Experimenting with Ledbat++: Fairness, Flow Scalability and LBE Compliance

Ioanna Angeliki Kapetanidou*†, Ioannis Zacharis*, Marios Kostopoulos* and Vassilis Tsaoussidis*†
*Democritus University of Thrace, Xanthi, Greece
†ATHENA Research and Innovation Center, Xanthi, Greece
Email: {ikapetan, ioanzach2, marikost9, vtsaousi}(at) ee.duth.gr

***Abstract*—The Ledbat++ protocol has been recently introduced as the evolution of Ledbat, a congestion control algorithm for background traffic. Ledbat++ encompasses improved congestion control mechanisms to mitigate inherent Ledbat issues stemming mainly from falsely estimated delays and resulting in degraded inter-flow fairness or undesired aggressiveness. However, the performance evaluation of Ledbat++ remains rather limited to date. In this paper, we introduce an ns-3-based Ledbat++ implementation which allows us to evaluate for the first time the protocol performance under realistic conditions with an increasing number of competing flows. To better assess the protocol's impact on regular TCP traffic we consider well-suited scenarios including competition with various congestion control algorithms. Our findings indicate that Ledbat++ satisfies its design goals in the sense that it solves the problems original Ledbat is inherently susceptible to, but also efficiently backs off and allows TCP flows to prevail, serving the 'less-than-best-effort' principle. We also found that Ledbat++ flows achieve throughput close to its competing flows only when competing with other bulk transfer-oriented algorithms, such as TCP Low Priority.**

***Index Terms*—Congestion Control, Ledbat, Ledbat++, ns-3**

I. Introduction

The Ledbat protocol, standing for Low Extra Delay Background Transport, is a 'less-than-best-effort' (LBE) congestion control algorithm, originally designed to handle background bulk data transfers. Ledbat monitors the queueing delay and interprets its increase above a prespecified target as a warning for forthcoming congestion. It then responds by backing off and yielding bandwidth to the competing TCP flow. However, Ledbat has been found to be inherently plagued by issues such as the unfair sharing of resources between Ledbat flows, inaccurate delay estimations, and exhibiting unwanted aggressive competing behaviors in networks involving small buffers.

To address these issues, Ledbat++ was recently introduced [1]. Ledbat++ improves the original Ledbat algorithm by incorporating enhanced congestion control mechanisms, such as dynamic gain values, multiplicative congestion window decreases, periodic slowdowns, and others. These mechanisms allow for overcoming the issues identified in Ledbat, ensuring reliable delay measurements, and restoring inter-flow fairness.

While Ledbat++ shows promise, experimentation with the protocol is still at an early stage. Currently, only a few works have experimented with Ledbat++, either over real-world deployments that involve Ledbat++-featured Microsoft Windows Servers [2], [3] or by testing its performance in emulated network environments using real-world traces [4]. Even though these evaluation endeavors provide useful insights, they are rather tied by the experimentation environments' constraints.

Considering this, our paper advances beyond the state-of-the-art by being the first work to evaluate Ledbat++ performance considering aspects such as a varying number of competing flows as well as the protocol's interoperability with regular TCP flows through scenarios in which Ledbat++ shares the same bottleneck with a variety of congestion control algorithms. To this end, we adopt an experimental methodology aligned with the respective followed in prior acknowledged works [2], [5]–[7].

Overall, our paper focuses on the following contributions:

- We have implemented Ledbat++ in the widely used ns-3 network simulator. Our implementation is released as open-source aiming to foster broader experimentation with Ledbat++.
- We show that Ledbat++ indeed resolves the innate Ledbat issues by juxtaposing their performance measurements.
- We perform a variety of experiments with diverse network configurations, including an increasing number of competing flows to assess also scalability; this aspect has not been considered in previous works.
- To date, Ledbat++ has only been compared to the CUBIC and Bottleneck Bandwidth and Round-trip propagation time (BBR) algorithms. Our study offers a more comprehensive evaluation testing its performance when competing with a wide range of congestion control protocols.

The rest of the paper is organized as follows: In Section II we outline the design of Ledbat++ and we review related research works. In Section III, we introduce our implementation and present briefly our evaluation methodology, while experiments' specifics along with extensive results are provided in Section IV-B. Finally, we summarize our concluding remarks in Section V.

II. Background & Related work

A. Ledbat++ protocol

The Ledbat++ algorithm has been proposed to address vulnerabilities identified in the original Ledbat protocol. Such issues include: (i) the *'Latecomer advantage'*, i.e., newcomers perceive the queuing delay of existing connections as their

This work has been supported by Athena Research Center

ISBN 978-3-903176-63-8 © 2024 IFIP

baseline delay and ramp up their transmission rate quickly, monopolizing the bandwidth; (ii) the *'Inter-Ledbat fairness'* issue, referring to the persistently unfair share of resources among competing Ledbat flows, stemming from the additive increases and decreases in Ledbat's congestion window update mechanism; (iii) the *'Latency drift'* caused by the estimation of the base delay based on fixed 10-minute intervals, rather than connection's actual duration, leading to likely obsolete delay estimations; (iv) the *'Low Latency Competition'* problem in networks with small buffers wherein queuing delays may never reach the predetermined target delays, leading Ledbat connections to aggressively compete for bandwidth like regular TCP connections, and (v) the *'Dependency on One-way Delay Measurements'* that renders Ledbat susceptible since TCP timestamps lack standardization in clock synchronization, requiring heuristics to compensate for clock skew.

