

A Highly Scalable Bandwidth Estimation of Commercial Hotspot Access Points

Xinyu Xing*, Jianxun Dang†, Shivakant Mishra* and Xue Liu†

*Department of Computer Science, University of Colorado at Boulder. xingx@colorado.edu, mishras@cs.colorado.edu

†School of Computer Science, McGill University. jianxun.dang@mail.mcgill.ca, xueliu@cs.mcgill.ca

Abstract—WiFi access points that provide Internet access to users have been steadily increasing in urban areas. Different access points differ from one another in terms of services that they provide, including available upstream and downstream bandwidths, overall network capacity, open/blocked ports, security features, and so on. However, there is no reliable service available at present that can aid a user in selecting an access point from the many that are available. The primary research challenge is how to accurately estimate the current backhaul bandwidth of different access points in an efficient manner without requiring any installation of special software on the access points, and not burdening the WiFi subscribers to perform any communication or computation intensive task. This paper presents a new highly scalable bandwidth estimation technique that is suitable for efficiently estimating the backhaul bandwidth of a large number of APs. This technique has been extensively evaluated via a prototype implementation in an indoor testbed and in the Amazon EC2 platform. The evaluation demonstrates that the proposed technique exhibits high measurement accuracy, low latency, high scalability, and minimal intrusiveness.

I. INTRODUCTION

Availability of commercial hotspots that offer Internet access over a wireless network has been increasing significantly in most public areas. These hotspot WiFi services are fully-featured and time-tested. However, experience with using these services has been poor for many users. For example, a recent article in 2010 [7] reported "... commercial access points do not always meet expectations ...". Consider the following scenario. When a business traveler checks into a hotel, connects to a WiFi network in the hotel and attempts to upload financial statements to a centralized file server, he realizes that his WiFi device is experiencing poor connectivity due to limited Access Point (AP) backhaul bandwidth¹. So, the traveler decides to use a WiFi locator service to search for alternate WiFi networks nearby. Generally, he can find several alternatives with ease. However, apart from location and navigation information, current locator services such as Jiwire[5] and AT&T WiFi locations[6] do not provide any other information. As a result, the traveler has no way to find out in advance, i.e. before he pays for a connection, which AP provides the best quality of service as per his needs. Our goal is to address this limitation of current WiFi locator services. The main challenge here is how do we collect commercial hotspot APs' relevant performance metrics

accurately, efficiently and in near real time? Complexity arises because of a need for scalability. We need to collect the performance metrics of potentially a very large number of APs. We address this challenge in this paper.

As a proof of concept, We have built a system called *xylophone* that collects AT&T commercial hotspot APs' performance. *Xylophone* consists of three logical components: (1) a set of bandwidth estimation components mounted on bandwidth measurement servers (also called instances) in the cloud; (2) a client-side component running on hotspot AP subscriber's mobile device; and (3) a centralized database. In this paper, we will focus on the design, implementation and evaluation of the bandwidth estimation component. In particular, we propose a new bandwidth estimation technique that is implemented on each bandwidth estimation component for measuring different hotspot APs' backhaul bandwidth. Each bandwidth estimation component runs on an instance of the Amazon EC2 platform [1]. Interested readers can refer to [23] for a detailed description of the complete *xylophone* system.

Earlier research has revealed that the bottleneck of an Internet connection via a WiFi hotspot is the relatively low throughput DSL or cable modem links that connect the hotspot AP to the Internet [7][14][20]. Measurement of hotspot AP backhaul bandwidth on a large scale involves several unique challenges:

(1) *Scalability*: It has been observed that enabling large-scale bandwidth estimation can be problematic, because performing simultaneous measurements on paths that share at least a first few hops suffer from severe interference and thus provide inaccurate results [10]. For example, when two hotspot clients simultaneously perform bandwidth estimation tasks through the same hotspot AP, measurement probes may simultaneously saturate the hotspot AP.

(2) *Client-side computation and communication-intensive measurement tasks*: A WiFi user can perform bandwidth measurement by running a bandwidth estimation software [9][13] on his mobile device. However, continuous bandwidth measurement severely depletes his computation and communication resources. This may give rise to a situation where a hotspot subscriber may not be willing to sacrifice his limited resources to perform computation and communication-intensive bandwidth measurement. Furthermore, bandwidth measurement launched by hotspot subscribers may involve potential security threats because the inbound traffic to clients'

* and † have equal contributions to this work.

¹Unless otherwise stated, bandwidth refers to the available bandwidth as defined in [13][22].

mobile devices may compromise clients' security.

(3) *Measurement intrusiveness*: Bandwidth estimation involves sending several probe messages. When measurement latency increases, this may significantly impact a hotspot subscriber's connectivity performance. Therefore, bandwidth estimation has to be as less intrusive as possible with low measurement latency.

(4) *Link characteristics*: Typically, ISPs divide up a physical access link into smaller pieces that they parcel out to their commercial hotspot APs. A link that connects a hotspot AP to the Internet has the following characteristics: (1) it does not provide FIFO ordering because of a centrally coordinated MAC; (2) it does not have a fixed or well-defined raw bandwidth due to token-bucket rate regulation; and (3) the downstream data rate is significantly higher than the upstream data rate. It has been observed that these characteristics significantly impede the operation of currently existing bandwidth estimation tools [18].

The new bandwidth estimation component presented in this paper makes use of TCP ACK/RST messages and runs on a separate server. An extensive evaluation in both controlled settings and in the wild demonstrates that the protocol exhibits high measurement accuracy, low latency, high scalability, and minimal intrusiveness. This paper makes three important contributions. First, it presents a novel, highly scalable bandwidth estimation technique that is suitable for measuring APs backhaul bandwidths. None of the currently existing bandwidth estimation techniques can be used for this purpose. Second, the proposed technique formally addresses the important problem of large-scale bandwidth measurement, and successfully performs large-scale bandwidth estimation in the wild. To the best of our knowledge, this problem of large-scale bandwidth measurement in the wild has not been addressed before. Finally, a prototype of the proposed technique has been implemented and extensively evaluated via a controlled indoor testbed and a testbed deployed in Colorado region containing 50 accessible AT&T commercial hotspot APs and 11 instances (virtual machines) running on the Amazon EC2 platform.

II. RELATED WORK

Bandwidth estimation research has a long history. Most prior work requires installing a special software at both ends of the path to be measured [16][12][13][22][15][21][18]. This requirement significantly limits their applicability for our system. Since the process of bandwidth estimation is computation and communication-intensive, it is unrealistic to install measurement software on each hotspot client. Our design options are confined to a non-cooperative environment where measurement software is mounted solely on a single end of the path to be measured.

