Reducing Mobile Web Latency Through Adaptively Selecting Transport Protocol

Jia Zhang, Shaorui Ren, Enhuan Dong, *Member, IEEE*, Zili Meng, *Graduate Student Member, IEEE*, Yuan Yang, *Member, IEEE*, Mingwei Xu, *Senior Member, IEEE*, Sijie Yang, Miao Zhang, and Yang Yue

***Abstract*— To improve the performance of mobile web services, a new transport protocol, QUIC, has been recently proposed as a substitute for TCP. However, with pros and cons of QUIC, it is challenging to decide whether and when to use QUIC in large-scale real-world mobile web services. Complex temporal correlation of network conditions, high user heterogeneity in a nationwide deployment, implementation diversity of QUIC variants limited, and resources on mobile devices all affect the selection of transport protocols. In this paper, we present WiseTrans, an adaptive transport protocol selection mechanism, to switch transport protocols for mobile web services online and improve the completion time of web requests. WiseTrans introduces machine learning techniques to deal with temporal heterogeneity, makes decisions with historical information to handle spatial heterogeneity, adopts an online learning method to keep pace with implementation variation, and switches transport protocols at the request level to reach high performance with acceptable overhead. We implement WiseTrans on two platforms (Android and iOS) in a popular mobile web service application of *Baidu*. Comprehensive experiments demonstrate that WiseTrans can reduce request completion time by up to 25.8% on average compared to the usage of a single protocol.**

***Index Terms*— Protocol selection, mobile web service, ML-based networking systems.**

Manuscript received 25 March 2022; revised 11 October 2022 and 29 December 2022; accepted 1 January 2023; approved by IEEE/ACM TRANSACTIONS ON NETWORKING Editor V. Aggarwal. This work was supported by the National Natural Science Foundation of China under Grant 62002192, Grant 62221003, Grant 61832013, and Grant 61872209. This work has been partly published in Proceedings of the Web Conference 2021 [DOI: 10.1145/3442381.3449958]. *(Corresponding authors: Mingwei Xu; Enhuan Dong.)*

Jia Zhang is with the Department of Computer Science and Technology, BNRist, Tsinghua University, Beijing 100084, China (e-mail: jia-zhan18@mails.tsinghua.edu.cn).

Shaorui Ren is with the Department of Electronic Engineering, BNRist, Tsinghua University, Beijing 100084, China (e-mail: rsr19@mails.tsinghua.edu.cn).

Enhuan Dong is with the Institute for Network Sciences and Cyberspace, BNRist, Tsinghua University, Beijing 100084, China, also with the Zhongguancun Laboratory, Beijing 100094, China, and also with the Quan Cheng Laboratory, Jinan 250103, China (e-mail: dongenhuan@tsinghua.edu.cn).

Zili Meng is with the Institute for Network Sciences and Cyberspace, BNRist, Tsinghua University, Beijing 100084, China (e-mail: zilim@ieee.org).

Yuan Yang is with the Department of Computer Science and Technology, BNRist, Tsinghua University, Beijing 100084, China, also with the Zhongguancun Laboratory, Beijing 100094, China, and also with the Peng Cheng Laboratory, Shenzhen 518066, China (e-mail: yangyuan_thu@mail.tsinghua.edu.cn).

Mingwei Xu is with the Department of Computer Science and Technology, Institute for Network Sciences and Cyberspace, Tsinghua University, Beijing 100084, China, and also with the Quan Cheng Laboratory, Jinan 250103, China (e-mail: xumw@tsinghua.edu.cn).

Sijie Yang is with Tencent Technology (Beijing) Company Ltd., Beijing 100000, China (e-mail: jeffyang@tencent.com).

Miao Zhang is with Baidu Inc., Beijing 100085, China (e-mail: zhangmiao02@baidu.com).

Yang Yue is with the Department of Automation, BNRist, Tsinghua University, Beijing 100084, China (e-mail: yueyang22@mails.tsinghua.edu.cn).

Digital Object Identifier 10.1109/TNET.2023.3235907

I. INTRODUCTION

RECENTLY, a sharp increase in the usage of mobile web services has been observed. The latest statistics demonstrate that mobile users have become the most significant proportion of global Internet users [2]. In this case, the performance of mobile web services, such as request completion time,[1] is becoming more and more critical for service providers. For example, Google reported that 53% of mobile users give up waiting if a page fails to be loaded within three seconds [3]. As a consequence, mobile web service providers strive to reduce the page load time and improve mobile web performance. In response, a new transport protocol, QUIC, has been proposed to cater for the increasingly stringent delay requirements from web services. As being standardized [4], QUIC has been becoming a new widely supported transport protocol for web services.

Yet, QUIC does not *always* perform better than TCP. Although QUIC could achieve better performance than TCP in various settings, recent studies do show that TCP sometimes works better in scenarios such as good network conditions, quality of service (QoS) strategies from Internet Service Providers (ISPs) [5], [6], [7]. Therefore, using a fixed transport protocol for all requests, all users will result in suboptimal performance. Moreover, in mobile networks, the network conditions could also frequently change. We are thus motivated to adapt the appropriate transport protocol for a better performance.

Therefore, we propose WiseTrans in this paper, an adaptive transport protocol selection mechanism to improve the performance of mobile web services. WiseTrans measures the network conditions on the client side and switches the transport protocols when necessary. By adaptively switching transport protocols, mobile web service users can enjoy benefits from both protocols and have a better experience.

However, it is non-trivial to decide whether a request in a mobile web service should use TCP or QUIC in a large-scale real-world mobile web service due to the following reasons (§II-B). First, the network condition during the connection could be fluctuating, especially in mobile network. In this case, the answer to whether using TCP or QUIC might

[1]There are also metrics in the optimization of mobile web service, such as page loading time. In this paper, we mainly focus on the HTTP request completion time.

also change with time. For example, even for one mobile user, the network conditions could be affected by the interference on wireless channels and the competition with other flows. It is difficult for providers to have a static answer of using QUIC or not. Second, for a large-scale service, connections from different users will traverse wide-area networks and mobile access networks with different behaviors (i.e., spatial heterogeneity). For example, different QoS preferences of ISPs and home router settings could affect the optimal choice between two protocols [5]. In this case, different users might have different protocol selection strategies. Third, the diverse configurations and implementations of protocols bring more challenges. As an evolving protocol, QUIC has various implementations of different parties and is still under development [8]. Even for TCP, emerging designs at the client side will also affect its performance [9]. In this case, the performance of TCP and QUIC will also be different among implementations. It is also challenging to consider the numerous variants during the selection between TCP with QUIC, and the need to make the selection mechanism future-proof to potential evolution. Finally, the computation resources on mobile clients are also limited, which requires WiseTrans to be lightweight enough for real-world deployment. For example, expressive but heavyweight models might not be able to be executed in the runtime [10]. Fine-grained actions (e.g., per-packet decision) will introduce considerable overhead in the statistics collection from the network stack. It is even more challenging to propose a protocol selection mechanism that could be deployed in practice for numerous users in the real world.

To address the above challenges, we design several building blocks in WiseTrans. First, to adapt to the temporal network fluctuations of mobile users, WiseTrans periodically measures the network conditions and decide whether using QUIC or TCP for upcoming requests. Second, to optimize towards heterogeneous users, WiseTrans collects historical decisions and performance, and then subsequently update and customize the protocol decision online for different users (§III-B). Third, to make WiseTrans generalizable to various protocol variants, we introduce a probabilistic exploration mechanism to update the behaviors of TCP and QUIC (§III-D). Finally, to make WiseTrans deployable for real-world large-scale services with minimum overhead while maintaining the performance improvements, we make a series of design choices including using tree-based machine learning algorithms for decision (§III-C), and carefully designing the decision-making granularity (§III-A).

We implement WiseTrans on two platforms (Android and iOS) in the production environment in *Baidu*. Extensive experiments demonstrate that WiseTrans could reduce the request completion time by about 25.6% on average compared to using a single protocol. WiseTrans can also achieve an accuracy of 88.3% of protocol selection for each request. Additionally, our results show WiseTrans' ability to generalize to unseen implementations. WiseTrans could reach more than 20% improvement on different implementations, consistent with the baseline method, which selects protocol based on an offline-trained model specialized for that implementation (§IV).

In summary, we make the following contributions:

- We motivate the problem and significance of transport protocol selection between TCP and QUIC with measurements from real-world mobile web service (§II).
- We propose WiseTrans, an adaptive transport protocol selection mechanism based on XGBoost for large-scale deployment of mobile web service (§III).
- We evaluate the performance improvements of WiseTrans against several baselines with extensive experiments with one of the mobile web services of *Baidu* (§IV).

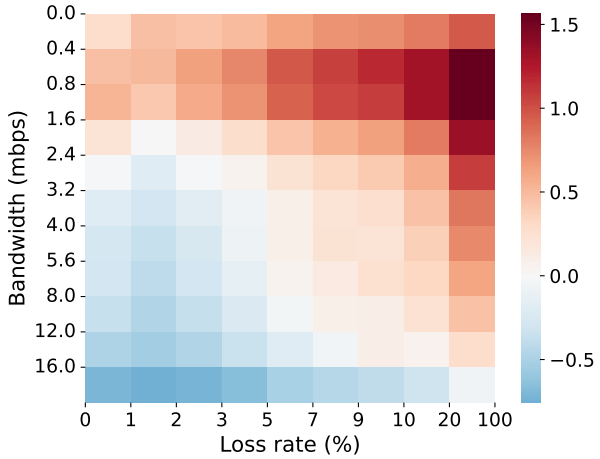


Fig. 1. $\ln \lambda$ with different bottleneck bandwidth and packet loss rate. Red color indicates QUIC outperforms TCP, and blue color indicates that TCP outperforms QUIC. λ is the ratio of the performance of TCP and QUIC. The performance here is the request completion time at the application layer for the mobile web service.

II. Background and Motivation

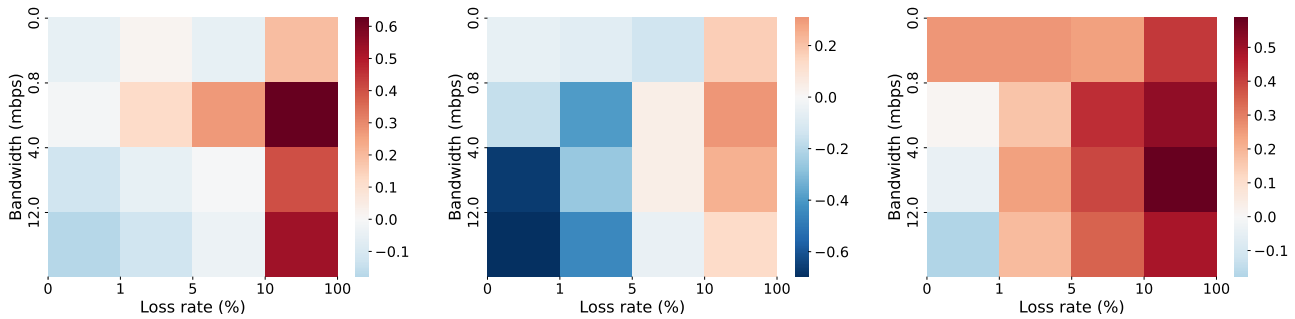
In this section, we first introduce the performance differences between QUIC and TCP for mobile web services in §II-A. We then present the challenges of the adaptive protocol switching for mobile web service in §II-B.

