This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/JSAC.2020.3000365, IEEE Journal on Selected Areas in Communications

Peekaboo: Learning-based Multipath Scheduling for Dynamic Heterogeneous Environments

Hongjia Wu, Özgü Alay, Anna Brunstrom, Simone Ferlin, Giuseppe Caso

***Abstract*—Multipath transport protocols utilize multiple network paths (e.g., WiFi and cellular) to achieve improved performance and reliability, compared with their single-path counterparts. The scheduler of a multipath transport protocol determines how to distribute the data packets onto different paths. However, state-of-the-art multipath schedulers face the challenge when dealing with heterogeneous paths with dynamic path characteristics (i.e., packet loss, fluctuation of delay). In this paper, we propose Peekaboo, a novel learning-based multipath scheduler that is aware of the dynamic characteristics of the heterogeneous paths. Peekaboo is able to learn scheduling decisions to adopt over time based on the current path characteristics and dynamicity levels - from both deterministic and stochastic perspectives. We implement Peekaboo in Multipath QUIC (MPQUIC) and compare it with state-of-the-art multipath schedulers for a wide range of dynamic heterogeneous environments, upon both emulated and real networks. Our results show that Peekaboo outperforms the other schedulers by up to 31.2% in emulated networks and up to 36.3% in real network scenarios.**

***Index Terms*—Multipath scheduling, Dynamic heterogeneous paths, Multi-armed bandit, Stochastic adjustment.**

I. Introduction

Multipath transport protocols allow the concurrent use of multiple radio access technologies, such as WiFi and cellular, for fast and reliable data exchange. In a generic scenario, the sender distributes application data onto different available radio interfaces, i.e., over different *paths*; the receiver reassembles and reorders the data from different paths, sending them transparently towards the application. By doing so, multipath transport protocols aim at improving both transmission capacity and reliability compared with their single-path counterparts.

In multipath transport protocols, the *scheduler* determines *how* to distribute data onto the available paths, as represented in Figure 1. The data packets from the application reside in the send buffer, and the scheduler assigns each packet to a different interface based on a particular scheduling policy. One of the significant challenges for designing a multipath scheduler is to deploy a policy, which deals with the *heterogeneous* characteristics of paths, e.g., the combination of WiFi and LTE networks. When the paths are heterogeneous, especially in terms of delay and loss, sent packets will arrive to the destination out of order, leading to head of line (HoL) blocking, ultimately reducing the performance. Recently, Blocking Estimation-based MPTCP Scheduler (BLEST) [1] and the Earliest Completion First (ECF) [2] schedulers have been proposed to target heterogeneous paths. They address the HoL blocking problem by introducing a *wait* action, where the scheduler can decide to wait for better conditions to come for the transmission of a packet. While these schedulers work well when the channel characteristics are rather stable, they were not designed for dynamically changing characteristics. However, the *path dynamicity level*, which is defined as the degree of path delay variation and packet loss, can vary significantly over time in real networks [3], [4]. For example, the delay characteristics of a WiFi network in a public library can vary significantly over time due to different numbers of users creating different congestion levels, hence different dynamicity levels during the day, as illustrated in Figure 1. Such changes of dynamicity is particularly common in the current WiFi and LTE networks, and expected to be even more prominent in the upcoming 5G technology. Considering that the paths can have dynamically changing channel characteristics, we tested the state-of-the-art multipath schedulers over different dynamicity scenarios, and we find that none of the state-of-the-art multipath schedulers show consistently better performance (§II).

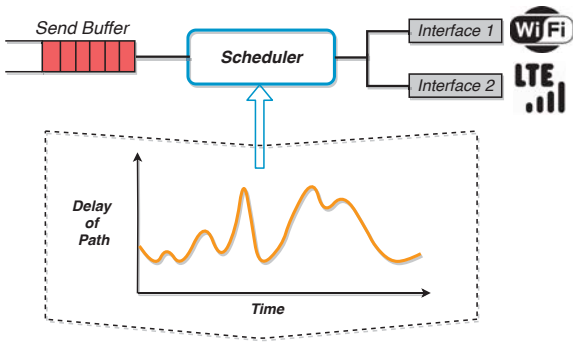


Figure 1. In a multipath scenario, the characteristics of the paths may be heterogeneous and they often vary over time, especially in wireless networks (e.g. WiFi or LTE). A multipath scheduler should take the dynamicity of each path into account while determining the scheduling policy.

In this paper, motivated by the above observations, we aim to address the following research problem: *How to design a multipath scheduler that learns and adapts to heterogeneous paths with dynamically varying channel conditions?* To address this problem, we propose Peekaboo, a novel learning-based multipath scheduler that keeps monitoring the impact caused by the current dynamicity level of each path and selects the most suitable scheduling strategy accordingly. In order to do so, Peekaboo first selects a *deterministic* strategy to deal with different levels of dynamicity, i.e., choose a particular

Manuscript received October 8, 2019; revised February 15, 2020; accepted March 16, 2020.

H.Wu is with SimulaMet and OsloMet, Oslo, Norway. E-mail: *hongjia@simula.no*. Ö.Alay is with University of Oslo and SimulaMet, Oslo, Norway. E-mail: *ozgua@ifi.uio.no*. A.Brunstrom is with Karlstad University, Karlstad, Sweden. E-mail: *anna.brunstrom@kau.se*. S.Ferlin is with Ericsson AB, Stockholm, Sweden. E-mail: *simone.ferlin@ericsson.com*. G.Caso is with SimulaMet, Oslo, Norway. E-mail: *giuseppe@simula.no*.

path or wait for better conditions for the transmission of a packet. This is achieved using an online, adaptive learning mechanism. It then applies a *stochastic* adjustment strategy on top of the deterministic decision, in order to better counteract the dynamicity experienced over the available paths.

From the methodology perspective, we treat multipath scheduling as a decision-making problem. Decision-making problems can be solved in different ways. Control theory is a well known approach in this domain. However, most efficient control theory tools fall into the domain of Linear Time Invariant (LTI) systems, because of the mathematical solvability [5]. Therefore, unless a specific module from the networking system can be safely assumed as LTI, it is not trivial to apply control theory into the multipath scheduling problem. Reinforcement Learning (RL) is a popular method to solve decision-making problems. A RL agent aims to learn the policy that maximizes the cumulative reward obtained by repeatedly interacting with its surrounding environment [6]. For example, the congestion control problem can also be treated as a decision-making problem and RL algorithms are shown to provide great benefits [7], [8]. Previous work has also considered using Deep RL (DRL) methods for decision-making problems, such as DeepRM [9] and Pensieve [10]. However, DRL increases the algorithm's reaction time dramatically [11] when the ongoing traffic deviates from the training traffic, because of the required large amount of training traffic and the slow convergence time.

In this paper, due to its proven benefits, we choose RL to derive the *deterministic* scheduling strategy. We deliberately adopt a lightweight approach in order to adaptively react to varying path dynamicity levels in a timely manner. The lightweight approach decreases the algorithm's reaction time but sacrifices the accuracy [12]. To mitigate the trade-off, we propose and incorporate a *stochastic* adjustment strategy to improve the algorithm accuracy. The contributions of our work can be summarized as follows:

- We first formulate the multipath scheduling problem and propose a lightweight and deployable online learning solution to this problem. More specifically, given a dynamicity level, a deterministic strategy is derived by using a RL algorithm applied in contextual Multi-Armed Bandit (MAB) scenarios [13] (§III-A).
- We further formulate a stochastic adjustment strategy and propose a lightweight derivation and selection of such an adjustment strategy, analyzing its impact on the overall scheduling policy, as a function of the dynamicity levels experienced on the paths (§III-B).
- We combine the online learning solution with the stochastic adjustment strategy in the final design of Peekaboo. Peekaboo runs without any initial input and assumption of the path dynamicity. To test its performance, we implement Peekaboo in the Multipath QUIC (MPQUIC) framework [14]. The source code of Peekaboo and the scripts to produce results are open source[1].
- Finally, we present a multifaceted evaluation in dynamic heterogeneous environments. First, we compare the performance of Peekaboo with state-of-the-art schedulers across a wide range of emulation scenarios, by changing bandwidth, delay, and loss rate variation (§IV). We then carry out experiments in real networks (§V). We show that Peekaboo consistently outperforms the state-of-the-art schedulers performance for heterogeneous paths both in emulations and in real network experiments.

The rest of this paper is organized as follows. In §II, we present the background and motivation of our work. We then detail the design of Peekaboo in §III and evaluate its performance via emulations in §IV and real world experiments in §V. We discuss our work in §VI and conclude in §VII.

II. Background and Motivation

In this section, we first review the multipath transport protocols (§II-A) and state-of-the-art multipath schedulers (§II-B). We then reveal the shortcomings of these multipath schedulers when dealing with different path dynamicity levels and present the two key insights that motivate Peekaboo's design (§II-C).

A. Multipath Transport Protocols

Today's endhosts are often equipped with more than a single network interface, and users expect to be able to use and switch between them seamlessly. To leverage multiple interfaces, several multipath extensions to transport protocols such as TCP and UDP have been proposed. Concurrent Multipath Transfer for SCTP (CMT-SCTP [15]), Multipath TCP (MPTCP [16]), and Multipath QUIC (MPQUIC [14]) are to date the most popular implementations. In multipath transport, three main building blocks are often the objects of research: congestion control [17], [18], [19], [20], [21], [22], path (or connection) management [23], [24], [25], [26] and the scheduler [27], [2], [28], [29], [1]. In this paper, we propose Peekaboo, a multipath scheduling algorithm that is designed and implemented to be easily embedded in the aforementioned protocols. Here, we have chosen MPQUIC as a basis to evaluate Peekaboo for two main reasons: (i) QUIC's development is currently attracting attention in different communities; (ii) QUIC's implementation is in user space, simplifying extension and adoption.

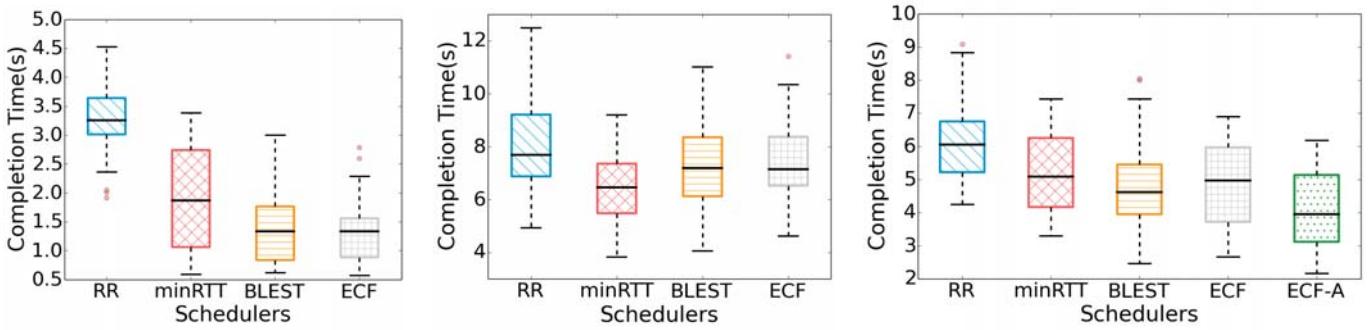
B. The State of the Art Schedulers

A common baseline for multipath scheduler evaluation is the Round-Robin (RR) scheduler algorithm, which cyclically transmits packets over each path, as long as there is space in the Congestion Window (CWND). However, since RR does not use any characteristics of the paths in the scheduling decision, it leads to poor performance when the underlying network paths are heterogeneous [27]. To address this issue, the minRTT scheduler, the default algorithm in MPTCP [16] and MPQUIC [14], prioritizes transmission on the path with the lowest estimated Round Trip Time (RTT) [30], after checking for space in the CWND. However, minRTT is in general unable to estimate how many packets should be sent on each path; simply utilizing all CWNDs and all available paths [28].

