Low Latency Friendliness for Multipath TCP

Yannis Thomas, Merkourios Karaliopoulos, George Xylomenos and George C. Polyzos
Mobile Multimedia Laboratory & Department of Informatics
School of Information Sciences and Technology
Athens University of Economics and Business
Patision 76, Athens 10434, Greece
e-mail: {thomasi, mkaralio, xgeorge, polyzos}@aueb.gr

***Abstract*—Efficient congestion control is critical to the operation of MPTCP, the Multipath extension of TCP. Congestion control in such an environment primarily aims at enhancing the cumulative TCP throughput over the available paths, while preserving TCP-friendliness by fairly sharing the available bandwidth with single-path TCP flows in each path. While most existing multipath congestion control algorithms fulfill the TCP-friendliness objective in their steady state, their throughput convergence latency is high, rendering them ineffective for short-lived flows. We have proposed *Normalized Multipath Congestion Control* (NMCC), an MPTCP congestion control algorithm that achieves TCP-friendliness faster, by normalizing the growth of individual sub-flow throughput rather than the throughput itself. As NMCC can become unfriendly when it experiences sparse congestion events, in this paper we introduce the *extended NMCC* (e-NMCC) protocol that caters for TCP-friendliness upon both throughput growth and throughput reduction epochs. We analytically characterize e-NMCC in terms of TCP-friendliness and responsiveness and compare it with alternative algorithms. Finally, we assess the performance of e-NMCC through experimentation with the *htsim* simulator and a real Linux implementation. Our results confirm that e-NMCC accelerates throughput convergence, thus ensuring TCP-friendliness regardless of connection duration and underlying network conditions.**

***Index Terms*—MPTCP, congestion control, TCP-friendliness**

I. Introduction

The Internet depends on transport layer protocols such as the *Transmission Control Protocol* (TCP) [1]–[3] to provide reliable end-to-end connectivity while efficiently utilizing network resources and preventing congestion collapse. Among many transport protocols proposed since the inception of the Internet, TCP has prevailed due to its simplicity, its low overhead, and its ability to adapt to diverse network conditions. Recently, the widespread availability of path diversity across the Internet, along with the proliferation of multihomed mobile devices and datacenter servers, have led to the incorporation of multipath features to TCP. Such features are expected to improve TCP's throughput, enable resource pooling and load balancing, and enhance its resilience to network failures [4].

Multipath TCP (MPTCP) [5] is an extended version of TCP that pools multiple paths into a single transport connection, transparently to applications, aiming to enhance connection resilience and resource efficiency. One of the most crucial elements of MPTCP is its congestion control mechanism, which regulates the amount of data entering the network via each path. Implementing congestion control in MPTCP is challenging, as it must address multipath-specific issues, such as TCP-friendliness, bandwidth aggregation, stability and responsiveness, in a complicated and dynamic environment [6].

Most congestion control algorithms proposed for MPTCP, such as the *Linked Increase Algorithm* (LIA) [7], the *Opportunistic Linked Increase Algorithm* (OLIA) [8] and the *Balanced Linked Adaptation* (BALIA) algorithm [8], were inspired by the fluid model of TCP by Kelly and Voice [9]. They draw upon this model to derive the appropriate amount of TCP congestion window increase upon receipt of an ACK and window decrease upon anticipation of a congestion event, in order to achieve high resource utilization, stability, responsiveness and TCP-friendliness. Whereas these algorithms are effective in the long run, they may take a long time to impose the intended bandwidth shares across the connection paths.

In practice, this convergence delay may subvert TCP-friendliness. Recent studies on Internet MPTCP traffic demonstrate that up to 95% of it is due to short-lived connections (transporting less that 1 MB) [10]. Web traffic exhibits a similar trend, where large video files that previously led to long-lived flows, are currently split into multiple smaller files for cacheability and performance [11]. With relatively short-lived flows, LIA, OLIA and BALIA are unlikely to have enough time to enter steady state and achieve TCP-friendliness. On the other hand, even when flows are long-lived, as with large file transfers, our evaluation results suggest that Linux MPTCP implementations with LIA, OLIA and BALIA converge rather slowly. Consequently, regardless of the transfer duration, we need algorithms that achieve TCP-friendliness quickly.

The *Normalized Multiflow Congestion Control* (NMCC) algorithm [12] is a multipath congestion control algorithm that primarily aims to satisfy this requirement. The distinguishing feature of NMCC is that it controls the throughput *growth* rate of the sub-flows rather than the throughput itself. NMCC pursues TCP-friendliness right from the onset of the connection, by being TCP-friendly during both the Slow Start and Congestion Avoidance phases. Unlike the other MPTCP congestion control algorithms, NMCC also simplifies the window management process. Overall, NMCC offers significant advantages, such as fast convergence to TCP-friendliness, enhanced responsiveness to network conditions and high resource utilization. However, it can become unfriendly when it experiences sparse congestion events.

In this paper, we make the following contributions:

- Using the Linux implementation of MPTCP, we provide experimental evidence that MPTCP with LIA, OLIA and

BALIA may demand hundreds of seconds until they achieve TCP-friendliness, thus limiting their friendliness to long-lived flows.
- We introduce a new algorithm, hereafter called *extended NMCC* (e-NMCC), that meets the TCP-friendliness goal throughout a connection's lifetime, *i.e.*, at window increase *and* decrease epochs.
- We characterize the e-NMCC algorithm mathematically, according to the analysis in [13], and comparatively explore its TCP-friendliness and responsiveness.
- We assess the performance of the e-NMCC algorithm, first via a real Linux implementation, and then in various benchmark topologies through the *htsim* simulator. We explore its efficiency in addressing TCP-friendliness, resource utilization and load balancing, in comparison with LIA, OLIA and BALIA. The simulation results validate our mathematical analysis in a wider range of conditions.

The remainder of this paper is organized as follows. In Section II we summarize existing work on MPTCP congestion control, while in Section III we briefly describe the base implementation of NMCC. In Section IV we introduce the e-NMCC algorithm that also handles throughput reduction epochs. In Section V we provide an analytical characterization of e-NMCC and its properties. In Section VI we experimentally evaluate the TCP-friendliness convergence of e-NMCC against LIA, OLIA and BALIA, using a real MPTCP Linux implementation. In Section VII we compare e-NMCC against LIA in benchmark scenarios using the *htsim* simulator. In Section VIII we discuss the integration of e-NMCC with MPTCP and, finally, provide our conclusions in Section IX.

II. Background and Related Work

A. MPTCP Congestion Control Goals

The revived interest in multipath transport can be traced to the widespread availability of multihomed devices; MPTCP is currently available in iOS and Linux, covering both mobile devices with multiple wireless interfaces and servers with multiple wired interfaces. On the surface, MPTCP offers the exact same application-level service as plain (single-path) TCP. Under the hood, however, it provides the necessary components to establish and manage multiple *sub-flows* of data sent over potentially disjoint paths, also known as *routes*.

In general, MPTCP establishes multiple sub-flows by introducing multiple TCP handshakes. In the initial handshake, the users exchange the necessary information for establishing the first sub-flow, but also advertise their capability to deploy additional sub-flows via additional IP addresses, that "encode" additional routes. If no additional routes are available, then MPTCP falls back to TCP. Otherwise, a TCP handshake takes place for each extra IP address, thus allowing the establishment of more sub-flows; sub-flows are individually controlled through private congestion and flow control primitives, such as congestion timer and congestion window.

One of the most important components of MPTCP is the congestion control module that regulates the cumulative and per sub-flow transfer rates, aiming to maximize throughput by utilizing the available routes, while avoiding congestion collapse. Most of the algorithms in the literature, *e.g.*, , [7], [8], [13]–[16], draw upon the building blocks of TCP, such as Slow Start, Congestion Avoidance and loss-based congestion detection [17], modifying them to spread the traffic over multiple paths. We summarize below the MPTCP congestion control objectives that are most frequently considered in the literature and are most relevant to our work.

- *Throughput aggregation:* The ability to aggregate the transfer rates of the sub-flows to achieve increased *cumulative* throughput.
- *TCP-friendliness:* The property of sharing the network resources fairly with single-path flows [3], expressed as: "When a multiflow connection competes with a single-flow connection for the same network resource, the former must not acquire a larger share of that resource than the latter."
- *Load balancing:* The ability to distribute the traffic load among multiple paths in order to balance the congestion level across the network [7].
- *Responsiveness:* The ability to respond quickly to dynamic changes in the network, but not so quickly as to endanger the stability of the sub-flows [9].
- *TCP-friendliness convergence latency:* The time needed to reach TCP-friendliness (while being TCP unfriendly). Minimizing this latency is critical for the effectiveness of multipath congestion control.

B. MPTCP Congestion Control Algorithms

The simplest multipath congestion control algorithm for MPTCP, known as *UNCOUPLED*, employs sub-flows with individual congestion windows and independent window management. This design offers enhanced throughput and fast adaptation to network conditions, but tends to be overly aggressive towards unicast connections, thus failing the TCP-friendliness constraint. To achieve TCP-friendliness, *Equally-Weighted TCP* (EWTCP) [14] splits traffic "evenly" among sub-flows so that it cumulatively grasps the same share of resources as a TCP connection. However, EWTCP does not satisfy the load balancing requirement, as the proportional management of the sub-flows disregards the individual properties of the dissemination paths.

The *COUPLED* algorithm [15] was the first to emphasize the importance of being TCP-friendly, while shifting traffic to the least congested path so as to enhance load balancing. It handles the available paths as a pool of resources, aiming to balance their congestion level and increase utilization. However, pushing all traffic to the least congested path has some drawbacks, such as performance degradation with heterogeneous paths and poor responsiveness to network changes.

The *Linked Increase Algorithm* (LIA) [7] was developed to tackle both TCP-friendliness and responsiveness. LIA pushes traffic to the least congested path so as to enhance load balancing, similarly to *COUPLED*, but also introduces an aggressiveness parameter that attempts to keep some traffic in the more congested paths in order to remain responsive. This parameter is based on two equilibrium conditions: first, LIA balances the congestion window increases and decreases

in steady state in order to be stable and, second, it equalizes the resource shares of MPTCP and TCP at the bottleneck in order to be TCP-friendly. Demonstrating sufficient friendliness and resource utilization, LIA soon became the established congestion control algorithm for MPTCP.