A. Background

After QUIC was proposed, lots of measurements have revealed the performance differences between TCP and QUIC. Specifically for mobile web service, we also conduct a week-long passive measurement campaign on a mobile web service with millions of requests. We summarize the key enablers that result in the performance difference of using TCP and QUIC. Our detailed passive measurements in the wild in Appendix also show that TCP and QUIC perform differently with different access network types in different geographical locations. See the appendix for the methodology and detailed results of our large-scale passive measurement on one mobile web service.

Network conditions. We present our measurement results of the performance of QUIC and TCP on different bandwidth and loss rate in Fig. 1. Red color indicates that QUIC outperforms TCP at that network condition, and deeper color indicates more significant improvements. The statistical average results of QUIC outperform TCP on those *slow net* (high loss rate and low bandwidth). Measurements on RTT demonstrate similar results. As also measured by recent advances, the higher performance of QUIC in those slow nets attributes to its better loss detection and recovery, as well as the elimination of Head-Of-Line (HOL) blocking and ACK ambiguity [5], [6], [7], [11].

Computation resources. Meanwhile, for mobile web services, the performance of QUIC is also affected by mobile device computation resource. Due to the high CPU overhead (up to



(a) Experiments in BJ with Ver. S (b) Experiments in SJZ with Ver. S (c) Experiments in BJ with Ver. E

Fig. 2. Performance difference between TCP and QUIC varies with user spatial heterogeneity and implementation diversity. Red color indicates QUIC outperforms TCP, and blue color indicates that TCP outperforms QUIC.

3.5× higher compared to TCP/TLS [5]), mobile devices with limited computation resources might experience performance degradation [6].

B. Design Challenges

However, as discussed in §I, deciding which protocol to use is non-trivial in a large-scale real-world deployment of a mobile web service. We have encountered the following major challenges:

Complex temporal correlation. Network condition fluctuates due to wireless channel fading or user moving. Therefore, if we categorize the best protocol for mobile users according to Fig. 1, the answer to whether using TCP or QUIC might also change with time. However, precisely measuring the current network condition and capturing the temporal relationship are challenging due to the complex correlation of network statistics. For example, existing research efforts have employed sophisticated mathematical tools to measure and predict even one network condition (e.g., Kalman filter for RTT [12]). Precisely combining the statistics from three network properties (bandwidth, RTT, and loss rate) and jointly optimizing the transport protocol selection would be more challenging.

High spatial heterogeneity. For a widely deployed mobile web service, users may come from different regions, using different ISPs and home devices. However, those in-network devices might have different preferences towards UDP, consequently affecting the performance of QUIC. For example, since UDP traffic is more likely to be adopted as malicious attack traffic, ISP gateways or home routers might have additional QoS strategies by randomly dropping, rate limiting, or even blocking UDP traffic [13], [14]. As Fig. 2a and Fig. 2b shows, users in different regions experience different protocol performance. As we illustrate in §IV-B, there is no CDN server cluster deployment in SJZ, users may experience more in-network devices and complicated ISP strategies. In our measurement, requests using QUIC encountered more loss events than TCP in SJZ. We believe that the performance difference could be due to the additional QoS strategies on UDP traffic. Since different ISPs and home routers might have different preferences, it is challenging to optimize the protocol selection for individual users.

Implementation diversity. Although QUIC protocol specifications have been published [4], many parties have constantly been updating more than 18 different QUIC/H3 implementations [8] and using them in their own applications. Different implementations, as well as the updates and optimizations in the process of deploying QUIC, will affect the protocol's performance [15], [16]. To verify the impact of implementation differences on performance, we use two different implementations. One is the stable version of one application (Ver. S), and the other is an experimental version that we print out some log for debugging (Ver. E). As Fig. 2a and Fig. 2c show, QUIC performs much better in almost all network conditions in Ver. E. Since different applications may implement different protocol versions, it is challenging to design a protocol selection scheme that suits all applications.

Limited resource on mobile devices. Deciding the proper operating granularity of a protocol switching algorithm is also challenging for mobile devices. On the one hand, packet-level measurements might result in burdensome overhead by querying the statistics of network stack [17]. Packet-level protocol switching will also result in consistency overhead at the transport layer. On the other hand, sparse granularity switching, like user-level and page-level, might lead to sub-optimal performance because of mobile users' temporal heterogeneity and may be too long to track the time-varying network condition [18].

Therefore, we are motivated to design an adaptive transport protocol switching mechanism for mobile web service with the above challenges in mind. In order to address the above challenges, the design goals of WiseTrans are: (1) being **expressive** enough to reflect the complex network conditions; (2) being **adaptive** enough to deal with unpredictable changes in protocol performance caused by user heterogeneity; (3) strong **generalization** ability among different implementations; (4) being **lightweight** enough to bring less additional overhead.

III. WiseTrans Design

In this section, we present the design of WiseTrans. We first present an overview (§III-A) of how WiseTrans achieves the design goals we present above with different design blocks. We then introduce each design component of WiseTrans, including a `Feature Extractor` (§III-B), a `Network Monitor` (§III-C), a `Probe Agent` (§III-D), and a `Rollback Checker` (§III-E).

A. Design Overview

WiseTrans architecture. To address the design challenges and achieve the design goals of expressive, adaptive and lightweight with the generalization ability among different implementations, we design several building blocks in WiseTrans, as shown in Fig. 3. First, to optimize for heterogeneous users, `Feature Extractor` collects not only historical

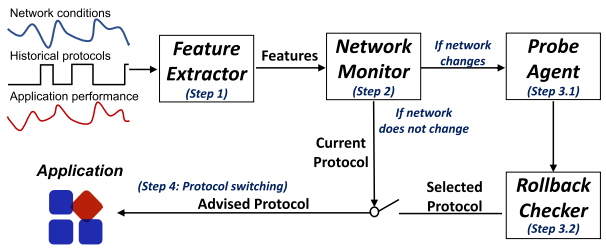


Fig. 3. Architecture of WiseTrans.

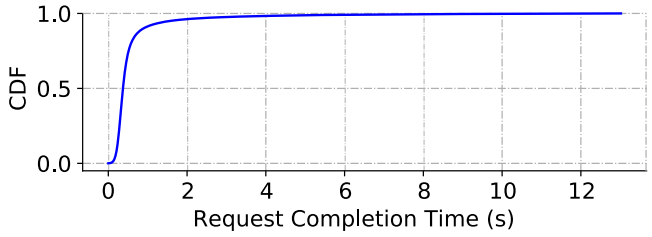


Fig. 4. Distribution of the request completion time of our mobile web service application in the wild. The measurement details are introduced in Appendix.

network conditions, but also historical decisions and their performance, and then subsequently updates and customizes protocol decisions online for different users (§III-B). Second, to handle the complex temporal relationship during a connection while ensuring the simplicity for large-scale deployment, `Network Monitor` employs a tree-based classification algorithm, XGBoost [19], to monitor the network condition (§III-C). Third, to make WiseTrans generalizable to various protocol variants and application implementations, `Probe Agent` explores the real-time behaviors of TCP and QUIC (§III-D). Finally, we introduce `Rollback Checker` as a guard for unexpected behaviors of network, and unexpected decisions of protocol (§III-E).

WiseTrans works on the client side. When the transport protocol selected by WiseTrans changes, the client simply sends subsequent requests to the server using the new protocol and the server can follow up with responses based on the type of protocol received. Therefore, WiseTrans only requires modification on the client side and it works well with any server that supports both TCP and QUIC protocols. Specifically, WiseTrans works as a shim layer between the application layer and transport layer. WiseTrans collects information from both application layer and transport layer, makes decisions, and allows the application to adopt the selected protocol.

WiseTrans workflow. As shown in Fig. 3, the workflow of WiseTrans is as follows:

Step 1: Feature extraction. Periodically, WiseTrans measures raw statistics for each user of the mobile web service. We will introduce which statistics to measure and how the `Feature Extractor` preprocesses historical data in (§III-B).

Step 2: Network monitoring. Next, WiseTrans employs an XGBoost-based [19] `Network Monitor` (§III-C) to detect changes of the network conditions based on the features from the `Feature Extractor`. When `Network Monitor` recognizes a significant change in network conditions, it is possible that the optimal protocol has changed, and then we enter the `Probe Agent`. Otherwise, we continue with the current protocol.

Step 3.1: Protocol exploration. From the perspective of `Probe Agent` (§III-D), WiseTrans consists of two phases, the exploration phase and the exploitation phase. When `Network Monitor` recognizes a significant difference in network conditions, `Probe Agent` will enter the exploration phase and re-select protocol by performing live experiments. `Probe Agent` sends requests with TCP and QUIC alternately, and select the protocol with better performance. In the exploitation phase, WiseTrans will use that selected protocol.

Step 3.2: Rollback check. After that, before putting the protocol selection into effect, WiseTrans checks if the user needs to roll back the selection with a `Rollback Checker` (§III-E). The rollback mechanism is designed to correct unexpected behaviors caused by the preferences of network devices or the decisions of the `Probe Agent`.

Step 4: Protocol switching. Finally, based on the output of the `Network Monitor` and `Probe Agent` with `Rollback Checker`, WiseTrans decides to still use the current transport protocol or switch to the other one for subsequent requests.

Operation granularity. As discussed before, an important design choice is the operation granularity of WiseTrans. Compared to the user-level classification in personal characterization of social networks [20] and packet-level decisions in network bandwidth prediction in the network stack [12], WiseTrans works on the request level due to the following considerations:

- **Overhead on mobile devices.** For mobile web services, the resources on mobile devices are limited. Therefore, packet-level measurements in the selection mechanism might result in burdensome overhead by frequently querying the statistics of the network stack [21]. Moreover, packet-level decisions usually need to modify the protocol stack [22], which is impractical for web service providers. In contrast, request-level measurements alleviate the overhead issue.
- **Timeliness of request.** Network conditions usually do not frequently change within a few RTTs, which are generally around hundreds of milliseconds or even longer [23]. As shown in Fig. 4, nearly 90% of requests are completed within one second from our measurements. Meanwhile, the latency in mobile web service could be up to hundreds of milliseconds [24]. In this way, switching the protocol at the request level is timely enough for mobile web service.
- **Consistency during protocol switching.** Moreover, operators need to consider the consistency during the protocol switching. For example, if we switch the protocol at the packet level, we need to reorganize packets from two connections into one request or response. Besides, packets also need reordering due to the potential out-of-order issues from two protocols. In contrast, one request does not depend on the completion of other requests in one page of mobile web service. Therefore, switching the protocol at the request level does not need to guarantee the order of request completion and reduce the overhead at the client side.