As a further improvement, the Blocking Estimation-based MPTCP Scheduler (BLEST) algorithm adds such estimation,

[1]https://mosaic-simulamet.com/peekaboo

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/JSAC.2020.3000365, IEEE Journal on Selected Areas in Communications



(a) Insight 1: Low dynamicity (b) Insight 1: High dynamicity (c) Insight 2: Medium dynamicity

Figure 2. Illustration of design insights based on 2MB file download completion times, under different dynamicity levels (i.e., loss rates and RTT variations).

introducing a *wait* mechanism [1]: If the network path with the highest RTT is the only one available, BLEST can decide to wait for the lowest RTT path to become available again, if it predicts that sending on the highest RTT path may block the receiver. The Earliest Completion First (ECF) algorithm also applies a similar *wait* mechanism, but the estimation is based on decreasing the idle time of the lowest RTT path [2]. Both BLEST and ECF have their merits; however, they fail to be generically applicable. Recently, RELES [31] has been proposed using a neural adaptive multipath scheduler based on deep reinforcement learning. RELES applies Deep Q-Network (DQN) to teach the multipath scheduler from online data.

To solve the receiver buffer problem [32] in multipath transport, [30] proposes the Penalisation and Retransmission (PR) mechanism. [28] derives a rule-of-thumb for buffer size for MPTCP based on the RTT difference of the paths; and [29] explores an implementation that could send packets out-of-order, both with the ambition to overcome head-of-line (HoL) blocking. These algorithms served as a reference in [1], where BLEST addresses some simplifications in both cases, improving the completion times of file downloads and reducing latency for web transfers and video streaming.

Addressing specific use cases and applications, [33], [34], [25] apply an adaptive packet duplication mechanism to guarantee robustness, which proves to be effective when extra data usage and battery consumption are not limiting factors. [35] proposes the Slide Together Multipath Scheduler (STMS) algorithm to reduce out-of-order packet arrivals and, thus, the receiver buffer problem. However, the authors do not focus on application performance. In [36] the authors introduce the concept of probability in the scheduler combined with Forward Error Correction (FEC). More recently, [37] proposes a loss-aware scheduler targeting networks with more than 20% loss rates. Finally, [32] proposes the Short Transfer Time First (STTF) scheduler. However, it specifically targets low-latency for short transfers, and it explores interactions with TCP specific aspects such as TCP Small Queues (TSQ).

Considering the use case tackled in this paper, i.e., multipath transmission with heterogeneous networks, we consider RR, minRTT, BLEST, and ECF as the closest relevant state-of-the-art schedulers to motivate Peekaboo's design and evaluation in the rest of the paper. Although RELES is the closest work to Peekaboo, the authors do not provide open source code of their implementation. Further, the paper does not offer sufficient information for us to reproduce the work. It is, therefore, not possible to compare Peekaboo against it.

C. Key Design Insights of Peekaboo

Insight 1: *A multipath scheduler should employ an adaptive scheme that takes into account both the current network path characteristics and their dynamicity levels.*

In order to illustrate how path dynamicity affects the performance of state-of-the-art multipath schedulers, we conduct a preliminary set of experiments and analyze the results. In our experiments, we choose bandwidth and RTT delay characteristics of the paths to quantify their heterogeneity. While *Path 1* mimics a slower link with 2Mbps bandwidth and RTT of 200ms, *Path 2* mimics a faster link with 50Mbps bandwidth and RTT of 40ms. Our experimental setup is further detailed in §IV-A. We further define three levels of *path dynamicity*, and specify the dynamicity range to show how paths change. For example, while *Low* dynamicity refers to very stable channel conditions with 0% random packet loss and 0% delay variation, *Medium* dynamicity refers to low variations in the channel with $1,5\%$ random packet loss and 8% delay variation and finally *High* dynamicity refers to significant variations in the channel with 3% random packet loss and 16% delay variation. The loss rate in *High* dynamicity is chosen based on the real world measurement study of a nation wide campaign [3]. However, the same study, or any other study we have found in the literature, does not provide path RTT variation. We thus choose the parameters based on our real world measurement as reported in §V. In order to evaluate if any multipath scheduler can consistently offer superior performance compared to other algorithms in different scenarios, we fix the dynamicity level of *Path 1* to *Medium* while switching dynamicity levels for *Path 2* between *Low* and *High*. Given a dynamicity scenario, the download time of a 2 MB data chunk is evaluated for each multipath scheduler. To make the experiments statistically relevant, each scheduler repeats such downloads 100 times.

As shown in Figure 2(a)-2(b), the dynamicity level of *Path 2* significantly impacts the performance of the schedulers, and the best performing scheduler is different for *Low* and *High* dynamicity levels. When the dynamicity level is low, BLEST and ECF easily outperform RR and minRTT, since

they favor the use of the faster path (i.e., they favor packets to be transmitted on *Path 2*). However, when the dynamicity level is high, minRTT becomes the best choice, since BLEST and ECF overuse the lossy and delay varying *Path 2*.

Our evaluation of the state-of-the art schedulers shows that, given different dynamicity levels, none of the existing solutions consistently outperforms the others. Considering that the conditions of a network path (i.e., the dynamicity of the path) often vary over time, especially in wireless scenarios, we need a multipath scheduler that can adapt to the changing network conditions. To illustrate the potential gains when embedding a simple adaptive scheme in a scheduling policy, we leverage a simple concept referred to as Best State of the Art (BSoA). Given a path dynamicity level, BSoA would select the scheduler that provides the best performance, among several state-of-the-art schedulers. For example, in Figure 2(a)-2(b), BSoA will choose ECF and minRTT in the low vs. high dynamicity scenarios, respectively.

While the idea behind BSoA is simple, we note that it is not practical. However, also motivated by the potential gains obtained with BSoA, we design Peekaboo, a practical lightweight adaptive packet scheduler that learns the scheduling decision to adopt over time based on the current path dynamicity levels. We describe the *online learning approach* adopted by Peekaboo in details in §III-A.

Insight 2: *Without an accurate estimate of the characteristics of the available paths, due to variability and incomplete information, a multipath scheduler should employ a stochastic adjustment strategy.*

Due to the complex nature of a network, it is virtually impossible for a multipath scheduler to always have complete and accurate information about current and imminent path characteristics. Related to *Insight 1*, even though an adaptive model is used, inaccurate information on path characteristics may lead to a suboptimal scheduling decision. To overcome such a limitation, we propose a *stochastic adjustment strategy* to address possible suboptimal deterministic scheduler decisions. Based on the current conditions, while the original multipath scheduling algorithms deterministically take an action, within the stochastic adjustment strategy, we insert a two-state Markov chain [38] and maintain a *nonzero* probability of selecting the action that is discarded during the previous deterministic decision step.

To illustrate the benefits of a stochastic adjustment strategy, we conduct preliminary experiments. Given a state-of-the-art scheduler as a starting point, we assign a probability value to each possible deterministic decision. Our goal is to find the probability values that maximize the scheduler performance (i.e. to minimize the median completion time of a file download). This is a multivariate optimization problem, and can be solved via several optimization tools. To this aim, we use the Particle Swarm Optimization (PSO) [39] approach to derive the probability values to be associated to the deterministic decisions of the scheduler. Without loss of generality, we built the stochastic adjustment on top of the ECF scheduler as an example and refer to the stochastically adjusted ECF as ECF-A.

In the experiment, we choose both *Path 1* and *Path 2* characteristics with a medium dynamicity level. This results in similar performance between ECF, BLEST, and minRTT, hence, it allows us to better appreciate the contribution of the stochastic adjustment applied to ECF. We illustrate the impact of the stochastic adjustment strategy in Figure 2(c), where we show that ECF-A outperforms the best state-of-the-art algorithm by providing a 17% median reduced download time. We observe that the difference between ECF-A and ECF is that when the original scheduler takes the action to wait for the faster path (i.e., *Path 2*), ECF-A has $P_1 = 0.73$ probability to wait but also $1-P_1 = 0.27$ probability to transmit the packet on the slower path (i.e., *Path 1*).

Although we show the effectiveness of the stochastic adjustment strategy, two open questions still remain: First, we need to formulate how the stochastic adjustment strategy can actually improve the performance of a multipath scheduler. Second, we need to find near-optimal probability values without running computationally-expensive optimizations such as PSO. We address these questions in §III-B.

III. Peekaboo Design

Based on the discussions in previous sections, we present here in detail the design and implementation of Peekaboo. We first describe Peekaboo's learning aspects, i.e., the online learning of a deterministic scheduling decision and the stochastic adjustment strategy. Then, we also depict how these aspects are combined and deployed in the Peekaboo algorithm. As shown in Figure 3, the Peekaboo workflow consists of two self-contained stages:

- *Learning stage* (§III-A–§III-B): at this stage, given the dynamicity level of the available paths, the deterministic scheduling policy is derived via online learning, possibly followed by a stochastic adjustment strategy. It is worth noting that this stage is inherently *on the fly*, using the normal data that is transmitted for learning.
- *Deployment stage* (§III-C): at this stage, Peekaboo deploys and realizes the policy derived in the previous stage, and periodically checks if the path characteristics are significantly changing (i.e., whether the change in dynamicity is above a pre-defined threshold). If so, Peekaboo reinitializes the Learning stage in order to adapt the policy to the new path characteristics and dynamicity levels.

A. Online Learning

In *Insight 1* of §II-C, we show that for different dynamicity scenarios, none of the existing schedulers consistently outperform the others, and we thus need a multipath scheduler that adapts well to the different dynamicity scenarios. In Peekaboo, we tackle this problem using an online learning based on contextual MAB theory [13].

In a general contextual MAB problem, a learning *agent* observes a d-dimensional vector of *features*, which represent the surrounding environment (i.e., the *context*). The goal of the agent is to maximize a *reward* function, which indicates how well the agent is adapting to the environment by selecting an *action* from the available set. By leveraging the exploration

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/JSAC.2020.3000365, IEEE Journal on Selected Areas in Communications

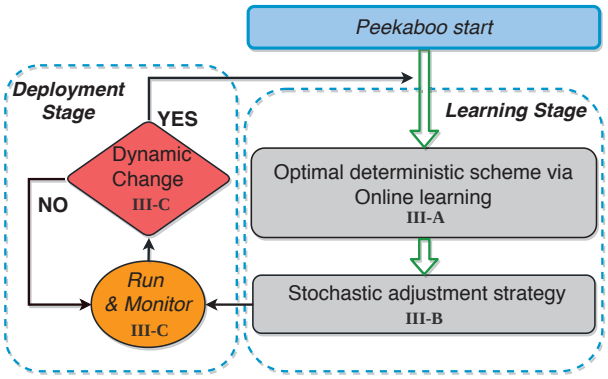


Figure 3. Building blocks of Peekaboo.

vs. exploitation dilemma, the agent moves towards a policy leading to the selection of the best action at each time.

In our multipath scheduling problem, the learning agent is the scheduler, the action is a scheduling decision, and the features are the path characteristics. In the following, we first give more details on actions and features, and then define the reward function to be maximized. Finally, we describe the strategy adopted to select the actions at each time.