To resolve these problems, Ledbat++ introduces several improvements across the congestion control phases. First of all, it relies on a *modified slow start mechanism* that monitors whether the queueing delay exceeds the $\frac{3}{4}$ of target delay, and if so, immediately reverts to congestion avoidance to avoid congestion spikes. This practice is germane solely to the initial slow start. As per congestion avoidance, Ledbat++ considers *dynamic gain values* to the congestion window increases, calculated in such a way that ensures that Ledbat++ connections ramp up slower than regular TCP connections, thus solving the low latency competition issue. This is coupled with gain-based *multiplicative decreases* that ensure inter-flow fairness. Moreover, to achieve stability, Ledbat++ introduces periodic *slowdowns*, i.e., intervals during which the Ledbat++ connections voluntarily freeze their congestion windows to 2 packets, allowing queues to drain and delay measurements to converge to the base delay, before entering the slow start phase again. By combining slowdowns with multiplicative decreases, Ledbat++ solves the latecomer advantage and the Inter-Ledbat fairness problems. Finally, Ledbat++ relies on round-trip measurements and sets the target delay to $60ms$ to overcome the limitations of one-way delay estimations.

B. Related works

The issues faced by the legacy Ledbat protocol are long known [8], [9] and various countermeasures have been investigated [10]. Ledbat++, standardized by IRTF in 2020 [1], incorporates novel features that mitigate the vulnerabilities of Ledbat and has even been deployed in production environments, being part of Microsoft software, since Windows Server 2016 and onwards[1].

Building upon this publicly available Ledbat++ implementation, authors in [2] provided a thorough experimental evaluation of the Ledbat++ congestion control algorithm under various network settings, via including one Windows 2019 Server that features Ledbat++ in their experimental setup. This is the first concrete endeavor to study the performance of Ledbat++ experimentally. In this work, they also conducted a performance evaluation in scenarios wherein Ledbat++ competes with CUBIC flows. Authors have also extended this initial evaluation to scenarios wherein Ledbat++ is competing with Google's BBR congestion control algorithm [3]. They observed that Ledbat++, when competing with CUBIC in small-buffer networks or with BBRv2 in low-latency networks, is still rather aggressive, and indicated using fixed target delay values as a potential reason for this. Motivated by this observation, [4] leveraged reinforcement learning techniques to adaptively adjust the target delay based on measured network conditions. Authors also use the mahimahi network emulator[2], to test Ledbat++'s performance over real-world traffic.

As obvious, experimenting with Ledbat++ constitutes a rather recent research direction. Prior works undoubtedly provide a lot of useful insights, but they are conducted over on-premise, fixed experimental setups that are not pertinent to support large-scale experiments. Aiming to bridge this gap and enable easily configurable, 'large-scale'-capable experimentation we implement Ledbat++ in ns-3. By doing so, we are able to also provide an extensive evaluation over diverse network conditions and also to evaluate Ledbat++'s performance when sharing the same bottleneck link with several TCP algorithms.

III. Implementation & Experimental Methodology

A. Ledbat++ Implementation Overview

For our evaluation purposes, we have implemented the Ledbat++ protocol over the ns-3 simulator. To this end, we used the implementation of the legacy Ledbat model as a basis but made significant revisions, adding new functionalities or modifying the existing ones, to incorporate the Ledbat++ features. More specifically, our implementation supports: (i) Ledbat++'s modified slow start that is adaptive to the difference between the target delay and the queueing delay, and explicitly during the initial slow start, allows exiting in cases of excessive delay, (ii) dynamic gain values, (iii) multiplicative decrease, (iv) slowdowns, and (v) using round-trip measurements instead of one-way delays.

To utilize our Ledbat++ model, ns-3 users need only to define *'TcpLedbatpp'* as the *"ns3::TcpL4Protocol::SocketType"* attribute. We have also enriched the code with relevant log messages that allow for tracking the protocol's transition from phase to phase when enabling ns-3 logging at the info level.

Our implementation has been realized in ns-3 v. 3.37. It has been released as open-source, along with stepwise instructions on its integration with ns-3 and is available at https://github.com/IoannaAngelikiKapetanidou/ns3-tcp-ledbatpp. That way, we allow for the reproducibility of our experiments but, most importantly, we hope that our work will facilitate experimentation with Ledbat++.

B. Methodology overview

The aim of our evaluation is twofold: firstly, to validate that Ledbat++ outperforms Ledbat and ensures fairness under

[1]https://techcommunity.microsoft.com/t5/networking-blog/ledbat-background-data-transfer-for-windows/ba-p/3639278

[2]https://manpages.ubuntu.com/manpages/focal/man1/mahimahi.1.html

diverse network conditions, and, secondly, to determine the inter-protocol priority, i.e., how the LBE Ledbat++ interacts with competing regular TCP flows.

To this end, we perform two respective sets of experiments: For the first goal to be satisfied, we consider a simple, single bottleneck scenario (Fig. 1a), yet involving various network configurations to illustrate the algorithms' salient features.

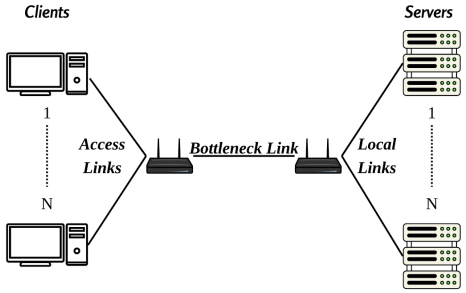
More specifically, we begin with increasing the number of competing flows to investigate whether Ledbat++ efficiently mitigates the *latency drift* and *latecomer advantage* issues in different network scales. Moreover, we experiment with different values of link bandwidths and bottleneck delays. The latter results in different round-trip times (RTTs) and thus, implies different slowdown periods for Ledbat++.

In our second experiment, we investigate how the performance of regular TCP traffic flows is affected when sharing the bottleneck link with Ledbat++ flows. We employ 16 different TCP variants including standard TCP protocols, such as New Reno, Westwood+ and Vegas, algorithms tailored particularly to high-speed networks, such as CUBIC, Scalable, Illinois, and others, e.g. the increasingly popular BBR, but also TCP-Low-Priority (TcpLp) that, similarly to Ledbat, it is designed for low-priority traffic. We also consider Ledbat and Ledbat++ as forward flows.

