WiseTrans: Adaptive Transport Protocol Selection for Mobile Web Service

Jia Zhang[1,3], Enhuan Dong[1,3], Zili Meng[2,3], Yuan Yang[1,3], Mingwei Xu[1,2,3],
Sijie Yang[4], Miao Zhang[4], Yang Yue[1]
[1]Department of Computer Science and Technology & BNRist, Tsinghua University
[2]Institute for Network Sciences and Cyberspace, Tsinghua University
[3]Peng Cheng Laboratory (PCL) [4]Baidu Inc.
{jia-zhan18,mzl19,ley18}@mails.tsinghua.edu.cn,{dongenhuan,yangyuan_thu,xumw}@tsinghua.edu.cn
{yangsijie,zhangmiao02}@baidu.com

ABSTRACT

To improve the performance of mobile web service, a new transport protocol, QUIC, has been recently proposed. However, for large-scale real-world deployments, deciding whether and when to use QUIC in mobile web service is challenging. Complex temporal correlation of network conditions, high spatial heterogeneity of users in a nationwide deployment, and limited resources on mobile devices all affect the selection of transport protocols. In this paper, we present WiseTrans to adaptively switch transport protocols for mobile web service 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, and switches transport protocols at the request level to reach both high performance and 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 26.5% on average compared to the usage of a single protocol.

CCS CONCEPTS

• **Networks** → **Transport protocols**; *Mobile networks.*

KEYWORDS

Protocol Selection; Mobile Web Service; ML-based Networking Systems

ACM Reference Format:
Jia Zhang, Enhuan Dong, Zili Meng, Yuan Yang, Mingwei Xu, Sijie Yang, Yang Yue. 2021. WiseTrans: Adaptive Transport Protocol Selection for Mobile Web Service. In *Proceedings of the Web Conference 2021 (WWW '21), April 19–23, 2021, Ljubljana, Slovenia.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3442381.3449958

1 INTRODUCTION

Recently, a sharp increase in the usage of mobile web service has been observed. The latest statistics demonstrate that mobile users become the largest proportion of global Internet users [2]. In this case, request completion time[1] of mobile web service is becoming more and more critical for service providers. For example, Google reported that 53% of mobile users give up waiting if the page visited fails to be loaded within three seconds [30]. 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 service. As being standardized [8], QUIC has been becoming a new widely supported transport protocol in web service.

Yet, QUIC does not *always* perform better than TCP. Generally speaking, with proper designs on the handling of packet retransmissions, QUIC achieves better performance than TCP in the scenario of slow nets. On the other hand, TCP works better in networks with better conditions due to the quality of service (QoS) strategies from Internet Service Providers (ISPs) [22, 23, 33]. Moreover, in mobile networks, the network condition of the mobile users could be frequently changed. Therefore, using a fixed transport protocol for all requests will result in sub-optimal performance.

Therefore, we propose WiseTrans in this paper, an adaptive transport protocol selection mechanism to improve the performance of mobile web service. WiseTrans measures the network conditions from mobile users and switches the transport protocols when needed. By adaptively switching transport protocols, mobile users can therefore enjoy benefits from both protocols and have a higher experience when using mobile web service. A straightforward method to enable the idea is to measure the protocol performance in advance and construct a decision boundary for online protocol switching. For example, operators can emulate different bandwidths, delay, and packet loss rate and measure the performance of both protocols as in the online optimization of congestion control parameters [29, 32].

However, in a large-scale real-world deployment of mobile web service, it is non-trivial to decide whether a request in mobile web service should use TCP or QUIC. First, the temporal heterogeneity of network conditions during the connection makes the problem more challenging. 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 operators to precisely measure the network conditions in real time. Second, due to the spatial heterogeneity of mobile users, the

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

WWW '21, April 19–23, 2021, Ljubljana, Slovenia

ACM ISBN 978-1-4503-8312-7/21/04.
https://doi.org/10.1145/3442381.3449958

web service provider needs to tackle different behaviors on wide-area networks and mobile access networks. For example, different QoS preferences of ISPs and home routers could all affect the optimal choice between two protocols [23]. In this case, different users might have different decision boundaries. Third, deciding the proper granularity of protocol switching is also challenging. Fine granularity of protocol switching (e.g., packet-level) will result in burdensome overhead to collect statistics from mobile clients and maintain consistency during switching. Switching the protocol in a sparse granularity (e.g., user-level and page-level) would degrade the performance due to the temporal heterogeneity of mobile users.

To address the above challenges, we design several building blocks in WiseTrans. First, we observe that the spatial heterogeneity of users could be learned by the combination of history network conditions, history decisions, and the performance of that decision. Therefore, we collect not only historical network conditions, but also the decisions and their consequent performance for those decisions (§3.2). Second, to handle the complex temporal relationship during a connection while ensuring the simplicity for large-scale deployment, we employ a tree-based classification algorithm, XGBoost, to optimize the protocol decisions (§3.3). Moreover, we utilize features of concurrent requests in HTTP page loading and decide to optimize the protocol selection at the granularity of web requests. Due to the independence of each request in web service, the timeliness of web requests, and the computation ability of mobile devices, switching the protocol at the request level can achieve both high performance and satisfactory overhead (§3.1).

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 26.5% on average compared to using a single protocol. With XGBoost, WiseTrans can also achieve an accuracy of 88.3% of protocol selection for each request.

In summary, we make the following contributions in this paper:

- We motivate the problem and significance of transport protocol selection with measurements from real-world mobile web service (§2).
- We propose WiseTrans, an adaptive transport protocol selection mechanism based on XGBoost for large-scale deployment of mobile web service (§3).
- We evaluate the performance improvements of WiseTrans against several baselines with extensive experiments with one of the mobile web service of *Baidu* (§4).

