Adaptive Cheapest Path First Scheduling in a Transport-Layer Multi-Path Tunnel Context

Marcus Pieska
Alexander Rabitsch
firstname.lastname@kau.se
Computer Science Department
Karlstads Universitet, Karlstad
Sweden

Anna Brunstrom
Andreas Kassler
firstname.lastname@kau.se
Computer Science Department
Karlstads Universitet, Karlstad
Sweden

Markus Amend
markus.amend@telekom.de
Deutsche Telekom
Darmstadt, Germany

ABSTRACT

Bundling multiple access technologies increases capacity, resiliency and robustness of network connections. Multi-access is currently being standardized in the ATSSS framework in 3GPP, supporting different access bundling strategies. Within ATSSS, a multipath scheduler needs to decide which path to use for each user packet based on path characteristics. The *Cheapest Path First* (CPF) scheduler aims to utilize the cheapest path (e.g. WiFi) before sending packets over other paths (e.g. cellular). In this paper, we demonstrate that using CPF with an MP-DCCP tunnel may lead to sub-optimal performance. This is due to adverse interactions between the scheduler and end-to-end and tunnel congestion control. Hence, we design the Adaptive Cheapest Path First (*ACPF*) scheduler that limits queue buildup in the primary bottleneck and moves traffic to the secondary path earlier. We implement ACPF over both TCP and DCCP congestion controlled tunnels. Our evaluation shows that ACPF improves the average throughput over CPF between 24% to 86%.

CCS CONCEPTS

• **Networks** → **Transport protocols**; *Mobile networks.*

KEYWORDS

Heterogeneous Wireless Access, 5G, ATSSS, Multi-Path, Unreliable Traffic, MP-DCCP, Transport Layer

ANRW '21, July 24–30, 2021, Virtual Event, USA

ACM ISBN 978-1-4503-8618-0/21/07…$15.00
https://doi.org/10.1145/3472305.3472316

ACM Reference Format:
Marcus Pieska, Alexander Rabitsch, Anna Brunstrom, Andreas Kassler, and Markus Amend. 2021. Adaptive Cheapest Path First Scheduling in a Transport-Layer Multi-Path Tunnel Context. In *Applied Networking Research Workshop (ANRW '21), July 24–30, 2021, Virtual Event, USA.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3472305.3472316

1 INTRODUCTION

Fifth generation cellular networks (5G) aim to support multi-access communication by using multiple wireless technologies through *Access Traffic Steering, Switching, and Splitting* (ATSSS, see 3GPP Rel. 16 [2]). Efficient bundling under a common framework can increase the capacity, resiliency and robustness of wireless networks. ATSSS supports flexible traffic steering and transparent capacity aggregation of multiple access networks. In this architecture, end-to-end TCP connections can be split at an anchor, transformed into *Multipath TCP* (MPTCP) [9] and routed over multiple wireless access networks between anchor and user. Unreliable traffic can instead be sent over *multi-path tunnels* [1] using novel multipath protocols that do not enforce strict reliability, e.g. *Multipath DCCP* (MP-DCCP) [4] or *Multipath QUIC* (MP-QUIC) [21] with the datagram extension [19].

Transparent and flexible *path aggregation* and prioritization of different access technologies is an important goal for ATSSS. Aggregation requires a packet scheduler, which determines the path for each packet. Ideally, such a scheduler should fully aggregate the available capacity of all access networks, which is difficult due to different path characteristics such as latency or available capacity. As ATSSS aims to offer cellular users aggregation of *non-trusted* wireless access, schedulers that express a preference for one or the other become desirable. Many cellular users would benefit from off-loading traffic onto WiFi whenever possible, and thereby save their limited cellular tariff — a trade-off which is also attractive to telecom operators. A scheduler may prioritize paths based on different criteria [5], including delay or *economic* aspects like *cost*. In that context, the *strict priority*

scheduler [5] aims to send packets over the primary path until the capacity is exhausted.

In this paper we:

(i) show that using CPF in a multipath congestion controlled tunneling framework leads to sub-optimal performance both when using DCCP and TCP-based tunnels. The main reason is the detrimental interaction between the scheduler, the per path congestion control algorithms and the end-host congestion controller.
(ii) design and implement the novel *Adaptive Cheapest Path First* (ACPF) scheduler, which uses the secondary path earlier, when ACPF infers congestion on the primary path. ACPF effectively limits congestion in favor of utilizing the secondary path, when the secondary path is underutilized.
(iii) implement and integrate ACPF with BBR-like congestion control over individual tunnels in both the Linux kernel and a user-space framework.
(iv) evaluate ACPF and show that it copes more effectively with the presence of nested congestion control loops, and that ACPF solicit the end-to-end connection to maintains more packets in-flight.

The rest of the paper is structured as follows: Section 2 reviews related work. Section 3 illustrates why CPF behaves poorly when used with congestion controlled tunnels and presents the design of our new ACPF scheduler. Section 4 details the implementation of ACPF. Section 5 compares the performance of CPF and ACPF. The paper concludes in section 6 discussing future work.