There has been some work recognizing and addressing issues that arise in non-cooperative environments. Abget[9] operates on the client side. Using the properties of TCP, Abget is able to force a remote HTTP host to generate a traffic flow at a certain rate, and estimates end-to-end bandwidth based on the latency of this flow. Since its operation is greatly

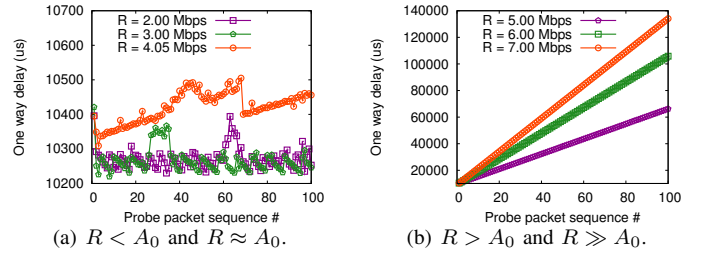


Fig. 1. RTT variations for a probe train (containing $K = 100$ TCP ACK packets) when the probe train passes through a link (with bandwidth $A_0 = 4Mbps$) at the different rate R .

dependent upon HTTP hosts [24] and hotspot APs are not HTTP accessible, it cannot be utilized in our system, unless it runs on client side. However, as mentioned earlier, there is no incentive for a hotspot client to perform a resource-intensive bandwidth measurement task on his mobile device. Abode[24] is another single-end bandwidth estimation software and it is not dependent on HTTP access. It uses ICMP instead of TCP. However, two issues limit the applicability of Abode in our system. First, our empirical observation indicates that most ISPs, including AT&T and T-Mobile constrain ICMP messages. Second, Abode requires that upstream bandwidth is equal to the downstream bandwidth.

Existing bandwidth estimation techniques have been mostly validated in controlled settings, e.g. PlanetLab [8], where all packets arriving at a link are serviced in FIFO order. However, as mentioned above, this FIFO ordering breaks down in broadband network settings [17]. In addition, existing bandwidth estimation techniques are not designed for large-scale deployments. According to a study [10], simultaneous bandwidth measurement can cause inaccurate estimate. To the best of our knowledge, the problem of large-scale bandwidth measurement in the wild has not been studied before. This is the first work that formally addresses this problem, and successfully performs large-scale bandwidth estimation in the wild.

III. BANDWIDTH MEASUREMENT TECHNIQUE

Our bandwidth measurement technique is based on TCP protocol behavior. In general, when a host sends a TCP acknowledgement (TCP ACK) packet to another host and if a TCP connection has not been established between them, a TCP reset (TCP RST) message with the fixed length of 40 bytes is sent in response regardless of the size of the TCP ACK packet. Since TCP ACK and RST packets traverse the network in two different directions (i.e. TCP ACK packets are transmitted from the bandwidth measurement server to hotspot APs - downstream link, and TCP RST packets are transmitted in the reverse direction - upstream link), we can saturate either downstream or upstream link by adjusting the size of the TCP ACK packet or the interval between back-to-back ACK packets. Our bandwidth measurement methodology consists of increasing the sending data rate either by increasing the ACK packet size or the interval between back-to-back ACK

packets, and observing the round-trip time (RTT). Here, we define RTT of a TCP ACK packet as the time elapsed between the time when the ACK packet is sent and the time when the corresponding RST packet is received. The key idea is that when none of the links are saturated, RTT does not change with increase in the data rate. On the other hand, when one of the links is saturated, RTTs between successive TCP ACK packets will have an overall increasing trend. This is because queuing delays start increasing when one of the link has been saturated. The bandwidth of the saturated link is then estimated as the sending data rate when RTT starts increasing from being same.

As an example, Figure 1 illustrates the variation in RTT for a sequence of ACK packets for different data rates. In this experiment, a bandwidth measurement server in the cloud sends 100 TCP ACK packets at different data rate on a 4 Mbps link. As shown in the figure, when sending rate is lower than 4 Mbps, i.e. for data rate 2 and 3 Mbps, the RTT of each TCP ACK packet is approximately same for ACK packets. On the other hand, RTTs increase for each successive ACK packet when data rate is above 4 Mbps, i.e. for data rates 4.05, 5, 6 and 7 Mbps. Furthermore, this increase in RTT is higher for higher sending data rates.

As mentioned earlier, upstream link bandwidth is lower than downstream link bandwidth. So, to measure the upstream link bandwidth, we saturate that link by increasing the rate at which 40B ACK packets are sent. As this rate is increased, the rate at which RST is sent also increases. Since upstream link bandwidth is lower than the downstream bandwidth, it saturates before the downstream link. On the other hand, to saturate the downstream link bandwidth, we saturate that link by increasing the size of the ACK packet sent at some predetermined rate. This ensures that the data rate on the upstream link doesn't change, but the data rate on the downstream link keeps increasing.

In practice, a traffic flow may not follow the following two assumptions that most bandwidth measurement tools make: (1) FIFO queuing is adopted at all routers along a path, and (2) a traffic flow follows a single routing path between the two end hosts. If the communication path between two hosts varies significantly in a short space of time, e.g. flow splitting and merging incur message out-of-order, we may see unexpected fluctuations in RTT. To address this, we adopt the concept of *time centroid*.

Assume a TCP RST stream, generated at a certain rate, contains $n + 1$ TCP RST messages: $m_0, m_1, m_2, \dots, m_n$. Let $t_i (i = 0, \dots, n)$ denote the absolute time stamp of TCP RST message m_i . Hence, $\Delta t_i = t_i - t_0$ is the relative time stamp of m_i from the time instance of the first TCP RST message m_0 . In addition, Δt_i also represents the offset of TCP RST message m_i from the starting point of the TCP RST stream.

Given any particular sequence of relative time stamps of TCP RST messages $\Delta t_0, \Delta t_1, \Delta t_2, \dots, \Delta t_n$ and a random interval length T ($T > 0$ and $T \ll t_n - t_0$), $\Delta' t_i = \Delta t_i \bmod T$ represents TCP RST message m_i 's offset from the starting point of its interval. Further, $\Delta' t_i$ is approximately

uniformly distributed in the range $(0, T)$. For example, Figure 2 shows the empirical distributions of the remainders of modulo 1000 operations over uniformly, normally and exponentially distributed random variables respectively. As we can see from Figure 2, the remainder of modulo operation over random variables of different distributions is approximately uniformly distributed. Consequently, for a TCP RST stream with sufficiently large number of TCP RST messages, at any interval length T which has $T > 0$ and $T \ll t_n - t_0$, the relative positions of TCP RST messages within their respective intervals ($\Delta' t_i$) are uniformly distributed.