2 BACKGROUND AND MOTIVATION

In this section, we first introduce the performance differences between QUIC and TCP for mobile web service in §2.1. We then present the challenges of the adaptive protocol switching for mobile web service in §2.2.

2.1 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 one mobile web service with millions of

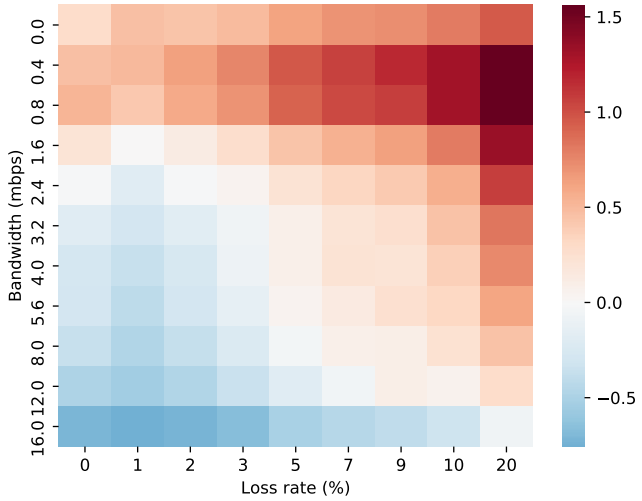


Figure 1: $\ln \lambda$ with different bottleneck bandwidth and packet loss rate. Red color indicates QUIC outperforms TCP and blue color indicates that TCP outperforms QUIC. λ is the ratio of the performance of TCP and QUIC. The performance here is the request completion time at the application layer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

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 A 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 outperforms 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 [9, 22, 23, 33].

Computation resources. Meanwhile, for mobile web service, the performance of QUIC is also affected by the computation resource on mobile users. Due to the high CPU overhead (up to 3.5× higher compared to TCP/TLS [23]), mobile devices with limited computation resources might experience performance degradation [22].

2.2 Design Challenge

However, as we discussed in §1, deciding which protocol to use is non-trivial in a large-scale real-world deployment of mobile web service. We have encountered the following major challenges:

Complex temporal correlation. Network condition fluctuates due to wireless channel fading, user moving, or network congestion. Therefore, if we categorize the best protocol for mobile users

according to Fig. 1, the optimal protocol might also change. However, precisely measuring the current network condition and capturing the temporal relationship is 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 [14]). 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 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, which will consequently affect 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 [1, 3]. Since different ISPs and home routers might have different preferences, it is challenging for us to optimize the protocol selection for individual users.

Limited resource on mobile devices. Deciding the proper operating granularity of a protocol switching algorithm is also challenging for mobile devices. On one hand, packet-level measurements might result in burdensome overhead by querying the statistics of network stack [12]. 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 [26].

Therefore, we are motivated to design an adaptive transport protocol switching mechanism for mobile web service with the above challenges in mind.

3 WISETRANS DESIGN

In this section, we present the design of WiseTrans. We first present an overview (§3.1), and then introduce each design components of WiseTrans, including a `Feature Extractor` (§3.2), a `Protocol Classifier` (§3.3), and a `Rollback Checker` (§3.4).

3.1 Design Overview

As shown in Fig. 2, 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 §3.2.

Step 2: Protocol classification. Next, WiseTrans employs an XGBoost-based [17] `Protocol Classifier` (§3.3). The `Protocol Classifier` selects a better transport protocol based on the features from the `Feature Extractor`.

Step 3: Rollback check. After that, before putting the protocol selection into effect, WiseTrans checks if the user needs to rollback the selection with a `Rollback Checker` (§3.4). The rollback mechanism is designed to correct unexpected behaviors caused by the preferences of network devices or the decisions of the `Protocol Classifier`.

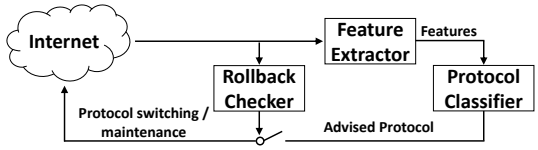


Figure 2: The architecture of WiseTrans.

Step 4: Protocol switching. Finally, based on the output of the `Protocol Classifier` and `Rollback Checker`, WiseTrans decides to still use the current transport protocol or switch to the other one for the next web service request.

However, as we discussed before, a key design choice is the operation granularity of WiseTrans. Compared to the user-level classification in personal characterization of social networks [6] and packet-level decisions in network bandwidth prediction in the network stack [14], WiseTrans works on the request level due to the following considerations:

- **Overhead on mobile devices.** For mobile web service, the resources on mobile devices are limited. Therefore, packet-level measurements at the protocol stack might result in burdensome overhead by frequently querying the statistics of network stack [27]. Moreover, packet-level decisions usually need to modify the protocol stack [13], 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 and are usually around hundreds of milliseconds or even longer [5]. As shown in Fig. 3, 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 [18]. 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, packets 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.

