TCP LoLa: Congestion Control for Low Latencies and High Throughput

Mario Hock, Felix Neumeister, Martina Zitterbart, Roland Bless
Karlsruhe Institute of Technology
Karlsruhe, Germany
E-Mail: mario.hock@kit.edu, felix.neumeister@student.kit.edu, zitterbart@kit.edu, bless@kit.edu

***Abstract*—TCP LoLa is a new delay-based congestion control that supports both, low queuing delay *and* high network utilization in high speed wide-area networks. This is particularly useful for traffic mixes consisting of bandwidth demanding and delay sensitive flows (e.g., long file transfer and interactive "web 2.0" traffic). TCP LoLa keeps the queuing delay at the bottleneck link low around a fixed target threshold value. This target value is independent from the number of flows sharing the bottleneck. TCP LoLa achieves high link utilization and attains convergence to fairness even among flows with different round-trip times, due to its novel mechanism called *"fair flow balancing"*.**

***Index Terms*—Congestion control**

I. Introduction

Congestion control is an essential mechanism to protect the Internet from severe overload situations. Today's widely used *loss-based* congestion control algorithms, however, adversely affect everyone's performance on the Internet: the inflicted latency may become unnecessarily high. This affects interactive and transaction-based applications (e.g., Voice-over-IP, multiplayer online games and "web 2.0" traffic).

In this paper, we present TCP LoLa, a new delay-based congestion control that *limits queuing delay* up to a fixed target threshold value while striving for a *high utilization* of the bottleneck link. TCP LoLa is, therefore, especially useful for traffic mixes consisting of bandwidth-demanding as well as delay-sensitive applications. Furthermore, TCP LoLa aims at providing flow rate fairness independent of the round-trip times of competing flows. The *convergence to fairness* is particularly challenging for low delay congestion control. If this process is not carefully coordinated among the senders either underutilization or high queuing delays occur. TCP LoLa, therefore, introduces a novel fairness mechanism called *"fair flow balancing"*. Currently TCP LoLa focuses on wired high-speed wide-area networks but is also scalable to lower speeds.

II. Controlling the Amount of In-flight Data

A major challenge of low delay and high throughput congestion control is to carefully control the amount of *in-flight data* (i.e., data that is sent but not yet acknowledged). In order to achieve a full utilization of the bottleneck link without any queuing delay the total amount of in-flight data has to exactly match the bandwidth delay product. If it is too small, the bottleneck link cannot be fully utilized. If it is too large, the excess in-flight data has to be queued. If this situation persists, a *standing queue* [9] builds up.

To control the amount of in-flight data a flow is allowed to have, TCP LoLa uses a *congestion window (CWnd)*. Based on *round-trip time (RTT)* measurements that are already conducted by TCP, each TCP LoLa flow calculates the following estimates (details given in section IV):

- $\widehat{RTT}_{now}$: RTT including the standing queue
- $\widehat{RTT}_{min}$: RTT without any queuing delays
- $\widehat{Q}_{delay}$: Queuing delay caused by the standing queue
- $\widehat{Q}_{data}$: Amount of data *the flow itself* has queued at the bottleneck

To achieve high throughput and low delays TCP LoLa deliberately tries to create a small standing queue and measures the according queuing delay. On the one hand, the existence of a standing queue indicates that the overall amount of in-flight data is sufficient to fully utilize the bottleneck link. On the other hand, the resulting queuing delay gives a detailed congestion signal, since it directly corresponds to the excess in-flight data.

To keep the standing queue low but above *zero*, TCP LoLa introduces two thresholds for $\widehat{Q}_{delay}$: Q_{low} and Q_{target} that are interpreted as follows:

- $\widehat{Q}_{delay} \leq Q_{low}$: No standing queue detected, link is most likely underutilized, thus, CWnd should be increased.
- $Q_{low} < \widehat{Q}_{delay} \leq Q_{target}$: Small standing queue, high throughput and low delay.
- $\widehat{Q}_{delay} > Q_{target}$: Congestion, CWnd has to be reduced.

