FABMon: Enabling Fast and Accurate Network Available Bandwidth Estimation

Tao Jin*†, Weichao Li†, Qing Li†, Qianyi Huang‡†, Yong Jiang*†, Shutao Xia*†
* Tsinghua Shenzhen International Graduate School, Shenzhen, China
† Peng Cheng Laboratory, Shenzhen, China
‡ Southern University of Science and Technology, Shenzhen, China

2022 IEEE/ACM 30th International Symposium on Quality of Service (IWQoS) | 978-1-6654-6824-4/22/$31.00 ©2022 IEEE | DOI: 10.1109/IWQoS54832.2022.9812891

***Abstract*—Characterizing the end-to-end network available bandwidth (ABW) is an important but challenging task. Although a number of ABW estimation tools have been introduced over the past two decades, applying them to the real-world networks is still difficult because of the biased results, heavy load, and long measurement time. In this paper, we propose a novel Burst Queue Recovery (BQR) model to infer the ABW. BQR first induces an instant network congestion and then observes the one-way delay (OWD) variation until the tight link recovers from the congestion. By correlating the OWDs with the queue length variation, BQR can calculate the ABW accurately. Compared to the traditional probe gap model (PGM) and probe rate model (PRM), our theoretical analysis and simulations show that BQR is more tolerant to the transient traffic burst and supports the scenarios with multiple congestible links. Based on the model, we build FABMon, a fast and accurate ABW estimation tool. Our experiments show that FABMon can measure ABW within 50 milliseconds, and achieve much more accurate measurement results than the existing tools with a very small volume of probe packets.**

I. Introduction

End-to-end available bandwidth (ABW) describes the maximum transmission capability of a network path during a period of time, which is widely used for many applications, such as video streaming [1], router selection [2], congestion control [3]. Therefore, it attracts considerable research interests. In the past two decades, many ABW estimation tools have emerged [4–19]. These actively-probing tools can be classified into Probe Gap Model (PGM) and Probe Rate Model (PRM), respectively. The PGM solutions employ the dispersion of the probe packets to estimate the ABW, while the PRM ones find the turning point of the sending rate that causes congestion.

The existing ABW estimation models are usually based on the assumption that the network is a *single-hop*, *lossless*, and *First-In, First Out* (FIFO) multiplexer of probing packets and cross traffic. However, today's Internet becomes more and more complex and cannot be modeled in such a simple way. When the network condition deviates from the assumption, the ABW estimations may be biased. In fact, the PGM has a high probability of underestimating the ABW [20, 21]. The PRM also has problems of inaccuracy and inefficiency. PRM-based tools send a very large volume of measurement traffic in terms of probe trains to find out the turning point of the sending rate that may cause network congestion either iteratively [4–6] or at predetermined rates [10, 13–19]. As the PRM usually needs to send trains at rates lower than the ABW, it is very inefficient and increases the measurement time. Moreover, a longer measurement time makes the model assumption of ideally constant cross traffic unguaranteed. As a result, PRM can produce inaccurate ABW estimations as well as induce high costs in both traffic and time.

In this paper, we try to tackle the fundamental limitations of the state-of-the-art ABW estimation tools. In particular, we aim to design a robust ABW estimation strategy that is adaptive to today's Internet environment with non-constant cross traffic and multiple congestible links[1]. To achieve this goal, we need to overcome the following challenges:

- **Practicability**. Unlike the unpractical assumptions of the PGM and PRM models, we need a new model that satisfies the real Internet scenarios, where multiple congestible links and dynamic cross traffic exist.
- **Robustness**. ABW measurements rely on accurate timestamps of probing packets, which can be affected by some operations of the system protocol stack. For example, the Interrupt Coalescing (IC), i.e. grouping multiple packets to a batch in a single interrupt, may delay the probes. These noises should be taken into account for robustness.

To this end, we first introduce the ***B**urst **Q**ueue **R**ecovery* (BQR) model. Unlike PGM and PRM respectively based on the information of probe dispersion and appropriate probe sending rate, BQR uses the one-way delay (OWD) of a network path to estimate its ABW. BQR sends a *loading train* to induce observable congestion, delicately controlled without packet loss. When the last packet of the loading train arrives at the tight link, the queue length of the tight link reaches its peak. The network congestion will be alleviated when the tight-link queue length decreases. We prove that ABW can be inferred from the queue length change with certain requirements. Since the queue length cannot be measured directly by the end, BQR instead utilizes a sparsely distributed train (named *inspection train*) to inspect the OWD variation for the ABW estimation.

Corresponding author: Qing Li, liq@pcl.ac.cn

978-1-6654-6824-4/22/$31.00 ©2022 IEEE

[1]Link l is a congestible link *iff* the maximum arrival rate of the probe traffic is larger than its ABW.

BQR outperforms the existing ABW models in two respects. First, compared to PGM, BQR can work under networks with multiple congestible links. Second, BQR saves the probe budget and shortens the measurement time by removing some inefficient low-rate trains, compared to PRM. BQR measures the sum of the cross traffic during the link recovery phase, which is more robust to the traffic burst. Accordingly, BQR significantly improves the estimation accuracy. In short, BQR is compatible with the current Internet with multiple congestible links and dynamic cross traffic.

Based on BQR, we then implement the ABW estimation tool of (**F**ast and **A**ccurate **B**andwidth **Mon**itoring). We compare FABMon with other four tools, including pathload [5], Spruce [9], PTR [6], and ASSOLO [16]. The testbed experiments show that FABMon outperforms them in terms of higher accuracy, smaller probe budget, and shorter measurement time in simple (two congestible links) and complex (multiple links with variable traffic loads) network conditions. The evaluation with real-world traffic traces also confirms the performance of FABMon.

The contributions of the paper are summarized as follows:

- We propose a novel ABW estimation model, BQR, to improve the accuracy and robustness of ABW estimation. We perform theoretical analysis to prove the correctness of BQR. BQR achieves very low relative error no matter the distribution of the cross traffic is constant, Poisson, or Pareto.
- We implement FABMon based on BQR in Linux as one applicable tool. FABMon involves probe train scheduling, recovery assessment, and recovery locating algorithms. In FABMon, the possible system noises induced by timestamping, Context Switch (CS), and IC are mitigated by our delicate optimization. The code is shared at: https://github.com/godblessforhimself/BurstQueueRecovery.
- We conduct comprehensive experiments to evaluate the performance of FABMon. To ensure the reliability of testbed, we verify the stability of the timing source, then check the precision of the traffic generation tool, iPerf3. FABMon is then compared with four ABW tools in both simple and complex network conditions. To further prove the effectiveness of FABMon, we also use two real network traces for the validation.

II. Related Work

Active ABW measurement techniques can be divided into PGM and PRM by the information used for estimation: packet dispersion for PGM and saturation rate for PRM [9].

A. Segmented Linear Relation

Let v_{send} and v_{recv} represent the sending and receiving rates of the train, respectively. In TOPP [4], as long as the cross traffic is constant and stable, "$\frac{v_{send}}{v_{recv}}$-$v_{send}$" can be modeled as a segmented linear function as below:

$$\frac{v_{send}}{v_{recv}} = \alpha \cdot v_{send} + \beta \tag{II-A.1}$$

where α and β are constants for each segment. The segment number is determined by the number of congestible links in the path. Let the sending rate at the start of the ith segment be v_{i-1}. The sending rate in the first segment satisfies $0 < v_{send} < v_1$. There is no congestion at the first segment, so $\alpha = 0, \beta = 1$ for it. In the second segment, $v_1 < v_{send} < v_2$ and $\alpha = \frac{1}{C_t}, \beta = \frac{v_x}{C_t}$ where C_t is the capacity of the tight link and v_x is the cross-traffic rate at the tight link. In fact, v_1 is the path ABW.