3.2 Feature Extractor

To address the spatial heterogeneity, we extract features not only representing historical network conditions, but also historical decisions and their consequent performance for those decisions. The decision boundary may be different for different mobile users and can be changeable for an individual user due to spatial heterogeneity. For example, since ISP gateways or 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

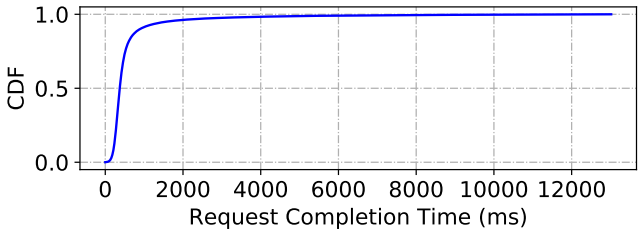


Figure 3: The distribution of the request completion time of our mobile web service application in the wild. The measurement details are introduced in Appendix A.

conditions cannot optimize protocol selection for individual users. In another scenario, the access network and ISP changes 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, as shown in Tab. 1

- **Historical network conditions.** We use *RTT* (round-trip time), *BtlBw* (bottleneck bandwidth) and *LossRate* (packet loss rate) to model a network path, following [13], which can describe the network conditions a user experienced. Based on the design choice mentioned in §3.1, we collect request-level average RTT (*RTT_avg*), throughput (*req_cmpl_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 from the similar network conditions and the same protocol, but different performance. In the meantime, the changes of decision boundary will be explored by the unchanged network conditions and protocol, but changing performance.
- **Historical requests' performance.** We select *TTFB* and *resp_rec_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, implies the properties of the request transport. *req_cmpl_G* measures request completion goodput, dividing the user's received bytes by the time interval between the client sending request, and receiving the whole response body.

As shown in Fig. 4, for each request, `Feature Extractor` records the required information of the request, as well as the RTT, retransmitted packets, and TTFB in the Record Window from the HTTP logs and the socket logs. Finally, all the features extracted from the requests in the Recog Window are passed to the `Protocol Classifier`, as shown in Fig. 5.

3.3 Protocol Classifier

As introduced before, the design goal of `Protocol Classifier` is to faithfully learn the temporal correlation and spatial heterogeneity based on the features from `Feature Extractor`. Potential classification models include linear regression, supporting vector machine, decision tree, random forests, and even neural networks.

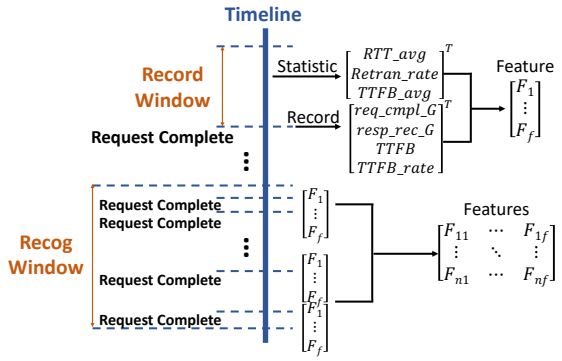


Figure 4: The detail process of `Feature Extractor`. `Feature Extractor` records and extracts features used for classification.

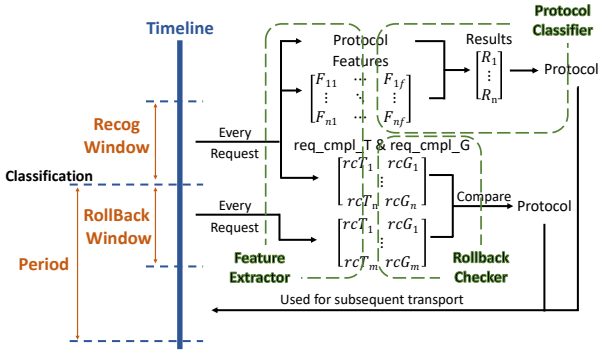


Figure 5: The detail process of `Protocol Classifier` and `Rollback Checker`.

In this paper, WiseTrans employs XGBoost, a tree-based classification model, due to the following reasons:

- **Expressive.** Due to the complexity of features, the classification algorithm should be expressive enough to capture the relationship among features. In our experiments in §4.3, XGBoost is capable of precisely capturing the relationship and has a satisfactory classification accuracy of around 90%.
- **Lightweight.** For mobile web service, due to the resource limitation on mobile devices, the classification 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 §4.2).
- **Interpretable and verifiable.** Meanwhile, deploying the classifier online in a production environment also requires the model to be interpretable and verifiable [25]. 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 [16].

As for the training of XGBoost, WiseTrans learns the classification rules offline. WiseTrans discovers the rules through a large number of fine-grained experiments, as shown in Fig. 6. We label the features from `Feature Extractor` according to the performance comparison of using TCP and QUIC. Therefore, WiseTrans can

Feature	Description	TCP		QUIC	
		Importance	Rank	Importance	Rank
Retran_rate	|Retransmitted Packets| / |Sent Packets| 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 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

Table 1: The description of the selected features and their ranking. The features selected by WiseTrans not only reflect historical network conditions, but also historical decisions and their performance.

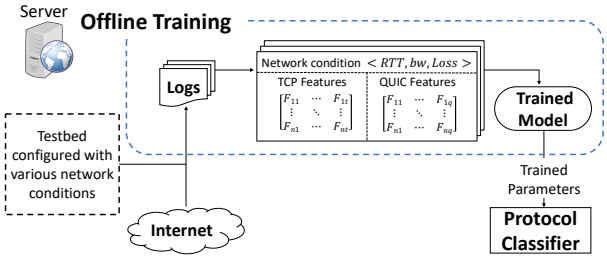


Figure 6: The Process of WiseTrans Offline Training. A server collects information of requests under various network conditions, using TCP and QUIC separately, and labels automatically.

learn the classification rules based on the extracted features (listed in Tab. 1) 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 §4.5.

3.4 Rollback Checker

We introduce `Rollback Checker` as a guard for unexpected behaviors of network and unexpected decisions of `Protocol Classifier`. On one hand, 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 compared by the `Protocol Classifier`. On the other hand, when a mobile user has only used one protocol before, which may occur 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 `Protocol Classifier` is challenged to capture and react to the instant changes. `Rollback Checker` efficiently captures the change and avoids severe performance degradation.