This will result in a high utilization of the bottleneck link and a low queuing delay, but will not lead to fairness among the flows. Therefore, we developed *fair flow balancing*, a novel mechanism that complements the just described strategy in order to provide a *convergence to fairness* (details given in section III-C).

III. TCP LoLa

As common in congestion control all competing TCP LoLa flows work together to achieve the design goals. If there is a significant amount of non-TCP LoLa traffic in the network, coexistence mechanisms are required, e.g., [7]. Moreover, Q_{low} and Q_{target} have to be identical for all competing flows. Thus, for Internet usage they have to be globally standardized.

Figure 1 shows the main states and state transitions of TCP LoLa, which will be explained in the following.

DOI 10.1109/LCN.2017.42

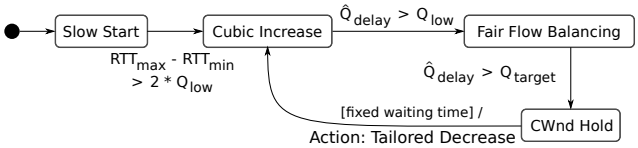


Fig. 1: Congestion control states

A. Slow Start

Similar to other congestion controls TCP LoLa enters the *slow start* state after its initial start or after a retransmission timeout. In addition to that, TCP LoLa keeps track of the minimal and maximal measured RTT. Once the difference is larger than $2 \cdot Q_{low}$, it transits from slow start to cubic increase.

B. Cubic Increase

During cubic increase TCP LoLa uses the same increase function as CUBIC TCP. In contrast to CUBIC TCP, TCP LoLa uses this function only if the potential bottleneck link is most likely not fully utilized, i.e., no standing queue is detected.

$$CWnd(t) = C \cdot (t - K)^3 + CWnd_{max} \quad (1)$$

$CWnd_{max}$: size of CWnd before last reduction, t: time since last window reduction, C: unit-less factor ($C = 0.4$, as in CUBIC TCP). K is recalculated whenever CWnd has to be reduced, details given in section III-E.

C. Fair Flow Balancing

The basic idea behind the RTT-independent flow rate fairness of TCP LoLa is that each flow should keep a low but similar amount of data (X) in the bottleneck queue. This results in correspondingly similar rate shares. To keep the overall queuing delay between the thresholds Q_{low} and Q_{target}, X has to be dynamically scaled to the number of flows and the bottleneck bandwidth. Both values are not known to the end systems. Fair flow balancing, therefore, makes X time-dependent (with $t = 0$ when fair flow balancing is entered; ϕ is a constant):

$$X(t)[Byte] = (\frac{t[ms]}{\phi})^3 \quad (2)$$

CWnd is adapted as described in fig. 2. Flows with a larger rate share typically have a larger $\widehat{Q}_{data}$ than flows with a smaller share. If a flow's rate share is above the fair share, $\widehat{Q}_{data} > X(t)$ applies, thus the flow may not increase its CWnd during fair flow balancing.

if $\widehat{Q}_{data} < X(t)$ **then**
  $CWnd \leftarrow CWnd + (X(t) - \widehat{Q}_{data})$
else
  $CWnd \leftarrow CWnd$ ▷ kept unchanged
end if

Fig. 2: CWnd adjustment during fair flow balancing

Since an appropriate value for X is not known in advance, $X(t)$ is a monotonically increasing function. In order to avoid falling below Q_{low}, CWnd is never decreased during fair flow balancing. An invocation of fair flow balancing, from entering the state until it is left, will be denoted as a *round*.

Fair flow balancing requires that all competing flows enter and leave it at similar points in time. Therefore, the whole TCP LoLa design puts a strong emphasis on synchronized state changes.

