Alexander Frömmgen, Boris Koldehofe
Demo: Programming Application-defined Multipath TCP Schedulers In: Proceedings ACM/IFIP/USENIX Middleware, Demo Track, December 2017

Demo: Programming Application-defined Multipath TCP Schedulers

Alexander Frömmgen
KOM – TU Darmstadt
alexander.froemmgen@kom.tu-darmstadt.de

Boris Koldehofe
KOM – TU Darmstadt
boris.koldehofe@kom.tu-darmstadt.de

Abstract

Multipath TCP is the de facto multipath protocol in today's Internet. In this demo, we show how the recently proposed *ProgMP* programming model for Multipath TCP scheduling enables application- and preference-aware Multipath TCP scheduling within the Multipath TCP Linux Kernel. We use *ProgMP* to systematically derive the specification of a novel scheduler which retains an application-specific acceptable round-trip time and subflow preferences. This scheduler only utilises non-preferred (e.g., metered cellular) subflows if all preferred subflows (e.g., WiFi) do not retain the acceptable upper round-trip time. We further evaluate this novel scheduler by executing the scheduler specification within the *ProgMP* runtime in the Multipath TCP Linux Kernel.

CCS Concepts • **Networks → Transport protocols**;

Keywords Multipath TCP, Scheduling, Specification Language

ACM Reference format:
Alexander Frömmgen and Boris Koldehofe. 2017. Demo: Programming Application-defined Multipath TCP Schedulers. In *Proceedings of Middleware Posters and Demos '17: Proceedings of the Posters and Demos Session of the 18th International Middleware Conference, Las Vegas, NV, USA, December 11–15, 2017 (Middleware Posters and Demos '17),* 2 pages.
DOI: 10.1145/3155016.3155017

1 Introduction

Multipath TCP (MPTCP) is the de facto multipath protocol in today's Internet. MPTCP splits traffic of a single logical connection on multiple TCP subflows, improving throughput and reliability, e.g., by using subflows on different network paths. On the sending side, the MPTCP *scheduler* maps outgoing packet on subflows. These packets rejoin at the receiver side, where in-order delivery to the application is ensured with global sequence numbers.

Today's MPTCP implementation in the Linux Kernel [6] provides three schedulers: The default scheduler, denoted *minRTT*, uses the subflow with the minimum round-trip time (RTT) and not exhausted its congestion window [7, 8]. The round-robin scheduler [7] chooses subflows in a round-robin fashion. Finally, the redundant scheduler sends packets redundantly on all subflows [3, 5]. These schedulers target improved throughput or latency while providing restricted preference-awareness. *ProgMP*[1], a programming model for Multipath TCP scheduling, was recently proposed to enable more flexible, application- and preference-aware Multipath TCP schedulers [4]. *ProgMP* provides a scheduler specification language, an execution environment in the Multipath TCP Linux Kernel, and an extended socket API for application-defined scheduling.

In this demo paper, we use *ProgMP* to exemplary develop a novel application- and preference-aware scheduler. Starting with an illustrating first scheduler, we analyse the dynamic scheduling environment with *ProgMP*'s PRINT functionality and derive a scheduler which optimizes for an application-defined acceptable upper round-trip time while considering subflow preferences.

[1]https://progmp.net provides a detailed language and framework overview.

2 A First *ProgMP* Scheduler Example

Figure 1 provides a *ProgMP* specification for a *lowest round-trip time first* scheduler (*MinRTT*), which basically behaves like the MPTCP default scheduler [7, 8]. The specified scheduler determines all subflow candidates which have not exhausted their congestion window by filtering the set of all subflows (`sbfCandis` in line 1–3). In case at least one subflow is available (line 5), the scheduler PUSHs the first packet from the sending queue Q on the subflow with the minimum round-trip time (line 11–12). A detailed presentation of all language primitives and a discussion of the reinjection queue handling is presented in [2].

```
1  VAR sbfCandis = SUBFLOWS.FILTER(sbf =>
2    sbf.CWND > sbf.SKBS_IN_FLIGHT + sbf.QUEUED AND
3    !sbf.THROTTLED AND !sbf.LOSSY);
4  IF(sbfCandis.EMPTY) { RETURN; }
5
6  ...         /* Reinjection queue handling omitted */
7
8  IF (!Q.EMPTY) {
9    sbfCandis.FILTER(sbf => sbf.HAS_WINDOW_FOR(Q.TOP)).
10     MIN(sbf => sbf.RTT).PUSH(Q.POP()); }
```

Figure 1. ProgMP specification of a *MinRTT* scheduler.

