Fast Uplink Bandwidth Testing for Internet Users

Zhenhua Li, *Senior Member, IEEE, ACM*, Xingyao Li, Xinlei Yang, *Graduate Student Member, IEEE*, Xianlong Wang, Feng Qian, *Senior Member, IEEE, ACM*, and Yunhao Liu, *Fellow, IEEE, ACM*

***Abstract*— Access bandwidth measurement is crucial to emerging Internet applications for network-aware content delivery. However, today's bandwidth testing services (BTSes) are slow and costly—the tests take a long time to run, consume a great deal of data usage, and usually require large-scale test server deployments. The inefficiency and high cost of BTSes root in their methodologies that use excessive temporal/spatial redundancies for combating noises in Internet measurement. In particular, compared to downlink BTSes, uplink BTSes are subject to more severe performance problems and technical challenges. This paper presents FastUpBTS to make uplink BTS fast and cheap while maintaining high accuracy. The key idea is to strategically accommodate and exploit the noise rather than repetitively and exhaustively suppress the impact of noise. This is achieved by a novel statistical sampling framework termed *fuzzy rejection sampling*. We build FastUpBTS as an end-to-end BTS that implements fuzzy rejection sampling based on memorization-reinforced throughput denoising, data-driven server selection, and informed multi-homing support. Our evaluation shows that with only 30 test servers, FastUpBTS achieves the same level of accuracy compared to the state-of-the-art BTS (`SpeedTest.net`) that deploys ∼16,000 servers. Most importantly, FastUpBTS makes bandwidth tests 5.4× faster and 6.8× more data-efficient.**

***Index Terms*— Uplink bandwidth testing, internet speed test, access bandwidth measurement.**

I. Introduction

ACCESS bandwidth of Internet users commonly constitutes the bottleneck of content delivery, especially for emerging applications like AR/VR. In traditional residential broadband networks, it is largely stable and matches ISPs' service plans [1], [2]. In recent years, however, it becomes less transparent and more dynamic, driven by virtual network operators (VNOs), user mobility, and infrastructure dynamics [3].

To effectively measure the access bandwidth, bandwidth testing services (BTSes) have been widely developed and deployed, serving as a core component of many applications that conduct network-aware content delivery [4], [5]. BTSes' data are cited in government reports, trade press [6], and ISPs' advertisements [7]; they play a key role in ISP customers' decision making [8]. During COVID-19, BTSes are top "home networking tips" to support telework [9], [10]. The following lists a few common use cases of BTSes:

- VNO has been a popular operation model that resells network services from base carrier(s) [11], [12]. The shared nature of VNOs and their complex interactions with the base carriers make it challenging to ensure service qualities [13], [14]. Many ISPs and VNOs today either build their own BTSes [15], or recommend end users to use public BTSes. For example, `SpeedTest.net`, a popular BTS, serves more than 500M unique visitors per year [16].
- Wireless access is becoming ubiquitous, exhibiting heterogeneous and dynamic performance. To assist users to locate good coverage areas, cellular carriers offer "performance maps" [17], and several commercial products (*e.g.,* WiFi-Master used by 800M mobile devices [5], [18]) employ crowd-sourced measurements to probe bandwidth.
- Emerging bandwidth-hungry apps (*e.g.,* UHD videos and VR/AR), together with bandwidth-fluctuating access networks (*e.g.,* 5G) [19], make BTSes an integral component of modern mobile platforms. For example, the newly released Android 11 provides 5G apps with a bandwidth estimation API that offers "a rough guide of the expected peak bandwidth for the first hop of the given transport [20]."

Most of today's BTSes work in three steps: (1) setup, (2) bandwidth probing, and (3) bandwidth estimation. During the setup process, the user client measures its latency to a number of candidate test servers and selects one or more servers with low latency. Then, it probes the available bandwidth by downloading and uploading large files from and to the test server(s), and records the measured throughput as samples. Finally, it estimates the overall downlink/uplink bandwidth.

The major difficulty of BTSes is to deal with *noises* of Internet measurements incurred by congestion control, link sharing, *etc.* Spatially, the noise inflates as the distance (the routing hop count) increases between the user client and test server. Temporally, the throughput samples may be constantly fluctuating over time—the shorter the test duration is, the severer impact on throughput samples the noise can induce. An effective BTS needs to accurately and efficiently measure the access bandwidth from noisy throughput samples.

Today's BTSes are slow and costly. For example, a 5G bandwidth test using SpeedTest.net for 1.15 Gbps

Manuscript received 23 May 2022; revised 8 November 2022; accepted 30 December 2022; approved by IEEE/ACM TRANSACTIONS ON NETWORKING Editor A. Balasubramanian. This work was supported in part by the National Key Research and Development Program of China under Grant 2022YFB4500703, in part by the Natural Science Foundation of China (NSFC) under Grant 62202266, and in part by Microsoft Research Asia. *(Corresponding author: Xinlei Yang.)*

Zhenhua Li, Xingyao Li, Xinlei Yang, Xianlong Wang, and Yunhao Liu are with the School of Information Science and Technology, Tsinghua University, Beijing 100190, China (e-mail: lizhenhua1983@gmail.com; lixingyao816@gmail.com; yangxinlei19971105@gmail.com; bugaosuni598@gmail.com; yunhaoliu@gmail.com).

Feng Qian is with the Department of Computer Science and Engineering, University of Minnesota–Twin Cities, Minneapolis, MN 55455 USA (e-mail: fengqian@umn.edu).

Digital Object Identifier 10.1109/TNET.2023.3234265

downlink + 248 Mbps uplink takes 15 + 15 seconds of time and incurs 1.94 + 0.41 GB of data usage on end users. To deploy an effective BTS, hundreds to thousands of test servers are typically needed. Such a level of cost (both at the client and server sides) and long test duration prevent BTSes from being a foundational, ubiquitous Internet service for high-speed, metered networks. Based on our measurements and reverse engineering of a variety of commercial BTSes (Section II), we find that the inefficiency and cost of these BTSes fundamentally root in their methodology of relying on temporal and/or spatial redundancy to deal with noises:

- Temporally, most BTSes rely on a *flooding-based* bandwidth probing approach, which simply injects an excessive number of packets to ensure that the bottleneck link is saturated by test data rather than noise data. Also, their test processes often intentionally last for a long time to ensure the convergence of the probing algorithm.
- Spatially, many BTSes deploy dense, redundant test servers close to the probing client, in order to avoid "long-distance" noises. For example, FAST.com and SpeedTest.net deploy ∼3,000 and ∼16,000 geo-distributed servers, respectively, while WiFiMaster controversially exploits a large Internet content provider's CDN server pool.

In addition, we note that quite a few BTSes (*e.g.,* Xfinity [21] and ThinkBroadBand [22]) do not measure the uplink bandwidth in a symmetric manner (with respect to the downlink bandwidth test logic), oftentimes leading to rather poor accuracies. On the other hand, some BTSes (*e.g.,* SpeedOf [23] and ThinkBroadBand [22]) use the downlink bandwidth test result to guide the uplink test process in a simple way, in the hopes of making the latter more efficient and accurate.

In the preliminary work [24], we developed the FastBTS system for efficient and accurate downlink bandwidth testing. Our key idea is to accommodate and exploit the noise through a novel statistical sampling framework, which eliminates the need for long test duration and exhaustive resource usage for suppressing the impact of noise. Our insight is that the workflow of BTS can be modeled as a process of *acceptance-rejection sampling* [25] (or *rejection sampling* for short). During a test, a sequence of throughput samples are generated by bandwidth probing and exhibit a measured distribution $P(x)$, where x denotes the throughput value of a sample. They are filtered by the bandwidth estimation algorithm, in the form of an acceptance-rejection function (ARF), which retains the accepted samples and discards the rejected samples to model the target distribution $T(x)$ for calculating the final test result.

The key challenge of FastBTS is that $T(x)$ is unknown beforehand. Hence, we cannot apply the traditional rejection sampling algorithm that assumes a $T(x)$ and uses it as input. In practice, our extensive measurement results show that, while the noise samples are scattered across a wide throughput interval, the true samples tend to concentrate within a narrow throughput interval (termed as a *crucial interval*). Therefore, one can reasonably model $T(x)$ using the crucial interval, as long as $T(x)$ is persistently covered by $P(x)$. We name the above-described technique *fuzzy rejection sampling*, which

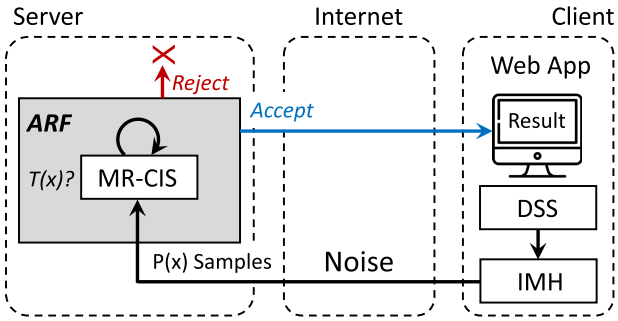


Fig. 1. Architectural overview of FastUpBTS. The arrows show the workflow of a bandwidth test in FastUpBTS.

is implemented in FastBTS through an Elastic Bandwidth Probing (EBP) mechanism generating high-quality $P(x)$ samples and a Crucial Interval Sampling (CIS) algorithm acting as the ARF. Also, the Data-driven Server Selection (DSS) and Adaptive Multi-Homing (AMH) mechanisms are used to establish multiple parallel connections with different test servers if necessary, especially when the client-side access bandwidth exceeds the capability of each test server.

In this paper, when attempting to reuse fuzzy rejection sampling for uplink bandwidth testing, we encounter two-fold additional challenges. First, EBP cannot be reused in uplink bandwidth testing because we cannot alter the user client (*e.g.,* the web browser), so the generated throughput samples may bear undesirable quality (*e.g.,* being subject to more severe fluctuations) and the bandwidth probing process could take much longer time to converge. Second, uplink bandwidth is typically much lower than downlink bandwidth, making the crucial interval of throughput samples less evident, *i.e.,* the accepted samples do not manifest sufficient concentration. On the other hand, there are certain opportunities facilitating our implementation of uplink bandwidth testing, *e.g.,* uplink testing always follows downlink testing whose result can provide helpful knowledge to the former.

Given the challenges and opportunities, we customize the uplink bandwidth test logic through two-fold innovations (Section III). First, we advance the original CIS mechanism by accumulatively reinforcing the significance of accepted samples in every period, so that the crucial interval can become evident in considerably less time; the resulting mechanism is dubbed Memorization Reinforced CIS or MR-CIS for short. Second, we leverage the downlink test result and the access media information to estimate how many servers are really needed for uplink testing, so that the original time-consuming AMH mechanism is upgraded into the efficient IMH (Informed Multi-Homing) mechanism. The high-level architecture of the new system, FastUpBTS, is depicted in Figure 1.

We have fully implemented FastUpBTS in a web-based manner for compatibility and usability considerations. We deploy it under diverse network scenarios (including Microsoft OpenNetLab [26] which offers us a variety of Internet edge nodes), using 30 geo-distributed budget cloud servers as the backend (Section IV). We also implement seven representative commercial BTSes on this testbed to ensure their apple-to-apple performance/overhead comparisons with

FastUpBTS. Our key evaluation results are summarized as follows:

- On the same testbed, FastUpBTS yields 3%–30% higher average accuracy than the other BTSes, while incurring 3.8–10.8× shorter test duration and 3.8–12× less data usage. At the same time, FastUpBTS flows incur trivial (<1%) interference to concurrent non-BTS network flows.
- Employing only 30 test servers, FastUpBTS achieves comparable accuracy compared with the production system of SpeedTest.net with ∼16,000 test servers, while incurring 5.4× shorter test duration and 6.8× less data usage.

To benefit the community as well as facilitate others' repeating and improving our work, we have released all the source code involved in this work at https://FastUpBTS.github.io.

