Implementing TCP-CUBIC in NDN

Shuo Yang

TCP-CUBIC Overview

TCP-CUBIC is an improvement over TCP-BIC in the way that while retaining most of strengths of BIC (especially, its stability and scalability under high BDP networks), it simplifies the window control function, enhances its TCP friendliness and achieves better RTT fairness. CUBIC also performs better under small BDP networks while BIC can still be too aggressive for TCP under short RTT or small BDP networks.

The window growth function of CUBIC is a function of elapsed time since the last packet loss event:

$$W(t) = C(t-K)^3 + W_{max}$$
$$K = \sqrt[3]{\frac{W_{max}\beta}{C}}$$

C is a constant scaling factor with default value of 0.4, β is the multiplicative decrease factor after a packet loss event, its default value is 0.2, and W_{max} is the window size just before the last window reduction. Here K represents the time period needed to increase W to W_{max}.

CUBIC Algorithm

Algorithm 1 TCP CUBIC algorithm for NDN

procedure INIT() ▷ CUBIC initialization
    $tcp\_friendliness \leftarrow$ true, $\beta \leftarrow 0.2$
    $fast\_convergence \leftarrow$ true, $C \leftarrow 0.4$
    $CubicReset()$

procedure ONDATA() ▷ Data packets act as selective acknowledgement of Interest packets
    if min_rtt **then** $min\_rtt \leftarrow min(min\_rtt, RTT)$
    else $min\_rtt \leftarrow RTT$
    if $cwnd \leq ssthresh$ **then**
        $cwnd \leftarrow cwnd + 1$ ▷ slow start
    else
        $CubicUpdate()$ ▷ congestion avoidance

procedure ONPACKETLOSS() ▷ Indicated either by timeouts or NACKs
    $epoch\_start \leftarrow 0$
    if $cwnd < W_{last\_max}$ & $fast\_convergence$ **then**
        $W_{last\_max} \leftarrow cwnd * \frac{2-\beta}{2}$ ▷ release more bandwidth for new flows to catch up
    else
        $W_{last\_max} \leftarrow cwnd$
    $cwnd \leftarrow cwnd * (1 - \beta)$
    $ssthresh \leftarrow cwnd$

procedure CUBICUPDATE() ▷ Update congestion window
    if $epoch\_start \leq 0$ **then**
        $epoch\_start \leftarrow current\_time$
        if $cwnd < W_{last\_max}$ **then**
            $K \leftarrow \sqrt[3]{\frac{W_{last\_max} - cwnd}{C}}$ ▷ k is in second
            $origin\_point \leftarrow W_{last\_max}$
        else
            $K \leftarrow 0$
            $origin\_point \leftarrow cwnd$
        $W_{tcp} \leftarrow cwnd$
    $t \leftarrow current\_time + min\_rtt - epoch\_start$ ▷ t is in second
    $target \leftarrow origin\_point + C(t - K)^3$ ▷ calculate $W(t + rtt)$
    if $target > cwnd$ **then**
        $cwnd\_update \leftarrow cwnd + \frac{target - cwnd}{cwnd}$
    else
        $cwnd\_update \leftarrow cwnd + \frac{0.01}{cwnd}$ ▷ only a small increment
    if $tcp\_friendliness$ **then** ▷ make sure window grows at least at the speed of TCP
        $W_{tcp} \leftarrow W_{tcp} + \frac{3\beta}{2-\beta} * \frac{t}{RTT}$
        if $W_{tcp} > cwnd$ & $W_{tcp} > target$ **then**
            $cwnd\_update \leftarrow cwnd + \frac{W_{tcp} - cwnd}{cwnd}$
    $cwnd \leftarrow cwnd\_update$ ▷ update window size

procedure CUBICRESET() ▷ Reset state variables
    $W_{last\_max} \leftarrow 0$, $epoch\_start \leftarrow 0$, $origin\_point \leftarrow 0$
    $min\_rtt \leftarrow 0$ $W_{tcp} \leftarrow 0$, $K \leftarrow 0$, $ssthresh \leftarrow MAX\_INT$

Implementation Challenges

The implementation we can refer to is the one implemented in Linux kernel (http://lxr.linux.no/linux/net/ipv4/tcp_cubic.c). It's quite hard to get because it uses a bunch of scaling factors and performs complex unit conversions. I think it's because Linux kernel doesn't allow floating point operations. Since we are going to implement CUBIC in user mode, it's not a problem. But the concern is that floating point operations may introduce numerical stability. At the bottom line, we can still mimic the kernel's implementation by using scaling factors.

Test Scenarios

Topology	Bottleneck Link Bandwidth	Flows	Purposes
Linear topology	5Mbps	single CUBIC flow	how CUBIC performs under low BDP network
Linear topology	100Mbps	single CUBIC flow	how CUBIC performs under high BDP network
Dumbbell topology	100Mbps	two CUBIC flows	see how fast two flow converges
Dumbbell topology	100Mbps	one CUBIC flow and one TCP-Reno flow	test intra-protocol fairness

References

Original CUBIC paper: http://netsrv.csc.ncsu.edu/export/cubic_a_new_tcp_2008.pdf
CUBIC implementation in ns-3: http://web.cs.wpi.edu/~rek/Research/Papers/WNS3_14.pdf
CUBIC implementation in Linux kernel: http://lxr.linux.no/linux/net/ipv4/tcp_cubic.c