FlexHTTP: An Intelligent and Scalable HTTP Version Selection System

Mengying Zhou[1,2], Zheng Li[1,2], Shihan Lin[1,2], Xin Wang[1,2], Yang Chen[1,2]
[1]School of Computer Science, Fudan University, China
[2]Shanghai Key Lab of Intelligent Information Processing, Fudan University, China
{myzhou19,zli20,shlin15,xinw,chenyang}@fudan.edu.cn

ABSTRACT

HTTP has been the primary protocol for web data transmission for decades. Since the late 1990s, HTTP/1.1 has been widely used. Recently, both HTTP/2 and HTTP/3 have been proposed to achieve a better experience on web browsing. However, it is still unclear which of them performs better in different scenarios. In this paper, we first leverage the controllable experimental environment of the Emulab testbed to conduct a series of measurements and find that under different network conditions and web page structures, neither HTTP/2 nor HTTP/3 can always perform better. Motivated by this finding, we propose *FlexHTTP*, an intelligent and scalable HTTP version selection system. FlexHTTP embeds a supervised machine learning-based classifier to select the appropriate HTTP version according to network conditions and web page structures. FlexHTTP adopts a set of distributed agent servers to ensure scalability and keep the classifier up-to-date under network dynamics. We implement a proof-of-concept prototype of FlexHTTP on the Emulab testbed. Experiments show that FlexHTTP achieves a reduction of Speed Index by up to 600ms.

CCS CONCEPTS

• **Networks → Application layer protocols**; *Network management*; *In-network processing*; • **Computing methodologies** → *Machine learning*.

KEYWORDS

QUIC, web data transmission, HTTP/3, HTTP/2, HTTP version selection, machine learning

EuroMLSys '22, April 5–8, 2022, RENNES, France
© 2022 Association for Computing Machinery.
ACM ISBN 978-1-4503-9254-9/22/04…$15.00
https://doi.org/10.1145/3517207.3526972

ACM Reference Format:
Mengying Zhou[1,2], Zheng Li[1,2], Shihan Lin[1,2], Xin Wang[1,2], Yang Chen[1,2]. 2022. FlexHTTP: An Intelligent and Scalable HTTP Version Selection System. In *2nd European Workshop on Machine Learning and Systems (EuroMLSys '22), April 5–8, 2022, RENNES, France.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3517207.3526972

1 INTRODUCTION

Web browsing is one of the most important applications for users to access information and services on the Internet. Therefore, the speed of web page loading is critical to the quality of service (QoS). Since the 1990s, HTTP has acted as the foundation of data communication on web browsing, and the version HTTP/1.1 (H1.1) has become the dominant version of HTTP.

With the development of the Internet and the popularity of mobile devices, H1.1 has hardly satisfied the users' higher demands for speed and stability. As a result, more advanced network protocols and architectures are proposed. HTTP/2 (H2) [1] is a milestone that brings many optimizations, such as multiplexing, header compression, and server push. H2 enhances H1.1 and makes the data transmission more efficient. Same as in H1.1, H2 also chooses TCP as the transport layer protocol. HTTP/3 (H3) [2] is the successor to the H2 proposal. H3 is also known as "HTTP-over-QUIC" since it uses the QUIC protocol [16, 20] for data transmission. QUIC improves the QoS of HTTP by reducing connection and transmission delays. The 0-RTT handshake function provides a fast connection establishment. Meanwhile, the feature of multiple streams avoids the unnecessary delay caused by the head-of-line (HoL) blocking problem in TCP.

Although both H2 and H3 are promising, which version would achieve better performance under different situations is still being explored. To study this essential problem, we start with a series of comparative measurements for H2 and H3 using the Emulab testbed. We find that both network conditions and web page structures will impact the comparison of two HTTP versions' performance. From the perspective of network conditions, we can see that H3 mainly displays advantages over H2 in poor network conditions with high latency ($> 150ms$) and high packet loss rate ($> 1.5\%$). In contrast, H2 shows better performance under good network conditions. From the perspective of web page structures, more files and larger page size can help H3 be superior to H2. In short, neither H2 nor H3 can always obtain a better performance, and a flexible HTTP version selection in different situations is desired.

Motivated by the above findings, we design an intelligent and scalable HTTP version selection system named *FlexHTTP*. FlexHTTP embeds a supervised machine learning-based classifier to determine the optimal HTTP version by referring to the network conditions and web page structures. To obtain the network-related features, we deploy a series of agent servers to represent real-time network conditions between clients and different sets of web servers. The design of agent servers controls the measurement traffic volume at a moderate level, which guarantees the scalability of FlexHTTP. Moreover, FlexHTTP adopts an update mechanism for the classifier to ensure that it is up-to-date. Finally, We build a proof-of-concept implementation of FlexHTTP and deploy it on the Emulab testbed. The preliminary evaluation demonstrates the effectiveness of FlexHTTP in HTTP version selection.