Actions. The action set depends on the number of paths and their availability. For simplicity, in this work we assume two paths (i.e., WiFi and cellular), which is a common configuration in off-the-shelf devices. However, the use of Peekaboo can be straightforwardly extended to the case of more than two paths. A path is considered available if there is space left in its CWND, and for each available path we have two actions: *transmit* and *wait*. For example, if only one path is available, the action set includes: (i) *transmit* on that path and (ii) *wait* until the other path becomes available again, and if both paths are available, each action corresponds to the selection of a path. For managing the actions, two cases can be considered. In the first case, the scheduler takes over all the actions it can perform and learns how to maximize its performance. In the second case, we rely on minRTT when two paths are available (same as BLEST and ECF), while optimizing the policy when the scheduler has to decide between *waiting* for the low latency path or *transmitting* on the high latency path.

The objective of action selection is to remove the actions that do not contribute positively to the learning performance. We take the action set used in the second case (selected for Peekaboo) as our baseline and adopt a forward selection methodology [40]. In forward selection, we test the addition of each new action on the baseline and select the action that can give the most significant improvement. The iteration continues until we cannot significantly improve performance.

We define and evaluate the following additions and their corresponding actions:

- *Addition 1*: When both paths are available, the scheduler learns how to transmit the packet over each path, instead of transmitting the packet based on the RTT values.
- *Addition 2*: If the faster path is available and the slower path is not available, the scheduler could wait for the slower path and transmit the packet on it.
- *Addition 3*: The sum of *Addition 1* and *Addition 2*, i.e., the learning agent learns how to perform all possible actions.

We performed preliminary experiments[2] across different test cases and we observe that none of the *Additions* brings performance increases to the baseline. In particular, *Addition 2* and *Addition 3* perform worse than the baseline across all test cases. The actions they add do not bring any positive contributions, but actually complicate the learning due to the need of exploring across more actions. This also holds for *Addition 1*, which however behaves similarly to the baseline for low dynamicity levels and gets worse when the dynamicity increases. Based on these observations, we keep the baseline as the action set in Peekaboo.

Features. We adopt four main parameters to create the features used to characterize the paths at the transport layer, these are: CWND, number of Inflight Packets (InP), Send Window (SWND) (the mirror of receive window at the sender) and RTT.

As discussed in the next paragraph, we define the reward as a function of the throughput, and for this reason, we use the selected parameters for three features with a throughput-like unit, dividing the first three parameters by the RTT experienced on the paths. Thus, the feature vector used by Peekaboo includes CWND/RTT, InP/RTT, and SWND/RTT values, for both *Path 1* and *Path 2*. Given a time t, in which a transmit/wait decision has to be taken by Peekaboo, the feature vector is defined as $\boldsymbol{x}_t$, having dimension $d \times 1$, with $d = 6$ in our case. Normalizing the CWND, InP and SWND by the RTT before embedding them in the feature vector provides a comparable scale and also boosts the learning speed [41].

Reward. The reward evaluates the value of the selected action. We define the reward, denoted as R, in the throughput unit, i.e., bytes per millisecond. R is in the form of a *discounted* reward [42]: once an action is taken, R takes into account the throughput obtained over the currently scheduled packet and the following ones, implying that the action impacts the experienced performance on both current and future scheduled packets. R is thus calculated by summing up the instantaneous reward r for each packet observed within a reference time interval. Given a packet, r is calculated as the ratio between its size in bytes and the time elapsed from packet transmission to Acknowledgement (ACK) reception. A time-decreasing discount factor γ is used to weight the instantaneous rewards r so that their contribution to R diminishes over the reference time interval.

Algorithm 1 shows the details of the calculation of R. First of all, the reference time interval is defined as $3T_{\mathsf{ref}}$, where T_{ref} approximates the time in which the currently scheduled packet will be ACK'ed, in case of either *transmit* or *wait* actions being selected. T_{ref} is calculated by considering the values of RTT and RTT variation on both faster and slower paths, as captured by the $\mathrm{RTT_f}$, $\mathrm{RTT_s}$, σ_{f}, and σ_{s} parameters [43], respectively (Line 2). We denote the elapsed time since performing the evaluated action as T_{elap}. As long as T_{elap} is within $3T_{\mathsf{ref}}$ and new packets are being ACK'ed (Line 3), R keeps increasing by the value $r\gamma$ (Lines 4–12), where PS

[2]The preliminary results are provided in Peekaboo's website.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/JSAC.2020.3000365, IEEE Journal on Selected Areas in Communications

Algorithm 1 Discounted Reward R

Input:
1) $\mathsf{c}_1, \mathsf{c}_2, \mathsf{c}_2$: weights for different time slots within $3T_{\mathsf{ref}}$

1: $\gamma = 1, R = 0$
2: $T_{\mathsf{ref}} = max(2(\mathrm{RTT}_{\mathsf{f}} + \sigma_{\mathsf{f}}), \mathrm{RTT}_{\mathsf{s}} + \sigma_{\mathsf{s}})$
3: **while** $T_{\mathsf{elap}} < 3T_{\mathsf{ref}}$ && new ACK **do**
4: $r = \mathsf{PS}/(T_{\mathsf{ACK}})$
5: $R = R + r\gamma$
6: **if** $T_{\mathsf{elap}} \leq T_{\mathsf{ref}}$ **then**
7: $\gamma = \mathsf{c}_1\gamma$
8: **else if** $T_{\mathsf{elap}} \leq 2T_{\mathsf{ref}}$ **then**
9: $\gamma = \mathsf{c}_2\gamma$
10: **else**
11: $\gamma = \mathsf{c}_3\gamma$
12: **end if**
13: **end while**

denotes the packet size in bytes. The value of γ is rearranged by considering the ACKs of the packets, and how they relate to the T_{elap} (Lines 5–11). The constants c_1, c_2, and c_3 are selected so that $\mathsf{c}_3 \leq \mathsf{c}_2 \leq \mathsf{c}_1$. Based on preliminary analysis, we fix these values to 0.9, 0.7, and 0.5, respectively. We note that a different setting results in negligible performance changes, as long as the selected values are appropriately spaced and ensure a decreasing γ.

On the calculation of r, we note that since each path has its own packet sequence space in MPQUIC, a later scheduled packet from the application can be ACK'ed earlier. In this case, the ACK is not released as a new ACK until all the previous packets have been ACK'ed, capturing the in-order-delivery of the packets in the calculation of r and thus R.

Online Learning Strategy. For the online learning strategy, we need an approach that: (i) converges quickly, to quickly react to the current path status, and (ii) is computationally lightweight, so that the operation of the network stack is not negatively affected. To this end, we adopt the LinUCB algorithm [44] to derive how Peekaboo deterministically selects between *transmit* and *wait* actions. LinUCB adopts a ridge regression [45] to evaluate the *expected* reward for a particular action a at time t, given the feature vector $\boldsymbol{x}_t$. It then applies an Upper Confidence Bound (UCB) [46] to the estimation before finally selecting the action that maximizes the estimation at time t. Adopting ridge regression with no UCB, the expected reward at time t for applying action a given $\boldsymbol{x}_t$, is:

$$\mathbb{E}[R_{t,a}|\boldsymbol{x}_t] = \boldsymbol{x}_t^\top \boldsymbol{\theta}_a, \tag{1}$$

where $R_{t,a}$ represents the reward for the considered action at time t. Note that in our specific case $\boldsymbol{x}_t$ is independent over the actions, given the above definitions of actions and features. Moreover, $\boldsymbol{\theta}_a$ is a $d\times 1$, action-specific vector, evaluated as follows [45]:

$$\boldsymbol{\theta}_a = (\mathbf{D}_a^\top \mathbf{D}_a + \mathbf{I}_d)^{-1} \mathbf{D}_a^\top \boldsymbol{c}_a, \tag{2}$$

where $\mathbf{D}_a$ is a $m\times d$ matrix containing the history of the feature vector observed when action a was selected, with m being the number of times this selection occurred, while $\boldsymbol{c}_a$ is a $m\times 1$ vector with the history of the reward obtained upon selection of a.

Algorithm 2 Online Learning via LinUCB

Input:
1) $\mathcal{A}$: action set;
2) $\boldsymbol{x}$: the selected d features
3) α: exploration hyperparameter

for all $a \in \mathcal{A}$ **do**
  $\mathbf{A}_a \leftarrow \mathbf{I}_d$ (d-dimensional identity matrix)
  $\boldsymbol{b}_a \leftarrow \mathbf{O}_{d\times 1}$ (d-dimensional zero vector)
end for
while Learning **do**
  for $t = 1, 2, 3, \ldots$ **do**
    for all $a \in \mathcal{A}$ **do**
      $\boldsymbol{\theta}_a \leftarrow \mathbf{A}_a^{-1}\boldsymbol{b}_a$
      $\mathbb{E}[R_{t,a}|\boldsymbol{x}_t] \leftarrow \boldsymbol{x}_t^\top \boldsymbol{\theta}_a + \alpha\sqrt{\boldsymbol{x}_t^\top \mathbf{A}_a^{-1}\boldsymbol{x}_t}$
    end for
    $a_t \leftarrow \arg\max_a \mathbb{E}[R_{t,a}|\boldsymbol{x}_t]$
    $\mathbf{A}_{a_t} = \mathbf{A}_{a_t} + \boldsymbol{x}_t\boldsymbol{x}_t^\top$
    $\boldsymbol{b}_{a_t} = \boldsymbol{b}_{a_t} + R_{t,a_t}\boldsymbol{x}_t$
  end for
end while
return $\boldsymbol{\Theta} \leftarrow \boldsymbol{\theta}_a \quad \forall a \in \mathcal{A}$

As mentioned before, we want to identify the action maximizing the $\mathbb{E}[R_{t,a}|\boldsymbol{x}_t]$. However, without sufficient exploration of each available action, the amount of data used for estimating $\boldsymbol{\theta}_a$ and thus $\mathbb{E}[R_{t,a}|\boldsymbol{x}_t]$ is not sufficient. In this case, the estimation can be inaccurate, thus leading to suboptimal decisions. For this reason, an UCB is added to Equation (1), so to introduce the exploration degree of each action and improve the estimates of $\boldsymbol{\theta}_a$, leading to the following new formulation of the expected reward [46]:

$$\mathbb{E}[R_{t,a}|\boldsymbol{x}_t] = \boldsymbol{x}_t^\top \boldsymbol{\theta}_a + \alpha\sqrt{\boldsymbol{x}_t^\top \mathbf{A}_a^{-1}\boldsymbol{x}_t}, \tag{3}$$

where, for the sake of simplicity, $\mathbf{A}_a = \mathbf{D}_a^\top \mathbf{D}_a + \mathbf{I}_d$. The α value is the hyperparameter of the LinUCB algorithm, and $\alpha\sqrt{\boldsymbol{x}_t^\top \mathbf{A}_a^{-1}\boldsymbol{x}_t}$ as a whole reflects the importance of exploring new actions for a higher confidence level of the estimations, instead of solely using the action resulting in the highest $\boldsymbol{x}_t^\top \boldsymbol{\theta}_a$. LinUCB finally chooses the action maximizing the expected reward, as follows:

$$a_t = \arg\max_a \mathbb{E}[R_{t,a}|\boldsymbol{x}_t]. \tag{4}$$

The overall approach is presented in Algorithm 2. It should, however, be noted that we adopt the LinUCB version proposed in [44], which transforms the matrix multiplication in Equation (2), so that only a $d\times d$ dimensional matrix is buffered in the computing unit instead of the original $m\times d$ matrix. Moreover, we highlight that the repeated execution of LinUCB during the Learning stage allows derivation of the vectors $\boldsymbol{\theta}_a \ \forall a \in \mathcal{A}$, which are then passed as input to the Deployment stage of the Peekaboo workflow.