II. Understanding State-of-the-Art BTSes

This section starts with our basic methodology of measuring a BTS (Section II-A), followed by our reverse engineering efforts towards understanding the uplink bandwidth test logic of mainstream BTSes (Section II-B). Then, we brief the measurement process and results (Section II-C), and describe in detail the cases of representative BTSes (Section II-D).

A. Methodology

We measure a BTS using the following three key metrics: (1) *Test Accuracy* measures how well the result (r) reported by a BTS matches the ground-truth access bandwidth (R). Then we calculate the test accuracy as $\frac{r}{R}$. In practice, we observe that all BTSes (including FastUpBTS) tend to underestimate the bottleneck bandwidth due to factors like TCP slow start and congestion control, so the accuracy values are always smaller than 1.0. (2) *Test Duration* measures the time needed to perform a bandwidth test—from starting a bandwidth test to returning the test result. (3) *Data Usage* measures the consumed network traffic for a test. This metric is of particular importance to metered LTE and 5G links. Since we concentrate on uplink bandwidth testing in this work, the three metrics are all calculated with regard to the uplink(-related) test process.

Obtaining ground truth or baseline reference bandwidth. It is ideal for us to measure the test accuracy with ground truth data. However, it is challenging to know all the ground truth bandwidths for massive measurements under heterogeneous environments. In this work, we use the best possible estimations for different types of access links:

- *Wired LANs for in-lab experiments.* We regard the (known) physical link bandwidth, with the impact of (our injected) cross traffic properly considered, as the ground truth.
- *Commercial residential broadband and cloud networks.* We collect the bandwidth claimed by the ISPs or cloud service providers from the service contract, denoted as T_C. We then verify T_C by conducting long-lived bulk data transfers (average value denoted as T_B) using `iPerf` [27] (a classic tool for active network measurement) before and after a bandwidth test. In 93% of our experiments, T_B and T_C match, with their difference being less than 5%; thus, we regard T_C as the *baseline reference bandwidth* with high confidence. Otherwise, we choose to use T_B as the baseline reference since T_B is more credible in this case.
- *Cellular networks (LTE and 5G).* Due to a lack of T_C and the high dynamics of cellular links and devices (in our controlled experiments the cellular devices are tested in both stationary and mobile scenarios), we leverage the results provided by SpeedTest.net as a baseline reference. Being the state-of-the-art BTS that owns a large number of (∼16,000) test servers across the globe, SpeedTest.net's results are usually more accurate than others' and thus are widely considered as a close approximation to the ground-truth bandwidth [28], [29], [30].

It is worth noting that UDP-based (in particular QUIC) measurements can also be used to validate the accuracy [31], [32]. Recently, Swiftest [33] even utilizes the data-driven UDP protocol to test mobile access bandwidth within one second or so. In this work, we concentrate on TCP-based measurements mostly for compatibility concerns, especially to meet the requirements of most web apps at the moment.

B. Reverse Engineering Mainstream BTSes

In this study we wish to unravel the bandwidth test logic of 21 state-of-the-art BTSes, including 20 widely-used web-based BTSes and one Android 11 uplink BTS API: `getLinkUpstreamBandwidthKbps`.[1] We run the 21 BTSes on three different PCs and four different smartphones listed in Table I (WiFiMaster and the Android API are only run on smartphones). To understand the implementation of these BTSes, we jointly analyze: (1) the network traffic (recorded during each test), (2) the client-side code, and (3) vendors' documentation. A typical analysis workflow is as follows.

First, we use `Wireshark` [49] (a popular network protocol analyzer) to capture and examine the packet-level network traffic, in order to reveal which server(s) the client interacts with during the test, as well as their interaction durations. We then inspect the captured HTTP(S) transactions to interpret the client's interactions with the server(s) such as server selection and file transfer. We also review the client-side code (typically written in JavaScript) of several BTSes such as Xfinity [50], LibreSpeed [51], and M-Lab NDT [52]. However, this attempt may not always succeed due to code obfuscation used by some BTSes like SpeedTest. In this case, we perform specialized benchmarks with controlled experiments where the peak bandwidth is strategically tuned, and meanwhile use the Chrome developer tool to monitor the entire test process in debug mode. Thereby, we can trigger the various latent, rare or extreme behaviors of a BTS, which effectively expose its internal (and oftentimes invisible) working procedures.

[1]The 20 web-based BTSes are ATTtest [15], BandwidthPlace (BWP) [34], CenturyLink [35], Cox [36], DSLReports [37], FAST [38], LibreSpeed [39], M-Lab NDT [40], NYSbroadband [41], Optimum [42], SFtest [43], SpeakEasy [44], Spectrum [45], SpeedOf [23], SpeedTest [46], ThinkBroadband (TBB) [22], Verizon [47], Xfinity [21], XYZtest [48], and WiFiMaster [5]. They are selected based on Alexa page ranks (regrettably retired on May 1, 2022) and Google page ranks (*i.e.,* by querying the keywords "bandwidth test OR speed test" with Google Search and selecting the high-rank BTSes in the first two pages of search results).

TABLE I
CLIENT DEVICES USED FOR REVERSE ENGINEERING THE BANDWIDTH TEST LOGIC OF 19 STATE-OF-THE-ART BTSES

Device	Location	Network	Ground Truth
PC-1	U.S.	Residential broadband	20 Mbps
PC-2	U.K.	Residential broadband	20 Mbps
PC-3	China	Residential broadband	20 Mbps
Samsung GS9	U.S.	LTE (60Mhz/1.9Ghz)	10–25 Mbps
Xiaomi XM8	China	LTE (40Mhz/1.8Ghz)	8–20 Mbps
Samsung GS10	U.S.	5G (400Mhz/28Ghz)	50–270 Mbps
Huawei HV30	China	5G (160Mhz/2.6Ghz)	30–210 Mbps

In addition, some BTSes (*e.g.,* SpeedOf, BWP, FAST, and SpeedTest) have provided official documentations for their bandwidth test logic [34], [53], [54], [55], which turn out to be very helpful (although not always up-to-date) references. With all the above efforts, we are able to "reverse engineer" the implementations of all these BTSes.

Our analysis shows that a complete bandwidth test in these BTSes (except the Android API) is typically done in five phases: (1) setup, (2) downlink bandwidth probing, (3) downlink bandwidth estimation, (4) uplink bandwidth probing, and (5) uplink bandwidth estimation.

In the setup phase, the BTS sends a list of candidate servers (based on the client's IP address or geo-location) to the client; for most BTSes such as BWP, Xfinity, LibreSpeed, FAST, and SpeedTest, the client then PINGs each candidate server over HTTPS. In detail, the client sends an HTTPS `GET()` request to each candidate server for fetching a small (typically $<$1 KB) piece of data, *e.g.,* an empty HTML or JSON file. Once the client gets a corresponding response from the server, the PING latency is measured by the round-trip time from when the request is sent to when the response is received.

Next, based on the servers' PING latencies (if available) or geo-locations, the client selects one or more nearby candidate servers as the test servers. Afterwards, the BTS performs download and upload bandwidth tests sequentially[2] by performing file transfer(s) between test servers and the client (bandwidth probing phase), and then processing the collected throughput statistics to estimate the client's bandwidth (bandwidth estimation phase).

Of course, we will pay special attention to the uplink test phases (4)+(5) and the uplink-related setup phase (1) in the remainder of this paper. For example, when calculating the "test duration" metric, we sum up the durations of phases (1)+(4)+(5); the calculation of data usage is alike.

C. Measurement Process and Results

We select 8 (out of 21) representative BTSes for more in-depth characterizations, as listed in Table II. They well cover different design paradigms of the remaining 13 ones, *i.e.,* these 8 BTSes were narrowed down out of 21 based on the key bandwidth test logic, rather than popularity or a random sample. Concretely, 7 BTSes (including ATTtest, CenturyLink, Cox, NYSbroadband, Optimum, SpeakEasy, and Spectrum) directly adopt SpeedTest's bandwidth test interface; the bandwidth test logic of Verizon and XYZtest is similar to that of SpeedTest. The bandwidth test logic of some BTSes is very similar. Specifically, the test process of WiFiMaster is almost the same as that of TBB, which delivers fixed-sized files for a certain period of time and then calculates the average throughput. M-Lab NDT works similarly as SpeedOf does: both of them sequentially transfer files with exponentially growing sizes (8 KB$\rightarrow$16 MB for M-Lab NDT and 128 KB$\rightarrow$128 MB for SpeedOf) for bandwidth probing, and calculate the overall average throughput (M-Lab NDT) or the last-file average throughput (SpeedOf) as the test result. DSLReports and SFtest quite resemble Xfinity by performing parallel file transfers to saturate the client's bandwidth.

Profiling the Performance. We build a geo-distributed heterogeneous testbed to comprehensively profile 7 representative BTSes, except the Android API (we will discuss separately in Section II-D). Our testbed is deployed on 10 Internet edge nodes offered by Microsoft OpenNetLab, consisting of one LTE node, one WiFi node, and eight wired-ADSL nodes. All these nodes are common desktop PCs contributed and shared by tens of organizations, with the incoming bandwidth ranging from 20 to 800 Mbps and the outgoing bandwidth ranging from 10 to 600 Mbps. Specially, the LTE node is a desktop PC connected to an LTE router, with $\sim$50 Mbps of downlink bandwidth and $\sim$15 Mbps of uplink bandwidth.

To avoid or minimize the impact of cross traffic among these shared nodes, we informed the administrator of OpenNetLab of the specific time period (usually lasting for a couple of hours) for our bandwidth tests in advance. The administrator then selected 10+ nodes with the least network overhead or the fewest users, and queried other users on these nodes whether they could evade our (bandwidth test) usage period. Only when all the other users on a node replied with their consents, the node was finally assigned to us.

For ease of presentation in the remainder of the paper, we denote one *test group* as using one edge client to run back-to-back (downlink and) uplink bandwidth tests across all the 7 BTSes in a random order. We perform in one day 300 groups of tests, *i.e.,* 10 edge clients $\times$ 3 different time-of-day (0:00, 8:00, and 16:00) $\times$ 10 repetitions.

Unravelling the Server Pool. Besides understanding the bandwidth test logic and performance of representative BTSes, we wish to demystify the server pool of each BTS, in particular the scale and distribution of test servers. This goal, however, cannot be reached with only the 10 edge nodes aforementioned, as most servers would be "hidden" due to the lack of nearby clients. To address this in a cost-effective manner, we take advantage of the globally-distributed, easy-to-deploy virtual machines (VMs) from multiple public cloud services providers (abbreviated as CSPs, including Azure, AWS, Ali Cloud, Digital Ocean, Vultr, and Tencent Cloud) as flexible client hosts, and monitor the setup phases regarding the same BTS to unravel its server pool as much as possible.

Specifically for a specific BTS, we begin with deploying a small number of VMs as the client hosts, and then gradually

[2]All the 20 web-based BTSes involved in our study peform download bandwidth tests before upload bandwidth tests, probably because common users are usually more concerned about their downlink bandwidths. Nevertheless, it is technically possible for a BTS to conduct upload bandwidth tests first, although we do not observe such cases in the measurements.

TABLE II
A BRIEF SUMMARY OF THE EIGHT REPRESENTATIVE BTSES IN TERMS OF UPLINK BANDWIDTH TESTING. THE PERFORMANCE RESULTS OF THE SEVEN WEB-BASED BTSES ARE ACQUIRED ON THE MICROSOFT OPENNETLAB-BASED TESTBED, WHILE THOSE OF THE ANDROID API ARE GOT ON THE TWO 5G PHONES LISTED IN TABLE I

BTS	Number of Test Servers	Uplink Bandwidth Test Logic	Duration	Accuracy	Data Usage
ThinkBroadband (TBB)	12	average of all the throughput samples	8 s	0.59	77 MB
SpeedOf	136	average throughput in the last connection	8–214 s	0.77	120 MB
BandwidthPlace (BWP)	18	average throughput in the fastest connection	13 s	0.74	199 MB
Xfinity	28	average of the last quarter of throughput samples	12 s	0.66	166 MB
LibreSpeed	33	average of all the throughput samples	7–15 s	0.62	118 MB
FAST	~3,000	average of stable throughput samples	8–30 s	0.84	270 MB
SpeedTest	~16,000	average of refined throughput samples	15 s	0.88	191 MB
Android API	0	direct bandwidth estimation using system configs	< 10 ms	0.09	0

add new VMs while keeping them geographically dispersed. After each VM starts the bandwidth test, we merely monitor the setup phase to record the information of candidate test servers – recall that in the setup phase, a client will be assigned a list of nearby candidate test servers from the BTS's server pool based on its IP address and geo-location (refer to Section II-B). As the deployed VMs become denser, we find that the total number of candidate test servers grows accordingly at first, then increases more and more slowly, and finally converges.