2 RELATED WORK

Packet scheduling over multiple paths is an active area of research [5, 13, 23]. An early evaluation of MPTCP schedulers [18] revealed that (i) *head-of-line blocking* and (ii) *receive window blocking* reduce the performance, in particular when the delay over the paths is highly asymmetrical. The MinRTT scheduler [20] is the default MPTCP scheduler, and prioritizes the path with the shortest estimated RTT. Blocking Estimation-based MPTCP scheduler algorithm [7] improves upon MinRTT by better estimating the amount of in-flight packets over each path. Other algorithms include Earliest Completion First [16], Peekaboo [23], and DEMS [10].

Congestion control algorithms (CCA) tend to regulate the number of in-flight packets to be no more than the congestion window ($CWND$). TCP-NewReno [3] is an early CCA which increases $CWND$ with one packet per RTT until a loss is detected, at which point $CWND$ is reduced. TCP-Cubic [11] adjusts $CWND$ according to a cubic function, but reacts to loss much like NewReno. More recently, TCP-BBR [6] tracks the throughput and the minimum RTT. BBR may then utilize the path fully by having $CWND$ equal roughly two bandwidth-delay products (BDP). BBR has been implemented for DCCP under the name CCID5[1]. Estimating the BDP when using a CCA other than BBR requires tracking the minimum RTT_{min} and limiting $CWND$ such that the RTT does not exceed RTT_{min} [8]. Finding a CCA suitable for wireless is an ongoing effort [12].

Explicit congestion notification (ECN) [22] is a method whereby the *network* indicates the presence of congestion to the sending-side transport layer — *as opposed to have the sender infer the amount of congestion.* CoDel [17] aims to control congestion by dropping packets when a certain queue latency is reached, thus triggering the end-to-end congestion control to reduce its sending rate.

3 DESIGN, ISSUES, & SOLUTIONS

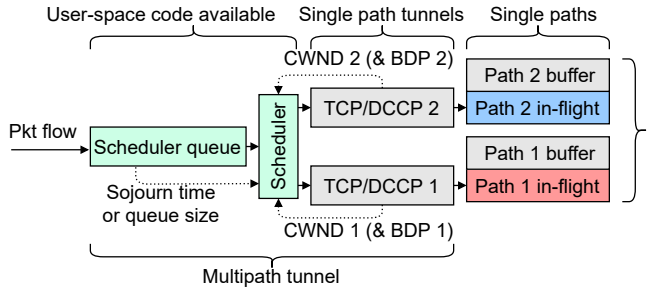
In the multi-access context, we assume p paths — *and single-path tunnels* — between the UE and the anchor, with each path having an independent congestion control regulating $CWND_p$ — *per Figure 2a.* Packets enter the multi-access framework via a scheduler queue, and for each packet a scheduler decides over which path to send it. At the anchor, packets leave the framework and are forwarded to the server. Thus, client and server applications need not to be modified to use the multi-access solution. Next, we review CPF and its performance problem in the multi-access context and illustrate the design rationale for ACPF.

3.1 Cheapest Path First

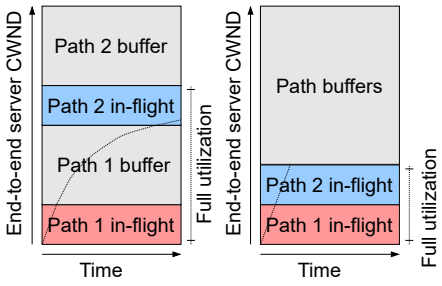
The *Cheapest Path First* (CPF, or *strict priority* [5]) packet scheduler ranks available paths according to cost in ascending order — *i.e., the cheapest path having highest priority, e.g. WiFi first, then Cellular.* The scheduler will consider path p *fully utilized* if the number of in-flight packets — *or bytes* — is equal to or greater than $CWND_p$. If this condition holds, path p will be marked as *not available* and the scheduler will proceed to schedule packets over lower priority path $p + 1$.

The problem with CPF is illustrated in Figure 1b, which shows how the server $CWND_s$ grows, and which parts of the multipath tunnel it occupies over time. The ideal outcome is to the right, whereas the outcome to the left is what tends to happen — *as shown in section 5.* As $CWND_s$ grows, $CWND_1$ tends to grow in parallel, leading CPF to schedule mostly on the primary path. This parallel growth can *delay*, or even *preclude* aggregation, depending on the server CCA. Most similar to Figure 1b is when the server uses NewReno; letting $CWND_s$ into the path 1 bottleneck buffer will *elongate* path 1, and slow down the $CWND_s$ growth. This is seen in Figure 1b as a decreasing growth rate of $CWND_s$ to the left. Ideally, the *size* of $CWND_1$ is such that $CWND_s$ first occupies path

[1] https://github.com/telekom/mp-dccp/tree/d9b4b7471a69184105885abbc4d45c011b82543d/net/dccp/ccids

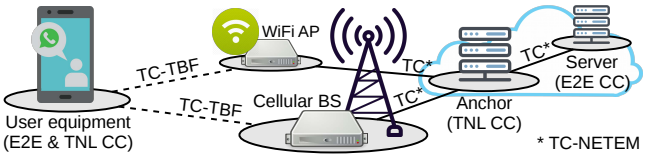


(a) Basic ACPF design; note the tunnel & path terminology.



