# Computer Networks Lab4


## 黄嘉祺 221220108


## 1 Program Structure and Design

### 1.1 Using Your TCP Implementation in the Real World

- 在前几次实验中我们已经分别实现了 `bytestream` 、 `reassembler` 、 `wrapper` 、 `tcp receiver` 和 `tcp sender` ，因此已经可以将这些模块组合起来实现一个简单的TCP连接
- 在lab0中，我们调用 `TCPSocket` 类型实现了 `webget` 程序，现在我们将其中的 `TCPSocket` 类型替换为集成了我们自己实现的TCP连接的 `CS144TCPSocket` 类型，并观察能否通过 `webget` 程序的测试

```cpp
#include "socket.hh"

#include <cstdlib>
#include <iostream>
#include <span>
#include <string>

#include "tcp_minnow_socket.hh"

using namespace std;

void get_URL( const string& host, const string& path )
{
  // host: cs144.keithw.org
  // path: /hello
  CS144TCPSocket socket;
```

### 1.2 Collecting Data

- 选取一个RTT大于100ms的地址（在此次实验中选择的是41.186.255.86 (MTN Rwanda)），使用 `ping` 命令与该地址进行通信与数据交换至少两个小时，观察数据包的情况

```
[1731770060.196515] 64 bytes from 41.186.255.86: icmp_seq=36228 ttl=128 time=449 ms
[1731770060.396983] 64 bytes from 41.186.255.86: icmp_seq=36229 ttl=128 time=449 ms
[1731770060.596641] 64 bytes from 41.186.255.86: icmp_seq=36230 ttl=128 time=449 ms
[1731770060.797587] 64 bytes from 41.186.255.86: icmp_seq=36231 ttl=128 time=449 ms
[1731770060.998915] 64 bytes from 41.186.255.86: icmp_seq=36232 ttl=128 time=449 ms
[1731770061.199803] 64 bytes from 41.186.255.86: icmp_seq=36233 ttl=128 time=449 ms
[1731770061.400931] 64 bytes from 41.186.255.86: icmp_seq=36234 ttl=128 time=449 ms
[1731770061.601060] 64 bytes from 41.186.255.86: icmp_seq=36235 ttl=128 time=448 ms
[1731770061.801578] 64 bytes from 41.186.255.86: icmp_seq=36236 ttl=128 time=448 ms
[1731770062.002910] 64 bytes from 41.186.255.86: icmp_seq=36237 ttl=128 time=449 ms
[1731770062.203545] 64 bytes from 41.186.255.86: icmp_seq=36238 ttl=128 time=449 ms
[1731770062.404141] 64 bytes from 41.186.255.86: icmp_seq=36239 ttl=128 time=448 ms
[1731770062.604837] 64 bytes from 41.186.255.86: icmp_seq=36240 ttl=128 time=448 ms
[1731770062.805397] 64 bytes from 41.186.255.86: icmp_seq=36241 ttl=128 time=448 ms
[1731770063.006127] 64 bytes from 41.186.255.86: icmp_seq=36242 ttl=128 time=448 ms
^Clefty777@EDVAC-2023:~/minnow$
 *  还原的历史记录

lefty777@EDVAC-2023:~/minnow$ ping -D -n -i 0.2 41.186.255.86 | tee data.txt
```

