Demo: Coexistence of Low Delay and Loss-based Congestion Controls in SDN-based Networks

Mario Hock, Michael König, Roland Bless, Martina Zitterbart
Karlsruhe Institute of Technology
Karlsruhe, Germany
E-Mail: mario.hock@kit.edu, michael.koenig2@student.kit.edu, bless@kit.edu, zitterbart@kit.edu

I. Introduction

The Internet serves an ever increasing variety of applications. Their requirements range from high throughput bulk transfers (i.e., high volume data transfers) to interactive or real-time applications that need low and bounded delays. Still, delay requirements are not properly considered in the current Internet. Hereby, the congestion control that is incorporated in TCP plays a crucial role. Currently, loss-based congestion controls (e.g., TCP Reno or CUBIC TCP) prevail.

It is well-known that loss-based congestion control can lead to large queues since they fill up the buffer to its maximum size in case of congestion. This, however, increases the delay that is experienced by the packets, thus resulting in a bad quality-of-experience for the above mentioned delay-sensitive applications. Deploying congestion controls that focus on the support of low delays could significantly improve this situation. However, a gradual deployment of these congestion controls or a parallel operation of low delay and loss-based congestion controls is challenging, since on regular tail-drop buffers the low delay congestion controls will be suppressed by the loss-based congestion controls.

In [1] we have shown that the capabilities of off-the-shelve OpenFlow switches can be used to facilitate the coexistence of low delay and loss-based congestion controls. Depending on the feature set of the deployed OpenFlow switches and on the existing network topology, different strategies can be used. In the demo we will use the *"Separate Queues"* approach. This means that all flows that use a low delay congestion control will be enqueued in a different queue than flows with a loss-based congestion control, at the bottleneck. The throughput ratio between the queues is determined by a scheduler, e.g., weighted round robin. In the demo, we use equal weights for both queues. The occupancy of each queue is determined by the congestion control of the respective flows and, thus, be usually high for the queue with *loss-based flows* and low for the queue with *low delay flows*. If separate queues are not supported in the used OpenFlow switch, either the *"Separate Paths"* or the *"Limit Queue Occupancy"* approach can be used. The first one is similar to the *"Separate Queues"* approach but requires redundant links at the bottleneck. The second one exploits rate limiters, so-called *OpenFlow meters*, to limit the maximal bandwidth used by the loss-based flows. This enables the low delay flows to control the buffer occupation at the bottleneck and, therefore, results in a low delay for all flows. However, due to the way loss-based congestion control works, the throughput of the loss-based flows is often reduced below the rate limit.

congestion control	coexistence mechanism	QoE
low delay	enabled	game feels *smooth* (high QoE), upload rate: *high*
	disabled	game feels sluggish (low QoE), upload rate: very low
loss-based	enabled	game feels sluggish (low QoE), upload rate: high
	disabled	game feels sluggish (low QoE), upload rate: very high (since other sender is suppressed)

TABLE I
EXPERIENCES OF THE PLAYERS DURING THE DEMO

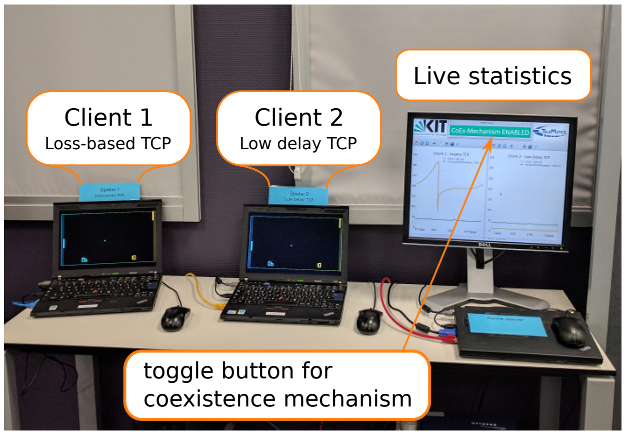

Fig. 1. Demo – Ready to play a round of "Pong"

II. Demo

The demo makes the delay that is created by the loss-based congestion control *perceivable* to the audience and, thus, demonstrates the advantages of low delay congestion control. The audience can play a network-enabled version of the well-known PONG game. In parallel, long living file transfers will generate congestion at a bottleneck that is shared by all flows.

Dependent on the congestion control in use and whether the co-existence mechanism is enabled, the quality-of-experience for the players will vary (summarized in table I and further explained in the following). Figure 1 shows how the demo setup will look like. In addition to the two laptops on which the game is played, a third laptop (or a dedicated monitor) will display live statistics of the network. The coexistence mechanism can be toggled anytime by clicking a button, placed atop the statistics graphs (shown in green in Fig. 1).

