Poster: Redesigning MPTCP for Edge Clouds

Nitinder Mohan
University of Helsinki, Finland

Tanya Shreedhar
IIIT Delhi

Aleksandr Zavodovski
University of Helsinki, Finland

Otto Waltari
University of Helsinki, Finland

Jussi Kangasharju
University of Helsinki, Finland

Sanjit K. Kaul
IIIT Delhi

ABSTRACT

Edge clouds are an attractive platform to support latency-sensitive applications by providing computations on servers deployed close to end-users. These servers aim to employ MPTCP to leverage multiple connections including wireless over a public network. In this paper, we show that the default MPTCP design does not adequately support reliability in these environments, which makes it unfit for use in edge clouds. We propose RAMPTCP, an extension to MPTCP which focuses on adding reliability over network paths.

ACM Reference Format:
Nitinder Mohan, Tanya Shreedhar, Aleksandr Zavodovski, Otto Waltari, Jussi Kangasharju, and Sanjit K. Kaul. 2018. Poster: Redesigning MPTCP for Edge Clouds. In *The 24th Annual International Conference on Mobile Computing and Networking (MobiCom '18), October 29-November 2, 2018, New Delhi, India.* ACM, New York, NY, USA, 3 pages. https://doi.org/10.1145/3241539.3267738

1 INTRODUCTION

Cloud computing has emerged as a de-facto paradigm for hosting a variety of compute-dependent applications and services. Even though the cloud is composed of controlled and managed inter-network of distributed data centers, the end-client connection to the cloud is usually via a congested public network. Time-critical and data-dependent applications such as Internet-of-Things (IoT), augmented/virtual reality etc. are heavily impacted due to delays and re-transmissions caused by the local network and are unable to fully benefit from the low compute time offered by cloud.

Recently, researchers have proposed *edge clouds* [2, 6] which aim to decouple network delay from computation time by deploying *compute servers* close to the users. The edge servers are equipped with a myriad of network interfaces (such as Ethernet, WiFi, cellular, etc.) to support all possible connectivity use-cases over any existing (public or managed) network [1]. For ease of interoperability with the cloud, edge cloud aim to support existing cloud-based applications such as live VM/container migration, etc. [9]. However, such applications were designed considering data-center networks and are incapable of overcoming frequent network delays, congestion and failures; which are highly probable in edge cloud environment.

To provide robustness in edge networks, researchers have considered utilizing Multipath TCP (MPTCP) on edge servers [3]. MPTCP is a standardized extension to TCP that allows devices to simultaneously leverage multiple available networks (cellular, WiFi) to form parallel TCP connections between end-hosts. Due to several benefits offered by MPTCP such as increased robustness, bandwidth aggregation and seamless handovers; researchers have highly advocated its usage in mobile environments [7]. However unlike typical mobile to cloud network interaction, where only the client is equipped with congestion prone wireless NIC, an edge-to-edge or mobile-to-edge connection is dominated by wireless networks at both ends of the communication.

In this paper, we show via real network experiments that default MPTCP prioritizes path utilization over reliability leading to significantly high packet delays and retransmissions, thereby rendering it unsuitable for edge networks. We propose *Receiver-Assisted* MPTCP (RAMPTCP) as a reliability-first extension to MPTCP. RAMPTCP focuses on mitigating recovery time from probable packet losses and delays by incorporating both sender and receiver-side last-hop MAC characteristics. Being a work-in-progress, we discuss several design aspects of RAMPTCP and show from our preliminary evaluation the possible gains in network reliability it achieves over default MPTCP.

2 MPTCP AT THE EDGE

2.1 MPTCP in a nutshell

Unlike regular TCP, MPTCP builds connections over two hosts and not two interfaces. It adds an abstraction control layer atop TCP which is responsible for actively managing packets injected into each connection. The protocol is composed of three modular control blocks: *path manager*,

MobiCom '18, October 29-November 2, 2018, New Delhi, India

ACM ISBN 978-1-4503-5903-0/18/10.
https://doi.org/10.1145/3241539.3267738

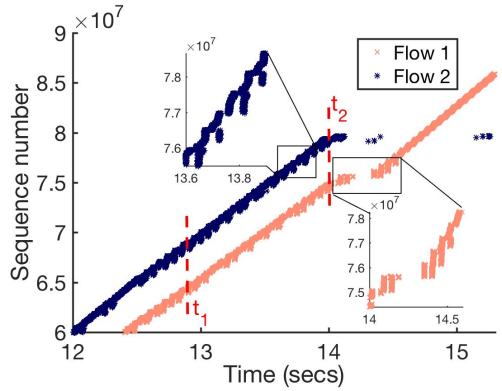

Figure 1: Default MPTCP operation over two subflows when flow 2 experiences packet errors from t_1. MPTCP keeps scheduling new packets to flow 2 resulting in heavy retransmissions over both subflows until RTT of flow 1 exceeds that of flow 2 at t_2.