Therefore, in view of such situations, `Rollback Checker` compares the performance of the requests in the RollBack Window before and after the switching, shown in Fig. 5. If a significant performance deterioration is found, which means the user is in a *rollback state*, `Rollback Checker` rolls back to the last protocol.

4 EVALUATION

In this section, we first introduce the setup of our evaluation in §4.1. We then evaluate WiseTrans in the following aspects:

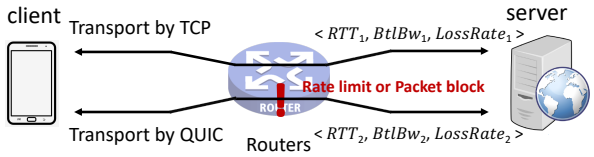


Figure 7: Transport model when meeting QUIC/UDP rate limit or packet block.

- *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 26.5% compared to using TCP only and 6.6% compared to QUIC (§4.2).
- *Component Effectiveness.* We then evaluate the effectiveness of the design of WiseTrans. Compared to other classification method, WiseTrans `Protocol Classifier` can achieve a classification accuracy of 91% (§4.3). Meanwhile, the `Rollback Checker` can also reduce the average completion time by 7.3% (§4.4).
- *Parameter Sensitivity.* Finally, we evaluate the performance of WiseTrans by varying the hyperparameters of WiseTrans. Experiments show that the performance improvements of WiseTrans can be maintained for a wide range of parameters (§4.5).

4.1 Experimental Setup

4.1.1 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.

For the implementation of XGBoost in WiseTrans `Protocol Classifier` in §3.3, 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.

To avoid switching too frequently, in our online implementation, we use a periodic decision-making method as shown in Fig. 5. We further discuss the time window parameter setting in §4.5.

The training traces and source code are published on https://github.com/joycezhangjia/WiseTrans.

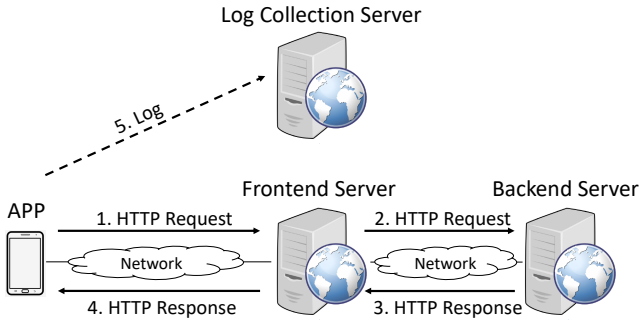


Figure 8: The experimental setup of WiseTrans.

4.1.2 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 connection 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.

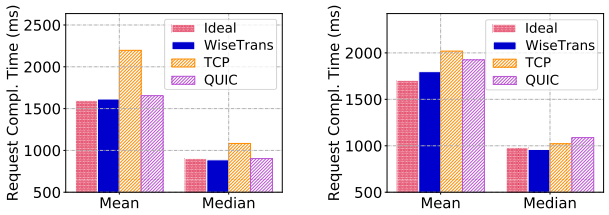
4.1.3 Dataset. For offline training, we collect 38 hours, more than 128,000 request logs on a mobile web service app of *Baidu*, as the training dataset. For evaluation, we enable WiseTrans for certain users in two cities (Beijing and Shijiazhuang in China), with and without CDN server cluster deployment, and collect logs for 40 hours, resulting in more than 64,000 request logs. Also, we randomly collect more than 64,000 request logs using TCP and QUIC as our baseline.

4.2 Request Completion Time

Request completion time is a key metric to measure the performance of web service [7]. Fig. 9 shows the average request completion time, with the median of it. We also select the shorter average completion time between TCP and QUIC as Ideal for each network condition, as a comparison. It can be found that WiseTrans outperforms TCP and QUIC. For the situation with a client located in Beijing, WiseTrans has a 26.5% reduction in average completion time compared to TCP, and 2.4% to QUIC. For the median completion time, WiseTrans also has an 18.9% and 1.6% reduction. As for the experiments which the client is located in Shijiazhuang, the reduction is about 11.0% and 6.6% on average and 6.0% and 11.7% for the median completion time. In all, WiseTrans is just about 1%-2% longer than Ideal, which demonstrates that WiseTrans is efficient.

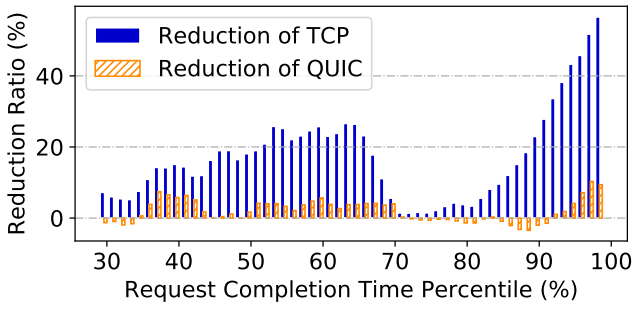
Fig. 10 shows the detailed reduction ratio of WiseTrans compared to TCP and QUIC as the request completion time increases. Obviously, WiseTrans shows great improvement compared to TCP and QUIC for the 99^{th} percentile (the tail) completion time. WiseTrans has a 57.1% reduction, about 8.52 s compared to TCP, and an 8.69% reduction compared to QUIC for the experiments in Beijing. For the experiments in Shijiazhuang, the reduction is about 37.6% and 12.8%. Results show that WiseTrans effectively improve 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,

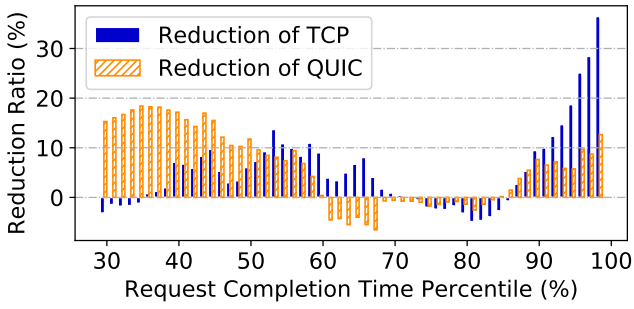


(a) Experiments in Beijing. **(b)** Experiments in Shijiazhuang.