B. Feature Extractor

`Feature Extractor` collects and extracts features used for protocol selection. To address the user heterogeneity, we extract features representing not only historical network conditions, but also historical decisions and the consequent performance of those decisions. The decision boundary may be different for different mobile users and can be changeable for

TABLE I

DESCRIPTION OF FEATURES AND THEIR RANKING. THE FEATURE IMPORTANCE IS ESTIMATED AS THE AVERAGE INFORMATION GAIN. THE FEATURES SELECTED BY WISETRANS REFLECT NOT ONLY HISTORICAL NETWORK CONDITIONS, BUT ALSO HISTORICAL DECISIONS AND THEIR PERFORMANCE

Feature	Description	TCP		QUIC	
		Importance	Rank	Importance	Rank
Retran_rate	Retransmitted Bytes / Sent Bytes in time window	0.2849	1	0.2294	1
RTT_avg	Average RTT in time window	0.1368	3	0.1214	5
resp_rec_G	Received Bytes / The time interval of the client receiving the first and last bytes of the response body	0.1273	5	0.1462	4
TTFB_avg	Average TTFB in time window	0.1421	2	0.1904	2
TTFB	TTFB of the current request	0.1308	4	0.0929	6
TTFB_rate	TTFB / TTFB_avg	0.1072	6	0.1737	3
req_cmpl_G	Received Bytes / The time interval between the client sending request and receiving the whole response body	0.0708	7	0.0469	7

an individual user due to spatial heterogeneity. For example, since ISP gateways and home routers might have additional QoS strategies by randomly dropping, rate limiting, or even blocking UDP traffic, different users may have different decision boundaries shown in Fig. 1. Simply considering network conditions cannot optimize protocol selection for individual users. In another scenario, the access network and ISP change when a user moves, which will bring the decision boundary changes. The same network condition may correspond to different protocol selection decisions with such changes, making historical network conditions not enough for decision-making.

Therefore, we adopt three kinds of features,[2] as shown in Tab. I.

- **Historical network conditions.** We use round-trip time (RTT), bottleneck bandwidth ($BtlBw$), and packet loss rate ($LossRate$) to model a network path, following [22], which can describe the network conditions a user experienced. Based on the design choice mentioned in §III-A, we collect request-level average RTT (RTT_avg), throughput ($resp\_rec\_G$), and retransmission rate ($Retran\_rate$) as network conditions.
- **Historical decisions.** The transport protocol should also be considered. Historical decisions and their subsequent performance can address the heterogeneity and variability of the decision boundary. The possible rate limit of QUIC/UDP for some users can be reflected in the similar network conditions and the same protocol but different performance. In this way, the changes in the decision boundary will be explored.
- **Historical requests' performance.** We select Time to First Byte ($TTFB$) and request completion goodput (req_cmpl_G) as the performance of the past decision. As shown in Fig. 8, TTFB in this paper is the time from the client sending the request to the client receiving the first byte, which implies the properties of the request transport. We collect historical average TTFB ($TTFB\_avg$) to estimate the application performance. We also calculate and the ratio of TTFB of current request to the average TTFB ($TTFB\_rate$), aiming to capture the transient performance changes. req_cmpl_G measures request-level goodput, dividing the user's received bytes by the time interval between the client sending a request and receiving the whole response body.

[2]We use the Information Gain to analyze the feature importance. Information Gain computes the difference of entropy between before and after split and specifies the impurity in class elements. We use the average gain across all splits the feature is used in as the importance.

For each request, `Feature Extractor` records the required information of the request, as well as the information of the connection from the HTTP logs and the socket logs.

C. Network Monitor

We build a model in `Network Monitor` to detect the changes in network conditions. As the network environment changes frequently and complicatedly, it is difficult to decide when to perform live experiments through only observing the current network conditions. The design of the `Network Monitor` is based on the following considerations. First, the criterion for the extent of network condition changes is supposed to be closely related to the performance of TCP and QUIC. For example, as shown in Tab. I, since loss is one of the critical factors affecting the performance difference between TCP and QUIC, a slight change in loss rate should result in a significant change in the indicator. Second, the model trained with specific implementations of QUIC and TCP can still be used to understand the changes in network conditions even in different implementations. This is mainly because that the performance difference between TCP and QUIC is due to their design principle. Therefore, the trend of performance differences between QUIC and TCP with network conditions is consistent across implementations [5], [25]. Therefore, we train the model offline to reflect the extent of network changes, which can generalize to other protocol implementations.

Model. As introduced before, `Network Monitor` should be able to faithfully learn the temporal correlation and spatial heterogeneity based on the features from `Feature Extractor`. Potential models include linear regression, supporting vector machine, decision tree, random forests, and even neural networks. In this paper, WiseTrans employs XGBoost, a tree-based model, due to the following reasons:

- **Expressive.** Due to the complexity of features, the algorithm should be expressive enough to capture the relationship among features. In our experiments in §IV-E.2, XGBoost is capable of precisely capturing the relationship and has a satisfactory accuracy of around 90%.
- **Lightweight.** Due to the resource limitation on mobile devices, the algorithm should also be lightweight enough to avoid additional overhead. Therefore, sophisticated algorithm (e.g., neural networks, integer programming) are not practical without additional hardware acceleration. As a tree-based algorithm, XGBoost could be efficiently executed on mobile devices in a negligible time (3ms in §IV-C).

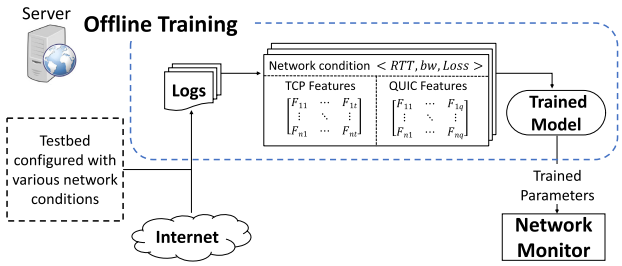


Fig. 5. Process of WiseTrans Offline Training. A server collects information of requests under various network conditions, using TCP and QUIC separately, and labels automatically.

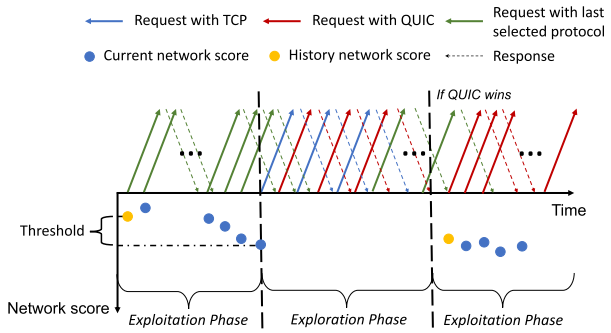


Fig. 6. Time-diagram of WiseTrans. WiseTrans includes two phases. WiseTrans performs live experiments in the exploration phase and uses the selected protocol in the exploitation phase.

- **Interpretable and verifiable.** Meanwhile, deploying the monitor online in a production environment also requires the model to be interpretable and verifiable [26]. Deploying a black-box model (e.g., neural networks) online might result in unexpected behaviors. In contrast, recent advances in the machine learning community have demonstrated the verifiability and robustness of tree-based models, including XGBoost [27].

Output and Training. As illustrated before, `Network Monitor` gives an indicator that reflects the current network conditions, and the fluctuation of this indicator will be used for `Probe Agent` as a condition for entering the exploration phase. For that purpose, the indicator should score the network conditions by their influence on the performance of TCP and QUIC. Therefore, the output of `Network Monitor` is not only the objective reflection of network conditions but also related to the performance of TCP and QUIC under current network conditions.

As for the training of XGBoost, WiseTrans learns the model offline. We label the features from `Feature Extractor` according to the performance comparison of using TCP and QUIC. WiseTrans learns the model with each network condition as features and the optimal protocol under it as labels. WiseTrans discovers how the features influence the performance of TCP and QUIC through numerous fine-grained experiments, as shown in Fig. 5. Therefore, WiseTrans can learn the model based on the extracted features (listed in Tab. I) and the data labels. We leverage Grid Search [28] and cross-validation for model selection and hyperparameter tuning. We further evaluate the sensitivity of parameters in §IV-E.2.

D. Probe Agent

`Probe Agent` performs live experiments, compares the performance of TCP and QUIC, and selects the subsequent protocol. `Probe Agent` includes two phases, the exploration phase and the exploitation phase. In the exploration phase, `Probe Agent` performs live experiments by sending requests using TCP and QUIC alternately and compares the performance. The proper protocol will be used in the exploitation phase. `Probe Agent` could address both the spatial heterogeneity and implementation diversity. Through live experiments, the different preference to UDP/QUIC will be reflected by the poor performance of QUIC, and the better TCP protocol will be selected. Meanwhile, with live experiments, `Probe Agent` could estimate TCP and QUIC performance under the current network and select without an advance analysis of the protocol implementations.

There are two observations from the online practice of WiseTrans that ensure the effectiveness of `Probe Agent`:

- **Infrequent switching.** Although wireless links fluctuate rapidly, the frequency of protocol switching is not high. As shown in Fig. 12b, the switching frequency is about 1%-2%, which means protocol switching is needed only when the network undergoes drastic changes. Therefore, we monitor network condition change and perform live experiment only when the network conditions vary significantly. In this case, although live experiments will introduce performance degradation, its low frequency could reduce the impact of performance degradation.
- **Timeliness of live experiment.** We run live experiments by using TCP and QUIC alternately for several requests. We found that users' click behavior on the mobile web service will trigger about 13.5 concurrent requests on average, which will complete in about 1.5 seconds. Moreover, due to the low switching frequency, we believe that network conditions generally will not undergo drastic changes within several seconds. Therefore, the performance of live experiments in the exploration phase can be considered valid and could guide the selection of subsequent transport protocol.

Fig. 6 shows a control cycle of `Probe Agent`, where the lines colored blue, red, and green represents requests sending with TCP, QUIC, and last selected protocol (TCP or QUIC). We describe the key design of `Probe Agent` by answering the following two questions.

How to perform live experiments? As introduced before, the design goal of `Probe Agent` is to discover the proper protocol by live experiment directly. `Probe Agent` performs live experiments by using TCP and QUIC alternately, as shown in Fig. 6. After receiving the responses of these requests, `Probe Agent` compares the performance and selects the better protocol.

A key design choice is the number of requests sent for probing during the exploration phase. On the one hand, live experiments could be affected by unpredictable external events in the network and at the server side. For example, if there is an accidental routing error, a sudden burst from other flows or a temporary overload on the server, delay could increase dramatically. On the other hand, if the live experiments take too long, the network might be changing over time for reasons unrelated to the `Probe Agent`'s action. Considering the above two points, we conducted experiments that TCP and QUIC each sent requests with a number of two, three, and four, and found that the value of three is the most appropriate. Due to the additional time of live experiment, the average request completion time that sending four requests is 0.92% longer than the value of three. While the performance of two requests is 4.02%

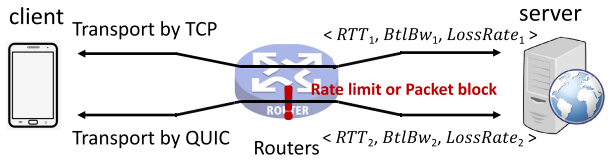


Fig. 7. Transport model when meeting QUIC/UDP rate limit or packet block.

worse than the value of three. This is because that about 5.38% of the live experiments showed abnormal results with the value of two, making the protocol selection decision wrong. Therefore, to obtain accurate and timely measurement results, we use TCP and QUIC alternatively three times each. We use the two similar performance values of the three requests for each protocol and select the protocol with the better average performance.

The exploration phase ends when `Probe Agent` receives the response of the third request with QUIC. Requests sent after the third request in the exploration phase will still use the last selected protocol. Note that `Probe Agent` performs live experiments by the application's real request and does not send occasional probes or use throwaway data for measurements.

