Lightweight and Flexible Single-Path Congestion Control Coupling

Safiqul Islam, Michael Welzl, Stein Gjessing
University of Oslo
Email: {safiquli, michawe, steing}@ifi.uio.no

***Abstract*—Communication between two Internet hosts using parallel connections may result in unwanted interference between the connections. In this dissertation, we propose a sender-side solution to address this problem by letting the congestion controllers of the different connections collaborate, correctly taking congestion control logic into account. Real-life experiments and simulations show that our solution works for a wide variety of congestion control mechanisms, provides great flexibility when allocating application traffic to the connections, and results in lower queuing delay and less packet loss.**

I. Introduction

When two Internet hosts communicate, several connections may be opened in order to exchange multiple files or streams of data or video. This could for example be the case when a user downloads a web page containing several images and other items from a server, or when a user is involved in interactive real-time communication using WebRTC[1]. These parallel connections, each having their own congestion control mechanism, compete over the Internet, resulting in undefined behaviour and unwanted interference between the flows. Such competition can induce undesired queuing delay and losses.

Generally, there are two approaches to combine the connections that are known to share a common path: by merging application layer data streams onto a single transport layer connection (SPDY [1] and HTTP/2 address this problem by multiplexing all data on one single TCP connection. Doing this at the application layer leads to transport layer Head-of-Line blocking delay; this has recently been addressed by QUIC [2], which operates over UDP.); or by combining the congestion controls of the connections at the transport layer. This thesis emphasizes on the latter: *coupled congestion control*.

Almost a decade ago, there were some efforts to combine the congestion controls of multiple TCP connections sharing the same bottleneck. To the best of our knowledge, Congestion Manager (CM) [3] is the oldest such effort. However, it was hard to implement: it requires revamping the entire transport layer implementation because the congestion control logic needs to be taken out of the transport protocols. Ensemble TCP (E-TCP) [4] and Ensemble Flow Congestion Management (EFCM) [5] are also two proposals along these lines; while E-TCP tried to be no more aggressive than one flow, EFCM tried to be as aggressive as n flows. Neither E-TCP nor EFCM correctly considered TCP congestion states.

All the aforementioned approaches have an additional, entirely different problem: they assume that multiple TCP connections sending to the same destination would take the same path. This is not always true – load-balancing mechanisms such as Equal-Cost Multi-Path (ECMP) and LAG [6] may force them to take different paths. Therefore, in this particular scenario, combining the congestion controllers would incur wrong behavior.

The overarching objective of this dissertation is to find an efficient and feasible (deployable) way to combine the congestion controllers of parallel connections between the same pair of hosts. The objective can be concretized to: 1) make our design simple and flexible while minimizing the changes to the existing mechanisms. 2) ensure that connections take the same path[2]. 3) apply our solution to different congestion control mechanisms ranging from bulk transfer to real-time media (this lets us derive general guidelines for applying it to congestion control mechanisms in the future). 4) reduce overall loss and delay.

II. Publications and Disseminations

Publications: the full text of the dissertation can be found in [7]. Publications that contain the results of this work are are available in [8, 9, 10, 11, 12]. The paper presented at the ACM SIGCOMM Capacity Sharing workshop won the best paper award and was published in ACM SIGCOMM Computer Communication Review [8].

Implementations: we implemented the solution in 1) the ns-2 simulator, 2) Chromium (which uses Google Congestion Control[3]) [13], and 3) the FreeBSD kernel in two different ways: i) with state shared across the freely available VirtualBox[4] hypervisor, and ii) as a single-OS implementation that couples connections from the same OS only. The source code of the implementation can be found here [14].

IETF/IRTF Standardizations: we have also proposed and presented our mechanism to the Internet Engineering Task

[1]The IETF counterpart of the W3C WebRTC standard is called RTCWEB. For simplicity, we will use the term “WebRTC” for the whole set of standards.

[2]Note that coupled congestion control for Multipath TCP (MPTCP) assumes that flows could take different paths, and ideally also traverse different bottlenecks. This is exactly the opposite of what we try to achieve here. Using a mechanism like ours in MPTCP would result in wrong behavior, e.g., giving a large share of the aggregate to a new connection can result in using a very large Initial Window on an entirely different path.

