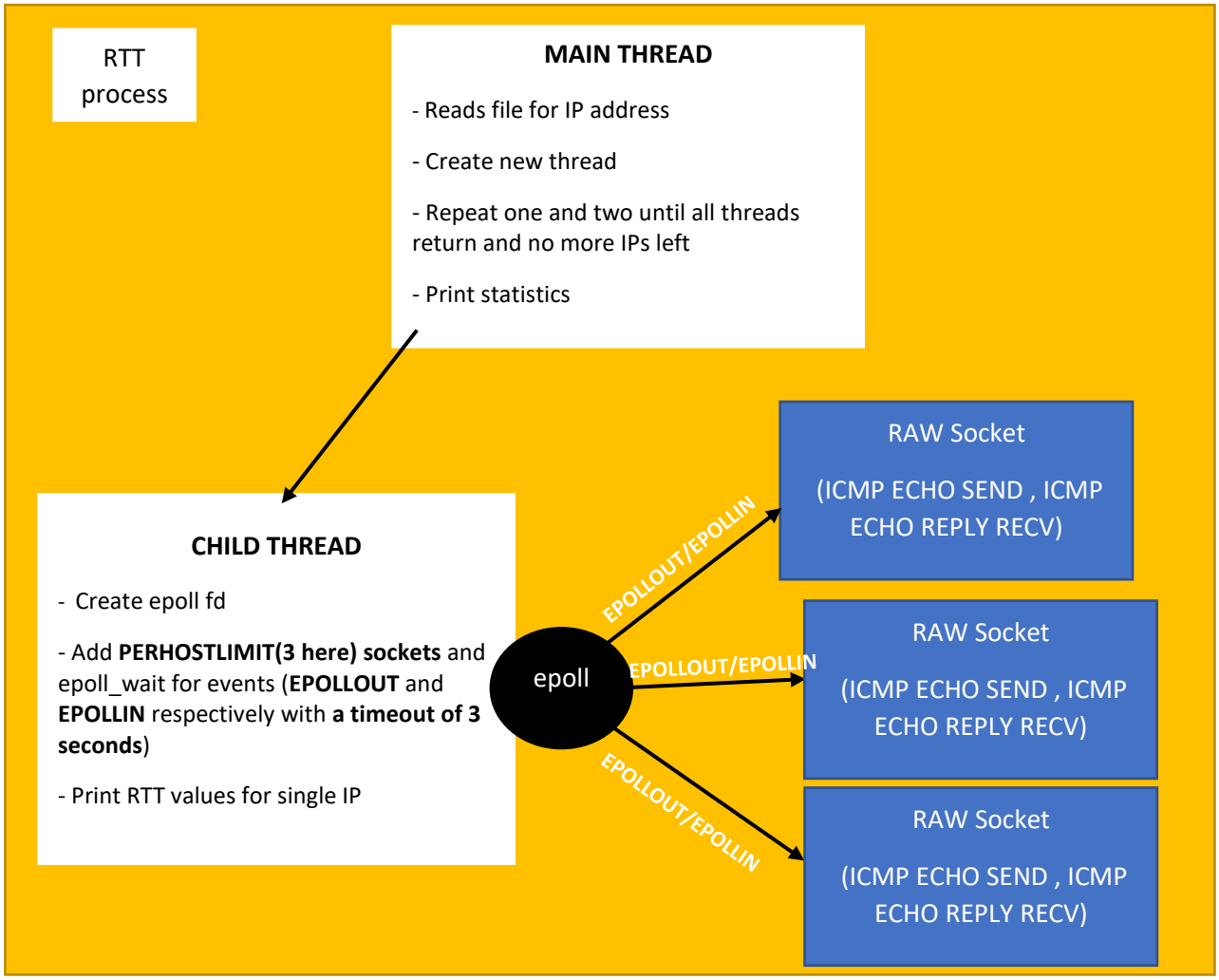


Fig1. Overall design layout

The Application exploits the power of multithreading and EPOLL in order to maximize the throughput. It uses PING technique to measure RTT values. For this we use RAW sockets and send ICMP ECHO messages and receive ICMP ECHO reply. The time difference between the two is used as RTT measure.