D. CWnd Hold

A flow leaves the fair flow balancing state when it detects that $\widehat{Q}_{delay} > Q_{target}$. After that it keeps its CWnd unchanged for a fixed amount of time t_{sync} (default value: 250 ms). This means that this flow will not increase its amount of in-flight data and, thus, will not increase the level of congestion at the bottleneck. The hold time is necessary to ensure that all flows quit the current round of fair flow balancing. After the hold time has elapsed, tailored decrease is performed.

E. Tailored Decrease

Tailored decrease adjusts the CWnd reduction to the amount of congestion: $CWnd \leftarrow (CWnd - \widehat{Q}_{data}) \cdot \gamma$

This means each flow reduces its CWnd by $\widehat{Q}_{data}$ – this *should* already empty the queue – but since TCP LoLa relies on good $\widehat{RTT}_{min}$ values, CWnd is further reduced by the factor $\gamma < 1$ to ensure that the queue will actually be drained completely. To achieve this, K is calculated as follows:

$$K = \sqrt[3]{\left(CWnd_{max} - \widehat{RTT}_{min} \cdot \frac{CWnd_{max}}{\widehat{RTT}_{now}} \cdot \gamma \right) / C} \quad (3)$$

IV. QUEUING DELAY MEASUREMENTS

The size of the standing queue depends on the size of the CWnds. In addition to that bursty traffic or short time cross traffic can increase the queuing delay. To filter out these interfering influences TCP LoLa uses a minimum filter of all measured RTT values within a certain time interval $t_{measure}$:

$$\widehat{RTT}_{now} = \min\{RTT(t_k) | t_k \in [t - t_{measure}, t]\} \quad (4)$$

$RTT(t_k)$ is an individual RTT measurement at time t_k. It is important for the fairness that $t_{measure}$ does not depend on a flow's RTT. Otherwise the filter functions of senders with different RTTs would have different granularities. Based on $\widehat{RTT}_{now}$ the following values are calculated:

$$\widehat{RTT}_{min} \leftarrow \min(\widehat{RTT}_{min}, \widehat{RTT}_{now}) \quad (5)$$

$$\widehat{Q}_{delay} = \widehat{RTT}_{now} - \widehat{RTT}_{min} \quad (6)$$

For the estimation of $\widehat{Q}_{data}$ we apply the function proposed in TCP Vegas:

$$\widehat{Q}_{data} = \widehat{Q}_{delay} \cdot \frac{CWnd}{\widehat{RTT}_{now}} \quad (7)$$

To cope with changing path characteristics (e.g., route changes), the validity of $\widehat{RTT}_{min}$ is checked after tailored decrease. $\widehat{RTT}_{min}$ is reset, if no $\widehat{RTT}_{now}$ value close to $\widehat{RTT}_{min}$ has been measured for a certain number (e.g., 100) of tailored decreases.

The $\widehat{Q}_{delay}$ values of flows that share the same bottleneck will usually be similar, since the queuing delay affects all flows in the same way. This synchronizes the state changes of TCP LoLa across all competing flows. Still, the $\widehat{Q}_{delay}$ values will usually not be exactly identical, since RTT measurements are always noisy and RTT measurements are not taken at exactly the same points in time.

V. Evaluation

A. Evaluation Set-up

We implemented TCP LoLa as Linux kernel module[1] and evaluated it in a physical testbed, shown in fig. 3.

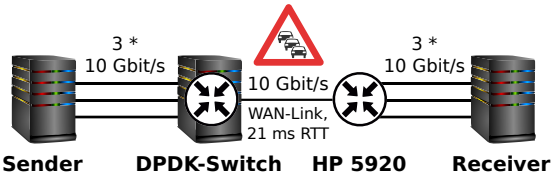


Fig. 3: Testbed

A *DPDK*-based software switch[2] was used at the bottleneck link. It provides detailed control over its buffers and also contains a delay emulator that was used to experiment with different RTTs. For experiments at lower speeds we connected an additional 100 Mbit/s link directly between the *DPDK-switch* and the *receiver* (due to transceiver speed limitations).