Figure 9: WiseTrans achieves lower request completion time than TCP and QUIC in both cities. The reduction could achieve 26.5% compared to using one fixed transport protocol.



(a) Experiments in Beijing.



(b) Experiments in Shijiazhuang.

Figure 10: WiseTrans's redution of TCP and QUIC with different percentile of request comlpetion time. WiseTrans has a reduction of 57.1% for the 99^{th} percentile (the tail) completion time.

QUIC has poor performance, and WiseTrans has significant improvements, especially in Shijiazhuang. 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 always be able to achieve the optimal performance of using QUIC or TCP. Therefore, WiseTrans just will have better performance than TCP or QUIC for one network condition, while can achieve the overall optimal for users across the network.

We also find negative value with an average ratio of -1.29% for experiments in Beijing and -2.37% for experiments in Shijiazhuang. The performance degradation among them is mainly due to the

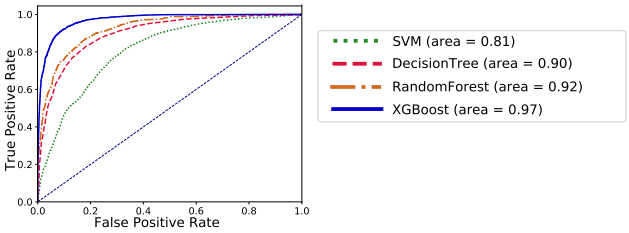
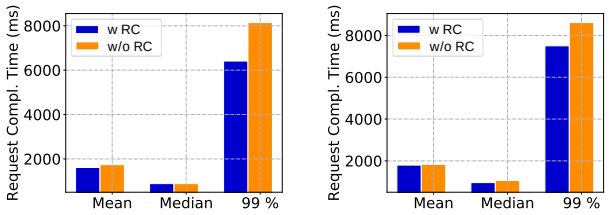


Figure 11: ROC curves used for algorithms comparison. XGBoost achieves the best performance.



(a) Experiments in Beijing. **(b)** Experiments in Shijiazhuang.

Figure 12: `Rollback Checker` contributes about 20% to the reduction of completion time.

inaccurate classification sometime during the classification and the transport in RollBack Window, leading to the suboptimal protocol.

Overhead of WiseTrans. The additional decision-making time consumed by WiseTrans is less than 3 ms in our experiment, which is negligible compared to the decision period. `Protocol Classifier` is the most time-consuming part, related to the value of Recog Window. For each classification, the model takes about 0.02 ms~0.04 ms. Feature Extractor takes about one-tenth of the overall process time, and is the second time-consuming part.

As for the switch operation, we switch the transport protocol of a request from the application layer. WiseTrans preferably reuses existing connections to reduce switching overhead. In our experiments, 89.73% of switches can reuse existing connections and do not need to establish new connections, which brings negligible overhead in general.

4.3 Classifier Deep Dive

XGBoost is utilized as a classification algorithm in consideration of the complex temporal correlation as well as the simplicity for large-scale deployment. We compare XGBoost with a set of common machine learning algorithms (SVM (Support Vector Machines) [15], Decision Tree [11] and Random Forest [10]) to explore whether XGBoost can address the complex temporal heterogeneity.

Tab. 2 and the ROC curves shown in Fig. 11 show that XGBoost achieves the best performance. SVM only achieves an accuracy of 76.87%, which means the decision boundary can hardly be found by simply employing hyperplanes in three-dimensional space $< RTT, BtlBw, LossRate >$. Meanwhile, the simple tree model, Decision Tree, can not describe the complex relationship between the historical network conditions, decisions, and performance as well. XGBoost outperforms Random Forest by its faster convergence and higher accuracy.

The accuracy of WiseTrans's classification is listed in Tab. 3. 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 Beijing and 87.19% in Shijiazhuang, which is basically consistent with the performance of offline testing. It proves that our classifier can also achieve high accuracy in the real network.

4.4 Rollback Checker Deep Dive

The `Rollback Checker` is introduced in consideration of the possible rate limit on QUIC or UDP packets in the network. [23] found that 0.3% of users encountered such a situation. Our evaluation also finds that the bottlenecks of the TCP and QUIC links are not the same. Specifically, for some users, TCP and QUIC have similar RTT and packet retransmission ratio, but with different goodput.

`Rollback Checker` contributes 21.96% and 16.67% to the whole reduction of request completion time in two cities. As shown in Tab. 4, the classification accuracy drops by 5% and 9% when `Rollback Checker` is disabled. Enabling `Rollback Checker` reduces the average request completion time by 7.3% and 2.0%, 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. 12.

However, `Rollback Checker` may also bring additional performance degradation, related to RollBack Window's value. A longer RollBack Window could improve the accuracy of selecting the optimal protocol, but also spend a longer time using the suboptimal protocol. In contrast, a shorter RollBack Window may switch back to the protocol incorrectly.

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.