[3]The implementation in the version of Chromium was 47.0.2494.0

[4]https://www.virtualbox.org

978-1-5386-3416-5/18/$31.00 © 2018 IEEE
DOI 10.1109/NOMS.2018.8406137

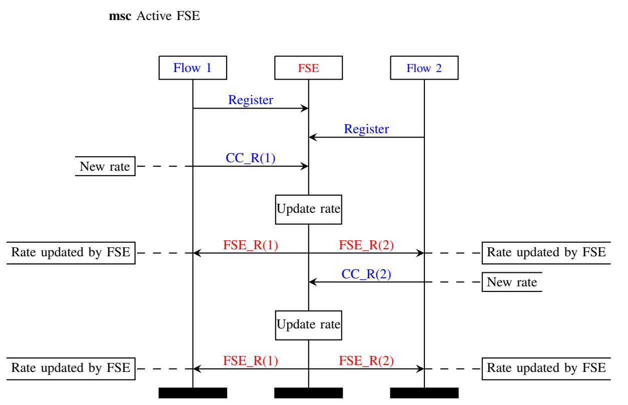
Force (IETF) community in order to bridge the gap between the industry and academia. The Internet drafts are available in [15, 16, 17]. We have actively contributed to three IETF/IRTF Working Groups:

1) **RTP Media Congestion Avoidance Techniques (RMCAT)**: We have proposed our solution to the IETF RMCAT Working Group (WG) and provided necessary guidelines to the implementers on how to apply the mechanisms to not only WebRTC congestion control mechanisms, Network Assisted Dynamic Adaptation (NADA) and Google Congestion Control (GCC), but also congestion control mechanisms beyond WebRTC [15]. Results are presented in the IETF 87, 90, 91, 92, 93, 94, 95 meetings. This work is going to be an RFC soon.
2) **Internet Congestion Control Research Group (ICCRG)**: We have proposed a coupled congestion control mechanism for TCP flows along with the ACK-Clock mechanism from [11] and TCP-in-UDP encapsulation from [10] in [16].
3) **TCP Maintenance and Minor Extensions (TCPM)**: We have also proposed an update of RFC2140, TCP Control Block Interdependence, in the IETF TCPM WG [17]. We include the current implementation status related to TCB sharing (Temporal sharing) in Linux kernel version 4.6, FreeBSD 10 and Windows (as of October 2016) and discuss the impact of sharing certain states (e.g., ssthresh, RTT). The draft has been accepted as a WG item.

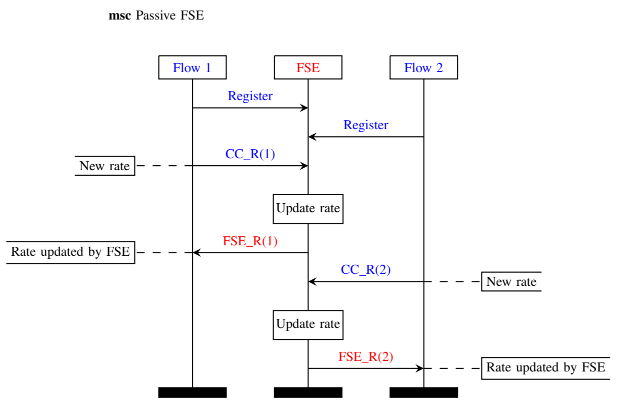
III. Contributions

We contribute to a better understanding of coupled congestion control by covering a wide range of congestion control mechanisms used for real-time media, background-transfer, and web-like traffic. An extensive experimentation of the proposed solution has been done, using the six congestion control mechanisms: 1) Rate Adaptation Protocol (RAP) [18], a simple rate based AIMD protocol; 2) TCP Friendly Rate Control (TFRC) [19], because it updates the rates based on an equation, and is currently the only standardized congestion control mechanism aimed at supporting media flows; 3) Network Assisted Dynamic Adaptation (NADA) [20], a work-in-progress congestion control algorithm for webRTC; 4) Google Congestion Control (GCC) [13], another work-in-progess congestion control mechanism for webRTC, and is currently deployed in web browsers (Chrome, Firefox); 5) Low Extra Delay Background Transport (LEDBAT) [21] - because it is a delay-based mechanism, and very well known as a less-than-best-effort transport protocol for services such as operating system updates; and 6) Transmission Control Protocol (TCP), because it is the most widely used transport protocol for web-like traffic as well as bulk transfers.

We first describe our solution for the real-time media in Section III-A, and then illustrate how we couple the TCP congestion control mechanisms in Section III-B, and finally discuss how to ensure a common path in Section III-C.



(a) Active FSE: the FSE initiates updates whenever the FSE state changes.