B. Probe Gap Model (PGM)

Representative PGM-based ABW estimation tools include CPROBE [8], Spruce [9], IGI/PTR [6], traceband [7], and DietTOPP [11]. PGM tools use the relation that the packet dispersion is caused by the arrival cross traffic during the gap g_{in} :

$$(g_{out} - g_{in}) \cdot C_t = v_x \cdot g_{in} \tag{II-B.1}$$

where C_t is the link capacity, g_{in} is the arrival gap, g_{out} is the departure gap, v_x is the cross-traffic rate. Eq. (II-B.1) is actually a special case in Eq. (II-A.1) when $v_{send} = C_t$ and $v_1 < v_{send} < v_2$:

$$\frac{v_{send}}{v_{recv}} = \frac{v_{send}}{C_t} + \frac{v_x}{C_t} \tag{II-B.2}$$

Therefore, the application of the PGM relies on several assumptions: the sending rate and the receiving rate should be equal to the arrival rate and the departure rate at the tight link; the sending rate should be equal to the capacity of the tight link; the rate $v_2 > C_t$. Once one assumption is not true, the PGM will be biased. For example, the PGM-based tool overestimates the cross-traffic rate if there exists cross traffic at the non-tight link, as shown in § VI-B.

C. Probe Rate Model (PRM)

PRM-based tools do not conform to the single-bottleneck assumption and are therefore more generic than PGM-based ones [22]. We further divide them into *feedback-based* and *multi-rate* sub types. Feedback-based PRM tools, such as TOPP [4], pathload [5], and PTR [6], adjust the probe sending rate according to the feedback of its last trial from the receiver. Only when the sending rate is close enough to the ABW will it stop the iteration. Due to the iterative nature, they need a longer measurement time and produce a larger overhead than other tools. Multi-rate PRM tools include PathChirp [10], ASSOLO [16], Yaz [14], etc. They send a fixed number of probe trains at predetermined rates and employ optimized algorithms or machine learning techniques, with upper bounds of measurement time and overhead.

Since PRM-based tools need multiple trains, it inevitably results in the problem of inefficiency. Another issue of the PRM is that the Internet is dynamic and the ABW could vary for different trains. In § VI-A, we find the numbers of trains for pathload and PTR are 25-97 and 9-67 for one measurement, respectively. The duration of each train is $\approx$ 10 ms. However, as discussed in [23], traffic rates at small time scales (1-100 ms) can be uncorrelated. Our analysis on real network traces

confirms that the Internet traffic is bursty(from -100% to 100% deviations from the average for bigFlows and from -20% to 20% for CAIDA [24]) in 10 ms time granularity. Therefore, PRM-based tools, especially feedback-based ones, have poor performance in real-world deployment.

III. Burst Queue Recovery (BQR) Model

A. Metrics and Measurement Model

We consider a n-hop network path between the measuring and remote nodes. As shown in Fig. 1, there are $n-1$ intermediate nodes, assumed to be store-and-forward devices with FIFO queues. Hop H_i comprises the node and its outgoing link L_i with a capacity of C_i in bits/s, where $1 \leq i \leq n$. Let the cross-traffic rate on L_i be T_i, and the ABW for L_i be $A_i = C_i - T_i$. The *narrow link* of the path is defined as the link with the smallest capacity and the *tight link* is the link with the smallest ABW. If there are multiple links with same smallest ABWs, we consider the first as the tight link. The ABW of the path (A) is determined by the tight link L_k, i.e., $A = \min_i(A_i) = A_k$.

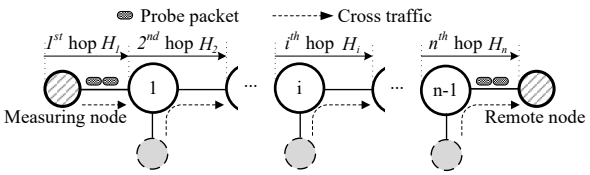


Fig. 1: The model of FABMon on a n-hop network path.

In the active ABW measurement paradigm, the measuring node injects a sequence of probe packets into the network. During the measurement phase, we assume that *the route does not change for the path*. Here we define m-hop traffic as the traffic that passes exactly m links in the measured path.

B. The BQR Model

BQR injects probes into the network in order to induce network congestion, and then observe the OWDs of the probes until the network recovers from congestion. Specifically, BQR injects an amount of P_1 traffic at a constant rate in the loading phase into the path. Due to the high rate of the loading train, the network, including the tight link, gets congested and the OWD increases. After P_1 is sent, BQR starts to send another P_2 traffic at a low rate in the inspection phase, for the purpose of monitoring the OWD. During the inspection phase, the queues of the tight link and other congestible links decrease, and finally the path OWD is back to the level before congestion. We define the *recovery time* as the time for a link from entering to exiting the congestion state. We find that the *recovery time* is in inversely proportional to the link ABW. As the tight link has the smallest ABW, *it has the longest recovery time, i.e., the OWD recovery time of the path.*

Let $t_i^{(0)}$ be the time when the loading train arrives at link L_i. Let $t_i^{(1)}$ be the time when the last packet of the loading train arrives at link L_i. Let $t_i^{(2)}$ be the time when the queue length of L_i first becomes zero after $t_i^{(0)}$. Denote the queue length at $t_i^{(0)}$, $t_i^{(1)}$ and $t_i^{(2)}$ by $Q_i^{(0)}$, $Q_i^{(1)}$ and $Q_i^{(2)}$, respectively. Ideally, the queue of L_i is increasing between $t_i^{(0)}$ and $Q_i^{(1)}$, and is decreasing between $Q_i^{(1)}$ and $Q_i^{(2)}$. Let $\Delta Q_i = Q_i^{(2)} - Q_i^{(0)}$ and $\Delta t_i = t_i^{(2)} - t_i^{(0)}$. We then have the first theorem:

Theorem III.1. *Let A_i be the ABW of link L_i, P be the volume of the packets that BQR has sent before $t_i^{(2)}$, then $A_i = \frac{P - \Delta Q_i}{\Delta t_i}$.*

Proof. Let $\overline{T_i}$ be the average cross-traffic rate at link L_i. Since the queue length variation ΔQ_i *equals to* the arrival traffic minus the departure traffic, we have

$$\Delta Q_i = \overline{T_i} \cdot \Delta t_i + P - C_i \cdot \Delta t_i \tag{III-B.1}$$

After transposition, we have

$$(C_i - \overline{T_i}) \cdot \Delta t_i = P - \Delta Q_i \tag{III-B.2}$$

By substituting $A_i = C_i - \overline{T_i}$ into the Eq. (III-B.2),

$$A_i = (P - \Delta Q_i)/\Delta t_i \tag{III-B.3}$$

□

Define $\Delta t = \max_i(\Delta t_i)$. Ideally, $Q_i^{(0)} = 0$ for all links. In practice, $Q_i^{(0)} > 0$ as there are bursts of traffic. Assume P is chosen so that $P \gg \max_i(Q_i^{(0)})$, then $\Delta t_i \approx \frac{P}{\Delta A_i}$, which means the link with the smallest ABW has the longest recovery time. So $\Delta t = \Delta t_k$, where t_k is the recovery time of the tight link. Thus, we have the ***BQR equation***:

$$\hat{A} = P/\Delta t \tag{III-B.4}$$

where $\hat{A}$ represents the ABW estimation.