α Tuning. α reflects the confidence level of Algorithm 2, and thus plays a key role. If it is too small, the algorithm does not perform sufficient exploration and may behave suboptimally. If

Algorithm 3 α Tuning

1: $\tau = 0.8, \alpha = 0$
2: **for** $n = 1, 2, 3, ...$ **do**
3: cand $= \alpha + n\tau$
4: **if** $\text{score}_{\text{cand}} > \text{score}_{\alpha}$ **then**
5: $\alpha =$ cand
6: $\tau = \tau/2$
7: **else**
8: $break$
9: **end if**
10: **end for**
11: **while** $\tau > 0.05$ **do**
12: $\alpha = \arg\max_{\text{cand} \in (\alpha, \alpha-\tau, \alpha+\tau)} \text{score}_{\text{cand}}$
13: $\tau = \tau/2$
14: **end while**

it is too large, LinUCB might struggle to converge. In order to understand the impact of α, we run Algorithm 2 using different α values over different dynamicity levels. We observe that as dynamicity increases, the optimal α decreases because the existence of high path dynamicity decreases the estimation confidence interval, and the mean optimal α is around $\alpha = 0.8$. Based on these observations, we develop a tuning algorithm for α, as described in Algorithm 3, and run it on top of LinUCB during the Learning stage.

The α Tuning algorithm examines the LinUCB performance while adopting different values of α. These are latter selected with a decreasing step size, and for each of them, a score is recorded. In Algorithm 3, score_{α} denotes the optimal performance obtained during the searching process, while $\text{score}_{\text{cand}}$ records the performance of the current α candidate, denoted as cand. The score represents, in short, how well the current policy learned from Algorithm 2 fits with the current path configuration, while adopting the current cand. Its formal definition equates to the definition of $\hat{q}$, as shown in Algorithm 4 presented later, and for this reason we refer to §III-B and §III-C for a more refined and complete description.

Algorithm 3 initiates a broad scale search (Lines 2-10), starting from $\alpha = 0$ and taking $\alpha = 0.8$ as a centroid point, based on our empirical observations. Once the score is obtained for $\alpha = 0$, we add a step size of 0.8 and repeat the comparison of the score by scaling down the step size and until we cannot gain any performance increase. Then the algorithm performs a small scale search (Lines 11-14), exploring any two candidates around the best candidate with half of the current step size to determine the next best candidate. This can lead us to the gradient descent [47] trajectory. The iteration stops when the step size is smaller than 0.05. Note that Algorithm 3 formally searches over a broad range of possible α values. However, to speed up the tuning process, the algorithm can directly perform a small scale search around $\alpha = 0.8$.

B. Stochastic Adjustment Strategy

We show in *Insight 2* of §II-C how a stochastic adjustment strategy can improve the performance of the deterministic scheduler. In this subsection, we formulate the stochastic adjustment problem and propose an approach to derive a near-optimal probability value to use in the adjustment without running a computationally-expensive optimization algorithm such as PSO.

Strategy Formulation. The results in §II-C show that it is not always wise to wait for a faster path when a deterministic scheduler makes the wait decision. In order to explain this result, we define two actions, $A1$ (wait for the faster path) and $A2$ (transmit on the slower path), and assume from now until the end of this subsection that the deterministic scheduler makes the decision to wait, i.e., always performing $A1$. On the other hand, a scheduler with the stochastic adjustment would instead perform $A1$ and $A2$ with certain probabilities. We define $C1$ and $C2$ as two configurations of the path. We assume that waiting for the faster path ($A1$) is desirable while being in $C1$; therefore, we define the payoff of $A1$ to be higher than or equal to the payoff of $A2$. Oppositely, $A2$ is desirable under $C2$ configuration, and thus we define the payoff of $A2$ to be higher than the payoff of $A1$. The reward defined in §III-A is the payoff metric, but in order to simplify the evaluation of the expected payoff of a stochastic strategy, we consider a no-regret payoff matrix [48], as shown in Table I. With no-regret payoff, when the stochastic strategy selects the action matching well with the current configurations, the payoff will be zero, indicating that there is no deviation from the optimal action in terms of payoff. Otherwise, the payoff will be negative. Given this payoff matrix, we also define the probabilities of performing $A1$ and $A2$ as p and $1-p$. In the following, p is also referred to as the *stochastic factor*. Then, we also define q as the probability of being in $C1$. It can be noted that q reflects *how the deterministic scheduler and its decision* (that is to wait in our example) *fits with the current configuration*. Similarly, the probability of being in $C2$ is $1-q$.

Table I
NO-REGRET PAYOFF MATRIX.

		Configurations $C1$	Configurations $C2$
Actions	$A1$	0	$-\mathsf{e}$
	$A2$	$-\mathsf{f}$	0

Given a *single* decision, the expected payoff of the stochastic strategy can be written as follows:

$$\begin{aligned}\mathbb{E}[\mathsf{payoff}] &= (p)(q)(0) + (p)(1-q)(-\mathsf{e}) + (1-p)(q)(-\mathsf{f}) + (1-p)(1-q)(0) \\ &= (p)(1-q)(-\mathsf{e}) + (1-p)(q)(-\mathsf{f}).\end{aligned} \tag{5}$$

Equation (5) shows that if, for example, $q = 1$, the stochastic strategy has to adopt $p = 1$ to maximize the expected payoff (so that the $A1$-$C1$ match is always verified), and conversely, if $q = 0$, then $p = 0$ is optimal (so that the $A2$-$C2$ match is always verified). Moreover, when $0 < q < 1$, the scheduler still maximizes its payoff (but to a value < 0), by adopting $p = 0$ (if $q < 0.5$) and $p = 1$ (if $q > 0.5$). At $q = 0.5$, the expected payoff of the stochastic strategy is the same for all values of p. For example, if this stochastic adjustment is applied to ECF, then $p = 1$ represents a pure ECF scheduling mechanism, while $p = 0$ indicates that wait is always avoided, leading ECF to be reduced to minRTT.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/JSAC.2020.3000365, IEEE Journal on Selected Areas in Communications

However, the above description only holds for a single decision, e.g., over a single packet, since, in the next ones, the values of e, f, and q are different. Taking into account that the applications consist of many packets, we focus on long-run payoffs, with both the current value and overall distribution of q unknown. Under this assumption, besides the deterministic strategies (i.e., $p = 0$, $p = 1$), another possible approach is thus to adopt a value of p in between 0 and 1 so that the observed performance is *indifferent* to q. Such an indifferent point, denoted as p_{indiff} , can be derived in two steps. First, we derive the expected payoffs separately when $C1$ and $C2$ are verified:

$$\begin{cases} \mathbb{E}[\mathsf{payoff}|C1] = (p)(0) + (1-p)(-\mathsf{f}) = (1-p)(-\mathsf{f}) \\ \mathbb{E}[\mathsf{payoff}|C2] = (p)(-\mathsf{e}) + (1-p)(0) = (p)(-\mathsf{e}) \end{cases} \tag{6}$$

Then, the scheduler is indifferent to the environment behaviour when $\mathbb{E}[\mathsf{payoff}|C1] = \mathbb{E}[\mathsf{payoff}|C2]$, that is, when $p = \dfrac{\mathsf{f}}{\mathsf{e}+\mathsf{f}}$. By adopting $p_{\text{indiff}} = \dfrac{\mathsf{f}}{\mathsf{e}+\mathsf{f}}$, the stochastic strategy also max-min [49] its expected payoff. This can be seen if we transform Equation (5) as follows:

$$\mathbb{E}[\mathsf{payoff}] = \frac{-\mathsf{ef}}{\mathsf{e}+\mathsf{f}} + (\mathsf{e}+\mathsf{f}) \times \left(p - \frac{\mathsf{f}}{\mathsf{e}+\mathsf{f}}\right) \times \left(q - \frac{\mathsf{e}}{\mathsf{e}+\mathsf{f}}\right). \tag{7}$$

From Equation (7), we see that the expected payoff is indifferent to q when $p = p_{\text{indiff}} = \dfrac{\mathsf{f}}{\mathsf{e}+\mathsf{f}}$, and this latter provides the max-min expected payoff.

From the above discussion, we derive that three main strategies are available for a scheduler embedding a stochastic adjustment in one of its actions, that is, $p = 0$, $p = 1$ or $p = p_{\text{indiff}}$. However, no single strategy is absolutely dominant over the other and which strategy is adopted depends on q. In Figure 6 of §IV, we further elaborate and illustrate the impact of the stochastic factor on the scheduler performance.

It is worth noting that we present the stochastic strategy in the particular case when the deterministic action to be adjusted is to *wait* (for the faster path to be available), but an equivalent adjustment can be applied in parallel to the *transmit* (on the slower path) action. However, by running several experiments on different path settings and configurations, we observe that the adjustment always occurs on a single action at a time, even though it is concurrently activated on both actions. This can be explained by considering that the deterministic algorithm might overestimate one action with respect to the other, and thus only a single adjustment occurs to balance the overestimation. However, in order to be general enough and adapt to all possible scenarios and path configurations, Peekaboo takes into account the possibility of stochastically adjusting both actions, as shown in Algorithm 4, as this may be needed in particular corner scenarios.

Estimation of p_{indiff} and q. The indifferent point is not a-priori known to the stochastic strategy, and we thus need to estimate an approximated value. For this reason, we propose the following procedure. The indifferent point includes the differences in the rewards observed by performing actions $A1$ and $A2$, i.e., the values e and f. These values cannot be obtained in parallel since the two actions cannot be performed at the same time. To solve this, we then look to the history of the obtained rewards and corresponding feature vectors for the action not currently selected. We then extract the configuration having the *closest* feature vector to the current one, in terms of Mahalanobis distance [50], and consider the reward that was observed. This allows us to evaluate the difference with the current reward and, by averaging over several iterations, we derive an approximation of the indifferent point, that is, $p_{\text{indiff}} = \dfrac{\mathsf{f}}{\mathsf{e}+\mathsf{f}}$, where we use sampled average values of e and f. We show in §IV that this procedure leads to good estimations of the indifferent points under several path dynamicity levels and configurations. The same procedure can be applied to find a heuristic estimate of q. Recall that q represents the probability that the action selected by the deterministic policy can offer a higher reward than the other action. By looking at the history, as done for the estimation of p_{indiff}, we can also calculate a ratio of the number of times the deterministic decision offered a higher reward, with respect to the total number of decisions. In the following, we denote $\hat{q}$ as the heuristic estimate of q. Since the stochastic adjustment to adopt depends on q, we use its estimate $\hat{q}$ to decide which adjustment to use, that is, $p = 0, 1$, or p_{indiff}, as analyzed in the previous subsection. This aspect is further discussed in the next section, when the overall Peekaboo workflow is described.

C. Overall Peekaboo Algorithm

In §III-A and §III-B, we show how to capitalize on the two key design insights presented in §II. We now combine these two mechanisms and present Peekaboo as a whole as shown in Algorithm 4.