In particular, when the number of VMs exceeds 100, four of the six representative BTSes' candidate server sets stay unchanged, indicating that their server pools have been mostly, if not entirely, identified. For example, we identify that SpeedOf's server pool consists of 129 servers, quite close to the 136 servers officially claimed by SpeedOf [56]. On the other hand, for FAST and SpeedTest, using ~100 VMs can hardly uncover their entire server pools, but we expect that their very large server pools (whose scales are claimed to be around 3K and 16K respectively) can usually guarantee the closeness between each VM and the selected test server.

Result Summary. We summarize the major results in Table II, which reveal the landscape of today's uplink BTSes. We discover that all but one (*i.e.,* the Android API) of the BTSes adopt *flooding-based* approaches to combat the test noises from a temporal perspective, thereby leading to enormous data usage. Meanwhile, they differ in many aspects: (1) bandwidth probing mechanism, (2) bandwidth estimation algorithm, (3) connection management strategy, (4) server selection policy, and (5) server pool size. Most notably, compared to downlink BTSes, uplink BTSes are subject to more severe performance problems, particularly in terms of accuracy, though some of them (*e.g.,* Speedof and TBB) have utilized the downlink test result to optimize the uplink test process.

D. Case Studies

Below we present in detail our case studies of the eight representative BTSes as listed in Table II.

ThinkBroadBand (TBB) [22] measures a client's downlink bandwidth before conducting the uplink bandwidth test. Concretely, in the setup phase, TBB selects a test server with the lowest latency to the client among its server pool. Then, it performs an eight-second downlink bandwidth test by sequentially delivering 20-MB files to the client. Afterwards, it calculates the average throughput (*i.e.,* data transfer rate) during the whole test as the estimated bandwidth [24].

When downlink bandwidth test is accomplished, TBB moves on to uplink bandwidth test. In the uplink bandwidth probing phase, the client first performs a pre-test by generating and uploading a random file to the test server, whose size is proportional to the measured downlink bandwidth (say B_{down}). Quantitatively, the file size equals $B_{down} \times t_1$, where t_1 is nearly 150 ms according to our measurement. Then, TBB uses the average upload throughput during the pre-test as a rough estimation of the uplink bandwidth (say $\hat{B_{up}}$), and further tunes the random file size as $\hat{B_{up}} \times t_2$, where t_2 is around 100 ms. After that, TBB constantly generates and uploads a sequence of random files with the tuned size for eight seconds, and the overall average throughput is taken as the final uplink bandwidth test result. Our measurement shows that the accuracy of TBB is merely 0.59, because using the average throughput for bandwidth estimation cannot rule out the impact of TCP slow start during file transfer.

SpeedOf [23]. Similar to TBB, SpeedOf also leverages the downlink bandwidth test result to guide uplink bandwidth testing. Specifically, for a downlink bandwidth test, SpeedOf sequentially retrieves files with exponentially growing sizes from 128 KB to up to 128 MB. The test process stops when all file transfers complete or the current file transfer takes longer than eight seconds to finish. The average download throughput of the last file is reported as the downlink bandwidth test result.

In comparison, the initial file used for SpeedOf's uplink bandwidth test is usually not 128 KB in size; instead, the file size is set as 2^i MB when the measured downlink bandwidth lies between 2^{i+3} and 2^{i+4} Mbps (i is a non-negative integer), *i.e.,* between 2^i and 2^{i+1} MBps. Suppose a client's downlink bandwidth test result is 50 Mbps, which lies between 2^5 and 2^6 Mbps; then $i = 2$, and SpeedOf starts with a 4-MB file transfer in the hopes of rapidly saturating the uplink. On average, SpeedOf also bears unsatisfactory test accuracy (0.77). Worse still, in the presence of a slow network connection, the test duration of SpeedOf can be extremely long—up to 214 seconds in our measurement.

BandwidthPlace (BWP) [34]. The uplink bandwidth test process of BWP generally resembles that of SpeedOf—during the 13-second test, BWP constantly uploads files with exponentially growing sizes (which start from 8 MB). The major distinction lies in that instead of taking the average upload throughput of the last file as the

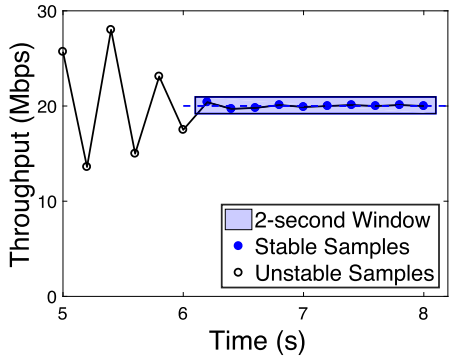


Fig. 2. Uplink test logic of FAST.

test result (SpeedOf's bandwidth test logic), BWP picks the file transfer session with the highest average throughput to generate the test result. As shown in Table II, BWP still bears low test accuracy (0.74), because even the fastest file transfer session is subject to the impact of TCP slow start when we average its throughput samples to generate the test result.

Xfinity [21] establishes four parallel HTTPS connections between a client and the latency-wise nearest test server for uplink bandwidth testing. Along each connection, Xfinity continuously generates and transfers a sequence of data chunks. The size of data chunks is initially set as 50 KB, and then adaptively adjusted according to the real-time upload situation. In detail, Xfinity keeps monitoring how many data chunks have been completely uploaded (along all the connections) during the last 200-ms time interval. If more than four data chunks have been completely uploaded within the time interval, Xfinity deems that the current chunk size is too small to saturate the client's upload bandwidth, and thus increases the chunk size by 10% (*e.g.,* from 50 KB to 55 KB) with the upper bound of 10 MB. Otherwise, the chunk size remains unchanged.

The above test process lasts 12 seconds, during which Xfinity calculates an overall throughput sample across all connections every 100 ms. After 12 seconds, the client's uplink bandwidth is estimated as the average of the throughput samples collected in the last quarter of the test process, *i.e.,* between 9 and 12 seconds. Table II indicates that Xfinity also yields a low accuracy (0.66). This is mainly because directly averaging the collected throughput samples in the last quarter as the bandwidth estimation still cannot effectively rule out the test noises due to network congestion, link sharing, *etc.*

LibreSpeed [39] is a popular open-source BTS with 8,600+ stars on Github. Similar to most BTSes, LibreSpeed selects a test server with the lowest latency to the client in the setup phase. Then, LibreSpeed progressively establishes three parallel HTTPS connections at the time of 0, 0.3, and 0.6 second, respectively. Along each connection, LibreSpeed sequentially generates and uploads 20-MB files to the test server.

LibreSpeed employs a unique early termination mechanism that adaptively shortens the test duration according to the client's real-time throughput. Specifically, LibreSpeed initializes the test duration as 15 seconds. Then, starting from the 3rd second, LibreSpeed calculates the client's current upload throughput (denoted as V_c) every 200 ms, and reduces the test duration accordingly by $\Delta T = \alpha \cdot V_c$. The parameter $\alpha = \frac{1}{160}$ is a constant factor that positively correlates V_c (in Mbps) with ΔT (in unit of seconds). That is to say, a higher upload throughput leads to a larger reduction in the test duration.

We note that ΔT has an upper bound of 0.4 second. Hence, when V_c stays above 64 Mbps (*i.e.,* $\alpha \cdot V_c > 0.4$), every 200 ms LibreSpeed reduces the test duration by 0.4 second, and then the total test duration is shortened from 15 to only 7 seconds. Finally, LibreSpeed averages all the collected throughput samples as the client's uplink bandwidth estimation.

As shown in Table II, benefiting from the early termination mechanism, LibreSpeed considerably saves the data usage (118 Mbps) compared with other web-based BTSes. However, this mechanism also induces inferior test accuracy (0.62). This is because a short test duration will undoubtedly magnify the impact of noises brought by TCP slow start, which usually takes a relatively long time (*e.g.,* ∼4 seconds for a typical 5G bandwidth test) for high-speed networks [33].

FAST [38] is an advanced BTS with a pool of about 3,000 test servers. It employs a two-step server selection process: the client first picks five nearby servers based on its IP address and geo-location, and then PINGs these five candidates to select the latency-wise nearest one as the test server. In the uplink bandwidth probing phase, the client starts with uploading a 25-MB file over a single HTTPS connection to the test server. When the file transfer completes, the client repeatedly uploads another 25-MB file until certain termination conditions are satisfied, which are elaborated below.

FAST estimates the uplink bandwidth as follows. As shown in Figure 2, it collects a throughput sample every 200 ms, and maintains a two-second window consisting of 10 most recent samples. After five seconds, FAST checks whether the in-window throughput samples are stable: $S_{max} - S_{min} \leq 3\% \cdot S_{avg}$, where S_{max}, S_{min}, and S_{avg} correspond to the maximum, minimum, and average value across all samples in the window, respectively. If the above inequality holds, FAST terminates the test and returns S_{avg}. Otherwise, the test will continue until reaching a time limit of 30 seconds; at that time, the last two-second window's S_{avg} will be returned to the user.

Unfortunately, our results show that the accuracy of FAST is still unsatisfactory (0.84). We ascribe this to FAST's window-based mechanism for early generation of the test result, which is vulnerable to throughput fluctuations. Under unstable network conditions, FAST can only use throughput samples in the last two seconds (rather than the entire 30-second samples) to roughly calculate the final test result.

SpeedTest [46] is generally considered the most advanced industrial BTS [28], [29], [30]. It has deployed a huge pool of ∼16,000 test servers as of May 2022. Similar to FAST, it employs a two-step server selection process: first identifying 10 candidate servers based on the client's IP address, and then selecting the latency-wise nearest. It also progressively increases the concurrency level: it begins with four parallel connections for quickly saturating the available bandwidth, and establishes a new connection when the throughput reaches 35 Mbps. It uses a fixed file size of 25 MB and a fixed duration of 15 seconds.

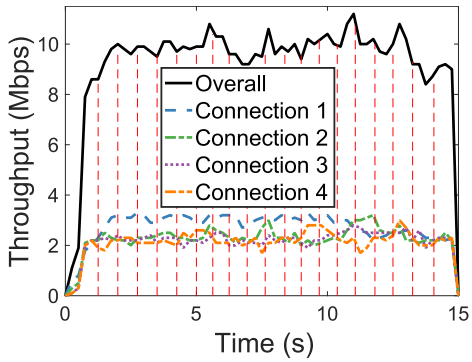


Fig. 3. Uplink test logic of SpeedTest.

SpeedTest's bandwidth estimation algorithm is quite different from FAST's. During the uplink bandwidth probing phase, it collects a throughput sample every 100 ms. Since the test duration is fixed to 15 seconds, all the 150 samples are used to construct 20 slices, each covering the same traffic volume, illustrated as the area under the throughput curve in Figure 3. Then, 10 slices with the lowest average throughput are discarded. This leaves 10 slices remaining, whose average throughput is returned as the final test result. This method may help mitigate the impact of throughput fluctuations, but the two fixed thresholds for noise filtering could be deficient under diverse network conditions.

Overall, SpeedTest exhibits the highest accuracy (0.88) among the measured BTSes. A key contributing factor is its largest server pool which can enable most bandwidth testing approaches (even when they are simplistic and naive) to yield high accuracy. In fact, when the server pool is large enough, every test client can be assigned with a very close test server, thereby avoiding or largely reducing the "long-distance" noises (*i.e.,* effective spatial denoising) during bandwidth testing.