Since a TCP stream experiences queuing delays when sending rate of the TCP stream goes above maximum bandwidth, estimating bandwidth is to determine whether a TCP stream experiences queuing delays. Hence, we define the centroid of a time interval as follow:

$$cent(T_j) = \frac{1}{n_j} \sum \Delta' t_i \quad (1)$$

where T_j is the j th time interval and $j = 1, \dots, \frac{\Delta t_n}{T}$; n_j is the number of TCP RST messages in the j th time interval and $\sum_{j=1}^{\frac{\Delta t_n}{T}} n_j = n$. Furthermore, $\Delta' t_i$ ranges from 0 to T . In case there is no TCP RST messages in the interval T_j , we define $cent(T_j)$ to be $\frac{T}{2}$. Extending the concept of centroid to a complete TCP RST stream, we then have

$$cent = \frac{T}{\Delta t_n} \sum_{j=1}^{\frac{\Delta t_n}{T}} cent(T_j) \quad (2)$$

Due to the fact that each interval T_j is uniformly distributed when queuing delays do not occur and a TCP stream follows a single routing path, the centroid of any time interval is approximately in the middle of the interval, i.e. $cent(T_j) = \frac{T}{2}$. Furthermore, the centroid of a complete TCP RST stream is equal to $\frac{T}{2}$. This property is stated and proved as follows.

Theorem 1: If the sending rate of a TCP ACK stream R_0 is less than maximum bandwidth A (i.e. $R_0 \leq A$), then $cent = \frac{T}{2}$.

Proof: Suppose that t_k is the absolute time stamp of TCP ACK message k and δ_k is the RTT of the TCP ACK message. Thus, the arrival time of the corresponding TCP RST message $t'_k = t_k + \delta_k$. When $R_0 \leq A$, $\delta_k = d_0$ for $k \in 1, 2, \dots$. Since the sending interval for back-to-back TCP ACK messages is $\tau = t_{k+1} - t_k$, the arrival interval for back-to-back TCP RST messages is $t'_{k+1} - t'_k = t_{k+1} + \delta_{k+1} - t_k - \delta_k = \tau$. Furthermore, we assume time interval T . Over the interval $[t'_k, t'_k + T)$, n TCP RST messages arrive (see Figure 3(a)) and so,

$$\begin{aligned} cent(T) &= \frac{1}{n} \cdot \sum_{j=0}^{n-1} t'_{k+j} = \frac{1}{n} \cdot \frac{n \cdot (n-1)}{2} \cdot \tau \\ &= \frac{(n-1) \cdot \tau}{2} = \frac{T}{2} \end{aligned} \quad (3)$$

²We tune the value of T and make Δt_n divisible by T .

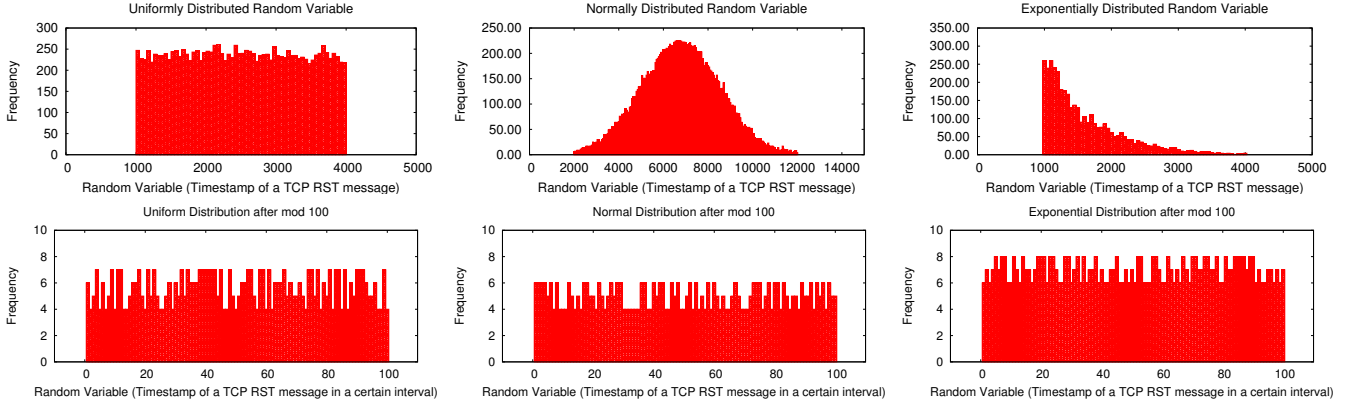


Fig. 2. Remainder Distribution of modulo operation over uniformly, normally, exponentially distributed random variable.

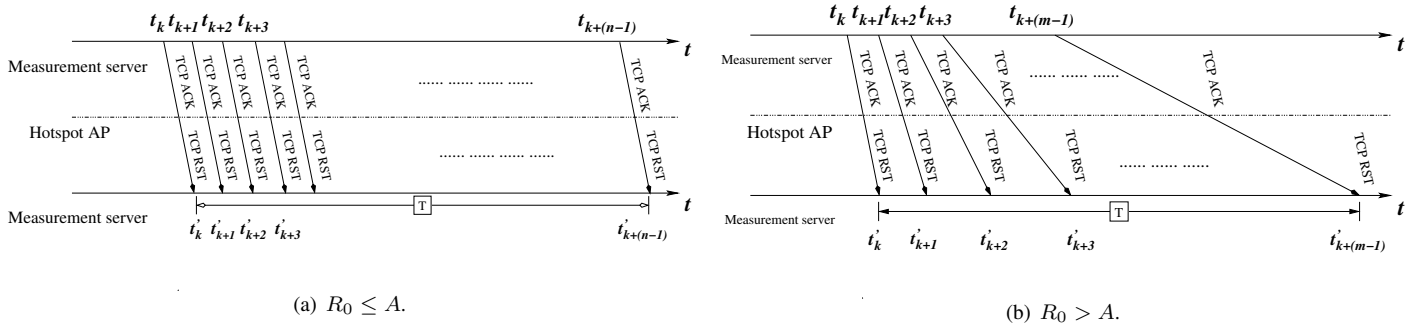


Fig. 3. A bandwidth measurement server in the cloud sends a TCP ACK stream to a hotspot AP, and then the hotspot AP responds back to the server with a TCP RST stream.