While favoring the least congested path, LIA does not push traffic exclusively there, thus penalizing overall network resource utilization under certain conditions. The *Opportunistic Linked Increase Algorithm* (OLIA) [8] extended LIA in order to enhance resource pooling, while maintaining high responsiveness. Specifically, OLIA increases faster the congestion window of sub-flows with a high transfer rate but relatively small windows. Moreover, OLIA introduces probing traffic over the worse paths to achieve sufficient responsiveness. Nevertheless, recent studies [13] show that OLIA does not respond well to abrupt load changes. In the same paper, the *Balanced Link Adaptation* (BALIA) algorithm is introduced, a generalization of existing algorithms that strikes a good balance between friendliness and responsiveness.

Finally, *weighted Vegas* (wVegas) [16], a delay-based congestion control algorithm inspired by TCP Vegas, uses queuing packet delay to detect congestion, unlike other proposals which exploit time-out timers. The main advantage of wVegas is quick traffic shifting, as it can be more sensitive to network load changes. However, tuning the algorithm's sensitivity is not trivial and the investigation of its behavior is not mature. For example, the handling of RTT variation in case of rerouting remains unclear. A survey of congestion control algorithms for multipath transport is presented in [6].

C. Modeling Multipath Rate Control

One of the key pillars in modeling MPTCP is the exploitation of the fluid model analysis by Kelly and Voice [9]. The fluid model tracks the transfer rate adaptation of a TCP-sender during the Congestion Avoidance phase, as the sender attempts to ensure system stability, protocol responsiveness and TCP-friendliness. It can be related to the window-based congestion control of TCP so as to deduce a sufficient condition for the desired amount of window increase or decrease upon receipt of an ACK or a congestion event, respectively.

For example, following the analysis about connection stability [9, Eq. 22], LIA tries to balance the window increase and decrease in steady state, by modeling them as the product of *event probability* times the *window modification*. For instance, the window decrease in steady state equals the product of steady state throughput, segment loss probability and the amount of the window reduction. To solve the equation, the authors assume that the segment loss probability is statistically negligible, a simplification that can lead to poor performance in error-prone and heterogeneous paths. Indeed, in Section VI we demonstrate that LIA achieves TCP-friendliness only after the steady state is reached *and* under low error-rate conditions.

A second common feature of the fluid model is the omission of the Slow Start phase, as it is considered a transient state with no measurable effect on long-term performance. Consequently, multipath connections are not TCP-friendly for a "brief" period after they are launched, gradually converging to the desired fair equilibrium. To the best of our knowledge, only the long-term TCP-friendliness of MPTCP congestion control has been evaluated (assuming so-called "long-lived flows"); the rate of convergence to TCP-friendliness and its correlation with network conditions has not been explored yet.

We note that protocol responsiveness and protocol convergence to TCP-friendliness are different concepts. The former refers to the efficiency of shifting traffic among sub-flows while being TCP-friendly; the latter refers to the time needed to achieve TCP-friendliness while being TCP-unfriendly. That "brief" period of imbalance, during which MPTCP is TCP-unfriendly, can be critical for network stability, affecting how well MPTCP fares with respect to TCP-friendliness.

D. Long-Term vs. Short-Term TCP-friendliness

The vast majority of previous works on TCP and MPTCP focus on long-term protocol performance [3], [13], that is, steady state performance when the system has stabilized.

Studying the TCP-friendliness of MPTCP only in the long-run follows the legacy analyses of TCP. During the first studies of TCP's impact on complex and diverse computer networks, modeling the properties of such volatile systems was much harder than modeling a stable system. In order to mitigate complexity then, significant simplifying assumptions were made, at the expense of challenging the realism of the resulting models under some conditions. For instance, the fluid model in [9] assumes that the RTTs of the network remain fixed in time, because the network has sufficient bandwidth and low enough total load that it never sustains any queues. However, in [18] the authors explore real traces of TCP connections that show measurable variability of RTT at the granularity of seconds, and significant variability across hours. We acknowledge that the long-term approach, where network performance is stabilized, constitutes a powerful tool that can provide essential information about the behavior of the network, but it can also become unrealistic when its assumptions are violated, *e.g.*, , when network congestion and RTT are volatile or when protocols do not operate long enough to enter steady state. For this reason, we consider essential the investigation of MPTCP before flows are stabilized.

A second common assumption is that the impact of a flow in the network is proportional to its duration. Short-term flows are thus considered too brief to affect the network, similarly to the Slow Start phase of a TCP connection [19], hence fair sharing cannot be affected by TCP-unfriendly short-term connections. We argue that the traffic volume of unfriendly flows should be taken into account when analyzing TCP-friendliness: network fairness can also be affected by a large number of unfriendly (even short-lived) flows operating in parallel. The observations that "mice" flows severely outnumber "elephant" flows in the current Internet and that users deploy multiple concurrent connections (Web traffic traces indicate that 50% of users deploy at least 5 connections in parallel) [11], emphasize the impact of small, yet numerous, flows, and indicate the need for pursuing short-term TCP-friendliness.

III. Normalized Multiflow Congestion Control (NMCC)

Normalized Multiflow Congestion Control (NMCC) [12] is a novel congestion control algorithm for multipath connections. It offers TCP-friendliness throughout the lifetime of a connection and high resource utilization under various path setups, including disjoint, overlapping and heterogeneous paths. It differs from alternative designs such as LIA, OLIA and BALIA, in two main ways: first, it introduces a new approach towards *pursuing* TCP-friendliness and, second, it embodies a new scheme for *implementing* TCP-friendliness.

a) Pursuing TCP-friendliness: NMCC achieves TCP-friendliness by normalizing the *growth* of the transfer rate rather than the transfer rate of each sub-flow. It exploits the fact that all connections start at the same state, that is, with the minimum congestion window, and remain TCP-friendly as long as their throughput increase rates are equal. NMCC distributes the throughput increase rate of its fastest sub-flow across its pool of sub-flows. The TCP-friendliness requirement is deterministically met upon each window increase epoch, making NMCC continuously TCP-friendly.

Assume that Ω_r and Ω'_r are the standard (or unfriendly) and TCP-friendly throughput growth rates of sub-flow r, respectively. If Ω_{max} is the growth rate in the most aggressive single path connection, the TCP-friendliness constraint demands that

$$\sum_{r \in S} \Omega'_r = \Omega_{max} \tag{1}$$

NMCC expresses the growth rate of sub-flow r as a function of its RTT, r_r, and its congestion window, w_r. Every r_r, the throughput of r increases by 1 MSS/r_r in Congestion Avoidance and by w_r/r_r in Slow Start, where MSS is the network's *Maximum Segment Size*. Therefore:

$$\Omega_r = \begin{cases} 1/r_r^2, & \text{in congestion avoidance} \\ \\ w_r/r_r^2, & \text{in slow start} \end{cases} \tag{2}$$

b) Implementing TCP-friendliness: NMCC exploits the well-known bias of TCP against large-RTT connections to control over-aggressiveness (*e.g.*, see [2]). Instead of adjusting the congestion window value as in other MPTCP solutions, NMCC inflates the RTT values used in the calculations. This greatly simplifies TCP-friendliness in the Slow Start phase.

Specifically, let $r'_r \geq r_r$ denote the *inflated* RTT for sub-flow r, which results in a slower growth of the congestion window compared to regular TCP. The actual reduction is controlled by the *friendliness factor* $m \geq 1$, defined as

$$r'_r = m r_r \tag{3}$$

Therefore, the friendly throughput growth rate, Ω'_r, is

$$\Omega'_r = \begin{cases} \frac{1}{r'^2_r} = \frac{1}{m^2 r_r^2}, & \text{in congestion avoidance} \\ \\ \frac{w_r}{r'^2_r} = \frac{w_r}{m^2 r_r^2}, & \text{in slow start} \end{cases} \tag{4}$$

The parameter m is set so that the throughput growth rate across all sub-flows matches the respective rate in the most aggressive single-path, Ω_{max}. As m is the same for all sub-flows, regardless of their state, it is easy to prove that $\Omega_r = m^2 \Omega'_r$. By substituting Ω'_r with Ω_r/m^2 in (1), we generate a unified formula for estimating m, incorporating sub-flows in both the Congestion Avoidance and Slow Start phases:

$$m^2 = \frac{\sum_{r \in S} \Omega_r}{\Omega_{max}} \tag{5}$$

By applying m to the RTTs of all sub-flows, NMCC limits their throughput growth rate. Therefore, although NMCC mostly utilizes the sub-flow in the "best" path, it takes into account all the slower paths, exhibiting good responsiveness. As a result, NMCC does not require probing to detect load changes on unused paths unlike LIA, which introduces a special parameter to keep a moderate amount of traffic over slow paths, or OLIA, which requires probing.

In addition, NMCC can perform efficiently in heterogeneous environments, adapting fast to path failures and congestion bursts. For instance, consider an integrated terrestrial-satellite network where the propagation delay is 10 ms in the terrestrial link and 250 ms over the satellite link. When both NMCC sub-flows are in Congestion Avoidance, then, according to (2), the throughput growth rates are $1/10^2$ and $1/250^2$. According to (5), $m = 10079$, which results in tiny adjustments to the throughput growth of the sub-flows, thus allowing NMCC to effectively grasp the available resources.

IV. Extended NMCC (*e-NMCC*)

NMCC eliminates TCP-friendliness convergence latency by fairly distributing the throughput increase among the sub-flows, thus achieving cumulative throughput growth that is equal to what would be achievable by regular TCP in the best path. However, as presented in [12], NMCC considers TCP-friendliness only when all sub-flows share the same bottleneck[1], meaning that all sub-flows will receive a congestion event (*i.e.*, a retransmission timer timeout or triple duplicate ACK) when congestion appears. In practice, however, congestion events can be triggered only for a *subset* of the sub-flows.

To understand the impact of this on TCP-friendliness, consider two connections competing at a bottleneck link, a single-path connection and a TCP-friendly multipath connection with two sub-flows on routes with equal RTTs. During a congestion escalation period, the single-path flow and *one* of the two sub-flows may experience a congestion event, thus reducing their transfer rate to roughly one-half of their current rate. But since only one sub-flow has noticed the event, the single-path and the multipath connections will suffer a reduction of 50% and 25% in their (cumulative) throughput, respectively, hence diverging from TCP-friendliness. This asymmetry intensifies with the frequency of congestion events that affect a subset of sub-flows (a *partial congestion event*). In the worst case, only a single sub-flow will experience a congestion event; in the best case, all sub-flows will do so (*global congestion event*).