In the **Learning Stage** of Peekaboo, we first run the $p = 0$ deterministic strategy for both adjustment actions, i.e., minRTT and always wait for the fast path (i.e., only use the fast path and wait if it is not available), and obtain h for both strategies, representing the history of rewards and features for both cases. Note that we use the subscripts tx and wt throughout Algorithm 4 to identify these two cases. We then run the deterministic online learning scheduler based on Algorithm 2 in §III-A, along with the α tuning process shown in Algorithm 3. This latter takes the histories as input to evaluate the scores (i.e., the value of $\hat{q}$ for both examined actions). This step finally returns $\boldsymbol{\Theta}$, representing the learning outcome, and final scores $\hat{q}$ for each evaluated action. Then, we use $\boldsymbol{\Theta}$ and the histories to estimate p_{indiff} for both actions, based on the approach described in §III-B. These values are used to choose the stochastic adjustment strategy to adopt on top of $\boldsymbol{\Theta}$ (Lines 5–18). If $\hat{q}$ is close to 1 (i.e., $\hat{q}$ is larger than the boundary BD_1), we behave almost deterministically by letting $p = 0.9$. Similarly, if $\hat{q}$ is close to 0, we choose $p = 0.1$. The reason to keep some randomicity is that we still want to have a stochastic behavior to be able to estimate $\hat{q}$ periodically in the deployment stage. If $\hat{q}$ does fall in between the boundaries BD_1 and BD_0, we use $p = p_{\text{indiff}}$. We set BD_1 and BD_0 to 70% and 30%, respectively, based on our extensive empirical observations.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/JSAC.2020.3000365, IEEE Journal on Selected Areas in Communications

Algorithm 4 Peekaboo

Input:

1) $\mathrm{BD}_0, \mathrm{BD}_1$: dominant boundaries of the deterministic strategy, i.e., $(p_{\mathrm{wt,tx}} = 0, 1)$;

2) $\hat{q}_{\mathsf{th}}$: threshold to detect the change of $\hat{q}$;

Learning:

$h_{p_{wt}=0} = Run\&Record\,()$; \\ i.e., minRTT, probability to wait is 0

$h_{p_{tx}=0} = Run\&Record\,()$; \\ i.e., only use the faster path and always wait for it, probability to transmit is 0

$\boldsymbol{\Theta}, \hat{q}_{\mathrm{wt}}, \hat{q}_{\mathrm{tx}} = Run\&Learn\,(h_{p_{\mathrm{wt}}=0}, h_{p_{\mathrm{tx}}=0})$

$p_{\mathrm{indif,wt}}, p_{\mathrm{indif,tx}} = Est\,(\boldsymbol{\Theta},\ h_{p_{\mathrm{wt}}=0},\ h_{p_{\mathrm{tx}}=0})$;

if $\hat{q}_{\mathrm{wt}} > \mathrm{BD}_1$ **then**

  $p_{\mathrm{wt}} = 0.9$;

else if $\hat{q}_{\mathrm{wt}} < \mathrm{BD}_0$ **then**

  $p_{\mathrm{wt}} = 0.1$;

else

  $p_{\mathrm{wt}} = p_{\mathrm{indif,wt}}$;

end if

if $\hat{q}_{\mathrm{tx}} > \mathrm{BD}_1$ **then**

  $p_{\mathrm{tx}} = 0.9$;

else if $\hat{q}_{\mathrm{tx}} < \mathrm{BD}_0$ **then**

  $p_{\mathrm{tx}} = 0.1$;

else

  $p_{\mathrm{tx}} = p_{\mathrm{indif,tx}}$;

end if

goto$(Deployment)$;

Deployment:

while True **do**

  $\hat{q}_{\mathsf{deploy,wt}}, \hat{q}_{\mathsf{deploy,tx}} = Deploy\&Est(\boldsymbol{\Theta}, p_{\mathrm{wt}}, p_{\mathrm{tx}})$;

  if $|\hat{q}_{\mathsf{deploy,wt}} - \hat{q}_{\mathrm{wt}}| > \hat{q}_{\mathsf{th}}$ **then**

    goto$(Learning)$;

  end if

  if $|\hat{q}_{\mathsf{deploy,tx}} - \hat{q}_{\mathrm{tx}}| > \hat{q}_{\mathsf{th}}$ **then**

    goto$(Learning)$;

  end if

end while

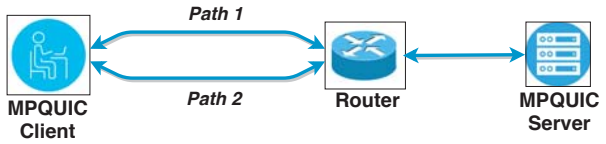


Figure 4. Multipath topology used in the experiments.

After the learning stage, we have $\boldsymbol{\Theta}$, the needed input for the deterministic decision; we have p, the value of the stochastic factor; and we have $\hat{q}$, the value which indicates how well $\boldsymbol{\Theta}$ fits in the current dynamicity. We then go to the **Deployment Stage** where we deploy the scheduler based on $\boldsymbol{\Theta}$ and p. Meanwhile, we estimate $\hat{q}_{\mathsf{deploy}}$ along with deployment. If, after a certain period of time, we find the difference of $\hat{q}_{\mathsf{deploy}}$ and $\hat{q}$ is larger than the threshold we define, $\hat{q}_{\mathsf{th}}$, then we assume the current dynamicity does not match with the $\boldsymbol{\Theta}$ and p obtained during learning. We will then re-enter into the learning stage to estimate new values for $\boldsymbol{\Theta}$ and p. Based on preliminary experiments, we found $\hat{q}_{\mathsf{th}}$=17.5% to be a suitable threshold.

Table II
EMULATION PARAMETERS. *PATH 2* IS CONFIGURED AT THREE DYNAMICITY LEVELS (E.G. LOW, MEDIUM AND HIGH), AND *PATH 1* IS ALWAYS KEPT FIXED AT A MEDIUM DYNAMICITY LEVEL.

Parameter	Dynamicity Level			
	Path 1		*Path 2*	
	Medium	*Low*	*Medium*	*High*
Bandwidth [Mbps]	2	50	50	50
One-way Delay [ms]	100	20	20	20
RTT Variation [%]	8	0	8	16
Random Loss [%]	1.5	0	1.5	3

IV. Evaluation in the Emulation

In this section, we evaluate the performance of Peekaboo in an emulated environment. This lets us assess the performance of the proposed algorithm with a wide range of network configurations. We present the experimental setup in §IV-A. The impact of online learning and the stochastic strategy are evaluated in §IV-B and §IV-C, respectively. Peekaboo as a whole is evaluated in §IV-D for two different applications, and the adaptivity of Peekaboo is discussed in §IV-E. Finally, Peekaboo is also examined in §IV-F within the whole design space composed of different path characteristics, from the heterogeneous to the homogeneous case.

A. Experimental Setup

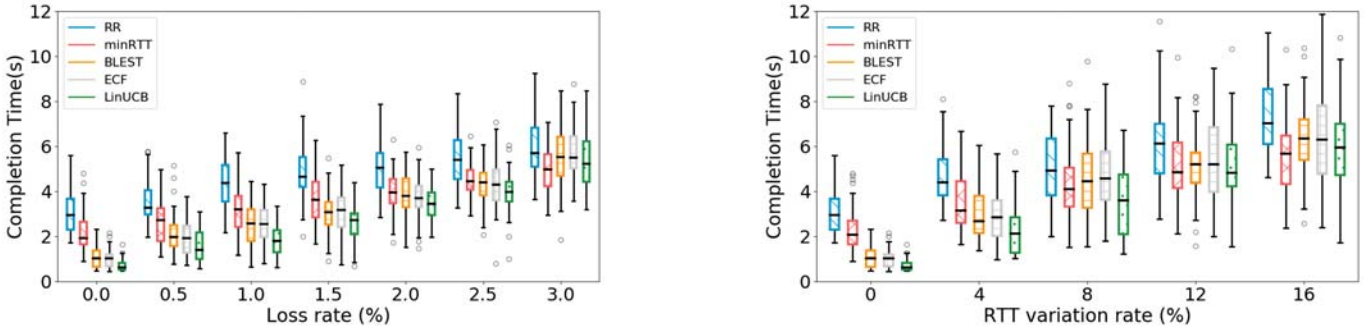
We use `Mininet` [51] as the emulation environment, since it can emulate a real network stack using Linux containers avoiding simplifications of simulation models. We employ the topology shown in Figure 4. This topology has two network interfaces to the client and one to the server over two partially disjoint paths, i.e., *Path 1* and *Path 2*. Each path is characterized by its bandwidth, One-Way Delay (OWD), RTT variation, and random loss. We use the Linux traffic control tool `NetEM` [52] to control these parameters and configure the characteristics of the paths. We generate multiple configurations to evaluate Peekaboo's performance with respect to other schedulers. To ensure statistical significant results, for each path configuration, we run 120 repetitions for each scheduler.

We consider two different applications in our evaluation: file download and real time streaming. For file downloads, we evaluate the download completion times. Inspired by [36], we also consider real-time streaming, where the application regularly sends equally spaced messages. The application has a deadline for the messages, and we evaluate the percentage of messages that arrive before the deadline and the delay of these late messages that arrive after the deadline. For example, to mimic real-time streaming with a bitrate of 2 Mbps, we regularly send messages composed of 8 QUIC packets of 1000 bytes. Each message is spaced by 33 milliseconds. We also assume the deadline is equal to the spacing, which is 33 milliseconds [36].

B. The Impact of Online Learning

We first investigate the performance gains of using a solely deterministic online learning scheduler over the state-of-the-art schedulers. Our goal here is to illustrate how and at which dynamicity levels the sole use of online learning can

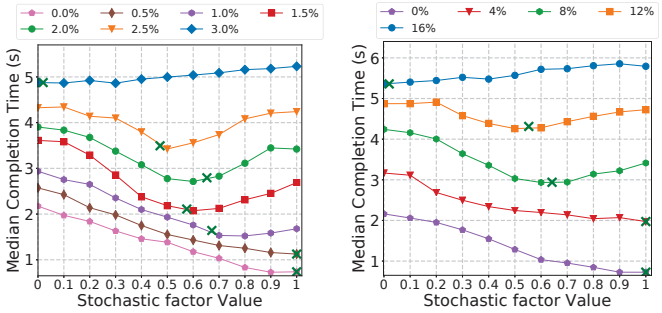
This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/JSAC.2020.3000365, IEEE Journal on Selected Areas in Communications



(a) Performance under different loss rates

(b) Performance under different RTT variations

Figure 5. Deterministic online learning based approach evaluation. The download completion times of a 2MB file under different Loss and RTT variations.



(a) LinUCB with loss

(b) LinUCB with RTT variation

Figure 6. Stochastic adjustment strategy evaluation. The download completion times of a 2MB file under different loss and RTT variations for various stochastic factors. The cross symbol on each curve represents our estimated optimal stochastic factor for each path dynamicity level.

provide performance gains. We use the same setting shown in Table II, except that we change the loss rate and RTT variation parameters of *Path 2* for different levels of dynamicity. We illustrate the results for file downloads in Figure 5 where in Figure 5(a), we keep the RTT variation rate as 0.0% and vary the random loss rate from 0.0% to 3.0% with a granularity of 0.5% while in Figure 5(b), we keep the random loss rate as 0.0% and vary the RTT variation from 0.0% to 16% with a granularity of 4%.

In Figure 5(a), we observe that the deterministic online learning, denoted as LinUCB, outperforms the state-of-the-art schedulers when the random loss rate is between 0.0% and 1.5%. For example, for the 1.0% random loss case, we achieve up to 28% shorter download completion time. However, when the random loss rate is in the range of 2.0% to 3.0%, the performance benefits are not as significant, with minRTT outperforming LinUCB when the random loss rate is 3.0%. We observe a similar trend in Figure 5(b). When the RTT variation is between 0.0% and 8.0%, online learning outperforms the state-of-the-art. However, when the RTT variation rate is in the range of 12.0% to 16.0%, the performance benefit is not significant. Similarly, minRTT outperforms online learning when the RTT variation rate is 16.0%.