Our main contributions are summarized as follows:

- According to our extensive studies based on Emulab, we find that both network conditions and web page structures are relevant to the performance comparison of H2 and H3. It is vital to select the suitable HTTP version in different scenarios.
- We design FlexHTTP, a machine learning-based intelligent and scalable HTTP version selection system, making use of network conditions and web page structures. The distributed agent servers and update mechanism ensure the information timeliness and scalability of FlexHTTP.
- We use the Emulab testbed to evaluate the performance of a proof-of-concept implementation of FlexHTTP. Our evaluation demonstrates that FlexHTTP is capable of choosing the appropriate HTTP version, which can reduce the Speed Index value by up to 600ms.

2 MOTIVATING MEASUREMENT

H2 and H3 are both advanced HTTP versions proposed to replace the widely-used H1.1. However, they enhance HTTP from different aspects, and each of them has its advantages and shortcomings. One key difference between H2 and H3 is the adopted transport layer network protocol. H2 chooses TCP while H3 uses QUIC, which is a UDP-based protocol.

Previous studies [17, 39] have demonstrated the performance difference between TCP and QUIC in some specified network conditions at the transport layer. [33, 38] have tried to compare H2 and H3 by visiting a set of selected Internet websites from a fixed network location. These studies provided a good start for comparing H2 and H3 "in the wild". Unfortunately, the network parameters of the end-to-end paths are not configurable. Thus, we cannot have a comprehensive view of the relationship between network parameters and the performance comparison.

Therefore, given the complexity and dynamics of the network conditions and the impact of application layer information (e.g., web pages structure), it is still unclear that under what situation H3 would perform better than H2, and vice versa. In this section, we introduce a controllable measurement to systematically study the performance difference between H2 and H3 in different situations.

2.1 Measurement Setup and Metric

We use Emulab [13], a representative network emulation testbed to emulate a series of controllable network conditions. To implement the web server, we choose Caddy [14], an open-source web server software supporting both H2 and H3. H2 enables TLS 1.2, and H3 adopts TLS 1.3. The congestion control algorithms for H2 and H3 are CUBIC [28] and NewReno [12], respectively. Caddy implements H3 with quic-go [6], which has been constantly developed following IETF drafts. In our study, Caddy-based web server uses IETF QUIC draft-29. H3 combines the handshake and TLS exchange process in one round trip. Therefore, once the client and server have reserved the communication key in the previous connection, they can enable the 0-RTT function. This key is known as the early data key [16]. To measure the performance of H3 with the 0-RTT function, we first request the server for a blank page to ensure the existence of the early data key. Then the 0-RTT connections can be established in the subsequent requests. We crawl the front pages of Alexa top 250 sites, and use the Caddy server to host these pages. To avoid the uncertainty from uncontrollable outer-domain URLs, we use dnsmasq [19] to intercept and discard all outer-domain requests. All machines used in our study belong to the pc3000 type [7] with CentOS 7, a single core 3GHz processor, and 2GB of RAM.

Speed Index (SI) [10, 27, 38] is a metric that indicates how quickly a page is loaded and visibly rendered. The smaller the SI value is, the faster the user can view the web page. We use the SI ratio, i.e., $R_{SI} = \frac{SI_{H2}}{SI_{H3}}$ to study the performance difference between the two HTTP versions. When $R_{SI} > 1$, we conclude that H3 performs better than H2. In contrast, if $R_{SI} < 1$, H2 would have a better performance than H3. We use Chrome (version 96.0.4664.93, released in December 2021) as the client browser and use Lighthouse [9] to record the SI values. Considering the possible network fluctuation, we browse each page 10 times and take the median value as the final measurement value.

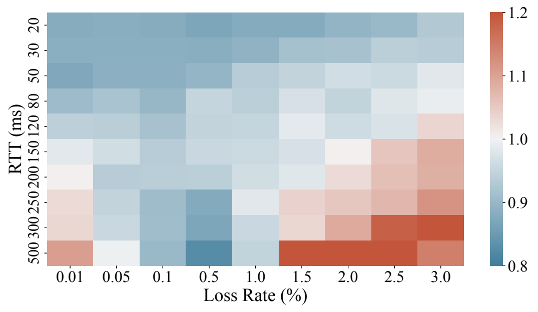


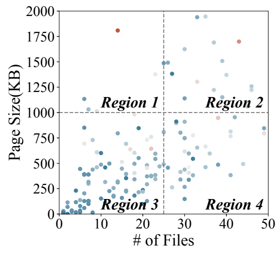
Figure 1: R_{SI} with different RTT values and packet loss rates. R_{SI} is the ratio of the SI values of H2 to H3. Red color indicates H3 outperforms H2 and blue color indicates that H2 outperforms H3.

2.2 Measurement Results and Findings

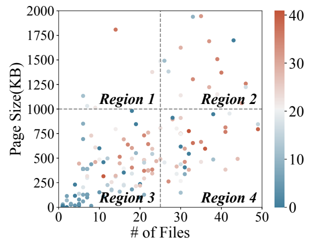
Impact of Network Condition. Emulab provides a controllable experimental environment to define network topologies, network parameters, machine configurations, etc. We evaluate 100 network conditions on Emulab with two key network parameters of RTT and packet loss rate. The network capacity of each link is configured as 10Mbps. The results shown in Fig. 1 exhibit some motivating insights. We find that most of the R_{SI} under poor network conditions are higher than 1. This means that H3 displays advantages (*median values 1076ms*) over H2 in network conditions with high latency ($> 150ms$) and high packet loss rate ($> 1.5\%$). In contrast, H2 shows better performance (*median values 467ms*) in good network conditions with low latency ($< 150ms$) and low packet loss rate ($< 1.5\%$). These advantages come from H3's better congestion control mechanism to reduce retransmissions, making it more suitable for networks with high loss rates and high latency [27].