(b) CPF issue to the left & ACPF solution to the right

Figure 1: Fig. a) High level ACPF illustration. Green modules are available as both user-space code and kernel code. Fig. b) Server (end-to-end) $CWND_s$ growth, and where over each path the $CWND_s$ is distributed over time.



(a) Topology; TBF sets throughput; NETEM sets delay.

```
manage_cwnd_frac (path p, sched s)
    QUEUE_SIZE_THRESHOLD = 10, RAW_INC = 0.05
    MIN_FRAC = 0.47, MAX_FRAC = 1.0
    if (s.queue_size > QUEUE_SIZE_THRESHOLD)
        INC = get_increase_inc (RAW_INC, p, s)
        p.cw_frac = MIN(p.cw_frac + INC, MAX_FRAC)
    else
        INC = get_decrease_inc (RAW_INC, p, s)
        p.cw_frac = MAX(p.cw_frac - INC, MIN_FRAC)
```

(b) Pseudo code for generic ACPF management.

```
CONSECUTIVE_INCREASE = 0, C1 = 10, C2 = 50

get_increase_inc (RAW_INC, path p, sched s)
    qsf = s.queue_size / QUEUE_SIZE_THRESHOLD
    cif = ++CONSECUTIVE_INCREASE / C1
    return RAW_INC / C2 * MIN(qsf, cif)^2

get_decrease_inc (RAW_INC, path p, sched s)
    CONSECUTIVE_INCREASE = 0
    return RAW_INC / C2
```

(c) Pseudo code for user-space ACPF management.

Figure 2: Multi-path context & ACPF logic.

1, then path 2, *and lastly* the the bottleneck buffers. This is our goal with ACPF, as seen to the right.

3.2 Adaptive Cheapest Path First

The main problem with CPF is that packets are sent over the cheaper path when they *should* be sent over the secondary path. The former will *reduce throughput* and *increase latency*, while the opposite is true for the latter. CoDel and ECN could be used at the bottleneck and would shift packets onto the secondary path by *shrinking* $CWND_1$. The main idea of ACPF is to use the secondary path *earlier*, when *congestion can be inferred* on the primary path and the sender still wants to increase its sending rate. The goal is to avoid occupying bottleneck buffers until the in-flight capacity (i.e. the BDP) have been reached on all paths. To achieve that, the amount of packets that can be scheduled over any path p is limited to a fraction of $CWND_p$, ideally *corresponding to the BDP* of path p. We denote $CWND_p$ as *raw*, and the fraction as the *live* $CWND_p$. The fraction is *periodically decreased* if the number of packets in the scheduling queue is below a given limit, and *vice versa*. This results in an *initial shallow occupancy* of the primary path buffer along with an earlier utilization of the secondary path. This is *not* congestion control, as the congestion reduction is likely often temporary. This key management is illustrated in Figure 1a, with the inputs being the scheduling queue occupancy and $CWND_p$.

Using TCP-BBR (or DCCP-CCID5) as tunnel CCA is particularly helpful to ACPF. BBR periodically drains the path to estimate the minimum RTT (RTT_{min_p}), while it will continuously probe for the throughput (BW_p). It will then set $CWND_p$ according to equation 1 where G is a gain typically set to 2. Thus, since the BBR $CWND$ is based on the BDP, ACPF may adjust how deeply the bottleneck buffer is filled by adjusting the live $CWND_p$ in equation 1 with a factor AG. Since the gain G is 2, setting AG to 0.5 will reduce the bottleneck buffer occupancy to near zero, while a factor of 1 will allow for the standard BBR buffer occupancy of one BDP. A important feature of BBR is that we know *approximately* what the *minimum gain* should be to achieve aggregation *without* needlessly splitting flows. Without this, ACPF would have no lower limit on AG, and if the live $CWND_p$ were to become smaller than the BDP *the flow would be split needlessly*. Figure 2b contains pseudo code which illustrates how to manage AG when working with BBR by *inferring* that the BDP should be roughly half the $CWND_p$.

$$CWND_p = RTT_{min_p} * BW_p * G \quad (1)$$

4 IMPLEMENTATION

We implemented ACPF *on top* of CPF as a management function *based on* Figure 2b. Each path has its own $CWND_p$ fraction, which is kept synchronized. A *queue size threshold* of 10 packets is used to determine the direction of the $CWND_p$ fraction adjustment; a queue size above will cause the live $CWND_p$ to grow and vice versa. The minimum fraction was determined *experimentally* to be 0.47. ACPF has been implemented in the Open Source MP-DCCP Linux Kernel reference implementation[2], as well as in a *protocol agnostic* user-space framework based on MP-DCCP.

