Hard Limit CoDel: An Extension of the CoDel Algorithm

Fengyu Gao, Hongyan Qian, and Xiaoling Yang

***Abstract*—The CoDel algorithm has been a significant advance in the field of AQM. However, it does not provide bounded delay. Rather than allowing bursts of traffic, we argue that a hard limit is necessary especially at the edge of the Internet where a single flow can congest a link. Thus in this paper we proposed an extension of the CoDel algorithm called "hard limit CoDel", which is fairly simple but effective. Instead of number of packets, this extension uses the packet sojourn time as the metric of limit. Simulation experiments showed that the maximum delay and jitter have been well controlled with an acceptable loss on throughput. Its performance is especially excellent with changing link rates.**

***Index Terms*—bufferbloat, CoDel, AQM.**

I. Introduction

The CoDel algorithm proposed by Kathleen Nichols and Van Jacobson in 2012 has been a great innovation in AQM. It's a "no-knobs" AQM that adapts to changing link rates, and is likely to be deployed across the Internet in several years after rigorous testing and analysis.

However, CoDel is designed to be an algorithm that allows bursts of traffic while controlling average queue length. When configured with a large buffer, it is suitable for backbone links, but not for the edge of the Internet where the link could be bottlenecked by a single flow. In a lot of occasions, e.g., when performing stock trading, playing online games, or playing music, jitter and maximum delay would become a major concern of end users that the default CoDel algorithm does not guarantees.

At the edge of the Internet, CoDel should be configured with a buffer which is much smaller, and when the link rate changes, the buffer size should also change accordingly.

Thus in this paper, we proposed an extension of the CoDel algorithm called "Hard Limit CoDel", which is specially designed for links at the edge of the Internet where a small and adaptive hard limit is necessary. "Hard Limit CoDel" has just one parameter: the max delay (specified in milliseconds). We have implemented the algorithm both in the ns-2 simulator and in real embedded Linux platforms. Experiment results show that the maximum delay could be well controlled without much loss on throughput in most circumstances. In particular, this extension brings about much better performance when the link rate changes significantly.

Manuscript received September 26, 2014.

Fengyu Gao, Hongyan Qian and Xiaoling Yang are with the Department of Computer Science and Technology, Nanjing University of Aeronautics and Astronautics (NUAA), P.R. China (e-mail: feng32@163.com; qhy98@nuaa.edu.cn; yxl_9009@163.com).

II. Background Information

Nowadays the Internet is suffering from high latency and jitter caused by unprotected large buffers. The "persistently full buffer problem", recently exposed as "bufferbloat" [1], [2], has been observed for decades, but is still with us.

When recognized the problem, Van Jacobson and Sally Floyd developed a simple and effective algorithm called RED (Random Early Detection) in 1993 [3]. In 1998, the Internet Research Task Force urged the deployment of active queue management in the Internet [4], specially recommending RED.

However, this algorithm has never been widely deployed because of implementation difficulties. The performance of RED depends heavily on the appropriate setting of at least four parameters:

- Minimum threshold
- Maximum threshold
- Exponential weighted moving average constant
- Maximum drop probability

As little guidance was available to set its parameters, RED functions poorly in a number of cases, which led to a general reluctance to use it. A number of variations and imitations have been proposed since then, most of which focus on creating an adaptive algorithm that is not so sensitive to the initial parameters. Although research continued, deployment did not.

Subsequent research succeeded to fix some of the flaws in the original RED but failed to create an AQM that could keep persistent buffers short without overdropping, until the proposal of the "CoDel" algorithm 2012 [5]. In that paper it is found that when the minimum packet sojourn time in a specific interval is used as a measure of bad queue, it can be efficiently controlled while still permitting bursts of traffic. The authors have selected the best values for delay control - 5ms for packet sojourn time and 100ms for the interval. Simulation results showed that under the suggested setting, this algorithm works well across a wide range of bandwidths, RTTs, and traffic loads with high utilization. As the results are so compelling, it becomes unnecessary to further tune the parameters.

The ns-2 model of CoDel, the novel "no-knobs" AQM was ported to the Linux kernel quickly, and a variant called "fq_codel" has also been implemented, which combines stochastic fair queuing (SFQ) and CoDel to further improve performance. In fq_codel, different flows are hashed into a number of queues; each queue is managed separately by the

ISBN: 978-988-19253-9-8
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

CoDel algorithm, and different queues take turns to send packets.