[1] In that paper, NMCC exploits an in-network assistance module that reports shared bottlenecks, hence the TCP-friendliness constraint is applied only when sub-flows share a bottleneck.

To gain insight into the resulting performance bounds, we simulate this toy-example (a multipath (MP) connection with two sub-flows over equal RTT paths and a single-path (SP) connection competing over one of these paths). Both connections are left to increase their transfer rate in a friendly manner and, when the link is full, the flows receive congestion events that reduce their throughput. We simulate three types of congestion events: the worst (only partial events), the best (only global events) and a realistic case, where global and partial congestion events co-exist and the faster connection get more congestion events.[2] Figure 1(1.a) plots the results of the worst-case scenario, where only the sub-flow with the largest congestion window experiences a packet loss and the MP connection ends up grasping 67% of the resources. Figure 1(1.b) illustrates the results of the realistic scenario with partial and global events, where the MP connection is still too aggressive, grasping 58% of the resources. With global congestion events only (not shown), the MP and SP connections equally share the medium, as shown in [12].

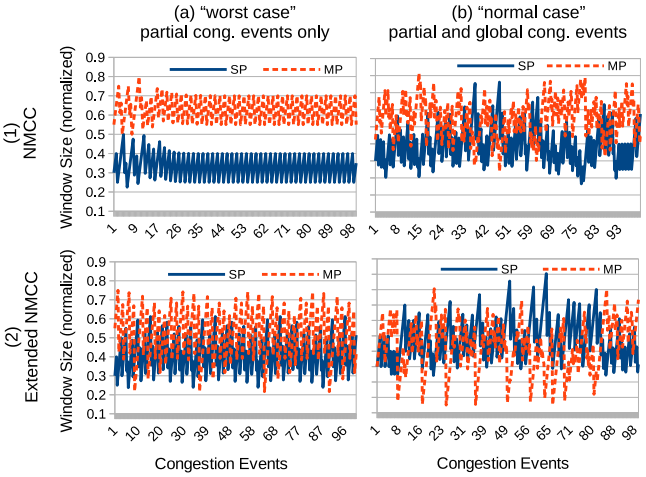

Fig. 1. NMCC (MP) and single-path (SP) window sizes with either "partial" (column a) or "partial and global" congestion events (column b); top row depicts NMCC performance, bottom row depicts e-NMCC performance.

A. Modeling throughput reduction

The over-aggressiveness of NMCC during partial congestion events motivates an extension for regulating throughput decrease under all types of congestion events (partial and global). Our approach consists of (a) keeping intact the NMCC part that controls throughput growth, as it delivers fast TCP-friendliness convergence, but (b) adding the following TCP-friendliness rule for regulating throughput reduction:

> *The cumulative throughput reduction of all multipath sub-flows after any number of contemporaneous window reductions due to congestion, should be equal to the corresponding reduction of a single-path flow over the "best" path.*

The path with the highest throughput increase rate, Ω_{max}, is marked as the "best" path, serving as the benchmark when the congestion windows either increase or decrease.

Unlike the deterministic throughput increase per ACK, a deterministic throughput reduction per congestion event is not straightforward. We do not know *a priori* the number of sub-flows that will be affected by a single congestion event, hence we cannot estimate the respective cumulative throughput decrease so as to instantly equalize the performance of multipath and single-path connections. Two different approaches can be used to tackle this problem; a *stateful* and *stateless* one.

a) Stateful approach: We estimate the total throughput reduction of sub-flows over a period of time, by keeping track of the reductions that took place over an interval t_w. Exploiting the temporal correlation of congestion events from a shared bottleneck, we could distinguish global from partial congestion events. More specifically, when a sub-flow receives a congestion event, it shrinks its window by half, as if this was a global event, it waits for time t_w and, then, checks whether other sub-flows are also affected by the same congestion event. If all sub-flows have received a congestion signal (indicating global congestion), then no additional window reduction takes place. If not (indicating partial congestion), the sub-flow applies an additional reduction to ensure TCP-friendliness.

The main weakness of this approach is that the protocol responds "accurately" to congestion with latency t_w, thus potentially affecting its friendliness. In order to accurately detect global events even in RTT-mismatched paths, the delay t_w should be equal to the maximum current RTT among all sub-flows, r_{max}, which also determines the maximum loss timer among the sub-flows.[3] Thereupon, if sub-flow r experiences a congestion event and it has the highest RTT, $r_r = r_{max}$, the impact of t_w is expected to be unnoticeable, since the unfriendly reduction is instant and the friendly (second) reduction is just one RTT late. If, however, sub-flow r has an RTT n times smaller than the maximum, $r_r = r_{max}/n$, then the impact of t_w can be measurable since the unfriendly reduction is instant but the friendly reduction is delayed by $n \cdot r_r$; during this period the sub-flow will be unfriendly. In [10] authors indicate that roughly 86% of MPTCP connections exhibit less than 50 ms of RTT-mismatch during connection establishment,[4] while approximately 47% of these RTTs are less than 100 ms. Hence, at least for 40% of MPTCP connections n would be less than 2, thus mitigating the impact of n. Nevertheless, further analysis and experimentation is needed to ensure that the stateful approach takes into account all possible congestion event scenarios. The e-NMCC protocol therefore adopts the stateless approach presented below, which is not affected by RTT-mismatch.

b) Stateless approach: We estimate the total throughput reduction of sub-flows over a period of time, by assuming that the throughput and the (packet) loss rates of the flows are relatively stable when the flows are stabilized. We acknowledge that a (TCP) flow converges to its fair share of

[2]When a link is full, a random sample is drawn from a uniform distribution [0,100) for each sub-flow, and a congestion event is triggered if the sample is smaller than the bandwidth share of the sub-flow.

[3]Although the *Smoothed RTT* is a more accurate representation of the loss timer, we use the actual RTT in the text for simplicity.

[4]Roughly 10%, 54.3% and 13.7% of MPTCP connections exhibit 0, less than 10 and more than 50 ms of RTT-mismatch.

resources after a few congestion rounds [20]. Thereupon, we denote the frequency of congestion events for the fastest single-path flow and for multipath sub-flow r as f_{sp} and f_r, respectively. Likewise, let d_{sp} and d_r be the throughput reduction after a congestion event for the fastest single-path flow and multipath sub-flow r, respectively. Demanding that the throughput reduction rates are equalized reduces to:

$$f_{sp}d_{sp} = \sum_{r\in S} f_r d_r \tag{6}$$

where S is the set of sub-flows.

A congestion event moves TCP into Slow Start or Fast Recovery. In the first case, the congestion window grows exponentially, reaching the Slow Start threshold very quickly, whereas, in the second case, the window is directly set to the Slow Start threshold plus 3 MSS, where MSS is the *Maximum Segment Size*.[5] We assume that throughput is reduced by approximately 50% in both cases, hence the throughput reduction of the single path connection can be written as:

$$d_{sp} = \frac{w_{sp} - w_{sp}/2}{r_{sp}} = \frac{1}{2}\frac{w_{sp}}{r_{sp}} \tag{7}$$

and the loss frequency f as the product of the throughput multiplied by the packet loss rate, p:

$$f = p\frac{w}{r} \tag{8}$$

We can formally demonstrate the friendliness issue by assuming that d_r, the throughput decrease of sub-flow r, is unregulated, thus following (7). Then, we can rewrite (6) as:

$$\begin{aligned} p_{sp}\frac{w_{sp}}{r_{sp}}\frac{w_{sp}}{2r_{sp}} &= \sum_{r\in S} p_r \frac{w_r}{r_r}\frac{w_r}{2r_r} \Rightarrow \\ \frac{p_{sp}w_{sp}^2}{r_{sp}^2} &= \sum_{r\in S}\frac{p_r w_r^2}{r_r^2} \end{aligned} \tag{9}$$

Now, consider the case where two sub-flows in the congestion avoidance phase share routes of equal bandwidth, latency and error rates. Then, according to (5), $m^2 = 2$, hence the sub-flow window growth rate is halved, as presented in the following:

$$r_r = r_{sp},\ p_r = p_{sp},\ w_r = \frac{w_{sp}}{2} \tag{10}$$

It is trivial to check that, given (10), condition (9) is not satisfied, verifying the need to address the unfriendliness issue.

B. TCP-Friendly Throughput reduction

To maintain TCP-friendliness under various types of congestion events, we introduce the *threshold factor*, m_{th}, a positive real number ($|S| \geq m_{th} \geq 1$), that regulates the throughput reduction of a sub-flow during a congestion event, as follows:

$$d_r = d_{sp}m_{th} = \frac{w_r - w_r/2}{r_r}m_{th} = \frac{w_r}{2r_r}m_{th} \tag{11}$$

[5]In both cases we assume that the RTT remains relatively static, since the bottleneck is shared by numerous flows.

We can now rewrite (6) , using (11) to express d_r, and estimate the threshold factor, m_{th}, as follows:

$$\begin{aligned} \frac{p_{sp}w_{sp}^2}{r_{sp}^2} &= \sum_{r\in S}\frac{p_r w_r^2}{r_r^2}m_{th} \Rightarrow \\ m_{th} &= \frac{p_{sp}w_{sp}^2/r_{sp}^2}{\sum_{r\in S} p_r w_r^2/r_r^2} \end{aligned} \tag{12}$$

The estimation of m_{th} by a multipath sub-flow, requires knowing w_{sp}, r_{sp} and p_{sp}, which express the performance of a TCP-like flow on the best available path (the one with the highest throughput increase rate, Ω_{max}). By definition, the packet drop rate of the path, being a feature of the medium, is not affected by NMCC, hence $p_{sp} = p_{max}$. The throughput of the single path connection is equal to the cumulative throughput of the multi-path connection, as a result of meeting the friendliness constraint during window increase, hence $w_{sp}/r_{sp} = \sum_{r\in S} w_r/r_r$. Consequently, any sub-flow can estimate m_{th} via the following formula:

$$m_{th} = \frac{p_{max}(\sum_{r\in S} w_r/r_r)^2}{\sum_{r\in S} p_r w_r^2/r_r^2} \tag{13}$$

This extension mirrors the proportional throughput growth scheme of NMCC, in that the throughput is most aggressively reduced when the sub-flows are equally fast. Specifically, when the connection deploys one sub-flow, then $m_{th} = 1$, thus falling back to single path behavior. When $|S|$ identical sub-flows (running over identical paths) are deployed, then $m_{th} = |S|$ and the throughput reduction for each sub-flow is maximized. Finally, as paths get more diverse and a subset of the sub-flows grasps most resources, $m_{th} \to 1$, hence throughput reduction is not regulated.

