Accuracy Evaluation of TCP FlightSize Estimation: Analytical and Experimental Study

Mingrui Zhang, Phuong Ha, Hamid Bagheri, Lisong Xu
School of Computing, University of Nebraska-Lincoln, Lincoln, NE 68588-0115
Email: mzhang23@huskers.unl.edu,{pha, bagheri, xu}@cse.unl.edu

***Abstract*—One critical piece of information required by a TCP connection is FlightSize, which is the total amount of outstanding data contributed by the connection to the network. The FlightSize information must be estimated by a TCP connection. Recent studies have found its inaccuracy phenomenon during TCP congestion control behavior modeling, and others have amplified its inaccuracy for congestion control algorithm (CCA) identification. However, they did not study what and how each factor impacts the estimation accuracy. In this paper, we present, to the best of our knowledge, the first analytical and experimental study on the accuracy of the TCP FlightSize estimation method. Our contributions are two-fold. 1) We design and develop two FlightSize evaluation platforms using different data sources with mutually corroborated results to obtain ground truth (i.e., accurate FlightSize) in various network conditions and evaluate the accuracy of TCP FlightSize estimation. 2) We analytically and experimentally study how different network factors, such as network bandwidth, network delay, packet loss, and packet reordering, affect the accuracy of TCP FlightSize estimation. Our findings benefit the networking community by facilitating the development of more accurate TCP FlightSize estimation methods and more robust TCP congestion control algorithms that account for inherent estimation inaccuracies.**

I. Introduction

Transmission Control Protocol (TCP) plays an important role in determining the overall performance of the Internet. One critical piece of information required by TCP is FlightSize [1], which is used to determine the future TCP throughput and avoid traffic congestion on the Internet. Many different TCP congestion control algorithms have been deployed on the Internet, such as RENO [1], CUBIC [6], and BBR [4], and they all use the FlightSize information to determine the future throughput.

Intuitively, the FlightSize of a TCP connection is its current number of outstanding packets on the network. TCP, however, has to estimate the FlightSize of a connection for the following two reasons. First, TCP operates at the end hosts (i.e., sender and receiver) and does not have direct access to the instantaneous information from the network devices along the connection. Second, TCP estimates the FlightSize information at the sender of a connection, but the sender does not have the instantaneous information of the receiver due to the network delay between the sender and receiver.

In this paper, we study the accuracy of the TCP estimation method, because inaccurate estimations of FlightSize may lead to unexpected TCP performance as illustrated in Fig. 1. There is a TCP connection running Linux BBR or CUBIC for 10 seconds in emulated networks, whose settings are at Fig. 1 caption. We modify the TCP FlightSize estimation function of the Linux kernel so that it returns 50%, 100%, or 200% of the original FlightSize estimate. The boxplot figure is obtained by repeating each estimation 100 times. We can see that the change in the FlightSize estimates noticeably affects the TCP throughput of Linux BBR and CUBIC, leading to TCP performance unexpected from the original design.

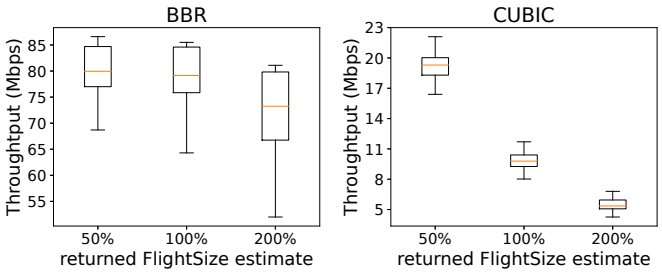

Fig. 1. The change of FlightSize estimates noticeably affects the throughput of Linux BBR (left) and CUBIC (right), leading to unexpected TCP performance. BBR runs in an emulation network with 100 Mbps bandwidth and 10% non-congestion random packet loss, and CUBIC runs in an emulation network with 100 Mbps bandwidth and 1% non-congestion loss.

Although all TCP congestion control algorithms rely on the estimated FlightSize, there has been a lack of study on the accuracy of TCP FlightSize estimation. In this paper, we present, to the best of our knowledge, the first analytical and experimental study of TCP FlightSize estimation. Our contributions are two-fold: 1) We design and develop two Linux TCP FlightSize evaluation platforms (Section VI) using different data sources with mutually corroborated results to obtain the ground truth (i.e., accurate FlightSize) in various network conditions can be obtained to evaluate the accuracy of Linux TCP FlightSize estimation. We choose Linux TCP in this work because the Linux TCP stack is the most feature-rich TCP implementation and is widely used on the Internet. 2) We analytically (Section V) and experimentally (Section VII) study how different network factors, such as network bandwidth and delay, and packet loss and reordering, affect the accuracy of TCP FlightSize estimation.