Impact of Web Page Structure. Further, we study the impact of the structure of web pages, which is also an important factor for web-based applications. In particular, we focus on the numbers of files and the page size [17, 27] to study the performance difference of the two HTTP versions. We select two typical network conditions, i.e., "good" (RTT=30ms, loss rate=0.01%, capacity=25Mbps) and "poor" (RTT=500ms, loss rate=1.00%, capacity=1Mbps).

We still use R_{SI} to display the performance comparison between H2 and H3. In a good network condition, Fig. 2(a) demonstrates that H2 performs better than H3 when visiting the pages with a small number of large files (Region 1 and Region 3 in Fig. 2(a)), but this performance gap between the two HTTP versions is reduced when the number of files in a page becomes larger (Region 2 and Region 4 in Fig. 2(a)). This phenomenon is due to QUIC's multiplexing feature that solves the HoL blocking problem [27]. This enables H3 to reduce unnecessary blocking delays when visiting the pages with a large number of files. While in a poor network condition, H3 exhibits a clear advantage over H2. Even when visiting the pages with a small number of large files (Region 1 in Fig. 2(b)), H3 performs better than H2 in some cases.



(a) Good Network: RTT=30ms, loss rate=0.01%, capacity=25Mbps



(b) Poor Network: RTT=500ms, loss rate=1.00%, capacity=1Mbps

Figure 2: The impact of number of files and page size on R_{SI}, the ratio of the SI values of H2 to H3.

2.3 Towards a Flexible HTTP Version Selection

Overall, both network conditions and web page structures would impact the performance comparison between H2 and H3. It is evident that a fixed HTTP version cannot always obtain the best transmission performance. A flexible selection of the appropriate HTTP version can achieve better data transmission efficiency in different scenarios.

There are literature related to the selection of network protocols [35, 39], buffer sizes [31], CDN servers [30, 41], and routing paths [21, 36] according to different situations and user demands. Inspired by previous work and the above findings, we design and implement an intelligent and scalable HTTP version selection system, called *FlexHTTP*. FlexHTTP empowers a machine learning-based classifier to determine the better HTTP version by referring to the network conditions and web structure features. With the help of FlexHTTP, the SI value could be reduced, and the user experience could be enhanced.

3 FLEXHTTP DESIGN

In this section, we present the design of FlexHTTP. First, we list the design principles and practical challenges for building such an intelligent and scalable HTTP version selection system (Section 3.1). Subsequently, with a careful consideration of these principles and challenges, we show the overall architecture of FlexHTTP (Section 3.2). Later, We introduce the three key components of FlexHTTP, i.e., agent servers at strategic network locations (Section 3.3), HTTP version selector at the client-side (Section 3.4), and classifier updater at the client-side (Section 3.5). The source code of FlexHTTP could be accessed via https://github.com/mengyingzhou/flexhttp.

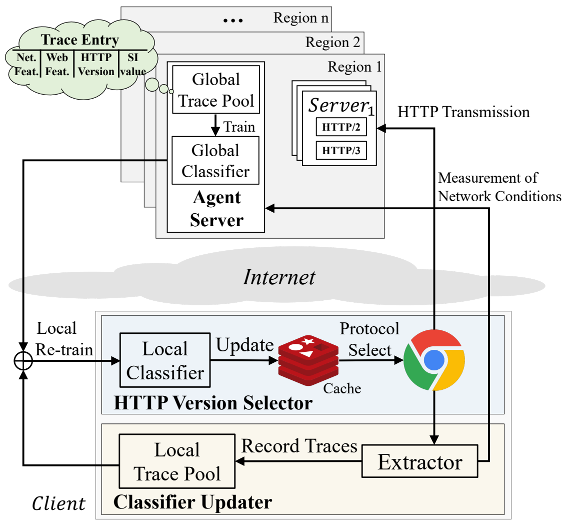


Figure 3: Overall architecture of FlexHTTP

3.1 Design Principles and Practical Challenges

Inspired by our findings in Section 2, building such an HTTP version selection system is a promising way to improve the user experience. FlexHTTP would extract a series of features from network conditions and web page structures, and the corresponding label representing the HTTP version that performs better between H2 and H3. Based on the above features and labels, FlexHTTP adopts a supervised machine learning-based classifier to determine whether H2 or H3 can achieve a more efficient data delivery. In addition, when deploying FlexHTTP in practical scenarios, we aim to address the following two main design challenges:

C1: Massive number of client-server pairs. As aforementioned, we need to extract network condition and page structures information as features for the machine learning-based classifier to make HTTP version selections. However, given the huge amount of clients and web servers, the possible number of client-server pairs and corresponding features would be tremendous. Real-time network measurements and web page structure features will not only delay the procedure of HTTP version selection, but also introduce a significant amount of measurement traffic. Some level of aggregation is needed to keep the overall measurement traffic volume at a modest level.