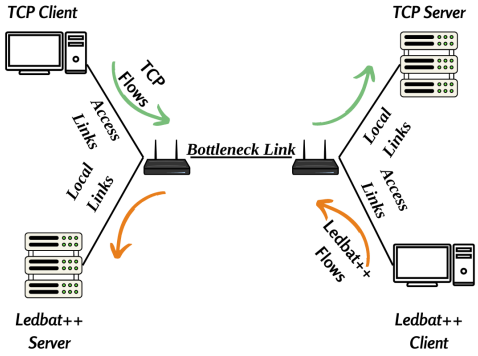
The evaluation proceeds with two distinct scenarios: We start with extending the initial topology to support reverse flows (Fig. 1b). In this context, we consider both "full-on" and "on-off" flows, but also vary the number of competing flows, in two distinct cases: First, we gradually increase the number of reverse flows that a single forward flow has to compete with. Then, we experiment with an increasing number of forward flows competing with 4 reverse flows. By varying the ratio of Ledbat++ and TCP flows to better assess the protocols' mutual influence, as discussed in [6], [7]. All in all, such a methodology is particularly suited for both highlighting the protocols' dynamics and investigating the inter-protocol impact [5].

In the second scenario, we revert to a multi-hop, multi-bottleneck topology (Fig. 1c), to better evaluate the tendency of Ledbat++ to yield bandwidth to competing TCP algorithms [5], [11].

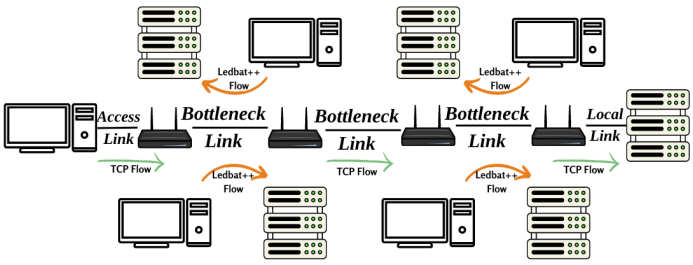
The experiments' setup details are thoroughly presented in the following section.



(a) Dumbbell Topology



(b) Single Bottleneck Topology with Reverse Flows



(c) Multi Bottleneck Topology with Cross Traffic

Fig. 1: Experimental Topologies

IV. Experiments

A. Basic Configuration

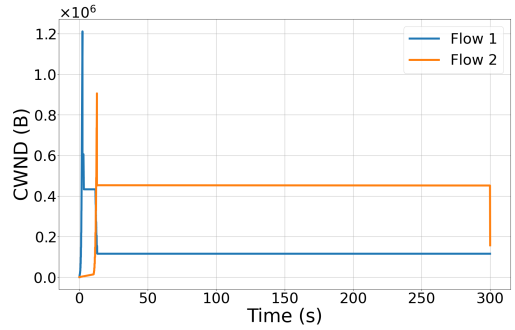
In all experiments, flows are initiated with a $10s$ interval. The maximum transmission unit (MTU) is set to $1400B$. The bottleneck link buffer size is equivalent to $\frac{size(B)}{mtu(B)}$ where $size(B)$ is computed like so:

$$\min(\text{access\_bw}, \text{bottle\_bw}) \times RTT_{delay} \quad (1)$$

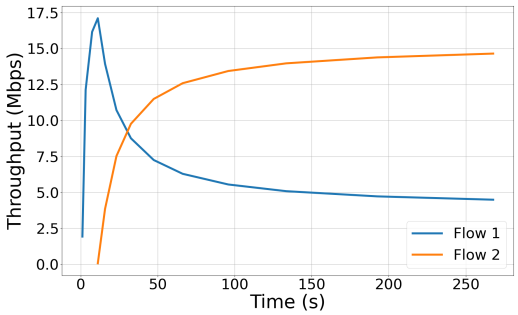
Each simulation lasts for 300 seconds. Further experimental setup details are described below.

B. Results

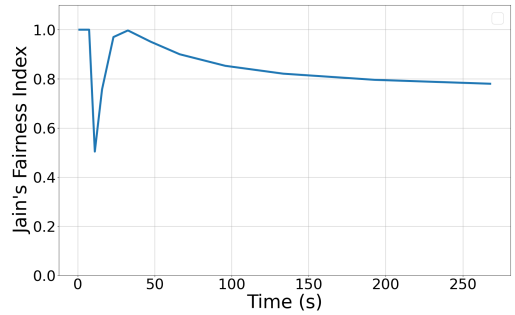
Experiment 1: Ledbat++ vs Ledbat:
In this experiment, we consider a dumbbell topology, as shown in Figure 1a, with $N = 2$ competing flows. This topology is deemed suitable for flow scalability testing [11].