(b) Passive FSE: flows asynchronously synchronize their state with the FSE whenever they change their sending rate or congestion window.

Fig. 1: Active and Passive versions of the FSE. CC_R is the rate received from the flow's congestion controller. FSE_R(f) is the rate calculated by the FSE.

A. Coupled Congestion Control for RTP Media

We devise a method which combines the congestion controls of multiple RTP based real-time media flows. The method involves a central storage element, the "Flow State Exchange (FSE)", and can be initiated in two ways: 1) Active FSE – initiates communication with the flows actively, and 2) Passive FSE – maintains the state of the ensemble and makes it available to *only* the flow requesting a new rate. Fig. 1 illustrates the interaction between flows and the FSE in the active and passive variants.

We have applied a very simple active version of the FSE to the RAP and TFRC congestion control mechanisms where we keep track of the aggregate of all the flows and assign each flow a weighted proportion of the aggregate. We have found with simulations that the mechanism yields perfect control over fairness, however it did not reduce overall queuing delay and loss. This, in turn, leads us to an important finding regarding synchronization and de-synchronization of flows: since, our solution de-synchronizes the flows, a flow experiencing congestion and halving its rate in an ensemble of 5 flows reduces the aggregate e.g. from $5Mbps$ to $4.5Mbps$, whereas, without the coupling, they sometimes synchronize and halve

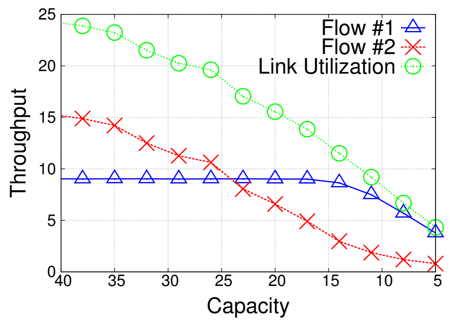


Fig. 2: High-priority (1) application-limited flow #1 is hardly affected by a low-priority (0.2) greedy flow #2 as long as there is enough capacity for flow #1.

their rates as well as the aggregate. We solve the problem by extending the mechanism to emulate the behavior of one flow on congestion (see chapter 5 in [7] for details).

The passive version (see Fig. 1(b)) is easier to implement than the active version. It requires less signalling, with inherently relaxed timing constraints, and does not require tight integration with the congestion control mechanism. This means that a passive FSE could be implemented as a stand-alone tool. However, a passive FSE can create problems with flows that do not update their rate for relatively longer periods – typically because RTTs are significantly different, e.g., TCP updates its rate as a function of the RTT (see [9]).

We have also found that applying the FSE to different congestion control mechanisms requires a small adaptation to the algorithm. In doing so, we have discovered that these two mechanisms exhibit aspects which lead us to an interesting finding: both GCC and NADA update their rates at fixed intervals—not as a function of the RTT. Thus, we can use a simple passive version of the FSE—a less time-constrained request-response style of signaling between the FSE and the congestion control mechanisms. Our evaluation covers a range of pertinent test cases [22] to show the efficacy of applying passive coupling on NADA and GCC; results are detailed in chapter 6 in [7]. In addition, we show—with experiments where we have delayed the feedback between a flow and the FSE—that this less time-constrained request-response style of signaling opens the possibilities to run the FSE as a stand-alone management tool.

Selected Performance Results: Fig. 2 illustrates the behavior of a greedy flow with low priority (0.2) and an application limited flow with a higher priority (1) that is sending based on a video trace. It can be observed that the low-priority flow can grab unused bandwidth as long as there is enough capacity. The bandwidth was not completely utilized in these tests because the simulation time was based on the total duration of the video trace, which was too short for the low priority flow to reach the capacity limit.

Fig. 3 shows the goodput values of two TFRC flows with FSE in the presence of background synthetic traffic[5] when the priority of the first flow is set to 1, while the other flows' priority is varied. As it can be seen from the graph, the goodputs of flows 1 and 2 are very close to the theoretical value that one might expect: for example, when the priority of flow 2 is 0.2, 0.5 and 0.8, the goodput ratio is 0.199 (instead of 0.2), 0.499 (instead of 0.5) and 0.799 (instead of 0.8), respectively. These are surprisingly precise values, seen by the receivers in the presence of synthetic background traffic with various numbers of arriving and departing flows and RTTs at any instance of time.

