ICS II Transhub Lab 报告

中国人民大学 李修羽

1 任务目标

于 `controller.cc` 中实现模拟拥塞控制算法，处理数据报文的发送与 ACK 的接受。

`sender.cc` 采用 QUIC 协议，发送若干次报文直到达到 $cwnd$，随后阻塞直到收到 ACK 或超时，此后通过 `controller.cc` 传回的 $cwnd$ 进行新一轮循环。

2 基本思路

2.1 算法实现

采用 BBR 算法进行拥塞控制，并对原有的参数进行调整：

State	pacing_gain	cwnd_gain
START_UP	2.0	3
DRAIN	1 / 2.0	1
PROBE_BW	[1.25, 0.75, 1, 1, 1, 1]	2
PROBE_RTT	1	cwnd = 4

由于 `sender.cc` 是单线程阻塞式的逻辑，无法直接通过记录下一次发送时间来控制发送报文速率。笔者采用延后发送的机制，记录上一次发送报文的时间，在下一次发送报文时，将间隔时间内应发送的报文全部一起发送。

在 0 时刻，令 $cwnd = 10$，使得阻塞一个 RTT 后，两次收到 ACK 的时间快速接近单向时延，而报文的发送速率通过 $\frac{1}{pacing\_gain \times BtlBw}$ 来计算。

每一次进入 START_UP 阶段，令 $RTT = 250, BtlBw = 0.05$，RTT 采用优先队列取近 10 秒内的最小值，$BtlBw$ 则取至下一次 START_UP 前的最大值，采用 $RTT \times BtlBw$ 计算 BDP。

计算 BtlBw 时，由于 $1ms$ 内接收端收到的可能不止一个报文，而时间戳精度只到毫秒，为了正确估计此时的 bw，采用取前 len 次报文的平均传输时间作为 bw 的估计值，len

的值采用当前 $BtlBw$ 来拟合：

$$len = 10 + 1.19^{10 \times BtlBw}$$

同时对于由接收端卡顿导致 bw 过大的情况，对其上界作限制（$bw \leq 6.5$），以维持发送速率并降低时延波动。

由于测试时带宽上限波动较快，`START_UP` 等待的 $3 \times RTT$ 改为 $\frac{3}{BtlBw}$，`PROBE_BW` 等待的 $3 \times RTT$ 改为 $1.5 \times RTT$，而 `PROBE_RTT` 的时限设为 $20ms$。

2.2 具体实现

2.2.1 `window_size`

维护 $sent$ 记录已发送的报文数，$cwnd$ 为待发送的报文数，返回 $sent + cwnd$。

2.2.2 `timeout_ms`

由于 `sender.cc` 采用的是 QUIC 协议，重传的报文编号依然正常累加，此报文会被当做新报文纳入计算，故超时值对丢包率的影响不大。

实测发现，改变超时值对实验结果无明显影响，此处设超时值为 $1.5 \times RTT$，丢包率约为 6%。

2.2.3 `datagram_was_sent`

更新 $sent, cwnd$，计算当前 bdp，若 $sent \geq bdp$ 则直接等待下一次 ACK，否则检测是否为 `DRAIN` 状态，更新为 `PROBW_BW` 状态。

由于 `sender.cc` 是直接采用 `while (sent <= cwnd)` 的方式来循环，此处若 `cwnd` 较大，发送速率较高。笔者用 100 次无用循环来达到 μs 量级的 `sleep()`，在降低 15% 吞吐量的同时，降低了 20% 以上的时延，起优化作用。

```
if (sent >= bdp)
    return;
for (int i = 1; i <= 100; i++)
    nextSendTime = timestamp_ms() + 1.0 / (pacing_gain * BtlBw);
if (state == DRAIN) {
    ...
}
```

2.2.4 `ack_received`

更新 $sent$，计算并更新 $RTT, BtlBw$，通过上次发送时间计算当前应发送的报文数

$$cwnd = \frac{time - lastTime}{pacing\_gain \times BtlBw}$$

随后检测并更新 BBR 状态即可。

3 实验结果

评分 1 为 4.9009，评分 2 为 4.85953，总评分 4.87，排名第 4。

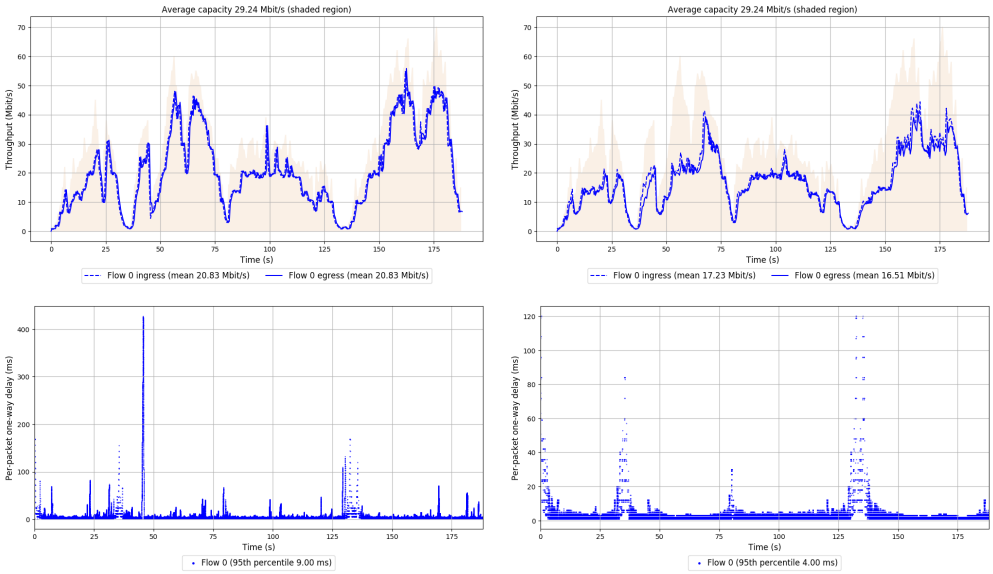

图 1: 性能图