Sender, receiver and the DPDK-switch run on *Ubuntu 16.04* and are equipped with two Intel Xeon E5-2630 v3 CPUs and a 4-port Intel X710 10 Gbit/s NIC. The sender uses the *queuing discipline* "`fq`", which provides *packet pacing*. Traffic is generated with `iperf3`. RTT and CWnd measurements are collected with `TCPlog`[3] using the `tcpprobe` kernel module; throughput values with `CPUnetLOG`[4]. At the receiver the "RX rings" of the 10 Gbit/s NICs were increased from 512 packets to 4096 packets to avoid packet loss within the end system.

For all experiments in this paper we used the following parameters for TCP LoLa: $Q_{low} := 1\,\text{ms}$, $Q_{target} := 5\,\text{ms}$, $t_{sync} := 250\,\text{ms}$, $t_{measure} := 40\,\text{ms}$, $\gamma := 973/1024 \approx 95\%$. $\phi \approx 35$. These values have not been thoroughly optimized. Finding good values for a broad range of networking scenarios is subject of future work.

Unless stated otherwise the experiments use the following setup: Two flows transmit data from the *sender* to the *receiver*. The flows are sent over different network interfaces. Flow 1 starts at *second 0*, flow 2 starts at *second 20*. The bottleneck buffer is 100 MByte ($\widehat{=}$80 ms queuing delay at 10 Gbit/s). *Base-RTT* denotes the RTT with empty buffers, i.e., for most cases the WAN delay of 21 ms. For other base-RTTs the delay emulator was used. We repeated each experiment at least ten times. In the following, we always show the results of a representatively chosen single run for clarity.

[1] https://git.scc.kit.edu/TCP-LoLa/TCP-LoLa_for_Linux
[2] https://git.scc.kit.edu/TM/DPDK_AQM_Switch
[3] https://git.scc.kit.edu/CPUnetLOG/TCPlog
[4] https://git.scc.kit.edu/CPUnetLOG/CPUnetLOG

B. Basic Characteristics of TCP LoLa

Figure 4 shows the basic characteristics of TCP LoLa. High link utilization and flow rate fairness can be observed in fig. 4a. This is also achieved if flows have different RTTs, as shown in fig. 4d (flow 1: 21 ms, flow 2 101 ms). Figure 4b shows the CWnds of the two flows in the period $[16\,s \ldots 60\,s]$ to better illustrate the dynamics of fair flow balancing; each round is highlighted in gray. It can be observed that flow 1 keeps its CWnd unchanged during fair flow balancing, while flow 2 increases its CWnd according to the cubic slope of $X(t)$. After four rounds a fair rate share is achieved. The short periods where flow 1 increases its CWnd corresponds to the cubic increase that follows a tailored decrease.

The limited and low queuing delay is shown in fig. 4c for different base-RTTs. The lines close to 21 ms show the RTT observed by flow 1 (green) and flow 2 (blue) in the just described experiment. Since both flows traverse the same buffer, they experience the same queuing delay, so the lines are nearly identical. It can be seen that the RTT periodically reaches the target (26 ms), then the CWnds are reduced and RTT falls back to the base-RTT of about 21 ms. This behavior is independent of the actual base-RTT. We conducted similar experiments that only differ in their base-RTT: 5 ms (no WAN link), 61 ms, and 101 ms respectively. The corresponding measured RTT values are shown as gray lines.

Results in fig. 5 show that TCP LoLa is also scalable to lower speeds. Figures 5a and 5b show throughput and RTT of two TCP LoLa flows at a 100 Mbit/s bottleneck. Still it achieves high link utilization, fair flow rates, and a low queuing delay.