C2: Outdated classifier over time. The network conditions are evolving over time. Therefore, an immutable classifier might be out-of-date after a certain period. Our classifier updating scheme should meet the following two requirements: (1) It needs to ensure both generality and timeliness of classifier; (2) It should not introduce too much overhead when updating the classifier.

3.2 Overview of FlexHTTP

As we discussed in Section 3.1, the two key challenges hinder the design of the system. Here, we show the architecture of FlexHTTP in Fig. 3 with three key components: a set of agent servers in different regions, an HTTP version selector component, and a classifier updater component. The three components work collaboratively to address the aforementioned challenges.

S1: Deploying agent servers as representatives to reduce measurements. Agent servers are a group of auxiliary servers at strategic network locations to represent the nearby web servers. For example, we can place agent servers on data centers of public clouds where many online web servers are deployed [25]. Then, the network conditions between the client and these web servers are similar to those between the client and the agent server. A representative agent server can cover a group of nearby web servers. Therefore, the measurement overhead of tracking the network conditions between the massive number of client-server pairs can be reduced to an acceptable volume of measurements with agent servers, which ensures the scalability of FlexHTTP.

S2: Updating the classifier with a hybrid global-local mechanism. The classifier will be outdated over time. FlexHTTP keeps classifiers up-to-date with considering global and local information. Each agent server provides a global classifier for the client to download. The global classifier is constructed based on the global trace pool. The global trace pool is aimed to aggregate the information of different client-server pairs and web pages worldwide and is managed by different agent servers in a distributed method. Moreover, FlexHTTP allows a client to improve the classifier locally based on the local situation. A local trace pool is introduced to further fine-tune the classifier. Therefore, the classifier has not only considered the global information but also taken the particular client's characteristics into account.

3.3 Agent Servers

The set of agent servers is a key component that supports the operation of FlexHTTP. It has two essential responsibilities. The first is to provide real-time network conditions between the client and the corresponding region. The second is to jointly maintain the global trace pool in a distributed way. Based on this global trace pool, each agent server can build a global classifier for HTTP version selection.

3.3.1 ***Aggregative Network Condition Measurement.*** It is a heavy traffic overhead to obtain the network conditions between the client and numerous servers, since users browse a wide range of web services. To solve this problem, we deploy an agent server to represent web servers in the same region. Here a region denotes a geographic zone, where servers within this zone can reach each other with small latency [37]. We believe that this agent server would achieve a reasonable approximation to represent those web servers [4, 15]. In this way, the client only needs to measure the network conditions between itself and agent servers without generating redundant traffic to various servers. We propose to deploy agent servers close to various major data centers at different strategic network locations to represent more web servers.

3.3.2 ***Global Trace Pool and Classifier.***

The global trace pool aims to 1) collect trace entries under different network conditions and page structures, and 2) integrate the collected information to build a global classifier. Here a trace entry records the information when browsing, including the network condition and web page structures, the HTTP version used, and the corresponding SI value. Please note that the trace entries are anonymized. No personal information will be collected.

Besides the information in trace entry, the classifier construction also needs the ground-truth labels representing HTTP version selection, that is, under the same specific network condition and web page structures, which of H2 or H3 performs better. FlexHTTP creates the ground-truth labels for trace entries in the global trace pool as follows. (1) We group trace entries into clusters using the k-means clustering algorithm [22]. Accordingly, entries with similar network conditions and web page structures will be categorized into the same cluster. (2) For each cluster, we compare the median SI values of H2 and H3, and then calculate the ground-truth label, that is, which HTTP version achieves a smaller median SI value. Entries in the cluster share the same label, which is prepared for the re-training of the local classifier on the client-side.

Apart from the global trace pool, each client also maintains a local trace pool to collect the individual user's trace entries. Each client will periodically upload the local trace entries to its nearby agent servers to enrich the coverage of the global trace pool. In return, the always-up-to-date global trace pool would serve clients worldwide. A global classifier will be trained using the labelled trace entries from the global trace pool and available for clients to download. For newly joined clients, the global classifier can provide enough knowledge to make HTTP version selection. In particular, with the hybrid global-local re-training mechanism (details see Section 3.5.2), the global classifier can help build a more robust and generalized classifier for each client.

3.4 HTTP Version Selector

The HTTP version selector includes a browser, a cache that maintains some knowledge to assist HTTP version selection, and a classifier to implement the HTTP version selection. When the user starts to visit a website, the browser will first make a query to the cache to see if there is a history that could determine the HTTP version instead of repeated measurements and classifier inference. A cache entry has three fields: timestamp, URL, and historical HTTP version selection. The timestamp represents the historical network conditions of the same moment in a week since network dynamics show weekly periodicity [8], and URL to describe the web page structure information since the web page structures do not change frequently [18]. The cache only keeps historical records for the last seven days.

