# Report for Computer Networks Practicum Lab3

BoEr Zhuang & Kenuo Xu

## Interface Scheduler by Kenuo Xu

First try to find out why RR Scheduler failed. Interestingly, RR Scheduler will choose the first interface most of the time, so when this interface fails, RR Scheduler will send almost no data. Looks like it is not an ideal RR scheduler, and recording data within the Scheduler is not a good idea.

The first two schedulers implemented are Low RTT Scheduler (see `include/LowRTTScheduler.hh`) and High Throughput Scheduler (see `include/HighThroughputScheduler.hh`). They both traverse one attribute from each interface's TCPBasicMetricValue, then find the relatively-best interface based on the attribute. Both of them work well, though Low-RTT performs a little better.

In order to combine the two attributes, a few experiments have been conducted, leading to Final Scheduler (see `include/FinalScheduler.hh`). The best method is to take $bw = \frac{throughput}{RTT}$ as the attribute, and choose the interface with the highest $bw$, which is similar to the BBR congestion control algorithm. One trick is that it seems that the influence of throughput is little when all the interfaces having a low throughput. So, in my implementation, a bias is added to the throughput. For example, consider a condition where

interface 1: RTT=10, throughput=30;
interface 2: RTT=20, throughput=70;

When no bias is added, interface 2 will be chosen. But if a bias of 50 is added to the throughput, then $bw_1 = \frac{30+50}{10} = 8$, while $bw_2 = \frac{70+50}{20} = 6$, so it is interface 1 that is to be chosen. From a few tests, the bias around 100 may results the best.

Another thing that needs mentioning is that in `PolicyManager.cc`, I simply change the scheduler for all the connections, because it seems that all the connections have a given priority of MEDIUM when no special work is done.

## Connection Set Scheduler by BoEr Zhuang

My implement is based on a simple idea: schedule a connection which have more packet to deal with. So, I first implement a greedy algorithm (see `GreedyConnectionSetScheduler`). It chooses the connection with the most unprocessed packages. However, due to the problem with the test program, I can't compare its performance with RR Scheduler or eval its performance bottleneck. So, I list several weaknesses and show my solution.

1. Starvation. This policy is good for high-throughput connections. Low throughput connections may get stuck in starvation. To prevent it, we prioritize connections that have not been processed for a while. On this way, we can also reduce its transfer delay. In my implement, I used previous schedule time as an indicator. (see `GreedyConnectionSetScheduler`)

2. The O(n) scheduler. In this algorithm, each schedule needs O(n) time to find the best connection. This may become a bottleneck. So, I set a threshold to reduce its complexity. After schedule all the connections upon the threshold, update all the connection. I also set a threshold dynamically like BBR algorithm. (see `ThresheldConnectionSetScheduler`)