(a) Congestion Window

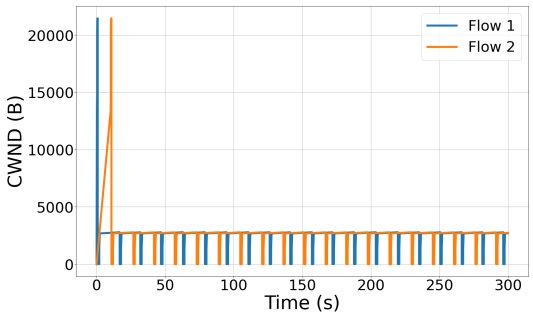


(b) Throughput

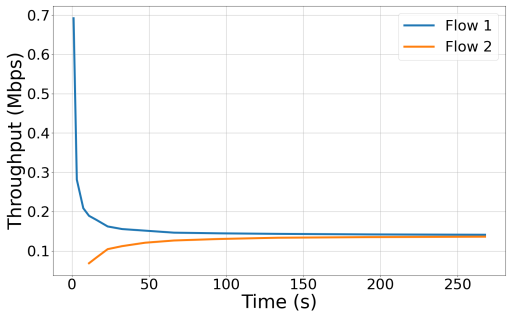


(c) Jain's Fairness Index

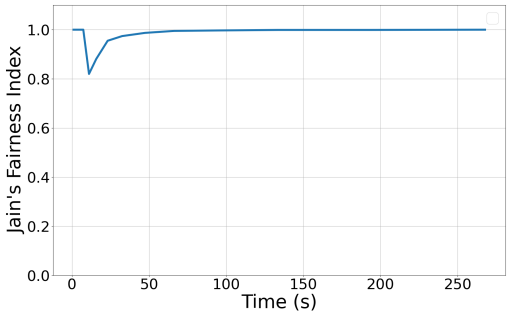
Fig. 2: Default Scenario: Ledbat



(a) Congestion Window



(b) Throughput



(c) Jain's Fairness Index

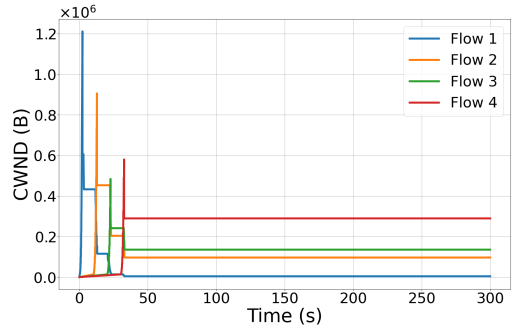
Fig. 3: Default Scenario: Ledbat++

Default Scenario: In the beginning, the topology is configured as follows: The access and local links bandwidth is set to $200Mbps$ with a corresponding delay of $10ms$. The flows share a bottleneck link with bandwidth set to $20Mbps$ and delay $50ms$.

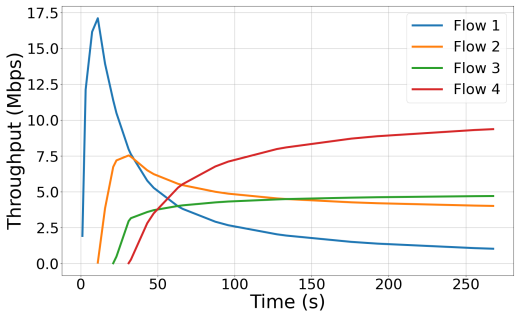
Associated results are depicted in Figures 2 and 3, for Ledbat and Lebdat++, respectively. To provide a comprehensive view of the algorithms' performance, we plot the flows' congestion windows, the per-flow throughput, and Jain's fairness index over time.

In Ledbat, the second flow monopolizes the network, reaching a significantly higher congestion window (Figs. 2a) and throughput (Figs. 3b) than the first. This results in decreased fairness values (Figs. 2c) and confirms that the legacy Ledbat protocol suffers from the *latecomer advantage* problem.

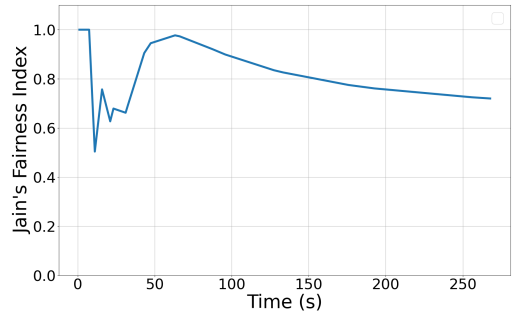
On the contrary, it becomes easily apparent that Ledbat++ handles the traffic generated by the two competing flows more fairly, maintaining an equal congestion window (Figs. 3a) and near throughput values (Figs. 3b). However, Ledbat++ flows fail to develop high throughput and, as a result, under-utilize the network capacity. It is important to remember though that the Ledbat family of LBE protocols is designed for background traffic, thereby meant not to occupy much of the network resources. The stability of the congestion window and its maintenance at low values is mainly attributed to the periodic slowdowns enforced by Ledbat++ that revert to start-up conditions, as a means to deal with the *latency drift* problem [1]. These slowdowns are visible in Figs. 3a, e.g. at $t = 94s$ for flow 1 and $t = 104s$ for flow 2. Still, the extent up to which Ledbat++ performance is harmed by the slowdowns



(a) Congestion Window

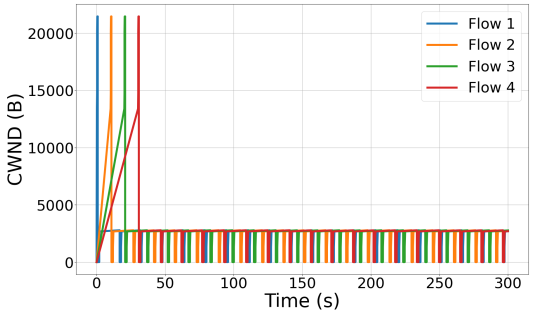


(b) Throughput

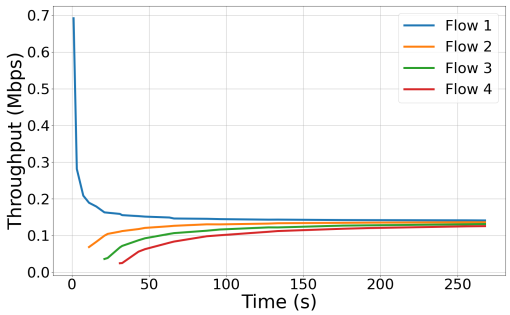


(c) Jain's Fairness Index

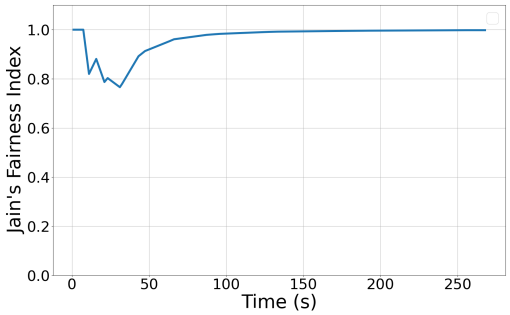
Fig. 4: Four Gradually Joining Flows: Tcp Ledbat



(a) Congestion Window



(b) Throughput



(c) Jain's Fairness Index

Fig. 5: Four Gradually Joining Flows: Tcp Ledbat++

relies on the base RTT and buffer size values [2], thus it can be mitigated. Overall, these slowdowns ensure that bandwidth is distributed fairly between the two flows. This is also validated by the fairness index (Figs. 3c) that drops only temporarily upon the entrance of the second flow, but recovers quickly to 1.0, achieving a fully fair split. Therefore, Ledbat++ solves the *latecomer advantage* issue at the cost of reduced efficiency [3].