In the following, we quantitatively analyze the influence of ΔQ_k on ABW estimation. The path OWD consists of the time of transmission, propagation, processing, and queuing along the path. Let $D(t)$ denote the OWD at time t, then:

$$D(t) = (\sum_i \frac{Q_i(t)}{C_i}) + \beta \tag{III-B.5}$$

where $Q_i(t)$ is the queue length of the link L_i at time t, C_i is the capacity of the link L_i, β is the rest of the delays which is relatively stable and can be considered as an intrinsic property of the path. We then have another theorem:

Theorem III.2. *The queue length change ΔQ_k of the tight link L_k can be bounded by the OWD variation and the capacity of the tight link (C_k).*

Proof. The queuing delay at the tight link (D_k) should be no more than the sum of the queuing delays of all links, i.e.:

$$D_k(t) \leq D(t) - \beta \tag{III-B.6}$$

Since $D_k(t) = \frac{Q_k(t)}{C_k}$, we have:

$$Q_k(t) \leq (D(t) - \beta) \cdot C_k \tag{III-B.7}$$

Let $\lambda_1 = D(t_1) - \beta$ and $\lambda_2 = D(t_2) - \beta$. Then we have:

$$0 \leq Q_k(t_1) \leq C_k \cdot \lambda_1 \tag{III-B.8}$$

$$0 \leq Q_k(t_2) \leq C_k \cdot \lambda_2 \tag{III-B.9}$$

Since $\Delta Q_k = Q_k(t_2) - Q_k(t_1)$:

$$-Q_k(t_1) \leq \Delta Q_k \leq Q_k(t_2) \tag{III-B.10}$$

The boundaries of ΔQ_k are:

$$-\lambda_1 \cdot C_k \leq \Delta Q_k \leq \lambda_2 \cdot C_k \tag{III-B.11}$$

The upper boundary of $|\Delta Q_k|$ is:

$$|\Delta Q_k| \leq \max(\lambda_1, \lambda_2) \cdot C_k \tag{III-B.12}$$

□

Theorem III.3. *The relative error of the ABW estimation can be bounded; P can be adjusted to achieve a certain relative error limit.*

Proof. Let A_k represent the ABW of the tight link, $\hat{A}_k$ be the estimation of ABW, ΔA_k be the estimation error, $\eta = \frac{\Delta A_k}{A_k}$ which represents the relative estimation error:

$$A_k = (P - \Delta Q_k)/\Delta t_k \tag{III-B.13}$$

$$\hat{A}_k = P/\Delta t_k \tag{III-B.14}$$

$$\Delta A_k = \Delta Q_k/\Delta t_k \tag{III-B.15}$$

$$\eta = \Delta Q_k/(P - \Delta Q_k) \tag{III-B.16}$$

The relative error can be bounded with Eq. (III-B.11):

$$-\frac{C_k \cdot \lambda_1}{P + C_k \cdot \lambda_1} \leq \eta \leq \frac{C_k \cdot \lambda_2}{P - C_k \cdot \lambda_2} \tag{III-B.17}$$

Given certain relative error limit γ so that $|\eta| \leq \gamma$, P should satisfy the below requirement according to Eq. (III-B.16):

$$P \geq (1+\gamma)/\gamma \cdot |\Delta Q_k| \tag{III-B.18}$$

P can be adjusted as below to achieve limit γ according to Eq. (III-B.12) and Eq. (III-B.18):

$$P \geq (1+\gamma)/\gamma \cdot \max(\lambda_1, \lambda_2) \cdot C_k \tag{III-B.19}$$

□

In fact, we satisfy the above condition by controlling the volume of packets in the loading phase since P is larger than it. Specifically, when $\max(\lambda_1, \lambda_2) = 0.1\,\text{ms}$, $C_k = 1\,\text{Gbps}$ and $\gamma = 10\,\%$, the requirement is $P \geq 137.5\,\text{KB}$, so we send 92 MTU-sized packets in the loading phase; When $\max(\lambda_1, \lambda_2) = 1\,\text{ms}$ and C_k and γ are the same, the requirement is $P \geq 1.375\,\text{MB}$, where we send 917 MTU-sized packets.

In conclusion, BQR uses Eq. (III-B.4) to compute path ABW; Eq. (III-B.11) to compute the relative error bound; and Eq. (III-B.19) to guide the adjustment of the probe budget.

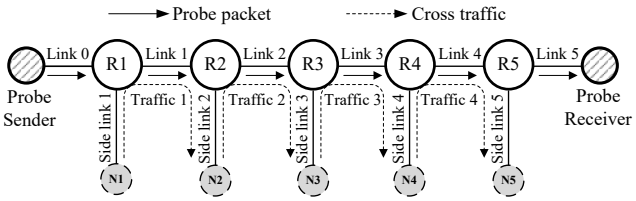


Fig. 2: Topology for ns-3 simulation.

TABLE I: Traffic rates of constant/Poisson/Pareto traffic at Link 3 (40 ms, 100 samples).

Type	10th percentile (Mbps)	90th percentile (Mbps)	Mean Rate (Mbps)
Constant	69.77	70.07	70.00
Poisson	64.18	75.66	69.51
Pareto	0.00	85.58	58.72

TABLE II: BQR ABW estimations, the real ABWs and REs for three types of traffic.

Traffic	$\hat{A}_T$(Mbps)	A(Mbps)	RE(%)
Constant	29.08	28.59	1.71
Poisson	38.06	37.72	0.90
Pareto	24.69	24.94	1.01

C. Simulation

We use NS3 to verify the theory of the BQR model. Fig. 2 shows the topology for the simulation. Our goal is to measure the ABW of the path from the probe sender to the probe receiver consisting of five links and five 1-hop cross traffic. The capacities are 100 Mbps for links in the path, and are 10 Gbps for the side links. Traffic rates from Links 1 to 5 are 50, 60, 70, 60, 50 Mbps respectively. Link 3 is the tight link.

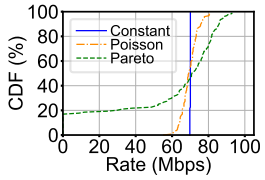
Three types of cross traffic are tested: constant, Poisson and Pareto ($\alpha = 1.5$). Their rate CDFs are shown in Fig. 3a (40 ms granularity). The 10th percentile, 90th percentile and mean rates at Link 3 are listed in Table I: constant traffic has no burstiness; Poisson traffic has a little burstiness, 8% bias from the target for the 10th percentile and 90th percentile; Pareto traffic is bursty with an average rate of 58.72 Mbps.

To emulate BQR, the sender sends 100 UDP packets (1500 Bytes) at 100 Mbps to induce congestion. After that, another 100 packets are sent at 10 Mbps to observe the OWD variation.

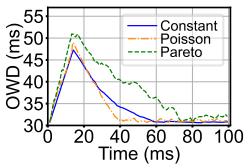
Fig. 3b shows the OWD during the measurement. BQR locates t_1 and t_2 in the OWD curves. It estimates the ABW by Eq. (III-B.4) where $\Delta t = t_2 - t_1$ and P is the volume of the packets sent within $[t_1, t_2]$. The ground truth of ABW is calculated as the capacity minus the average cross-traffic rate in $[t_1, t_2]$.