We repeated the simulations of Fig. 1, this time including the threshold modification factor, m_{th}, in the congestion control scheme. In the worst-case scenario, multipath (MP) gains 54% of the resources (see Fig. 1(2.a)), thus improving friendliness by 13% (compared to Fig. 1(1.a)). In the more realistic scenario, MP gets 49% of the resources (see Fig. 1(2.b)), thus achieving friendliness and providing preliminary evidence for the validity of our solution.

Finally, we point out that this extension does not delay the convergence of the protocol towards TCP-friendliness. The NMCC connection is TCP-friendly in Slow Start and our extension is activated only after the first packet loss. At that time, the sub-flows can estimate the packet drop probability of the routes, hence (assuming a credible estimation of drop probability) e-NMCC has converged. We experimentally asses the convergence latency of MPTCP with e-NMCC in Section VI.

The e-NMCC algorithm is presented below:

ALGORITHM: *extended* NMCC (e-NMCC)

- For each ACK on path r,

$$w_r \leftarrow \begin{cases} w_r + \frac{1}{w_r}\frac{\Omega_{max}}{\sum_{r\in S}\Omega_r} & \text{in congestion avoidance} \\ w_r + \frac{\Omega_{max}}{\sum_{r\in S}\Omega_r} & \text{in slow start} \end{cases} \tag{14}$$

TABLE I
(K, Φ) TUPLES FOR DIFFERENT MPTCP MODELS UNDER THE FLUID MODEL (17)

	EWTCP	COUPLED MPTCP	LIA MPTCP
$k_s(\mathbf{x}_s)$	$x_r^2/2$	$x_r^2/2$	$x_r^2/2$
$\phi_s(\mathbf{x}_s)$	$\frac{2\alpha}{x_r^2\tau_r^2}$	$\frac{2}{\tau_r^2(\sum_{k\in S} x_k)^2}$	
Friendliness	COUPLED $\succ_f$ LIA $\succ_f$ EWTCP($\alpha > 1$)		
Responsiveness	COUPLED $\prec_r$ LIA $\prec_r$ EWTCP($\alpha > 1$)		
Legend	$\succ_f$: "friendlier than" $\prec_r$: "less responsive than"		

- For each loss on path r,

$$ss\_thresh \leftarrow w_r - \frac{w_r}{2} \cdot \frac{p_{max}(\sum_{r\in S} w_r/r_r)^2}{\sum_{r\in S} p_r w_r^2/r_r^2}$$

$$w_r \leftarrow 2$$

V. ANALYTICAL CHARACTERIZATION OF E-NMCC

A. Background

The analytic framework in [13] provides a solid reference for comparing a broad class of MPTCP algorithms with respect to their fairness and responsiveness properties. Briefly, the dynamics of window-based MPTCP algorithms that upon an ACK on sub-flow r increase its window w_r to

$$w_r \leftarrow w_r + I_r(\mathbf{w}_s) \tag{15}$$

and upon a loss event reduce it to

$$w_r \leftarrow w_r - D_r(\mathbf{w}_s) \tag{16}$$

can be captured by the system of differential equations

$$\begin{aligned} \dot{x}_r &= k_r(\mathbf{x}_s)(\phi_r(\mathbf{x}_s) - q_r)^+_{x_r} \qquad r \in s,\ s \in S \\ \dot{p}_l &= \gamma_l(y_l - c_l)^+_{p_l} \qquad l \in L \end{aligned} \tag{17}$$

where $\mathbf{x}_s, s \in S$ is the vector of rates in all sub-flows of a connection, p_l and q_r are the probabilities of loss at individual link l and cumulatively in route r, respectively, and y_l is the aggregate traffic rate in link l, as determined by the traffic and routing matrices. This fluid congestion control model jointly determines the send rates of multipath connections and the data loss rates at the network buffers.

The variants of MPTCP that are captured by this modeling framework are mapped to distinct tuples (K_s, Φ_s), where $K_s(\mathbf{x}_s) \cdot (k_r(\mathbf{x}_s, r \in s)$ and $\Phi_s(\mathbf{x}_s) \cdot (\phi_r(\mathbf{x}_s, r \in s)$. The shape of these vectors determine the dynamic properties and the equilibria of the fluid model in (17) and rank the different variants with respect to their friendliness and responsiveness. More specifically, it is shown in [13] that:

- (A.1) EWTCP [14], COUPLED MPTCP [15] and LIA MPTCP [7] can be mapped to (17), whereas OLIA cannot. Table I lists the (K_s, Φ_s) tuples for each protocol.
- (A.2) Under certain assumptions[6] an MPTCP algorithm $(\tilde{K}, \tilde{\Phi})$ is friendlier than another algorithm $(\hat{K}, \hat{\Phi})$ if $\hat{\Phi}_s(\mathbf{x}_s) \leq \tilde{\Phi}_s(\mathbf{x}_s)$
- (A.3) Under certain assumptions (a subset of those required for the friendliness comparison) an MPTCP algorithm $(\tilde{K}, \tilde{\Phi})$ is more responsive than another algorithm $(\hat{K}, \hat{\Phi})$ if $\tilde{K}_s(\mathbf{x}_s) \leq \hat{K}_s(\mathbf{x}_s)$ and $\frac{\partial \tilde{\Phi}_s(\mathbf{x}_s)}{\partial \mathbf{x}_s} \leq \frac{\partial \hat{\Phi}_s(\mathbf{x}_s)}{\partial \mathbf{x}_s}$
- (A.4) There is a fundamental trade-off between the friendliness and responsiveness properties. For two algorithms $(\tilde{K}, \tilde{\Phi})$ and $(\hat{K}, \hat{\Phi})$, with $\tilde{K} = \hat{K}$, if one algorithm is superior in friendliness, it will be inferior in responsiveness. Table I ranks protocols that fit the framework in terms of friendliness and responsiveness.

[6]For the full details of the modeling framework, see [13].

B. NMCC positioning

Unfortunately, the framework in [13] is not of much help in characterizing the e-NMCC algorithm since the response to packet loss is more involved than what (16) prescribes, including the error rate of the path, p, and window management in Slow Start. However, we can integrate the e-NMCC algorithm with the framework by assuming that the error rates of the paths are equal and that all sub-flows are in Congestion Avoidance. We acknowledge that modeling e-NMCC under special conditions mitigates the usefulness of this analysis, but we present it as a part of a cohesive study, which also includes thorough experimental evaluation, in order to shed light on the properties of e-NMCC under certain conditions.

Thereupon, in e-NMCC, the window increment $I_r(\mathbf{w}_s)$ and decrement $D_r(\mathbf{w}_s)$ take the form:

$$\begin{aligned} I_r(\mathbf{w}_s) &= \frac{2\max_{k\in S}(1/\tau_k^2)}{w_r^2 \sum_{k\in S} 1/\tau_k^2} \\ D_r(\mathbf{w}_s) &= \frac{w_r(\sum_{k\in S} x_k)^2}{2\sum_{k\in S} x_k^2} \end{aligned} \tag{18}$$

Hence, the fluid model representation of the e-NMCC algorithm is $(K^{e-nmcc}, \Phi^{e-nmcc})$, with

$$\begin{aligned} k_r^{e-nmcc}(\mathbf{x}_s) &= \frac{1}{2x_r^2} \\ \phi_r^{e-nmcc}(\mathbf{x}_s) &= \frac{2\max_{k\in S}(1/\tau_k^2)\sum_{k\in S} x_k^2}{x_r^2\tau_r^2 \sum_{k\in S} 1/\tau_k^2(\sum_{k\in S} x_k)^2} \end{aligned} \tag{19}$$

In view of the (K, Φ) values for the three protocols in Table I, it is straightforward to show that under equal RTTs for all sub-flows $r \in s$ (the case studied in [13]), e-NMCC is equally friendly with COUPLED, the most TCP-friendly algorithm in the analysis of [13]. It is straightforward to show that e-NMCC is friendlier that the least TCP-friendly algorithm, EWTCP:

$$\phi_r^{e-nmcc}(\mathbf{x}_s) = \frac{2\sum_{k\in S} x_k^2}{x_r^2\tau_r^2|s|(\sum_{k\in S} x_k)^2} < \phi_r^{ewtcp}(\mathbf{x}_s) \quad \forall r \in s \tag{20}$$

According to the fluid model, the window size is related to RTT and error rate, hence the window sizes of paths will be equalized when paths present equal RTTs and error rates. Under these conditions, we conclude that $\sum_{k\in S} x_k^2 = x_k^2|s|$ and in turn:

$$\phi_r^{coupled}(\mathbf{x}_s) = \phi_r^{e-nmcc}(\mathbf{x}_s) < \phi_r^{LIA}(\mathbf{x}_s) \quad \forall r \in s \tag{21}$$

The analysis shows that e-NMCC addresses TCP-friendliness more effectively than LIA when the sub-flows experience

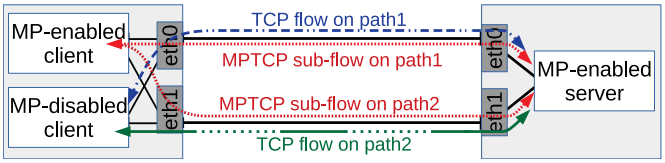

Fig. 2. Experimental topology for evaluating MPTCP in Linux. Due to space constraints, we plot only one MPTCP connection and two TCP connections; in the experiments we deploy four MPTCP and four TCP connections in total.

identical error rate and latency. On the other hand, we cannot definitely conclude that e-NMCC is the most TCP-friendly algorithm (and the least responsive) under all conditions. In what follows, we resort to experimentation to delve deeper into its friendliness and responsiveness properties.

VI. Evaluation in the Linux Kernel

To compare the convergence latency of e-NMCC against other multipath congestion control algorithms, we conducted experiments using the Linux kernel implementation of MPTCP.[7] For each algorithm, we investigated the correlation between the convergence latency of MPTCP and several parameters, such as route propagation delay, bandwidth, error rate, number of sub-flows and background traffic.