We have the following findings observed both analytically and experimentally. Network delay and packet loss each alone leads to an overestimated FlightSize. Packet reordering alone leads to an underestimated FlightSize. Network bandwidth alone does not lead to overestimation or underestimation of FlightSize, but it can influence the magnitude of the estimation error caused by other factors. The overall estimation error

979-8-3315-2096-0/25/$31.00 ©2025 IEEE

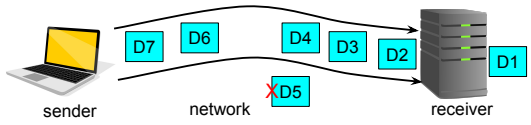

Fig. 2. FlightSize is the total number of outstanding data packets that have been sent by a TCP sender and are still in the network. In this example, FlightSize is 5 packets: D2, D3, D4, D6, and D7.

is contingent on the predominance of specific factors. Our findings benefit the networking community by facilitating the development of more accurate TCP FlightSize estimation methods and more robust TCP congestion control algorithms that account for inherent estimation inaccuracies.

II. Related Work

FlightSize, also known as packets in flight or bytes in flight in other studies, describes the amount of outstanding data in the network. Mishra *et al.* [8] proposed to use FlightSize for congestion control algorithm (CCA) identification. Arun *et al.* [3] discovered a FlightSize behavior that causes network underperforming during Reno modeling. They found part of the impact factors of FlightSize estimation accuracy in practice, e.g. high propagation delay and late acknowledgment of packet loss, however, they do not show the inherent causes. In this work, we use the FlightSize original definition rather than its estimation, and present a comprehensive analytical evaluation of FlightSize estimation accuracy.

Mishra *et al.* [8] introduce additional propagation delay time to obtain the FlightSize curve approaching the actual shape. Qian *et al.* [11] estimate the FlightSize of a TCP connection by passively monitoring TCP traffic in the network. Different from those works, we developed two evaluation platforms where we can actively and accurately obtain the value of FlightSize at packet-level resolution.

Most TCP works use the FlightSize values estimated by the operating systems, such as various TCP algorithms [1], [6], [4] and TCP measurement and analysis work [7] [9]. Different from them, we evaluate the accuracy of the TCP FlightSize estimation method.

III. TCP FlightSize Definition

As defined in RFC 5681 [1], "FlightSize is the amount of outstanding data in the network." A data packet is outstanding if it is still in the network. The FlightSize $F(t)$ of a TCP connection at time t is defined by Eq. (1), which is the total number of data packets that have been sent by the TCP sender to the network and are still in the network at time t.

$$F(t) = S(t) - R(t) - L(t) \tag{1}$$

$S(t)$ is the total number of data packets that have been **S**ent to the network from the beginning until time t. This information is available at the TCP sender. $R(t)$ is the total number of data packets that have already been **R**eceived by the receiver from the beginning until time t. This information is available at the TCP receiver. $L(t)$ is the total number of data packets that have not been received by the receiver but have already **L**eft out from the network from the beginning

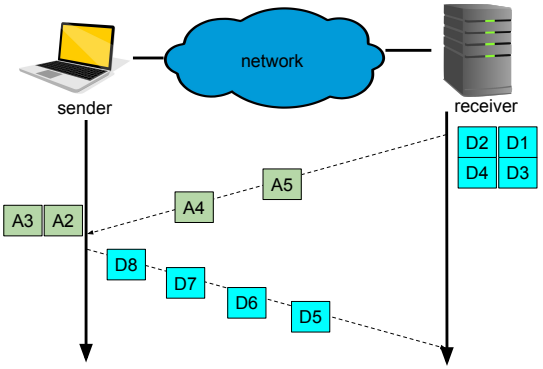

Fig. 3. The accurate FlightSize $F(t)$ is 4 packets (i.e., D5, ..., D8), but the FlightSize $F_{TCP}(t)$ estimated by TCP is 6 (i.e., D3, D4, D5, ..., D8).

until time t for various reasons, such as packet loss. This information is available on the network devices inside the network. Note that a packet may be lost at any device inside the network, and thus we need to access all these devices in order to accurately measure $L(t)$.