Android API [4], [57]. To cater to the needs of bandwidth estimation for bandwidth-hungry apps (*e.g.,* UHD videos and VR/AR) over 5G, Android 11 offers a "Bandwidth Estimator" API to "make it easier to check bandwidth for uploading content [4]", *i.e.,* `getLinkUpstreamBandwidthKbps`. It statically calculates the access bandwidth by "taking into account link parameters (including radio technology, allocated channels, and so forth) [20]". It uses a pre-defined dictionary (`KEY_BANDWIDTH_STRING_ARRAY`) to map device hardware information to certain bandwidth values. For example, if the end-user's device is connected to the new-radio non-standalone mmWave 5G network, it searches the dictionary which records `NR_NSA_MMWAVE:145000,60000`, indicating that the downlink bandwidth is 145,000 Kbps and the uplink bandwidth is 60,000 Kbps. We test the API's performance in a similar manner as introduced in Section II-C with the 5G phones in Table I. The results show that the Android API bears extremely poor accuracy (0.09) in realistic scenarios.

III. Design of FastUpBTS

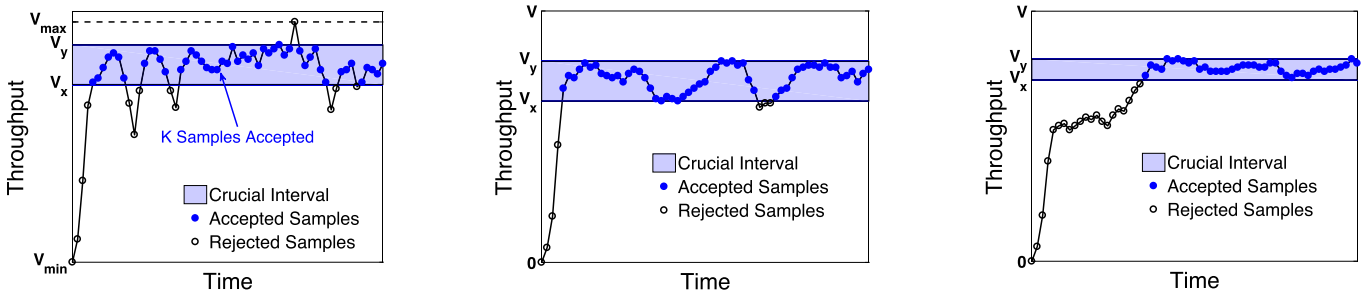
At a high level, estimating the client's access bandwidth based on a sequence of collected throughput samples is essentially an outlier removal problem that appears to be a well studied topic in statistics and data mining. But to our surprise, existing signal de-noising or anomaly detection techniques [58], [59], [60] can hardly be applied to our case, as they generally require quantitative characteristics of the outliers (*e.g.,* numerical thresholds, frequency-domain features, and pre-defined models), or are too heavyweight to carry out. In our case, the outliers stem from various complex sources where it is difficult to identify common characteristics. Also, our desired web-based access bandwidth testing prohibits any heavyweight implementations.

On the other hand, as mentioned in Section I, we notice that the classic approach of *acceptance-rejection sampling* [25] (or *rejection sampling* for short) seem to be a promising baseline solution due to its generic and lightweight nature. As a result, in this paper we propose and develop FastUpBTS by basing on and heavily customizing the framework of rejection sampling. FastUpBTS strategically *accommodates and exploits* noises (instead of suppressing them) to significantly reduce the resource footprint and accelerate the test process, while retaining high test accuracy. Its key technique is *fuzzy rejection sampling* which automatically identifies true samples that represent the target distribution and filters out false samples due to measurement noises, without apriori knowledge of the target distribution. Figure 1 shows its main components and the basic workflow of an uplink bandwidth test.

- *Memorization Reinforced Crucial Interval Sampling (MR-CIS)* implements the acceptance rejection function of fuzzy rejection sampling for uplink bandwidth testing. In FastBTS [24], the CIS mechanism searches for a *dense* and *narrow* interval that covers the majority of the desirable samples, and uses computational geometry to drastically reduce the searching complexity. Since uplink bandwidth is typically much lower than downlink bandwidth, MR-CIS advances CIS by accumulatively reinforcing the significance of accepted samples in every period, so that the crucial interval can become evident in considerably less time.
- *Data-driven Server Selection (DSS)* selects the server(s) with the highest bandwidth estimation(s) through a data-driven model. We show that a simple model can significantly improve server selection results compared to the de-facto approach that ranks servers purely by round-trip time.
- *Informed Multi-Homing (IMH)* is important for saturating the access link when the "last-mile" is not the bottleneck, *e.g.,* in some cases of 5G [61]. In FastBTS, Adaptive Multi-Homing (AMH) is adopted to progressively establish multiple parallel connections with different test servers, which is oftentimes time-consuming due to the gradual exploration. Since in mainstream BTSes, uplink testing always follows downlink testing, IMH leverages the downlink bandwidth test result and the access media information to estimate *in one shot* how many servers are really needed for uplink testing, so that the gradual exploration process of AMH can be almost fully avoided in FastUpBTS.

A. Memorization Reinforced CIS (MR-CIS)

CIS is designed based on the key observation: while noise samples are scattered across a wide throughput interval, the



(a) SpeedTest (b) TBB, BWP, XFinity (c) FAST

Fig. 4. Common scenarios where the true samples fall in a crucial interval during an uplink bandwidth test.

desirable samples tend to concentrate within a narrow interval, referred to as the *crucial interval*. As shown in Figure 4, in each subfigure, although the crucial interval is narrow, it can cover the vast majority of the desirable samples. Thus, while the target distribution $T(x)$ is unknown, we can approximate $T(x)$ with the crucial interval. Also, as more noise samples accumulate, the test accuracy would typically increase as randomly scattered noise samples help better "contrast" the crucial interval, leading to its improved approximation.

Crucial Interval Algorithm. Based on the above insights, our designed bandwidth estimation approach for FastUpBTS aims at finding this crucial interval ($[V_x, V_y]$) that has both *a high sample density* and *a large sample size*. Assuming there are N throughput samples ranging from V_{min} to V_{max}, our aim is formulated as maximizing the *product of density and size*. We denote the size as $K(V_x, V_y)$, *i.e.,* the number of samples that fall into $[V_x, V_y]$. The density can be calculated as the ratio between $K(V_x, V_y)$ and $N' = N(V_y - V_x)/(V_{max} - V_{min})$, where N' is the "baseline" corresponding to the number of samples falling into $[V_x, V_y]$ *if* all N samples are uniformly distributed in $[V_{min}, V_{max}]$. To prevent a pathological case where the density is too high, we enforce a lower bound of the interval: $V_y - V_x$ should be at least L_{min}, which is empirically set to $(V_{max} - V_{min})/(N-1)$. Given the above, the objective function to be maximized is:

$$F(V_x, V_y) = Density \times Size = C \cdot \frac{K^2(V_x, V_y)}{V_y - V_x}, \quad (1)$$

where $C = (V_{max} - V_{min})/N$ is a constant. Once the optimal $[V_x, V_y]$ is calculated, we can derive the bandwidth estimation by averaging all the samples falling into this interval.

The crucial interval is computed as the flooding-based bandwidth probing is in progress, which serves as the acceptance-rejection function (ARF) of rejection sampling. When a new sample is available, the server computes a crucial interval by maximizing Equation (1). It thus produces a series of intervals $[V_{x3}, V_{y3}]$, $[V_{x4}, V_{y4}]$, $\cdots$ where $[V_{xi}, V_{yi}]$ corresponds to the interval generated when the i-th sample is available.

Searching Crucial Interval with Convex Hull. We now consider how to actually solve the maximization problem in Equation (1). To enhance the readability, we use L to denote $V_y - V_x$, use K to denote $K(V_x, V_y)$, and let the maximum value of $F(V_x, V_y)$ be F_{max}, which lies in $(0, \frac{C \cdot N^2}{L_{min}}]$. Clearly, a naïve exhaustive search takes $O(N^2)$ time, which usually lies between 10 to 25 ms in practice with a relatively small N (ranging from 100 to 500). Nonetheless, when very fine-grained throughput samples are desired in special scenarios, N can be quite large (*e.g.,* 5,000–10,000) and then the exhaustive search would last quite long (*e.g.,* 1.2–5.4 seconds).

Our key result is that this can be done much more efficiently in $O(N \log N)$ by strategically searching on a convex hull dynamically constructed from the samples. Our high-level approach is to perform a binary search for F_{max}. The initial midpoint is set to $\lfloor \frac{C \cdot N^2}{2 \cdot L_{min}} \rfloor$. In each binary search iteration, we examine whether the inequality $\frac{C \cdot K^2}{L} - m \geq 0$ holds for any interval(s), where $0 < m \leq F_{max}$ is the current midpoint. Based on the result, we adjust the midpoint and continue with the next iteration. Please refer to Section 3.1 in our preliminary work [24] about how each iteration is performed exactly.

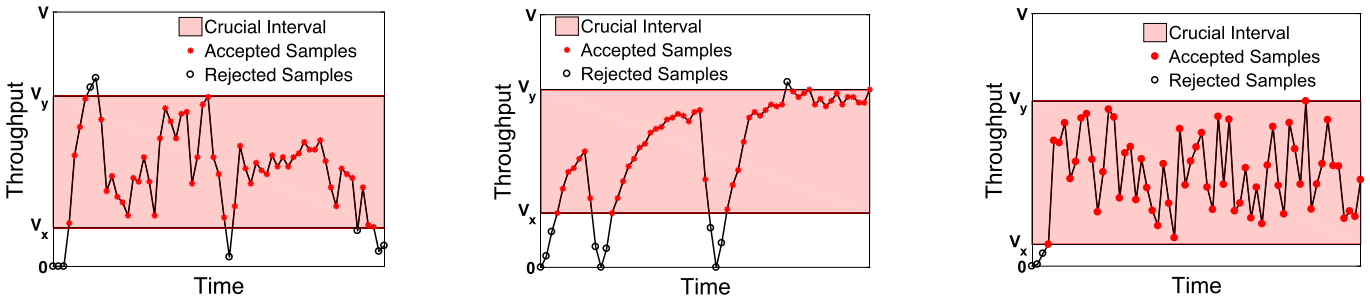
Fast Result Generation. CIS selects a group of samples that well fit $T(x)$ as soon as possible while ensuring data reliability. Given two intervals $[V_{xi}, V_{yi}]$ and $[V_{xj}, V_{yj}]$, we regard their similarity as the Jaccard Coefficient [62]:

$$S_{i,j} = \frac{([V_{xi}, V_{yi}] \cap [V_{xj}, V_{yj}])}{([V_{xi}, V_{yi}] \cup [V_{xj}, V_{yj}])}. \quad (2)$$

CIS then keeps track of the similarity values of consecutive interval pairs *i.e.,* $S_{3,4}, S_{4,5}, \ldots$ If the test result stabilizes, the consecutive interval pairs' similarity value will keep growing from a certain value β, satisfying $\beta \leq S_{i,i+1} \leq \cdots \leq S_{i+k,i+k+1} \leq 1$. If the above sequence is observed, CIS determines that the result has stabilized and reports the bottleneck bandwidth as the average value of the throughput samples belonging to the most recent interval.

The parameters β and k pose a tradeoff between accuracy and cost in terms of test duration and data traffic. Specifically, increasing β and k can yield a higher test accuracy while incurring a longer test duration and more data usage. Currently, we empirically set $\beta = 0.9$ and $k = 2$, which are found to well balance the tradeoff between test duration and accuracy. Nevertheless, when dealing with those relatively rare cases that are not covered by this paper, BTS providers are recommended to do pre-tests in order to find the suitable parameter settings before putting CIS into actual use.

Memorization Reinforcement. In this work, when attempting to reuse CIS (including Fast Result Generation)



(a) FAST (b) SpeedOf (c) SpeedTest

Fig. 5. Undesirable scenarios where the accepted samples within the crucial interval do not manifest sufficient concentration.

for uplink bandwidth testing, we encounter two-fold challenges.