The ABW estimations from BQR, the ground-truth ABWs and the Relative Errors (REs) are shown in Table II, where RE is $\varepsilon = \frac{|\hat{A}_k - A|}{A} * 100\%$, A is the ground truth and $\hat{A}_k$ is the ABW estimation. In the simulation, the REs for all three types of traffic are low, i.e., 1.71%, 0.90% and 1.01% respectively.

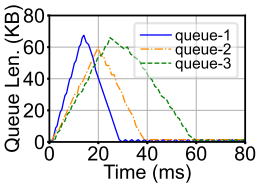
Fig. 3c, 3d, and 3e illustrate queue length variation over



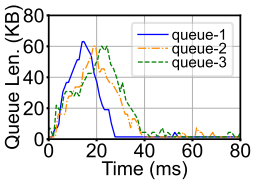
(a) CDF of traffic rates (40 ms)



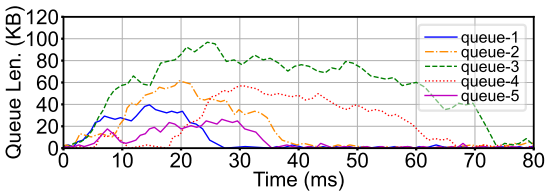
(b) "OWD-time"



(c) "queue len.-time" (constant)



(d) "queue len.-time" (Poisson)



(e) "queue length-time" curves of five queues for Pareto traffic

Fig. 3: NS3 Simulation Results

TABLE III: Symbols in Algorithms 1, 3 and 4

	Algorithm 1
n_l	the packet number of the loading train
s_l	the packet size of the loading train
r_l	the sending rate of the loading train
s_i	the packet size of the inspection train
n_a	the packet number of the main probes
n_b	the packet number of the auxiliary probes
a_m	the lower bound of the ABW
a_M	the upper bound of the ABW
g_m	the lower bound of the packet gap of the main probes
g_M	the upper bound of the packet gap of the main probes
g_n	the packet gap of the auxiliary probes
t_a	the planned sending times of all packets
N	# of all packets
	Algorithm 3
d	the OWDs of all packets
th_a	threshold above which the OWD is unrecovered
th_b	threshold to determine whether the OWD is not stable
m	the number of packets whose OWDs are stable
rec	whether the OWD has recovered
th_c	threshold below which the OWD is recovered
	Algorithm 4
t_b	the actual sending times of all packets
L	the lower bound of the recovery time
E	the estimation of the recovery time
U	the upper bound of the recovery time
rec_{id}	the smallest index of packet whose OWD is recovered

time of all congestible links for three types of traffic. The results of constant and Poisson traffic are close to piecewise linear. For Pareto, the curve is different. The reason is that the OWD variation is a linear combination of the queuing delays of all congestible links.

In summary, queues start to increase during the loading phase of BQR, and then gradually decrease to the level before congestion during the inspection phase. For all three types of traffic, the queue of the tight link, i.e., queue-3 has the longest recovery time, which determines the recovery time of the OWD. This is how and why BQR can infer the path ABW from the OWD precisely, even when the cross traffic is bursty.

IV. Design of FABMon

In this section, we present the detailed design of *FABMon*.

Probing mechanism. According to the BQR, FABMon is deployed on both the measuring and remote nodes. The measurement starts with a *loading phase*, where a certain number of packets (*loading train*) are sent at rate r_l to fill the queue of the tight link. In the subsequent *inspection phase*, two consecutive sets of probe packets (*inspection train*) are sent to detect the OWD change: *main probes* for calculating the exact ABW of the network path, and *auxiliary probes* for detecting the upper and lower bounds of the OWD and determining whether the network path has recovered from congestion to a stable state. Algorithm 1 is designed to compute the packet sending times of the loading and inspection trains. The loading train has a constant rate with a packet gap of $\frac{s_l}{r_l}$. The main probes in the inspection phase are a low-rate train. Algorithm 2 computes the sending times of the main probes, with the lower and upper bounds of the packet gap. It computes the next sending time and calls itself recursively. The auxiliary probes have a constant rate with a packet gap of g_n, determining whether the OWD has recovered stably.

Recovery assessment and location. The *RecoveryAssess* Algorithm (3) checks whether the OWD has recovered to the state before the loading train. If recovered, it outputs th_c, a threshold below which the OWD is considered recovered.

The *RecoveryLocate* Algorithm (4) finds the smallest-index packet whose OWD is $< th_c$ in the main probes. This packet is considered as *recovered* and the previous packet is *unrecovered*. Function *XvalueF*(x_a, y_a, x_b, y_b, y) in Algorithm 4 gives x so that points (x, y), (x_a, y_a) and (x_b, y_b) are collinear. We use *XvalueF* to compute the estimation of the recovery time E so that the unrecovered point $(L, d[rec_{id}-1])$, the recovered point $(U, d[rec_{id}])$ and the estimation point (E, th_c) are collinear. L and U are the lower and upper boundaries of the recovery time. They are used to compute the upper and lower boundaries of the ABW, respectively.

Measurement procedure of FABMon. After arranging the sending time of each packet as Algorithm 1), the sender module of FABMon sends the probe packets as planned. Once the receiver program receives all probe packets, Algorithm 3) is executed and generate two outputs: rec and th_c. If rec is true, Algorithm 4 is called to output the recovery time E and boundaries L, U, as well as the recovery index rec_{id}. Finally, the path ABW is estimated based on Eq. (III-B.4):

$$\hat{A} = (n_l \cdot s_l + rec_{id} \cdot s_i)/E \qquad \text{(IV-.1)}$$

Algorithm 1: Probe Train Design

Input: n_l, s_l, r_l, s_i, n_a, n_b, a_m, a_M, g_m, g_M, g_n, N
Output: t_a

$t_a \leftarrow []$
/* push_back($array$, $element$) adds $element$ at the end of $array$ */
for $(i \leftarrow 0; i < N; i \leftarrow i+1)$ **do**
  push_back(t_a, 0)
end
/* Loading phase: constant rate */
for $(i \leftarrow 0; i < n_l; i \leftarrow i+1)$ **do**
  $t_a[i] \leftarrow s_l \cdot i / r_l$
end
/* Inspection phase - Main probes: decreasing rate */
$P_1 \leftarrow s_l \cdot n_l, \quad P_2 \leftarrow s_l \cdot n_l + (n_a - 1) \cdot s_i$
$t_a[n_l] \leftarrow P_1/a_M, \quad t_a[n_l + n_a - 1] \leftarrow P_2/a_m$
ArrF(P_1, n_l, $n_l + n_a - 1$, a_M, a_m, $t_a[n_l]$, $t_a[n_l + n_a - 1]$, g_m, g_M, s_i, t_a)
/* inspection phase - Auxiliary probes: constant rate */
for $(i \leftarrow n_l + n_a; i < N; i \leftarrow i+1)$ **do**
  $t_a[i] \leftarrow t_a[i-1] + g_n$
end
return t_a

Algorithm 2: ArrangeFunc(ArrF for short)