Consequently, the centroid of a TCP RST stream is

$$cent = \frac{N \cdot cent(T)}{N} = \frac{T}{2} \quad (4)$$

where N is the number of interval in a TCP RST stream. ■

Recall the assumption that we makes in Theorem 1: a TCP stream follows a single routing path. However, this assumption is not always true in a real network [19]. A TCP stream may be split or merged, which causes slight transmission delays are imposed upon some packets in the TCP stream. The following theoretical proof illustrates that the centroid of a stream approximately remains $\frac{T}{2}$ even though stream splitting and merging happen.

Theorem 2: When the sending rate of a TCP ACK stream R_0 is less than maximum bandwidth A (i.e. $R_0 \leq A$), stream splitting and merging will not skew the centroid of a TCP RST stream.

Proof: Suppose the centroid of an interval fluctuates, the centroid of the interval becomes $\frac{T}{2} + o$ where o is the offset of the expected centroid. Then the centroid of a TCP RST stream is

$$cent = \frac{T}{2} + \frac{T}{\Delta t_n} \cdot o \quad (5)$$

If $\frac{\Delta t_n}{T}$ is large enough, the value of $\frac{T}{\Delta t_n} \cdot o$ approaches zero. ■

Intuitively, when a bandwidth measurement server saturates either upstream or downstream channel of a hotspot AP, a

queuing delay is imposed upon a TCP stream. Therefore, there is an obvious centroid offset for a TCP RST stream. This property is stated and proved formally as follows.

Theorem 3: When the sending rate of a TCP ACK stream R_0 is greater than maximum bandwidth A (i.e. $R_0 > A$), the centroid of a TCP RST stream skews from $\frac{T}{2}$ to $\frac{T}{2} + d$, where d is an offset of the centroid.

Proof: In this case, the arrival time t'_k for TCP RST message k is the sum of the absolute time stamp of corresponding TCP ACK message k and RTT of the TCP ACK δ_k , i.e. $t'_k = t_k + \delta_k$ where t_k is the absolute time stamp of corresponding TCP ACK message k (see Figure 3(b)). Different from Theorem 1 where δ_k is a constant (i.e. $\delta_k = d_0$ for $k \in 1, 2, \dots$), δ_k is a variable when $R_0 > A$, since each TCP message may experience a variable queuing delay. Therefore, the arrival time for back-to-back TCP RST messages is denoted as follow

$$\begin{aligned} t'_{k+1} - t'_k &= t_{k+1} + \delta_{k+1} - t_k - \delta_k \\ &= \tau + (\delta_{k+1} - \delta_k) = \tau + \alpha_k \end{aligned} \quad (6)$$

where $\tau = t_{k+1} - t_k$ and $\alpha_k = \delta_{k+1} - \delta_k$ with $\alpha_k > 0$. Over the interval $[t'_k, t'_k + T)$, the centroid for m TCP RST messages is

$$cent(T) = \frac{1}{m} \cdot \sum_{j=0}^{m-1} t'_{k+j}$$

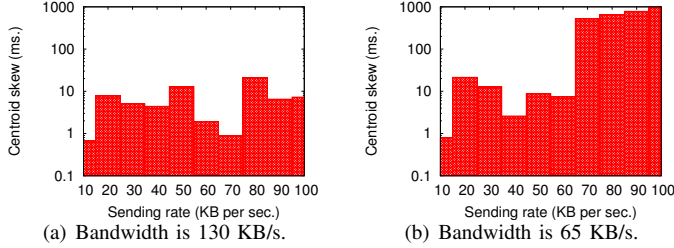


Fig. 4. Centroid skew variation with varying bandwidth.

$$\begin{aligned}
 &= \frac{1}{m} \cdot \left[\frac{m \cdot (m-1)}{2} \cdot \tau + \sum_{j=1}^{m-1} \alpha_j \right] \\
 &= \frac{\tau \cdot (m-1)}{2} + \frac{1}{m} \cdot \sum_{j=1}^{m-1} \alpha_j \\
 &= \frac{T}{2} + d_i
 \end{aligned} \tag{7}$$

where $d_i = \frac{1}{m} \cdot \sum_{j=1}^{m-1} \alpha_j$ and $d = \frac{\sum_{i=1}^N d_i}{N}$ (N denotes the number of intervals in a TCP RST stream). Consequently, the centroid of a TCP RST stream is

$$cent = \frac{N \cdot \frac{T}{2} + \sum_{i=1}^N d_i}{N} = \frac{T}{2} + d \tag{8}$$

To verify our theorems, we investigate two hotspot APs that have the same downstream bandwidth (450 KB/s) and different upstream bandwidths (130 KB/s and 65 KB/s). Two measurement servers in the Amazon's EC2 platform send TCP ACK packets of size 40 bytes to the two hotspot APs respectively with varying data rate, ranging from 10 KB/s to 100 KB/s. Figure 4(a) and 4(b) show the fluctuation of the centroid of the TCP RST stream around the time centroid for the AP with 130 KB/s and 65 KB/s upstream bandwidths respectively. We notice that this fluctuation is on both sides of the time centroid for 130 KB/s upstream link, where the sending rate is less than the bandwidth. On the other hand, for the 65 KB/s upstream link, when the sending rate is increased to 70 KB/s, the centroid of the TCP RST stream experiences a sudden jump. In addition, the centroid skew continues an upward trend beyond this sending rate. Therefore, to estimate the bandwidth, the measurement servers monitor the variation of the centroid of the TCP RST stream and locate the point of sudden jump.

To locate the point of sudden jump in centroid, we use a binary search algorithm. Initially, we set lower and upper bounds (R_{lower} and R_{upper}) and a termination condition ($R_{upper} - R_{lower} \leq R_0$, where R_0 is the measurement resolution, which is the nearest bounding range between the lower and upper bound). The specific values for R_0 , R_{lower} and R_{upper} are discussed in the next section. The search process iteratively adjusts the lower and upper bounds until $R_{upper} - R_{lower}$ is less than the measurement resolution R_0 . The measurement server sends a TCP ACK stream at

a certain rate $R(n) = (R_{upper} + R_{lower})/2$. If a centroid skew appears, the server adjusts the upper bound to $R_{upper} = (R_{upper} + R_{lower})/2$. On the other hand, if no centroid skew appears, the server adjusts the lower bound to $R_{lower} = (R_{upper} + R_{lower})/2$. This process is repeated until the termination condition is satisfied. The sending rate at this time is the estimated bandwidth.