4.5 Parameters Analysis

4.5.1 Hyperparameters of Classifier. We perform hyperparameters tuning to achieve the best performance. We mainly consider the maximum depth and minimum child weight in `Protocol Classifier`. Fig. 13 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 classifiers.

4.5.2 Time Windows. We analyze Period and Record Window's impact in Fig. 4 and Fig. 5 on classification, mainly relevant to the classification speed and accuracy when the network changes. Record Window is the time window used in calculating the average RTT (RTT_rate), TTFB ($TTFB$) and Retransmission rate($Retran\_rate$). For the purpose of avoiding frequent switch, in our implementation, the protocol selection is a process independent of sending the requests, and WiseTrans makes the selection every Period. In our evaluation, Record Window is set to 20 s, 30 s, while Period values are set to 1 s, 3 s, and 5 s, respectively. We set the Recog Window exactly the same as the Period and the `Rollback Checker` is disabled. By manually adding packet loss on the client side, we

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**

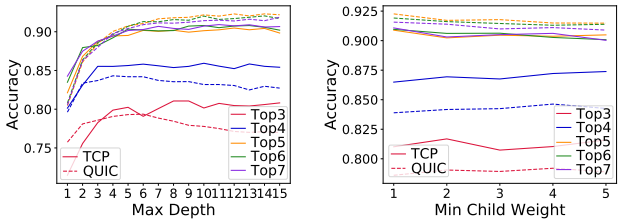
Table 2: The comparison of machine learning algorithm candidates. XGBoost achieves the best performance.

Dataset	Offline Test Set	Beijing	Shijiazhuang
Accuracy	0.9034	0.8829 (-2.26%)	0.8719 (-3.48%)

Table 3: The accuracy of classification.

Dataset	w or w/o Rollback Checker	Mean Accuracy
Beijing	w Rollback Checker	0.8829
	w/o Rollback Checker	0.8256
Shijiazhuang	w Rollback Checker	0.8719
	w/o Rollback Checker	0.7868

Table 4: The mean accuracy of WiseTrans with Rollback Checker enabled or disabled. `Rollback Checker` improves the accuracy of classification by 5% and 9%.



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

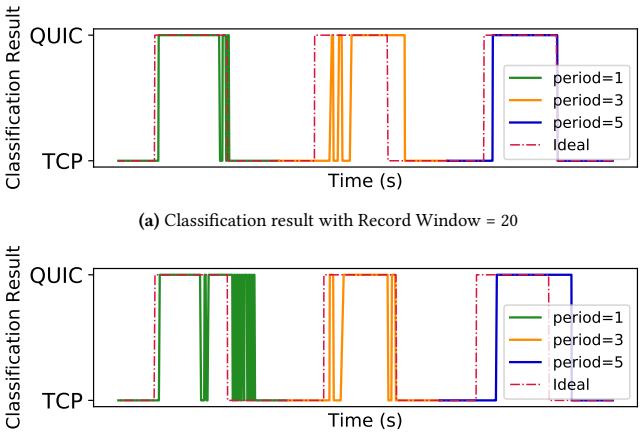
Figure 13: Grid search results (mean accuracy) of XGBoost for the optimal model and features.

change the network status and observe WiseTrans's performance, as shown in Fig. 14. The red dotted line reflects the changes of network conditions over time, which means protocol selection's ideal performance. The solid lines show the real classification results of WiseTrans under six different parameters.

Both Period and Record Windows would affect the classification accuracy, and it's just a trade-off between sensitivity and stability. When the Period is short, the amount of features used for classification is not enough, affecting the accuracy. Therefore, as shown by the green line and yellow line in Fig. 14, there are jumps and instability. Similarly, for the Record Window, statistics of RTT and other features within a short period can reflect changes relatively quickly, but may be affected by outliers and thus reduce accuracy. However, the classification sensitivity to changes decreases with long windows, as shown by the blue line in Fig. 14b. Longer windows ensure stability but reduce sensitivity.

5 RELATED WORK

Adaptive optimizations on transport parameters. There are few previous research efforts on adaptively adjusting the transport protocol online. The earliest related direction of WiseTrans



(a) Classification result with Record Window = 20

(b) Classification result with Record Window = 30

Figure 14: Sensitivity and stability of WiseTrans classification with different Period and Record Windows.

is congestion control algorithms. Traditionally, researchers design different mechanisms to dynamically adjust the congestion window and adapt to the network conditions [19, 20, 31]. Subsequently, recent researchers proposed to dynamically adjust the configurations of existing protocols [29, 32] to further enlarge the range of adaptation. However, as we discussed in §2.2, 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 in network prediction. Examples ranges from decision trees [32], hidden Markov models [24], to neural networks [4, 21]. We refer the readers to [34] 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.

6 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 spatial heterogeneity of users, complex temporal correlation of network conditions, 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`, `Protocol Classifier`, 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 transport protocol.

This work does not raise any ethical issues.

ACKNOWLEDGEMENT

We thank the reviewers and the shepherd for their valuable comments. The co-authors, Dong, Yang, and Xu, from Tsinghua University, are supported by the National Key R&D Program of China under Grant (2019YFB1802504), the National Natural Science Foundation of China under Grant (62002192, 61625203, 61832013, and 61872209), and the Independent Scientific Research Project of NUDT (ZZKYZX-03-02-02). Prof. Mingwei Xu and Dr. Yuan Yang are the corresponding authors.