/* Set the packet sending times of the main probes. */
function ArrF(p, i_l, i_r, a_l, a_r, t_l, t_r, g_m, g_M, s_i, t_a)
begin
  if $i_l + 1 \geq i_r$ **then**
    return
  $q \leftarrow (i_r - i_l)/(i_r - i_l + 1)$,
  $a_{temp} \leftarrow q \cdot a_l + (1-q) \cdot a_r, \quad t_{temp} \leftarrow p/a_{temp}$
  if $t_{temp} < t_l + g_m$ **then**
    $t_{temp} \leftarrow t_l + g_m$
  else
    $t_{temp} \leftarrow t_l + g_M$
  $t_a[i_l + 1] \leftarrow t_{temp}, \quad a_{temp} \leftarrow p/t_{temp}$
  ArrF($p + s_i$, $i_l + 1$, i_r, a_{temp}, a_r, t_{temp}, t_r, g_m, g_M, s_i, t_a)
end function

Algorithm 3: RecoveryAssess

Input: d, N, th_a, th_b, m
Output: rec, th_c

$v_a \leftarrow d[N-1] - d[0]$
if $v_a > th_a$ **then**
  /* last is too larger than first */
  $rec \leftarrow False$, $th_c \leftarrow 0$
else
  $v_b \leftarrow \max_{N-m \leq i < N}(d[i]) - \min_{N-m \leq i < N}(d[i])$
  if $v_b > th_b$ **then**
    /* OWDs are not stable */
    $rec \leftarrow False$, $th_c \leftarrow 0$
  else
    /* OWDs have recovered stably */
    $rec \leftarrow True$, $th_c \leftarrow \max_{N-m \leq i < N}(d[i])$
  end
return rec,th_c

Algorithm 4: RecoveryLocate

Input: n_l, d, N, th_c, t_b
Output: L, E, U, rec_{id}

for $(rec_{id} \leftarrow N-1; rec_{id} \geq n_l; rec_{id} \leftarrow rec_{id} - 1)$ **do**
  if $d[rec_{id}] > th_c$ **then**
    Break
end
$rec_{id} \leftarrow rec_{id} + 1$, $L \leftarrow t_b[rec_{id} - 1]$, $U \leftarrow t_b[rec_{id}]$
/* **XvalueF**(x_a, y_a, x_b, y_b, y) returns x with $(x, y), (x_a, y_a), (x_b, y_b)$ on one line. */
$pre \leftarrow rec_{id} - 1$
$E \leftarrow$ **XvalueF**$(t_b[pre], d[pre], t_b[rec_{id}], d[rec_{id}], th_c)$
return L,E,U,rec_{id}

V. Experiments

A. Testbed and Settings

Testbed. Fig. 4 depicts our testbed of 5 nodes, including a sender, a receiver and three side nodes. The 5 nodes are all AMAX servers with the Intel(R) Xeon(R) Silver 4216 CPU (@2.10GHz) and two NICs (Intel 82599ES 10-Gb and X722 1Gb NICs. The operating systems are Ubuntu 18.04. Three routers are Huawei S5731-S48T4X with 10 Gbps SFP ports and 10/100/1000BASE-T electrical ports.

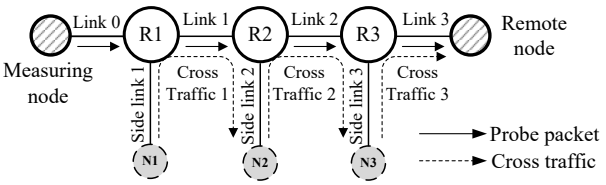


Fig. 4: Testbed topology.

Link 0 and side link 1/2 are 10 GE links. The capacity of link 1/2/3 can be switched from 10 Mbps to 10 Gbps. To

achieve higher timestamping accuracy, the traffic at link 0/3 is physically mirrored to a packet capturer of an Endace Data Acquisition and Generation (DAG) card, with 4 nanosecond timing precision. We use C_i to represent the capacity of Link i, T_i to represent the average rate of cross traffic at Link i ($i \in [0, 3]$).

Baselines. Four typical open-sourced ABW estimation tools are used as the baselines: 1) ASSOLO [16], a multi-rate PRM tool based on pathChirp; 2) PTR [6], a feedback-based PRM tool; 3) pathload [5], a feedback-based PRM tool; and 4) Spruce [9], a PGM tool.

Timing Source. As mentioned above, the Linux system clock may result in inaccurate timestamps. We only care about the OWD variation instead of the absolute value, thus we investigate how much the system clock can drift over time.

We deploy FABMon at the measuring/remote nodes and use the DAG hardware as stable timing sources. We find the clock deviation between the sender and receiver is at most $700\,\mu$s after 100 s. Since FABMon finishes within one second, the clock skew will be $\approx 7\,\mu$s, which can be ignored compared to OWD noises (≈ 10 ms). Therefore, the influence from the system clock deviation can be ignored.

Traffic generators. Two tools are employed to generate cross traffic, including iPerf3 [25] for generating traffic and Tcpreplay [26] for replaying real-world traffic traces. iPerf3 generates 500 Mbps constant traffic with < 1 Mbps error for 95% chance at 1 ms granularity, which can be regarded as accurate and unbiased traffic. As for Tcpreplay, we use DAG to capture the ground-truth rate, thus do not rely on its accuracy.

B. Parameter Selection

We set C_0=10Gbps, $C_{1,2,3}$=1Gbps, $T_{1,3}$=0, T_2=500Mbps.

Packet size. Fig. 5 shows how packet size s_l matters. The three curves represent the OWDs with $s_l = 1472, 736, 368$ B respectively. Note the sum of the loading packets remains the same and $r_l = 1$ Gbps. We can find that the 1472 B curve is close to the 736 B curve, but the 368 B curve is different, with a smaller slope and lower peak. This is caused by the minimum time cost of Linux syscall, i.e., it takes $6\,\mu$s to send a packet. To reach the 1000 Mbps rate, the packet size should be at least 750 B. This explains why the sending rate of the 368 B one is lower than the others. As long as the loading phase can cause congestion, the recovery time remains unchanged, thus FABMon can still measure ABW correctly. However, larger packets are suggested to avoid the lower sending rate and higher CPU load. In practice, we set s_l, s_i=1472 B.

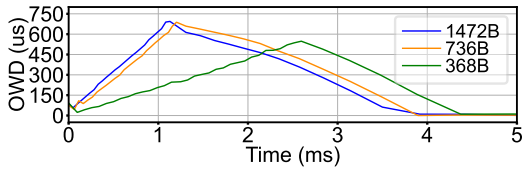
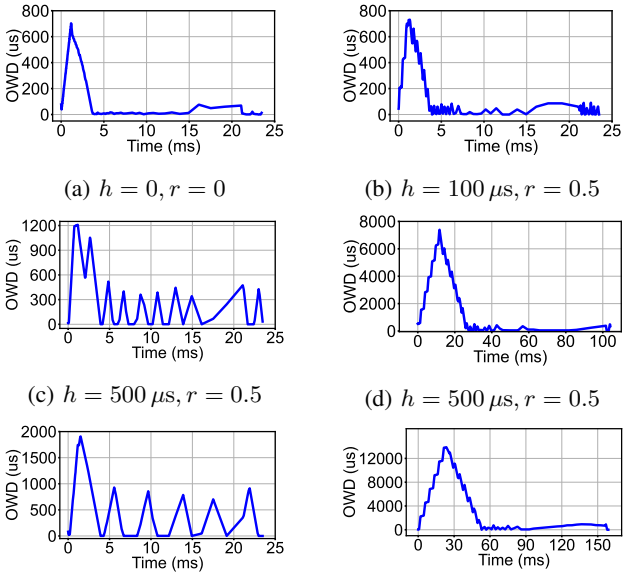


Fig. 5: Effect of packet size on "OWD-time" curves.