IV. IMPLEMENTATION IN THE CLOUD

To estimate hotspot AP backhaul bandwidth on a large scale, we have implemented our bandwidth estimation technique in the Amazon EC2 platform [1]. The current Amazon EC2 platform offers 250 Mbps bandwidth for each instance (virtual machine). As the number of APs increases, two resources become limited, computing and bandwidth. Since cloud allows us to add new instances as needed, we can ensure that there are enough compute resources to perform bandwidth measurement even when the number of APs is significantly large. However, a concern is "will the outbound bandwidth of multiple instances (between the cloud and the Internet) become a bottleneck if thousands of instances perform the measurement?" This situation does not arise in our system because of the following three reasons.

First, we note that the outbound bandwidth of an instance is far larger than the outbound and inbound bandwidth of a hotspot AP (250 Mbps vs. 8 Mbps). Also, the bandwidth of the intermediate path between the cloud and hotspot APs are typically several hundreds of Gigabit per second [11]. As a result, the bandwidth component mounted on an instance is capable of estimating hotspot AP backhaul bandwidth even when thousands of instances perform bandwidth measurement simultaneously. Second, our observation is that bandwidth measurement of different APs in different geographical regions do not share same paths. For example, we measured the bandwidth of two APs, one in New York region and the other in Denver region. We noted that the paths which the measurement probes pass through did not share any links. We repeated this experiment for several other APs located in Quebec, California, Texas, etc. and observed the same property. Even when the APs are deployed in a same geographical region, the measurement probes still first saturate hotspot AP backhaul bandwidth, even though measurement probes share a few hops between the cloud and hotspot APs. This is because of the vast difference between the cloud bandwidth and AP backhaul bandwidth. For example, measuring 50 APs simultaneously from 50 instances requires 400 Mbps bandwidth ($50 \times 8Mbps$), which is far less than the bandwidth between the cloud and hotspot APs. Finally, bandwidth measurement of different APs from different instances does not happen at the same time because of measurement sampling rate. For example, assume measuring a hotspot AP takes 20 seconds and we need to measure the bandwidths of 900 APs within 30 minutes. One instance can then theoretically measure bandwidths of 90 ($1800seconds \div 20seconds$) APs, and we only need 10 instances to perform bandwidth measurement. This means only 10 measurements will be progressing simultaneously.

Our bandwidth estimation in the cloud is implemented using three components: the hardware infrastructure, a distributed framework and a bandwidth estimation component. The hardware infrastructure serves as the underpinning of the cloud, providing support for computing, networking and storage. EC2 is employed in our implementation as the hardware infrastructure. We harness 11 medium High-CPU instances (one instance serves as master, the other ten serve as slaves), each of which serves as a measurement server in the cloud. Each instance is a 32-bit platform with 5 EC2 Compute Units³ (2 virtual cores with 2.5 EC2 Compute Units each), 1.7 GB of Memory, 350 GB of local instance storage and 250 Mbps network capability [2]. To organize and manage these computing resources, we chose Hadoop [3] to serve as our distributed framework. Finally, the bandwidth estimation component runs on each instance.

We first assign measurement tasks to instances. In our prototype implementation, measurement tasks are described in a set of text files. Each file contains the IP addresses of hotspot APs and corresponding measurement parameters, such as measurement sampling rate and measurement resolution. The master instance in the cloud (Hadoop’s JobTracker) uniformly assigns the text files to those slave instances in the cloud (Hadoop’s TaskTrackers). Following the description in the text file that the master instance assigns, a Hadoop’s TaskTracker runs the bandwidth measurement component, collects AP backhaul bandwidth, and outputs the estimation results (bandwidth and latency) to another text file. Since output files are stored in Hadoop Distributed File System (HDFS), the Hadoop’s JobTracker is able to access these output files, incorporate them into a measurement report, and store the report into a centralized database. While this implementation is relatively straightforward, we encountered several important practical issues that we had to take care of:

(1) Automatic task assignment in Hadoop platform is dependent upon the size of the input file. By default, HDFS cuts a huge file into 128 MB chunks. Each chunk is automatically assigned to one TaskTracker. Unfortunately, we cannot utilize this automatic task assignment. Recall that the purpose of our system is to estimate hotspot AP backhaul bandwidth. Assume that tens of thousands of measurement tasks are described in a single file. The size of the file is at most several megabytes. Automatic task assignment will assign all measurement tasks to one TaskTracker in this case. To address this problem, we manually divide all the measurement tasks into a set of files. Thus, Hadoop platform can treat these files as separate HDFS chunks and distribute the files to multiple TaskTrackers. As part of our future work, we will automate this task assignment process.

(2) Since our measurement tasks are performed by instances in the cloud, it is possible that multiple instances share a single physical machine. Hence, instances may experience suspend and resume. Once an instance is suspended, the measurement

component running on the instance will be interrupted until the instance resumes. In practice, we observe that inter-instance suspension times for EC2’s small default instances are large enough to invalidate our bandwidth estimation results, whereas medium High-CPU instances have much shorter suspension times. Consequently, we chose medium High-CPU instances, rather than the small default ones.

(3) The NICs (Network Interface Cards) in EC2 instances are also virtual devices. Therefore, the driver and the network protocol stack in EC2 might be slightly different from the driver and network stack in physical machines. Based on our experiment, we observe that a virtual NIC driver does not timestamp incoming packets. This can cause the incoming packets stay at the system buffer for a long time without time stamps. To address this problem, we implement dual threads – one thread for sending outgoing TCP ACK packets and the other for receiving incoming TCP RST packets. This enables us to timestamp each incoming packet instantly and precisely.

(4) To enhance reliability, some environment parameters of Hadoop platform must be set properly. In particular, *mapred.task.timeout* should be set to 300, indicating that a measurement task should be canceled by the TaskTracker if it cannot output measurement results within 300 seconds. Note that a measurement task contains at most five APs in our current deployment, and estimating bandwidth of one AP takes at most 60 seconds. In addition, we further set *setNumMapTasks* to 1. It indicates that at most one measurement task runs on a TaskTracker at any point in time. This is because if multiple measurement tasks run on one instance simultaneously, their measurement timings may interfere with one another.