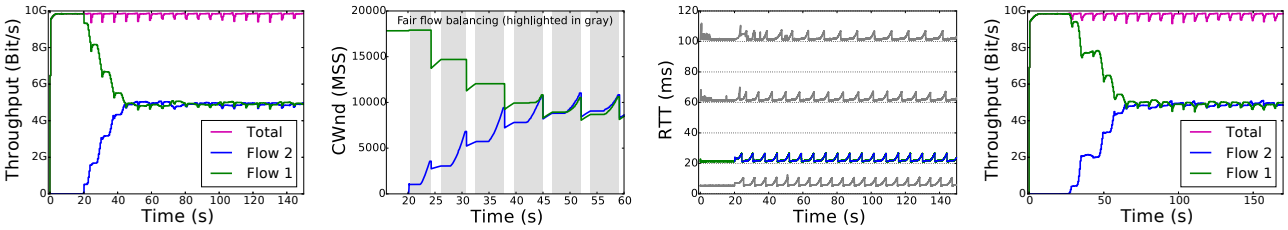
C. Many Starting Flows

TCP LoLa can also keep the delay low if multiple flows are started consecutively and run in parallel. Figure 6 shows the RTT and the CWnds of the flows in the following scenario: Every four seconds a new flow is started up until 18 flows run in parallel, each flow lasts for 200 s. So for the first 68 s, flows are started successively, afterwards all 18 flows run in parallel until *second 200*. The measured RTT values are nearly identical for all flows, so we show the RTT measurements from two of the flows only.

The queuing delay is kept around Q_{target} (see fig. 6b) and *no packet loss* was observed. Newly starting flows quickly converge toward a fair rate share, as can be seen in fig. 6a. Here, the CWnds of the flows are shown, but since all flows have the same RTT, equal CWnds lead to equal throughput.

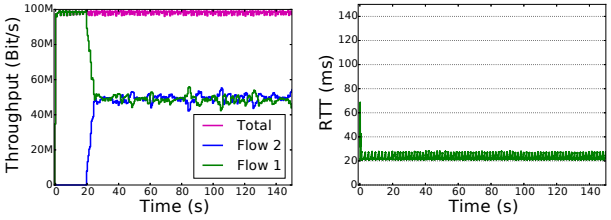
VI. Related Work

Since the introduction of TCP a large amount of congestion controls have been developed. A comprehensive survey can be found in [1]. TCP Reno and CUBIC TCP, the former and current standard congestion controls in Linux, are both loss-based. Compound TCP [10] (standard for Microsoft) is a *hybrid* congestion control, i.e., incorporates delay-based and loss-based elements but does not focus on low delay. TCP Vegas [3] is purely delay-based and can achieve low delays in many cases. Its queue size estimation is reused in many other congestion



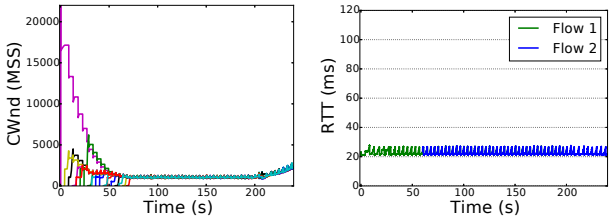
(a) Throughput and Fairness (b) CWnd (zoom) – fair flow balancing (c) RTT (different Base-RTTs) (d) Flows with different RTTs

Fig. 4: Behavior of TCP LoLa (10 Gbit/s)



(a) Throughput (b) RTT

Fig. 5: TCP LoLa, 100 Mbit/s bottleneck link



(a) CWnds (b) RTT (only two flows shown)

Fig. 6: Many starting TCP LoLa flows (10 Gbit/s)

controls, e.g., Compound TCP, YeAH TCP [2] and TCP LoLa. CDG [6] is based on delay gradients but has a TCP Reno-like additive increase that is not scalable for high-speed networks. BBR [5] focuses on high throughput even in networks with small buffers and tries to maintain a reasonable queuing delay. It can, however, produce increased delays or massive packet loss in certain scenarios as shown in [8].