The rate and length of loading train. In Fig. 5, the loading train rate controls the increasing speed and the peak of the OWD. In practice, we set r_l to the path capacity, obtained from path capacity measurement tools like pathrate [27]. As for the loading train length, it does not change the shape of the "OWD-time" curve, but scales it up or down proportionally. A larger length can reduce the noise influence as discussed in §III-B. In practice, we use Eq. (IV-.1) to set the length of the loading train. Though a larger loading train increases accuracy, it also increases the chance of network congestion and packet loss. Therefore, we set a maximum limit of 2 MB for one loading phase, no more than the buffer size of most routers.

FABMon under different traffic burstiness. Fig. 6a, 6b, 6c, 6d, 6e, 6f show how to adjust the parameters of FABMon for different burstiness of traffic. Note, h represents the maximum deviation of the OWD caused by the cross traffic, r represents the ratio of the burstiness period to the stable period of the OWD caused by the cross traffic. The larger h is, the more bursty the cross traffic is; the larger r is, the more frequent the burst happens.



(a) $h = 0, r = 0$ (b) $h = 100\,\mu$s, $r = 0.5$

(c) $h = 500\,\mu$s, $r = 0.5$ (d) $h = 500\,\mu$s, $r = 0.5$

(e) $h = 1$ ms, $r = 0.5$ (f) $h = 1$ ms, $r = 0.5$

Fig. 6: Parameter adjustment for different burstiness of traffic

FABMon can handle $h \leq 100\,\mu$s in Fig. 6a, 6b with r_l=1000 Mbps, n_l=n_i=100, s_l=s_i=1472 B, as the OWD peak reaches $800\,\mu$s and the OWD noise is around $100\,\mu$s; when h=500 μs in Fig. 6c, the peak ($1200\,\mu$s) is not clearly distinguishable from the noise ($500\,\mu$s), thus the recovery time is hard to estimate. We set n_l=$1k$ in Fig. 6d to improve it, whose peak increases to $7500\,\mu$s. The REs are 19.42% for Fig. 6c and 9.50% for Fig. 6d, the latter of which shows a great improvement. Therefore, larger trains do increase the measurement accuracy. Next $h = 1$ ms in Fig. 6e and Fig. 6f. We set $n_l = 100$ in Fig. 6e and $n_l = 2k$ in Fig. 6f. The OWD noises are both $900\,\mu$s for them; The OWD peaks are 2 ms in Fig. 6e and 15 ms in Fig. 6f. As a result, the standard deviation of the estimations in Fig. 6e is 56.14 Mbps, larger

than that in Fig. 6f (4.13 Mbps). In conclusion, larger trains make the estimation more stable too.

Suggestions Table IV summarizes the suggestions and guidance on parameter selection for FABMon.

TABLE IV: Suggestions and Guidance for FABMon

Parameter	Suggestion
n_l	Set to 100 when $\max(\lambda_1, \lambda_2) = 100\,\mu$s and $C_T = 1$ Gbps and $\gamma = 10\%$
s_l, s_i	Set to MTU size (1500 B)
r_l	Set to the path capacity (1 Gbps)
n_i	Set to 100
n_a	80% inspection packets
n_b	20% inspection packets
a_m	10% path capacity
a_M	90% path capacity
g_m	20 μs
g_M	500 μs
g_n	Set to 100 μs
th_a	100 μs. Set to 1 $\sim$ 2 times the OWD variation range
th_b	100 μs. Set to the OWD variation range
m	Set to n_b

VI. Experiment Results

We demonstrate the performance of FABMon in the following aspects: 1) we compare FABMon with other four baselines in terms of precision, packet overhead and measurement time in a network with multiple links; 2) we compare their precision and stability in a network with a variable non-tight-link traffic load; 3) we test FABMon under real-world traffic.

A. Multiple Links with Traffic Loads

We set the capacities of the four links in Fig. 4 as C_0=10 Gbps, $C_{1,2,3}$=1 Gbps. We set Links 1, 2 and 3 with the same cross traffic load, i.e., T_1=T_2=T_3, from 100 Mbps to 900 Mbps, by the step of 100 Mbps . Detailed results are shown in Table V. Each result is the average of 100 repetitions. The parameters are manually adjusted to achieve the best performance for each tool.

Relative error. PTR and pathload perform well with small REs; ASSOLO has large REs; Spruce has the largest REs due to inaccurate timing and the shortcoming of PGM. On average, FABMon has the lowest RE, almost one order of magnitude lower than PTR and pathload, two orders of magnitude lower than ASSOLO and Spruce.

Packet overhead. According to Table V, FABMon, ASSOLO, Spruce have low packet cost below 1 MByte. PTR has variable overhead: lower overhead under lower traffic load and higher overhead under higher traffic load. Pathload is the tool with the highest overhead, 11.52 MByte on average.

In PTR, the sending rate of the ith train is v_i:

$$v_i = \frac{C_T}{\frac{1}{2} + \frac{i-1}{8}}, i = 1, 2, 3, \cdots \quad \text{(VI-A.1)}$$

where C_T is the capacity of the tight link. The sending rate of the first train is $v_1 = 2\,C_T$. The train number is the smallest k to satisfy $v_k < A_T$ where A_T is the tight-link ABW.

$$k = \lceil 8C_T/A_T \rceil - 3. \quad \text{(VI-A.2)}$$

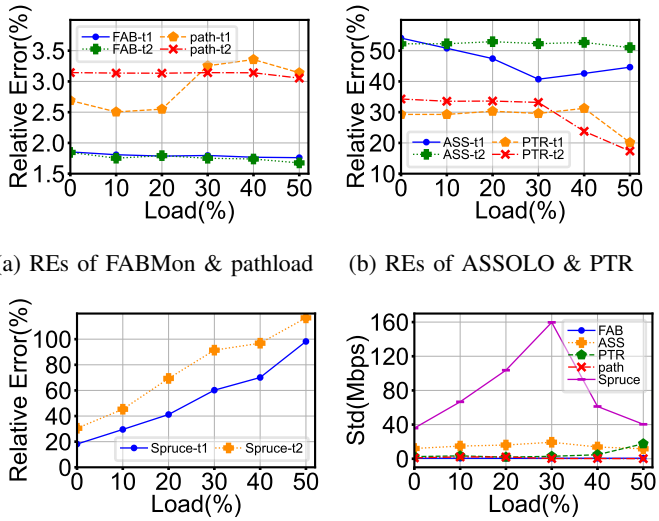
Therefore, k increases when A_T decreases, i.e., the packet overhead is higher when traffic load is higher.

The packet overhead pattern of PTR is not preferred as severe congestion or packet loss can be caused by high overhead under high traffic load.

Measurement time. As Table V shows, ASSOLO measures fastest with 2 ms for each run; FABMon is the second fastest that takes within 43.2 ms; Pathload has the longest measurement time, 4509.9 ms on average.

PTR and pathload take more time to measure as they use multiple probe trains iteratively; Spruce uses packet pairs with intervals to do a Poisson sampling, so it takes relatively medium time; ASSOLO employs chirp trains (compressed multiple trains) with less time, but has higher REs. FABMon uses one loading train and one inspection train with relatively less time, meanwhile providing the lowest REs.

Summary. Generally, fast measurement and low overhead are conflict with high measurement precision. ASSOLO has high measurement speed and low overhead, but high REs; PTR and pathload have high measurement precision and stability, but suffers from slow measurement and high overhead; Spruce takes medium measurement time and low overhead, but suffers from inaccuracy. FABMon has the highest measurement precision and stability, and meanwhile is efficient with fast measurement and low overhead.