A. Delay

Within the game the delay can be perceived in the movement of the paddle. If the delay is high, the paddle moves sluggishly with noticeable lag. This makes the game hard to play and gives a bad quality-of-experience. As an illustration, the game shows where the paddle *should be*, i.e., the *local view* of the client itself, and where the paddle actually is, i.e., the location that is echoed back from the game server to the client (*server view)*. At the beginning the coexistence mechanism is enabled, `Client 1` uses a loss-based congestion control and `Client 2` uses a low delay congestion control. In this case the player at `Client 2` has a strong advantage in the game and both paddle markers of this player (local view and server view) are almost coincident. At the same time, the paddle markers of the other player will splitting up on every movement with the server view lagging behind. If the coexistence mechanism is switched off, the delay created by the loss-based congestion control will also affect the connection of the other player. Thus, both players will experience the same bad quality-of-experience. This shows that network-based coexistence mechanisms play an important role for the gradual deployment of low delay congestion control.

B. Throughput

Besides the quality-of-experience of the game, the throughput of the long living file transfers also depend on the congestion control and the coexistence mechanism. Without a coexistence mechanism, flows with loss-based congestion control can suppress flows with low delay congestion control at a common bottleneck. Some congestion controls are designed to resist such a suppression. CDG [2] and YeAH TCP [3], for example, contain a special *compatibility mode* that can alter their behavior to act similar to a loss-based congestion control. Such a mechanism, however, can only avoid suppression in terms of throughput but this still results in a high queuing delay. With an SDN-based coexistence mechanism, in contrast, the low delay congestion control can achieve a fair throughput and a low delay.

This behavior can be seen in the throughput and delay plots that are displayed in real time on one of the screens. Figure 2 shows a screenshot of these plots for `Client 2`. Within the displayed timespan the coexistence mechanism was toggled from *off* to *on*. It can be seen that, at first, the throughput of the file transfer is close to zero and the queuing delay is high. After the coexistence mechanism was enabled, the delay quickly drops to a low value and the throughput rate raises.

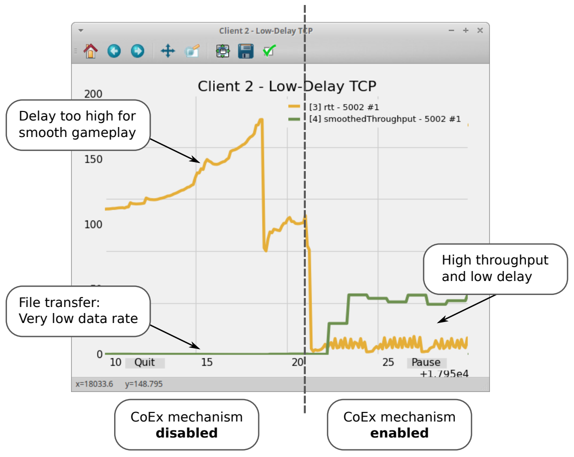

Fig. 2. Throughput and delay of the TCP LoLa client

C. Standard Demo Setup

The demo setup is shown in Fig. 3. It consists of two laptops (`Client 1` and `Client 2`) on which the `Pong` game is running. The game is implemented as a client-server application. The server component runs on a third laptop (`Game-Server`). In addition to that `Client 1` and `Client 2` perform a file transfer (to the `Game-Server`).

`Client 1` and `Client 2` are connected to a simple 1 Gbit/s non-SDN switch. The `Game-Server` is connected with 100 Mbit/s to an SDN switch. This is the bottleneck link for both clients. The two switches are interconnected with 1 Gbit/s.

In order to increase the portability of the demo, the SDN switch is emulated by a fourth laptop, equipped with multiple network interfaces. The network statistics are gathered and displayed with the open source tool `TCPlog`[1] and `TCPlivePLOT`[2]. `TCPlog` is running on the clients, `TCPlivePLOT` on the server. The measurement data is also transmitted over the given network links. In order to avoid any interference with the actual data streams, a fixed bandwidth is allocated at the bottleneck switch for this control traffic. The coexistence mechanism can be toggled by pressing a button on the `Game-Server`, the toggling information is also transmitted over this control channel.

We picked the broadly used CUBIC TCP as loss-based congestion control and the novel TCP LoLa as low delay congestion control (more infos in section III). In a typical run of the demo `Client 1` represents a legacy host, running CUBIC TCP. `Client 2` runs TCP LoLa.