4.1 Kernel-space ACPF

Kernel ACPF calls the $CWND_p$ management function (Figure 2b) to update the $CWND_p$ fractions on *all* paths, whenever a packet is scheduled onto any path. The rate at which the live $CWND_p$ fraction is adjusted per packet (`get_increase_inc` and `get_decrease_inc` in Figure 2b) is set according to equation 2. This allows the $CWND_p$ fraction to increase as needed even for small $CWND_p$, while remaining stable for larger $CWND_p$. When BBR enters the *ProbeRTT* state it is likely that the scheduler buffer occupancy temporarily increases as the server still sends at high rate. As the rate at which the $CWND_p$ fraction is adjusted is a function of the $CWND_p$, this can lead to an unnecessarily rapid growth of the $CWND_p$ fractions. The $CWND_p$ management is therefore temporarily disabled whenever BBR is in this state.

$$INC = RAW\_INC/CWND_p \tag{2}$$

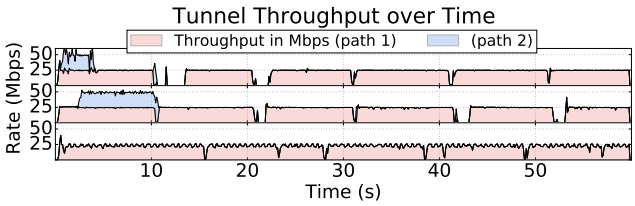
4.2 User-space ACPF

User-space ACPF will invoke its $CWND_p$ management function in a dedicated thread every 1 ms. Using the naming convention in Figure 2b, Figure 2c illustrates how the increase rate is scaled. The increment INC is scaled down by a factor $C_2 = 50$, leading to a *default* time required to bring the fraction from its minimum to its maximum of 530 ms. However, ACPF is designed to react more aggressively to a queue-buildup which is *persistent*, as opposed to transient. Given the 1 ms invocation interval, $C_1 = 10$ determines that any queue lasting for more than 10 ms is persistent. Without C_1 and C_2, the management becomes too aggressive.

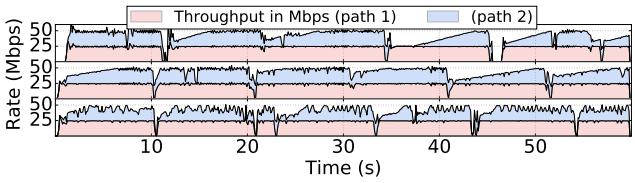
5 EVALUATION

We present results from both the user and kernel-space frameworks. We leverage the former for its ability to also use TCP, its extensive inbuilt tracing, and the ease of running large sets of test. Using TCP-BBR as tunneling CCA is attractive since it is more mature than its DCCP counterpart.

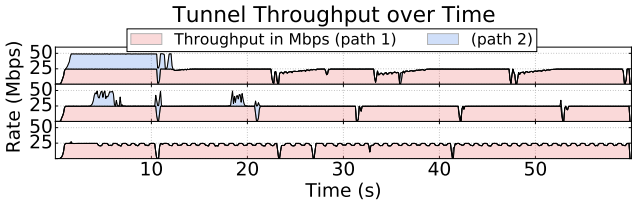
[2]https://multipath-dccp.org



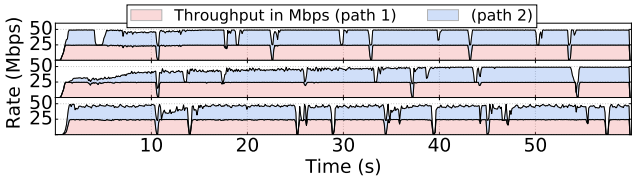
(a) CPF, short RTT, E2E CCA (from top): NewReno; Cubic; BBR.



(b) ACPF, short RTT, E2E CCA (from top): NewReno; Cubic; BBR.



(c) CPF, long RTT, E2E CCA (from top): NewReno; Cubic; BBR.