When to enter the exploration phase? Since we use both TCP and QUIC in the exploration phase, there will be several requests sent by the wrong protocol and might bring performance degradation. Therefore, frequent live experiments will suffer sub-optimal performance in the exploration phase, while low exploration frequency can lead to the use of a wrong protocol for a long time. `Probe Agent` should enter the exploration phase when the optimal protocol is much more likely to change, i.e., the following two scenarios:

- **Establish new connections.** As the preference of ISP and in-network devices could change with path, every time a new connection is established and a new path is used, exploration is needed to determine the optimal protocol.
- **Network conditions change significantly.** As introduced before, when network condition changes slightly, the optimal protocol may not change. Therefore, we use the network conditions scored by `Network Monitor` and retain the network condition score after the last exploration phase as the history score. When the difference between the current network score and the history score exceeds a threshold we set, `Probe Agent` enters the exploration phase.

E. Rollback Checker

We introduce `Rollback Checker` as a guard for unexpected behaviors of network, and unexpected decisions of protocol. In most cases, the protocol decided by the `Network Monitor` and `Probe Agent` is effective, but there are also some cases where the unexpected behavior of the network, server or user can upset the conclusions. Firstly, the sudden fluctuation of network conditions will reduce the effectiveness of `Probe Agent`. Furthermore, it is challenging for `Network Monitor` and `Probe Agent` to capture the possible UDP/QUIC block and rate limit and react to the instant changes, as shown in Fig. 7. In addition, as described before, the protocol selection could be affected by unpredictable external events in the network and at the server side, such as accidental routing error or a sudden overload of the server. Therefore, `Rollback Checker` is needed for the wrong decisions.

`Rollback Checker` starts working when a protocol switching occurs. In view of the above situations, `Rollback Checker` compares the performance of requests sent in the exploitation phase over a period of time before and after the switching. If a significant performance deterioration is found, `Rollback Checker` rolls back to the last protocol.

IV. Evaluation

In this section, we first introduce the baseline algorithms we use in our evaluations (§IV-A), and then the setup of our evaluation (§IV-B). We evaluate WiseTrans in the following aspects:

- *Performance in the Wild.* We evaluate the performance of WiseTrans against a fixed transport protocol in the real world. From our experiments, WiseTrans can reduce the average request completion time by about 26.5% compared to using one fixed protocol (§IV-C).
- *Generalization analysis.* We evaluate WiseTrans's ability to generalize to unseen implementations. WiseTrans could achieve more than 20% improvement on different implementations, consistent with the baseline method which selects protocol based on a offline-trained model specialized for that implementation (§IV-D).
- *Component Effectiveness.* We then evaluate the necessity and effectiveness of the design of WiseTrans. Compared to simple heuristic methods, WiseTrans shows more than 6.23% performance advantages (§IV-E.1). The model used in WiseTrans can achieve a classification accuracy of 91% compared to other methods (§IV-E.2). Meanwhile, about 78% of the decisions made by `Probe Agent` bring positive performance gain on average (§IV-E.3). Finally, the `Rollback Checker` can also reduce the average completion time by 7.3% (§IV-E.4).

A. The Baseline Algorithms

To demonstrate the performance improvements of WiseTrans, we have implemented the following algorithms in our experiments.

1) WiseTrans-v1: In the earlier version of this work [1], we did not introduce the `Probe Agent` for the online learning during a connection. Therefore, we differentiate WiseTrans into WiseTrans-v1 and WiseTrans-v2. WiseTrans-v1 [1] uses an offline-trained ML-based protocol classifier to directly select the protocol without any live experiments. We present the results of WiseTrans-v1 to demonstrate the necessity to enhance the generalization ability in the large-scale deployment.

2) WiseTrans-v2: Consequently, WiseTrans-v2 implements all components presented in this paper.

B. Experimental Setup

1) WiseTrans Client: We implement WiseTrans on two platforms (Android and iOS) of the client of a mobile web service application of *Baidu*. The implementation contains about 4000 lines of C code. To implement XGBoost in WiseTrans `Network Monitor` in §III-C, we train two models separately according to the current transport protocol. We utilize the maximum depth of 7 and minimum child weight of 1, and the top 5 features for requests transported by QUIC while the top 7 features for requests transported by TCP.

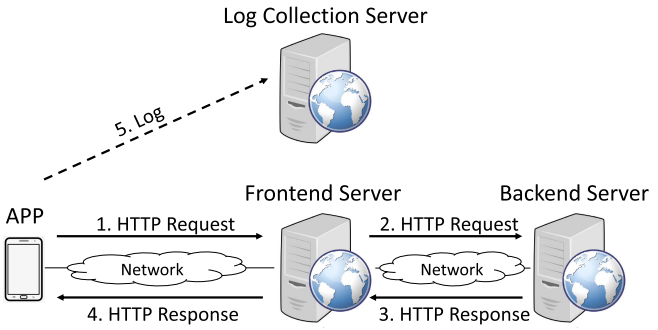


Fig. 8. Experimental setup of WiseTrans.

2) Web Service Server: In the server-side implementation of the mobile web service, we have frontend servers for load balancing and backend servers for request processing. The connections between frontend servers and backend servers are in the internal enterprise network of *Baidu* and are over private protocols. Therefore, connections optimized by WiseTrans are referred to the connections between the mobile user and the frontend servers.

3) Dataset: For offline training, we collect 38 hours, more than 128,000 request logs on a mobile web service app of *Baidu* as the training dataset. For evaluation, we enable WiseTrans for certain users in two cities (BJ and SJZ in China), with and without CDN server cluster deployment, and collect logs for 105 hours, resulting in more than 160,000 request logs. Also, we randomly collect more than 96,000 request logs using TCP and QUIC as our baseline.

C. Request Completion Time

Fig. 9 shows the improvement of WiseTrans on request completion time. We also select the shorter average completion time between TCP and QUIC as Ideal for each network condition, as a comparison. As the result shows, WiseTrans outperforms TCP and QUIC. WiseTrans-v1 has a 20.65% reduction in average completion time compared to TCP and 4.45% to QUIC. WiseTrans-v2 further improves performance and achieves a 25.8% reduction of TCP and 10.65% to QUIC. For the median completion time, WiseTrans-v1 also has a 13.38% and 10.69% reduction, and WiseTrans-v2 has a 14.10% and 11.42% reduction. WiseTrans-v1 also improves the stability of performance.[3] WiseTrans-v1 has a 34.23% reduction in the standard deviation of the request completion time compared to TCP and 5.47% to QUIC. WiseTrans-v2 further has a 37.72% reduction to TCP and 10.47% to QUIC. In all, both WiseTrans-v1 and WiseTrans-v2 is just about 1%-2% longer than Ideal, which demonstrates that WiseTrans is efficient.

Fig. 10 shows the detailed reduction ratio of WiseTrans compared to TCP and QUIC as the request completion time increases. Obviously, WiseTrans-v2 shows great improvement compared to TCP and QUIC for the 99^{th} percentile (the tail) completion time. WiseTrans-v2 has a 49.1% reduction, about 7s compared to TCP, and a 7.68% reduction compared to QUIC. For WiseTrans-v1, the reduction is about 45.9% and 1.67%. Results show that WiseTrans effectively improves

[3]Note that we evaluate WiseTrans in production environment on real users, the value of the error bar is the standard deviation of the request completion time.

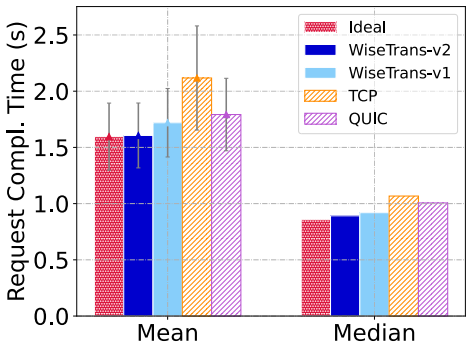


Fig. 9. WiseTrans achieves lower request completion time than TCP and QUIC in both cities. The reduction could achieve 25.8% compared to using one fixed transport protocol.

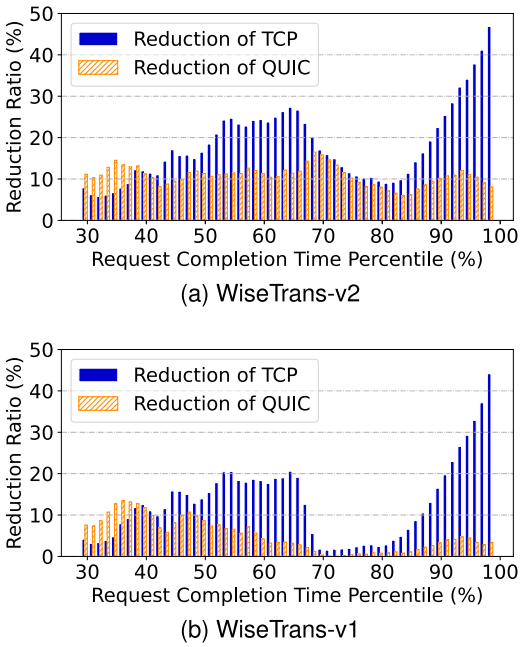


Fig. 10. WiseTrans's reduction of TCP and QUIC with different percentile of request completion time. WiseTrans-v2 has a reduction of 49.1% for the 99^{th} percentile (the tail) completion time.

users' experience who have experienced an extremely long request completion time.

The reductions of TCP and QUIC are always complimentary. From Fig. 10, it can be seen that when request completion time is in the shorter half, which means under better network conditions, QUIC has poor performance, and WiseTrans has significant improvements. While TCP always causes a long tail, WiseTrans significantly improves TCP's performance when the completion time is longer.

Such results reflect the key idea of WiseTrans, selecting the protocol with better performance in the current network condition. Specifically, WiseTrans should consistently achieve the optimal performance of using QUIC or TCP. Therefore, WiseTrans will just have better performance than TCP or QUIC for one network condition while can achieve the overall optimal for users across the network.

Overhead. We evaluate the overhead from two aspects.

- Additional delay. The additional decision-making time consumed by WiseTrans is less than 3 ms in our experiment, which is negligible compared to the request completion time. `Feature Extractor` is the most

time-consuming part and takes about 0.7 ms, due to the per-ACK and per-request feature collection and extraction. The second time-consuming part is the `Network Monitor`. For each predict, the model takes about 0.02 ms $\sim$ 0.04 ms. As for the additional establishment time, 89.73% of switches reuse existing connections in our experiments, which brings negligible overhead in general.

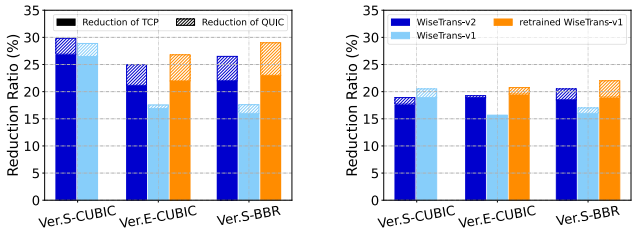
- Resource consumption. Firstly, the switching frequency is less than 3%, as shown in Fig. 12b and Fig. 15, which reduces the additional consumption caused by WiseTrans. Secondly, we measured the overhead in power on the mobile devices. We found that WiseTrans's power consumption was about 1.15 times higher than using TCP, and is approximately the same as QUIC. This suggests that the overhead introduced by WiseTrans is acceptable.

D. Generalization Analysis

In the experiments above, WiseTrans-v2 and WiseTrans-v1 were trained with a set of traces collected in the stable version of a mobile web service application of *Baidu*. The real world experiment was conducted by the same stable version. This stable version uses CUBIC as the congestion control algorithm for both TCP and QUIC. However, in practice, the implementation of the application and transport protocols, especially the adopted TCP and QUIC variant, could be different between applications and will be updated in the future, which will result in changes of the decision boundary. We conduct a comparative experiment to evaluate WiseTrans's ability to generalize to other implementations.