III. CoDel At The Edge of the Internet

As stated above, CoDel is proposed as an adaptive algorithm that works for different RTTs, changing traffic loads and link rates. It can be deployed in a number of network scenarios including backbone links in the Internet, broadband access servers of the ISPs, and home wireless networks in everyone's house.

However, can CoDel guarantee the expected quality of service of end users? When the ISP configures a large buffer managed by CoDel, though persistent queues are controlled, allowing bursts of traffic means the queuing delay may be very high in some short periods. The result is, though average delay is controlled, maximum delay and jitter are not.

High delay and jitter affect a lot of applications of the Internet including stock trading, online games, VoIP, etc. For tomorrow's virtual reality applications, the requirement on latency is even higher: it should be kept under 30ms so that something feels attached to your hand.

Therefore, at the edge of the Internet, the last bottleneck along the path, more restriction should be introduced so that maximum delay and jitter can be better controlled [1].

In the ns-2 and Linux kernel implementation of CoDel, users can specify the buffer size by setting the limit parameter, which is the maximum number of packets in queue. Since a buffer of 1000 packets is often too large for most users, does it make sense to decrease the buffer size at the Internet edge? Unfortunately, though it works for a link with a static bandwidth, it does not for changing link rates. In a wireless network where the signal strength varies, or when the ISP has a "boost" feature that raise the bandwidth by 5 times for a few seconds when the link gets out of idle state, the maximum delay might be 5 times larger than usual when the bandwidth finally falls back.

As a result, number of packets is not a good measure of buffer size. To provide consistent user experience (maximum delay and jitter) at the edge of the Internet, packet sojourn instead of number of packets should be used as the metric of "limit" - and that's all about our extension named hard limit CoDel.

Hard limit CoDel has just one parameter: the max packet sojourn time (max delay). The default value of the parameter is 50ms, which is selected according to the demand of end users. Most of today's Internet applications can run smoothly when the maximum delay is controlled in 50ms. But after all, this value should be configurable and exposed to end users.

The pseudo code of hard limit CoDel is also very simple, with only a few lines added to the original algorithm, as shown in Figure 1. We believe that this extension of CoDel can also be efficiently implemented in silicon.

[1] Although we just focus on the edge of the Internet in this paper, it does not necessarily make sense even for backbone links when packets are early dropped according to one criterion (the packet sojourn time), and dropped according to another (the number of packets) when the limit is reached.

```
typedef struct {
    packet_t* p;
    flag_t ok_to_drop;
+   flag_t hard_drop;
} dodeque_result;

dodeque_result codel_queue_t::dodeque(time_t now)
{
    dodeque_result r = { queue::deque(), 0, 0 };
    if (r.p == NULL) {
        first_above_time = 0;
    } else {
        time_t sojourn_time = now - r.p->tstamp;
        if (sojourn_time < target ||
            bytes() < maxpacket) {
            first_above_time = 0;
        } else {
            if (first_above_time == 0) {
                first_above_time = now + interval;
            } else if (now >= first_above_time) {
                r.ok_to_drop = 1;
            }
        }
+       if (sojourn_time > max_delay) {
+           r.ok_to_drop = 1;
+           r.hard_drop = 1;
+       }
    }
    return r;
}

packet_t* codel_queue_t::deque()
{
    time_t now = clock();
    dodeque_result r;
+   do {
+       r = dodeque();
+   } while (r.hard_drop && drop(r.p));
    if (r.p == NULL) {
        dropping = 0;
        return r.p;
    }
    if (dropping) {
        if (! r.ok_to_drop) {
            dropping = 0;
        } else if (now >= drop_next) {
            while (now >= drop_next && dropping) {
                drop(r.p);
                ++count;
                r = dodeque();
                if (! r.ok_to_drop)
                    dropping = 0;
                else
                    drop_next = control_law(drop_next);
            }
        }
    } else if (r.ok_to_drop &&
        ((now - drop_next < interval) ||
         (now - first_above_time >= interval))) {
        drop(r.p);
        r = dodeque();
        dropping = 1;
        if (now - drop_next < interval)
            count = count>2? count-2 : 1;
        else
            count = 1;
        drop_next = control_law(now);
    }
    return (r.p);
}
```

Fig. 1. The pseudo code of hard limit CoDel

ISBN: 978-988-19253-9-8
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

IV. Simulation Analysis

We used the typical dumbbell topology with a single congested link in our simulations, as shown in Figure 2.