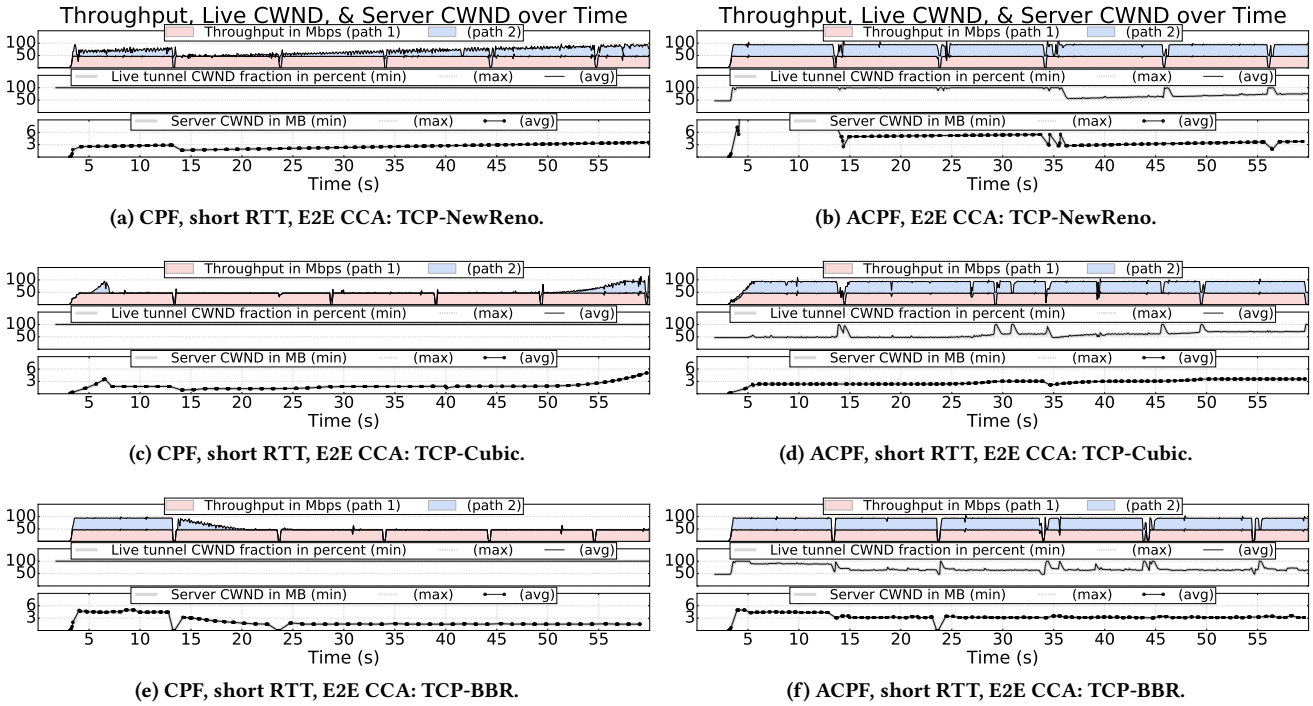


(d) ACPF, long RTT, E2E CCA (from top): NewReno; Cubic; BBR.

Figure 3: Kernel-space comparison of CPF & ACPF.

5.1 Evaluation setup

We used Mininet for our evaluation with the topology in Figure 2a. In each case, values in parenthesis are for user-space tests. We set the bottleneck *capacity* for both paths to 25 (50) Mbps. DCCP-CCID5 (TCP-BBR) is used as tunneling protocol. TC-TBF [14] is used to limit the capacity, and TC-Netem [15] is used to emulate propagation delay. This setup allow us to vary the anchor deployment. As a default, we set a 4 ms *latency* between anchor and server, and 16 and 26 ms over the primary and secondary path, respectively — our *short* scenario. We also use a *long* RTT scenario where the aforementioned delays instead are 8 ms, 32 ms, and 42 ms. Both cases result in a path *latency asymmetry* of 10 ms. The bottleneck buffer is large enough to not overflow from BBR. TCP-Iperf is used to generate a *greedy* flow.



(a) CPF, short RTT, E2E CCA: TCP-NewReno.

(b) ACPF, E2E CCA: TCP-NewReno.

(c) CPF, short RTT, E2E CCA: TCP-Cubic.

(d) ACPF, short RTT, E2E CCA: TCP-Cubic.

(e) CPF, short RTT, E2E CCA: TCP-BBR.

(f) ACPF, short RTT, E2E CCA: TCP-BBR.

Figure 4: User-space comparison of CPF (left) & ACPF (right) for different end-to-end CCAs.

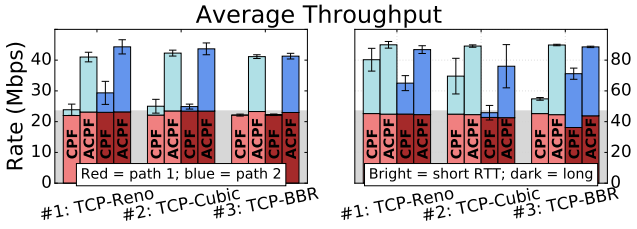
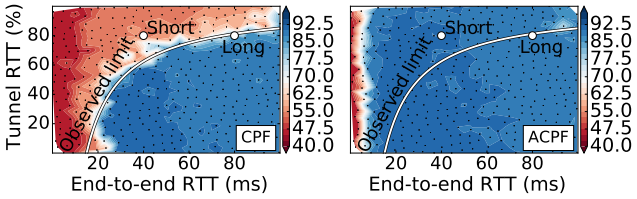


Figure 5: Kernel-space (left) & user-space (right).