scheduler and *congestion control*. Lets consider a VM migration scenario between two edge servers both equipped with two wireless NICs on both ends. Assuming that both edge servers support MPTCP, the *path manager* is responsible for establishing and managing subflows[1] between the hosts over available NICs. Once TCP subflows between the machines are established, the *scheduler* injects application packets into each connection while prioritizing network path with lower *sender-perceived* TCP smoothed round-trip time (SRTT). Further, unknowing to underlying TCP, MPTCP adds its end-to-end congestion control and packet sequencing to support out-of-order delivery of packets over heterogeneous paths.

2.2 Sender vs the Network

MPTCP architecture is designed to be sender-centric wherein the sender host is primarily responsible for network decisions (packet injection, rate control) based on sender-perceived path information (read SRTT). MPTCP's reliance on sender-focused decisions disables it to timely account for packet delays and losses on the *receiver-side* last mile which, as motivated in Section 1, is highly prevalent in today's network.

To show the impact of MPTCP's behavior, we set up a real edge-to-edge communication experiment discussed in Section 2.1. Both servers are equipped with two 802.11g NICs and support latest default MPTCP v0.94. Figure 1 shows per-flow sequence number of packets over time. We inject path errors on B2 associated with flow 2 starting t_1 with 0.2 probability, which triggers increased TCP retransmissions on affected path due to retransmission timeouts (RTO). However, MPTCP keeps injecting new packets on affected flow 2 even after encountering errors, until the delay is reflected in its

[1]We use the term TCP connection and subflow interchangeably

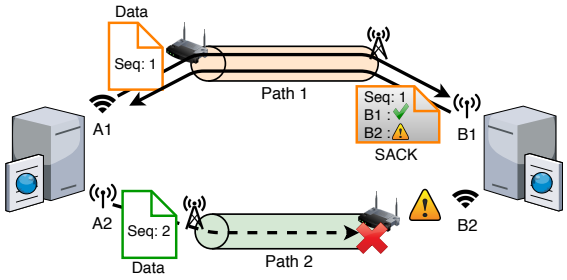

Figure 2: RAMPTCP in action.

TCP SRTT (at t_2). After t_2 all lost packets are retransmitted on unaffected flow 1. We found that MPTCP injected 43% new packets on flow 2, of which 74% had to be re-transmitted. Such heavy re-transmissions lead to large receive buffers, re-ordering delays and bandwidth wastage.

3 RECEIVER-ASSISTED MPTCP

We focus on redesigning MPTCP to incorporate network *reliability* and reduce time delays due to retransmissions and packet reordering in congestion-prone networks. Receiver-Assisted MPTCP (*RAMPTCP*) is an extension to MPTCP which enables the sender to consider receiver-side last-mile characteristics in its control decisions actively. We draw RAMPTCP design motivation from TCP congestion control where timely knowledge of receiver's congestion window is pivotal for sender rate control decisions. In experiment above, we observe that the sender continually receives ACKs on either subflow despite packet losses on one flow. RAMPTCP facilitates the receiver to report several local last-hop metrics by leveraging reverse-path ACKs to sender. As RAMPTCP is a work-in-progress, in the following sections we discuss several "possible" design aspects to incorporate receiver characteristics effectively.

3.1 System design

Figure 2 portrays the experiment scenario discussed in Section 2.2 where receiver-side last hop (B2) is error-prone due to channel interference. Therefore, all packets sent on path 2 is dropped at B2's access point. RAMPTCP enables the receiver to report back last-hop metrics such as path loss, channel utilization, SNR, queue length, channel interference, etc. to the sender via modified MPTCP ACK. This information effectively captures any packet delays/losses at receiver last-hop and is *readily available* in default Linux kernel. The receiver embeds the selected characteristics along with their unique MPTCP *path ID* in all ACKs (for any path) to ensure its delivery. The sender leverages received path information and augments it with local path characteristics and SRTT to estimate packet delays at sub-network level. Essentially,

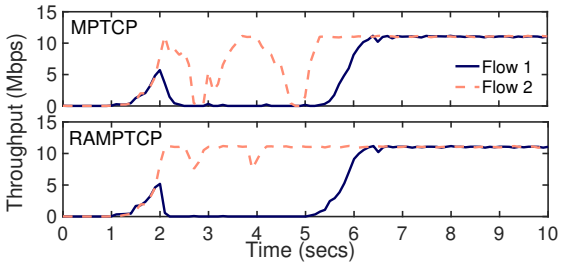

Figure 3: Per-flow throughput comparison between MPTCP and RAMPTCP for topology in Figure ?? when interference on B1 affects flow 1 from 2-5s

RAMPTCP breaks down overall SRTT in

$$\text{SRTT} = \text{T}_{sender} + \text{T}_{core} + \text{T}_{receiver},$$

i.e., time delay at sender last-hop, core network and receiver last-hop respectively. To estimate T_{sender} and $\text{T}_{receiver}$, RAMPTCP can utilize similar principles as sender-side MAC-aware methodologies [5, 8] and is thereby able to estimate time delay in core network. By computing per-segment delay for each subflow, RAMPTCP can enforce several control decisions for affected flow quickly, a few of which are listed below.