According to the timestamp and URL, we can locate the cached HTTP version selection. When a cache hit event occurs, the cache will respond to the browser with a preferable HTTP version directly. If there is no cache hit event, the HTTP version selector will randomly select a version (H2 or H3). Meanwhile, the cache will be updated by the local classifier. First, the extractor module in the classifier updater component will obtain the features required by the classifier to make selections. Subsequently, the local classifier performs the prediction based on these features to output the HTTP version selection and update the cache accordingly. The local classifier is a fine-tuned decision model based on network conditions and web page features. It is trained on the local trace pool by a selected machine learning algorithm in the classifier updater component.

3.5 Classifier Updater

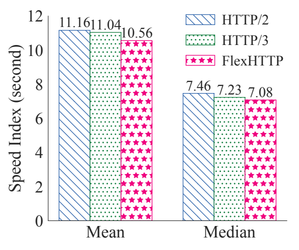
The local classifier will become invalid over time due to the dynamics of network conditions and web pages. The classifier updater aims to re-train the classifier periodically to update the classifier based on the user's dynamic browsing activities. We record each trace entry and extract the corresponding network condition and web page structure features to build a local trace pool for updating. We conduct this re-training process during the off-peak hours.

3.5.1 ***Feature Extractor and Labeling.***

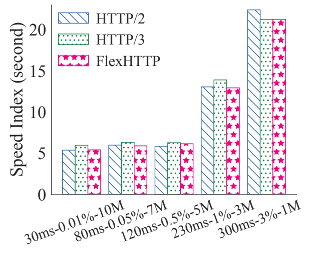
As we described above, FlexHTTP deploys agent servers at different strategic network locations to represent the network conditions of its nearby web servers. The client adopts these network conditions between itself and each agent server. To avoid unnecessary sensitivity to temporary network bursts and heavy measurement overhead, the extractor measures the network condition features within the time window t. When the network condition between the client and a specific server is needed, the measurement result of the corresponding agent, which represents the server in the current time window, will be adopted. RTT and loss rate are obtained by Ping [24], and network capacity is measured by iPerf [32]. The web page structure features can be analyzed from the HTML files after the page load completes. The detailed features are listed in Table 1.

Feature Set	Description	Experiment Settings		
Network	Capacity (Mbps)	0.5, 1, 3, 5, 7, 10, 25		
	RTT (ms)	30, 50, 80, 120, 200, 300, 500		
	Packet Loss Rate (%)	0.01, 0.05, 0.1, 0.5, 1, 2.5, 3		
Feature Set	Description	Avg	Std	Max
Web Page	# of all files	36.06	25	255
	Size of all files (MB)	1.08	0.56	22.99
	# of CSS files	2.28	2	37
	Size of CSS files (MB)	0.04	0.03	0.41
	# of JavaScript files	5.66	4	55
	Size of JavaScript files (MB)	0.18	0.10	1.53
	# of image files	24.58	10	231
	Size of image files (MB)	0.72	0.20	22.64

Table 1: List of the selected features and experiment settings



(a) Average and median values of Speed Index



(b) Median values of Speed Index of each configuration

Figure 4: The Speed Index performance comparison

Similar to the trace entries in the global trace pool, a trace entry here only contains the browsing information and lacks the ground-truth label of which version performs better. Fortunately, the global trace pool has already calculated the labels by clustering entries. Therefore, we can request each trace entries' labels from the global trace pool in the following way. The entries in the global trace pool have been clustered into clusters [22], and each cluster will have a cluster center and a shared ground-truth label. For each entry, we compute the Euclidean distance between this entry and each cluster center, and select the label of the cluster with the smallest Euclidean distance as the label of this entry.

3.5.2 ***Local Trace Pool and Local Classifier Re-training***. The network conditions and web pages are evolving from time to time. Therefore, a local classifier without a timely update might make a wrong HTTP version selection due to the out-of-date information.

Therefore, FlexHTTP periodically re-trains the local classifier for keeping it up-to-date. The local trace pool is introduced to collect the trace entries for re-training from the perspective of an individual client. Each collected local entry is processed by the feature extracting and labeling. Also, trace entries exceeding a pre-defined reservation time H will not be used for the next re-training iteration. However, only using the local trace pool for re-training will make the classifier underfit and perform poorly. Therefore, the re-training will combine the information from global and local pools. The global information is conveyed through a publicly accessible global classifier. With the client's browsing preference and activity pattern, the generalized global classifier can be fine-tuned by the local trace pool to satisfy the user's characteristics without losing generality. The idea of customizing the global classifier with local trace entries is taken from the widely used "fine-tune" technique [11].

4 PROOF-OF-CONCEPT EVALUATION

In this section, we use the Emulab testbed to conduct a proof-of-concept evaluation of FlexHTTP.

Experiment Setup. Before performing the evaluation, FlexHTTP requires an initial classifier at the beginning rounds. Benefiting from the controllable Emulab testbed, we collected 556,800 experiment configurations in various network conditions. Each experiment configuration represents a specific `RTT-loss_rate-capacity` composition. The information of different experiment configurations are shown in Table 1. Since HTTP version selection is a binary classification problem, we set "H3 better" traces as positive samples and "H2 better" traces as negative samples.