# 2 Implementation Challanges

由于之前的实现并没有较大的问题，此次实验将 `TCPSocket` 类替换后成功通过了测试，并未遇到较大的困难

# 3 Remaining Bugs

目前我的代码已经通过了全部的测试，暂时未发现明显的bug

# 4 Experimental Results and Performance

## 4.1 check_webget

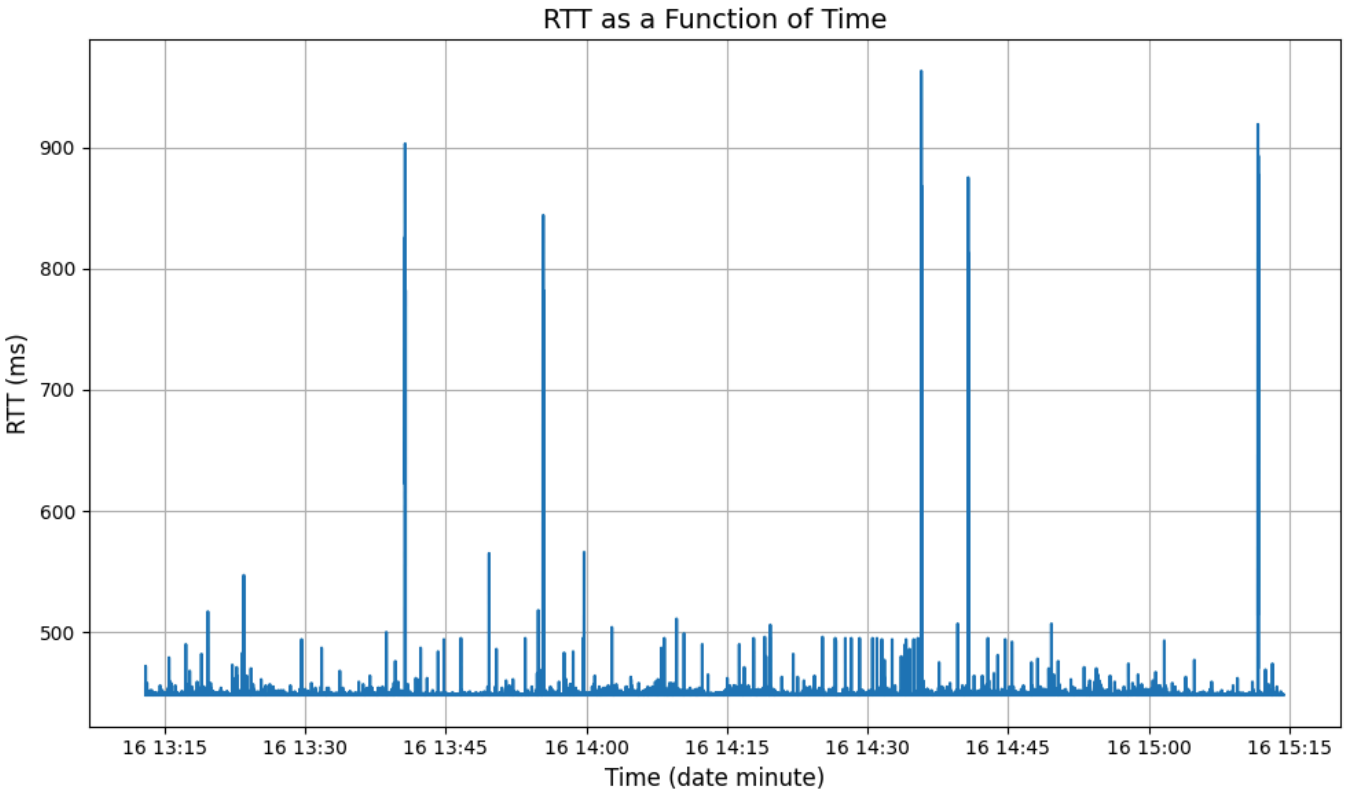
```
lefty777@EDVAC-2023:~/minnow$ ./scripts/tun.sh  start  144
[sudo] password for lefty777:
lefty777@EDVAC-2023:~/minnow$ echo 0 | sudo tee /proc/sys/kernel/randomize_va_space
0
lefty777@EDVAC-2023:~/minnow$ cmake --build build --target check_webget
Test project /home/lefty777/minnow/build
    Start 1: compile with bug-checkers
1/2 Test #1: compile with bug-checkers ........   Passed    0.96 sec
    Start 2: t_webget
2/2 Test #2: t_webget .........................   Passed    1.35 sec

100% tests passed, 0 tests failed out of 2

Total Test time (real) =   2.32 sec
Built target check_webget
lefty777@EDVAC-2023:~/minnow$
```