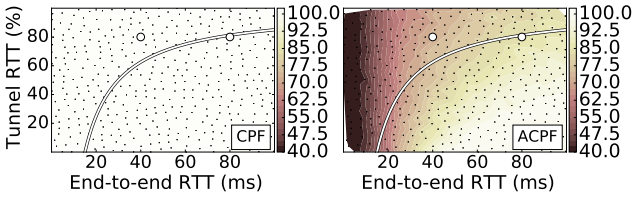
5.2 Overview evaluation

Figures 3a to 3d show throughput obtained from the kernel-space framework. CCID5 is used as the tunnel CC in all cases, while TCP-NewReno, TCP-Cubic, or TCP-BBR is used end-to-end. The full set of tests is repeated for a short and long RTT. The path throughput is *stacked*, and should ideally reach approximately the *sum of both paths*. Each result was picked in a representative way from a set containing 5 repetitions, and illustrate that ACPF improves aggregation over CPF in *each* scenario. The average throughput for each set is depicted in Figure 5, with an advantage to ACPF in all scenarios, ranging from 69% - 86% improvement in throughput.

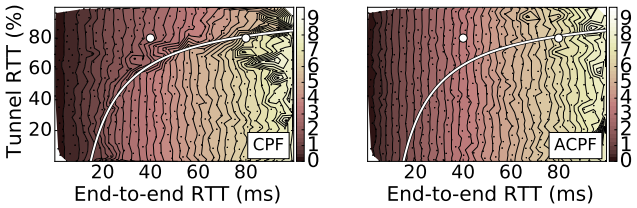
Figures 4a to 4f further illustrate why ACPF is able to achieve better aggregation than CPF. Each Figure depicts the result for a single *user-space* test for *3 parameters* over time: (i) *throughput*; (ii) *live CWND*; (iii) *server CWND*. TCP-BBR is used in the tunnel in all cases, but the test is otherwise the same as previous. Figures 4a, 4c, and 4e, show low aggregation for CPF. Figures 4b, 4d, and 4f show good aggregation for ACPF. These cases were *carefully selected* from a set of 10 repetitions each and represent those repetitions where default CPF performs *particularly bad*. Note that when the server uses TCP-NewReno or TCP-Cubic, the *ProbeRTT* periods that occur every 10 seconds tend to be followed by a period of severe under-utilization of the secondary path. These periods clearly solicit an adverse reaction from the end-to-end CCA. One interesting case is shown in Figure 4b around the 35 second mark; the $CWND_p$ fraction crashed quite quickly to near its minimum value and $CWND_1$ *would* have accepted most of $CWND_s$ and thereby prevented aggregation, *had* the fraction been at 100% after this event. Also, the user-space tunnel cause TCP-Cubic to leave slow-start early, often before utilization of the secondary path and sometimes at the very beginning of the flow. The latter is a particularly bad outcome and causes a high standard deviation. The average throughput for each set is again depicted in Figure 5, with an improvement of ACPF in all scenarios ranging from 24% - 66%. One encouraging result is that ACPF reliably yields nearly perfect aggregation for all scenarios, with the TCP-Cubic exception.



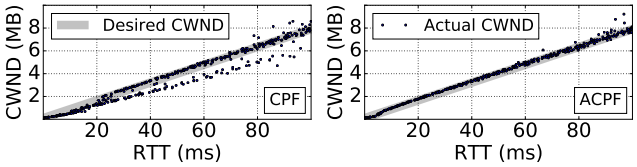
(a) Average throughput in Mbps.



(b) Average live $CWND_p$ fraction in percent.



(c) Average server $CWND_s$ in MB.



(d) Average server $CWND_s$ as a function of end-to-end RTT.

Figure 6: Outcome for CPF (left) & ACPF (right), for different RTT values & anchor deployment locations.

5.3 Detailed evaluation

The CPF issue was first discovered when tunneling BRR-over-BBR in the user-space framework via the process described here. For this, 400 points in a 100-by-100 square area are generated randomly, with the exception that any point *too close* to previously generated points is discarded and re-generated. The coordinates of these points are then translated into the end-to-end RTT in milliseconds, and the *percent* of that RTT which is within the tunnel — *as opposed to between anchor and server* — and are used as the configuration input for a test. The outcome is presented as a contour plot with each test marked, and interpolation between neighbouring tests.

Figure 6a illustrates the throughput for all tests. A good outcome is deep blue, whereas a deep red outcome indicates *no aggregation*. Both (i) a *short end-to-end RTT*, and (ii) an *anchor close to the server* will inhibit aggregation for CPF. The reason for the elusive aggregation appears to be that $CWND_1$ is *as large* as $CWND_s$. Since CPF will fill $CWND_1$ first, $CWND_s$ will have to be *larger* than $CWND_1$ if any traffic is to "spill over" onto the expensive path. It is particularly difficult for the server BBR to overcome the tunnel BBR in this way, as BBR will have its $CWND$ size remain close to 2 BDP. Hence the clear patterns in Figure 6.