A. Testbed Setup

To avoid biases due to the hardware and software used, we configured a testbed where the same hardware was used by multipath and single-path connections. We cloned a virtual machine (VM) that runs MPTCP v0.91 in Linux kernel v.4.1.37 and hosted two such clones as MPTCP clients on the same host machine allocating identical resources to both VMs (2 CPU cores at 4 GHz and 2 GB of RAM). A third clone of this VM, acting as the MPTCP server, was hosted by a different machine, ending up with the topology of Fig. 2. We configured and assigned IP addresses to these VMs so that we could establish two sub-flows between each client-VM and the server-VM, even though MPTCP was enabled in one client-VM only, as shown in the figure.

In each experiment, we simultaneously initiated 4 iperf[8] connections from each client-VM, thus creating 2 single-path flows and 4 multipath sub-flows over each path. Each run lasted 30 minutes and was repeated 30 times. We deployed TCP Reno in the MP-disabled host and MPTCP with one of the e-NMCC, LIA and OLIA algorithms in the MP-enabled host. We also tested MPTCP with the (unfriendly) UNCOUPLED algorithm, where each sub-flow uses the Reno algorithm (MP-RENO), to establish a performance baseline.

The netem[9] tool was used to emulate different propagation delays, link capacities and packet loss rates. Different configurations of these parameters were expected to have an impact on the convergence time of the congestion algorithms. In the default configuration, the link capacities were set to 8 Mbps, the RTT to 100 ms and the error rate to 0%.

[7]https://www.multipath-tcp.org/
[8]https://iperf.fr/
[9]http://man7.org/linux/man-pages/man8/tc-netem.8.html

CISCO's TREX[10] traffic generator was used to simulate background traffic. TREX generates Layer 4-7 traffic based on pre-processing and smart replay of real traffic templates. More specifically, TREX replays numerous connections of different types, such as HTTPS, DNS and RTP, each type exhibiting distinct frequency, size and RTT properties. By proportionally adjusting the launch frequency of the different connection types, we can vary the volume of generated traffic, while retaining the characteristics of real traffic templates.

B. Impact of Propagation Delay

We first investigate the impact of path propagation delay on the convergence of MPTCP. We tested three delay setups, namely, 10 ms, 100 ms and 200 ms, with equal delays in both paths. The results are plotted in Fig. 3, where the rows and columns depict the performance of different algorithms under different path latencies, respectively. Specifically, each plot illustrates the average bandwidth share of the MPTCP and TCP RENO flows, normalized to the overall traffic load, for every second of the experiment.

Confirming our intuition, e-NMCC and MP-RENO connections are not affected by network latency, converging very fast to friendliness and unfriendliness, respectively. LIA, OLIA and BALIA on the other hand, exhibit high convergence latency times that reach 600 s and 1300 s, under 10 ms and 200 ms latencies, respectively. As expected, increasing network delay leads to increased convergence time, since congestion feedback is more frequent for lower latency paths. While LIA and OLIA are TCP-friendly in the long run (BALIA is over-aggressive), they do not achieve fairness until some minutes into a session, thus being unfriendly for connections that last less than 600 s. Finally, the resource utilization is roughly 1% more than MP-RENO for all four friendly algorithms.

C. Impact of Link Bandwidth

We next investigate the impact of link bandwidth on MPTCP convergence. We tested three setups with the same transfer rate in both paths, 4, 8 and 16 Mbps, and equal delays. Figure 4 shows the results (the 8 Mbps case is in column b of Fig. 3).

We observe that e-NMCC is not affected by the amount of available bandwidth, as it converges to its steady state as fast as MP-RENO in both narrow and wide links. However, LIA, OLIA and BALIA are struggling in narrow links, exhibiting unfriendliness for roughly 1400 s, 800 s and 1100 s, respectively, until their bandwidth share is stabilized. By design, MPTCP is slightly more aggressive in wide links, as MP-RENO gains more than its theoretical perfect share (0.66%). This explains the relative over-aggressiveness of OLIA and e-NMCC. However, BALIA is again noticeably over-aggressive, grasping 56% of the bandwidth in wide links. Finally, although we repeated each experiment 30 times, the graphs for narrower links indicate more fluctuation, implying that the increased flow competition in narrow links challenges convergence. The resource utilization under all four friendly algorithms is 1-2% more than what MP-RENO achieves.

[10]https://trex-tgn.cisco.com/

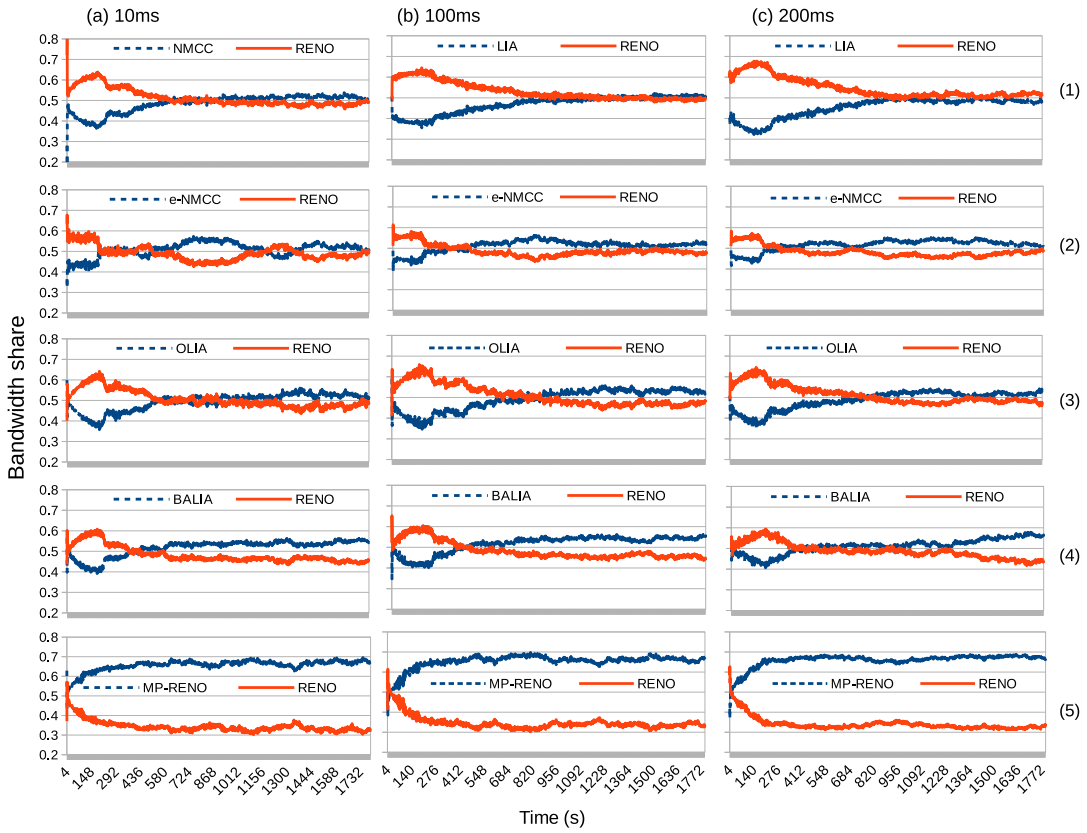

Fig. 3. MPTCP performance in a LAN testbed with two disjoint paths, different congestion control algorithms (rows 1-5) and propagation delays (columns a-c).

D. Impact of Packet Loss

We then investigate the impact of uniformly distributed path losses on MPTCP convergence. We tested three setups with the same path packet drop rates, namely, no loss, 0.0001% and 0.1%. Figure 5 shows the results (the no loss case is in column b of Fig. 3). We observe that e-NMCC is not affected by the error rate, as it converges to its steady state as fast as MP-RENO in all cases. In contrast, LIA and OLIA deviate most from their fair share in the beginning of the experiment with a small packet drop rate, although their convergence time is not altered with regard to the zero loss scenario. On the other hand, all algorithms adapt instantly to their steady state under more frequent losses, since frequent losses accelerate the detection of network condition changes, allowing the algorithms to adapt rapidly. However, note that both LIA and OLIA are consistently less aggressive than needed, getting 47% and 48% of the bandwidth, respectively. Conversely, e-NMCC is slightly over-aggressive (53%), while BALIA achieves perfect sharing. In the case of 0.1% drop rate, the resource utilization under all friendly algorithms is 1-2% more than what MP-RENO achieves. With the 0.0001% drop rate, the resource utilization of MP-RENO is roughly 10% less than the friendly algorithms, implying that over-aggressiveness can penalize performance when congestion events occur less frequently.

E. Impact of Heterogeneous Paths

We next investigate the impact of path heterogeneity on the convergence of MPTCP. We tested three setups with heterogeneity in terms of bandwidth, error rate or propagation delay. Figure 6 shows the results when error rates (column a) and RTTs (column b) vary across paths; the results with different bandwidths are similar to column b of Fig. 3. In both setups e-NMCC converges as fast as MP-RENO. In contrast, on paths with different error rates LIA, OLIA and BALIA converge much slower to their steady state, requiring roughly 400 s, 900 s and 800 s until their throughput is stabilized. In paths with different RTTs, LIA, OLIA and BALIA present even slower convergence, taking approximately 1000 s to reach steady state. Finally, the resource utilization is roughly 1% more than MP-RENO for all friendly algorithms.

F. Impact of Background Traffic

We then investigate the impact of background traffic on the convergence of MPTCP. We evaluate MPTCPs performance under three different levels of congestion, namely, when TREX traffic constitutes approximately 80%, 40% and 20% of the available bandwidth.[11] Given that TREX does not perform congestion control, hence the TREX traffic is unresponsive to network congestion, we deploy real TCP flows along with MPTCP ones in order to make the race for resources realistic.

Figure 7 shows the results in the three setups. We first notice that convergence is faster when the background traffic volume is higher, causing higher error-rates and less room

[11]TREX traffic is highly dynamic, presenting frequent bursts. The indicated percentages represent the maximum seen ratios during the experiments.