We have tried Decision Tree [29], Random Forest [3], XGBoost [5], and CatBoost [26] algorithms to implement the classifier and apply 10-fold cross-validation for training and validation. We adopt the best-performing (88.8% accuracy) model trained by the Random Forest algorithm as the initial classifier. When building the initial model, we do not perform hyperparameter tuning for these supervised machine learning algorithms. The hyperparameters are configured as default values, except that the tree depth is set to 4. FlexHTTP can improve its performance over time with the mechanisms of global updating and local re-training. Therefore, the initial classifier does not need to be tuned to prevent overfitting.

We select five common network environments (RTT-loss rate-capacity: 30ms-0.01%-10Mbps, 80ms-0.05%-7Mbps, 120ms-0.5%-5Mbps, 230ms-1%-3Mbps, 300ms-3%-1Mbps) as the evaluated configurations. Each configuration owns its independent client-server pair to emulate the configured network conditions, but all these configurations share the same agent server for information synchronization. The HTTP protocol versions, test websites, and machine setup are the same as those in Section 2. For each test website, we browse each page 10 times to avoid the impact of possible network fluctuation.
Evaluation Results. We use SI to evaluate H2, H3, and FlexHTTP. Fig. 4(a) shows the SI improvement brought from FlexHTTP owing to flexible HTTP version selection between H2 and H3. The mean SI value of FlexHTTP is 600ms and 480ms shorter than those of H2 and H3, respectively. As for median SI values, FlexHTTP also achieves 380ms and 150ms

reduction separately. Furthermore, we investigate the SI performance of H2, H3 and FlexHTTP on each configuration to depth in Fig. 4(b). We find that, although the best-performing HTTP versions on different configurations and web pages are different, FlexHTTP can always capture the appropriate HTTP version in almost every experiment configuration. In a word, FlexHTTP can select a more appropriate HTTP version to improve browsing performance according to the network environment and web page structure.

5 RELATED WORK

QUIC is built on top of UDP with the goal of efficient and reliable data transport [16, 20]. Lots of attention from both academia and industry have been paid to QUIC continuously. There is a considerable amount literature about the measurement of the performance of QUIC adoption under a controllable environment [17, 23] and the real Internet [20, 40].

Recently, the latest versions of HTTP, i.e., H2 [1] on TCP and H3 [2] on QUIC, have gradually been recognized and widely used in various Internet products and services. Varvello et al. [34] measured the difference between H2 and H1.1 in detail, showing that in most of the studied websites, H2 could reduce the PLT of H1.1. Trevisan et al. [33] conducted a comparative study on H1.1, H2, and H3 by using a client to access various websites on the Internet. They concluded that they could not find one HTTP version always performed better than the rest two. Furthermore, Yu and Benson [38] explored the difference between the H2 and H3 in the production environment by measuring the web services deployed by Google, Facebook, and Cloudflare.

These studies provided an opportunity to unravel the performance difference between H2 and H3. However, the network parameters of the end-to-end paths between the client and the web servers could not be configured flexibly. Thus these studies could not clearly summarize the comprehensive relationship between network parameters and performance comparison. To fill the gap, we conduct a controllable measurement to investigate the performance difference between H2 and H3 in different situations. We find that both network conditions and web page structures are relevant to the performance comparison of H2 and H3. It is vital to select the suitable HTTP version in different scenarios.

Under such argument that the appropriate HTTP version needs to be selected in different conditions, there is one prior work on transport protocol selection. Zhang et al. [39] proposed WiseTrans, an adaptive transport protocol selection approach according to the network condition, intending to improve the performance of mobile web services. However, WiseTrans only works for the transport layer, while FlexHTTP works at the application layer and will also consider the application layer information such as web page structures. Moreover, FlexHTTP has the following advantages over WiseTrans. (1) WiseTrans aims at serving the Baidu app's mobile users only, and has been evaluated only within the Baidu app's user environments. In contrast, FlexHTTP has the goal of serving Internet users getting connectivity from various types of services, and we do not limit it to be applied in mobile scenarios. (2) WiseTrans uses a global model to serve all clients, while FlexHTTP allows each client to use a local trace pool to further enhance the classifier.

6 CONCLUSION AND FUTURE WORK

Based on the phenomenon that neither HTTP/2 nor HTTP/3 can always perform better under different situations, we propose FlexHTTP, an intelligent and scalable HTTP version selection system. FlexHTTP leverages a supervised machine learning classifier to select the appropriate HTTP version according to different network conditions and page structures. A group of distributed agent servers are deployed to ensure the scalability of FlexHTTP and keep the classifier up-to-date. We build a proof-of-concept implementation of FlexHTTP and conduct an evaluation of FlexHTTP using the Emulab testbed. The evaluation results show that FlexHTTP can reduce the Speed Index value by up to 600ms.