- First, the Elastic Bandwidth Probing (EBP) mechanism in FastBTS cannot be reused in FastUpBTS because we cannot alter the user client (*e.g.,* the web browser). In comparison, we can customize the server system with our specially developed congestion control algorithm in FastBTS, which is transparent to the user client. Consequently, the generated $P(x)$ throughput samples may well bear undesirable quality, and the conventional flooding-based bandwidth probing process could take much longer time to converge.
- Second, for an Internet user's access link, its uplink bandwidth is typically much lower than its downlink bandwidth, making the crucial interval of throughput samples less evident, *i.e.,* the accepted samples do not manifest sufficient concentration, as demonstrated in Figure 5. As a matter of fact, this challenge aggravates the first one, thus causing the uplink bandwidth test time to get even longer.

To address these challenges, we devise the novel MR-CIS mechanism geared for FastUpBTS, which advances CIS by accumulatively reinforcing the significance of accepted samples in every period, so that the crucial interval can become evident in considerably less time. The effectiveness of this optimization heuristic stems from intuitive observations—if a throughput sample appears in most crucial intervals during the whole test process (each sampling period corresponds to a crucial interval), it is more likely to be accepted to generate the final bandwidth test result.

Concretely, on the basis of CIS, we additionally maintain an array $S = [s_1, s_2, \cdots, s_k]$, where s_k denotes the current significance value of the k-th throughput sample; all the significance values in S are initialized as zero in the beginning. Every time a new throughput sample is generated, we set its significance value as 1.0. In the i-th sampling period, we take each sample's significance into consideration when seeking the crucial interval by adjusting Equation (1) into:

$$F^*(V_x, V_y) = C \cdot \frac{\mathbb{S}^2(V_x, V_y)}{V_y - V_x}, \tag{3}$$

where $\mathbb{S}(V_x, V_y)$ denotes the significance sum of the throughput samples that fall in the interval $[V_x, V_y]$, and C is the same constant as that in Equation (1). When the crucial interval is calculated, we increase the significance values of the accepted samples in S by multiplying them with a fixed factor γ (empirically set as 1.1 to well balance the tradeoff between test accuracy and duration). In this way, each acceptance of a sample is "memorized" to reinforce the concentration of subsequent crucial intervals, thus mechanizing our aforementioned optimization heuristic.

B. Data-Driven Server Selection (DSS)

Similar to FastBTS, FastUpBTS also includes a history-based server selection method. We find that selecting the test server(s) with the lowest PING latency, widely used in existing BTSes, is ineffective. Both our measurement results and existing researches [63], [64], [65] indicate that the latency and available downlink/uplink bandwidth are not highly correlated—the servers yielding the highest throughput may not always be those with the lowest PING latency. In fact, latency and bandwidth are oftentimes orthogonal due to quite a few persistent or transient reasons, such as network capacity planning, traffic shaping, congestion control, link sharing, intermediate aggregation, and buffer bloat.

To address this drawback, FastUpBTS takes a *data-driven approach* to server selection, which leverages the historical data collected from the past tests to predict a test server's *expected* throughput for the upcoming bandwidth test. Each test server maintains a database containing {latency, throughput} pairs obtained from the setup and bandwidth probing phases of past tests. Figure 6 visualizes the database on a test server, where each node represents a data sample collected from a past test. Then in a new setup phase, the client still PINGs the test servers, while each server returns an expected throughput value based on the PING latency by looking up the database. The client will then rank the selected server(s) based on their expected throughput values rather than PING latencies only.

When estimating a test server's expected throughput, FastUpBTS takes two key factors into consideration. First, similar to most of today's BTSes, FastUpBTS utilizes the PING latency (l) between the test server and the client, since it generally reflects the quality of network connections from the spatial perspective. Furthermore, FastUpBTS adopts another key parameter w to extend the single latency value l into a more robust latency interval $[l - w, l + w]$ (as depicted in Figure 6). In this way, FastUpBTS accommodates and

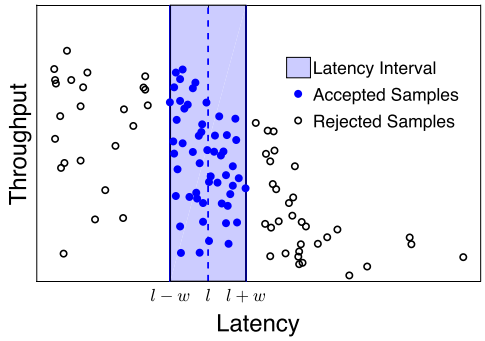


Fig. 6. Data-driven server selection that takes both historical latency and throughput information into account.

leverages more useful historical samples (including the noises) in the database from the statistical perspective.

While employing the similar insights as introduced by CIS in Section III-A, FastUpBTS further makes practical simplifications based on the key observation that only considering the sampling density can yield decent results when sufficient data have been accumulated. Concretely, with the measured PING latency l, the server tunes the parameter w to search for the desirable latency interval $[l-w, l+w]$ with the highest density

$$Density(l) = K(l, w)/2w, \tag{4}$$

where $K(l, w)$ denotes the number of historical data samples falling in the latency interval. Note that similar to the enforcement condition in CIS, we also empirically set a lower bound for the parameter w, *i.e.,* $w_{min} = 0.1 \times l$, to avoid the pathological cases where the density is unrealistically high. Finally, the expected throughput is calculated as an average of all samples in $[l-w, l+w]$. In addition, the server also returns the maximum throughput (using the 99-percentile value) belonging to $[l-w, l+w]$ to the client. Both values will be used in the bandwidth probing phase (Section III-C).

Due to its data-driven nature, the effectiveness of DSS largely depends on the quality of the database. In particular, DSS is subject to the intrinsic "cold start" (*a.k.a,* "cold boot") problem, *i.e.,* it may not be effective when there are no or very few tests conducted with regard to the same server. Therefore, during bootstrapping when servers have not yet accumulated enough samples, the client can fall back to the traditional latency-based selection strategy. Moreover, given that the "outdated" samples in the database cannot well represent the test server's recent status, FastUpBTS only maintains the most up-to-date (typically within one week) historical data in the databases, so as to facilitate more precise estimation of the expected throughput from the temporal perspective.

C. Informed Multi-Homing (IMH)

For high-speed access networks like 5G, the last-mile access link may not always be the bottleneck. To saturate the access link, we design the Adaptive Multi-Homing (AMH) mechanism in FastBTS to dynamically adjust the concurrency level, *i.e.,* the number of concurrent connections between the servers and client. In brief, AMH starts with a single connection to cope with possibly low-speed access links. At this time, the client establishes another connection with the second highest-ranking server. After that, we monitor the throughput variations of the two connections—if they do not seem to influence each other (due to cross-flow contention), the client establishes yet another connection with the third highest-ranking server; otherwise, the access link is considered saturated and the client stops further probing. More details of AMH can be found in our preliminary work [24].

AMH is quite effective in practice, but oftentimes time-consuming due to the gradual exploration process. Fortunately, there is a crucial opportunity for FastUpBTS to overcome this shortcoming, *i.e.,* in mainstream BTSes uplink testing always follows downlink testing. As a result, we advance AMH into the more efficient Informed Multi-Homing (IMH) mechanism, by leveraging the access media information and the downlink test result to estimate *in one shot* how many servers are really needed for uplink testing, so that the gradual exploration process of AMH can be fully avoided in FastUpBTS.

Specifically, in FastUpBTS each test server (*i.e.,* a budget VM server rented from Aliyun ECS) has 100-Mbps downlink bandwidth, and from our measurement results in Section II-C we obtain two key observations: (1) all the uplink bandwidth test results under 2G/3G/4G and WiFi 4 networks are below 100 Mbps; (2) for the vast majority (97%) of clients, their uplink bandwidths are lower (in fact much lower in practice) than half of the downlink bandwidths. Hence, when the client's access type is 2G/3G/4G or WiFi 4, or the measured downlink bandwidth is lower than 200 Mbps, we have high confidence that the client's uplink bandwidth will not exceed 100 Mbps, and thus only need to allocate one test server for the uplink bandwidth test. Otherwise, we conservatively over-estimate the upper bound of the uplink bandwidth as $\frac{B_{down}}{2}$ (B_{down} denotes the downlink bandwidth), and allocate $\lceil \frac{B_{down}}{2*100} \rceil$ servers for the test. In very few ($<$1%) cases, we observe that the downlink bandwidths of all the allocated servers are saturated; at this time, we will fall back to AMH to gradually add test servers.

IV. Evaluation

In this section, we compare the performance and overhead of FastUpBTS with those of the seven state-of-the-art BTSes (excluding the Android API for its extreme simplicity) studied in Section II. For fair comparisons, we re-implement all the BTSes based on our reverse engineering efforts or directly leveraging their open-source versions, and use the same setup (Section IV-A) for all these re-implemented BTSes. For comprehensive comparisons, we conduct both end-to-end evaluations for whole systems (Section IV-B) and in-depth evaluations for individual components (Section IV-C).

A. Experiment Setup

Edge-to-Cloud Testbed. We build a geo-distributed testbed consisting of 30 Aliyun ECS cloud servers (see Figure 7 for their locations) and 10 Microsoft OpenNetLab edge nodes (as mentioned in Section II-C). The cloud servers have the same

Fig. 7. Geographical distribution of our deployed 30 budget test servers rented from the Aliyun ECS cloud.

configurations: dual-core Intel CPU@2.5 GHz, 8-GB DDR memory, 100 Mbps incoming bandwidth (which is crucial in uplink bandwidth tests), and 100 Mbps outgoing bandwidth. Note that the abovementioned network capacity of cloud VM servers only guarantees the throughput between VMs within the cloud; their actual network bandwidth could be affected by other factors such as outbound traffic shaping mechanisms, load balancing strategies, and multi-tenant bandwidth sharing.

For a fair comparison with FastUpBTS, we replicate the seven other popular BTSes: TBB, SpeedOf, BWP, Xfinity, LibreSpeed, FAST, and SpeedTest with our uniform implementations (whose source code is available at https://FastUpBTS.github.io), and deploy them on the edge-to-cloud testbed. Exceptionally, we evaluate the Android API on the two 5G phones as described in Table I; since its performance (test accuracy) is extremely poor and its overhead (test duration and data usage) is almost zero, we choose not to list its evaluation results in this section.

Diverse Networks. Apart from the edge-to-cloud testbed, we conduct extensive evaluations under heterogeneous realistic networks, whose setups are detailed below.

- *LAN.* We create an in-lab LAN testbed to perform controlled experiments, where we can craft background traffic. The testbed consists of two test servers (S_1, S_2) and two clients (C_1, C_2), each equipped with a 2.5 Gbps NIC. They are connected by a commodity switch with a 1 Gbps forwarding capability, thus being the bottleneck. When running bandwidth tests on this testbed, we maintain two parallel flows: one background flow between S_1 and C_1, and the other bandwidth test flow between S_2 and C_2. Both of them are TCP flows using the conventional CUBIC congestion control algorithm: we first start a 500-Mbps background flow with `iPerf`, and then keep monitoring its real-time throughput; when the throughput stays around 500 Mbps, we further start the bandwidth test flow.
- *Residential Broadband.* We deploy three PCs in China, U.S., and U.K. (see Table I). All the PCs' uplink access is 20 Mbps residential broadband. The three clients communicate with the aforementioned 30 test servers to perform bandwidth tests. We perform in one day 90 groups of tests, consisting of 3 clients $\times$ 3 different time-of-day (0:00, 8:00, and 16:00) $\times$ 10 repetitions. Please refer to Section II-C for the meaning of a test group.
- *5G* experiments were conducted in a Chinese city using an HV30 phone over China Mobile. We perform in a day 30 groups of tests: 1 client $\times$ 3 time-of-day $\times$ 10 repetitions.
- *LTE* experiments were conducted in both China (a university campus) and U.S. (a large city's downtown area) using XM8 and GS9, respectively, with a total of 60 groups of tests, *i.e.,* 2 clients $\times$ 3 time-of-day $\times$ 10 repetitions.