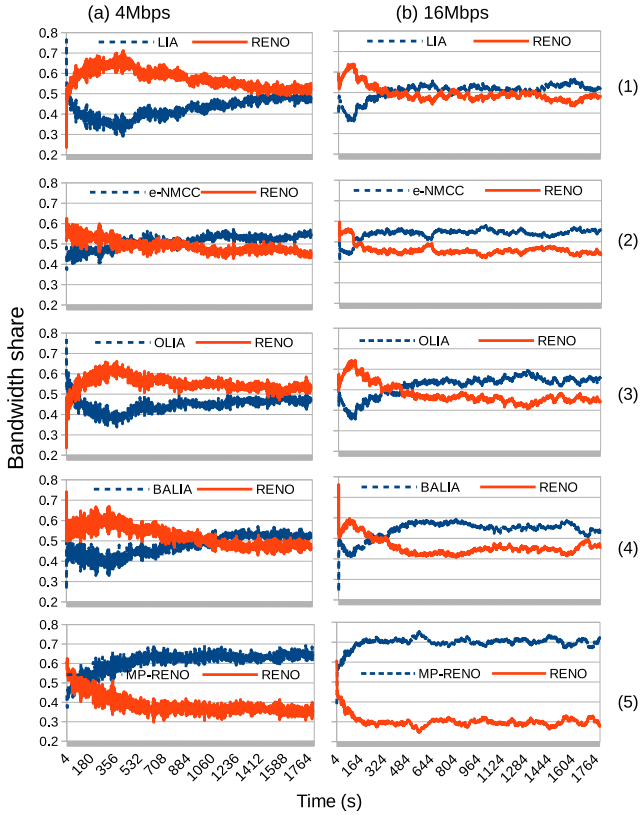

Fig. 4. MPTCP performance in a LAN with two disjoint paths, different congestion control algorithms (rows 1-5) and transfer rates (columns a-b).

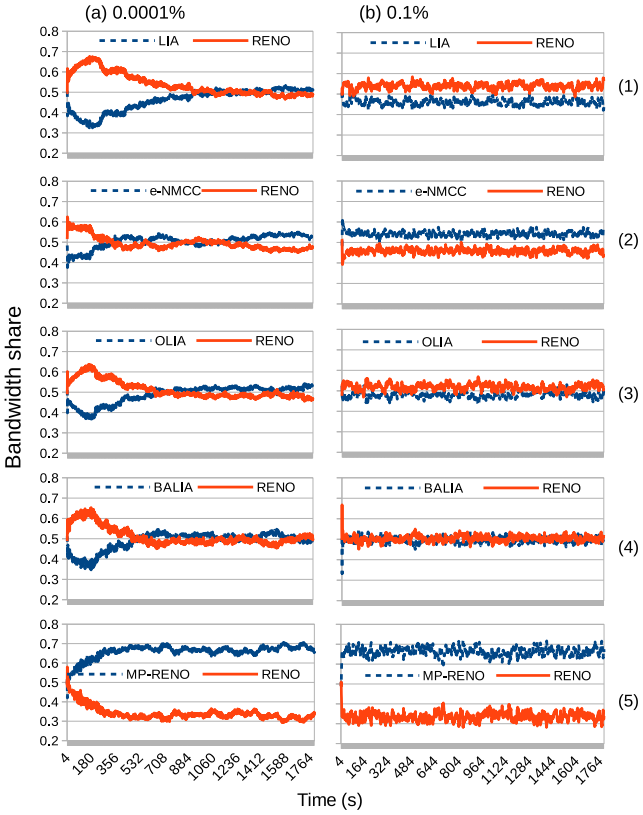

Fig. 5. MPTCP performance in a LAN with two disjoint paths, different congestion algorithms (rows 1-5) and packet drop rates (columns a-b).

for connections to compete; the same correlation is observed in Fig. 5. The convergence of e-NMCC is almost instant (compared to RENO) in all setups. When TREX traffic is low (column a), LIA and BALIA require approximately 600 s to stabilize, while OLIA converges roughly after 300 s. With regard to friendliness accuracy, LIA and OLIA grasp less than their fair share of resources when moderate background traffic is generated (Fig. 7(a-b)), thus performing poorly compared to TCP; this issue was not met in previous setups. MPTCP with NMCC grasps slightly more resources than TCP (up to 57%) regardless of the background traffic volume.

G. Impact of a Third sub-flow

Finally, we investigate the impact of a third route on the convergence of MPTCP. We extended our testbed by adding a third network interface at each node in order to deploy a third sub-flow. We tested three setups with different configurations. Figure 8 shows the results when paths are identical (column a), when latency varies (column b) and when bandwidth varies (column c) across paths. The results of column a differ from the equivalent setup with two paths (Fig. 3(b)), thus showing that both the accuracy and convergence of MPTCP friendliness are affected by the third route. The convergence of MPTCP with MP-RENO is slower, requiring roughly 500 s, and even more unfriendly, taking roughly 75% of resources, thus showing that MPTCP can become unfriendly regardless of congestion control. In all setups, we observe two consecutive performance phases for all friendly algorithms; in the initial phase, TCP increases its bandwidth share, while, in the later phase, MPTCP increases its share. The phase transition happens roughly simultaneously for all algorithms, transitioning the soonest and the latest in the setup of column c and a, respectively. In all setups, the transition takes place faster for e-NMCC, showing that it responds faster than the other algorithms in this setup too. With regard to friendliness, the benchmark algorithm, MP-RENO, is 10-15% more aggressive that its theoretical performance, hence we can not safely conclude that e-NMCC is unfriendly due to getting 10-20% more resources that its prefect share.

H. Discussion of results

Our evaluation yielded interesting results about the accuracy and the speed at which the different algorithms achieve TCP-friendliness. In general, these results confirm that friendliness towards single-path connections is respected by LIA, OLIA, BALIA and e-NMCC, but the schemes vary considerably as to how fast they reach it.

First, we point out that the Linux MPTCP implementation inherently introduces some convergence latency. Even MP-RENO that does not include any friendliness mechanism requires several seconds to enter the steady state. Second, we found that e-NMCC converges as quickly as possible, in the sense that it reaches stability simultaneously with MP-RENO in all experiments, except for when three sub-flows are deployed. In the last case, e-NMCC delivers friendliness instantly

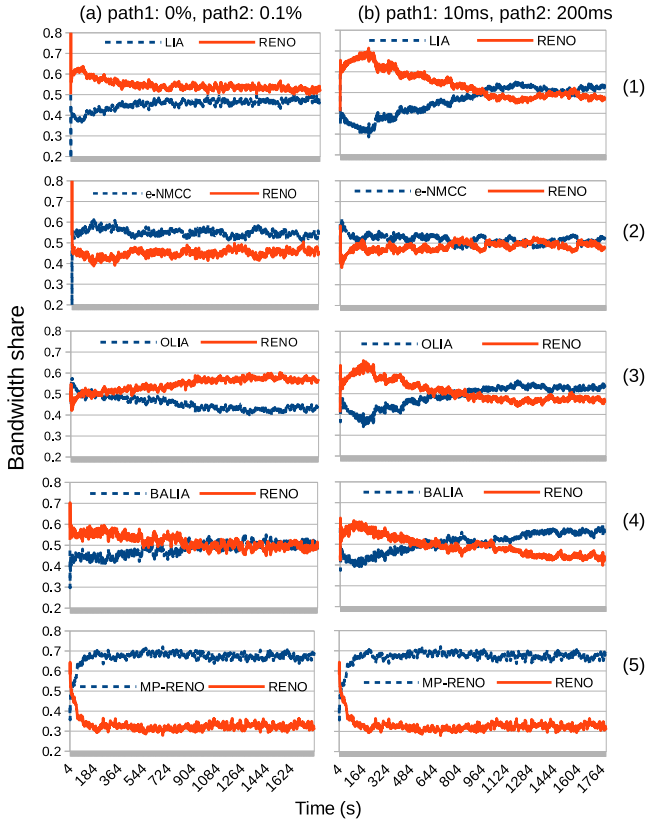

Fig. 6. MPTCP performance in a LAN with two disjoint paths, different congestion control algorithms (rows 1-5) and heterogeneous paths in terms of error-rate (column a) and RTT (column b).

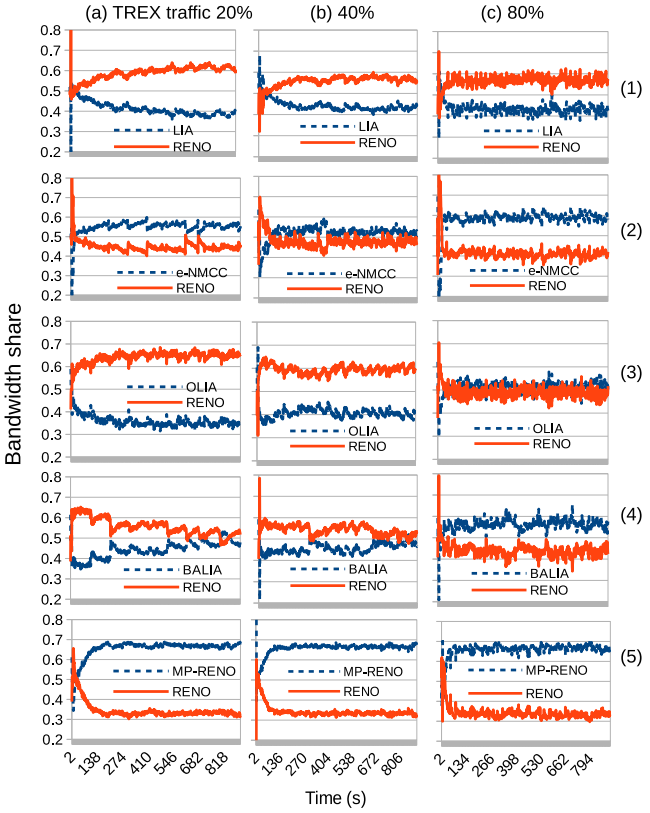

Fig. 7. MPTCP performance in a LAN with two disjoint paths, different congestion control algorithms (rows 1-5) and different volumes of background traffic (columns a-c).

but becomes unfriendly after roughly 400 s. Third, LIA, OLIA and BALIA converge to TCP-friendliness considerably more slowly than e-NMCC, the respective lag ranging from roughly 200 s (Fig. 3(a), 4(b)) up to 1000 s (Fig. 3(c), 4(a)). Their performance is better over fat links with low propagation delay, deteriorating noticeably as paths get narrower and longer. However, all algorithms converge instantly when both paths include high error-rate links (Fig. 5(b)) since the frequent congestion events enable faster estimation of the path properties.