[5]The TMIX [23] traffic, used in our simulation, is taken from 60-minute trace of campus traffic at the University of North Carolina [24]

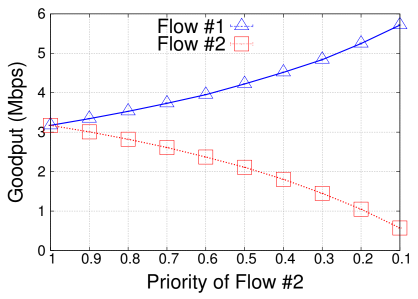


Fig. 3: Goodput of two FSE-controlled flows competing with synthetic traffic

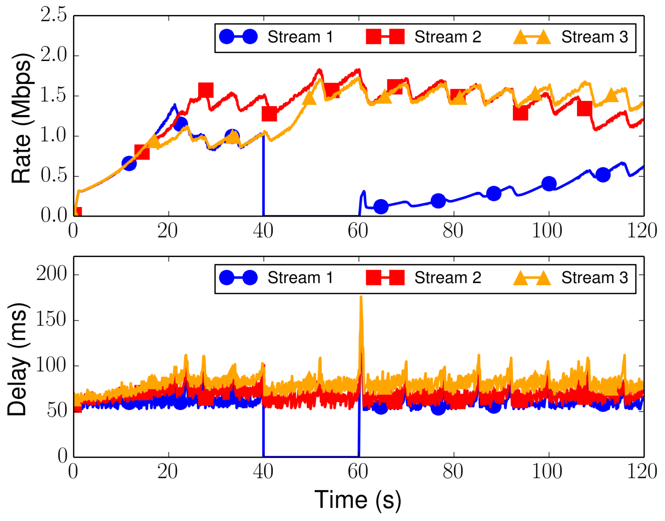
Fig. 4 shows results for GCC with two continuous and one intermittent RMCAT flows in the "media pause and resume" test case [22]. Fig. 4(a) shows the rate and delay characteristics without the FSE, and Fig. 4(b) shows results with the FSE. In this test, all three flows start with the same priority. At around 40 s flow 1 was paused for 20 seconds. Fig. 4(b) shows that the FSE distributes the aggregate fairly by enforcing strict fairness. The FSE also improves the delay spike introduced a GCC flow (see Fig. 4) after a long pause. More NADA and GCC results are detailed in [7, 9].

B. TCP Congestion Control Coupling

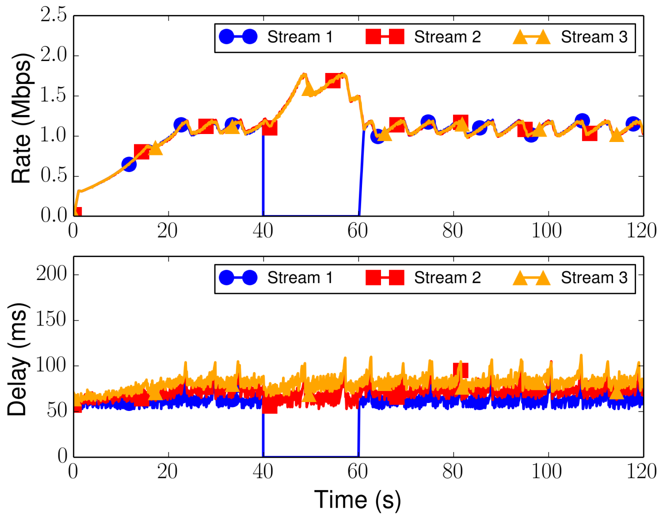
WebRTC media flows are rate-based and stateless, hence it is easier to combine their congestion controls than with TCP. To couple TCP flows, we adopt a different design approach by correctly honouring the stateful nature of the TCP congestion control algorithms. We introduce *ctrlTCP*, a minimally-invasive solution that is flexible enough to cater for needs ranging from weighted fairness to potentially offering Internet-wide benefits from reduced inter-flow competition.

TCP operates on loss events (one or more packet losses per RTT), not individual packet losses. When congestion controls are combined, this logic should be preserved; we explain this by assuming two connections traversing the same bottleneck. A single packet drop from connection 1, two drops from connections 1 and 2 or multiple packet drops from connection 2 only, should all result in the same behavior of the traffic aggregate. Simply sharing TCP variables such as $cwnd$ or $ssthresh$ cannot achieve this.