In summary, we observe that the performance benefit from sole deterministic online learning depends on the path dynamicity. While deterministic online learning alone provides performance gains when the path dynamicity is low, for high dynamicity, we choose to incorporate the stochastic adjustment strategy.

C. The Impact of Stochastic Strategy

We now explore the performance of the stochastic adjustment strategy. We use the same path characteristics from §IV-B, and choose the stochastic factor, p, between 0 and 1 with steps of 0.1, showing its impact on the download completion time.

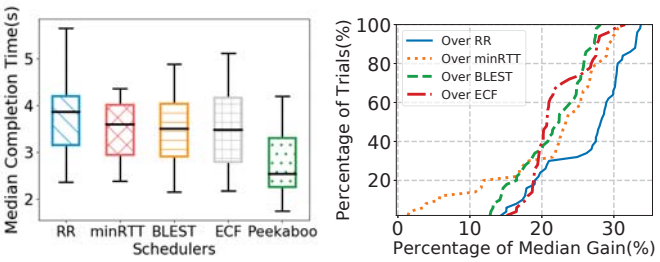
We illustrate the impact of the stochastic factor in Figure 6 for LinUCB. Each curve shown in Figure 6 operates under one specific path dynamicity level from §IV-B. The cross symbol on each curve represents our estimated optimal stochastic factor for each path dynamicity level. We observe similar results for BLEST and ECF, and due to space limitations, those results are not presented. We observe that the performance as a function of the stochastic factor highly depends on the path dynamicity. For example, when the dynamicity level is low (i.e., relatively low loss and RTT variation rate), we observe that the stochastic factor of 1 offers the best performance. However, as we increase the dynamicity levels, the stochastic factor that offers the best performance varies, and the value generally decreases. When the dynamicity level is the highest, the stochastic factor of 0 offers the best performance. This trend also matches our analysis in §III-B. In summary, depending on how the deterministic scheduler fits into the dynamicity level of the path (i.e., q in Equation (7)), one of the strategies among $p = 0$, $p = 1$ and $p = p_{\text{indiff}}$ outperforms the others. Examining our estimated optimal value of p on each graph (i.e., the cross symbol on each curve), we see that although the estimation error exists, it does not result in any significant performance penalty.

In summary, we observe that the proposed stochastic adjustment strategy provides performance gains, especially for *medium* to *high* dynamicity. Therefore, online learning and the stochastic adjustment strategies complement each other.

D. Performance of Peekaboo as a Whole

Next, we evaluate Peekaboo in a wider range of configurations to show the effectiveness of Peekaboo under different settings. To achieve this, we adopt the experimental design principle, WSP (Wooton, Sergent, Phan-Tan-Luu) algorithm, to distribute the design parameters equally across the experiments, based on uniform random sampling of the input design parameters [53]. We use bandwidth, OWD, RTT variation, and random loss rate as parameters. As presented in Table III, we

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/JSAC.2020.3000365, IEEE Journal on Selected Areas in Communications



(a) Median completion time (b) Peekaboo's percentage gain

Figure 7. File download performance of Peekaboo and other schedulers.

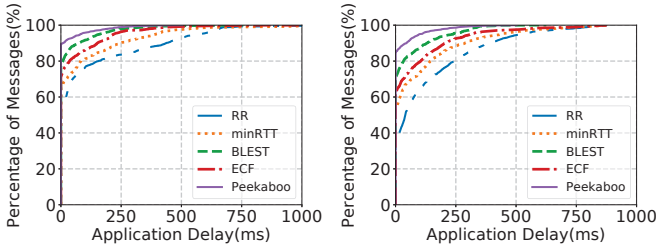
Table III
THE RANGE OF PARAMETERS REFLECTING PATH HETEROGENEITY IN TERMS OF BANDWIDTH, DELAY AND DIFFERENT DYNAMICITY LEVELS.

Path	Bandwidth	OWD	RTT variation	Loss
1	2-10Mbps	80-100ms	0-16%	0-3%
2	40-50Mbps	20-30ms	0-16%	0-3%

select the range of these parameters in such a way that the two paths are heterogeneous in terms of bandwidth and delay, and furthermore the paths can have different dynamicity levels. Compared to Table II, Table III extends the coverage range of bandwidth and OWD for both *Path 1* and *Path 2*, allowing us to evaluate the performance of Peekaboo over different heterogeneity levels. We use WSP to generate 100 distinct path configurations based on Table III and we evaluate two applications, i.e., file download and real-time streaming.

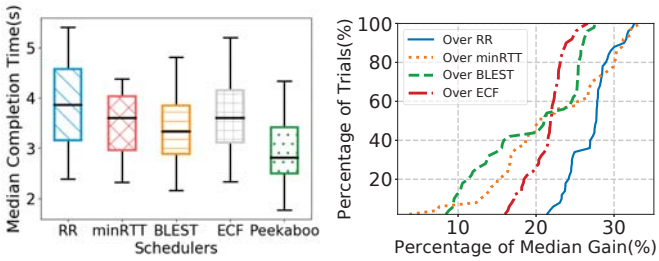
File Download. We evaluate Peekaboo's performance compared to other schedulers in Figure 7. We first illustrate the median file download completion times for different schedulers in Figure 7(a). Here, for each path configuration, we first take the median values of completion time over the repetitions, and then plot them over different path settings. We observe that Peekaboo outperforms all the state-of-the-art schedulers. Next, we compute the Empirical Cumulative Distribution Function (ECDF) of the performance improvement percentage of Peekaboo compared to the other schedulers across different path configurations and illustrate the results in Figure 7(b). We observe that compared to ECF, Peekaboo can achieve up to 30% shorter download completion times for certain path configurations. And we observe that those certain path configurations are mostly of low dynamicity levels, where the deterministic learning part of Peekaboo can provide the most benefit. Further, the gains are over 20% for more than 50% of the path configurations.

Real Time Streaming. We evaluate Peekaboo's performance for real-time streaming with two different bitrates: 1 Mbps and 2 Mbps. For each path configuration and bitrate, we stream 20 MB of data for each scheduler and the results are shown in Figure 8. We plot the ECDF of the application delay for all the messages. As mentioned earlier, if the message arrives before the deadline, the application delay is 0 ms. We observe that Peekaboo in total achieves 10.6% and 16.1% higher percentages for the messages that arrive before their deadline, compared with the best state-of-the-art scheduler at 1 Mbps and 2 Mbps, respectively. This is because Peekaboo



(a) 1 Mbps streaming bitrate (b) 2 Mbps streaming bitrate

Figure 8. Performance of the real-time streaming application: The ECDF of the application delay for the messages.



(a) Median completion time (b) Peekaboo's percentage gain

Figure 9. Adaptivity test in heterogeneous network.

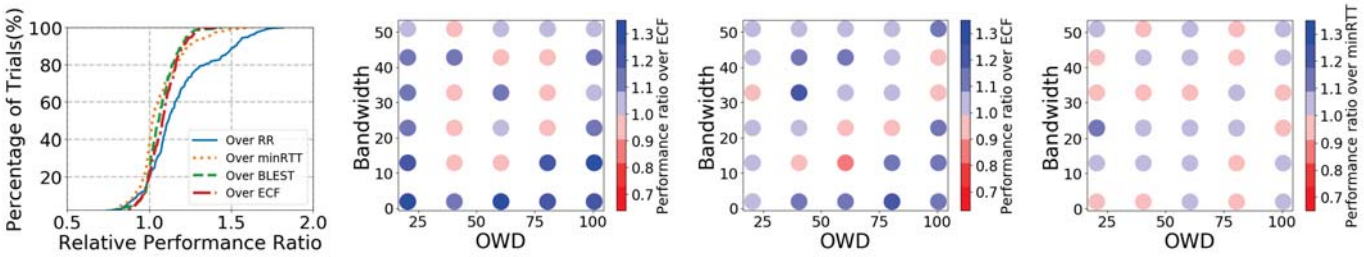
utilizes the discounted reward to quantify the impact between different packets. It penalizes the case where the current packet may delay the receipt of future packets. Besides that, we also observe that for the packets that missed the deadline, Peekaboo provides shorter delays.

E. Peekaboo's Adaptivity

We examine now Peekaboo's adaptivity under changing bandwidth, OWD, and dynamicity levels. The goal of the adaptivity test is to show that Peekaboo is capable of learning the changing environment and adapt accordingly. To stress test Peekaboo, we examine a network where bandwidth, OWD, RTT variation and random loss rates change every 60 seconds for 3000 seconds. Changes are random within the values shown in Table III.

In Figure 9, we illustrate the median file download completion times of different schedulers and the percentage gain Peekaboo provides over the other schedulers. As depicted in Figure 9(b), we achieve up to 26.4% shorter median completion times than ECF. We can see that the performance gains over the state-of-the-art schedulers is slightly less as compared to Figure 7. This is because, with rapid changes in network characteristics, Peekaboo needs to adapt regularly resulting in a learning overhead. We allocate 4 MB learning overhead in each learning round, which is equivalent to 4-8 seconds in the emulated networks depending on the path characteristics, and 0.5-1.4 seconds in real networks as shown in §V. We exploit this emulated scenario to mimic the case of a relatively static user connected to a network where the network characteristics change over time. However, for a mobile user [54], the changing speed of the networks can surpass the learning speed achieved through the online learning. This is further discussed in §VI.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/JSAC.2020.3000365, IEEE Journal on Selected Areas in Communications



(a) Performance ratio over the whole design space (b) Medium Low over ECF (c) Medium Medium over ECF (d) Medium High over minRTT

Figure 10. Design space exploration of Peekaboo. The bandwidth and OWD of *Path 2* are fixed to 50 Mbps and 20 ms and varied between 2 Mbps-50 Mbps and 20 ms-100 ms in *Path 1*. For subfigures (b), (c) and (d), the bluer the point is, the higher is the gain of Peekaboo over ECF.

F. Peekaboo's Design Space Exploration

Peekaboo is designed for heterogeneous scenarios. In a heterogeneous network, the action to wait for the other path takes effect due to the difference in path characteristics. The difference, however, does not hold in a homogeneous scenario. Thus, it is necessary to show Peekaboo's performance across the whole design space composed of different path characteristics, from the heterogeneous to the homogeneous case. We perform the following design space exploration to verify it.

We fix the bandwidth and OWD of *Path 2* as 50 Mbps and 20 ms, respectively. We assume both *Path 2* and *Path 1* can have 3 different dynamicityclasses: Low, medium and high, as also shown in Table II. In total, *Path 2* and *Path 1* give 9 different combinations of path dynamicity. For each dynamicity, we vary the bandwidth and OWD of *Path 1* in the range of 2 Mbps-50 Mbps and 20 ms-100 ms, respectively. For each path configuration, we perform file downloads and collect the median completion times shown in Figure 10.