In terms of TCP-friendliness consistency, e-NMCC exhibits fewer deviations across different setups compared to LIA, OLIA and BALIA. Specifically, e-NMCC is slightly more aggressive than single-path TCP in all scenarios, grasping roughly 5% more resources than the perfect share. This mild over-aggressiveness is an inherent characteristic of e-NMCC when deployed in disjoint paths (where friendliness is not an issue). Thereby, it normalizes the performance of the fastest available path, a non-fixed characteristic over prolonged periods, and gains a slight performance advantage due to path diversity. On the other hand, the performance of LIA, OLIA and BALIA varies across experiments, being faster than single-path in some scenarios (Fig. 4(b)) and slower in others (Fig. 4(a)), indicating that their friendliness scores are sensitive to network conditions. Under sensible error rates, BALIA is the most accurate (albeit, slow) algorithm, splitting the network resources evenly. When background traffic is considered, LIA and OLIA are overly friendly (Fig. 7(a-b)).

VII. Evaluation with the *htsim* Simulator

We next compare e-NMCC with LIA, NMCC, and MP-RENO (UNCOUPLED) in five benchmark scenarios from LIA's evaluation [7] using the same simulator, *htsim*.[12] As htsim focuses on congestion control, ignoring packet scheduling, it is lightweight and simulates large flows quickly, an important property when studying performance in steady state. The benchmark scenarios investigate MPTCP behavior in terms of TCP-friendliness, resource utilization and load balancing.

Each scenario was repeated 500 times and each run lasted 1000 s; we consider only the final 200 s, to assess the connections in steady state. The htsim simulator uses a coarse grained *Retransmission TimeOut* (RTO) estimation, therefore we consider only flows with non-frequent timeouts, as otherwise timeouts would be incorrectly grouped in time. The congestion events reported by htsim are 99% triple duplicates and 1% timeouts, even though timeouts typically outnumber triple duplicates in the Internet [20]. The frequency of triple duplicates is expected to magnify the friendliness issue of NMCC (Section IV), since Fast Retransmit controls congestion in a more timely manner, preventing global congestion events.

A. TCP-friendliness

The first topology (see [7, Fig. 1] and Fig. 9(a')) uses a single bottleneck to investigate resource sharing between a

[12]OLIA and BALIA are not implemented in htsim.

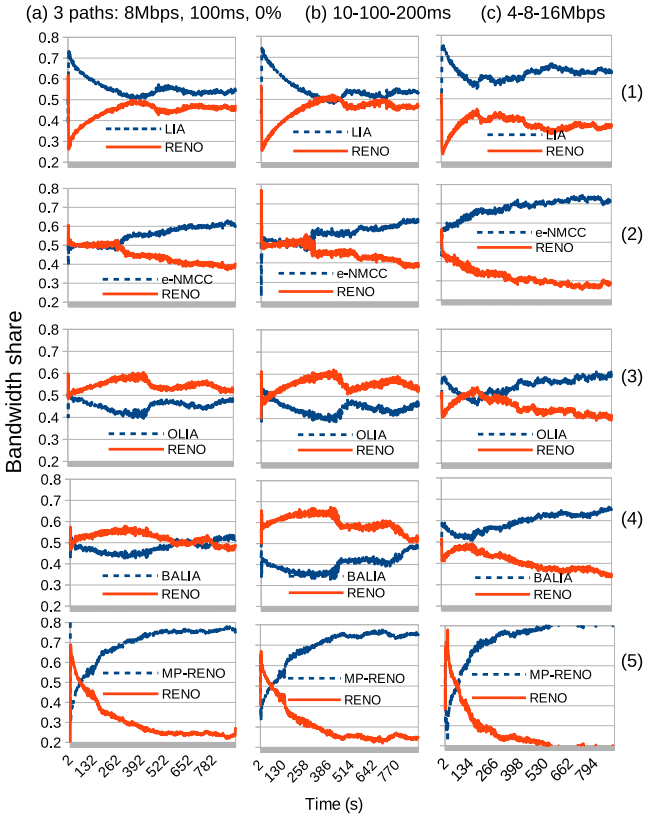

Fig. 8. MPTCP performance in a LAN with three disjoint paths, different congestion control algorithms (rows 1-5), identical paths (column a) and paths differing in latency (column b) or bandwidth (column c).

multipath (with two sub-flows) and a single-path connection. We replicated the experiment and found that e-NMCC exhibits perfect sharing, grasping 50% of the available resources, while LIA is slightly more aggressive, grasping 53%, as shown in Fig. 9(a); the results also validate the conclusions of the analysis in section V. NMCC exhibits the expected over-aggressiveness due to partial congestion events, thus getting 60% of resources (global congestion events are roughly 50% of total events). We repeated the experiment for link throughputs of 200-1000 packets/s, without measurable differences.

B. Resource Utilization

The second topology (see [7, Fig. 4] and Fig. 9(b')) uses two mismatched paths to explore resource utilization by the multipath connection, which competes with a single-path connection on each path. All algorithms achieve 100% resource utilization but offer different levels of friendliness: e-NMCC gets 53% of the resources on the widest path, LIA gets 52%, NMCC 59% and MP-RENO 59%. The results are presented in Fig. 9(b), where flow throughput is normalized to the bandwidth of the widest path. NMCC performs similarly to MP-RENO for two reasons: first, the RTT of the narrow path is ten times the RTT of the wide path, hence $m \simeq 1$, and, second, disjoint paths, by definition, generate partial congestion events.

The third topology (see [7, Fig. 2] and Fig. 9(c')) uses three links of capacity C to evaluate resource utilization as a result of choosing the least-congested path. Specifically, three multipath connections are deployed, each having one sub-flow through one link and a second one through the other two links, so that each link is used by three sub-flows. Ideally, each multipath session should use only the least congested path (the single link) and get a cumulative transfer rate of C, instead of using the two links shared by the other sub-flows, which would lead to a transfer rate of only $2C/3$. Figure 9(c) shows the throughput of each sub-flow normalized to C. The results indicate that the algorithms do not maximize resource utilization, but LIA performs slightly better than e-NMCC, which performs sightly better than NMCC and MP-RENO, scoring 81%, 79%, 77% and 77%, respectively. While this under-utilization is the core motivation for OLIA [8], the importance of pushing traffic exclusively to the "best" path is debatable, as it can lead to poor responsiveness [13, Fig. 4] and penalize performance in datacenters [7].

C. Load balancing

The fourth topology (see [7, Fig. 3] and Fig. 9(d')) uses four parallel links with different capacities to estimate the load balancing efficiency of the multipath algorithm. Three multipath connections are deployed, each competing with two different multipath connections on two different links. Ideally, the connections will balance congestion across all links and perform similarly, achieving a cumulative capacity of C. Figure 9(d) illustrates the performance of the three connections normalized to C. e-NMCC utilizes 98% of the network resources, while the other algorithms reach 100%. The slight performance degradation occurs at the connection using the widest paths, thus achieving better load balancing (st. deviation of multipath connections' performance is 12%, 14%, 15% and 17% for e-NMCC, LIA, NMCC and MP-RENO accordingly), at the cost of lower resource utilization.

Finally, the fifth topology (see [7, Fig. 7] and Fig. 9(e')) is a torus with five parallel links, again used to assess load balancing efficiency. Each link is used by two sub-flows from different connections, but one link is considerably narrower. The multipath connections must balance congestion in all links and perform similarly. The results are presented in Fig. 9(e), which illustrates the throughput of the five flows normalized to the fastest single-path measured. In all cases, resource utilization is at 100% but throughput is not perfectly equalized, as the connections sharing the narrow link differ from the average, with the standard deviation being 9%, 9%, 12% and 15% for e-NMCC, LIA, NMCC and MP-RENO, respectively.

D. Discussion of results

The results of the simulations in the benchmark topologies indicate that the TCP-friendliness goals of e-NMCC are met under various conditions, with efficient resource utilization and load balancing. We observe that e-NMCC is slightly better than LIA (and every other algorithm in our evaluation) in sharing the bottlenecks, since it tackles friendliness nearly perfectly (Fig. 9(a)), but it exhibits a minor resource underutilization compared to LIA (2% less in Fig. 9(c)) due to not pushing all traffic in the least congested path, while

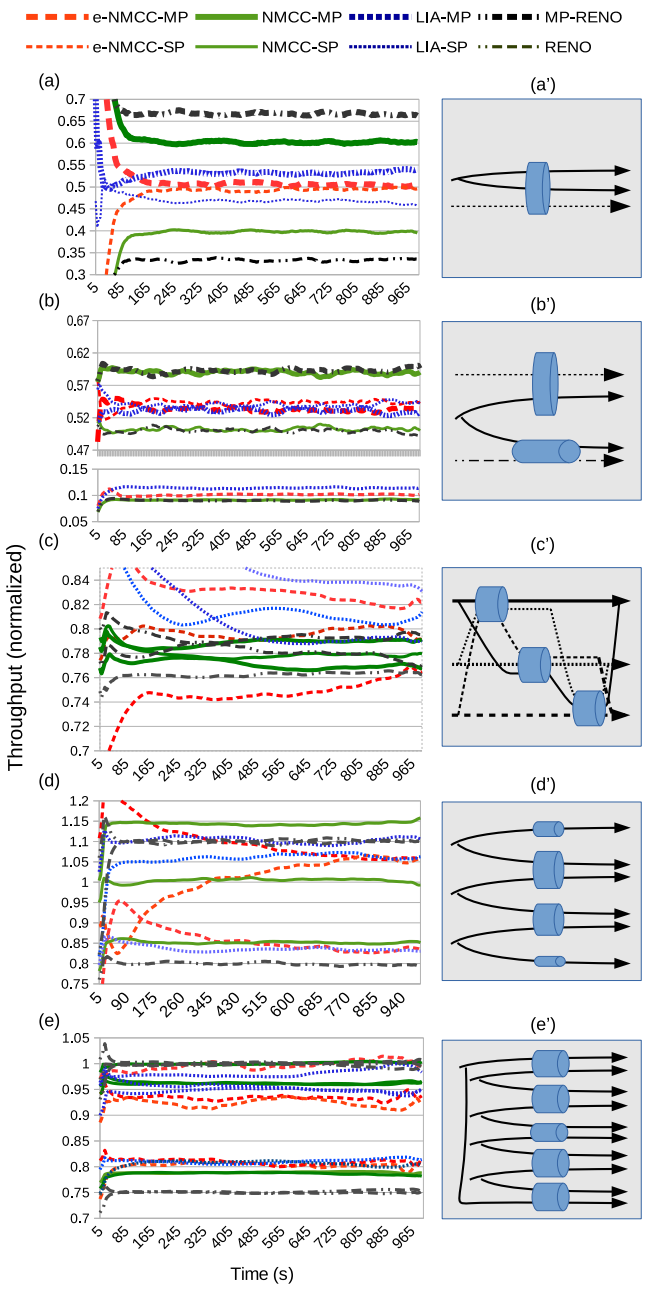

Fig. 9. MPTCP performance in the benchmark topologies of [7]. Figures illustrate the instant bandwidth share of multipath (MP) and single-path (SP) connections normalized to the overall network capacity, unless otherwise stated. Figure (a) examines TCP-friendliness, (b)-(c) resource utilization and (e)-(d) load balancing.

offering similar load balancing with LIA (Fig. 9(d,e)). This behavior is reasonable, since e-NMCC is designed to offer instant and accurate TCP-friendliness, instead of using only the "best" path. Therefore, e-NMCC is friendly to single-path flows throughout the entire connection lifespan, as it exploits all paths proportionally, thus offering enhanced responsiveness and high performance, even in cases of RTT-mismatch [12].