V. EVALUATION

A. Testbed

We have deployed two testbeds to experiment with and evaluate our bandwidth estimation component. The first testbed – a controlled indoor testbed – is deployed in McGill University. It is used for evaluating the bandwidth estimation component in terms of accuracy, latency and intrusiveness. It consists of a Cisco Aironet 1131 AG series wireless AP, a Linux box and a Dell OptiPlex 980 Mini Tower. The wireless AP is used as a hotspot AP in our experiments and a bandwidth estimation component running on the Mini Tower is used to perform the bandwidth estimation task. To emulate an asymmetric connection, we used the Linux box as a traffic shaper. It bridges the wireless AP and the Mini Tower. The traffic passing through the Linux box is shaped using the Linux bandwidth (tbf) and latency (netem) shaping modules.

The second testbed is deployed in Colorado region, which contains 50 accessible AT&T commercial hotspot APs in Denver, CO area and 11 instances (virtual machines) running on the Amazon EC2 platform. To ensure our measurement probes to arrive at these AT&T hotspot APs without experiencing packet drop-off, we configure these hotspot APs’ firewall IP access control lists (ACL). This testbed is used for evaluating our bandwidth estimation component in real broadband networks.

³One EC2 Compute Unit equals 1.0-1.2 GHz 2007 Opteron or 2007 Xeon processor.

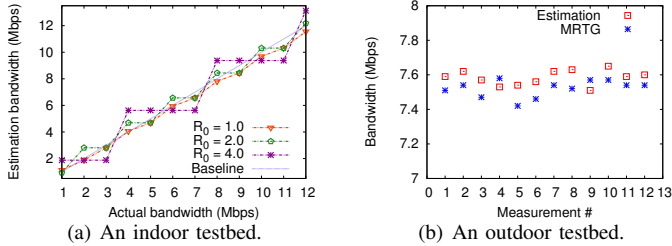


Fig. 5. Bandwidth estimation in different testbeds.

B. Evaluation in controlled settings

1) *Accuracy*: By using the bandwidth shaping module (tbf), we emulate different hotspot AP backhaul bandwidth. In this experiment, we set up upstream bandwidth of the AP to 500 Kbps and vary downstream bandwidth of the AP from 1 Mbps to 12 Mbps. The bandwidth estimation component running on the Mini Tower sends probes to estimate downstream bandwidth with different measurement resolutions. Figure 5(a) plots our estimated bandwidth. Overall, higher the measurement resolution is, less accurate our estimation result is.

2) *Latency*: The Internet is highly dynamic in terms of transmission latency, bandwidth between the cloud and a hotspot AP, etc. In general, there is no way to predict bandwidth estimation latency. We examine several factors that cause bandwidth estimation latency variation over our indoor testbed. By using latency shaping module (*netem*) in our indoor testbed, we emulate different transmission delays between the cloud and a hotspot AP. In one set of experiments, we measure the upstream bandwidth by setting upstream bandwidth equal to 3.5 Mbps, and in another set of experiments, we measure the downstream bandwidth by setting downstream bandwidth equal to 3.5 Mbps. Figure 6 shows the estimation latency for estimating upstream and downstream bandwidths as a function of transmission delay between the measurement server and the hotspot AP varies from these two sets of experiments. The first observation here is that the estimation latencies are reasonably low, less than 27 seconds for round trip times as high as 150 ms. Second, we notice that the measurement latency shows approximately a linear increase with respect to increase in RTT. Based on this, we can say that the measurement latency will remain reasonably low even when the RTT goes beyond 150 ms.

Finally, we notice that measurement latency is significantly different for upstream bandwidth estimation and downstream bandwidth estimation. The main reason for this is that the TCP ACK packets that the instance sends out have different packet sizes while measuring upstream and downstream bandwidths. Recall that the instance uses 40-byte ACK packets to estimate upstream bandwidth but larger ACK packets ($\gg 40$ bytes) to estimate downstream bandwidth. Thus it takes much longer to send the same number of ACK packets while measuring downstream bandwidth than upstream bandwidth.

To understand the impact of packet size on bandwidth estimation latency, we conducted another set of experiments.

In these experiments, the downstream bandwidth was set to three different values: 3.5 Mbps, 5.5 Mbps and 7.5 Mbps. By using different size of TCP ACK packets to estimate downstream bandwidth in our indoor testbed with a fixed RTT, we can observe in Figure 7 that the bandwidth estimation latency shows a linear increasing trend. The reason is that less delays are involved when an instance sends out a TCP ACK stream at a certain data rate using smaller TCP ACK packets. Another interesting observation in Figure 7 is that lower bandwidth shows higher estimation latency when an instance uses same size of TCP ACK packets to estimate bandwidth. The reason for this is quite straightforward – an instance spends less time in sending the same number of TCP ACK packets if these packets are transmitted through a high-speed link. This explanation is also validated in Figure 8.

Finally, we examine the effect of measurement resolution on bandwidth estimation latency (See Figure 8). In this experiment, we vary measurement resolution from 0.1 to 4.0. Note that the estimation latency exponentially decreases as resolution value increases. To understand this, recall that measurement resolution is part of the termination condition of our bandwidth estimation. Since bandwidth estimation is the process of a binary search, larger the resolution is, faster the search process will terminate. Therefore, we can easily reduce bandwidth estimation latency by choosing a greater value of measurement resolution. Intuition suggests that choosing a greater value of resolution, however, may harm the accuracy of our bandwidth estimation. In order to balance this tradeoff between estimation accuracy and latency, the prototype implementation of *xylophone* uses the configuration – *measurement resolution* = 1.5. We exclude the selection of resolution value from discussion. For more information, interested readers could refer to [23].

3) *Intrusiveness*: Since bandwidth estimation is based on the concept of self-induced congestion, i.e. an instance saturates hotspot AP backhaul bandwidth in the process of the bandwidth estimation, an important question is how does it impact the performance of ongoing user computations? For example, does it result in increase in transmission delays for hotspot subscribers? To quantitatively investigate this potential impact on transmission delays, we conducted an experiment in our indoor testbed. A client uses *hping2* to generate a TCP packet every 100 milliseconds through the AP we deployed in our testbed. Over this period of time, the instance probes the AP backhaul bandwidth. Figure 10 shows the transmission delays for some TCP packets from the client side increases. Furthermore, we can see that overall transmission delays show an obvious increasing trend when the TCP stream from the client side passes through a low-bandwidth link. The reason is that when TCP packets are buffered in a queue, lower bandwidth routers need longer time to empty the packet buffered in the queue. To examine the overall impact of bandwidth estimation on a TCP stream sent from a client side, we investigate what percentage of TCP packets from the client side experience longer transmission delays. Figure 11 indicates that only 10%–20% TCP packets generated from

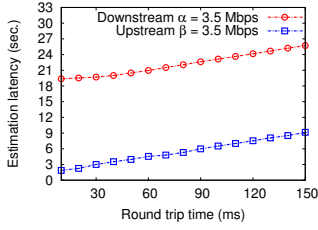


Fig. 6. Measurement latency at varying transmission delays.