Figure 10(a) illustrates the ECDF of the performance ratio of Peekaboo compared to the other schedulers across different path configurations. As depicted in Figure 10(a), Peekaboo performs better than BLEST, ECF, and RR in around 80% of the whole design space, and better than minRTT in around 60% of the whole design space with more than 50% performance improvement for some path configurations. As an example, this can be visualized in the cases shown in Figures 10(b), 10(c) and 10(d), where the dynamicity of *Path 1* is at medium level and the dynamicity of *Path 2* increases from low to high level. Further, for each dynamicity combination, we show the relative performance ratio of Peekaboo over the state of the art scheduler that shows the overall best performance. In the first two cases, BLEST and ECF show similar performance. We thus compare with ECF. In the third case, we compare with minRTT. As shown in Figures 10(b) and 10(c), Peekaboo outperforms ECF especially in heterogeneous path configurations. Although experimental randomness exist, this can be visualized as the appearance of relative stronger blue points in the lower right corner of the figures. This is because the action set of Peekaboo is designed for the heterogeneous case. Moreover, from Figure 10(b) to Figure 10(c), we see a decrease of stronger blue points. This is because, in the case of Figure 10(b), the dynamicty level is relatively lower and the deterministic learning algorithm can make more contribution. With the dynamicity increase, although the stochastic strategy can aid the deterministic learning algorithm to maintain the performance aggregation, the overall accuracy of the algorithm decreases. Further, in Figure 10(d), Peekaboo converges back to the same policy of minRTT in the heterogeneous case due to the high dynamicity. As a result, they perform similarly (i.e., the points are in the 0.9-1.1 range), which maps to our earlier analysis. This also holds for the other relatively homogeneous points in Figure 10(d). We can also observe the similar performance of ECF and Peekaboo for the points which are in the 0.9-1.1 range in Figures 10(b) and 10(c). This occurs when the path characteristics become homogeneous and both schedulers adopt the minRTT policy. Overall, we see that Peekaboo performs better than or similar to the best state-of-the-art scheduler across the examined design space. With the concept of Peekaboo, it is also possible to further push its performance aggregation over the other schedulers when the heterogeneous degree of the network is not significant, by adding new actions in the learning process. We come back to this in §VI when discussing limitations and future work.

V. Evaluation in the Wild

So far, we examined Peekaboo's performance via emulation experiments. Now, we evaluate Peekaboo in real networks with file downloads and real-time streaming applications.

A. Experimental Setup

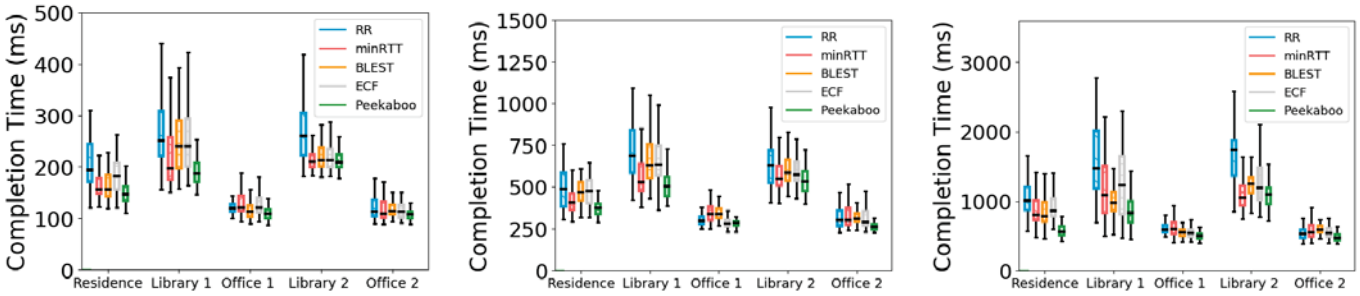
We deploy an MPQUIC server in a European capital, and the MPQUIC client is locally on a laptop in the same city. The client communicates with the server over the Internet, using both a WiFi access point and an LTE network.

During the experiments, we keep the LTE network provider fixed and test over both a private and a public WiFi (Eduroam [55]). We consider five different scenarios. The first one is a residence, where we use private WiFi. The second and third scenarios are at one of the universities in the city, where we test with Eduroam both in the library and office. The fourth and fifth scenarios are at another university in the city, where we also test with Eduroam both in the library and office. We report the dynamicity level of the WiFi and LTE paths of each scenario in Table IV.

B. Experimental Results

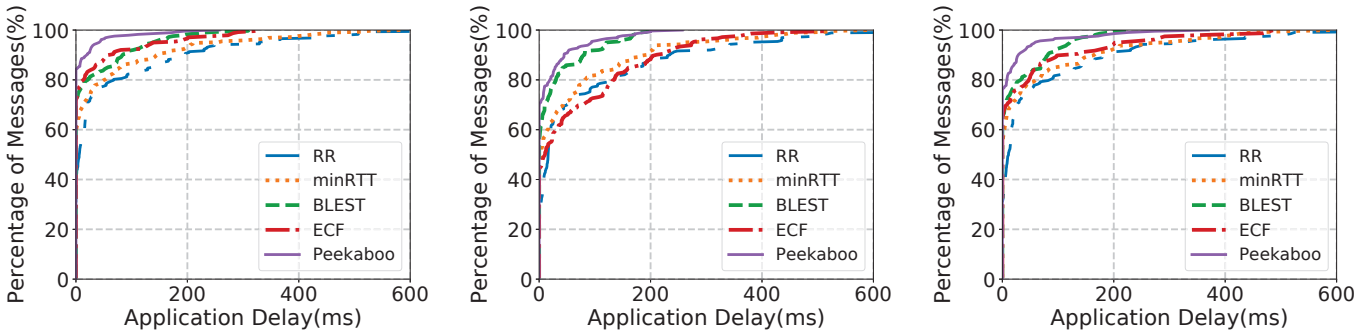
File Download. We first investigate the distribution of file download completion times using Peekaboo and state-of-the-art schedulers. Here, we download files of different sizes, i.e.,

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/JSAC.2020.3000365, IEEE Journal on Selected Areas in Communications



(a) 512 KB File (b) 2 MB File (c) 4 MB File

Figure 11. File download completion times for different file sizes at different scenarios.



(a) Residence (b) Library1 (c) Office1

Figure 12. Real-time streaming at 2 Mbps at different scenarios.

Table IV
MEDIAN RTT VARIATION AND LOSS RATE OF REAL WORLD SCENARIOS.

Scenario	WiFi RTT variation	WiFi loss	LTE RTT variation	LTE loss
Residence	8.8%	0.02%	9.7%	0.03%
Library 1	15.8%	0.41%	11.2%	0.19%
Office 1	9.8%	0.21%	10.6%	0.02%
Library 2	14.7%	0.40%	11.7%	0.08%
Office 2	10.3%	0.19%	12.3%	0.14%

256KB, 512KB, 2MB, and 4MB, repeating the experiment 120 times for each scheduler at each file size. We present the results of 512KB, 2MB, and 4MB in Figure 11, and omit the result of 256KB due to similarity with the 512KB case. As shown, we observe that Peekaboo can reduce file download completion times, which can reach up to 36.3% in the 4MB file download case at the Residence scenario, compared to the best state-of-the-art scheduler. However, we also observe the performance of Peekaboo is only slightly better than minRTT at Library 1 and 2 when, e.g., downloading the 2MB file. This is because Peekaboo can adapt to the policy of minRTT in that scenario when detecting the high dynamicity.

Real-Time Streaming. We explore Peekaboo's performance and state-of-the-art schedulers with real-time streaming at 2 Mbps as defined in §IV-A. We observe that Peekaboo can always guarantee more messages arriving before the deadline at different scenarios. This is because Peekaboo essentially sets the function of packet delay as its reward and the real-time streaming scenario has a higher requirement on packet delay compared to the file download. We only report scenarios of Residence, Library 1 and Office 1 in Figure 12, as the results of Library 2 and Office 2 are similar to Library 1 and Office 1. As shown, Peekaboo achieves up to 15.1% higher percentage for messages that arrive before the deadlines compared with the best state-of-the-art scheduler, in Figure 12(b).

Our real world evaluation confirms that Peekaboo outperforms the state-of-the-art schedulers in operational networks with real dynamic path characteristics.

C. Runtime Overhead

Peekaboo incurs negligible runtime overhead, as both learning and scheduling algorithm phases have low complexity. We profiled Peekaboo's CPU and memory utilization comparing it to the other schedulers, without noticing significant penalty in CPU and memory usage.

VI. Limitations

Peekaboo is an online learning multipath scheduler, i.e., both the training and deployment of the training outcome are conducted online. Although Peekaboo adopts the light weight learning approach, this still can be a limiting factor when it comes to the scenarios in which Peekaboo is applicable. For example, in mobility scenarios the speed at which the network changes can surpass the learning speed achieved through online learning. This can be even more difficult when there is not enough traffic for Peekaboo to learn the mobility scenario. To make Peekaboo also perform well in such scenarios, it is possible to incorporate offline learning that learns from sufficient offline traffic to make adaptation proactively. In other words, the offline learning performs the training offline and deploy the training outcome as it is. Offline learning predictions can be both explicit like regression or implicit like the assumed state transition in a Markov Decision Process used for calculating the accumulated reward. The downside of offline learning is that once the offline traffic used for learning deviates from the current online traffic, it will lack a new

learning outcome due to the amount of data required and the convergence time. While each approach has its merits, we plan to take the cooperation of both online learning and offline learning into account as our future work.

Furthermore, we expect there to be a significant number of vertical applications in the upcoming 5G era, where the exploited action and reward in this work (i.e., mainly targeting at heterogeneous networks and existing applications) will need to be extended in order to support a wide range of applications, as well as link characteristics. For example, when interfacing with tactile Internet applications [56], the hard real-time performance should be guaranteed. In that case, robustness is more important than extra data and battery usage. Thus, the action set can have redundancy characteristics, e.g., packet duplication or FEC [57], etc. The reward should also be modified so that it can be mathematically proven to have a delay bound. Moreover, Peekaboo currently does not explore stream prioritization capabilities in MPQUIC, a feature that has been shown to improve performance [58]. We plan to extend the work in this direction and with multi-streaming.

VII. Conclusion

In this paper, we consider the multipath scheduling problem when the paths are heterogeneous with dynamically changing path characteristics. To address this problem, we propose Peekaboo, an adaptive multipath scheduler that leverages an online learning mechanism in combination with a stochastic adjustment strategy to adapt to the dynamic characteristics of the paths. Peekaboo is computationally lightweight and easily deployable. We implement Peekaboo in MPQUIC and compare its performance with state-of-the-art multipath schedulers for a wide range of dynamicity levels, using both emulated networks and real network scenarios. Across the examined scenarios and applications, Peekaboo consistently offers superior or similar performance to the best state of the art schedulers, with the performance improvements of Peekaboo reaching by up to 31.2% in emulated networks and up to 36.3% in real network scenarios.

VIII. Acknowledgment

This work is partially funded by European Union's Horizon 2020 research and innovation programme under grant agreement No. 815178 (5GENESIS).

References

[1] S. Ferlin, Ö. Alay, O. Mehani, and R. Boreli, "BLEST: Blocking estimation-based MPTCP scheduler for heterogeneous networks," in *IFIP Networking*, 2016.

[2] Y.-s. Lim, E. M. Nahum, D. Towsley, and R. J. Gibbens, "ECF: An MPTCP path scheduler to manage heterogeneous paths," in *ACM CoNEXT*, 2017.

[3] "Employing QUIC protocol to optimize uber's app performance," https://eng.uber.com/employing-quic-protocol/, 2019.

[4] S. Deng, R. Netravali, A. Sivaraman, and H. Balakrishnan, "WiFi, LTE, or both?: Measuring multi-homed wireless internet performance," in *ACM IMC*, 2014.

[5] K. Ogata and Y. Yang, *Modern control engineering*. Pearson Upper Saddle River, NJ, 2010, vol. 17.

[6] R. S. Sutton, A. G. Barto *et al.*, *Introduction to reinforcement learning*. MIT press Cambridge, 1998, vol. 135.

[7] M. Dong, Q. Li, D. Zarchy, P. B. Godfrey, and M. Schapira, "PCC: Re-architecting congestion control for consistent high performance," in *12th USENIX Symposium on Networked Systems Design and Implementation (NSDI 15)*, 2015, pp. 395–408.