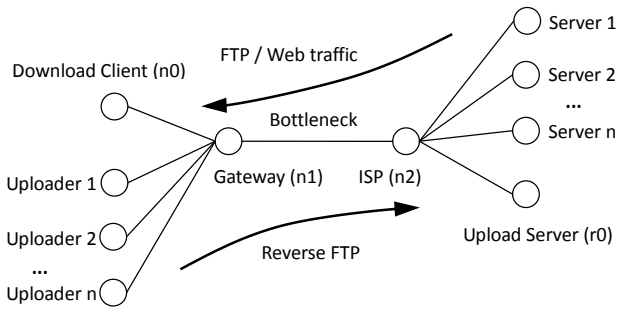

Fig. 2. The network scenario

As a new hard limit is introduced, the average and maximum delay in different network scenarios are expected to be lower, and the throughput also lower. To see what exactly the performance would be, we've written a few scripts and run thousands of simulations, with bottleneck bandwidth from 1 Mbps to 100 Mbps, round-trip-time from 10ms to 500ms, simultaneous bulk data transfer from 1 to 64, PackMime web traffic intensity form 0 to 80, and reverse bulk data transfers from 0 to 4. Each experiment is executed three times, one for the original CoDel algorithm with a limit of 1000 packets, one for hard limit CoDel with a max delay of 50ms, and the last for a FIFO queue which is 50ms long. TCP New Reno is used as the congestion control algorithm.

Just as the name implies, the most important feature of hard limit CoDel is that theres a "hard limit" of queuing delay. The default value of the hard limit is 50ms. However, as the original CoDel is designed to be an algorithm that allows bursts of traffic while controlling bad queues, the default limit in both the ns-2 and the Linux kernel implementation is quite large.

As a result, the maximum delay and jitter are not controlled when therere a few bulk data transfers. Figure 3 shows what exactly the maximum delay would be under different level of congestion.

When the number of ftp flows increases, the maximum delay also increases linearly. In particular, the delay may be easily raised above 200ms when there are only 2 to 4 flows in low and medium bandwidth (below 16 Mbps) environments. If the ISP has configured a CoDel algorithm with a buffer of 1000 packets, such jitter may be observed frequently when the users are surfing the Internet, as todays web pages contain more and more big pictures. When running P2P applications, it may be even worse.

The most significant advance of the original CoDel is the effective control of bad queues and average delay. However, as more packets are dropped in hard limit CoDel, its average delay is even lower.

Figures 4 to 7 present the average delay under different bandwidths and level of congestion, and there are several noteworthy points:

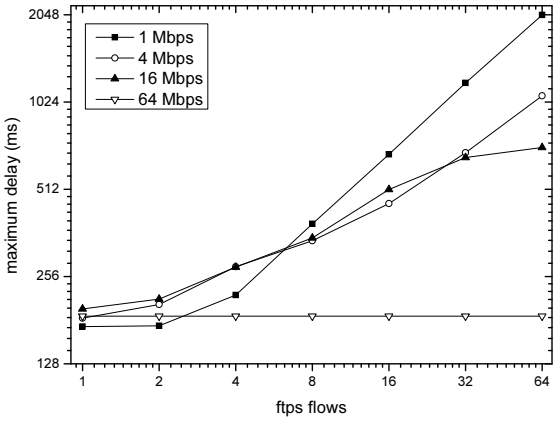

Fig. 3. The maximum delay of CoDel under different level of congestion

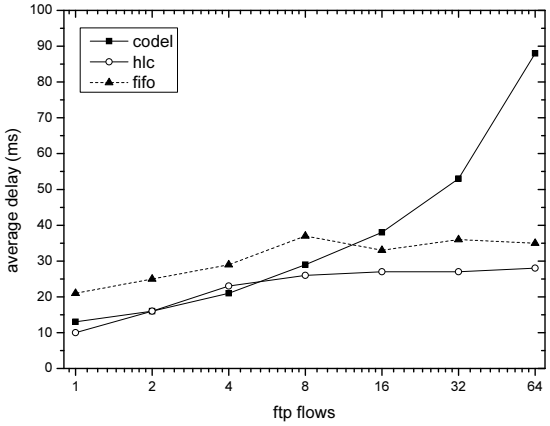

Fig. 4. The average delay under bandwidth 1 Mbps

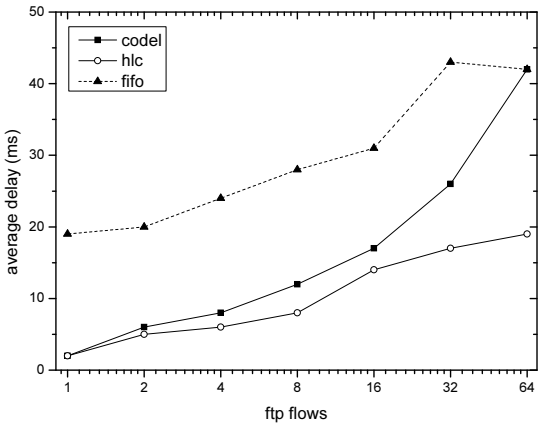

Fig. 5. The average delay under bandwidth 4 Mbps

- The average delay of CoDel has been well controlled. It is above 50ms only when the bandwidth is limited (1 Mbps) and there are a lot of bulk data transfers.
- Under high bandwidth, the average queue length of the FIFO queue is the longest among the three. So instead of an unprotected large buffer, we get an "unprotected small buffer".