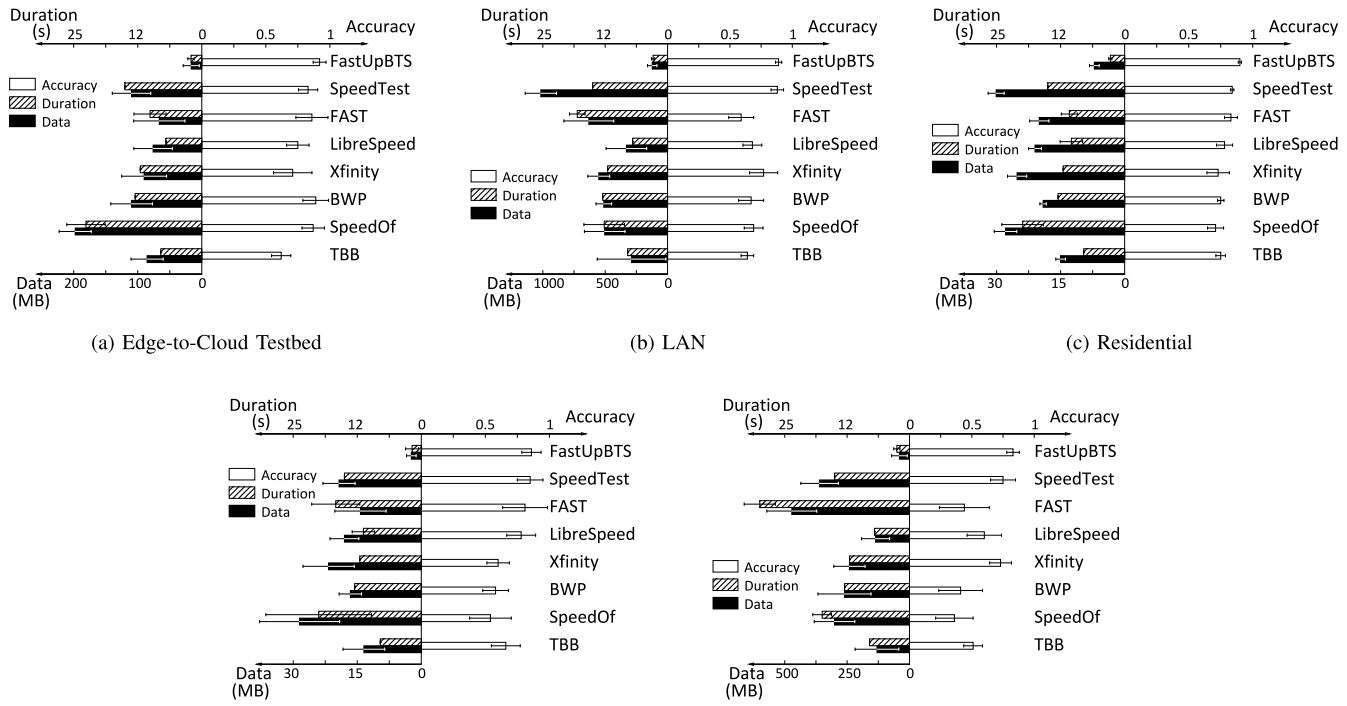
We use the three metrics (test duration, data usage, and test accuracy) and the methodology for obtaining the ground truth as described in Section II-A to assess uplink BTSes.

B. End-to-End Performance

Edge-to-Cloud Testbed. Figure 8a shows the performance of different BTSes atop the edge-to-cloud testbed. FastUpBTS outperforms the other BTSes by yielding the highest or sometimes comparable accuracy (0.93 on average), the shortest test duration (2.1 seconds on average), and the smallest data usage (16.5 MB on average). In contrast, the other BTSes' average accuracy ranges between 0.62 and 0.89; their test duration is also much longer, from 8.0 to 22.6 seconds, and they consume much more data (from 63 to 198 MB) compared to FastUpBTS. In particular, we find that Xfinity establishes too many parallel connections, thus leading to poor performance due to the excessive contention across the connections. FastUpBTS addresses this issue through IMH (Section III-C) that intelligently configures the concurrency level according to the network condition.

LAN and Residential Networks. As shown in Figures 8b and 8c, FastUpBTS yields the highest accuracy (0.89 for LAN and 0.9 for residential network) among the other six BTSes, whose accuracy lies within 0.59–0.88 for LAN, and 0.71–0.84 for residential network. The average test duration of FastUpBTS for LAN and residential network is 2.9 and 2.8 seconds respectively, which are 2.8–7.1$\times$ shorter than the other BTSes. The average data usage of FastUpBTS is 123 MB for LAN and 13.1 MB for residential network, which are 1.2–7.8$\times$ less than the other BTSes. The short test duration and small data usage are attributed to MR-CIS, which strategically trades a bit accuracy for remarkably shorter test duration.

LTE and 5G Networks. We evaluate the BTSes' performance on commercial LTE and 5G networks. Over LTE, as plotted in Figure 8d, FastUpBTS owns the highest or comparable (to the case of SpeedTest) accuracy (0.86 on average), the smallest data usage (2.34 MB on average), and the shortest test duration (1.8 seconds on average). Such outstanding performance mainly stems from MR-CIS which quickly and accurately estimates the client uplink bandwidth. By contrast, the other seven BTSes have relatively lower or equal accuracy (0.54–0.86) and far more data usage (13.5–28.5 MB). SpeedOf bears the lowest accuracy (0.54) because its bandwidth estimation algorithm (*i.e.,* adopting the average throughput in the last file transfer session) can be easily disturbed by many factors like TCP slow start and congestion control, thus leading to severely underestimated results. FAST, despite having a relatively high accuracy (0.86), incurs quite long test duration (16.7 seconds) due to the constant throughput fluctuations in LTE networks.



(a) Edge-to-Cloud Testbed (b) LAN (c) Residential

(d) LTE (e) 5G

Fig. 8. Test duration, data usage, and test accuracy of FastUpBTS, compared with seven representative BTSes under diverse networks. Note that the error bars indicate the standard deviations rather than min-max deviations.

Figure 8e shows the results for 5G access. It is also encouraging to see that FastUpBTS outperforms the seven other BTSes across all three metrics (0.83 vs. 0.36–0.75 for average accuracy, 40.3 vs. 130.4–471.8 MB for data usage, and 2.5 vs. 8–30 seconds for test duration). BWP, SpeedOf, and FAST bear rather low accuracy ($<$0.5); in comparison, SpeedTest and Xfinity have relatively high accuracy (0.75 and 0.73). However, the high data usage issue due to their flooding nature is drastically amplified in the 5G scenario. For example, SpeedTest incurs very high data usage—up to 361 MB per test. The data usage for FAST is even as high as 471.8 MB. FastUpBTS addresses this issue through the synergy of its key features (MR-CIS + IMH) for fuzzy rejection sampling. IMH helps quickly saturate the client's access bandwidth and MR-CIS effectively accelerates the convergence of throughput samples, thus greatly saving test duration and data usage.

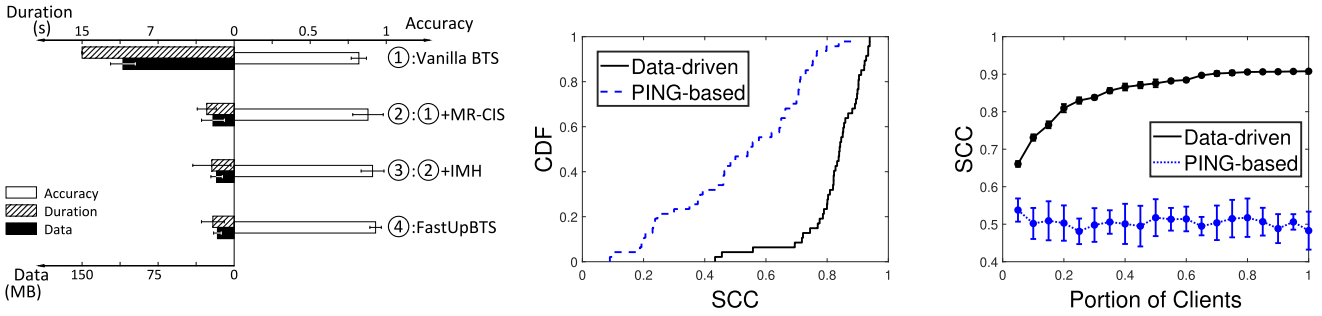
C. Individual Components

We evaluate the benefits of each component of FastUpBTS by starting from the vanilla BTS and then incrementally enabling one at a time atop the edge-to-cloud testbed. Concretely, when MR-CIS is not enabled, we average the throughput samples during 15 seconds to calculate the test result. When IMH is not enabled, we apply SpeedTest's (single-homing) connection management logic. When DSS is not enabled, test server(s) are selected based on purely the PING latency. It is worth clarifying that in all the following results, the test duration and data usage are calculated as the total duration and traffic cost of each entire experiment, rather than only (as a sum of) the crucial intervals.

Memorization-Reinforced Crucial Interval Sampling (MR-CIS). As shown in Figure 9a, by strategically removing outliers, MR-CIS increases the average accuracy from 0.82 to 0.88. Further thanks to the fast result generation mechanism, the test duration is reduced from 15 seconds to 2.7 seconds and the data usage is reduced by 5.2 times.

We next compare MR-CIS with the sampling approaches used by the other seven BTSes (refer to Section IV-A), which use a total of five bandwidth sampling algorithms because Xfinity employ the same trivial approach of simply averaging the throughput samples. To fairly compare them with MR-CIS, we take a replay-based methodology. Specifically, we select one "template" BTS from which we collect the network traces during the bandwidth probing phase; the time series of the aggregated throughput across all connections is then obtained from the traces and fed to all the sampling algorithms. We exclude SpeedOf and BWP from this experiment because they calculate the bandwidth based on the last or fastest connection that cannot be precisely reconstructed by our replay method.

We then show the results by using SpeedTest as the template BTS. The simplest algorithm (averaging) bears the lowest accuracy (0.82) because it is poor at eliminating the noises caused by, for example, TCP congestion control; the accuracy values of FAST and SpeedTest are 0.83 and 0.85, respectively. In contrast, MR-CIS owns the highest accuracy (0.88). This confirms the effectiveness of MR-CIS's sampling algorithm. The efficiency and effectiveness of MR-CIS lies in that, instead of incurring much redundancy in test duration and data usage to achieve a decent test accuracy, MR-CIS keeps calculating



(a) (b) (c)

Fig. 9. (a) Impact of individual modules of FastUpBTS on top of the edge-to-cloud testbed. (b) Distributions of SCC_{gp} (PING-based) and SCC_{gd} (Data-driven) when P=20%; (c) P (portion of clients) vs. SCC_{gp} and SCC_{gd}.

the crucial interval of the gathered throughput samples. Once the crucial interval stabilizes, MR-CIS immediately stops the test, thus significantly saving test duration and data usage.

Informed Multi-Homing (IMH). When IMH is further enabled, the test accuracy increases from 0.88 (MR-CIS) to 0.91 (MR-CIS + IMH), as shown in Figure 9a. We repeat the above experiments over 5G networks where the bottleneck is more likely to shift to the Internet side. The results show that IMH improves the average accuracy from 0.84 to 0.89, while reducing the average data usage from 58.6 MB to 47.3 MB and shortening the average test duration from 3.1 to 2.7 seconds. This is attributed to IMH's principled multi-homing strategy, which can efficiently saturate the client's access bandwidth especially under high-speed networks like 5G. IMH also facilitates MR-CIS by generating more accepted throughput samples, thus increasing the test accuracy.

It is worth noting that different from AMH whose testing over more connections takes additional time, IMH does not lengthen the test duration or increase the data usage. This is because in more than 99% cases, IMH determines the number of parallel connections in one shot, instead of progressively adjusting the concurrency level as AMH does.