## 4.2 Analyzing Data

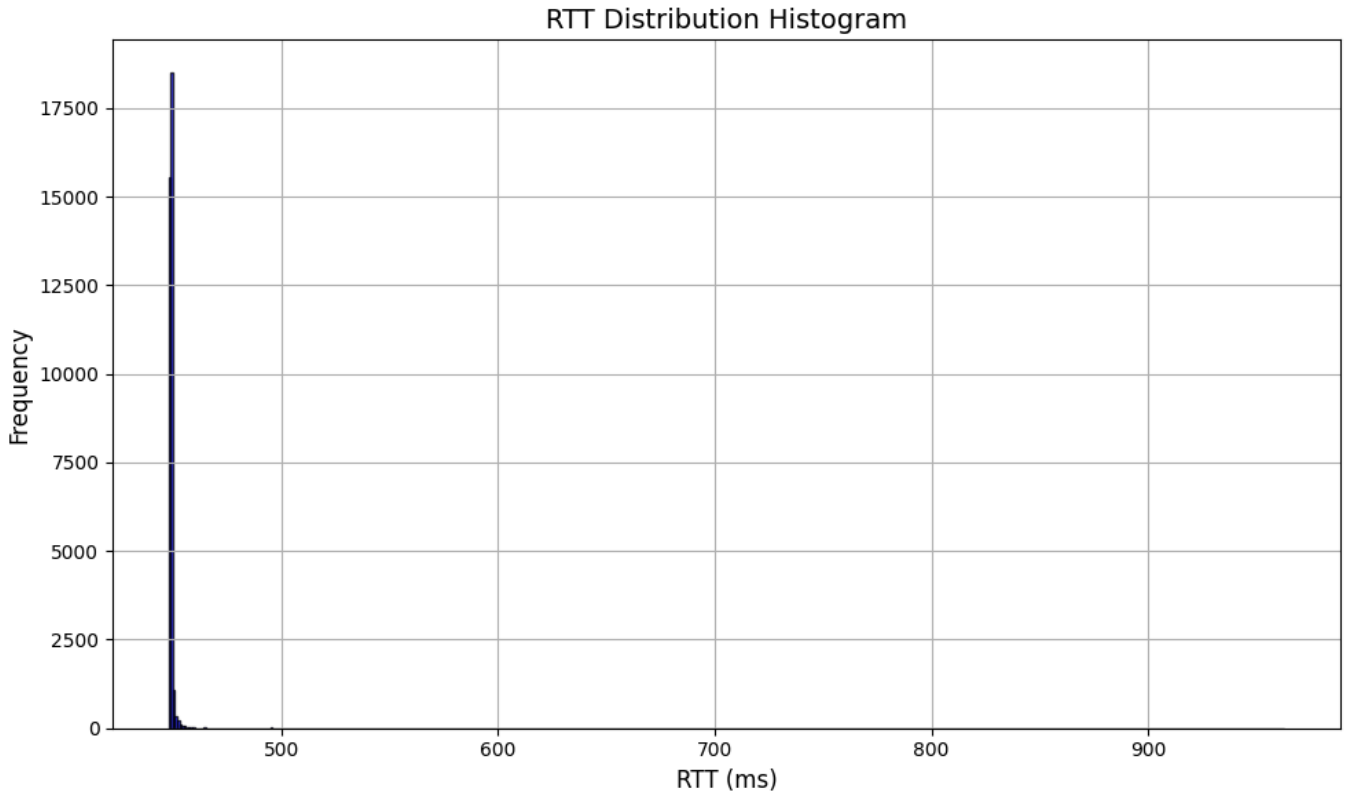
1. **What was the overall delivery rate over the entire interval? In other words: how many echo replies were received, divided by how many echo requests were sent? (Note: ping on GNU/Linux doesn't print any message about echo replies that are not received. You'll**

**have to identify missing replies by looking for missing sequence numbers.)**
0.9994

2. **What was the longest consecutive string of successful pings (all replied-to in a row)?**
   9218
3. **What was the longest burst of losses (all not replied-to in a row)?**
   3
4. **How independent or correlated is the event of “packet loss” over time? In other words:**
   - **Given that echo request #N received a reply, what is the probability that echo request #(N+1) was also successfully replied-to?**
     0.9996
   - **Given that echo request #N did not receive a reply, what is the probability that echo request #(N+1) was successfully replied-to?**
     0.5714
   - **How do these figures (the conditional delivery rates) compare with the overall “unconditional” packet delivery rate in the first question? How independent or bursty were the losses?**
     Apparently, the conditional delivery rates are lower than the overall delivery rate, which means that the packet loss is not independent and bursty, as we can see, if a packet is lost, the probability of the next packet being successfully delivered will drop from 0.9996 to 0.5714, which is a huge decrease.
5. **What was the minimum RTT seen over the entire interval? (This is probably a reasonable approximation of the true MinRTT...)**
   448.0ms
6. **What was the maximum RTT seen over the entire interval?**
   963.0ms
7. **Make a graph of the RTT as a function of time. Label the x-axis with the actual time of day (covering the 2+-hour period), and the y-axis should be the number of milliseconds of RTT.**