(a) REs of FABMon & pathload (b) REs of ASSOLO & PTR

(c) REs for Spruce (d) Standard deviations ("t1")

Fig. 7: The Results of REs and deviations

B. Variable Traffic Load at the Non-tight Link

To study the systematic errors of PGM, we use DAG timestamps to remove the noises caused by inaccurate timing. DAG can provide nanosecond level timing precision. Traffic is generated to pass Links 1 and 2. The cross traffic rate at the tight link is fixed, and the rate at the other link is alterable. Links 1/2 is the tight link. T_2/T_1=0, 100, ..., 500 Mbps. T_1/T_2=600 Mbps and T_3=0. Each result is the average of 100

TABLE V: Relative error, packet overhead and measurement time of 5 tools.

Load (%)	FABMon			ASSOLO			PTR			pathload			Spruce		
	RE (%)	Time (ms)	Cost (MB)	RE (%)	Time (ms)	Cost (MB)	RE (%)	Time (ms)	Cost (MB)	RE (%)	Time (ms)	Cost (MB)	RE (%)	Time (ms)	Cost (MB)
0	0.28	6.5	0.52	18.66	2.0	0.17	3.92	15.5	0.78	2.30	3572.1	9.51	98.04	283.0	0.14
10	0.21	6.2	0.47	22.63	2.0	0.17	4.64	15.7	0.78	6.78	5037.6	13.35	100.00	283.0	0.14
20	0.27	6.0	0.42	19.37	2.0	0.17	3.79	111.3	1.05	9.35	3395.3	9.07	99.92	283.0	0.14
30	0.32	5.7	0.38	35.50	2.0	0.17	1.43	116.8	1.06	4.47	3308.8	8.84	100.00	283.0	0.14
40	0.44	5.6	0.33	30.91	2.0	0.17	1.12	108.7	1.04	3.55	8397.4	22.00	100.00	283.0	0.14
50	0.63	5.8	0.29	31.82	2.0	0.17	0.99	231.3	1.27	2.60	5035.8	13.22	100.00	283.0	0.14
60	0.71	7.3	0.29	60.18	2.0	0.17	0.85	251.7	1.31	3.08	5259.1	13.75	100.00	283.0	0.14
70	0.55	10.0	0.29	45.40	2.0	0.17	4.36	380.4	1.54	5.46	5588.1	13.89	100.00	283.0	0.14
80	0.59	16.0	0.29	10.65	2.0	0.17	6.73	669.5	2.08	8.96	2932.1	6.29	100.00	283.0	0.14
90	0.65	43.2	0.29	145.73	2.0	0.17	13.34	2064.9	4.69	79.53	2572.5	5.24	100.00	283.0	0.14
avg	0.47	11.2	0.36	42.08	2.0	0.17	4.12	396.6	1.56	12.61	4509.9	11.52	99.80	283.0	0.14

repetitions to avoid randomness. In Fig. 7, t1 or t2 means Link 1 or Link 2 is set as the tight link. FAB, path and ASS are short for FABMon, pathload, ASSOLO, respectively.

As Fig. 7a shows, FABMon and pathload have low REs. ASSOLO and PTR have medium REs of $20\%-60\%$ in Fig. 7b. As shown in Fig. 7c, Spruce has high REs, increasing as the traffic load increase. Spruce is also influenced by the location of the tight link that the REs are higher when Link 2 is the tight link than when Link 1 is.

The standard deviations of 100 repetitions are shown in Fig. 7d. FABMon and pathload are most stable.

We have two significant conclusions. First, packet pairs are not as reliable as packet trains. Packet delays between applications to the NICs and random noises from context switch or timing sources are inevitable. The influence of noises can be reduced by a number of packets, while it can not be eliminated with packet pairs. Second, PGM is good at scenarios where the traffic rates at other links are all lower compared to the traffic rate at the tight link. In other words, the errors of PGM increase when the path contains multiple links whose traffic rates are close to the tight link traffic rate. Therefore, ABW estimation tools should use packet trains as probes; PGM tools should be cautiously used only in the scenarios with one single heavy-load tight link.

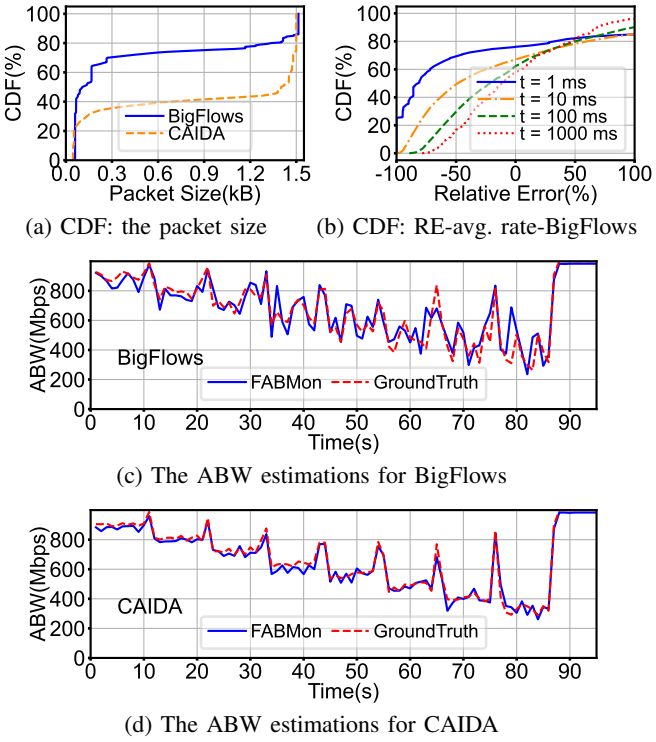
TABLE VI: Dataset information.

Metric	bigFlows1.pcap	caida2.pcap
Duration (s)	300	50
Average Rate (Mbps)	9.47	4301.75
Average Packet Size (B)	451.11	912.99
Packet Numer	786K	29M

C. Real-world Traffic

So far, we have evaluated FABMon with constant background traffic of MTU-sized UDP packets generated by iPerf3. However, real-world traffic consists of variable-size packets and is bursty. Thus, we further evaluate FABMon with real traffic traces (Table VI) replayed by Tcpreplay [26].

1) **The BigFlows trace** [28]. It is a 5-minute trace of real network traffic on a busy private network's access point to the Internet. It contains traffic of 132 applications.
2) **The CAIDA trace** [24]. It traces traffic from São Paulo to an Equinix datacenter in New York. They use Endace 6.2 DAG cards to record the traffic in bi-directional backbone links. The original pcap file includes 0.93% IPv6 packets, which are removed in our experiment.



(a) CDF: the packet size (b) CDF: RE-avg. rate-BigFlows

(c) The ABW estimations for BigFlows

(d) The ABW estimations for CAIDA

Fig. 8: Second level estimation.

Fig. 8a shows the CDF of packet sizes for the two traces. The BigFlows trace contains more small packets than the CAIDA trace. Fig. 8b shows that the relative errors from the average rate for the bigFlows trace are evenly distributed from -80% to $+100\%$ even in $1\,\mathrm{s}$ granularity. Relative errors for the CAIDA trace is similar, with a narrower range from -20% to $+20\%$ in $100\,\mathrm{ms}$ granularity, and from -10% to $+10\%$ in $1\,\mathrm{s}$ granularity.