Let's consider an example as illustrated in Fig. 2, where a TCP sender sends a sequence of data packets to a TCP receiver over a network. Each blue square is a data packet, and the TCP sender has sent a total of 7 data packets. Specifically, the packet just sent is packet D7. Thus, $S(t) = 7$. The TCP receiver has received a total of 1 packet. Thus, $R(t) = 1$. One packet (i.e., packet D5) is dropped from the network. Thus, $L(t) = 1$. Therefore, FlightSize is

$$F(t) = S(t) - R(t) - L(t) = 7 - 1 - 1 = 5$$

IV. TCP FlightSize Estimation

In this section, we describe how TCP estimates FlightSize. TCP runs the acknowledgment-based estimation method at the sender of a connection using only the information available at the sender. Specifically, a sender estimates FlightSize using the information obtained from the received ACKs that carry the ACK sequence numbers and Selective Acknowledgment (SACK) data. An ACK received by the sender indicates that all the data packets with sequence numbers smaller than that carried by the ACK have been successfully received by the receiver. For example, an ACK with an ACK sequence number 100 indicates that all the data packets with sequence numbers smaller than 100 (e.g., 99, 98, and so on) have been successfully received by the receiver.

Let $F_{TCP}(t)$ denote the FlightSize estimated by TCP at time t, and $R_{TCP}(t)$ and $L_{TCP}(t)$ denote $R(t)$ and $L(t)$ estimated by the sender using the received ACKs, respectively. The subscript $_{TCP}$ indicates a variable estimated by TCP. Because the sender has accurate $S(t)$, we have

$$F_{TCP}(t) = S(t) - R_{TCP}(t) - L_{TCP}(t) \tag{2}$$

Fig. 3 shows an example to illustrate how TCP estimates FlightSize. The receiver has successfully received a total of 4 data packets (i.e., D1, ..., D4), and then sent 4 ACKs back to the sender (i.e., A2, ..., A5). After receiving ACKs A2 and A3, the sender sends 4 more data packets (i.e., D5, ..., D8).

In this example, the accurate FlightSize is

$$F(t) = S(t) - R(t) - L(t) = 8 - 4 - 0 = 4$$

The FlightSize estimated by TCP is

$$F_{TCP}(t) = S(t) - R_{TCP}(t) - L_{TCP}(t) = 8 - 2 - 0 = 6$$

We can see that $F_{TCP}(t) = 6$ estimated by TCP is different from the accurate $F(t) = 4$. This difference is because the sender at that time does not know (yet) that 2 data packets (i.e., D3 and D4) have already been received by the receiver. Specifically, the sender does not have the instantaneous information of $R(t)$ and only estimates it using the ACKs that, however, are received by the sender after the network delay.

This simple example already demonstrates the inaccurate TCP FlightSize estimation due to network delay. In the following sections, we analytically and experimentally study the impact of various network factors.

V. Analytical Study of FlightSize Estimation

In this section, we analytically study the impact of different network factors, including network delay, packet loss, packet reordering, and network bandwidth, on the accuracy of TCP FlightSize estimation.

Impact of network delay: We consider a network without packet loss and reordering to study the impact of network delay alone. Let Δ denote the one-way network delay between a sender and a receiver. The following theorem shows that network delay Δ leads to an overestimated FlightSize.

***Theorem** 1: $F_{TCP}(t) \geq F(t)$ for $\forall t$ in a network with one-way network delay Δ and without packet loss and reordering.*

Proof: In such a network, the TCP sender receives the accurate $R(t)$ for $\forall t$ from the receiver after Δ, and thus $R_{TCP}(t) = R(t - \Delta)$. Because there is no packet loss and reordering, we have $L_{TCP}(t) = L(t) = 0$. Therefore, we have

$$\begin{aligned} F_{TCP}(t) &= S(t) - R_{TCP}(t) - L_{TCP}(t) \\ &= S(t) - R(t - \Delta) - L(t) \\ &\geq S(t) - R(t) - L(t) = F(t) \qquad \text{by Lemma 1} \end{aligned}$$ ■

The last step of the above proof is due to $R(t-\Delta) \leq R(t)$, which is based on the following lemma that $S(t)$, $R(t)$, and $L(t)$ are all monotonically non-decreasing functions.