In addition to the default scenario, we consider a few sub-experiments that allow for evaluating the performance of Ledbat++ in varying network configurations [2].

Varying number of competing flows: Our evaluation is extended to consider 4, 6 or 8 flows, each one being initiated $10s$ after its preceding flow. Due to space limitations, only the results for 4 flows are indicatively provided in detail in Figures 4 and 5. The results for 6 and 8 flows are incorporated in the next sub-experiment. As shown, the results for both Ledbat and Ledbat++ are consistent with the respective presented before for 2 flows. In other words, Ledbat allows latecomers to gain an advantage over already active flows, while Ledbat++ accomplishes maintaining a fair split among competitors regardless of their number.

Varying data rate: This time we experiment with different values of link bandwidths while varying the number of flows, too. The selected bandwidth values range from $2Mbps$, covering communications and services with low requirements, up to high-speed networks of $1Gbps$. The bottleneck data rate takes the values shown on the x-axis of Figures 6 and 7.

We focus on the congestion window sizes and the fairness

[3]Efficiency is defined as the ratio of available network capacity being utilized

index. Ledbat's behavior is not seriously affected by the data rate changes for the low bandwidth values, i.e., up to $50Mbps$, and fairness is kept to a satisfactory level. This insight is in line with the results of the experimental evaluation conducted at [12]. For larger bandwidths, larger congestion window sizes are measured but this does not reflect the performance of all competing flows. The difference between flows is amplified, especially as the number of competing flows increases, as inferred by the corresponding drop in the fairness indices.

Ledbat++, on the other hand, performs better for bandwidths up to $20Mbps$, exhibiting higher congestion window sizes and fairness indices. However, for bandwidth values ranging from $50Mbps$ to $200Mbps$, thus higher capacities, fairness is affected; especially for 2 or 4 flows. Such performance has been also observed for other LBE algorithms as well, in particular for Vegas and TcpLp, in prior studies [12].

Varying bottleneck delay: In this sub-experiment, we vary the bottleneck delay values to achieve different RTT measurements and consequently, slowdown periods for Ledbat++. There are two competing flows.

As inferred by the fairness calculated across the different bottleneck delay values, Ledbat is more prone to delay variations (Fig. 8a), with inter-flow fairness dropping as delay increases. Ledbat++ is almost oblivious to the delay, maintaining fairness always above 0.95 (Fig. 8b). The few outliers correspond to the time the second flow enters until the fairness approaches maximum again.

Experiment 2: Ledbat++ vs standard TCP algorithms: The evaluation is performed for high-speed networks, with bottleneck links equal to $1Gbps$. Due to space constraints, we plot the total throughput achieved by competing flows as a pertinent evaluation metric.

In this context, we evaluate performance in two distinct scenarios, each one corresponding to a different topology.

Scenario 1: In this scenario, we consider three sub-experiments performed over the topology shown in Figure 1b. We clarify that the reverse Ledbat++ flows, in all cases, reach throughput values nearly identical to the ones illustrated for the forward Ledbat++ flows.

Varying reverse flows activity: Initially, we consider 4 constant reverse flows which operate in an "on-off" fashion, similar to the approach taken in [5]. These reverse flows are active only during the intervals $[75, 150]$ and $[225, 300]$ to explore the effect of Ledbat++ reverse traffic.

The throughput measured for each TCP congestion control protocol over time when the reverse Ledbat++ flows are periodically activated and de-activated for $75s$ is illustrated in Figure 9. As indicated in Figure 9a, the majority of algorithms, Ledbat included, occupy most of the bandwidth irrespective of the existence of reverse flows. Vegas, due to its innate behavior aiming to maintain fairness, and TcpLp, being an LBE algorithm, does not achieve high throughput, regardless of the existence of the reverse flows. However, this behavior is observed mainly due to the high bandwidth availability. To better illustrate this, we repeat this experiment for bottleneck bandwidth equal to $100Mbps$ (Fig. 9b. This time, the activity of the reverse flows clearly affects the performance of the forward TCP flows; the achieved throughput is decreased upon their entrance at $75s$, while their dis-activation after $t = 150s$, allows for throughput to be increased again. This is particularly evident for Vegas, according to the inclination of its throughput line (the orange one), e.g. its increase is impeded in the interval $[75, 150]$ and easier afterward.

Varying bottleneck delay: Thereafter, we assume that traffic sources are always active (initiated with a $10s$ interval), while we vary the bottleneck delay. The throughput for 5 distinct bottleneck delay values is presented in Figure 10. It is worth noting that Ledbat is quite aggressive, achieving throughput similar to other TCP protocols, as illustrated, since the line of its measured throughput overlaps with the respective for regular TCP.

For small delays, restrained behavior is observed solely for TcpLp and Vegas. Especially TcpLp seems to be almost out-of-competition for delays below $100ms$, while it begins to come to the fore for delays set to $150ms$ and $200ms$. The competence with Vegas is also worth observing, since, similar to TcpLp, Vegas seems to be striving to maintain fairness even for low delays. In fact, for very low delays ($10ms$), TcpLp and Vegas are the only algorithms that allow the competing reverse Ledbat++ flow to gain a larger share of resources. The rest of the algorithms are rather aggressive, getting a clear advantage over the retreating reverse Ledbat++ flow.