We compare the generalization ability of WiseTrans-v2 and WiseTrans-v1 between different application implementations and transport protocol variants. For the application implementations, besides the stable version (Ver. S) used by real users, we evaluate on an experimental version (Ver. E) that we manually print out some log for debugging. We just add print commands for TCP. Since I/O operations bring much overhead, QUIC performs much better in almost every status (Fig. 20c). For protocol variants, we also consider BBR as the congestion control algorithm. For further analysis, we retrain a new WiseTrans-v1 specialized for the new implementation and compare the performance of WiseTrans-v2, the initial WiseTrans-v1, and the retrained WiseTrans-v1 in the two different implementations.

Fig. 11 shows the improvement of WiseTrans with two different versions on request completion time compared to using one fixed protocol. The solid bar represents the reduction of request completion time compared to using TCP only, and the dashed bar represents the reduction to QUIC. Both WiseTrans-v1 for Ver. S-CUBIC and the retrained WiseTrans-v1 for unseen Ver. E-CUBIC and Ver. S-BBR could achieve a 17%-27% reduction in average completion time compared to TCP and 2%-6% to QUIC. It means that, by collecting traces from one certain application, training the classification model offline, and selecting the protocol by online historical information, a specialized WiseTrans-v1 could achieve significant performance improvement on that application. However, a problem with the supervised-learning-based WiseTrans-v1 is that, when the protocol implementation significantly changes, the original WiseTrans-v1 will bring performance degradation and a new WiseTrans-v1 is needed to be re-learned. We find that for the unseen implementation Ver. E and congestion control algorithm BBR, initial WiseTrans-v1's improvement degrades about 2.9%-10.68%.



(a) Reduction of the average Compl. Time (b) Reduction of the median Compl. Time

Fig. 11. WiseTrans achieves consistent significant improvement with unseen transport layer algorithms (BBR) and application implementations (Ver. E). The solid bar represents the reduction of request completion time compared to TCP, and the dashed bar represents the reduction to QUIC.

WiseTrans-v2 does not use a hardwired mapping of network conditions to the advised protocol but select protocol by performing live experiments and observing protocol performance. Therefore, for the unseen implementation Ver. E and BBR, WiseTrans-v2 basically maintaines consistent high performance, achieving a 24.98% total improvement of the average request completion time for Ver. E and 26.5% for BBR.

These results suggest that, in practice, WiseTrans-v2 will likely be able to generalize to a broad range of implementations and protocol variants adopted by its applications. Although WiseTrans-v1 could achieve good performance, it could not generalize well to other implementations. Further, for content providers, the process of collecting traces, retraining new models, and implementing them carefully in applications will bring heavy workloads. Also, the retraining frequency depends on the degree of difference between the new implementation and the existing one, which is also an extensive experimental work.

E. WiseTrans Deep Dive

In this section, we evaluate the effectiveness of the design of WiseTrans and provide a deeper understanding. We begin with illustrating the necessity of using machine learning methods by comparing WiseTrans to a naive strawman method. We then evaluate the performance of the model used in `Network Monitor` and evaluate its training process. After that, we demonstrate the frequency and effectiveness of live experiment. Finally, we conduct experiments to understand the effectiveness `Rollback Checker`.

1) Necessity Analysis of Machine Learning: WiseTrans uses machine learning methods complemented by multiple features to handle the temporal network fluctuations (§II-B). To analyze the necessity of using machine learning methods, we compare WiseTrans-v1 with a simple heuristic method. Both WiseTrans-v1 and the strawman method adopt static model for protocol selection.

Strawman method. A simple heuristic method is to select the better protocol from the offline measured results. For example, as shown in Fig. 1, the optimal decision of using TCP or QUIC has a clear linear decision boundary. Therefore, a straightforward strawman solution fit the results of the better protocol with different network conditions $< RTT, BtlBw, LossRate >$ from the real-world measurements into a linear decision boundary. When running online, the strawman method measures the current network conditions

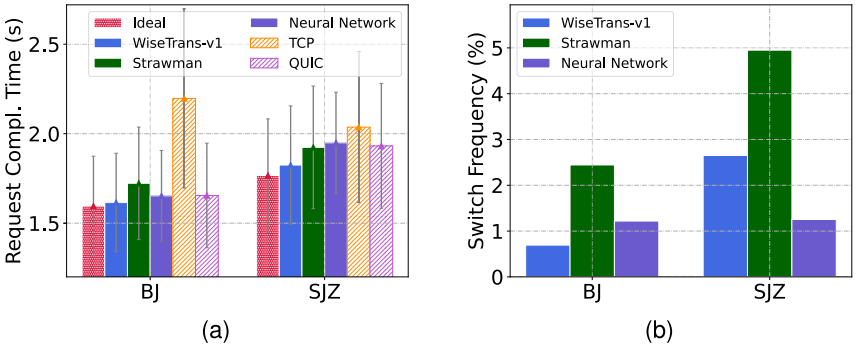


Fig. 12. Simple heuristic method and neural network model could not capture the complex temporal relationship of network conditions and result in sub-optimal performance. (a) The request completion time of using the heuristic method, neural network model and WiseTrans. (b) The protocol switch frequency of using the heuristic method, neural network model and WiseTrans.

and looks up for the optimal decision based on the pre-computed boundary.

Fig. 12a shows a significant performance gap (6.23%) between the strawman method and WiseTrans-v1. WiseTrans-v1 outperforms using one fixed protocol in two cities, while the strawman method is sometimes worse than using QUIC only. For the situation with a client located in BJ, WiseTrans-v1 has a 26.5% reduction in average completion time compared to TCP and 2.4% to QUIC. As Fig. 12b shows, the strawman method performs more protocol switching (2-3 times more than WiseTrans-v1), which suggests that simple heuristic methods fail to capture the complex temporal relationship of network conditions. Unlike heuristic methods, WiseTrans can incorporate a large amount of historical network conditions into its model to optimize the protocol selection.

These results suggest that simple modeling and linear mapping method is not sufficient to handle the temporal fluctuation. Firstly, the three features we extracted in Strawman method are not enough to model the network conditions. RTT, throughput and retransmission rate are observation variables, and there is a certain gap between them and the real state of the network. Furthermore, the observed variables are not independent of each other. For example, the throughput a flow could achieve is correlated with its RTT and retransmission rate. Therefore, `Feature Extractor` extracts more features to assist the link state mapping. In addition, simple mapping could not address the complex relationship of network conditions. Coarse-grained user chunking and average performance statistics may lose important information and reduce accuracy. However, fine-grained one-to-one mapping based on selected features will be limited by sparsity of measurement results. Meanwhile, one-to-one mapping itself could address the complexity of temporal relations. Therefore, we use a mush more expressive algorithm in `Network Monitor`.

2) Network Monitor Deep Dive: XGBoost is utilized in `Network Monitor` in consideration of the complex temporal correlation as well as the simplicity for large-scale deployment. `Network Monitor` outputs probability that the network condition is changed during online running. We compare XGBoost with a set of standard machine learning algorithms (Support Vector Machines (SVM) [29], Decision Tree [30], Random Forest [31] and Neural Network [32]) to explore whether XGBoost can address the complex temporal heterogeneity.

Accuracy. Tab. II and the ROC curves in Fig. 13 show that XGBoost achieves the best performance. SVM only achieves an accuracy of 76.87%, which means the decision boundary

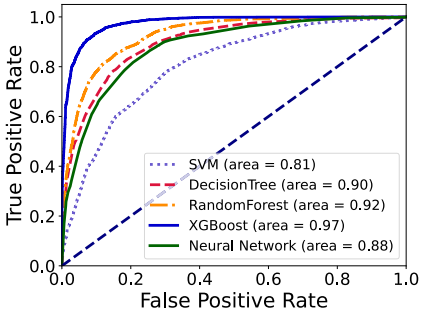


Fig. 13. ROC curves used for algorithms comparison. XGBoost achieves the best performance.

can hardly be found by simply employing hyperplanes in three-dimensional space $< RTT, BtlBw, LossRate >$. Meanwhile, the simple tree model, Decision Tree, cannot describe the complex relationship between the historical network conditions, decisions, and performance by only one single tree as well. Among the algorithms with multiple trees, XGBoost outperforms Random Forest by its faster convergence and higher accuracy. Also, the neural network[4] does not show good performance due to our training dataset with heterogeneous features, small dataset size, and large extreme values [33], [34].

The accuracy of WiseTrans's model is listed in Tab. III. Its model is trained offline using XGBoost. We apply it to the real-world network to examine its generalization ability. The overall accuracy of WiseTrans is about 88.29% in BJ and 87.19% in SJZ, which is basically consistent with the performance of offline testing. It proves that our model can also achieve high accuracy in the real network.

Performance. We compared WiseTrans-v1 with the neural network model in Fig. 12a and Fig. 12b, and find a performance advantage of about 2.27% to 6.81%. This indicates that XGBoost, which is more accurate at offline training period, also achieves better performance in the real environment. Although the neural network model has the highest recall (more than 95% in Tab. III) for selecting QUIC protocol, it still performs poorly and fails to make proper decisions when TCP and QUIC have similar performance (e.g. users in SJZ).

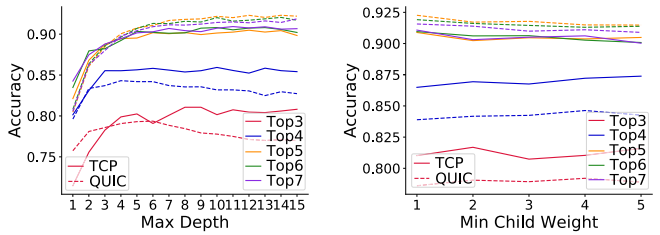
[4]We adopted a simple three-layer neural network with two hidden layers with 128 and 64 units respectively. During the training and parameter tuning process, we found that scaling up the network size and depth could not bring performance improvement.

TABLE II
COMPARISON OF MACHINE LEARNING ALGORITHM CANDIDATES. XGBOOST ACHIEVES THE BEST PERFORMANCE

Algorithm	TCP			QUIC		
	Train time (s)	Accuracy	Precision/Recall/F1-score	Train time (s)	Accuracy	Precision/Recall/F1-score
SVM	776.216 s	0.7687	0.7687/0.7696/0.7691	2036.439 s	0.7059	0.7056/0.7012/0.7034
Decision Tree	**0.094 s**	0.8246	0.8270/0.8269/0.8269	**0.137 s**	0.8257	0.8250/0.8265/0.8257
Random Forest	18.524 s	0.8612	0.8612/0.8604/0.8608	26.652 s	0.8502	0.8492/0.8504/0.8498
XGBoost	2.069 s	**0.8961**	**0.8968**/0.8950/**0.8959**	2.772 s	**0.9106**	**0.9098**/0.9105/**0.9102**
Neural network	627.408 s	0.8355	0.8483/**0.9532**/0.8977	844.599 s	0.7652	0.7744/**0.9655**/0.8595

TABLE III
ACCURACY OF CLASSIFICATION

Dataset	Offline Test Set	BJ	SJZ
Accuracy	0.9034	0.8829 (-2.26%)	0.8719 (-3.48%)



(a) Train with maximum depth. (b) Train with minimum child weight.

Fig. 14. Grid search results (mean accuracy) of XGBoost for the optimal model and features.