***Lemma** 1: $S(t_2) \geq S(t_1)$, $R(t_2) \geq R(t_1)$, and $L(t_2) \geq L(t_1)$, for $\forall t_2 > t_1$.*

Proof: $S(t)$, $R(t)$, and $L(t)$ are the cumulative numbers of data packets that have been sent, received, and left from the beginning until time t, respectively. Thus, they are all monotonically non-decreasing functions. ■

The proof also implies that the longer the network delay Δ, the bigger the $R(t) - R(t - \Delta)$, and thus the bigger the difference between $F(t)$ and $F_{TCP}(t)$ (i.e., estimation error).

Impact of packet loss: The following theorem shows that packet loss leads to an overestimated FlightSize.

***Theorem** 2: $F_{TCP}(t) \geq F(t)$ for $\forall t$ in a network with packet loss and without network delay and packet reordering.*

Proof: In such a network, $R_{TCP}(t) = R(t - \Delta) = R(t)$ because $\Delta = 0$. Also, $L_{TCP}(t) \leq L(t)$, because it takes some time (e.g., after three duplicate ACKs) for a TCP sender to detect the packet loss after a packet is lost in the network. Therefore, we have

$$\begin{aligned} F_{TCP}(t) &= S(t) - R_{TCP}(t) - L_{TCP}(t) \\ &\geq S(t) - R(t) - L(t) = F(t) \end{aligned}$$ ■

Impact of packet reordering: The following theorem shows that packet reordering leads to an underestimated FlightSize.

***Theorem** 3: $F_{TCP}(t) \leq F(t)$ for $\forall t$ in a network with packet reordering and without network delay and packet loss.*

Proof: In such a network, $R_{TCP}(t) = R(t - \Delta) = R(t)$ because $\Delta = 0$. Also, $L_{TCP}(t) \geq L(t) = 0$, because TCP may mistakenly treat reordered packets as lost packets. Therefore, we have

$$\begin{aligned} F_{TCP}(t) &= S(t) - R_{TCP}(t) - L_{TCP}(t) \\ &\leq S(t) - R(t) - L(t) = F(t) \end{aligned}$$ ■

Impact of network bandwidth: We consider a network without network delay, packet loss, and packet reordering to study the impact of network bandwidth alone. The following theorem shows that network bandwidth alone does not lead to overestimation or underestimation of FlightSize.

***Theorem** 4: $F_{TCP}(t) = F(t)$ for $\forall t$ in a network without network delay, packet loss, and packet reordering.*

Proof: In such a network, $R_{TCP}(t) = R(t - \Delta) = R(t)$ because $\Delta = 0$. Also, $L_{TCP}(t) = L(t) = 0$ because there is no packet loss and reordering. Therefore, we have

$$\begin{aligned} F_{TCP}(t) &= S(t) - R_{TCP}(t) - L_{TCP}(t) \\ &= S(t) - R(t) - L(t) = F(t) \end{aligned}$$ ■

However, because the maximum values of $F_{TCP}(t)$ and $F(t)$ increase as network bandwidth increases, network bandwidth can influence the magnitude of the estimation error caused by other factors, such as network delay, packet loss, or packet reordering, as shown in the experiments in Section VII.

VI. Evaluation Platforms for Experimental Study

To experimentally study the accuracy of TCP FlightSize estimation, we design two complementary evaluation platforms using different data sources with mutually corroborated results.

A. Design goals

We have the following design goals for the evaluation platforms. **Goal 1**: The platforms should be able to run the latest Linux kernel, in order to evaluate the accuracy of the latest Linux TCP FlightSize estimation method. **Goal 2**: The platforms should be able to test various network conditions, in order to study the impact of different network factors, such as network bandwidth and delay, and packet loss and reordering. **Goal 3**: The platforms should be able to obtain the ground truth (i.e., accurate FlightSize information $S(t)$, $R(t)$, and $L(t)$) of a TCP connection in various network conditions.

We chose Linux TCP for the evaluation platforms because the Linux TCP stack is the most feature-rich TCP implementation, is widely used on the Internet, and is open source.

B. Challenges

There are two challenges to achieve Goal 3. 1) **Anywhere packet loss challenge**: Accurately measuring $L(t)$ of a network requires capturing and analyzing all packet losses on a network, which may happen anywhere on a network including any device on a network and any packet buffer of a device (e.g., send and receive buffers of a network interface card, IP, and TCP). However, it is challenging to capture the packet events at all packet buffers of each device and all the devices on a network. 2) **Information synchronization challenge**: It is challenging to determine the exact order of the packet events at different devices (e.g., sender, receiver, all the network devices). This is because different devices may have different clocks, and also because device clocks have limited resolutions and thus packet events happening at slightly different times on different devices may have the same clock times.

