QUIC pacing

Willy Tarreau – Amaury Denoyelle
HAProxy

HTTP Workshop 2024

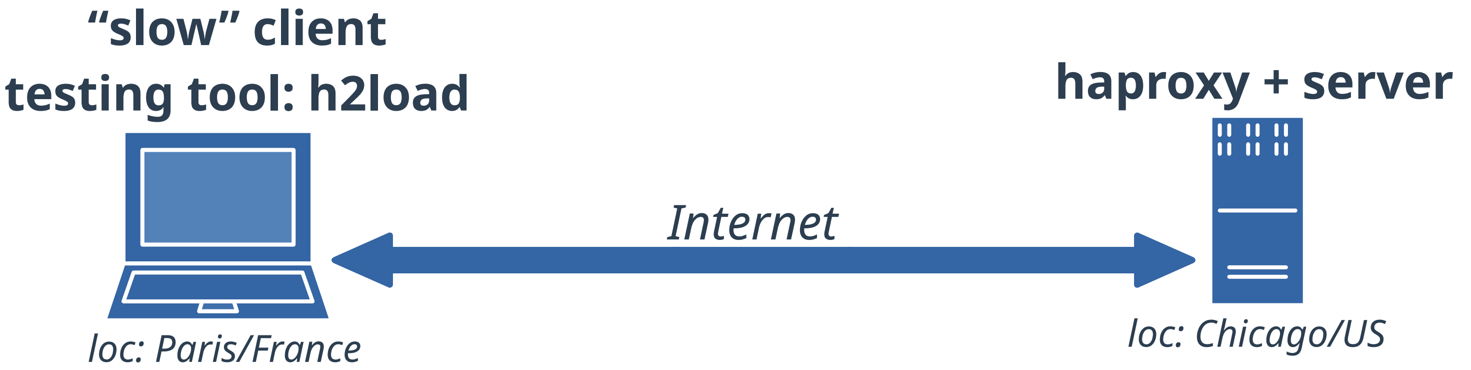
Background

- QUIC implemented into haproxy since 2.6 (2022-05)
- Mostly full featured now
- Tested against multiple implementations https://interop.seemann.io/

Benchmarks

- Many benchmarks performed during our development
- One recurring factor : CUBIC intolerance to loss
- Big impact on the throughput depending on the client CPU, highlighted by using a big.LITTLE architecture

Benchmark environment



“slow” client

testing tool: h2load

haproxy + server

Internet

loc: Paris/France

loc: Chicago/US

Without pacing

- Client app read too slow
- Frequent drops due to small client rxbuf

```
time        ikb  ipk  okb     okp
1730708836 28.3 28.8 18023.7 1924.4
1730708837 38.7 51.1 12677.6 1360.0
1730708838 35.1 56.6  8956.4  972.2
1730708839 28.1 47.7  9736.0 1048.8
1730708840 29.2 49.9  8355.4  911.1
1730708841 39.6 65.5  9307.5 1012.2
1730708842 38.2 64.4 10335.1 1124.4
1730708843 25.7 43.3  8842.2  955.5
1730708844 28.5 47.7  9026.4  970.0
1730708845 28.0 47.7 10602.4 1150.0
1730708846 14.8 25.5  8743.9  936.6
1730708847  8.8 15.5  8772.0  940.0
1730708848 21.5 23.3  9753.4 1046.6
1730708849 22.3 35.5  8538.4  923.3
1730708850 35.2 59.9  8339.1  903.3
```

First pacing implementation : ns resolution

- Fixed sleep between each STREAM datagram emission
- Then, use window and RTT into account
- Nanosecond resolution with active wait

```
time         ikb    ipk    okb      okp
1730709700  808.3 1256.6 210095.3 22226.6
1730709701 1090.7 1738.8 179771.7 19181.1
1730709702 1231.1 1974.4 180006.7 19204.4
1730709703 1209.9 1940.0 179803.5 19194.4
1730709704 1195.1 1917.7 198519.8 21206.6
1730709705 1525.4 2436.6 204119.9 21836.6
1730709706 1584.7 2541.1 205749.4 21998.8
1730709707 1547.2 2481.1 201117.4 21438.8
1730709708 1585.0 2542.2 185062.4 19736.6
1730709709 1402.3 2250.0 206563.2 22035.5
1730709710 1650.4 2649.9 207627.8 22157.7
1730709711 1572.2 2522.2 205548.3 21945.5
1730709712 1535.2 2463.3 185206.0 19766.6
1730709713 1453.6 2331.1 202223.4 21565.5
1730709714 1449.6 2327.7 204784.8 21834.4
```

Second implementation : ms resolution

- ms resolution with passive wait
- Better performance than without pacing
- Resolution not precise enough to reach ns resolution

```
time       ikb   ipk   okb      okp
1730712899 109.6 183.3  31352.2 3237.7
1730712900 438.2 719.9 148113.3 15508.8
1730712901 356.1 516.6 130484.2 13675.5
1730712902 362.4 582.2 143155.0 15010.0
1730712903 356.7 574.4 173882.4 18229.9
1730712904 463.4 743.3 153107.8 16053.3
1730712905 308.5 495.5 119343.1 12518.8
1730712906 209.8 337.7 120889.0 12672.2
1730712907 366.4 588.8 148989.2 15622.2
1730712908 249.3 400.0 122228.4 12816.6
1730712909 319.6 513.3 152392.4 15979.9
1730712910 291.3 468.8 128052.7 13415.5
1730712911 230.0 370.0 130899.8 13716.6
1730712912 254.4 409.9 122859.2 12873.3
1730712913 422.0 677.7 152305.6 15955.5
```

Conclusion

- Big impact from max rcv client buffer size SO_RCVBUF / SO_RCVBUFFORCE
- On server side pacing is a must-have implementation may be difficult though and contradictory with other optimizations such as GSO
- SO_TXTIME as an alternative but only with fq qdisc