Hyperparameters analysis. We perform hyperparameters tuning to achieve the best performance. We mainly consider the maximum depth and minimum child weight in `Network Monitor`. Fig. 14 shows the accuracy with different hyperparameters using the grid search with 10-fold cross-validation. The accuracy is improved with the maximum depth increasing, while the maximum depth deeper than seven can only bring limited improvement but may introduce additional overhead. Also, the minimum child weight is robust enough. Therefore, we utilize the maximum depth of 7 and the minimum child weight of 1 for both models.

3) Probe Agent Deep Dive: We introduce `Probe Agent` to select protocol directly by the online performance. Since we use both TCP and QUIC in the exploration phase, there will be several requests sent by the wrong protocol and bring performance degradation. Therefore, frequent live experiments will suffer sub-optimal performance in the exploration phase, while low exploration frequency will lead to the use of wrong protocol for a long time.

In our evaluations, `Probe Agent` conducts live experiments with the frequency of about 1.76% in ver. S and 1.85% in ver. E. As shown in Fig. 15, we counted the probability of live experiments occurring in different network conditions. Similar to Fig. 2, `Probe Agent` performs fewer live experiments when QUIC has obvious advantages, which is under high packet loss rate, as well as when TCP has obvious advantages, which is under low packet loss rate and high bandwidth. `Probe Agent` is more likely to perform live experiment when TCP and QUIC have similar performance.

To evaluate the effectiveness of `Probe Agent`, we analyze and plot the changes in average request completion time before and after each protocol switching after the live experiment in Fig. 16. We observe that about 78% of the protocol switches bring positive performance gain on average. The result validates that live experiment can bring performance improvement. Also, we find that about 75% of the switches resulted in a slight performance change of less than 20%. It is because that the network condition did not change significantly in most cases. It also shows that WiseTrans-v2 is sensitive enough to capture even small performance improvement. Unfortunately, WiseTrans-v2 still has a probability (i.e., 20%) of worsening performance. This might be due to the unexpected network behaviors that not captured by `Probe Agent`.

4) Rollback Checker Deep Dive: We introduce the `Rollback Checker` to consider the unpredictable behaviours of network and unexpected protocol selection. Actually, `Rollback Checker` works in different situations in WiseTrans-v1 and WiseTrans-v2, and plays a more important role in WiseTrans-v1. In WiseTrans-v2, `Rollback Checker` captures the sudden network fluctuation in the exploration phase and avoids the wrong decision of `Probe Agent`. Due to the high frequency of requests and the effectiveness of the `Probe Agent`, as illustrated in §III-D, rollbacks have occurred only in rare cases. While `Rollback Checker` in WiseTrans-v1 is used to address both the spatial heterogeneity and the unexpected behaviour of the protocol classifier. Possible UDP/QUIC block or rate limit makes that the decision-making is to compare TCP and QUIC's performance under two different $< RTT, BtlBw, LossRate >$, as shown in Fig. 7, which cannot be learned by the classifier. Meanwhile, when a mobile user has only used one protocol before, which may occur when the first time a user visits a web service or the user changes ISP, it is challenging for the offline trained classifier to make a proper choice. In the meantime, when network conditions suddenly change, the sliding window in `Network Monitor` is challenged to capture and react to the instant changes. `Rollback Checker` efficiently captures the change and avoids severe performance degradation.

Therefore, we mainly evaluate the effectiveness of `Rollback Checker` in WiseTrans-v1. The probability of rollback after protocol switching is approximately 2.71% in BJ and 5.19% in SJZ. We speculate that the more frequent rollbacks in SJZ may be due to the fact that users may experience more in-network devices and complicated ISP strategies. Our evaluations did find that the bottlenecks of the TCP and QUIC links are not the same in some cases. Specifically, for some users in SJZ, TCP and QUIC connections have similar RTT and packet retransmission ratios, but with different goodput.

As for the effectiveness, `Rollback Checker` contributes 21.96% and 16.67% to the whole reduction of request completion time in two cities. As shown in Tab. IV, the selection accuracy drops by 5% and 9% when `Rollback Checker`

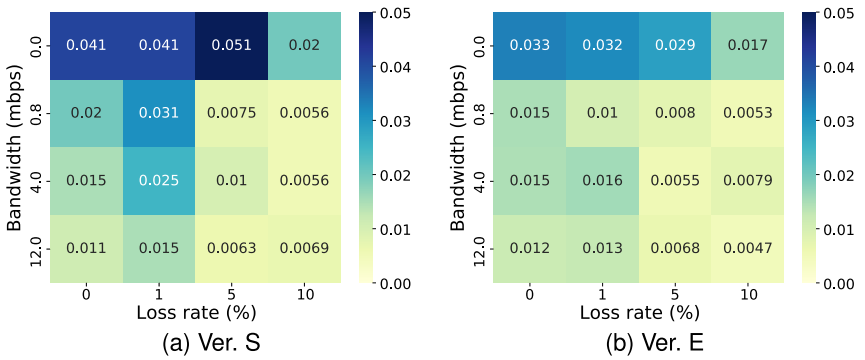


Fig. 15. Frequency of live experiments under different network conditions. In general, `Probe Agent` conducts live experiments with the frequency of about 1.76% in ver. S and 1.85% in ver. E.

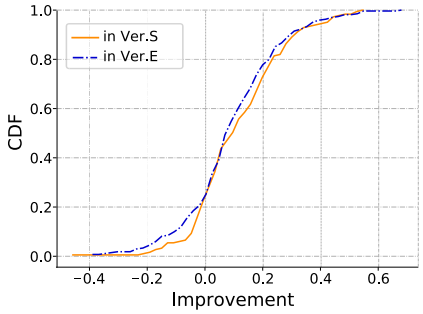


Fig. 16. Request completion time gap before and after protocol switching.

TABLE IV
MEAN ACCURACY OF WISETRANS WITH ROLLBACK CHECKER ENABLED OR DISABLED. ROLLBACK CHECKER IMPROVES THE ACCURACY OF CLASSIFICATION BY 5% AND 9%

. Dataset	w or w/o Rollback Checker	Mean Accuracy
BJ	w Rollback Checker	0.8829
	w/o Rollback Checker	0.8256
SJZ	w Rollback Checker	0.8719
	w/o Rollback Checker	0.7868

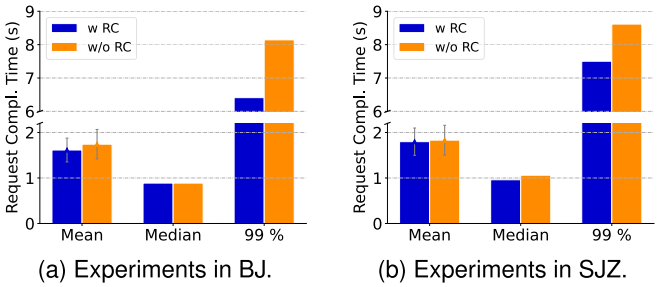


Fig. 17. `Rollback Checker` contributes about 20% to the reduction of completion time.

is disabled. Enabling `Rollback Checker` reduces the average request completion time by 7.3% and 2.0%, reduces the standard deviation request completion time by 19.12% and 8.01%, and shows great improvement for the 99^{th} percentile (the tail) completion time with a reduction of 21.28% and 12.92%, as shown in Fig. 17.

However, `Rollback Checker` may introduce additional performance degradation related to the time interval between the rollback and the protocol switch. This time interval can be set by the content providers, and in our experiments it was set to 10s. A longer time interval could improve the accuracy of selecting the optimal protocol, but may also spend a long time using the suboptimal protocol. In contrast, a shorter time interval may incorrectly switch back to the protocol. Therefore, we set the `Rollback Checker` as an optional module, and the content providers can choose whether to enable it to achieve their performance targets.

V. RELATED WORK

Performance measurement of TCP and QUIC. QUIC, a user-space transport protocol over UDP [5], has changed the landscape of web transport. Given its benefits over TCP, there has been tremendous effort to analyze and benchmark its performance both on testbed [6], [7], [11] and in production environments [5], [25], [35], [36], [37]. Extensive studies have concluded that neither QUIC nor TCP could achieve consistent high performance, and QUIC cannot completely replace TCP at present, which is the basis of our protocol selection method. However, the specific results of these measurements could not directly guide the protocol selection. Firstly, most early research focus on primitive gQUIC (Google's initial version), which is fundamentally different from the standardized QUIC [5], [7]. Meanwhile, some studies compares a highly tuned QUIC with an unoptimized TCP [6], [11]. It implies that those works are biased and are not applicable to a majority of today's QUIC [38]. Secondly, even if QUIC has been standardized [4], consistency between diverse implementations has not yet reached. Difference of performance measurements could be due to the optimization strategy adopted, specific design choices (e.g. the congestion control algorithms) and even implementation bugs, etc [16], [25], [39]. For the above factors, we perform an application-specific performance measurements and select protocol through live experiments.

Implementation diversity of QUIC. The standardization of QUIC has been accompanied by the maturity and widespread usage of numerous implementations [8]. One of the main challenges of QUIC is to ensure that any implementation follows the IETF specification [4]. Several methods have been proposed to verify QUIC implementations. The most common approach is interoperability testing, which manually generates test sets and tests the functionality of other implementations. This is done both manually and automatically in QUIC-Tracker [15] and QuicInteropRunner [40]. Another approach is to generate a mathematical model and use formal validation to automatically test functionality [41]. Also, a recent series of works combine the above two ideas [42], [43]. Meanwhile, research have analyzed different implementations,

such as advanced QUIC debugging and analysis techniques [16], [44], [45], and performance experiments between different implementations [25]. These works show difficulty to analyze the performance of TCP and QUIC under different implementations. Therefore, we performe live experiments to improve the generalization ability among different implementations.

Adaptive optimizations on transport parameters. There are few previous research efforts on adaptively adjusting the transport protocol online. The earliest related direction of WiseTrans is congestion control. Traditionally, researchers design different mechanisms to dynamically adjust the CWND and adapt to the network conditions [46], [47], [48]. Subsequently, recent researchers proposed to dynamically adjust the configurations of existing protocols [49], [50], [51], [52] to further enlarge the range of adaptation. However, as discussed in §II-B, due to the restriction in the network layer, adjusting the parameters of the transport layer is not enough in the large-scale deployment of mobile web service. In contrast, WiseTrans optimizes performance by adaptively switching the transport protocol between TCP and QUIC to address the challenges.

Machine learning for network prediction. Due to the complexity of temporal correlations in network conditions, existing methods have already employed machine learning techniques to network prediction. Examples range from decision trees [49], hidden Markov models [53], to neural networks [54], [55]. We refer the readers to [56] for a comprehensive understanding. The major difference between those research efforts and WiseTrans is that WiseTrans is discretized at the request level and considers not only the network conditions but also the previous decisions and consequences.

Enhancements to the earlier protocol selection work. The earlier version of this paper [1] first proposed a protocol selection mechanism, WiseTrans, and evaluated it in the real world. It is specialized for one of the mobile web services of `Baidu`, which is the WiseTrans-v1 in this paper. We have made substantive enhancements in this manuscript. First, We make a new observation that the implementation diversity affects the generalization and performance of the protocol selection mechanism. Therefore, we expand our design goals to strong generalization ability among different implementations. Then, we propose a new general protocol selection scheme, WiseTrans-v2. WiseTrans-v2 does not use a hardwired mapping of network conditions to the advised protocol [1] but discover the proper protocol by live experiments. Compared to WiseTrans-v1, this paper delivers an additional 5.15% improvement, as well as a consistent improvement of over 20% among different implementations. Finally, we updated a more comprehensive evaluation to give an in-depth analysis of design and guidance on the deployment options between WiseTrans-v1 and v2.