We used the canonical RTT value 100ms in the previous

ISBN: 978-988-19253-9-8
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

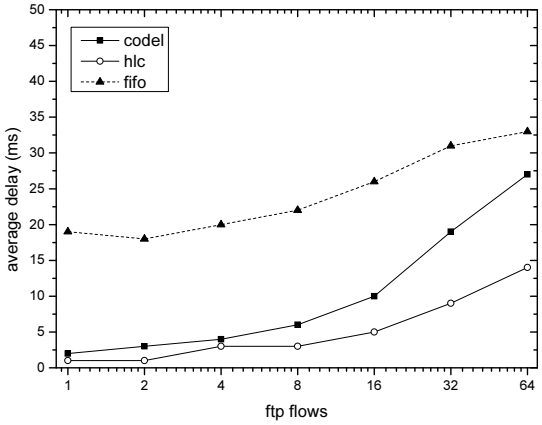

Fig. 6. The average delay under bandwidth 16 Mbps

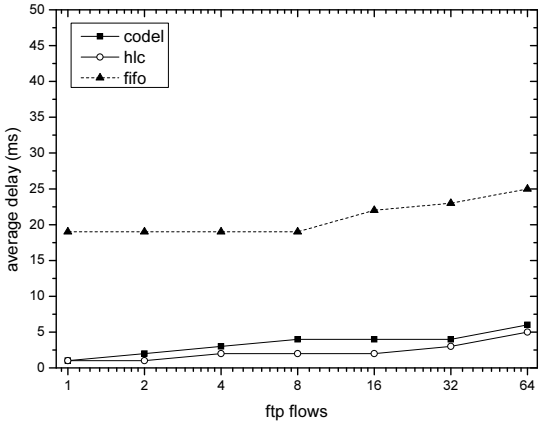

Fig. 7. The average delay under bandwidth 64 Mbps

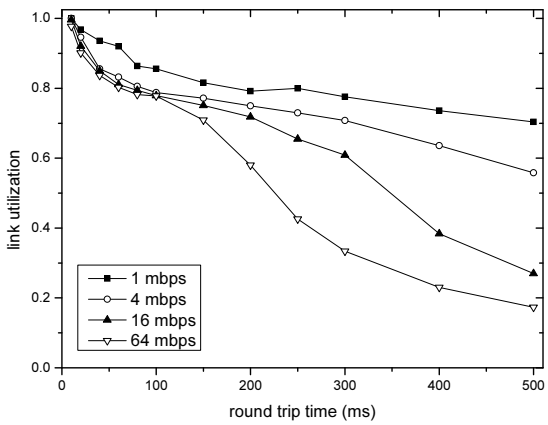

Fig. 8. The link utilization of CoDel with 1 ftp

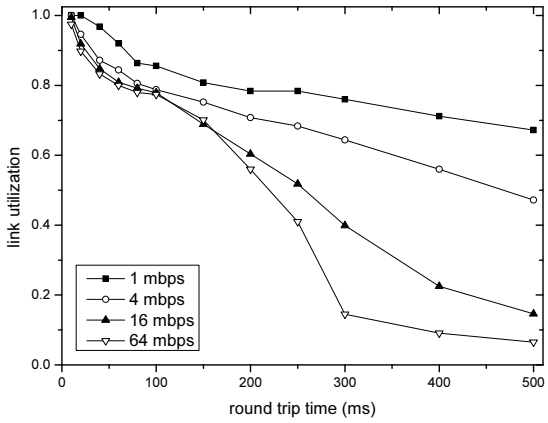

Fig. 9. The link utilization of hard limit CoDel with 1 ftp

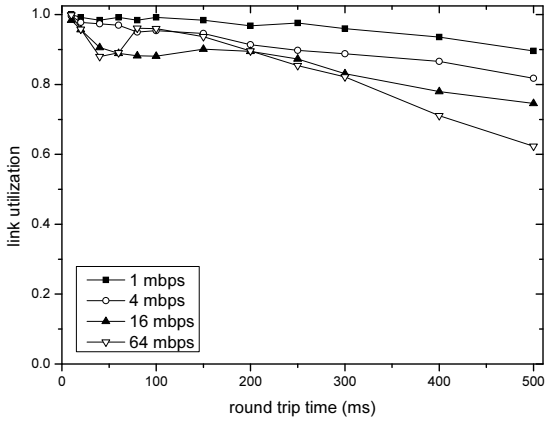

Fig. 10. The link utilization of CoDel with 8 ftp

experiments. Smaller and larger RTTs have also been tested. Smaller RTTs generally lead to a bit higher delays, and larger RTTs lead to a bit lower delays. As there is no significant difference, they are not displayed here to save space.

Although the measured delay is apparently lower with hard limit CoDel, it is not achieved at no cost. It is found that the link utilization is affected to some extent, and its most sensitive to the round trip time and the number of bulk transfers.

As presented in Figures 8 to 11, a small RTT leads to higher throughput. A large RTT, on the other hand, leads to lower throughput. When the number of ftp flows increases, link utilization also rises quickly.

Actually, the throughput of hard limit CoDel is significantly lower than the original CoDel only when the RTT is large (500ms), the bandwidth is high (64 Mbps) with only one TCP flow, in which case the throughput is only 4.1 Mbps, 63% lower than that of the original CoDel. Though it may seem to be a significant loss, we argue that it is acceptable because even in the worst case, a bit rate of 4 Mbps is sufficient to support today's 720p videos. The link utilization is much higher when either of the three conditions changes.

When there're some reverse bulk data transfers, the average delay is a lot higher but under 50ms most of the time and the throughput is a bit lower.

When there's a lot of web traffic, the bursts of web traffic may "attack" the long-lived bulk TCP flows, leading to a slightly lower throughput under high bandwidth. When the bandwidth is lower, however, web traffic helps to improve utilization.

As the figures related to reverse ftp and web traffic are not surprising, they are not displayed here.