C. Possible methods to build an evaluation platform

In this subsection, we discuss and compare three possible methods to build an evaluation platform, and then present our proposed platforms in the next subsections. 1) The *lab testbed method* builds an evaluation platform using a lab testbed consisting of multiple interconnected physical computers. 2) The *network simulator method* builds an evaluation platform using network simulators, such as NS-3 [10]. 3) The *network emulator method* builds an evaluation platform using network emulators, such as Mininet [2], which emulate a network of devices using interconnected virtual machines (or containers).

Goal 1: The lab testbed method and network emulator method can easily achieve the first goal, whereas the network simulator method cannot. First, network simulators usually implement their network stacks that are different from the Linux kernel. Second, some network simulators (e.g., NS-3 with Direct Code Execution (DCE) [12]) may support Linux kernels but usually support limited and outdated Linux kernels. *Goal 2:* All three methods can easily achieve the second goal. *Goal 3:* Achieving the third goal is most challenging with the lab testbed method, followed by the network emulator method, and then the network simulator method. This is because the lab testbed method may have different clock times on different devices and have limited clock resolutions, the network emulator method usually runs all the virtual machines on the same host machine and then shares the same clock, and the network simulator method uses only one simulated clock in a network simulation.

D. Proposed online evaluation platform

We first present an **online evaluation platform** that can obtain the ground truth in real-time. We choose to develop this evaluation platform using the network emulator method because both Goals 1 and 2 can be easily achieved and Goal 3 can also be achieved with our proposed modifications.

Our proposed online evaluation platform is illustrated in Fig. 4. It uses a popular network emulator - Mininet [2] to emulate a network with at least three nodes, H1, H2, and H3, all running the latest Linux kernel (i.e., Goal 1). H1 is the TCP sender, H2 is the TCP receiver, and both H1 and H2 connect to H3 and other nodes in the network. H3 runs `netem` to emulate network bandwidth, network delay, and packet reordering, and runs `iptable` to emulate and keep track of specific packet loss (i.e., Goal 2). A platform user can add more nodes to emulate more complicated network topologies.

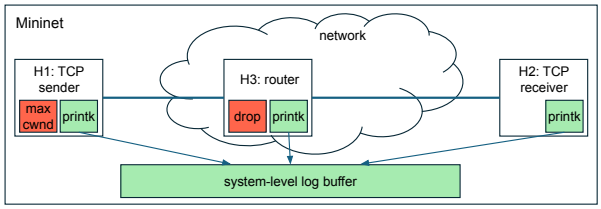

Fig. 4. Proposed online evaluation platform using network emulator Mininet. The red shaded blocks are the modifications to address the anywhere packet loss challenge, and the green shaded blocks are the modifications to address the information synchronization challenge.

The shaded blocks in Fig. 4 are our modifications to Mininet to achieve Goal 3. The red shaded blocks are the modifications to address the anywhere packet loss challenge, and the green shaded blocks are the modifications to address the information synchronization challenge. With these modifications, we can collect all the information related to $S(t)$, $R(t)$, and $L(t)$ in an experiment, and then use them to calculate accurate $F(t)$ after the experiment and compare it with $F_{TCP}(t)$ measured in the experiment. We explain our modifications in detail below.

In order to address the anywhere packet loss challenge, we propose to avoid anywhere packet loss and explicitly introduce packet loss only at H3 and/or other specific nodes. By doing so, we can accurately measure $L(t)$ by keeping track of only the packet loss at H3 instead of all possible packet buffers in all possible nodes of a network. 1) To avoid anywhere packet loss, we propose to limit the TCP throughput below a threshold. Specifically, we change Linux TCP parameter `snd_cwnd_clamp` from the default infinity to an appropriate value in order to limit the maximum TCP congestion window size and then the maximum TCP throughput, which is illustrated as the red shaded block in H1 in Fig. 4. The value should not be too high or too low. If the value is sufficiently low, there is no packet loss anywhere in the network. However, the value should not be too low, otherwise, the TCP throughput is too low to study the accuracy of Linux FlightSize estimation. To choose an appropriate value for a network condition, we gradually reduce the value until there is no packet loss in a network. 2) To explicitly introduce packet loss only at H3, we use the `iptable` rules to drop specific packets at H3, which is illustrated as the red shaded block in H3 in Fig. 4.