VI. Discussion

Integration with mechanisms of adaptive transport parameters. WiseTrans could be integrated with mechanisms of selecting among TCP variants [57], [58] or online configuring TCP parameters [51], [52]. Selecting among TCP variants is also a variant of TCP protocol [59]. Both WiseTrans-v1 and v2 could learn the performance of the implemented TCP variant and QUIC variant, and use the better protocol.

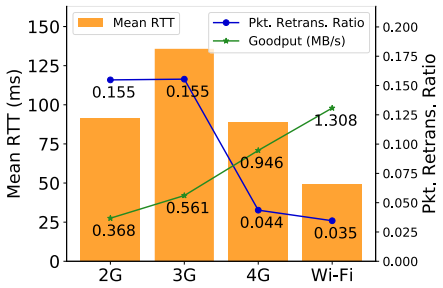


Fig. 18. Overall status of the measured request.

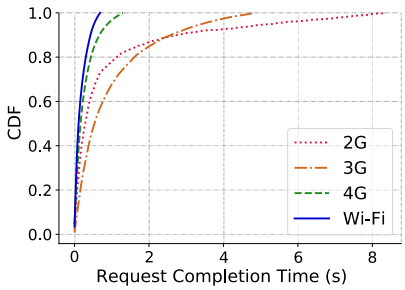


Fig. 19. Distribution of request completion time with different access network types.

The future of Internet transport protocol. Notwithstanding the rapid adoption of QUIC in recent years [5], [37], [39], past measurements suggest that TCP is not about to disappear or to be completely replaced anytime soon. This mean that using TCP and QUIC adaptively could be at least an incremental deployment and adopted by the industry.

Limitations in generalization analysis. Our evaluation only validates WiseTrans generalization ability among two similar implementations. Further analysis are required for generalization performance among widely varying implementations.

VII. Conclusion

In a large-scale real-world deployment of mobile web service, it is non-trivial to adaptively select transport protocols due to the huge complex temporal correlation of network conditions, spatial heterogeneity, implementation diversity, and limited computation resources on mobile clients. In response, we propose WiseTrans, the first solution that adaptively selects the transport protocols in an optimized way and is deployed in the real world. We introduce `Feature Extractor`, `Network Monitor`, `Probe Agent` and `Rollback Checker` in WiseTrans to address the challenges above. Our extensive evaluations with hundreds of thousands of requests demonstrate that WiseTrans could improve mobile web performance compared to a fixed protocol.

This work does not raise any ethical issues.

Appendix

A. Methodology

We conduct large-scale passive measurements of one popular mobile app of *Baidu*. Users can access the short video mobile web service of *Baidu* with this app. We collect and sample feed refresh request data from the users, which is sent by the app when a user wants to refresh the current list of recommended videos. On receiving a feed refresh request, the web servers of *Baidu* will return response data to the app, and the data amount is less than 50 KB, which is

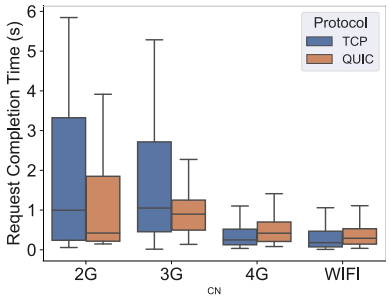


Fig. 20. Request completion time of users in China.

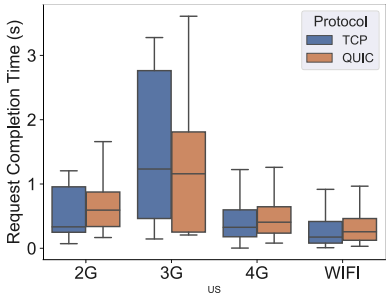


Fig. 21. Request completion time of users in US.

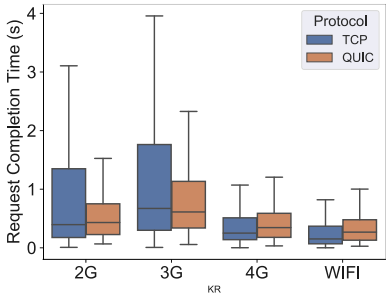


Fig. 22. Request completion time of users in South Korea.

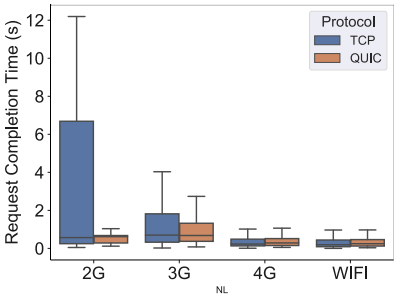


Fig. 23. Request completion time of users in Netherlands.

independent of the video size. After the request is completed, the instrumentation collects app-level logs and socket-level logs describing the finished transport process. The logs include a timestamp, data amount received, the completion time of the feed refresh request, RTT, number of retransmitted packets, geographical location of the user, access network type, etc. We sampled about 11.6 million requests during two weeks with a sampling ratio of 0.5‰, covering the users from more than 50 countries and regions.

The mobile web service supports both TCP and QUIC. In our measured data, 8.58% requests were using QUIC due to user choice, and 5% requests were due to our configurations, i.e., 13.58% in total. Note that QUIC is used only between the user app and the frontend server of *Baidu*. The frontend server acts as a load balancer and a proxy, which chooses a backend server and establishes a TCP connection. The HTTP response returned by the backend server is then returned to the user app, which completes the user request. Fig. 8 shows the transport model of our measurements. All requests access the backend resources through several frontend server clusters, geographically located in 6 cities.

B. Measurement Results

Fig. 18 shows the overall status of the measured requests. WiFi has the best condition, with small RTT, small packet retransmission ratio, and large goodput. 4G has better network conditions than 2G and 3G. 3G has a goodput better than 2G, a packet retransmission ratio similar to 2G, and the greatest average RTT among the four access network types. Fig. 19 shows the distribution of the request completion time (RCT) with each access type. It is not surprising that WiFi is the fastest and 4G is better than 3G and 2G. Recall that 3G networks have an RTT, which may be greater than that in 2G networks, and our request and response data have a small amount (tens of kilobytes). These facts may explain why 2G is faster than 3G in some cases.

We evaluate the RCT of TCP and QUIC in different access network types. The results are grouped by regions because users in different geographical locations may have different latency due to the distance to servers. Fig. 20 to Fig. 23 show the results in four typical countries. We can see that the RCT is short in 4G and WiFi for both TCP and QUIC, and TCP performs a little better in all four countries. The RCT is greater in 3G and 2G, and QUIC has better performance in general, except for 2G users in US. The results in different countries are not similar. For example, the RCT of 2G users in China is greater than other users, for both TCP and QUIC. In comparison, the RCT of 2G users in US and South Korea is less than their 3G users. This reflects the complicated network conditions in the real world.

Note that a user who uses TCP cannot use QUIC at the same time, the measurement results cannot tell whether a user should use TCP or QUIC for better performance directly. Instead, our passive measurement is like an A/B test, reflecting the probability that QUIC may outperform TCP with certain access network types in each area. Due to time-varying network conditions, the performance gap within one access network type is bigger than the gap between different types. For an individual user, it is difficult to ensure that a certain protocol will consistently achieve better performance under a certain access network/region. Thus, it is necessary for individual users to select a proper transport protocol according to specific network conditions.

REFERENCES

[1] J. Zhang et al., "WiseTrans: Adaptive transport protocol selection for mobile web service," in *Proc. WWW*, 2021, pp. 284–294.

[2] (2020). *Desktop vs Mobile vs Tablet Market Share Worldwide*. Accessed: May 5, 2020. [Online]. Available: https://gs.statcounter.com/platform-market-share/desktop-mobile-tablet

[3] (2017). *SPEED MATTERS. Designing for Mobile Performance*. [Online]. Available: https://www.awwwards.com/brainfood-mobile-performance-vol3.pdf

[4] J. Iyengar and M. Thomson, *QUIC: A UDP-Based Multiplexed and Secure Transport*, document RFC 9000, May 2021. [Online]. Available: https://rfc-editor.org/rfc/rfc9000.txt

[5] A. Langley et al., "The QUIC transport protocol: Design and internet-scale deployment," in *Proc. ACM SIGCOMM*, 2017, pp. 183–196.

[6] A. M. Kakhki, S. Jero, D. Choffnes, C. Nita-Rotaru, and A. Mislove, "Taking a long look at QUIC: An approach for rigorous evaluation of rapidly evolving transport protocols," in *Proc. ACM IMC*, Nov. 2017, pp. 290–303.

[7] Y. Yu, M. Xu, and Y. Yang, "When QUIC meets TCP: An experimental study," in *Proc. IEEE IPCCC*, Dec. 2017, pp. 1–8.

[8] (2021). *Active QUIC Implementations*. [Online]. Available: https://github.com/quicwg/base-drafts/wiki/Implementations

[9] T. Li et al., "Tack: Improving wireless transport performance by taming acknowledgments," in *Proc. ACM SIGCOMM*, 2020, pp. 15–30.

[10] Z. Meng, J. Chen, Y. Guo, C. Sun, H. Hu, and M. Xu, "PiTree: Practical implementation of ABR algorithms using decision trees," in *Proc. ACM Multimedia*, 2019, pp. 2431–2439.

[11] P. Biswal and O. Gnawali, "Does QUIC make the web faster?" in *Proc. IEEE GLOBECOM*, Dec. 2016, pp. 1–6.

[12] G. Carlucci, L. De Cicco, S. Holmer, and S. Mascolo, "Congestion control for web real-time communication," *IEEE/ACM Trans. Netw.*, vol. 25, no. 5, pp. 2629–2642, Oct. 2017.

[13] (2020). *Article: K07150625—Mitigating UDP Flood Attacks Using a Rate Limiting IRule*. [Online]. Available: https://support.f5.com/csp/article/K07150625

[14] (2020). *Looking for Measurements on UDP Blocking and Rate Limiting*. Accessed: May 5, 2020. [Online]. Available: http://www.postel.org/pipermail/end2end-interest/2011-March/008129.html

[15] M. Piraux, Q. De Coninck, and O. Bonaventure, "Observing the evolution of QUIC implementations," in *Proc. EPIQ*, Dec. 2018, pp. 8–14.

[16] R. Marx, J. Herbots, W. Lamotte, and P. Quax, "Same standards, different decisions: A study of QUIC and HTTP/3 implementation diversity," in *Proc. EPIQ*, Aug. 2020, pp. 14–20.

[17] K. Brown and S. Singh, "M-TCP: TCP for mobile cellular networks," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 27, no. 5, pp. 19–43, Oct. 1997.

[18] R. Netravali, A. Sivaraman, J. Mickens, and H. Balakrishnan, "WatchTower: Fast, secure mobile page loads using remote dependency resolution," in *Proc. ACM MobiSys*, Jun. 2019, pp. 430–443.

[19] T. Chen and C. Guestrin, "XGBoost: A scalable tree boosting system," in *Proc. ACM SIGKDD*, Aug. 2016, pp. 785–794.

[20] R. Baden, A. Bender, N. Spring, B. Bhattacharjee, and D. Starin, "Persona: An online social network with user-defined privacy," in *Proc. ACM SIGCOMM*, Aug. 2009, pp. 135–146.