[8] K. Winstein and H. Balakrishnan, "TCP ex machina: Computer-generated congestion control," in *Proceedings of the ACM SIGCOMM Conference on SIGCOMM*, 2013, pp. 123–134.

[9] H. Mao, M. Alizadeh, I. Menache, and S. Kandula, "Resource management with deep reinforcement learning," in *Proceedings of the 15th ACM Workshop on Hot Topics in Networks*. ACM, 2016, pp. 50–56.

[10] H. Mao, R. Netravali, and M. Alizadeh, "Neural adaptive video streaming with pensieve," in *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, 2017, pp. 197–210.

[11] F. Y. Yan, H. Ayers, C. Zhu, S. Fouladi, J. Hong, K. Zhang, P. Levis, and K. Winstein, "Learning in situ: a randomized experiment in video streaming," 2019.

[12] A. Padmanabha Iyer, L. Erran Li, M. Chowdhury, and I. Stoica, "Mitigating the latency-accuracy trade-off in mobile data analytics systems," in *Proceedings of the 24th Annual International Conference on Mobile Computing and Networking*. ACM, 2018, pp. 513–528.

[13] T. Lu, D. Pál, and M. Pál, "Contextual multi-armed bandits," in *AISTAS*, 2010.

[14] Q. De Coninck and O. Bonaventure, "Multipath QUIC: Design and evaluation," in *ACM CoNEXT*, 2017.

[15] J. R. Iyengar, P. D. Amer, and R. Stewart, "Concurrent multipath transfer using SCTP multihoming over independent end-to-end paths," *IEEE/ACM Transactions on Networking (ToN)*, vol. 14, no. 5, pp. 951–964, 2006.

[16] A. Ford, C. Raiciu, M. Handley, and O. Bonaventure, "TCP extensions for multipath operation with multiple addresses," Tech. Rep., 2013.

[17] C. Raiciu, M. Handley, and D. Wischik, "Coupled congestion control for multipath transport protocols," Internet Requests for Comments (RFC), Tech. Rep. 6356, October 2011, http://www.rfc-editor.org/rfc/rfc6356.txt.

[18] C. Raiciu, D. Wischik, and M. Handley, "Practical congestion control for multipath transport protocols," *University College London, London/United Kingdom, Tech. Rep*, 2009.

[19] R. Khalili, N. Gast, M. Popovic, and J.-Y. Le Boudec, "MPTCP is not pareto-optimal: performance issues and a possible solution," *IEEE/ACM Transactions on Networking (ToN)*, vol. 21, no. 5, pp. 1651–1665, 2013.

[20] C. Raiciu, M. Handley, and D. Wischik, "Coupled congestion control for multipath transport protocols," Tech. Rep., 2011.

[21] A. Walid, J. Hwang, Q. Peng, and S. Low, "BALIA (balanced linked adaptation)–a new MPTCP congestion control algorithm," in *IETF Meeting*, 2014.

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

[23] C. Pluntke, L. Eggert, and N. Kiukkonen, "Saving mobile device energy with multipath TCP," in *ACM MobiArch*, 2011.

[24] Y.-s. Lim, Y.-C. Chen, E. M. Nahum, D. Towsley, and K.-W. Lee, "Cross-layer path management in multi-path transport protocol for mobile devices." in *IEEE INFOCOM*, 2014.

[25] Y. E. Guo, A. Nikravesh, Z. M. Mao, F. Qian, and S. Sen, "Accelerating multipath transport through balanced subflow completion," in *ACM MobiCom*, 2017.

[26] S. Ferlin, T. Dreibholz, and Ö. Alay, "Multi-path transport over heterogeneous wireless networks: Does it really pay off?" in *IEEE GLOBECOM*, 2014.

[27] C. Paasch, S. Ferlin, O. Alay, and O. Bonaventure, "Experimental evaluation of multipath TCP schedulers," in *Proceedings of the ACM SIGCOMM workshop on Capacity sharing workshop*. ACM, 2014, pp. 27–32.

[28] N. Kuhn, E. Lochin, A. Mifdaoui, G. Sarwar, O. Mehani, and R. Boreli, "DAPS: Intelligent delay-aware packet scheduling for multipath transport," in *IEEE ICC*, 2014.

[29] Y. Fan, A. Paul, and E. Nasif, "A scheduler for multipath TCP," in *IEEE ICCCN*, 2013.

[30] 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 *USENIX NSDI*, 2012.

[31] Z. Xu, J. Tang, J. Meng, W. Zhang, Y. Wang, C. H. Liu, and D. Yang, "ReLeS: A neural adaptive multipath scheduler based on deep reinforcement learning," in *IEEE INFOCOM*, 2019.

[32] P. Hurtig, K.-J. Grinnemo, A. Brunstrom, S. Ferlin, O. Alay, and N. Kuhn, "Low-latency scheduling in MPTCP," *IEEE/ACM Transactions on Networking (ToN)*, vol. 27, no. 1, pp. 302–315, 2019.

[33] A. Frommgen, T. Erbshäußer, A. Buchmann, T. Zimmermann, and K. Wehrle, "ReMP TCP: Low latency multipath TCP," in *IEEE ICC*, 2016.

[34] H. Lee, J. Flinn, and B. Tonshal, "Raven: Improving interactive latency for the connected car," in *ACM MobiCom*, 2018.

[35] H. Shi, Y. Cui, X. Wang, Y. Hu, M. Dai, F. Wang, and K. Zheng, "STMS: Improving MPTCP throughput under heterogeneous networks," in *USENIX ATC*, 2018.

[36] F. Michel, Q. De Coninck, and O. Bonaventure, "Adding forward erasure correction to QUIC," in *arXiv preprint arXiv:1809.04822*, 2018.

[37] E. Dong, M. Xu, X. Fu, and Y. Cao, "A loss aware MPTCP scheduler for highly lossy networks," *Computer Networks*, 2019.

[38] D. Gamerman and H. F. Lopes, *Markov chain Monte Carlo: stochastic simulation for Bayesian inference*. Chapman and Hall/CRC, 2006.

[39] J. Kennedy, "Particle swarm optimization," *Encyclopedia of Machine Learning*, pp. 760–766, 2010.

[40] F. G. Blanchet, P. Legendre, and D. Borcard, "Forward selection of explanatory variables," *Ecology*, vol. 89, no. 9, pp. 2623–2632, 2008.

[41] P. Harrington, *Machine Learning in Action*. Manning Publications., 2012.

[42] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.

[43] V. Paxson, M. Allman, J. Chu, and M. Sargent, "Computing TCP's retransmission timer," Tech. Rep., 2011.

[44] L. Li, W. Chu, J. Langford, and R. E. Schapire, "A contextual-bandit approach to personalized news article recommendation," in *ACM WWW*, 2010.

[45] S. Le Cessie and J. C. Van Houwelingen, "Ridge estimators in logistic regression," *Journal of the Royal Statistical Society: Series C (Applied Statistics)*, vol. 41, no. 1, pp. 191–201, 1992.

[46] T. J. Walsh, I. Szita, C. Diuk, and M. L. Littman, "Exploring compact reinforcement-learning representations with linear regression," in *UAI*, 2009.

[47] L. Bottou, "Large-scale machine learning with stochastic gradient descent," in *Springer CompStat*, 2010.

[48] S. Ross, G. Gordon, and D. Bagnell, "A reduction of imitation learning and structured prediction to no-regret online learning," in *AISTAS*, 2011.

[49] H. Aissi, C. Bazgan, and D. Vanderpooten, "Min–max and min–max regret versions of combinatorial optimization problems: A survey," *European journal of operational research*, vol. 197, no. 2, pp. 427–438, 2009.

[50] P. M. Roth, M. Hirzer, M. Köstinger, C. Beleznai, and H. Bischof, "Mahalanobis distance learning for person re-identification," in *Springer Person Re-Identification*, 2014.

[51] N. Handigol, B. Heller, V. Jeyakumar, B. Lantz, and N. McKeown, "Reproducible network experiments using container-based emulation," in *ACM CoNEXT*, 2012.

[52] S. Hemminger *et al.*, "Network emulation with NetEm," in *Linux conf au*, 2005.

[53] C. Paasch, R. Khalili, and O. Bonaventure, "On the benefits of applying experimental design to improve multipath TCP," in *ACM CoNEXT*, 2013.

[54] Q. De Coninck and O. Bonaventure, "Multipathtester: Comparing MPTCP and MPQUIC in mobile environments," in *TMA MNM Workshop*, 2019.

[55] "Eduroam," https://www.eduroam.org, 2019.

[56] G. P. Fettweis, "The tactile internet: Applications and challenges," *IEEE Vehicular Technology Magazine*, vol. 9, no. 1, pp. 64–70, 2014.

[57] P. Garrido, I. Sánchez, S. Ferlin, R. Agüero, and O. Alay, "rquic: Integrating fec with quic for robust wireless communications."

[58] A. Rabitsch, P. Hurtig, and A. Brunstrom, "A stream-aware multipath QUIC scheduler for heterogeneous paths," in *Proceedings of the Workshop on the Evolution, Performance, and Interoperability of QUIC*, 2018, pp. 29–35.

Hongjia Wu is a Ph.D. candidate at Simula and OsloMet. He obtained his M.Sc. in Embedded Systems from TU Delft and B.Sc in Automatic Control from Northeastern University. His research interests include multipath protocols and robotic systems.

Özgü Alay Dr. Ozgu Alay received the B.S. and M.S. degrees in Electrical and Electronic Engineering from Middle East Technical University, Turkey, and Ph.D. degree in Electrical and Computer Engineering at Tandon School of Engineering at New York University. Currently, she is an Associate Professor in University of Oslo, Norway and Head of Department at Mobile Systems and Analytics (MOSAIC) of Simula Metropolitan, Norway. Her research interests lie in the areas of mobile broadband networks, multipath protocols and robust multimedia transmission over wireless networks. She is author of more than 70 peer-reviewed IEEE and ACM publications and she actively serves on technical boards of major conferences and journals.

Anna Brunstrom received a B.Sc. in Computer Science and Mathematics from Pepperdine University, CA, in 1991, and a M.Sc. and Ph.D. in Computer Science from College of William & Mary, VA, in 1993 and 1996, respectively. She joined the Department of Computer Science at Karlstad University, Sweden, in 1996, where she is currently a Full Professor and Research Manager for the Distributed Systems and Communications Research Group. Her research interests include Internet architectures and protocols, techniques for low latency Internet communication, multi-path communication and performance evaluation of mobile broadband systems including 5G. She has authored/coauthored over 170 international peer-reviewed journal and conference papers.

Simone Ferlin is a software researcher at Ericsson AB in radio networks. She received her Dipl.-Ing. degree in Information Technology with major in Telecommunications from Friedrich-Alexander Erlangen-Nuernberg University, Germany in 2010 and her PhD degree in computer science from the University of Oslo, Norway in 2017. Her interests lie in the intersection of cellular networks and the Internet, with her research focusing on computer networking, QoS and cross-layer design, transport protocols, congestion control, network performance, security, and measurements. Her dissertation focused on improving robustness in multipath transport for heterogeneous networks with MPTCP. She actively serves on technical boards of major conferences and journals in these areas.

Giuseppe Caso is a Postdoctoral Fellow with the MOSAIC Dept., SimulaMet. In 2016, he received the Ph.D. degree from Sapienza University of Rome, where he was a Postdoctoral Fellow until 2018. From 2012 to 2018, he has held visiting positions at Leibniz University of Hannover, King's College London, Technical University of Berlin, and Karlstad University. His research interests include cognitive and distributed communications, IoT technologies, and WiFi/UWB positioning systems. He is an IEEE Member.