In order to address the information synchronization challenge, we modify the Linux kernel to print out the information of all packet events at H1, H2, and H3 to the same system-level log buffer. The order of the packet events printed into the log buffer is the same time order of these packet events at different nodes. This is because all nodes are implemented as containers on the same host machine by Mininet. Thus, they share the same system clock and Linux kernel. As a result, if a packet event occurs before another packet event even on different

Algorithm 1: Online FlightSize Evaluation

Data: BufferFile, FwdQueue[], Duration
Result: Result[N]
$Result[] = []$;
$FlightSize \leftarrow 0$;
$EndTime \leftarrow current\_time() + Duration$;
while *current_time()* $<$ *EndTime* **do**
  $Current\_line \leftarrow BufferFile.get\_nextline()$;
  if *Current_line.rule* == *sender* **then**
    $FlightSize \leftarrow FlightSize + 1$;
  else if *Current_line.rule* == *router_in* **then**
    $FwdQueue.EnQueue(Current\_line.seq)$;
  else if *Current_line.rule* == *router_out* **then**
    if *FwdQueue.head()* == *Current_line.seq* **then**
      $FwdQueue.DeQueue()$;
      $continue$;
    else
      $FlightSize \leftarrow FlightSize - 1$;
    end
  else if *Current_line.rule* == *receiver* **then**
    $FlightSize \leftarrow FlightSize - 1$;
  $time \leftarrow Current\_line.time$;
  $Result[i] \leftarrow \{time, FlightSize\}$;
  $i \leftarrow i + 1$;
end

nodes, its information is printed to the log buffer before that of another packet event. Specifically, we use `printk` which is a Linux kernel function to print to the system-level log buffer, which is a ring buffer in the memory and is designed to quickly store system-level debug information. We identify all possible locations of Linux kernel where a packet event related to $S(t)$, $R(t)$, $L(t)$ may occur, and then add `printk` to these locations to print out the Linux network stack variables related to these variables. The modifications are illustrated as the green shaded blocks in Fig. 4.

The Online FlightSize Evaluation is illustrated in Algorithm 1, which runs in real-time as a TCP connection.

E. Proposed offline evaluation platform

The above online evaluation platform obtains real-time $L(t)$ information by avoiding anywhere packet loss and explicitly introducing packet loss only at specific nodes. However, this limits the capability of the platform to emulate complex network environments. Therefore, we propose an **offline evaluation platform** that estimates the ground truth offline with high accuracy but can emulate various complex network environments.

The offline evaluation platform also uses the network emulator method and the popular network emulator - Mininet. Different from the online evaluation platform, this platform can emulate any network environment including any number of nodes, any network topology, and any network conditions, supported by Mininet. This platform only collects the TCP packets sent at the sender and received at the receiver using tcpdump. Below we explain how this platform estimates the ground truth offline with high accuracy.

In order to address the anywhere packet loss challenge, we estimate the FlightSize information using only the tcpdump information. The accurate $S(t)$ and $R(t)$ can be obtained using

Algorithm 2: Offline FlightSize Evaluation

Data: $LogFile[N]$
/*Each line of store a Send or Receive event*/;
Result: $Result[N]$
$Result[] = []$;
$FlightSize \leftarrow 0$;
while $i < len(LogFile[])$ **do**
  if *LogFile[i].action* == *SEND* **then**
    $seq \leftarrow LogFile[i].seq$;
    $tsval \leftarrow LogFile[i].tsval$;
    $action \leftarrow RECV$;
    if $\{seq, tsval, action\}$ *exist in LogFile[]* **then**
      $FlightSize \leftarrow FlightSize + 1$;
    end
  else if *LogFile[i].action* == *RECV* **then**
    $FlightSize \leftarrow FlightSize - 1$;
  $time \leftarrow LogFile[i].time$;
  $Result[i] \leftarrow \{time, FlightSize\}$;
  $i \leftarrow i + 1$;
end

the packet departure information and the packet arrival information collected from the sender and receiver, respectively. The accurate $L(t)$ is estimated using both the packet departure and arrival information. Specifically, **1)** if a data packet is sent at time t_1 from the sender and received at time t_2 at the receiver, we know that the packet is inflight from time t_1 to t_2 and counts to FlightSize during that time interval; **2)** If a data packet is sent at time t_1 and has never been received during the TCP connection, we consider that the packet is lost at time t_1 (i.e., $L(t)$ increments by one at time t_1) and does not count to FlightSize. As a result, this offline evaluation platform slightly underestimates the FlightSize for a lost packet during the time interval between the packet departure time t_1 and the actual packet loss time.