ISBN: 978-988-19253-9-8
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

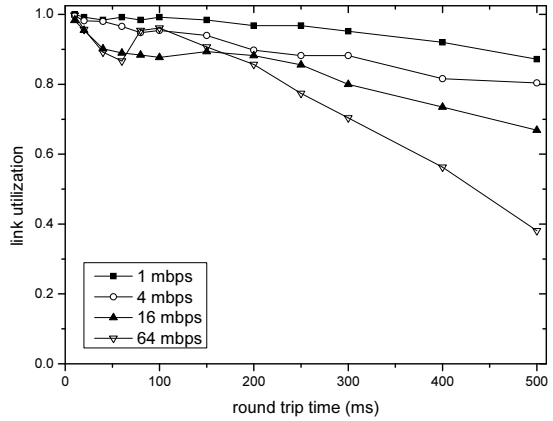

Fig. 11. The link utilization of hard limit CoDel with 8 ftp

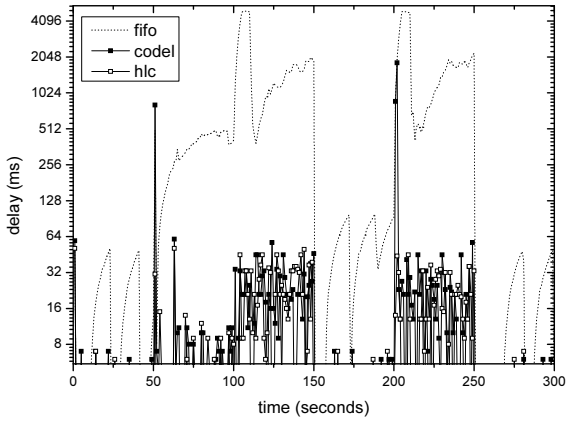

Fig. 12. Measured delay under dynamic bandwidth

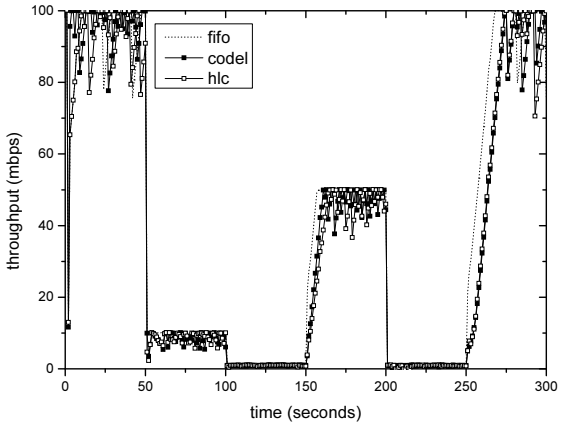

Fig. 13. Measured throughput under dynamic bandwidth

Finally, the network scenario for changing bandwidth has also been examined. We used a load of 4 FTPs and 5 web connections per second to emulate an unstable 100 Mbps wireless link, and the round trip time is 100ms. The link rate is changed every 50 seconds, first dropping to 10 Mbps, then to 1 Mbps, then jumping to 50 Mbps, dropping to 1 Mbps, and finally jumping back to 100 Mbps. The measured delay and throughput are presented in Figure 12 and 13.

Though it is pointed out in [5] that CoDel works much better than tail drop and RED under dynamic bandwidth, the authors did not pay attention to the fact that the maximum delay of CoDel could be rather high (up to 2000ms at the 50 Mbps to 1 Mbps transition). However the maximum delay is well controlled with Hard limit CoDel, with an acceptable (about 2.6%) loss on overall throughput. We have also tested CoDel with a smaller buffer (50ms @ 100Mbps). However the buffer still becomes too large when the bandwidth decreases. As its curve is similar to that of CoDel with a buffer of 1000 packets, it's not displayed here.

We have also implemented hard limit CoDel in real Linux platforms [2], and organized a series of experiments. As the results are similar, they're not display here to save space.

V. Conclusion

Though CoDel has been a fundamental advance in the field of AQM, it does not provide a delay bound which is urgently needed at the edge of the Internet where the link could be bottlenecked by a single flow. The current implementation of CoDel in the ns-2 simulator and the Linux kernel uses the number of packets as the limit, which may cause extremely high latency when the link rate decreases significantly.

As a result, we argue that number of packets is not a good metric of limit. Instead of number of packets, the packet sojourn time should be used as the metric of limit. This modification of CoDel is simple but effective. Simulation experiments showed that delay has been well controlled with an acceptable loss on throughput. It works much better than the original algorithm especially when the bandwidth varies.

Although the extension is called "hard limit CoDel" in this paper, we argue that it should be merged to original CoDel algorithm as a standard option.

Acknowledgment

The authors would like to thank the reviewers for their comments which helped improve the paper.

References

[1] J. Gettys, "Bufferbloat: dark buffers in the internet," *IEEE Internet Computing*, vol. 15, no. 3, pp. 95–96, 2011.
[2] J. Gettys and K. Nichols, "Bufferbloat: dark buffers in the internet," *Communications of the ACM*, vol. 55, no. 1, pp. 57–65, 2012.
[3] S. Floyd and V. Jacobson, "Random early detection gateways for congestion avoidance," *IEEE/ACM Transactions on Networking*, vol. 1, no. 4, pp. 397–413, 1993.
[4] V. Jacobson, B. Braden, S. Floyd, B. Davie, D. Estrin, J. Crowcroft, and S. Deering, "Recommendations on queue management and congestion avoidance in the internet," RFC 2309, 1998.
[5] K. Nichols and V. Jacobson, "Controlling queue delay," *Communications of the ACM*, vol. 55, no. 7, pp. 42–50, 2012.

[2]The OpenWRT patch of hard limit CoDel is available at the svn repository: http://algts.googlecode.com/svn/trunk/utils/hard-limit-codel