D. Varying Setup: Quiz

In addition to that, the demo setup also offers the possibility to change the used congestion controls. This makes it possible to conduct a little quiz with the audience: For both clients a random (or specifically chosen) congestion control is set. Participants can then play the game and have to decide what kind of congestion controls are in use. Hereby, toggling the

[1]https://git.scc.kit.edu/CPUnetLOG/TCPlog
[2]https://git.scc.kit.edu/CPUnetLOG/TCPlivePLOT

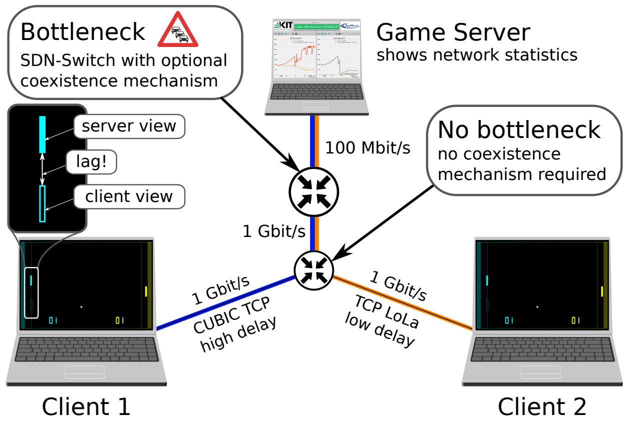

Fig. 3. Demo Setup

coexistence mechanism can either be a way to get more insights about the situation, or its state is also kept hidden and has to be found out, as well.

III. Congestion Controls

This section gives some background about the different congestion controls that are used in the context of this demo.

A. *TCP Reno (loss-based)*

TCP Reno can be considered the base-line congestion control in the Internet. It is loss-based and follows the well-known *AIMD* principle. AIMD stands for "Additive Increase, Multiplicative Decrease". This means that TCP Reno typically increases its *Congestion Window (CWnd)* linearly, if no congestion is assumed. If a packet loss occurs, TCP Reno assumes a congestion and reduces its CWnd by factor 0.5.

The congestion window describes the amount of data that corresponding TCP flow is allowed to have *"in flight"* (i.e., data that is sent but not yet acknowledged). Most congestion controls use a congestion window, but use different algorithms to determine an appropriate size.

It is known that TCP Reno does not scale well in networks with high bandwidth and/or high propagation delays.

B. Cubic TCP *(loss-based)*

Cubic TCP is the standard congestion control in Linux. It solves the scalability issue of TCP Reno by changing the CWnd increase function and the decrease factor. Instead of a linear increase, Cubic TCP uses a cubic function. The inflection point of the cubic function is usually shifted to the value on which the last packet loss occurred. This means that the slope of the function is very low around this point but increases with distance to this point. This leads to an improved scalability. Instead of 0.5 Cubic TCP reduces its CWnd merely to a factor of 0.7 after a packet loss.

Both TCP Reno and Cubic TCP reduce their CWnd only after a packet loss. This usually only happens if the bottleneck buffer is completely exhausted. This means that with large buffers, both congestion controls induce large queuing delays. With small buffers both congestion controls are known to only achieve a reduced throughput.

C. *TCP Vegas (low delay)*

TCP Vegas is a low delay congestion control and was used in the evaluation of [1]. It is able to detect congestion before the bottleneck buffer is exhausted. Instead of packet loss TCP Vegas uses an increased *round-trip-time (RTT)* as congestion signal. TCP Vegas, however, does not scale well with the bandwidth, also TCP Vegas is prone to the so-called *late-comer advantage problem*: If multiple long living flows that are not started at the same time share a bottleneck, the "late comer" often gets a larger bandwidth share. In addition to that this situation can lead to increased queuing delays.

D. *TCP LoLa (low delay)*

TCP LoLa [4] is a novel congestion control that explicitly focus on keeping low queuing delays and on scalability to high bandwidths. In addition to that TCP LoLa provides *convergence to fairness* among competing TCP LoLa flows, independent of their RTTs.

TCP LoLa is also delay-based. It monitors the RTT and tries to detect if a persisting *standing queue* [5] is formed at the bottleneck and tries to determine its size. Since the main task of a congestion control is to regulate CWnd, the standing queue is of particular interest, due to the following relation: If CWnd is too small, the bottleneck link cannot be fully utilized. In this case no standing queue builds up. If CWnd is too large, the excess in-flight data forms a persisting standing queue. In order to achieve high throughput and low delay, TCP LoLa tries to keep the size of the standing queue on a low level, but above *zero*.

IV. Requirements for the Demo

- A table with space for 4 laptops
- Power supply for the laptops
- A big screen to display the statistics (if possible)
- Poster board
- Setup-time: 1 hour
- No Internet connection required

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] 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.

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

[3] 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.

[4] M. Hock, F. Neumeister, M. Zitterbart, and R. Bless, "TCP LoLa: Congestion Control for Low Latencies and High Throughput," in *2017 IEEE 42st Conference on Local Computer Networks (to be published)*, October 2017.

[5] K. Nichols, V. Jacobson, A. McGregor, and J. Iyengar, "Controlled Delay Active Queue Management," Internet-Draft, Internet Engineering Task Force, Work in Progress, March 2017.