A Retransmission Timeout (RTO) should only occur when no more acknowledgments arrive (TCP has been greatly improved over the years to avoid RTOs when ACKs arrive in the Fast Recovery mode). However, if connection 1 sees no ACKs in Fast Recovery, yet they do arrive for connection 2, then connection 1 makes a mistake when it fires an RTO and enters Slow Start. Again, sharing TCP variables fails unless the



(a) Without FSE



(b) With FSE

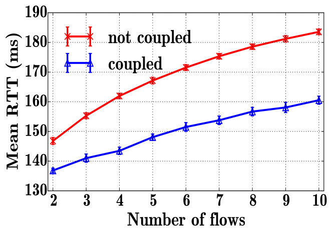
Fig. 4: Sending rates and delays of two continuous and one intermittent GCC flows, with and without the FSE. (Note that markers identify the line and not plotted points)

states of TCP's congestion control mechanism are considered. With E-TCP, connection 1 can force connection 2 to reduce its share of the *cwnd* in Fast Recovery, reducing the chance for connection 2 to correctly terminate this phase.

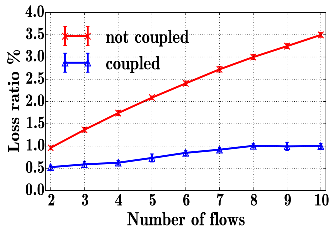
We have shown the problems of not sharing states carefully using simulations with both E-TCP and EFCM (see chapter 9 in [7] for TCP state sharing problem of E-TCP and EFCM in detail). To solve this, *ctrlTCP* basically keeps the multiple TCP connections intact and lets the TCP congestion controllers communicate by correctly taking the TCP states. We have highlighted how *ctrlTCP* is built on the state-of-the-art, and showcased the advantages of the solution using the ns-2 simulations and an implementation in the FreeBSD kernel.

ACK-Clocking: Coupled congestion control can let a window-based connection quickly increase its window by giving it a large share of the aggregate, e.g., when a connection joins, or resumes after a quiescent period. The crux of such sharing is that it can produce bursts and this can be mitigated using some form of *pacing*. We show the problem in [11]. Prior approaches use timer-based pacing which is known to have some disadvantages. We have proposed a novel solution to pace the packets by correctly maintaining the ACK clock instead of using a timer, thereby minimizing the impact on the dynamics in the network. This mechanism is an add-on to our ctrlTCP mechanism, but can easily be ported to better initialize new connections in order to improve the transient performance ("temporal sharing" and "ensemble sharing" of RFC2140 [25]). We have shown its positive impact on web-like short flows which can complete much faster with our *ctrlTCP*. For more details, see chapter 7 in [7].

Combining a Heterogeneous Set of Congestion Controllers: Combining a heterogeneous set of congestion control mechanisms can yield several performance benefits, especially when one of the mechanisms reacts on a congestion event earlier than the others. As a first step, we have applied our solution to combine parallel LEDBAT flows, and after that, having shown our results with LEDBAT flows, we have tested a combination of a LEDBAT and a TCP connection. This leads us to an important finding: since LEDBAT notices the increasing delay as soon as the queue grows, our solution, in fact, ensures that the queue does not grow even for TCP (see chapter 8 in [7]). This step—combining a heterogeneous set of congestion controllers—could allow to combine the WebRTC data channel and video flows (multiplexed onto a UDP port pair) that use a loss-based and a delay-based congestion control mechanisms, respectively.



(a) Average delay (in ms)



(b) Loss ratio

Fig. 5: Average delay, and loss ratio as the number of TCP connections is varied, with and without coupled congestion control (emulation)

Selected Performance Results: Fig. 5(a) and 5(b) illustrate the average RTT, and loss ratio, with and without coupling for varying numbers of flows, respectively. It can be seen from the graphs that *ctrlTCP* reduces both the average RTT and loss without significantly affecting the throughput as we varied the number of flows (see [10] for more results).

We have shown that we even can control TCP connections originating from different VMs by placing our solution in a hypervisor. This allows us to take a major step forward where our solution can be applied in multi-tenant data-center networks. In multi-tenant datacenters, the guest OSes of clients may be diverse and utilize an Internet-like mix of old and new TCP congestion control implementations, with and without ECN, following Reno, Cubic, Westwood or any other TCP "flavor" that e.g. Linux and FreeBSD allow to configure via their pluggable congestion control frameworks [26]. This may put some users at a disadvantage, depending on how aggressively their congestion control probes for the available capacity. Moreover, unfair users may have an incentive to obtain a larger share of the capacity by opening multiple TCP connections. This is illustrated in Fig. 6, which shows Jain's Fairness Index [27] ($(\sum_{i=1}^{N} x_i(t))^2 / N \sum_{i=1}^{N} x_i(t)^2$) for $N = 2$ aggregate flows x_1 and x_2, calculated using the traffic that originated from two VMs across a 10 Mbit/s$\times$100 ms bottleneck with and without *ctrlTCP*. For details, see chapter 3 in [7].