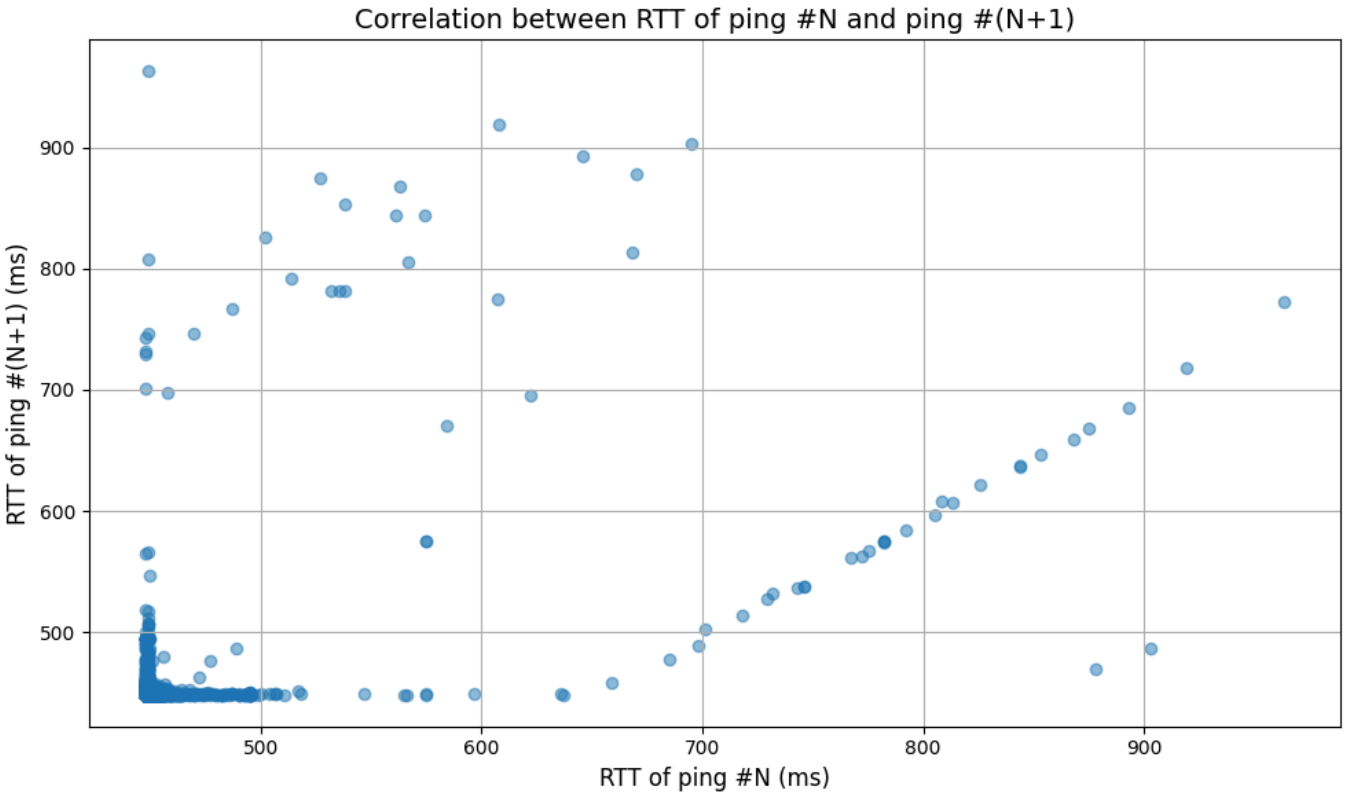



8. **Make a histogram or Cumulative Distribution Function of the distribution of RTTs observed. What rough shape is the distribution?**




The distribution converges to somewhere near 300ms, as most of the RTTs are around 300ms.

9. **make a graph of the correlation between “RTT of ping #N” and “RTT of ping #N+1”. The x-axis should be the number of milliseconds from the first RTT, and the y-axis should be the number of milliseconds from the second RTT. How correlated is the RTT over time?**




As we can see, the relation between RTT of #N and RTT of #N+1 is almost positive linear, which means that RTT is correlated over time.

10. **What are your conclusions from the data? Did the network path behave the way you were expecting? What (if anything) surprised you from looking at the graphs and summary statistics?**

    My conclusion is that the network path failed at a rather small probability, as it will successfully transmit most packets to its destination and get the reply. Another thing is that the successful delivery of packets is correlated, which means that the result of the #N+1 delivery tend to be corresponded to the previous. The most surprising thing is that the RTT is almost linearly correlated over time, which means that the network path is stable and the RTT is not fluctuating too much.