Figure 6b depicts the *average* size of the live $CWND_1$ as a percent of the raw $CWND_1$. ACPF tends to have the *smallest* live $CWND_1$ where CPF was *least able to aggregate*, which illustrates that ACPF will not consistently deploy its offloading mechanism when it is not needed — or alternatively, that ACPF functions more like CPF, when CPF is sufficient.

Figure 6c suggests that ACPF will trigger the server CC to *perceive the second path*, and that the server therefore issues a *larger* $CWND_s$. Note how the topographic lines *veer to the right* above the white reference line for CPF. It *should* be the case that BBR maintain $CWND_s$ as *a function of the RTT*, but Figure 6d illustrates that this is *only* the case for ACPF. The noise to the left in Figure 6d is caused by the server CCA sometimes *losing track* of the secondary path, which causes BBR to perceive a different BDP — *resulting in the red regions in Figure 6a.* A similar analysis may be done for other CCA combinations, but is omitted for lack of space.

6 CONCLUSION AND FUTURE WORK

In this paper, we demonstrated a weakness of CPF, the reason behind this flaw, and how to overcome it. The solution presented is a modified version named ACPF that is able to achieve path aggregation when default CPF is unable, and thereby increase throughput by 24% - 86% in the scenarios studied. ACPF enable the end-to-end CC to *perceive* parallel paths *more as a single path.* We used *greedy* E2E flows to confirm that ACPF aggregates paths more or less as readily *as if* there was a single path by limiting *initial* usage of the cheapest path to 1 BDP. Whether aggregation is *actually desirable* for any given greedy flow is arguably a decision made by e.g. the user or by a management agent.

There should be additional benefits to ACPF, e.g. an ability to perform a *soft handover* from a degrading link without experiencing a temporary increase in delay. Of course, for such a switching scenario, ACPF may *needlessly* increase the utilization of the expansive path if it *underestimates* the BDP of the cheap path; a good BDP estimation is a prioritized goal. ACPF was developed in kernel-space MP-DCCP, as well as a user-space multipath framework. The former allow us to learn the BDP from the CC state, thus removing the need to *infer* the BDP. It would also be desirable to be able *contain bursts* on a single path to avoid using the secondary path too aggressively. These are left as future work.

REFERENCES

[1] 3GPP. 2021. *Study on Access Traffic Steering, Switch and Splitting support in the 5G system architecture Phase 2.* Technical Specification (TS) 23.700-93. 3rd Generation Partnership Project (3GPP). Version 2.0.0.

[2] 3GPP. 2021. *System architecture for the 5G System (5GS).* Technical Specification (TS) 23.501. 3rd Generation Partnership Project (3GPP). Version 17.0.0.

[3] M. Allman, V. Paxson, and W. Stevens. 1999. *TCP Congestion Control.* RFC 2581. RFC Editor.

[4] M. Amend, E. Bogenfeld, M. Cvjetkovic, V. Rakocevic, M. Pieska, A. Kassler, and A. Brunstrom. 2019. A Framework for Multiaccess Support for Unreliable Internet Traffic using Multipath DCCP. In *2019 IEEE 44th Conference on Local Computer Networks (LCN).* IEEE, Osnabrueck, Germany, 316–323. https://doi.org/10.1109/LCN44214.2019.8990746

[5] Olivier Bonaventure, Maxime Piraux, Quentin De Coninck, Matthieu Baerts, Christoph Paasch, and Markus Amend. 2020. *Multipath schedulers.* Internet-Draft draft-bonaventure-iccrg-schedulers-00. IETF Secretariat. https://www.ietf.org/archive/id/draft-bonaventure-iccrg-schedulers-00.txt https://www.ietf.org/archive/id/draft-bonaventure-iccrg-schedulers-00.txt.

[6] Neal Cardwell, Yuchung Cheng, C Stephen Gunn, Soheil Hassas Yeganeh, and Van Jacobson. 2016. BBR: Congestion-Based Congestion Control: Measuring bottleneck bandwidth and round-trip propagation time. *Queue* 14, 5 (2016), 20–53.

[7] Simone Ferlin, Özgü Alay, Olivier Mehani, and Roksana Boreli. 2016. BLEST: Blocking estimation-based MPTCP scheduler for heterogeneous networks. In *2016 IFIP Networking Conference (IFIP Networking) and Workshops.* IEEE, IEEE, Vienna, Austria, 431–439.