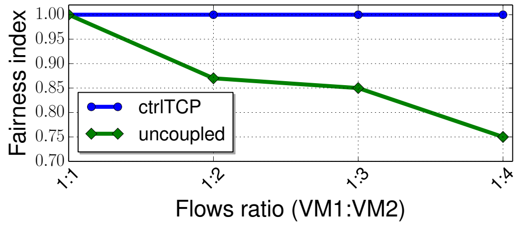


Fig. 6: Fairness between two VMs, with 1 flow in VM1 and 1 to 4 flows in VM2. Without coupling, fairness deteriorates; our *ctrlTCP* algorithm achieves perfect fairness.

Fig. 7 demonstrates that *ctrlTCP*'s coupling can yield a significant improvement in the short flow's completion time (FCT). We repeated this test 10 times with randomly picked flow start times over the first second for the long flow (25 Mb) and the sixth second for the short flow (200 Kb). We show the reduction of FCTs in emulation while varying the bottleneck capacity (6, 8, and 10 Mbps) in Fig. 7. The impact on the long flow was negligible.

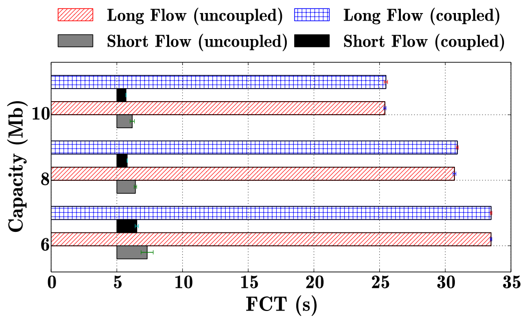


Fig. 7: Flow completion time (FCT) of short flows (emulation).

C. Ensuring a Common Bottleneck

All the prior approaches assume that connections traverse the same path between the same pair of hosts. In practice, this becomes problematic due to load-balancing mechanisms like ECMP and LAG. Such a case has not been considered before, and thus gives a new insight: how can we ensure that flows traverse the same path? In the course of answering this, we survey mechanisms that ensure the same path between the same pair of hosts. In addition, we propose a light-weight, dynamically configured TCP-in-UDP (TiU) encapsulation scheme—several TCP connections in UDP datagrams carrying a single port number pair—that ensures our coupled flows do indeed share all bottlenecks along a single path. TiU is optional. Our coupled congestion control strategy is applicable to scenarios wherever overlapping TCP flows must follow the same path (multiplexed onto a single transport layer connections such as webRTC, VPNs). We document the study and our encapsulation method in detail in chapter 9 in [7].

IV. Conclusions

This dissertation has proposed and evaluated a simple and flexible coupled congestion control solution, built on top of the state-of-the-art, to manage parallel connections between the same host-pair. The solution has been applied to different congestion control mechanisms, and the results have shown that it can reduce overall delay and loss.

Addressing the research objectives introduced in Section I: 1) We have introduced a simple and flexible sender-side coupled congestion control mechanism that minimizes the changes to the existing mechanisms. We have implemented our solution in the ns-2, Chromium and FreeBSD kernel. 2) Methods to ensure that flows take the same path were discussed. 3) We have shown how to apply our solutions to different congestion control mechanisms. Applying a coupled congestion mechanism requires a case-by-case analysis of the congestion control algorithms (see chapter 3 in [7]). 4) We have shown that our mechanism can precisely allocate the shares of the available bandwidth as well as reduce delay and losses. The inventors of the prior approaches designed their coupling mechanisms considering TCP flows, not real-time media flows. Therefore, impact on FCT and throughput gain was more important than reducing latency and losses.

The dissertation has created a platform for additional research work. With measurement based Shared Bottleneck Detection (SBD) [28], it could be possible to combine connections between different host pairs. It would therefore be interesting to explore this further. In [9], we have shown that our solution can work with relaxed time constraints for the signaling between congestion control mechanisms and the coupling entity; it could therefore be possible to use a configuration-based SBD approach where we place our solution on a wireless access point in order to control multiple stations associated with it.