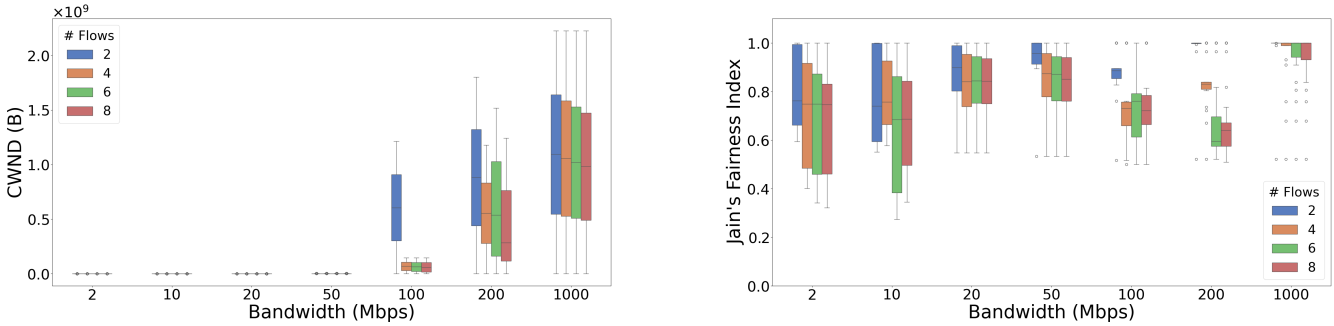
As expected, larger delays result in lower throughput values for all, Ledbat++ being no exception. Hence, the resource split becomes fairer, but this is mostly due to the network constraints, rather than the algorithms' behavior per se.

Varying number of competing flows: Finally, we vary the ratio of the competing flows. In this context, we present two rounds of sub-experiments. We start off by increasing the number of the reverse Ledbat++ flows (Figure 11a). We then set the number of reverse flows to be constantly 4 and vary the number of forward flows to either be half or equal reverse flows (Figure 11b).

Increasing reverse flows: We increase the number of reverse flows to 4, 8, 50 and 100 flows gradually joining flows. The joining interval is $1s$ so that all reverse flows join early.

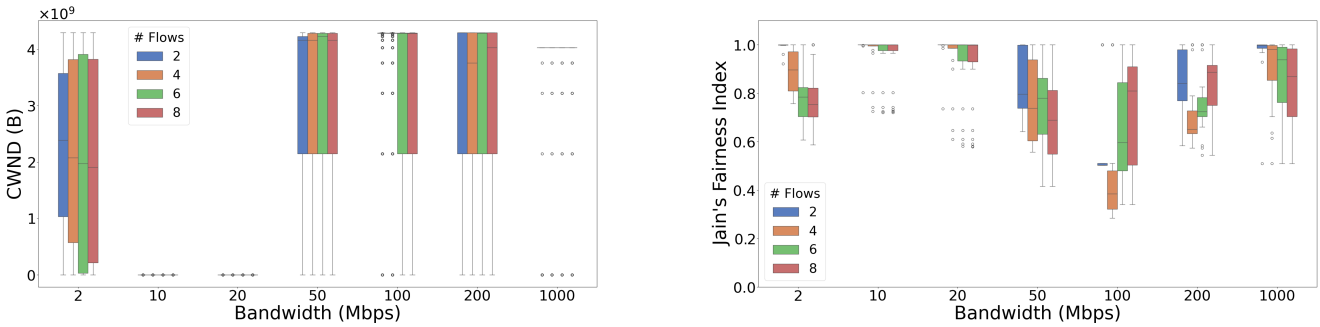
The results of the first round are consistent with the ones presented in *'Experiment 1'*, in the sense that the competing reverse flows back off regularly due to the imposed slowdowns, thus giving space to forward flows to reach high throughput. Most algorithms remain almost unaffected by the increasing number of competing flows. Conversely, once again, TcpLp is the only candidate that does not overshadow Ledbat++. Vegas is also less aggressive than the rest TCP algorithms, achieving significantly lower throughput than them in its effort to be fair. Nevertheless, it is important to note that in these cases network resources are not utilized efficiently, leaving a considerable deal of available bandwidth unexploited.

Varying forward/reverse flows ratio: The same performance trend is exhibited for different ratios between per-direction flows, when the forward flows are 2, i.e., half the reverse ones, or 4, i.e., equal the reverse flows. The Ledbat++-generated



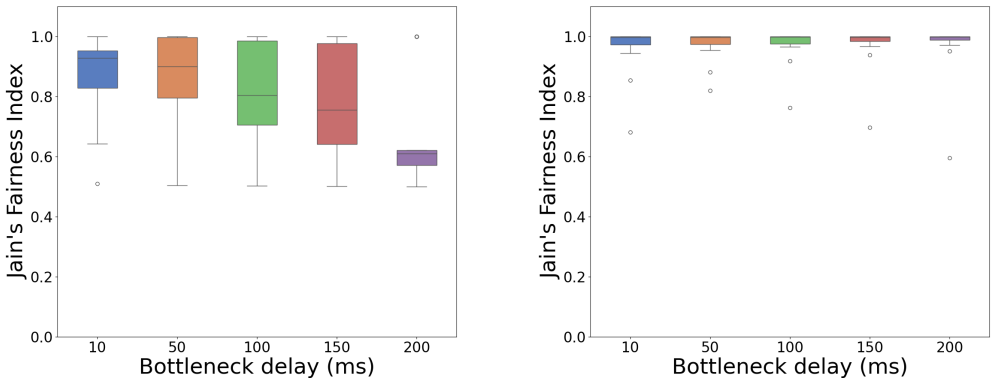
(a) Congestion Window (b) Jain's Fairness Indices