The Offline FlightSize Evaluation is illustarted in Algorithm 2, which runs offline after a TCP connection.

F. Summary

We propose two complementary evaluation platforms, summarized in Table I. The online method captures the information on whether a packet is lost and the exact time it is lost.However, this limited its ability to emulate complex network topologies and conditions. The offline method uses the sending and receiving information to determine whether a packet is lost but doesn't hold the exact time when the loss occurs, while it can emulate complex network topologies and conditions.

Difference	Online Platform	Offline Platform
Topology	Only simple Topo	complex networks
Kernel Configuration	Limit CWND Clamp	No requirement
Packet loss detection	Router not forward	Sent but not received
Packet loss precision	Accurate in realtime	At one-way delay resolution
Tools	Kernel Printk	tcpdump

TABLE I
DIFFERENCE OF TWO EVALUATION PLATFORMS

VII. Experimental Study of FlightSize Estimation

A. Overview

Questions to study: We design and conduct experiments to study and answer the following research questions (RQ). **RQ1**: Can the online and offline evaluation platforms have mutually corroborated results? **RQ2**: How do different network factors affect the accuracy of TCP FlightSize estimation? **RQ3**: Are experimental results consistent with the analytical results in Section V? **RQ4**: Do different TCP congestion control algorithms have similar trends of estimation error?

Experiments to run: To answer RQ1, we use both the online and offline platforms using the network environments supported by the online platform and then compare their results. To answer RQ2, we emulate and study the impact of different network factors, including network bandwidth, network delay, packet loss, and packet reordering, on the accuracy of TCP FlightSize estimation. To answer RQ3, we compare the experimental results with the analytical results. To answer RQ4, we run experiments for two representative TCP congestion control algorithms, CUBIC and BBR, both of which are widely used on the current Internet [8] and are available in Linux networking stack.

Accuracy Metric: We measure both the absolute error AE and percentage error PE of TCP FlightSize estimation. AE defined in Eq.(3) is the average difference between $F_{TCP}(t_i)$ and $F(t_i)$, and PE defined in Eq.(4) is AE normalized by the average $F(t_i)$, where N is the total number of packet departure and arrival events at a TCP sender in an experiment and t_i is the time of the i-th event at the sender. We measure AE and PE at every t_i because TCP updates its estimated FlightSize at every t_i.

$$AE = \frac{\sum_{i=1}^{N} \left(F_{TCP}(t_i) - F(t_i)\right)}{N} \quad (3)$$

$$PE = \frac{AE}{\sum_{i=1}^{N} F(t_i)/N} \quad (4)$$

If AE and PE are zero, TCP overall accurately estimates FlightSize. If they are greater than 0, TCP overall overestimates FlightSize; otherwise, TCP overall underestimates FlightSize.

Source Code: The source code of our evaluation platforms and the scripts and settings of all the experiments are released at https://github.com/zmrui/FlightSize.

B. Study Impact of Network Factors via Both Platforms

In this subsection, we use both online and offline evaluation platforms to mutually corroborate their results, and then study the impact of network factors. In all the experiments of this subsection, we use both evaluation platforms to emulate the network shown in Fig. 4. The bottleneck bandwidth is up to 100 Mbps and the round-trip time (RTT) is up to 100 ms, which are selected based on the Mininet emulation capability of our computer. We conduct three groups of experiments to study the impact of different network factors. The first group studies the impact of network bandwidth and delay, the second group studies the impact of packet loss, and the third group studies the impact of packet reordering. All three groups are also used to mutually corroborate the results of the two evaluation platforms.

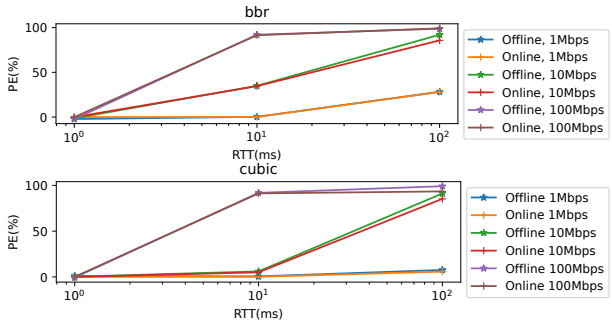

Fig. 5. Impact of network bandwidth and delay: Network delay leads to an overestimated FlightSize. Although network bandwidth alone does not lead to estimation error, it amplifies the estimation error caused by network delay. Both BBR and CUBIC have similar estimation error trends.