REFERENCES

[1] 2020. Article: K07150625 - Mitigating UDP flood attacks using a rate limiting iRule. https://support.f5.com/csp/article/K07150625.
[2] 2020. Desktop vs Mobile vs Tablet Market Share Worldwide. https://gs.statcounter.com/platform-market-share/desktop-mobile-tablet. Accessed: 2020-05-05.
[3] 2020. Looking for measurements on UDP blocking and rate limiting. http://www.postel.org/pipermail/end2end-interest/2011-March/008129.html. Accessed: 2020-05-05.
[4] Soheil Abbasloo, Chen-Yu Yen, and H Jonathan Chao. 2020. Classic meets modern: a pragmatic learning-based congestion control for the internet. In *Proceedings of the Annual conference of the ACM Special Interest Group on Data Communication on the applications, technologies, architectures, and protocols for computer communication*. 632–647.
[5] Venkat Arun and Hari Balakrishnan. 2018. Copa: Practical delay-based congestion control for the internet. In *Proc. USENIX NSDI*. 329–342.
[6] Randy Baden, Adam Bender, Neil Spring, Bobby Bhattacharjee, and Daniel Starin. 2009. Persona: an online social network with user-defined privacy. In *Proc. ACM SIGCOMM*. 135–146.
[7] Nina Bhatti and Rich Friedrich. 1999. Web server support for tiered services. *IEEE network* 13, 5 (1999), 64–71.
[8] Mike Bishop et al. 2020. Hypertext transfer protocol version 3 (HTTP/3). *Internet Engineering Task Force, Internet-Draft draft-ietf-quic-http-31* (2020).
[9] P. Biswal and O. Gnawali. 2016. Does QUIC Make the Web Faster?. In *IEEE GLOBECOM*. IEEE, 1–6.
[10] L. Breiman. 2001. Random Forests. *Machine Learning* 45, 1 (2001), 5–32.
[11] L. Breiman, J. Friedman, C. Stone, and R. Olshen. 1984. *Classification and Regression Trees*. CRC press.
[12] Kevin Brown and Suresh Singh. 1997. M-TCP: TCP for mobile cellular networks. *ACM SIGCOMM Computer Communication Review* 27, 5 (1997), 19–43.
[13] N. Cardwell, Y. Cheng, C. S. Gunn, S. Yeganeh, and V. Jacobson. 2016. BBR: Congestion-based Congestion Control. *ACM Queue* (2016).
[14] Gaetano Carlucci, Luca De Cicco, Stefan Holmer, and Saverio Mascolo. 2017. Congestion control for web real-time communication. *IEEE/ACM Transactions on Networking* 25, 5 (2017), 2629–2642.
[15] C. Chang and C. Lin. 2011. LIBSVM: A Library for Support Vector Machines. *ACM TIST* 2, 3 (2011), 1–27.
[16] Hongge Chen, Huan Zhang, Si Si, Yang Li, Duane Boning, and Cho-Jui Hsieh. 2019. Robustness verification of tree-based models. In *Advances in Neural Information Processing Systems*. 12317–12328.
[17] T. Chen and C. Guestrin. 2016. Xgboost: A Scalable Tree Boosting System. In *Proc. ACM SIGKDD*. 785–794.
[18] Sam Dutton. 2020. Understanding Low Bandwidth and High Latency | Web Fundamentals. https://developers.google.com/web/fundamentals/performance/poor-connectivity.
[19] Sangtae Ha, Injong Rhee, and Lisong Xu. 2008. CUBIC: a new TCP-friendly high-speed TCP variant. *ACM SIGOPS operating systems review* 42, 5 (2008), 64–74.
[20] Janey C Hoe. 1996. Improving the start-up behavior of a congestion control scheme for TCP. *ACM SIGCOMM Computer Communication Review* 26, 4 (1996), 270–280.
[21] Nathan Jay, Noga Rotman, Brighten Godfrey, Michael Schapira, and Aviv Tamar. 2019. A deep reinforcement learning perspective on internet congestion control. In *International Conference on Machine Learning*. 3050–3059.
[22] A. Kakhki, S. Jero, D. Choffnes, C. Nita-Rotaru, and A. Mislove. 2017. Taking a Long Look at QUIC: an Approach for Rigorous Evaluation of Rapidly Evolving Transport Protocols. In *Proc. ACM IMC*. 290–303.
[23] A. Langley, A. Riddoch, A. Wilk, A. Vicente, C. Krasic, D. Zhang, F. Yang, F. Kouranov, I. Swett, J. Iyengar, et al. 2017. The QUIC Transport Protocol: Design and Internet-scale Deployment. In *ACM SIGCOMM*. 183–196.
[24] Andrew R Liu and Robert R Bitmead. 2010. Observability and reconstructibility of hidden Markov models: Implications for control and network congestion control. In *49th IEEE Conference on Decision and Control (CDC)*. IEEE, 918–923.
[25] Zili Meng, Minhu Wang, Jiasong Bai, Mingwei Xu, Hongzi Mao, and Hongxin Hu. 2020. Interpreting Deep Learning-Based Networking Systems. In *Proc. ACM SIGCOMM*. 154–171.
[26] Ravi Netravali, Anirudh Sivaraman, James Mickens, and Hari Balakrishnan. 2019. Watchtower: Fast, secure mobile page loads using remote dependency resolution. In *Proceedings of the 17th Annual International Conference on Mobile Systems, Applications, and Services*. 430–443.
[27] Zhixiong Niu, Hong Xu, Dongsu Han, Peng Cheng, Yongqiang Xiong, Guo Chen, and Keith Winstein. 2017. Network Stack as a Service in the Cloud. In *Proceedings of the 16th ACM Workshop on Hot Topics in Networks*. 65–71.
[28] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, et al. 2011. Scikit-learn: Machine learning in Python. *the Journal of Machine Learning Research* 12 (2011), 2825–2830.
[29] Anirudh Sivaraman, Keith Winstein, Pratiksha Thaker, and Hari Balakrishnan. 2014. An experimental study of the learnability of congestion control. *ACM SIGCOMM Computer Communication Review* 44, 4 (2014), 479–490.
[30] Speed Matters 2017. SPEED MATTERS. Designing for Mobile Performance. https://www.awwwards.com/brainfood-mobile-performance-vol3.pdf.
[31] David X Wei, Cheng Jin, Steven H Low, and Sanjay Hegde. 2006. FAST TCP: motivation, architecture, algorithms, performance. *IEEE/ACM transactions on Networking* 14, 6 (2006), 1246–1259.
[32] Keith Winstein and Hari Balakrishnan. 2013. Tcp ex machina: Computer-generated congestion control. *ACM SIGCOMM Computer Communication Review* 43, 4 (2013), 123–134.
[33] Yajun Yu, Mingwei Xu, and Yuan Yang. 2017. When QUIC meets TCP: An experimental study. In *2017 IEEE 36th International Performance Computing and Communications Conference (IPCCC)*. IEEE, 1–8.
[34] Lei Zhang, Yong Cui, Mowei Wang, Zhenjie Yang, and Yong Jiang. 2019. Machine learning for internet congestion control: Techniques and challenges. *IEEE Internet Computing* 23, 5 (2019), 59–64.