[21] Z. Niu et al., "Network stack as a service in the cloud," in *Proc. ACM HotNets*, 2017, pp. 65–71.

[22] N. Cardwell, Y. Cheng, C. S. Gunn, S. H. Yeganeh, and V. Jacobson, "BBR: Congestion-based congestion control," *Queue*, vol. 14, no. 5, pp. 20–53, Oct. 2016.

[23] V. Arun and H. Balakrishnan, "Copa: Practical delay-based congestion control for the internet," in *Proc. Appl. Netw. Res. Workshop*, Jul. 2018, pp. 329–342.

[24] S. Dutton. (2020). *Understanding Low Bandwidth and High Latency | Web Fundamentals*. [Online]. Available: https://developers.google.com/web/fundamentals/performance/poor-connectivity

[25] A. Yu and T. A. Benson, "Dissecting performance of production QUIC," in *Proc. WWW*, Apr. 2021, pp. 1157–1168.

[26] Z. Meng, M. Wang, J. Bai, M. Xu, H. Mao, and H. Hu, "Interpreting deep learning-based networking systems," in *Proc. ACM SIGCOMM*, Jul. 2020, pp. 154–171.

[27] H. Chen, H. Zhang, S. Si, Y. Li, D. Boning, and C.-J. Hsieh, "Robustness verification of tree-based models," in *Proc. Adv. Neural Inf. Process. Syst.*, 2019, pp. 12317–12328.

[28] F. Pedregosa et al., "Scikit-learn: Machine learning in Python," *J. Mach. Learn. Res.*, vol. 12, pp. 2825–2830, Nov. 2011.

[29] C. C. Chang and C. J. Lin, "LIBSVM: A library for support vector machines," *ACM Trans. Intell. Syst. Technol.*, vol. 2, no. 3, pp. 1–27, 2011.

[30] L. Breiman, J. Friedman, C. Stone, and R. Olshen, *Classification and Regression Trees*. Boca Raton, FL, USA: CRC Press, 1984.

[31] L. Breiman, "Random forests," *Mach. Learn.*, vol. 45, no. 1, pp. 5–32, 2001.

[32] S.-C. Wang, "Artificial neural network," in *Interdisciplinary Computing in Java Programming*. Boston, MA, USA: Springer, 2003, pp. 81–100.

[33] L. Grinsztajn, E. Oyallon, and G. Varoquaux, "Why do tree-based models still outperform deep learning on tabular data?" 2022, *arXiv:2207.08815*.

[34] V. Borisov, T. Leemann, K. Seßler, J. Haug, M. Pawelczyk, and G. Kasneci, "Deep neural networks and tabular data: A survey," 2021, *arXiv:2110.01889*.

[35] S. Iyengar, "Moving fast at scale: Experience deploying IETF QUIC at Facebook," in *Proc. EPIQ*, Dec. 2018.

[36] S. Tellakula. (2010). *Comparing HTTP/3 vs. HTTP/2 Performance*. Accessed: Aug. 5, 2020. [Online]. Available: https://blog.cloudflare.com/http-3-vs-http-2/

[37] N. Banks. (2010). *QUIC Usage at Microsoft*. Accessed: Dec. 7, 2021. [Online]. Available: https://github.com/epiq21/epiq21.github.io/raw/main/slides/keynote1_microsoft.pdf

[38] K. Wolsing, J. Rüth, K. Wehrle, and O. Hohlfeld, "A performance perspective on web optimized protocol stacks: TCP+TLS+HTTP/2 vs. QUIC," in *Proc. Appl. Netw. Res. Workshop*, Jul. 2019.

[39] K. Oku and J. Iyengar. (2010). *Can QUIC Match TCP's Computational Efficiency?* Accessed: Aug. 4, 2020. [Online]. Available: https://www.fastly.com/blog/measuringquic-vs-tcp-computational-efficiency

[40] M. Seemann and J. Iyengar, "Automating QUIC interoperability testing," in *Proc. EPIQ*, Aug. 2020pp. 8–13.

[41] T. Ferreira, H. Brewton, L. D'Antoni, and A. Silva, "Prognosis: Closed-box analysis of network protocol implementations," in *Proc. ACM SIGCOMM*, Aug. 2021, pp. 762–774.

[42] K. L. McMillan and L. D. Zuck, "Formal specification and testing of QUIC," in *Proc. ACM SIGCOMM*, Aug. 2019, pp. 227–240.

[43] C. Crochet, T. Rousseaux, M. Piraux, J.-F. Sambon, and A. Legay, "Verifying QUIC implementations using ivy," in *Proc. EPIQ*, Dec. 2021, pp. 35–41.

[44] R. Marx, W. Lamotte, J. Reynders, K. Pittevils, and P. Quax, "Towards QUIC debuggability," in *Proc. EPIQ*, Dec. 2018, pp. 1–7.

[45] R. Marx, M. Piraux, P. Quax, and W. Lamotte, "Debugging modern web protocols with qlog," in *Proc. ANRW*, 2020, pp. 1–9.

[46] J. C. Hoe, "Improving the start-up behavior of a congestion control scheme for TCP," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 26, no. 4, pp. 270–280, 1996.

[47] S. Ha, I. Rhee, and L. Xu, "CUBIC: A new TCP-friendly high-speed TCP variant," *ACM SIGOPS Operating Syst. Rev.*, vol. 42, no. 5, pp. 64–74, Jul. 2008.

[48] D. X. Wei, C. Jin, S. H. Low, and S. Hegde, "Fast TCP: Motivation, architecture, algorithms, performance," *IEEE/ACM Trans. Netw.*, vol. 14, no. 6, pp. 1246–1259, Dec. 2006.

[49] K. Winstein and H. Balakrishnan, "TCP ex machina: Computer-generated congestion control," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 43, no. 4, pp. 123–134, Aug. 2013.

[50] A. Sivaraman, K. Winstein, P. Thaker, and H. Balakrishnan, "An experimental study of the learnability of congestion control," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 44, no. 4, pp. 479–490, Aug. 2014.

[51] M. Dong, Q. Li, D. Zarchy, P. B. Godfrey, and M. Schapira, "PCC: Re-architecting congestion control for consistent high performance," in *Proc. USENIX NSDI*, 2015, pp. 395–408.

[52] M. Dong et al., "PCC Vivace: Online-learning congestion control," in *Proc. USENIX NSDI*, 2018, pp. 343–356.

[53] A. R. Liu and R. R. Bitmead, "Observability and reconstructibility of hidden Markov models: Implications for control and network congestion control," in *Proc. IEEE CDC*, Dec. 2010, pp. 918–923.

[54] S. Abbasloo, C.-Y. Yen, and H. J. Chao, "Classic meets modern: A pragmatic learning-based congestion control for the internet," in *Proc. ACM SIGCOMM*, Jul. 2020, pp. 632–647.

[55] N. Jay, N. Rotman, B. Godfrey, M. Schapira, and A. Tamar, "A deep reinforcement learning perspective on internet congestion control," in *Proc. ICML*, 2019, pp. 3050–3059.

[56] L. Zhang, Y. Cui, M. Wang, Z. Yang, and Y. Jiang, "Machine learning for internet congestion control: Techniques and challenges," *IEEE Internet Comput.*, vol. 23, no. 5, pp. 59–64, Sep. 2019.

[57] K. Chen, D. Shan, X. Luo, T. Zhang, Y. Yang, and F. Ren, "One rein to rule them all: A framework for datacenter-to-user congestion control," in *Proc. APNet*, Aug. 2020, pp. 44–51.

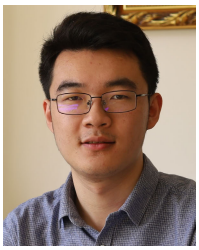
[58] Z. Du, J. Zheng, H. Yu, L. Kong, and G. Chen, "A unified congestion control framework for diverse application preferences and network conditions," in *Proc. ACM CoNEXT*, Dec. 2021, pp. 282–296.

[59] P. Goyal, A. Narayan, F. Cangialosi, S. Narayana, M. Alizadeh, and H. Balakrishnan, "Elasticity detection: A building block for internet congestion control," 2018, *arXiv:1802.08730*.

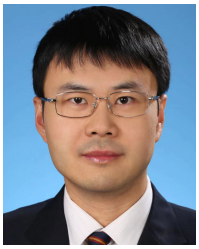
Jia Zhang received the B.Eng. degree from the Department of Electronic Engineering, Tsinghua University, in 2018. She is currently pursuing the Ph.D. degree with the Department of Computer Science and Technology, Tsinghua University. Her research interests include learning-based network systems and network transport optimization.

Shaorui Ren is currently pursuing the bachelor's degree with the Department of Electronic Engineering, Tsinghua University. His research interests include transport layer and computing methodologies.

Enhuan Dong (Member, IEEE) received the B.E. degree from the Harbin Institute of Technology, Harbin, China, in 2013, and the Ph.D. degree from Tsinghua University, Beijing, China, in 2019. He was a Visiting Ph.D. Student with the University of Goettingen from 2016 to 2017. He is currently an Assistant Research Professor with the Institute for Network Sciences and Cyberspace, Tsinghua University. His research interests include network security, network operations, and network transport.



Zili Meng (Graduate Student Member, IEEE) received the B.Eng. degree from the Department of Electronic Engineering, Tsinghua University, in 2019. He is currently pursuing the Ph.D. degree with the Institute for Network Science and Cyberspace, Tsinghua University. He has published papers in ACM SIGCOMM and USENIX NSDI. His research interests include learning-based networked systems and video streaming. He was a recipient of the Microsoft Research Asia Fellowship in 2020. He is also the Winner of the Student Research Competition in ACM SIGCOMM 2018 and several best paper awards.



Yuan Yang (Member, IEEE) received the B.Sc., M.Sc., and Ph.D. degrees from Tsinghua University. He was a Visiting Ph.D. Student with The Hong Kong Polytechnic University. He is currently an Assistant Researcher with the Department of Computer Science and Technology, Tsinghua University. His research interests include computer network architecture, routing protocol, and green networking.

Mingwei Xu (Senior Member, IEEE) is currently a Full Professor with the Department of Computer Science and Technology, Tsinghua University. He has chaired or participated in more than 30 research projects and published over 200 papers. His research interests include computer network architecture, Internet routing, and cybersecurity. He is the Winner of National Science Foundation for Distinguished Young Scholars of China. He has served as the TPC Chair or a member for several IEEE conferences, such as ICPP, Infocom, GLOBECOM, and ICC.

Sijie Yang received the B.Sc. and M.Sc. degrees from the Beijing University of Posts and Telecommunications in 2008 and 2011, respectively. He is leading the BFE Open Source Project with CNCF. He joined Tencent. His research interests include data center networking, network protocols, and learning-based networked systems.

Miao Zhang received the B.Sc. and Ph.D. degrees from Tsinghua University. He was an Assistant Researcher with the Network Center, Tsinghua University, until 2006. He is currently a Senior Software Engineer with Baidu Inc. He is also the Founder of BFE Open Source Project. BFE is a modern layer seven load balancer written with Go language and it is a CNCF Sandbox Project. His research interests include cloud network architecture.

Yang Yue received the B.S. degree from the Department of Computer Science and Technology, Tsinghua University, in 2022. He is currently pursuing the Ph.D. degree with the Department of Automation, Tsinghua University. His research interests include efficient deep learning and computer vision.