Fig. 6: Experiment 1: Varying Bottleneck Bandwidth: Tcp Ledbat



(a) Congestion Window (b) Jain's Fairness Indices

Fig. 7: Experiment 1: Varying Bottleneck Bandwidth: Tcp Ledbat++



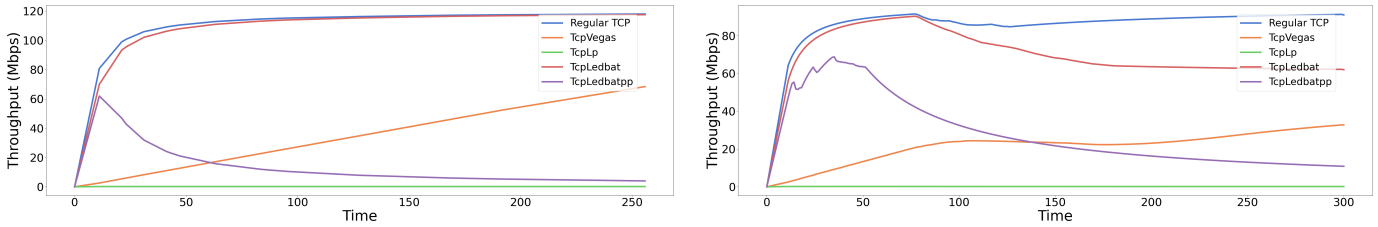
(a) Tcp Ledbat (b) Tcp Ledbat++

Fig. 8: Experiment 1: Varying Bottleneck Delay

reverse traffic is rather limited, hence it does not impact TCP protocols' performance. Once again, we demonstrate that Ledbat++ flows tend to retreat, innately allowing regular TCP traffic to be prioritized.

Scenario 2: We now consider a multi-hop, multi-bottleneck scenario, as illustrated in Figure 1c, to test how Ledbat++ flows affect regular TCP traffic in more realistic settings. This topology is indicative of realistic scenarios and can be easily extended to support experiments of even larger scales. In this scenario, the cross-traffic is generated by Ledbat++ flows, while the straightforward flow is controlled by one of the considered TCP variants.

Overall, the measured throughput for all candidates is lower than in previous scenarios, indicating a non-negligible impact of cross-traffic on their performance. As shown in Fig. 12, Ledbat++ demonstrates the highest throughput in this scenario.



(a) BW = $1Gbps$ (b) BW = $100Mbps$

Fig. 9: Experiment 2/Scenario 1: Varying Reverse Flows Activity

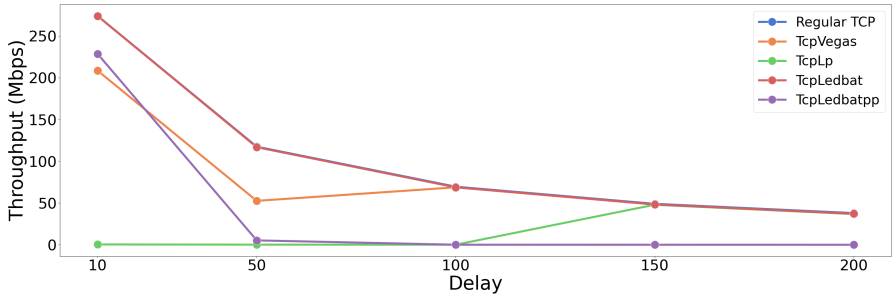
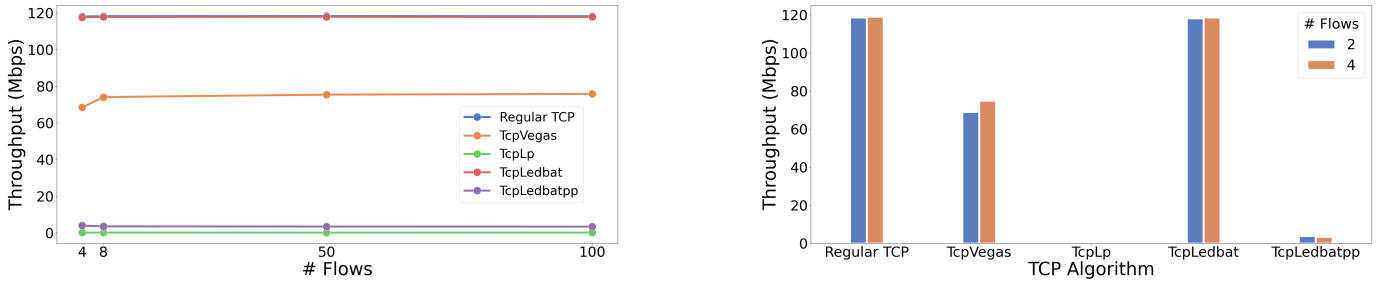


Fig. 10: Experiment 2/Scenario 1: Varying Bottleneck Delay



(a) Varying No. Reverse Flows competing with 1 Forward TCP Flow

(b) Varying No. Forward TCP Flows competing with 4 Reverse Flows

Fig. 11: Experiment 2/Scenario 1: Varying No. Flows

A reason for this is the intra-protocol friendliness, resulting in unconfined traffic flow. In contrast, competing algorithms often detect congestion and resort to occasional fallbacks. TcpLp and Vegas exhibit behaviors consistent with previous scenarios, prioritizing Ledbat++ flows. Ledbat behaves less aggressively in this scenario, conforming to its expected behavior as an LBE protocol. The measured throughput of some algorithms, such as TcpNewReno, is notably reduced, while WestwoodPlus is the least influenced algorithm. The performance of other algorithms falls between these extremes.