Main Thread:

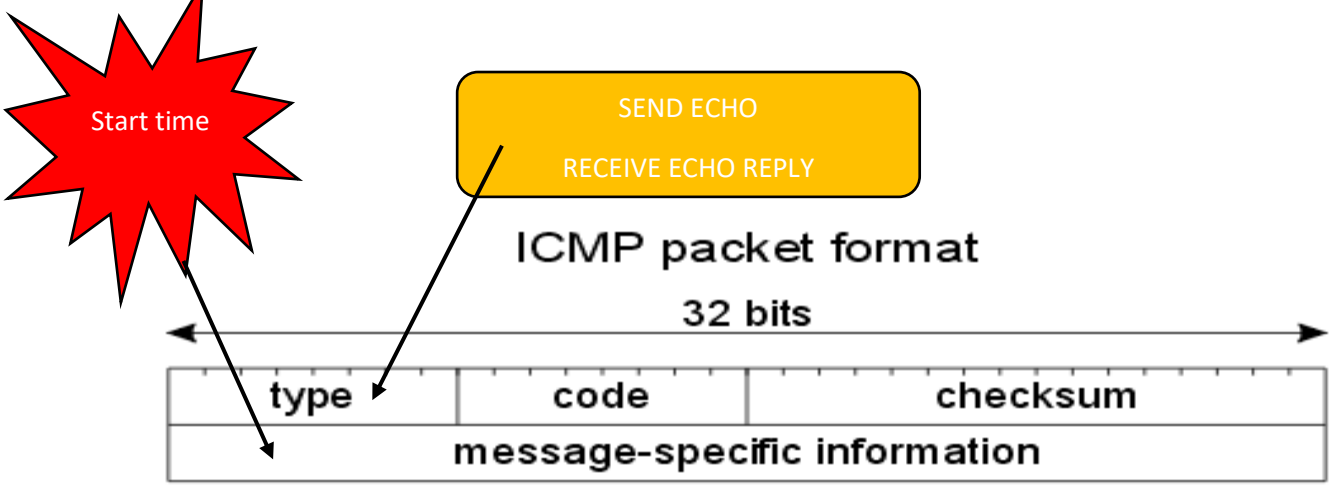
- For each IP address in file, main thread creates a child thread. This step is repeated till no more inputs from file are left to read.
- Main thread also restricts the number of threads so that the socket count does not go beyond a process's maximum limit by using **MAXTHREAD** count which is determined by the formula MAXTHREAD=max(300/PERHOSTLIMIT,1). The formula keeps a balance between epoll and multithreading powers. [**Note**: PERHOSTLIMIT is the number of RTTs to be measured per IP]
- Main thread then waits for all the threads to return by testing the thread count which is guarded by a mutex lock.
- Finally main thread prints final statistics and the process exits.

Child Thread:

- For the IP address, child thread determines the type of IP i.e. v4 or v6. It also tests if the address is valid or not.
- It creates an EPOLL fd for itself.
- Child thread then creates array of size PERHOSTLIMIT of myRTT structures and gets RAW sockets for each of them with IPPROTO_ICMP / IPPROTO_ICMPV6 based on the ipv value.
- Each of these sockets have their sockopts set and have a SNDTIMEO/RCVTIMEO of 1 sec.

```
typedef struct myRTT{
  char addr[40];//IP addr
  int ipv;//IP version
  float rtt;
  int sockfd;
  int seq;
  int rcv_cnt;
}myRTT;
```

- These values are added to EPOLL for EPOLLOUT event.
- In a while loop, we repeat until total received equals the PERHOSTLIMIT and do epoll_wait with a timeout of 3 seconds.
    - In case of EPOLLOUT, we send ICMP message and modify the epoll event to EPOLLIN for the socket. (If failure, we close the socket)
    - In case of EPOLLIN, we read the ICMP, verify that it belongs to us using pid, socket and sequence number. RTT field of the corresponding struct is populated. (In case of success, we close socket else we retry again)
- Once all of PERHOSTLIMIT processing is done, we print statistics and the thread exits.



ICMP Message

Using ICMP allows to ping any host without needing to bother if any port is being used to listen for these messages, and ICMP replies are also automatically sent by hosts.

ICMP also provides ECHO and ECHO REPLY that allows us to receive the data we sent as it is. This allowed for sending start time in the message body of ICMP message and on receipt of this message, we could compute RTT (end – start) time.

ICMP messages were separately created based on ICMPv4 or ICMPv6 format and checksum was also computed for ICMPv4 whereas OS does it on pseudo IPv6 header in ICMPv6.

Each ICMP message was processed in order to check if it belonged to the right process, right socket, and had the right sequence number and format. RTT values are then computed and stored in the myRTT struct finally.

3. Increasing Throughput

A balance between multithreading powers and Epoll is created.

MAXTHREAD=max(300/PERHOSTLIMIT,1)

Threads	Epoll (PERHOSTLIMIT fds)
1	300
10	30
100	3

As Threads increase, EPOLL requirements goes down since each thread has lesser fds to monitor and there is no gain of using EPOLL in such low number of fds. Multithreading allows parallel operations for different IP addresses to be done simultaneously.As fds per IP increase, Threads decrease since we have to keep total fds limit of process in check, thus per threads, we monitor more parallel pings via Epoll. Epoll allows us to operate on all fds in O(1) time plus timeout allows us to prevent indefinite blocking.

4. Results

Testing done with PERHOSTLIMIT 3 and 100 Threads on testIP.txt file having (6069 IPs both v4 and v6), on average yielded a total running time of 2sec.