While the above results demonstrate the potential of FlexHTTP in improving the experience of web browsing, some open issues still need to be further addressed to make FlexHTTP more practical. **1) Evaluating the performance of FlexHTTP in the wild.** This paper evaluates a proof-of-concept prototype of FlexHTTP deployed on an entirely controlled testbed. It is necessary for going through a comprehensive set of possible combinations of parameters. Still, we could evaluate FlexHTTP on the real Internet to see how it performs in environments with more complicated network dynamics. **2) Investigating the building blocks of FlexHTTP in a deep dive.** We only finish the preliminary evaluation of the FlexHTTP prototype without a detailed study of every component. The deployment strategy of agent servers, the overhead of classifier training and updating, and the setting of update period and retention time should be further explored. **3) Performance robustness verification.** In addition to the SI metric, we also need to discuss the performance of FlexHTTP using other performance metrics, e.g., Page Load Time and Core Web Vitals. Meanwhile, we will further enhance FlexHTTP to well serve mobile devices with limited computation capability and battery life.

ACKNOWLEDGMENTS

This work is supported by the National Natural Science Foundation of China (No. 61971145). Yang Chen is the corresponding author.

REFERENCES

[1] Mike Belshe, Roberto Peon, and Martin Thomson. 2015. Hypertext transfer protocol version 2 (HTTP/2). Available: https://datatracker.ietf.org/doc/html/rfc7540. Accessed: 2022-03-21.

[2] Mike Bishop. 2021. Hypertext Transfer Protocol Version 3 (HTTP/3). Available: https://datatracker.ietf.org/doc/html/draft-ietf-quic-http. Accessed: 2022-03-21.

[3] Leo Breiman. 2001. Random Forests. *Machine Learning* 45, 1 (2001), 5–32.

[4] Tian Bu and Don Towsley. 2002. On Distinguishing Between Internet Power Law Topology Generators. In *Proc. of INFOCOM*.

[5] Tianqi Chen and Carlos Guestrin. 2016. XGBoost: A Scalable Tree Boosting System. In *Proc. of SIGKDD*.

[6] Lucas Clemente. 2022. A QUIC Implementation in Pure Go. Available: https://github.com/lucas-clemente/quic-go. Accessed: 2022-03-21.

[7] Emulab. 2022. Type info for pc3000. Available: https://www.emulab.net/apt/show-nodetype.php?type=pc3000. Accessed: 2022-03-21.

[8] Phillipa Gill, Martin Arlitt, Zongpeng Li, and Anirban Mahanti. 2007. Youtube Traffic Characterization: A View From the Edge. In *Proc. of IMC*.

[9] Google LLC. 2022. Lighthouse. Available: https://developers.google.com/web/tools/lighthouse. Accessed: 2022-03-21.

[10] Google LLC. 2022. Speed Index. Available: https://web.dev/speed-index. Accessed: 2022-03-21.

[11] Suchin Gururangan, Ana Marasović, Swabha Swayamdipta, Kyle Lo, Iz Beltagy, Doug Downey, and Noah A Smith. 2020. Don't Stop Pretraining: Adapt Language Models to Domains and Tasks. In *Proc. of ACL*.

[12] Tom Henderson, Sally Floyd, Andrei Gurtov, and Yoshifumi Nishida. 2012. The NewReno Modification to TCP's Fast Recovery Algorithm. Available: https://datatracker.ietf.org/doc/html/rfc6582. Accessed: 2022-03-21.

[13] Fabien Hermenier and Robert Ricci. 2012. How to Build a Better Testbed: Lessons From a Decade of Network Experiments on Emulab. In *Proc. of TRIDENTCOM*.

[14] Matt Holt. 2022. Fast, cross-platform HTTP/2 web server with automatic HTTPS. Available: https://github.com/mholt/caddy. Accessed: 2022-03-21.

[15] Yi Hu, Feixiong Zhang, K. K. Ramakrishnan, and Dipankar Raychaudhuri. 2015. GeoTopo: A PoP-level Topology Generator for Evaluation of Future Internet Architectures. In *Proc. of ICNP*.

[16] Jana Iyengar and Martin Thomson. 2021. QUIC: A UDP-Based Multiplexed and Secure Transport. Available: https://www.rfc-editor.org/rfc/rfc9000.html. Accessed: 2022-03-21.

[17] Arash Molavi Kakhki, Samuel Jero, David Choffnes, Cristina Nita-Rotaru, and Alan Mislove. 2017. Taking a Long Look at QUIC: An Approach for Rigorous Evaluation of Rapidly Evolving Transport Protocols. In *Proc. of IMC*.

[18] Nikhil Kansal, Murali Ramanujam, and Ravi Netravali. 2021. Alohamora: Reviving HTTP/2 Push and Preload by Adapting Policies On the Fly. In *Proc. of NSDI*.

[19] Simon Kelley. 2022. Dnsmasq. Available: http://www.thekelleys.org.uk/dnsmasq/doc.html. Accessed: 2022-03-21.

[20] Adam Langley, Alistair Riddoch, Alyssa Wilk, Antonio Vicente, Charles Krasic, Dan Zhang, Fan Yang, Fedor Kouranov, Ian Swett, Janardhan Iyengar, et al. 2017. The QUIC Transport Protocol: Design and Internet-scale Deployment. In *Proc. of SIGCOMM*.