APPENDICES

A LARGE-SCALE MEASUREMENTS OF ONE POPULAR MOBILE WEB SERVICE

A.1 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 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 timestamp, data amount received, 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. A user can choose to use one of the transport protocols in the app by himself or herself. We also configure a small percentage of the requests to use QUIC, so as to collect more samples of QUIC. In our measured

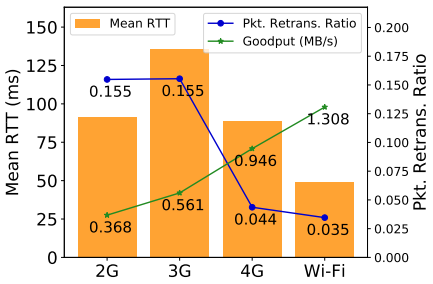


Figure 15: Overall status of measured request.

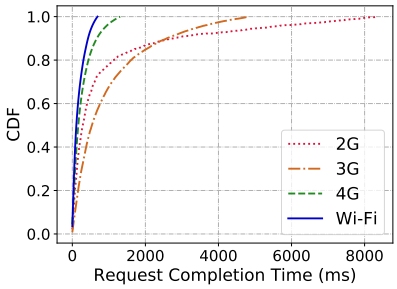


Figure 16: The distribution of request completion time with different access network types.

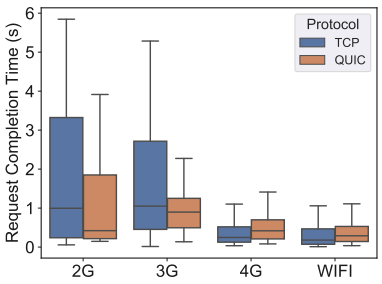


Figure 17: Request completion time of users in China.

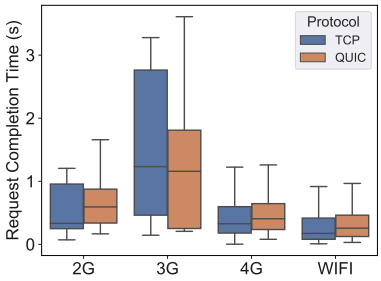


Figure 18: Request completion time of users in US.

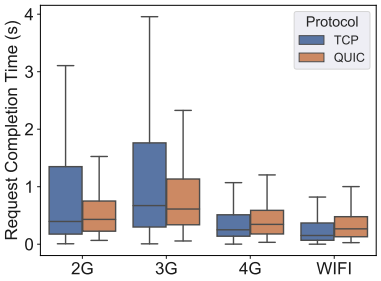


Figure 19: Request completion time of users in South Korea.

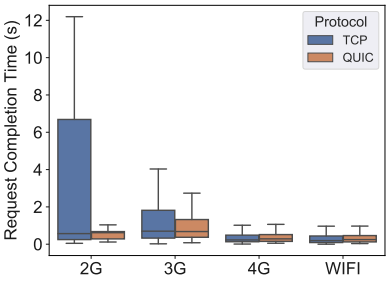


Figure 20: Request completion time of users in Netherland.

data, 8.58% requests were using QUIC due to user choice, and 5% requests were using QUIC 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 to it. 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.

A.2 Measurement Results

Fig. 15 shows the overall status of the measured requests. Among the four common types of access networks, WiFi has the best network 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. 16 shows the distribution of the request completion time with each access network 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 the reason why 2G is faster than 3G in some cases.

We evaluate the request completion time 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. 17 to Fig. 20 show the results in four typical countries. We can see that the request completion time is short in 4G and WiFi for both TCP and QUIC, and TCP performs a little better in all four countries. The request completion time is greater in 3G and 2G, and QUIC has a better performance in general, except for 2G users in US. The results in different countries are not similar. For example, the request completion time of 2G users in China is greater than other users, for both TCP and QUIC, while the request completion time of 2G users in US and South Korea is less than their 3G users, and the request completion time of 2G users in Netherland who use QUIC is very short. This reflects the complicated network conditions in the real world.

The observations are: 1) TCP in 2G and 3G networks is about 4X slower than QUIC on average, but it is still possible that TCP outperforms QUIC in some areas; and 2) TCP is faster than QUIC in 4G and WiFi networks on average, but the difference is small.

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. The performance gap within one access network type is bigger than the gap between different types due to time-varying network conditions. 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.