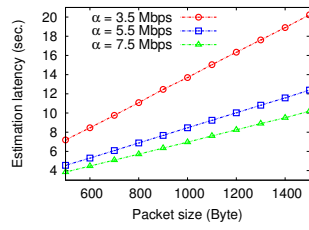


Fig. 7. Measurement latency at varying size of a TCP ACK packet.

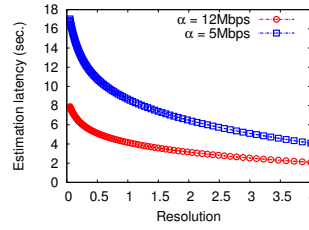


Fig. 8. Measurement latency at different resolutions.

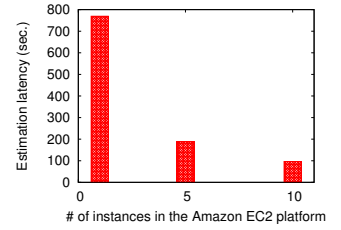


Fig. 9. *Xylophone* scalability in terms of bandwidth estimation.

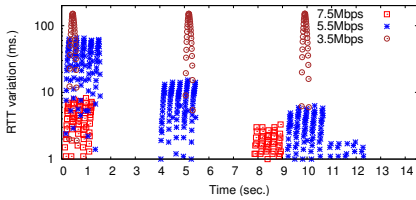


Fig. 10. RTT variation during bandwidth estimation.

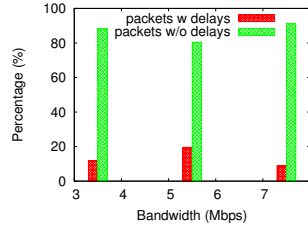


Fig. 11. The impact of bandwidth estimation on crossing traffic.

| | w/o est. | w est. | var. |
|------|-------------|-------------|--------|
| FTP | 57.668 sec. | 64.260 sec. | 11.43% |
| HTTP | 1.625 sec. | 1.922 sec. | 18.31% |

Fig. 12. The impact of bandwidth estimation on HTTP and FTP applications.

the client side experience longer transmission delays. The reasons for this relatively lower percentages are: (1) bandwidth estimation does not cause a persistent increase in queue size because a probe stream is never sent before the previous probe stream has been acknowledged; and (2) the process of bandwidth estimation is a process of binary search, and therefore, not every probe stream will saturate hotspot AP backhaul bandwidth.

C. Evaluation in the wild

1) *Accuracy*: We have experimented with measuring the bandwidths of several in-use commercial hotspot APs. Since these APs were used by hotspot clients while we performed the experiment, we have no way of knowing the actual hotspot AP backhaul bandwidths that we can compare with our measurements. Instead, we compare our measurement results with bandwidth measured using another popular tool, MRTG[4]. We deploy a Linux box with MRTG installation between an AT&T hotspot AP and the AT&T broadband infrastructure. MRTG measures the hotspot AP backhaul bandwidth by monitoring incoming and outgoing traffic within a 5-minute time slot. Notice that bandwidth estimation component of *xylophone* takes far less estimation latency. To compare these short-term estimates with MRTG measurement, we continuously perform bandwidth estimation from an instance in the cloud for 5 minutes. Based on these measurements over a 5-minute time slot, we calculate an average 5-minute-bandwidth, \bar{R} as follows:

$$\bar{R} = \sum_{i=1}^K \frac{T_i}{5} \times R_i \quad (9)$$

where K is the number of times the bandwidth is estimated in the 5-minute interval; T_i and R_i are the estimation latency and the estimated bandwidth respectively in the i^{th} time during the 5-minute interval.

Figure 5(b) shows the bandwidth estimated using MRTG during a 5-minute interval and the average 5-minute bandwidth during the same 5-minute interval for 12 different 5-minute intervals for one in-use commercial hotspot AP. The main observation is that *xylophone*'s bandwidth estimation component is able to accurately measure AT&T hotspot AP backhaul bandwidth.

2) *Scalability*: To evaluate scalability, we use our implementation in the cloud. We first perform our measurement task (i.e. estimating hotspot AP backhaul bandwidth of 50 AT&T hotspot APs in Denver and Boulder region) from a single instance. Then, we increase the number of instances and evenly divide bandwidth measurement tasks of 50 AT&T hotspot APs. Figure 9 shows the latency of measuring upstream and downstream bandwidths of all 50 APs for three different configurations: 1 instance, 5 instances and 10 instances. We notice that the measurement latency is significantly reduced when the number of instances in the cloud is increased from 1 to 10. This result is quite encouraging. In particular, with 10 instances, we can measure the bandwidths of all 50 APs in less than 100 seconds. So, if we have to update the bandwidths of APs every 30 minutes, 10 instances can manage approximately as many as 900 APs. The issue of how frequently the bandwidths should be measured is discussed in [23].

Another observation from Figure 9 is that though the number of instances are increased to 5 times and 10 times, the bandwidth estimation latency is not decreased by 5 times and 10 times. Two major factors contribute to this. First, the overhead of managing all instances in the cloud. The other is the RTTs between an instance and different hotspot APs are not equal, so the actual distribution of which set of APs is assigned to different instances can impact the latency.

3) *Intrusiveness*: To understand the intrusiveness of our *xylophone* system in a real environment, we further explore

the most common applications (i.e. web browsing and file downloading). The purpose of the experiment is to quantify the impact of bandwidth estimation component on hotspot subscribers' HTTP and FTP applications. In our first experiment, we repeatedly download a 48 MB file through an AT&T hotspot AP. As shown in Figure 12, when no bandwidth estimation is performed, the downloading delay is 57.668 seconds on average. When the bandwidth estimation component performs measurement, the downloading delay increases by 11%. Similarly, we also experiment with a HTTP application. By using a web browser to send HTTP requests to Google.com repeatedly, we found out that the HTTP request delay increases by 18% when bandwidth estimation is performed. An interesting observation is that the HTTP application experiences higher delay than the FTP application (18% vs. 11%). The reason is that we perform bandwidth estimation once per 30 minutes and each measurement takes approximately 20 seconds. When HTTP application is running, it's traffic is affected by the measurement probes from the beginning to the end. On the other hand, the FTP's traffic is only affected by the measurement probes for the first 20 seconds.