AQMs like Codel [9] are a network-based approach to keep a low queuing delay even in the presence of loss-based congestion controls. With *L4S* [4] (Low Latency, Low Loss, Scalable throughput) an approach is proposed that should provide ultra-low latency for all applications. L4S, however, requires specialized AQMs.

VII. Conclusion

This paper presents TCP LoLa, a new congestion control for wide-area networks that primarily aims at keeping the queuing delay low at a fixed target and *simultaneously* achieves high throughput and high link utilization. Additionally, it avoids packet losses and is scalable from lower speed networks (i.e., 100 Mbit/s) to high speed networks (i.e., 10 Gbit/s). It achieves flow rate fairness independent of the flows' RTTs due to its new mechanism *fair flow balancing*. TCP LoLa is implemented as a Linux kernel module and has been evaluated in a physical testbed with 10 Gbit/s.

TCP LoLa is under active development. Next steps include modifications toward a fair coexistence with loss-based congestion controls, if AQM is present at the bottleneck. Also, more extensive evaluations are planned. Furthermore, we want to develop explicit congestion feedback like ECN, but tailored to delay-based congestion control, in order to improve the performance of TCP LoLa by reducing noise and in order to cope with multi bottleneck scenarios.

Acknowledgment

This work was supported by the bwNET100G+ project, which is funded by the Ministry of Science, Research, and the Arts Baden-Württemberg (MWK). The authors alone are responsible for the content of this paper.

References

[1] A. Afanasyev, N. Tilley, P. Reiher, and L. Kleinrock, "Host-to-Host Congestion Control for TCP," *Communications Surveys Tutorials, IEEE*, vol. 12, no. 3, pp. 304–342, Third 2010.

[2] A. Baiocchi, A. P. Castellani, and F. Vacirca, "YeAH-TCP: Yet Another Highspeed TCP," in *Int. Workshop on Protocols for Future, Large-Scale and Diverse Network Transports (PFLDNeT)*, vol. 7, 2007, pp. 37–42.

[3] L. S. Brakmo, S. W. O'Malley, and L. L. Peterson, "TCP Vegas: New Techniques for Congestion Detection and Avoidance," in *SIGCOMM '94*. New York, NY, USA: ACM, 1994, pp. 24–35.

[4] B. Briscoe, K. D. Schepper, and M. Bagnulo, "Low Latency, Low Loss, Scalable Throughput (L4S) Internet Service: Architecture," Internet Engineering Task Force, Internet-Draft draft-briscoe-tsvwg-l4s-arch-02, Mar. 2017, work in Progress. [Online]. Available: https://datatracker.ietf.org/doc/html/draft-briscoe-tsvwg-l4s-arch-02

[5] N. Cardwell, Y. Cheng, C. S. Gunn, S. H. Yeganeh, and V. Jacobson, "BBR: Congestion-Based Congestion Control," *ACM Queue*, vol. 14, no. 5, pp. 50:20–50:53, Oct. 2016.

[6] D. A. Hayes and G. Armitage, "Revisiting TCP Congestion Control Using Delay Gradients," in *NETWORKING'11*. Springer-Verlag, 2011, pp. 328–341.

[7] M. Hock, R. Bless, and M. Zitterbart, "Toward Coexistence of Different Congestion Control Mechanisms," in *2016 IEEE 41st Conference on Local Computer Networks*, November 2016, pp. 567–570.

[8] ——, "Experimental Evaluation of BBR Congestion Control," in *2017 IEEE 25th International Conference on Network Protocols (ICNP)*, Oct 2017.

[9] K. Nichols and V. Jacobson, "Controlling Queue Delay," *ACM Queue*, vol. 10, no. 5, pp. 20–34, May 2012.

[10] K. Tan, J. Song, Q. Zhang, and M. Sridharan, "A Compound TCP Approach for High-Speed and Long Distance Networks," in *INFOCOM 2006. 25th IEEE International Conference on Computer Communications. Proceedings*, April 2006, pp. 1–12.