3 Analysing Scheduling Behavior

In order to analyse the scheduling behavior and decisions, we extend the previous scheduler with PRINT statements (Figure 2). PRINTs enable a convenient and efficient analysis during the scheduler execution in the Linux Kernel. To reduce the induced overhead, we restrict PRINT operations to occur every 200ms (line 1–3).

```
1  VAR interPrintInterval = 200;
2  IF (R6 < CURRENT_TIME_MS) {
3     SET(R6, CURRENT_TIME_MS + interPrintInterval);
4     PRINT("Number of subflows is %d", SUBFLOWS.COUNT);
5     FOREACH(VAR sbf IN SUBFLOWS) {
6        PRINT("Subflow with id = %d ...", sbf.ID);
7        PRINT("... has RTT = %d ...", sbf.RTT);
8        PRINT("... and is backup %d.", sbf.IS_BACKUP);
9  }  }
```

Figure 2. *ProgMP* snippet which provides analysis information.

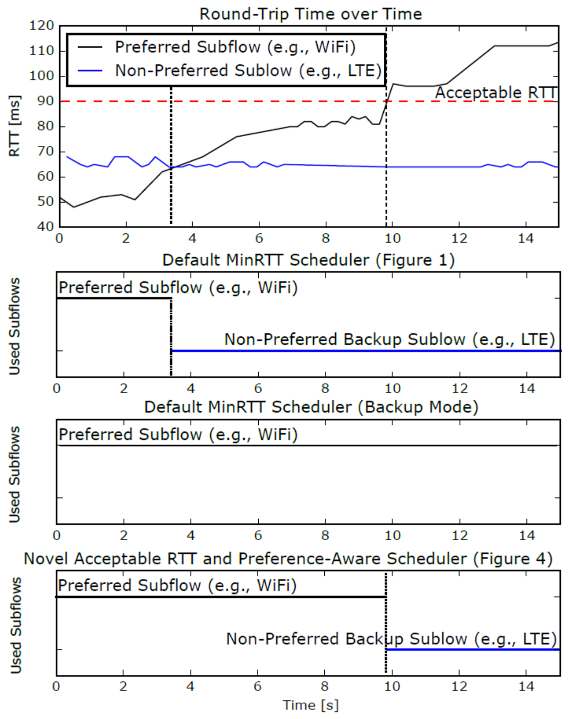

Figure 3. Scheduler comparison for a dynamic Mininet scenario. Compared to the existing *MinRTT* scheduler configurations, the novel scheduler retains the acceptable round-trip time (90ms) and subflow preferences (prefer WiFi).

Figure 3 (top) shows a graphical representation of the PRINT output for an execution in a Mininet emulation. In this Mininet emulation, we systematically increase the round-trip time on the preferred subflow for illustration. The PRINT output reflects these changes. A closer analysis shows that the default *MinRTT* scheduler uses the non-preferred subflow after 3 seconds (Figure 3, II graph). As backup subflows are only used when all non-backup subflows fail, the backup subflow is not used in the presented scenario (Figure 3, III graph). Both examples show that the existing *MinRTT* scheduler is not flexible with regard to the backup subflow usage.

4 Preference-aware Scheduling

In many real-world scenarios, users have preferences with regard to the utilized network resources and corresponding subflows. Users of mobile devices, e.g., often prefer WiFi over metered cellular traffic. In datacenters, paths might be associated with different costs.