[21] Xuebing Li, Bingyang Liu, Yang Chen, Yu Xiao, Jiaxin Tang, and Xin Wang. 2019. Artemis: A Practical Low-latency Naming and Routing System. In *Proc. of ICPP*.

[22] James MacQueen. 1967. Some Methods for Classification and Analysis of Multivariate Observations. In *Proc. of 5th Berkeley Symp. Math. Statist. Probability*.

[23] Péter Megyesi, Zsolt Krämer, and Sándor Molnár. 2016. How quick is QUIC?. In *Proc. of ICC*.

[24] David L. Mills. 1983. Internet Delay Experiments. RFC 889. Available: https://www.rfc-editor.org/rfc/rfc889.html. Accessed: 2022-03-21.

[25] Giovane C. M. Moura, Sebastian Castro, Wes Hardaker, Maarten Wullink, and Cristian Hesselman. 2020. Clouding up the Internet: how centralized is DNS traffic becoming?. In *Proc. of IMC*.

[26] Liudmila Prokhorenkova, Gleb Gusev, Aleksandr Vorobev, Anna Veronika Dorogush, and Andrey Gulin. 2018. CatBoost: Unbiased Boosting with Categorical Features. In *Proc. of NeurIPS*.

[27] Mohammad Rajiullah, Andra Lutu, Ali Safari Khatouni, Mah-Rukh Fida, Marco Mellia, Anna Brunstrom, Ozgu Alay, Stefan Alfredsson, and Vincenzo Mancuso. 2019. Web Experience in Mobile Networks: Lessons from Two Million Page Visits. In *Proc. of WWW*.

[28] Injong Rhee, Lisong Xu, Sangtae Ha, Alexander Zimmermann, Lars Eggert, and Richard Scheffenegger. 2018. CUBIC for Fast Long-Distance Networks. Available: https://datatracker.ietf.org/doc/html/rfc8312. Accessed: 2022-03-21.

[29] S. Rasoul Safavian and David Landgrebe. 1991. A Survey of Decision Tree Classifier Methodology. *IEEE Transactions on Systems, Man, and Cybernetics* 21, 3 (1991), 660–674.

[30] Salvatore Scellato, Cecilia Mascolo, Mirco Musolesi, and Jon Crowcroft. 2011. Track Globally, Deliver Locally: Improving Content Delivery Networks by Tracking Geographic Social Cascades. In *Proc. of WWW*.

[31] Jiaxin Tang, Sen Liu, Yang Xu, Zehua Guo, Junjie Zhang, Peixuan Gao, Yang Chen, Xin Wang, and H. Jonathan Chao. 2022. ABS: Adaptive Buffer Sizing via Augmented Programmability with Machine Learning. In *Proc. of INFOCOM*.

[32] Ajay Tirumala. 1999. iPerf: The TCP/UDP bandwidth measurement tool. Available: https://iperf.fr. Accessed: 2022-03-21.

[33] Martino Trevisan, Danilo Giordano, and Ali Safari Khatouni. 2021. Measuring HTTP/3: Adoption and Performance. In *Proc. of MedComNet*.

[34] Matteo Varvello, Kyle Schomp, David Naylor, Jeremy Blackburn, Alessandro Finamore, and Konstantina Papagiannaki. 2016. Is the Web HTTP/2 Yet?. In *Proc. of PAM*.

[35] Keith Winstein and Hari Balakrishnan. 2013. TCP ex Machina: Computer-Generated Congestion Control. In *Proc. of SIGCOMM*.

[36] Jie Wu and Yunsheng Wang. 2012. Social Feature-based Multi-path Routing in Delay Tolerant Networks. In *Proc. of INFOCOM*.

[37] Mengwei Xu, Zhe Fu, Xiao Ma, Li Zhang, Yanan Li, Feng Qian, Shangguang Wang, Ke Li, Jingyu Yang, and Xuanzhe Liu. 2021. From Cloud to Edge: A First Look at Public Edge Platforms. In *Proc. of IMC*.

[38] Alexander Yu and Theophilus A. Benson. 2021. Dissecting Performance of Production QUIC. In *Proc. of WWW*.

[39] Jia Zhang, Enhuan Dong, Zili Meng, Yuan Yang, Mingwei Xu, Sijie Yang, Miao Zhang, and Yang Yue. 2021. WiseTrans : Adaptive Transport Protocol Selection for Mobile Web Service. In *Proc. of WWW*.

[40] Zhilong Zheng, Yunfei Ma, Yanmei Liu, Furong Yang, Zhenyu Li, Yuanbo Zhang, Jiuhai Zhang, Wei Shi, Wentao Chen, Ding Li, et al. 2021. XLINK: QoE-Driven Multi-Path QUIC Transport in Large-scale Video Services. In *Proc. of SIGCOMM*.

[41] Mengying Zhou, Tiancheng Guo, Yang Chen, Junjie Wan, and Xin Wang. 2021. Polygon: A QUIC-Based CDN Server Selection System Supporting Multiple Resource Demands. In *Proc. of Middleware*.