VI. DISCUSSION AND FUTURE WORK

This paper introduces a novel bandwidth estimation technique that is suitable for estimating the backhaul bandwidth of hotspot APs. The technique is highly scalable, and provides accurate results with low latency and intrusiveness.

(1) *Addressing firewall issue:* Behind hotspot APs, an ISP usually deploys a firewall to block unauthorized access. Therefore, our measurement probes (TCP ACK and RST) may be blocked. So, our deployment may require reconfiguring the access control list (ACL) of hotspot AP firewall. Since our application is deployed and installed by ISP (e.g. our *xylophone* prototype is running on AT&T platform), there is no concern for our *xylophone*'s bandwidth estimation component because ISPs can easily configure their firewall ACL and ensure that our measurement probes are not blocked.

(2) *Why not use MRTG for estimating the bandwidth?:* Though we use MRTG to compare our estimate with MRTG readings, it cannot be used to estimate AP backhaul bandwidth on a large scale. MRTG has to know the capacity of the link that it measures. Unfortunately, the capacities of the links that different hotspot APs use are quite different. Therefore, we will have to measure each hotspot AP's backhaul capacity first. Also, each MRTG instance has to run on a separate machine. So, a large number of MRTG machines (one for each AP) are needed. This is not a scalable solution.

(3) *Extending bandwidth estimation technique:* Though our bandwidth estimation technique is designed specifically for asymmetric links, it can be extended to be a general bandwidth measurement tool that can run on any client-side devices and measure any type of link. One solution is to substitute TCP ACK/RST with general TCP request/response. Similar to Abget, the extended bandwidth estimation will then rely on HTTP hosts. Different from Abget, the extended bandwidth

estimation can be utilized in non-FIFO scheduling networks. We are in the process of extending our bandwidth estimation technique to such a client-side measurement tool.

ACKNOWLEDGMENT

We would like to thank the anonymous reviewers for comments on earlier versions of this paper. Portions of this work were supported by NSF Grant DBI-0754832, NSERC Discovery Grant 341823-07 and NSERC Strategic Grant STPGP 364910-08.

REFERENCES

- [1] <http://aws.amazon.com/ec2/>.
- [2] http://en.wikipedia.org/wiki/amazon_elastic_compute_cloud/.
- [3] <http://en.wikipedia.org/wiki/hadoop>.
- [4] <http://oss.oetiker.ch/mrtg/index.en.html>.
- [5] <http://v4.jiwire.com/search-hotspot-locations.htm>.
- [6] <http://www.att.com/gen/general?pid=5949>.
- [7] <http://www.electricpig.co.uk/2010/02/01/wi-fi-hotspots-slow-speeds-hiding-behind-the-scenes/>.
- [8] <http://www.planet-lab.org/>.
- [9] Demetres Antoniadis, Manos Athanatos, Antonis Papadogiannakis, Evangelos P. Markatos, and Constantine Dovrolis. Available bandwidth measurement as simple as running wget. In *Proc. of PAM*, 2006.
- [10] Daniele Croce, Marco Mellia, and Emilio Leonardi. The quest for bandwidth estimation techniques for large-scale distributed systems. In *ACM SIGMETRICS Performance Evaluation Review*, pages 20–25, 2010.
- [11] Kensuke Fukuda, Kenjiro Cho, and Hiroshi Esaki. The impact of residential broadband traffic on japanese isp backbones. In *ACM SIGCOMM Computer Communication Review*, pages 15–22, 2005.
- [12] Ningning Hu and Peter Steenkiste. Evaluation and characterization of available bandwidth probing techniques. *IEEE JOURNAL ON SELECTED AREAS IN COMMUNICATIONS*, 21(6):879–894, 2003.
- [13] Manish Jain and Constantinos Dovrolis. End-to-end available bandwidth: measurement methodology, dynamics, and relation with tcp throughput. *IEEE/ACM TON*, pages 537–549, 2003.
- [14] Srikanth Kandula, Kate Ching-Ju Lin, Tural Badir Khanli, and Dina Katabi. Fatvap: Aggregating ap backhaul bandwidth. In *Proc. of USENIX NSDI*, 2008.
- [15] Seong-Ryong Kang and Dmitri Loguinov. Imr-pathload: Robust available bandwidth estimation under end-host interrupt delay. In *Proc. of PAM*, 2008.
- [16] George Kola and Mary K. Vernon. Quickprobe: available bandwidth estimation in two roundtrips. In *SIGMETRICS Performance Evaluation Review*, pages 359–360, 2006.
- [17] Karthik Lakshminarayanan and Venkata N. Padmanabhan. Some findings on the network performance of broadband hosts. In *Proc. of IMC*, 2003.
- [18] Karthik Lakshminarayanan, Venkata N. Padmanabhan, and Jitendra Padhye. Bandwidth estimation in broadband access networks. In *Proc. of IMC*, 2004.
- [19] Xiapu Luo, Edmond W.W. Chan, and Rocky K.C. Chang. Design and implementation of tcp data probes for reliable and metric-rich network path monitoring. In *Proc. of USENIX Annual Technical Conference*, 2009.
- [20] Anthony J. Nicholson, Yatin Chawathe, Mike Y. Chen, Brian D. Noble, and David Wetherall. Improved access point selection. In *Proc. of ACM MOBISYS*, 2006.
- [21] Pavlos Papageorge, Justin McCann, and Michael Hicks. Passive aggressive measurement with mgrp. In *Proc. of ACM SIGCOMM*, 2009.
- [22] Jacob Strauss, Dina Katabi, and Frans Kaashoek. A measurement study of available bandwidth estimation tools. In *Proc. of ACM IMC*, pages 39–44, 2003.
- [23] Xinyu Xing, Jianxun Dang, Shivakant Mishra, and Xue Liu. The design, implementation and performance evaluation of xylophone: A scalable qos-enabled wifi locator service. In *Technical Report CU-CS-1072-10*, 2010.
- [24] Xinyu Xing and Shivakant Mishra. Where is the tight link in a home wireless broadband environment? In *Proc. of IEEE/ACM MASCOTS*, 2009.