VIII. Integration with Linux MPTCP

e-NMCC can be directly integrated in the Linux kernel, as TCP (and MPTCP) offers pluggable congestion control via a special handler interface [21]. The e-NMCC code simply overrides the handlers estimating the window increase upon the successful delivery of an ACK and the Slow Start threshold upon a congestion event. The information required to calculate m and m_{th}, such as the RTT, congestion window and Slow Start threshold of the sub-flows, is available to each sub-flow through the parameters *srtt_us*, *snd_cwnd* and *ssthresh* respectively, hence the implementation is direct.[13]

The primary challenge is converting the normalized window growth algorithm from RTT-based to packet-based. Assuming a sub-flow in Congestion Avoidance where the throughput increase rate with NMCC is

$$\frac{1}{r'^2} = \frac{1}{(mr)^2} = \frac{1/m^2}{r^2}$$

the increase of a friendly congestion window is $1/m^2$ over the unmodified RTT. Measuring the congestion window, w, in bytes, TCP grows by MSS^2/w bytes, w/MSS times within an RTT, for an overall growth of 1 MSS. By reducing the amount of per-ACK increase of a sub-flow to $MSS^2/(m^2w)$ bytes, the cumulative increase of e-NMCC within an RTT is MSS/m^2, thus satisfying the friendliness requirement. Hence, the friendliness factor m can be used to directly control the growth of the congestion window upon the receipt of an ACK, thus allowing e-NMCC to be integrated with TCP-like transport protocols. We similarly adapt the e-NMCC algorithm to handle the Slow Start phase.

The second challenge is estimating the segment loss probability, in order to ensure friendliness during partial congestion events. We can approximate the segment loss probability of a sub-flow through the *max_packets_out* parameter, which holds the maximum number of on-the-fly packets of the previous congestion round, thus approximating the maximum window during that round. Assuming that throughput increases and reductions balance out in steady state [7, Eq. 2], the relationship of window size and error rate is $w = \sqrt{2/p}$, hence we can estimate p as:

$$p = 2/max\_packets\_out^2$$

The pseudocode for estimating m_{th} in Linux MPTCP is presented in Algorithm 1.

IX. Conclusions

We have focused on a drawback shared by the probabilistic multipath congestion control of LIA, OLIA and BALIA, namely, the long convergence period to TCP-friendliness. Unlike previous studies dedicated to steady state results, we monitored the instantaneous performance of the connections throughout a session, using the real Linux implementation of MPTCP. In most cases, the convergence latency was on the order of minutes, questioning the effectiveness of these algorithms with shorter flows.

The previously proposed NMCC algorithm meets the TCP-friendliness objective instantly, but may become TCP-unfriendly when congestion events affect only a subset of

[13]The source code of e-NMCC for Linux is available at https://mm.aueb.gr/software.

Algorithm 1 Estimation of m_{th} upon loss.

procedure ESTIMATE A & B
  $A \leftarrow 0$, $B \leftarrow 0$, $p_{max} \leftarrow 0$
  for $(r \in subflows)$ **do**
    $r_t \leftarrow srtt\_us_r$
    $w_r \leftarrow snd\_cwnd_r$
    $p_r \leftarrow 2/max\_packets\_out_r^2$
    $A \leftarrow A + w_r/r_r$
    $B \leftarrow B + p_r * w_r^2/r_r^2$
    if $p_{max} < p_r$ **then**
      $p_{max} \leftarrow p_r$
    end if
  end for
  $m_{th} \leftarrow p_{max} * A * A/B$
  $m_{th} \leftarrow \max(1, m_{th})$ ▷ Avoid over-aggressiveness
  $m_{th} \leftarrow \min(2, m_{th})$ ▷ Avoid infeasible decrease
end procedure

the sub-flows. We have, therefore, introduced e-NMCC, which caters for TCP-friendliness upon both throughput increase and decrease epochs, thus providing a comprehensive solution.

We characterized mathematically the behavior of the proposed protocol in terms of TCP-friendliness, responsiveness and utilization. Using the Linux implementation of MPTCP, we have shown that e-NMCC offers fair resource sharing timely, accurately, and consistently, thus being effective for all types of flows, regardless of their duration. Finally, using the *htsim* simulator, we explored the performance of e-NMCC under a set of well-known benchmark scenarios, validating that TCP-friendliness is enhanced, while resource utilization and load balancing (in the long-run) are comparable to LIA.

REFERENCES

[1] V. Jacobson, "Congestion avoidance and control," in *Proc. of the ACM SIGCOMM*, August 1988.

[2] T. Henderson, E. Sahouria, S. McCanne, and R. Katz, "On improving the fairness of TCP congestion avoidance," in *Proc. of the IEEE GLOBECOM*, August 1998.

[3] J. Widmer, R. Denda, and M. Mauve, "A survey on TCP-friendly congestion control," *IEEE Network*, vol. 15, no. 3, pp. 28–37, 2001.

[4] J. Qadir, A. Ali, K.-L. A. Yau, A. Sathiaseelan, and J. Crowcroft, "Exploiting the power of multiplicity: a holistic survey of network-layer multipath," *IEEE Communications Surveys & Tutorials*, vol. 17, no. 4, pp. 2176–2213, 2015.

[5] A. Ford, C. Raiciu, M. Handley, S. Barre, and J. Iyengar, "Architectural guidelines for multipath TCP development," IETF, RFC 6182, 2011. [Online]. Available: http://www.rfc-editor.org/rfc/rfc6182.txt

[6] C. Xu, J. Zhao, and G.-M. Muntean, "Congestion control design for multipath transport protocols: a survey," *IEEE Communications Surveys & Tutorials*, vol. 18, no. 4, pp. 2948–2969, 2016.

[7] D. Wischik, C. Raiciu, A. Greenhalgh, and M. Handley, "Design, implementation and evaluation of congestion control for multipath TCP," in *Proc. of the USENIX NSDI*, March 2011.

[8] R. Khalili *et al.*, "Opportunistic linked-increases congestion control algorithm for MPTCP," IETF, Internet Draft, 2015. [Online]. Available: https://www.ietf.org/archive/id/draft-khalili-mptcp-congestion-control-05.txt

[9] F. Kelly and T. Voice, "Stability of end-to-end algorithms for joint routing and rate control," *ACM SIGCOMM Computer Communication Review*, vol. 35, no. 2, pp. 5–12, 2005.

[10] B. Hesmans, H. Tran-Viet, R. Sadre, and O. Bonaventure, "A first look at real Multipath TCP traffic," in *Proc. of International Workshop on Traffic Monitoring and Analysis (TMA)*, April 2015.

[11] S. Ihm and V. S. Pai, "Towards understanding modern web traffic," in *Proc. of ACM IMC*, June 2011.

[12] Y. Thomas, G. Xylomenos, C. Tsilopoulos, and G. C. Polyzos, "Multi-flow congestion control with network assistance," in *Proc. of the IFIP Networking*, May 2016.

[13] Q. Peng, A. Walid, J. Hwang, and S. H. Low, "Multipath TCP: Analysis, design, and implementation," *IEEE/ACM Transactions on Networking*, vol. 24, no. 1, pp. 596–609, 2016.

[14] M. Honda, Y. Nishida, L. Eggert, P. Sarolahti, and H. Tokuda, "Multipath congestion control for shared bottleneck," in *Proc. of the PLFDNeT Workshop*, May 2009.

[15] H. Han, S. Shakkottai, C. V. Hollot, R. Srikant, and D. Towsley, "Multi-path TCP: a joint congestion control and routing scheme to exploit path diversity in the internet," *IEEE/ACM Transactions on Networking*, vol. 14, no. 6, pp. 1260–1271, 2006.

[16] Y. Cao, M. Xu, and X. Fu, "Delay-based congestion control for multipath TCP," in *Proc. of the IEEE ICNP*, October 2012.

[17] M. Allman, V. Paxson, and E. Blanton, "TCP congestion control," IETF, RFC 5681, 2009. [Online]. Available: http://www.rfc-editor.org/rfc/rfc5681.txt

[18] H. Jiang and C. Dovrolis, "Passive estimation of TCP round-trip times," *ACM SIGCOMM Computer Communication Review*, vol. 32, no. 3, pp. 75–88, 2002.

[19] J. Padhye, V. Firoiu, D. Towsley, and J. Kurose, "Modeling TCP throughput: A simple model and its empirical validation," *ACM SIGCOMM Computer Communication Review*, vol. 28, no. 4, pp. 303–314, 1998.

[20] ——, "Modeling TCP Reno performance: a simple model and its empirical validation," *IEEE/ACM Transactions on Networking*, vol. 8, no. 2, pp. 133–145, 2000.

[21] S. Arianfar, "TCP's congestion control implementation in Linux kernel," in *Proc. of the Seminar on Network Protocols in Operating Systems*, January 2012.