Data-driven Server Selection (DSS). We employ *cross-validation* for a fair comparison between the PING-based method and DSS in three steps: (1) We do file transfers between every server and a randomly selected portion (P) of all clients to gather throughput samples. (2) Each client C runs a bandwidth test towards every server. In each test, the clients' historical test records (excluding the record of C) gathered in the previous step is utilized by each server to calculate the expected bandwidth, which is then returned to C. (3) Each client calculates three rankings of the servers based on the server-returned expected bandwidth: $Rank_g$, $Rank_p$, and $Rank_d$. $Rank_g$ refers to the server ranking based on the ground truth. $Rank_p$ is the ranking calculated based on PING latency; and $Rank_d$ is the ranking computed by DSS. We use the *Spearman Correlation Coefficient* (SCC [66]) to calculate the similarity SCC_{gp} between $Rank_g$ and $Rank_p$, as well as the similarity SCC_{gd} between $Rank_g$ and $Rank_d$.

The distributions of SCC_{gp} and SCC_{gd} when $P = 20\%$ are shown in Figure 9b. We find that SCC_{gd} is much higher than SCC_{gp} in terms of the median (0.84 *vs.* 0.54), average (0.82 *vs.* 0.52), and maximum (0.94 *vs.* 0.88) values. Further, Figure 9c shows that SCC_{gd} drops as P decreases; however, even when P decreases to 5%, SCC_{gd} (0.66) is still 27% larger than SCC_{gp} (0.52). These results show that DSS works reasonably well even with limited historical data, which confirms the key insight of DSS—the available bandwidth of a test server is not highly correlated with the real-time PING latency to the client, but the server's expected throughput estimated from its recent performance. We enable DSS in our experiments of Figure 9a with $P = 20\%$. Compared to MR-CIS + IMH, enabling DSS improves the average accuracy from 0.91 to 0.93; the test duration and data usage slightly reduce.

Apart from the performance gain, we also comparatively analyze the overhead of DSS and the traditional PING-based server selection strategy. Compared with the PING-based approach, DSS incurs more memory and CPU overhead at the server side, as it requires additional storage space for historical data (*i.e.,* {throughput, latency} pairs of the past tests) and extra calculation of latency interval (which is detailed in Section III-B). Quantitatively, when there are historical data of 10K past tests, the memory overhead is only 172 KB on average, and the latency interval can be figured out in 4 ms on average on a budget VM (whose configuration is introduced in Section IV-A).

Bandwidth Probing Intrusiveness. Moreover, we wish to evaluate the probing intrusiveness of FastUpBTS using the LAN testbed (refer to Section IV-A). We simultaneously run a 200 Mbps background flow and the bandwidth test flow that shares a 1 Gbps bottleneck at the switch. Ideally, a BTS should measure the bottleneck bandwidth to be 800 Mbps without interfering with the background flow, whose average/stdev throughput is thus used as a metric to assess the intrusiveness.

Our test procedure is as follows. We first run the background flow alone for one minute and measure its throughput as 200 Mbps. We then run FastUpBTS with the background flow and measure the average (standard deviation) of both the background flow throughput and the testing flow throughput during the test. We demonstrate the two groups' throughput samples with their timestamps normalized by the test duration in Figure 10. As we can see, FastUpBTS incurs trivial impact (0.3% degradation on the average throughput and 0.8% increase on the standard deviation) on the background flow. We repeat the above test procedure under other settings, with the background flow's bandwidth varying from 100 to 900 Mbps, and observe consistent results.

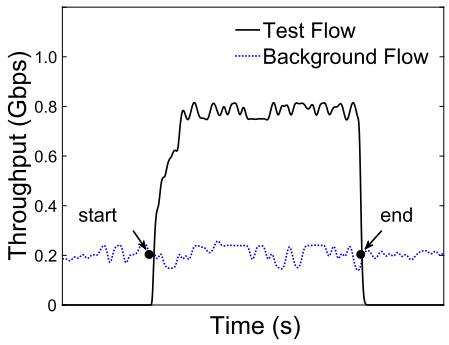


Fig. 10. Evaluating the probing intrusiveness of FastUpBTS.

Overall Runtime Overhead. Apart from test duration and data usage (as quantified in Section IV-B and the front part of Section IV-C), we also pay attention to the CPU and memory overheads incurred by FastUpBTS at client and server sides.

At the client side, our evaluation involves different types of devices including smartphones, PCs, and VMs (as mentioned in Section IV-A). On the four smartphones (*i.e.,* Samsung Galaxy S9, S10, Xiaomi M8, and Huawei Honor V30), the average CPU and memory overheads during bandwidth testing are 11% and 47 MB, respectively, which are quite acceptable. Besides, on the client PCs and VMs, the CPU overhead of FastUpBTS is even lower (5.4% on average), and the memory overhead is also trivial (17 MB on average).

At the server side, the incurred overhead is also low. When the bottleneck bandwidth is 100 Mbps, the CPU overhead is as small as 3.2% (single core, tested on Intel CPU@2.5 GHz, 8-GB memory). The CPU overhead is only 9% even when the bottleneck bandwidth is 1 Gbps.

V. Related Work

Bandwidth measurement is an essential component for many networked systems that empower a variety of important applications and use cases [65], [67], [68]. Apart from the mainstream BTSes described in Section II, other bandwidth measurement methods mostly target specific types of networks (*e.g.,* datacenter [69], LTE [70], and WiFi [71]) and require special support from the deployed infrastructure. For example, AuTO [69] conducts bandwidth estimation over DCTCP in data centers; it needs switch support to tag ECN marks on the data packets, and thus is challenging to be applied in WAN. Besides, Huang et al. [70] propose to deploy monitors inside the cellular core network for bandwidth measurement. In addition, Dischinger et al. [72] devise a bandwidth measurement tool which concurrently leverages multiple packet trains with different sending rates to measure the link bandwidth of residential broadband network.

While almost all the commercial BTSes we study in Section II employ flooding-based methods to combat measurement noises, there exists quite a few non-flooding ultra-light methods in academia. Some of them (*e.g.,* IGI [73] and Spruce [74]) employ a *packet gap model* that indirectly infers the access bandwidth based on the timing information of strategically crafted packets. Others (*e.g.,* pathload [75] and FlowTrace [76]) utilize a *packet rate model* that sends packet trains at different rates to explore the highest possible rate that does not lead to congestion. Unfortunately, these methods are highly sensitive to the packet gaps' timing information or packet trains' sending rates. Therefore, in practice they can be easily disrupted by many factors like packet loss [73], [77], queueing [73], and data/ACK aggregation [77], especially in high-speed networks.

Designed as a generic network service for Internet users, FastUpBTS differs from and complements the above work. FastUpBTS targets at conducting fast and light uplink bandwidth tests especially for high-speed wide-area networks (*e.g.,* 5G), significantly reducing data usage and test duration for clients and servers. It does not require any hardware support at the client or server side; also, it does not require the OS kernel modification at the server side, which however is a must in FastBTS to enable Elastic Bandwidth Probing (EBP) [24].

VI. Concluding Remarks

In this paper we present FastUpBTS, a novel Internet bandwidth testing system, to make uplink bandwidth testing fast and light as well as accurate. By strategically accommodating and exploiting the test noises, FastUpBTS achieves the highest level of accuracy among commercial BTSes, while significantly reducing data usage and test duration. Further, FastUpBTS only employs a small number of test servers, several orders of magnitude fewer than the state of the arts.

Despite the above merits, FastUpBTS still bears several limitations at the moment. First, when testing a client's uplink bandwidth, FastUpBTS requires the preceding test result of downlink bandwidth at the client side. Nevertheless, we note that there exist certain application scenarios where an uplink bandwidth test happens all alone (not following a downlink bandwidth test). In this case, the performance of FastUpBTS may degrade due to the potential inaccuracy of Informed Multi-Homing (IMH). Second, the performance of Data-driven Server Selection (DSS) can be affected by its cold start phase as well as the specific deployment of test servers. We have been exploring practical ways to overcome these limitations.

Acknowledgment

The authors would like to appreciate the editors and anonymous reviewers for their valuable comments.

References

[1] *Measuring Broadband America Fixed Broadband Report: A Report on Consumer Fixed Broadband Performance in the U.S.*, Federal Commun. Commission, Washington, DC, USA, 2014.
[2] *Measuring Broadband America Fixed Broadband Report*, Federal Commun. Commission, Washington, DC, USA, 2016.
[3] *SpaceX Starlink Speeds Revealed as Beta Users Get Downloads of 11 to 60 Mbps*. Accessed: Mar. 17, 2022. [Online]. Available: https://arstechnica.com/information-technology/2020/08/spacex-starlink-beta-tests-show-speeds-up-to-60mbps-latency-as-low-as-31ms/
[4] *Add 5G Capabilities to Your App*. Accessed: Mar. 17, 2022. [Online]. Available: https://developer.android.com/about/versions/11/features/5g
[5] *WiFiMaster*. Accessed: Mar. 19, 2022. [Online]. Available: https://en.wifi.com/wifimaster/
[6] S. Bauer, D. D. Clark, and W. Lehr, "Understanding broadband speed measurements," TPRC, 2010, pp. 1–38.
[7] *Understanding Internet Speeds*. Accessed: Mar. 22, 2022. [Online]. Available: https://www.att.com/support/article/u-verse-high-speed-internet/KM1010095
[8] Z. S. Bischof, J. S. Otto, M. A. Sánchez, J. P. Rula, D. R. Choffnes, and F. E. Bustamante, "Crowdsourcing ISP characterization to the network edge," in *Proc. SIGCOMM W-MUST Meas. Up Stack Workshop*, 2011, pp. 61–66.

[9] *Home Network Tips for the Coronavirus Pandemic*. Accessed: Mar. 26, 2022. [Online]. Available: https://www.fcc.gov/home-network-tips-coronavirus-pandemic

[10] *How Coronavirus Affects Internet Usage and What You Can Do to Make Your Wi-Fi Faster*. Accessed: Mar. 26, 2022. [Online]. Available: https://www.nbcnewyork.com/news/local/how-coronavirus-affects-internet-usage-and-what-you-can-do-to-make-your-wi-fi-faster/2332117/

[11] A. Xiao et al., "An in-depth study of commercial MVNO: Measurement and optimization," in *Proc. MobiSys*, Jun. 2019, pp. 457–468.

[12] Y. Li et al., "Understanding the ecosystem and addressing the fundamental concerns of commercial MVNO," *IEEE/ACM Trans. Netw.*, vol. 28, no. 3, pp. 1364–1377, Jun. 2020.

[13] F. Zarinni, A. Chakraborty, V. Sekar, S. R. Das, and P. Gill, "A first look at performance in mobile virtual network operators," in *Proc. IMC*, Nov. 2014, pp. 165–172.

[14] P. Schmitt, M. Vigil, and E. Belding, "A study of MVNO data paths and performance," in *Proc. Passive Act. Meas. (PAM)*. Cham, Switzerland: Springer, 2016, pp. 83–94.

[15] *AT&T BTS*. Accessed: Mar. 10, 2022. [Online]. Available: http://speedtest.att.com/speedtest/

[16] *SpeedTest Insights*. Accessed: Mar. 10, 2022. [Online]. Available: https://www.speedtest.net/insights

[17] *nPerf BTS*. Accessed: Mar. 10, 2022. [Online]. Available: https://www.nperf.com/en/map/US/-/2420.ATT-Mobility/signal/?ll=37.59682400108367&lg=-109.44030761718751&zoom=8

[18] D. Gao et al., "A nationwide census on WiFi security threats: Prevalence, riskiness, and the economics," in *Proc. MobiCom*, Sep. 2021, pp. 242–255.

[19] Y. Li et al., "A nationwide study on cellular reliability: Measurement, analysis, and enhancements," in *Proc. ACM SIGCOMM Conf.*, Aug. 2021, pp. 597–609.