Pure throughput optimizing schedulers might aggregate *all* subflows regardless of preferences. However, traffic of interactive applications such as voice-based personal assistant systems and most datacenter request-response patterns usually consist of a few packets which fit on a single subflow. For such traffic, customers do not want to waste their metered cellular traffic in case the round-trip time on the preferred subflows is still acceptable. At the same time, significantly higher round-trip times, as experienced with the backup mode in the example after 10 seconds, reduce the user experience. Around 15% of all measurement samples in a huge recent mobile study experienced a significantly higher RTT on WiFi compared with LTE [1], showing the relevance of round-trip time comparisons between WiFi and LTE. As of today, there is

```
1  VAR sbfCandis = SUBFLOWS.FILTER(sbf =>
2    sbf.CWND > sbf.SKBS_IN_FLIGHT + sbf.QUEUED AND
3    !sbf.THROTTLED AND !sbf.LOSSY AND
4    sbf.HAS_WINDOW_FOR(Q.TOP));
5
6  VAR considerBackup = SUBFLOWS.FILTER(sbf =>
7    sbf.RTT < R1 AND !sbf.IS_BACKUP).EMPTY;
8
9  IF (considerBackup) {
10   VAR backupSbf = sbfCandis.FILTER(sbf => sbf.
11     IS_BACKUP AND sbf.RTT < R1).MIN(sbf => sbf.RTT);
12   IF (backupSbf != NULL) {
13       backupSbf.PUSH(Q.POP());
14       RETURN;
15 } }
16
17 /* no backup found, take best non-backup */
18 sbfCandis.FILTER(sbf => !sbf.IS_BACKUP).
19   MIN(sbf => sbf.RTT).PUSH(Q.POP());
```

Figure 4. *ProgMP* specification of a novel scheduler, which retains an acceptable upper round-trip time and subflow preferences.

no scheduler which schedules packets of interactive applications preference-aware while considering *acceptable* round-trip times.

We propose a novel application- and preference-aware scheduler which retains an acceptable upper round-trip time. *ProgMP* enables a convenient specification of such a scheduler (Figure 4). This scheduler only considers backup subflows if no non-backup subflow has a sufficient round-trip time to retain the acceptable upper round-trip time (line 9). The application can inform the scheduler about the acceptable upper round-trip time by setting the R1 register with *ProgMP*'s extended socket API. Figure 3 (bottom) shows the used subflows for this RTT- and preference-aware scheduler in the Mininet example scenario, given an acceptable RTT of 90ms.

The specified scheduler favors subflow preferences over increased throughput, backup subflows are only used if all non-backup subflows have exhausted their congestion window. *ProgMP* enables the specification of a scheduler which favors throughput over subflow preferences, i.e., relies on backup subflows in case all other subflows have exhausted their congestion window, by replacing a single expression, SUBFLOWS (line 6) with sbfCandis.

These examples show that *ProgMP* enables a convenient specification of application- and preference-aware MPTCP schedulers.

Acknowledgment

This work has been funded by the German Research Foundation (DFG) as part of the projects C2 in the Collaborative Research Center (SFB) 1053 MAKI.

References

[1] S. Deng, R. Netravali, A. Sivaraman, and H. Balakrishnan. WiFi, LTE, or both?: Measuring multi-homed wireless internet performance. In *IMC*, 2014.
[2] A. Frömmgen. ProgMP: Progammable MP Scheduling. https://progmp.net, 2017.
[3] A. Frömmgen, T. Erbshaeusser, T. Zimmermann, K. Wehrle, and A. Buchmann. ReMP TCP: Low Latency Multipath TCP. In *ICC*, 2016.
[4] A. Frömmgen, A. Rizk, T. Erbshaeusser, M. Weller, B. Koldehofe, A. Buchmann, and R. Steinmetz. A Programming Model for Application-defined Multipath TCP Scheduling. In *ACM/IFIP/USNIX Middleware*, 2017.
[5] I. Lopez, M. Aguado, C. Pinedo, and E. Jacob. SCADA Systems in the Railway Domain: Enhancing Reliability through Redundant MultipathTCP. In *ITSC*, 2015.
[6] C. Paasch and S. Barre. Multipath TCP in the Linux Kernel. available from http://www.multipath-tcp.org.
[7] C. Paasch, S. Ferlin, O. Alay, and O. Bonaventure. Experimental evaluation of Multipath TCP schedulers. In *ACM SIGCOMM Capacity Sharing Workshop*, 2014.
[8] C. Raiciu, C. Paasch, S. Barre, A. Ford, M. Honda, F. Duchene, O. Bonaventure, and M. Handley. How hard can it be? Designing and implementing a deployable Multipath TCP. In *NSDI*, 2012.