C. Key Takeaways

We hereafter lay out the key takeaways of this study:

- There is a trade-off between Ledbat++'s potential to solve the latecomer advantage issue and efficiency, requiring fine-tuning its parameters based on the network specifications to achieve the best possible performance.
- For bandwidths larger than $50Mbps$, and especially for a small number of flows, fairness can be harmed.
- In shared bottleneck scenarios, Ledbat++ works as anticipated, yielding bandwidth to regular TCP flows even when Ledbat++ flows outnumber significantly the latter. TcpLp is the only algorithm that allows the competing reverse Ledbat++ flow to grab a larger share of resources.
- In more realistic settings including cross-flows, Ledbat++'s impact on standard TCP performance is greater.

V. Conclusions

In this paper, we provided an extensive performance evaluation of Ledbat++ over different experimental settings, a modern congestion control algorithm for background traffic. Building upon our newly introduced implementation in ns-3, we conducted the first Ledbat++ performance evaluation to flow scalability and various regular TCP algorithms. Our results validated, on the one hand, that Ledbat++ overcomes the inherent Ledbat weaknesses. On the other hand, the performance measurements demonstrated that Ledbat++ handles background traffic fairly while allowing aggressive TCP

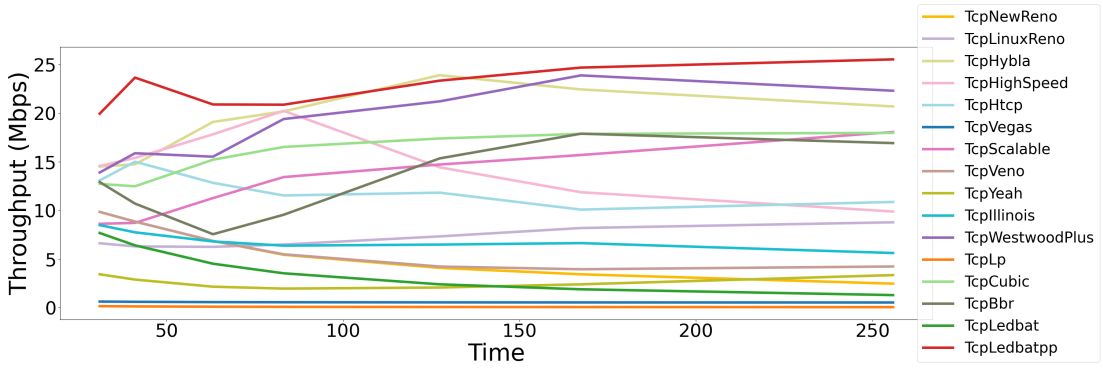


Fig. 12: Experiment 2/Scenario 2: Throughput

flows to dominate, although, in more realistic scenarios, its effect on TCP traffic becomes perceptible. Motivated by this observation, we intend to expand the evaluation to consider lossy scenarios as well (e.g. wireless networks) [5], [13].

We also showed that Ledbat++ competes head-to-head only when confronting other LBE flows. We plan to zoom into the performance comparison of Ledbat++ specifically with LBE models further in the future.

References

[1] P. Balasubramanian, O. Ertugay, and D. Havey, "LEDBAT++: Congestion Control for Background Traffic," Internet Engineering Task Force, Internet-Draft draft-irtf-iccrg-ledbat-plus-plus-01, Aug. 2020, work in Progress. [Online]. Available: https://datatracker.ietf.org/doc/draft-irtf-iccrg-ledbat-plus-plus/01/

[2] M. Bagnulo and A. García-Martínez, "An experimental evaluation of ledbat++," *Computer Networks*, vol. 212, p. 109036, 2022.

[3] M. Bagnulo and A. García-Martinez, "When less is more: Bbr versus ledbat++," *Computer Networks*, vol. 219, p. 109460, 2022.

[4] S. Tang, X. Jiang, M. Zhang, G. Jin, and H. Chen, "Combining reinforcement learning method to enhance ledbat++ over diversified network environments," *Journal of King Saud University-Computer and Information Sciences*, vol. 35, no. 9, p. 101730, 2023.

[5] L. A. Grieco and S. Mascolo, "Performance evaluation and comparison of westwood+, new reno, and vegas tcp congestion control," *ACM SIGCOMM Computer Communication Review*, vol. 34, no. 2, pp. 25–38, 2004.

[6] D. Rossi, C. Testa, and S. Valenti, "Yes, we ledbat: Playing with the new bittorrent congestion control algorithm," in *Passive and Active Measurement: 11th International Conference, PAM 2010, Zurich, Switzerland, April 7-9, 2010. Proceedings 11*. Springer, 2010, pp. 31–40.

[7] I. Komnios, A. Sathiaseelan, and J. Crowcroft, "Ledbat performance in sub-packet regimes," in *2014 11th Annual Conference on Wireless On-demand Network Systems and Services (WONS)*. IEEE, 2014, pp. 154–161.

[8] D. Rossi, C. Testa, S. Valenti, and L. Muscariello, "Ledbat: the new bittorrent congestion control protocol," in *2010 Proceedings of 19th International Conference on Computer Communications and Networks*. IEEE, 2010, pp. 1–6.

[9] G. Carofiglio, L. Muscariello, D. Rossi, C. Testa, and S. Valenti, "Rethinking the low extra delay background transport (ledbat) protocol," *Computer Networks*, vol. 57, no. 8, pp. 1838–1852, 2013.

[10] G. Carofiglio, L. Muscariello, D. Rossi, and S. Valenti, "The quest for ledbat fairness," in *2010 IEEE Global Telecommunications Conference GLOBECOM 2010*, 2010, pp. 1–6.

[11] M. Arumaithurai, "Network friendly congestion control: Framework, protocol design and evaluation," Ph.D. dissertation, Niedersächsische Staats-und Universitätsbibliothek Göttingen, 2010.

[12] K. Ong, "Evaluation and optimisation of less-than-best-effort tcp congestion control mechanisms," Ph.D. dissertation, Murdoch University, 2020.

[13] V. Tsaoussidis and I. Matta, "Open issues on tcp for mobile computing," *Wireless Communications and Mobile Computing*, vol. 2, no. 1, pp. 3–20, 2002.