[20] *Source of Android/NetworkCapabilities.Java*. Accessed: Mar. 15, 2022. [Online]. Available: https://cs.android.com/android/platform/superproject/+/master:frameworks/base/packages/Connectivity/framework/src/android/net/NetworkCapabilities.java

[21] *Xfinity BTS*. Accessed: Mar. 10, 2022. [Online]. Available: http://speedtest.xfinity.com/

[22] *ThinkBroadband BTS*. Accessed: Mar. 10, 2022. [Online]. Available: https://www.thinkbroadband.com/speedtest/

[23] *SpeedOf.Me BTS*. Accessed: Mar. 10, 2022. [Online]. Available: https://www.speedof.me/

[24] X. Yang et al., "Fast and light bandwidth testing for internet users," in *Proc. NSDI*. Berkeley, CA, USA: USENIX, 2021, pp. 1011–1026.

[25] G. Casella, C. P. Robert, and M. T. Wells, "Generalized accept-reject sampling schemes," *Festschrift Herman Rubin*, vol. 45, pp. 342–347, Jan. 2004.

[26] *Microsoft OpenNetLab: The Next-Gen Platform for AI-assisted Networking*. Accessed: Mar. 10, 2022. [Online]. Available: https://opennetlab.org/

[27] *iPerf*. Accessed: Mar. 10, 2022. [Online]. Available: https://iperf.fr/

[28] E. Alimpertis, A. Markopoulou, and U. Irvine, "A system for crowdsourcing passive mobile network measurements," in *Proc. NSDI*. Berkeley, CA, USA: USENIX, 2017, pp. 1–2.

[29] I. Canadi, P. Barford, and J. Sommers, "Revisiting broadband performance," in *Proc. IMC*, 2012, pp. 273–286.

[30] J. Sommers and P. Barford, "Cell vs. WiFi: On the performance of metro area mobile connections," in *Proc. IMC*, 2012, pp. 301–314.

[31] *TestMy.net: A UDP-based Bandwidth Testing Tool*. Accessed: Mar. 10, 2022. [Online]. Available: https://testmy.net/hoststats/udp

[32] *Framework for QUIC Throughput Testing*. Accessed: Mar. 10, 2022. [Online]. Available: https://www.ietf.org/archive/id/draft-corre-quic-throughput-testing-00.html

[33] X. Yang et al., "Mobile access bandwidth in practice: Measurement, analysis, and implications," in *Proc. ACM SIGCOMM Conf.*, Aug. 2022, pp. 114–128.

[34] *BandwidthPlace BTS*. Accessed: Mar. 10, 2022. [Online]. Available: https://www.bandwidthplace.com/

[35] *Centurylink BTS*. Accessed: Mar. 10, 2022. [Online]. Available: https://www.centurylink.com/home/help/internet/internet-speed-test.html/

[36] *Cox BTS*. Accessed: Mar. 10, 2022. [Online]. Available: https://speedtest.cox.com

[37] *DSLReports*. Accessed: Mar. 10, 2022. [Online]. Available:http://www.dslreports.com/speedtest/

[38] *FAST BTS*. Accessed: Mar. 10, 2022. [Online]. Available: https://fast.com/

[39] *LibreSpeed BTS*. Accessed: Mar. 10, 2022. [Online]. Available: https://librespeed.org/

[40] *M-Lab NDT BTS*. Accessed: Mar. 10, 2022. [Online]. Available: https://speed.measurementlab.net/#/

[41] *NYSbroadband BTS*. Accessed: Mar. 10, 2022. [Online]. Available: http://nysbroadband.speedtestcustom.com/

[42] *Optimum BTS*. Accessed: Mar. 10, 2022. [Online]. Available: https://www.optimum.net/pages/speedtest.html

[43] *HTML5 Speed Test by SourceForge*. Accessed: Mar. 10, 2022. [Online]. Available: https://sourceforge.net/speedtest/

[44] *Speakeasy*. Accessed: Mar. 10, 2022. [Online]. Available: https://www.speakeasy.net/speedtest/

[45] *Spectrum BTS*. Accessed: Mar. 10, 2022. [Online]. Available: https://www.spectrum.com/internet/speedtest-only/

[46] *SpeedTest BTS*. Accessed: Mar. 10, 2022. [Online]. Available: https://www.speedtest.net

[47] *Verizon BTS*. Accessed: Mar. 10, 2022. [Online]. Available: https://www.verizon.com/speedtest/

[48] *Speedtest.xyz BTS*. Accessed: Mar. 10, 2022. [Online]. Available: https://speedtest.xyz/

[49] *Wireshark*. Accessed: Mar. 10, 2022. [Online]. Available: https://www.wireshark.org/

[50] *Source Code of Xfinity*. Accessed: Mar. 10, 2022. [Online]. Available: https://github.com/Comcast/Speed-testJS

[51] *Source Code of LibreSpeed*. Accessed: Mar. 10, 2022. [Online]. Available: https://github.com/librespeed/speedtest

[52] *M-Lab NDT BTS*. Accessed: Mar. 10, 2022. [Online]. Available: https://github.com/m-lab/ndt7-client-go

[53] *Documentation of SpeedOf*. Accessed: Mar. 10, 2022. [Online]. Available: https://speedof.me/howitworks.html

[54] *Documentation of FAST*. Accessed: Mar. 10, 2022. [Online]. Available: https://netflixtechblog.com/building-fast-com-4857fe0f8adb

[55] *Documentation of SpeedTest*. Accessed: Mar. 10, 2022. [Online]. Available: https://help.speedtest.net/hc/en-us/articles/360038679354-How-does-Speedtest-measure-my-network-speeds

[56] *SpeedOf.Me: How It Works*. Accessed: Mar. 10, 2022. [Online]. Available: https://speedof.me/howitworks.html

[57] *BandwidthTest API in Android*. Accessed: Mar. 15, 2022. [Online]. Available: https://cs.android.com/android/platform/superproject/+/master:frameworks/base/core/tests/bandwidthtests/src/com/android/bandwidthtest/BandwidthTest.java

[58] B. Boashash, *Time-Frequency Signal Analysis and Processing: A Comprehensive Reference*. New York, NY, USA: Academic, 2015.

[59] D. L. Donoho, "De-noising by soft-thresholding," *IEEE Trans. Inf. Theory*, vol. 41, no. 3, pp. 613–627, Mar. 1995.

[60] V. Chandola, A. Banerjee, and V. Kumar, "Anomaly detection: A survey," *Comput. Surv.*, vol. 41, no. 3, pp. 1–58, Jul. 2009.

[61] A. Narayanan et al., "A first measurement study of commercial mmWave 5G performance on smartphones," in *Scanning Electron Microscope*. Cambridge, U.K.: Cambridge Univ. Press, 2019.

[62] M. Levandowsky and D. Winter, "Distance between sets," *Nature*, vol. 234, no. 5323, pp. 34–35, 1971.

[63] A. Narayanan et al., "A variegated look at 5G in the wild: Performance, power, and QoE implications," in *Proc. ACM SIGCOMM Conf.*, Aug. 2021, pp. 610–625.

[64] J. Wang et al., "An active-passive measurement study of TCP performance over LTE on high-speed rails," in *Proc. Mobicom*, 2019, pp. 1–16.

[65] R. K. P. Mok, H. Zou, R. Yang, T. Koch, E. Katz-Bassett, and K. C. Claffy, "Measuring the network performance of Google cloud platform," in *Proc. IMC*, Nov. 2021, pp. 54–61.

[66] A. Lehman, *JMP for Basic Univariate and Multivariate Statistics: A Step-by-Step Guide*. Cary, North CA, USA: SAS Institute, 2005.

[67] F. Li, A. A. Niaki, D. Choffnes, P. Gill, and A. Mislove, "A large-scale analysis of deployed traffic differentiation practices," in *Proc. SIGCOMM*, Aug. 2019, pp. 130–144.

[68] R. Yang, R. K. Mok, S. Wu, X. Luo, H. Zou, and W. Li, "Design and implementation of web-based speed test analysis tool kit," in *Proc. Passive Act. Meas. (PAM)*. Berlin, Germany: Springer, 2022, pp. 83–96.

[69] L. Chen, J. Lingys, K. Chen, and F. Liu, "AuTO: Scaling deep reinforcement learning for datacenter-scale automatic traffic optimization," in *Proc. SIGCOMM*, Aug. 2018, pp. 191–205.

[70] J. Huang et al., "An in-depth study of LTE: Effect of network protocol and application behavior on performance," in *Proc. ACM SIGCOMM*, Aug. 2013, pp. 363–374.

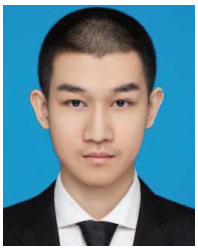
[71] T. Yang, Y. Jin, Y. Chen, and Y. Jin, "RT-WABest: A novel end-to-end bandwidth estimation tool in IEEE 802.11 wireless network," *Int. J. Distrib. Sensor Netw.*, vol. 13, no. 2, 2017, Art. no. 1550147717694889.

[72] M. Dischinger, A. Haeberlen, K. P. Gummadi, and S. Saroiu, "Characterizing residential broadband networks," in *Proc. IMC*, 2007, pp. 43–56.
[73] N. Hu, L. Li, Z. M. Mao, P. Steenkiste, and J. Wang, "Locating internet bottlenecks: Algorithms, measurements, and implications," in *Proc. SIGCOMM*, 2004, pp. 41–54.
[74] J. Strauss, D. Katabi, and F. Kaashoek, "A measurement study of available bandwidth estimation tools," in *Proc. IMC*, 2003, pp. 39–44.
[75] M. Jain and C. Dovrolis, "Pathload: A measurement tool for end-to-end available bandwidth," in *Proc. PAM*. Fort Collins, CO, USA: Springer, 2002, pp. 14–25.
[76] A. Ahmed, R. Mok, and Z. Shafiq, "FLOWTRACE: A framework for active bandwidth measurements using in-band packet trains," in *Proc. Passive Act. Meas. (PAM)*. Cham, Switzerland: Springer, 2020, pp. 37–51.
[77] B. Melander, M. Bjorkman, and P. Gunningberg, "Regression-based available bandwidth measurements," in *Proc. SPECTS*, 2002, pp. 14–19.

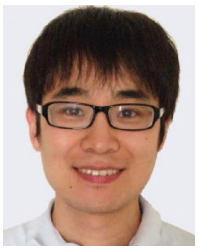
Zhenhua Li (Senior Member, IEEE) received the B.Sc. and M.Sc. degrees in computer science and technology from Nanjing University in 2005 and 2008, respectively, and the Ph.D. degree in computer science and technology from Peking University in 2013. He is currently a Tenured Associate Professor with the School of Software, Tsinghua University. His research interests include mobile networking/emulation and cloud computing/storage. He is a Senior Member of ACM.

Xingyao Li received the B.S. degree from the School of Software, Tsinghua University, Beijing, China, in 2021, where he is currently pursuing the M.Eng. degree. His research interests include cloud computing/storage and network measurement.

Xinlei Yang (Graduate Student Member, IEEE) received the B.S. degree from the School of Software, Tsinghua University, Beijing, China, in 2019, where he is currently pursuing the Ph.D. degree. His research interests include network measurement and web techniques.



Xianlong Wang received the B.S. degree from the School of Software, Beijing Jiaotong University, in 2019. He is currently pursuing the M.Eng. degree with the School of Software, Tsinghua University. His research interests include network measurement and cloud computing/storage.



Feng Qian (Senior Member, IEEE) received the B.S. degree from Shanghai Jiao Tong University and the Ph.D. degree from the University of Michigan. He is currently an Associate Professor with the Computer Science and Engineering Department, University of Minnesota–Twin Cities. His research interests include mobile systems, AR/VR, mobile networking, wearable computing, real-world system measurements, and system security. He is a Senior Member of ACM.

Yunhao Liu (Fellow, IEEE) received the B.S. degree from the Automation Department, Tsinghua University, and the M.S. and Ph.D. degrees in computer science and engineering from Michigan State University. He is currently a Full Professor and the Dean of Global Innovation Exchange (GIX), Tsinghua University. His research interests include sensor networks, the IoT, localization, RFID, distributed systems, and cloud computing. He is a Fellow of ACM.