Scheduler: limit packet injections, out-of-order injection for heterogeneous delays, packet duplication.
Congestion control: lower/increase TCP send rate.
Path manager: boycott subflow usage, utilize subflow using the best combination of last-hop links.

3.2 Preliminary Results

To evaluate the effectiveness of RAMPTCP, we implement the topology shown in Figure 2 in the ns3 network simulator. Both source and destination are equipped with two 802.11g NICs associated to their respective APs on orthogonal channels. The source is aware of receiver channel utilization, SNR and path loss %. Figure 3 shows per-flow throughput achieved on both subflows when flow 1 encounters high packet loss due to WiFi interference on the B1 interface from 2-5s. In its preliminary stage, we design RAMPTCP to simply mark a subflow active/inactive based on *channel interference* and *signal strength threshold*. While MPTCP relies on RTO's to invoke congestion control, RAMPTCP avoids possible packet losses by marking flow 1 "inactive" based on reverse-path information from the destination for the affected time. Overall, RAMPTCP achieves ≈ 58% reduction in retransmissions and ≈ 19% increase in application goodput.

4 DISCUSSION AND FUTURE WORK

We design RAMPTCP to be backward compatible with MPTCP and therefore support existing machines and network middleboxes. In this section, we discuss several open problems in future work on RAMPTCP and possible solutions.

I. Reporting receiver path characteristics: As shown in our evaluation, considering receiver last-hop characteristics in network decisions enables RAMPTCP to adapt to network instabilities quickly. In our current approach, we relay the required characteristics through ACKs. Incorporating this approach in Linux kernel would require extending MPTCP Data Sequence Signal (DSS) packet with an additional four octet block [4]. Even though the approach easily enables RAMPTCP to function, it also limits its compatibility with Performance Enhancing Proxies (PEP) and middlebox inclusive networks which block any TCP extensions. In future, we plan to consider the limited yet existing "Options" field in TCP packet. One can employ an effective encoding algorithm at sender and receiver to compress required information into restricted space. Further, we also plan to analyze the effects of increased ACK sizes on end-to-end MPTCP communication.

II. Effectiveness of RAMPTCP control decisions In Section 3.1, we enumerated several control decisions that can be undertaken by RAMPTCP to ensure reliability upon detecting path delays. As such decisions can directly affect system performance, they must be bounded by certain minimum requirements. First, RAMPTCP must avoid switching states too frequently due to intermittent path delays. Second, as mandated by MPTCP, RAMPTCP must always perform better than single path TCP, and lastly, RAMPTCP must ensure required Quality-of-Service by avoiding complete loss of service on highly heterogeneous/error prone paths.

In future work, we plan to develop a utility function which incorporates all last-hop metrics to denote "path-health" which predicts the probability of successful packet delivery.

REFERENCES

[1] CISCO. CISCO edge server data spec sheet. "www.cisco.com/c/en/us/products/collateral/routers/829-industrial-router/datasheet-c78-734981.pdf", 2018.
[2] Ahmed et al. A survey on mobile edge computing. In *Intelligent Systems and Control (ISCO)*, 2016.
[3] Chaufournier et al. Fast transparent virtual machine migration in distributed edge clouds. In *Symposium on Edge Computing*, 2017.
[4] Handley et al. Tcp extensions for multipath operation with multiple addresses. 2013.
[5] Lim et al. Cross-layer path management in multi-path transport protocol for mobile devices. In *IEEE INFOCOM 2014*, 2014.
[6] Mohan et al. Edge-fog cloud: A distributed cloud for internet of things computations. In *Cloudification of the Internet of Things*, 2016.
[7] Paasch et al. Exploring mobile/wifi handover with multipath tcp. In *ACM SIGCOMM workshop on Cellular networks*, 2012.
[8] Shreedhar et al. Qaware: A cross-layer approach to mptcp scheduling. In *IFIP Networking 2018*, 2018.
[9] Silvestro and Mohan et al. Mute: Multi-tier edge networks. EuroSys workshop on CrossCloud. ACM, 2018.