[8] Simone Ferlin-Oliveira, Thomas Dreibholz, and Özgü Alay. 2014. Tackling the challenge of bufferbloat in Multi-Path Transport over heterogeneous wireless networks. In *2014 IEEE 22nd International Symposium of Quality of Service (IWQoS).* IEEE, Hong Kong, China, 123–128. https://doi.org/10.1109/IWQoS.2014.6914310

[9] A. Ford, C. Raiciu, M. Handley, O. Bonaventure, and C. Paasch. 2020. *TCP Extensions for Multipath Operation with Multiple Addresses.* RFC 8684. RFC Editor.

[10] Yihua Ethan Guo, Ashkan Nikravesh, Z. Morley Mao, Feng Qian, and Subhabrata Sen. 2017. Accelerating Multipath Transport Through Balanced Subflow Completion. In *Proceedings of the 23rd Annual International Conference on Mobile Computing and Networking* (Snowbird, Utah, USA) *(MobiCom '17).* Association for Computing Machinery, New York, NY, USA, 141–153. https://doi.org/10.1145/3117811.3117829

[11] Sangtae Ha, Injong Rhee, and Lisong Xu. 2008. CUBIC: A New TCP-Friendly High-Speed TCP Variant. *SIGOPS Oper. Syst. Rev.* 42, 5 (July 2008), 64–74. https://doi.org/10.1145/1400097.1400105

[12] Habtegebreil Kassaye Haile, Karl-Johan Grinnemo, Simone Ferlin, Per Hurtig, and Anna Brunström. 2021. End-to-end congestion control approaches for high throughput and low delay in 4G/5G cellular networks. *Computer Networks* 186 (2021), 1–22.

[13] Per Hurtig, Karl-Johan Grinnemo, Anna Brunstrom, Simone Ferlin, Özgü Alay, and Nicolas Kuhn. 2019. Low-Latency Scheduling in MPTCP. *IEEE/ACM Transactions on Networking* 27, 1 (2019), 302–315. https://doi.org/10.1109/TNET.2018.2884791

[14] Michael Kerrisk. 2001. *tc-tbf(8) — Linux manual page.* Linux man-pages project. https://man7.org/linux/man-pages/man8/tc-tbf.8.html

[15] Michael Kerrisk. 2011. *tc-netem(8) — Linux manual page.* Linux man-pages project. https://man7.org/linux/man-pages/man8/tc-netem.8.html

[16] Yeon-sup Lim, Erich M. Nahum, Don Towsley, and Richard J. Gibbens. 2017. ECF: An MPTCP Path Scheduler to Manage Heterogeneous Paths. In *Proceedings of the 13th International Conference on Emerging Networking EXperiments and Technologies* (Incheon, Republic of Korea) *(CoNEXT '17).* Association for Computing Machinery, New York, NY, USA, 147–159. https://doi.org/10.1145/3143361.3143376

[17] Kathleen Nichols and Van Jacobson. 2012. Controlling Queue Delay. *Commun. ACM* 55, 7 (July 2012), 42–50. https://doi.org/10.1145/2209249.2209264

[18] Christoph Paasch, Simone Ferlin, Ozgu Alay, and Olivier Bonaventure. 2014. Experimental Evaluation of Multipath TCP Schedulers. In *Proceedings of the 2014 ACM SIGCOMM Workshop on Capacity Sharing Workshop* (Chicago, Illinois, USA) *(CSWS '14).* Association for Computing Machinery, New York, NY, USA, 27–32. https://doi.org/10.1145/2630088.2631977

[19] Tommy Pauly, Eric Kinnear, and David Schinazi. 2021. *An Unreliable Datagram Extension to QUIC.* Internet-Draft draft-ietf-quic-datagram-02. Internet Engineering Task Force. https://datatracker.ietf.org/doc/html/draft-ietf-quic-datagram-02 Work in Progress.

[20] Costin Raiciu, Christoph Paasch, Sebastien Barre, Alan Ford, Michio Honda, Fabien Duchene, Olivier Bonaventure, and Mark Handley. 2012. How Hard Can It Be? Designing and Implementing a Deployable Multipath TCP. In *9th USENIX Symposium on Networked Systems Design and Implementation (NSDI 12).* USENIX Association, San Jose, CA, 399–412. https://www.usenix.org/conference/nsdi12/technical-sessions/presentation/raiciu

[21] T. Viernickel, A. Froemmgen, A. Rizk, B. Koldehofe, and R. Steinmetz. 2018. Multipath QUIC: A Deployable Multipath Transport Protocol. In *2018 IEEE International Conference on Communications (ICC).* IEEE, Kansas City, MO, USA, 1–7. https://doi.org/10.1109/ICC.2018.8422951

[22] M Welzl and W Eddy. 2010. *Congestion control in the RFC series.* Technical Report. RFC 5783, February.

[23] Hongjia Wu, Özgü Alay, Anna Brunstrom, Simone Ferlin, and Giuseppe Caso. 2020. Peekaboo: Learning-based multipath scheduling for dynamic heterogeneous environments. *IEEE Journal on Selected Areas in Communications* 38, 10 (2020), 2295–2310.