We have shown that a loss-based congestion control mechanism can benefit from a latency-sensitive delay-based mechanism when they are coupled together. This could allow us to combine WebRTC data and video flows since they use a loss-based and a delay-based congestion control mechanism. We therefore encourage others to develop such extensions of our algorithm in order to investigate controlling a larger variety of congestion control mechanisms with it.

Our ongoing work on coupled congestion control focuses on the practical deployments of our results; the WebRTC Internet draft [15] is close to publication as an RFC, and we are working on an FSE-SBD implementation in the Chromium browser. The FreeBSD implementation of the TCP coupling is used by the NEAT European research project [29].

V. Acknowledgments

This work has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No. 644334 (NEAT).

References

[1] "SPDY: An experimental protocol for a faster web," http://www.chromium.org/spdy/spdy-whitepaper.

[2] J. Iyengar, I. Swett, R. Hamilton, and A. Wilk, "QUIC: A UDP-Based Secure and Reliable Transport for HTTP/2," IETF, I-D draft-tsvwg-quic-protocol-02, Jan. 2016, work in Progress. [Online]. Available: https://tools.ietf.org/html/draft-tsvwg-quic-protocol-02

[3] H. Balakrishnan, H. S. Rahul, and S. Seshan, "An integrated congestion management architecture for internet hosts," in *Proceedings of the Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, ser. SIGCOMM '99. New York, NY, USA: ACM, 1999, pp. 175–187. [Online]. Available: http://doi.acm.org/10.1145/316188.316220

[4] L. Eggert, J. Heidemann, and J. Touch, "Effects of ensemble TCP," *USC/Information Sciences Institute*, vol. 7, no. 1, December 1999.

[5] M. Savorić, H. Karl, M. Schläger, T. Poschwatta, and A. Wolisz, "Analysis and performance evaluation of the EFCM common congestion controller for TCP connections," *Computer Networks*, vol. 49, no. 2, pp. 269–294, 2005.

[6] R. Krishnan, L. Yong, A. Ghanwani, N. So, and B. Khasnabish, "Mechanisms for Optimizing Link Aggregation Group (LAG) and Equal-Cost Multipath (ECMP) Component Link Utilization in Networks," RFC 7424 (Informational), Internet Engineering Task Force, Jan. 2015. [Online]. Available: http://www.ietf.org/rfc/rfc7424.txt

[7] S. Islam, "Lightweight and flexible single-path congestion control coupling," Univ. of Oslo, PhD Thesis, July 2017. [Online]. Available: https://www.duo.uio.no/handle/10852/56951

[8] S. Islam, M. Welzl, S. Gjessing, and N. Khademi, "Coupled congestion control for RTP media," *SIGCOMM Comput. Commun. Rev.*, vol. 44, no. 4, Aug. 2014. [Online]. Available: http://doi.acm.org/10.1145/2630088.2630089

[9] S. Islam, M. Welzl, D. Hayes, and S. Gjessing, "Managing real-time media flows through a flow state exchange," in *Network Operations and Management Symposium (NOMS), 2016 IEEE/IFIP*. IEEE, 2016, pp. 112–120.

[10] S. Islam, M. Welzl, K. Hiorth, D. Hayes, O. Dale, G. Armitage, and S. Gjessing, "Single-path TCP congestion control coupling," http://safiquli.at.ifi.uio.no/paper/tcp-ccc-techrep.pdf, under submission.

[11] S. Islam and M. Welzl, "Start me up: Determining and sharing TCP's initial congestion window," in *Proceedings of the 2016 Applied Networking Research Workshop*, ser. ANRW '16. New York, NY, USA: ACM, 2016, pp. 52–54. [Online]. Available: http://doi.acm.org/10.1145/2959424.2959440

[12] S. Islam, M. Welzl, S. Gjessing, and J. You, "OpenTCP: Combining congestion controls of parallel TCP connections," in *Advanced Information Management, Communicates, Electronic and Automation Control Conference (IMCEC), 2016 IEEE*. IEEE, 2016, pp. 194–198.

[13] S. Holmer, H. Lundin, G. Carlucci, L. D. Cicco, and S. Mascolo, "A google congestion control algorithm for real-time communication," draft-alvestrand-rmcat-congestion-03 (work in progress), 2015.