We use Tcpreplay to reproduce the traffic at certain average rates. We replay each pcap file as follows: Node N1 sends the traffic at a certain average rate to the probe receiver iteratively. At the first iteration the average rate is v_1=100 Mbps for 10 seconds; At the second iteration, the average rate is v_2=200 Mbps for 10 seconds. We increase the average rate

step by step until the 8th iteration, v_8=800 Mbps. The total process lasts for 80 seconds. At the same time, FABMon is running at the probe sender and receiver to measure ABW. FABMon takes 50 ms for one measurement, and we set the interval between measurements to be 40 ms. Thus, FABMon measures every 90 ms. We set C_i=1 Gbps, i=0, 1, 2, 3 during this experiment.

We capture the cross traffic with the DAG card to compute the ground truth. Note that the real traffic rate is not constant. Therefore, we cluster the cross traffic collected by the DAG card every second and average the results given by FABMon every second too.

Experiment results are shown in Fig. **??** and Fig. 8d. The red curve is the prediction of FABMon and the blue curve is the ground truth. Both figures show that, as the cross traffic rate varies, FABMon quickly updates the measurement results to the new value. In general, FABMon predicts ABW close to the ground truth, and the error is smaller when the cross traffic is more constant (like in the CAIDA dataset).

VII. Conclusion

In this paper, we proposed a novel ABW estimation model named BQR, aiming to cut down the measurement time, achieve high precision and apply to scenarios with multiple congestible links and non-constant traffic. We developed FABMon based on BQR. The testbed experiments show that FABMon outperforms other ABW estimation tools with faster measurement speed, higher accuracy, lower overhead. Meanwhile, it shows robustness under scenarios with multiple congestible links and variable real-world traffic. In the future, we will explore how to adapt the BQR model to diverse scenarios, including high-speed data centers, wide area networks with different RTTs and throughputs, etc.

Acknowledgements

We thank our reviewers for their valuable and constructive comments. This work is supported in part by the National Key Research and Development Program of China (2018YFB1800204), the National Natural Science Foundation of China (62002150, 61972189), the Major Key Project of PCL (PCL2021A15), the R&D Program of Shenzhen (JCYJ20180508152204044), the Key-Area Research and Development Program of Guangdong Province (2020B010164001) and the Shenzhen Key Lab of Software Defined Networking (ZDSYS20140509172959989).

References

[1] Taner Arsan. Review of bandwidth estimation tools and application to bandwidth adaptive video streaming. In *HONET*, pages 152–156, 2012.

[2] Manish Jain and Constantine Dovrolis. Path selection using available bandwidth estimation in overlay-based video streaming. *Computer Networks*, 52(12):2411–2418, 2008.

[3] Vishnu Konda and Jasleen Kaur. Rapid: Shrinking the congestion-control timescale. In *INFOCOM*, pages 1–9, 2009.

[4] Bob Melander, Mats Bjorkman, and Per Gunningberg. A new end-to-end probing and analysis method for estimating bandwidth bottlenecks. In *Globecom*, pages 415–420, 2000.

[5] Manish Jain and Constantinos Dovrolis. Pathload: A measurement tool for end-to-end available bandwidth. In *In PAM Workshop*, 2002.

[6] Ningning Hu and Peter Steenkiste. Evaluation and characterization of available bandwidth probing techniques. *IEEE J-SAC*, 21(6):879–894, 2003.

[7] Cesar D Guerrero and Miguel A Labrador. Traceband: A fast, low overhead and accurate tool for available bandwidth estimation and monitoring. *Computer Networks*, 54(6):977–990, 2010.

[8] Robert L Carter, Mark E Crovella, et al. Measuring bottleneck link speed in packet-switched networks. *Performance evaluation*, 27(4):297–318, 1996.

[9] Jacob Strauss, Dina Katabi, and Frans Kaashoek. A measurement study of available bandwidth estimation tools. In *IMC*, pages 39–44, 2003.

[10] Vinay Joseph Ribeiro, Rudolf H Riedi, Richard G Baraniuk, Jiri Navratil, and Les Cottrell. pathchirp: Efficient available bandwidth estimation for network paths. In *PAM workshop*, 2003.

[11] Andreas Johnsson, Bob Melander, Mats Björkman, and M Bjorkman. Diettopp: A first implementation and evaluation of a simplified bandwidth measurement method. In *SNCNW*, pages 1–6, 2004.

[12] Vinay J Ribeiro, Rudolf H Riedi, and Richard G Baraniuk. Locating available bandwidth bottlenecks. *IEEE Internet Computing*, 8(5):34–41, 2004.

[13] Svante Ekelin, Martin Nilsson, Erik Hartikainen, Andreas Johnsson, J-E Mangs, Bob Melander, and Mats Bjorkman. Real-time measurement of end-to-end available bandwidth using kalman filtering. In *NOMS*, pages 73–84, 2006.

[14] Joel Sommers, Paul Barford, and Walter Willinger. A proposed framework for calibration of available bandwidth estimation tools. In *ISCC*, pages 709–718, 2006.

[15] Ling-Jyh Chen, Cheng-Fu Chou, and Bo-Chun Wang. A machine learning-based approach for estimating available bandwidth. In *TENCON*, pages 1–4, 2007.

[16] Emanuele Goldoni, Giuseppe Rossi, and Alberto Torelli. Assolo, a new method for available bandwidth estimation. In *ICIMP*, pages 130–136, 2009.

[17] Qianwen Yin, Jasleen Kaur, and F Donelson Smith. Can bandwidth estimation tackle noise at ultra-high speeds? In *ICNP*, pages 107–118, 2014.

[18] Qianwen Yin and Jasleen Kaur. Can machine learning benefit bandwidth estimation at ultra-high speeds? In *PAM*, pages 397–411, 2016.

[19] Sukhpreet Kaur Khangura. *Machine learning-based available bandwidth estimation*. PhD thesis, Hannover: Institutionelles Repositorium der Leibniz Universität Hannover, 2019.

[20] Manish Jain and Constantinos Dovrolis. Ten fallacies and pitfalls on end-to-end available bandwidth estimation. In *IMC*, pages 272–277, 2004.

[21] Li Lao, Constantine Dovrolis, and MY Sanadidi. The probe gap model can underestimate the available bandwidth of multihop paths. *ACM SIGCOMM CCR*, 36(5):29–34, 2006.

[22] Cesar D Guerrero and Miguel A Labrador. On the applicability of available bandwidth estimation techniques and tools. *Computer Communications*, 33(1):11–22, 2010.

[23] Z-L Zhang, Vinay J Ribeiro, Sue Moon, and Christophe Diot. Small-time scaling behaviors of internet backbone traffic: An empirical study. In *INFOCOM*, pages 1826–1836, 2003.

[24] The caida ucsd statistical information for the caida anonymized internet traces. https://www.caida.org/data/passive/passive_trace_statistics, 2018. Accessed: 2021-05-01.

[25] iperf3 tool. https://iperf.fr, 2021. Accessed: 2021-05-01.

[26] Tcpreplay suite. https://tcpreplay.appneta.com, 2021. Accessed: 2021-05-01.

[27] Constantinos Dovrolis, Parameswaran Ramanathan, and David Moore. Packet-dispersion techniques and a capacity-estimation methodology. *IEEE/ACM Transactions On Networking*, 12(6):963–977, 2004.

[28] Sample captures from tcpreplay. https://tcpreplay.appneta.com/wiki/captures.html, 2021. Accessed: 2021-05-01.