[14] "Freebsd implementation of the coupled congestion control," source code. [Online]. Available: https://bitbucket.org/safiqul

[15] S. Islam, M. Welzl, and S. Gjessing, "Coupled congestion control for RTP media," Internet Engineering Task Force, Internet-Draft draft-ietf-rmcat-coupled-cc-07, Sep. 2017, work in Progress. [Online]. Available: https://datatracker.ietf.org/doc/html/draft-ietf-rmcat-coupled-cc-07

[16] M. Welzl, S. Islam, K. Hiorth, and J. You, "TCP-CCC: single-path TCP congestion control coupling," Internet Engineering Task Force, Internet-Draft draft-welzl-tcp-ccc-00, Oct. 2016, work in Progress. [Online]. Available: https://datatracker.ietf.org/doc/html/draft-welzl-tcp-ccc-00

[17] D. J. D. Touch, M. Welzl, S. Islam, and J. You, "TCP Control Block Interdependence," Internet Engineering Task Force, Internet-Draft draft-touch-tcpm-2140bis-02, Jan. 2017, work in Progress. [Online]. Available: https://datatracker.ietf.org/doc/html/draft-touch-tcpm-2140bis-02

[18] R. Rejaie, M. Handley, and D. Estrin, "RAP: An end-to-end rate-based congestion control mechanism for realtime streams in the internet," in *IEEE INFOCOM*, 1999.

[19] M. Handley, S. Floyd, J. Padhye, and J. Widmer, "TCP Friendly Rate Control (TFRC): Protocol Specification," RFC 3448 (Proposed Standard), Internet Engineering Task Force, Jan. 2003, obsoleted by RFC 5348. [Online]. Available: http://www.ietf.org/rfc/rfc3448.txt

[20] X. Zhu, R. Pan, M. A. Ramalho, S. Mena, C. Ganzhorn, P. E. Jones, and S. D. Aronco, "NADA: A unified congestion control scheme for real-time media," draft-ietf-rmcat-nada-00 (work in progress), 2015.

[21] S. Shalunov, G. Hazel, J. Iyengar, and M. Kuehlewind, "Low Extra Delay Background Transport (LEDBAT)," RFC 6817 (Experimental), Internet Engineering Task Force, Dec. 2012. [Online]. Available: http://www.ietf.org/rfc/rfc6817.txt

[22] Z. Sarker, V. Singh, X. Zhu, and M. A. Ramalho, "Test cases for evaluating RMCAT proposals," draft-ietf-rmcat-eval-test-01 (work in progress), 2015.

[23] M. C. Weigle, P. Adurthi, F. Hernández-Campos, K. Jeffay, and F. D. Smith, "Tmix: A tool for generating realistic TCP application workloads in ns-2," *SIGCOMM Comput. Commun. Rev.*, vol. 36, no. 3, pp. 65–76, Jul. 2006. [Online]. Available: http://doi.acm.org/10.1145/1140086.1140094

[24] L. Andrew, S. Floyd, and G. Wang, "Common TCP Evaluation Suite," IETF, Internet-Draft draft-irtf-tmrg-tests-02, Jul. 2009, work in Progress. [Online]. Available: https://tools.ietf.org/html/draft-irtf-tmrg-tests-02

[25] J. Touch, "TCP Control Block Interdependence," RFC 2140 (Informational), Internet Engineering Task Force, Apr. 1997. [Online]. Available: http://www.ietf.org/rfc/rfc2140.txt

[26] A. Afanasyev, N. Tilley, P. Reiher, and L. Kleinrock, "Host-to-host congestion control for TCP," *Commun. Surveys Tuts.*, vol. 12, no. 3, pp. 304–342, Jul. 2010. [Online]. Available: http://dx.doi.org/10.1109/SURV.2010.042710.00114

[27] R. Jain, D. Chiu, and W. Hawe, "A Quantitative Measure of Fairness and Discrimination for Resource Allocation in Shared Computer Systems," DEC Research, Technical report TR-301, Sep. 1984.

[28] D. A. Hayes, S. Ferlin, and M. Welzl, "Practical passive shared bottleneck detection using shape summary statistics," in *2014 IEEE 39th Conference on Local Computer Networks (LCN)*. IEEE, 2014, pp. 150–158.

[29] N. Khademi, D. Ros, M. Welzl, Z. Bozakov, A. Brunstrom, G. Fairhurst, K.-J. Grinnemo, D. Hayes, P. Hurtig, T. Jones, S. Mangiante, M. Tüxen, and F. Weinrank, "NEAT: A Platform- and Protocol-Independent Internet Transport API," *IEEE Communications Magazine*, vol. 55, no. 6, Feb. 2017.