The first group of experiments changes the bottleneck bandwidth from 1 Mbps to 100 Mbps and the RTT from 1 ms to 100 ms. To isolate the impact of network bandwidth and delay from that of packet loss and reordering, we use the method described in Section VI-D to emulate a network without packet loss and reordering. We use `iperf` to run a TCP connection with BBR or CUBIC between H1 and H2 for 18 seconds. We choose 18 seconds because the maximum capacity of the Linux ring buffer used in the online evaluation platform is 2^{25} Bytes and is only long enough to hold the experiment results of 18 seconds.

Fig. 5 shows the percentage error PE of BBR (top) and CUBIC (bottom). There are **slight differences** between online and offline platforms. This is because these two platforms collect packet information at different layers. The online platform directly collects the packet information at the TCP layer using printk inside Linux TCP kernel, while the offline platform collects the information at the data link layer using tcpdump at the network interface card. This leads to slight microsecond-scale timestamp differences for the same packet departures and arrivals. We can see that the offline platform can estimate the ground truth with high accuracy and thus can be used to study the accuracy of TCP FlightSize estimation.

We have the following observations from Fig. 5. **1)** When RTT is close to zero (i.e., 1 ms), PE is close to zero for both BBR and CUBIC and for all bandwidth rates. This is consistent with Theorem 4 that *network bandwidth alone does not lead to overestimation or underestimation of FlightSize in a network without network delay, packet loss, and packet reordering*. **2)** As RTT increases, PE is greater than zero (i.e., overestimation) and increases for both BBR and CUBIC and for all bandwidth rates. This is consistent with Theorem 1 that *network delay leads to an overestimated FlightSize*. **3)** As the bandwidth increases, PE increases for both BBR and CUBIC and for all the RTTs. That is, *although network bandwidth alone does not lead to estimation error, it amplifies the estimation error caused by network delay*. **4)** *Both BBR and CUBIC have similar trends of estimation error*. This is

because they use the same TCP FlightSize estimation method, and the difference in their PE values is caused by their different congestion window sizes that are determined by their congestion control algorithms.

The second group of experiments emulates a network with various packet loss rates. The experimental results of both BBR and CUBIC are consistent with Theorem 2 that *packet loss leads to an overestimated FlightSize*. The third group of experiments emulates a network with various packet reordering. The experimental results of both BBR and CUBIC are consistent with Theorem 3 that *packet reordering leads to an underestimated FlightSize*. Due to the page limit, their figures are not shown in the paper. All the scripts, results, and figures are released at https://github.com/zmrui/FlightSize.

C. Dynamic Real-World Emulation via Offline Platform

In this subsection, we use only the offline evaluation platform to evaluate TCP FlightSize estimation in dynamic real-world network environments which are not supported by the online evaluation platform. Specifically, the offline evaluation platform emulates a dynamic cellular network [5] by replaying the traffic captured from a U.S. major cellular ISP. As the emulated network condition varies from good to poor, the emulation introduces lower average network bandwidth, higher average packet loss rate, and higher average network delay.

In these experiments, the estimation errors are affected by multiple factors, such as network bandwidth, network delay, packet loss, packet reordering, and congestion control algorithms. The overall estimation error is contingent on the predominance of specific factors. Fig. 6 shows the percentage error PE (top) and the absolute error AE (bottom) of BBR and CUBIC. We can see that both PE and AE are greater than zero. That is, both BBR and CUBIC overall over-estimate FlightSize. This is because packet reordering is less predominant than other factors in these experiments. By comparing PE and AE, we can also see that both BBR and CUBIC have similar absolute errors but different percentage errors. This is because BBR achieves higher throughput (i.e., higher $F(t_i)$ in Eq.(4)) than CUBIC in cellular networks.

D. Discussions

RQ1: Can the online and offline evaluation platforms have mutually corroborated results? Yes, they obtain similar FlightSize results with acceptable minor differences caused by different timestamping and loss inference methods.

RQ2: How do different network factors affect the accuracy of TCP FlightSize estimation? Network delay and packet loss each alone leads to an overestimated FlightSize, whereas packet reordering alone leads to an underestimated FlightSize. Although network bandwidth alone does not lead to estimation error, it can influence the magnitude of the estimation error caused by other factors.

RQ3: Are experimental results consistent with the analytical results in Section V? Yes, the experimental results are consistent with the analytical results described in Theorems 1, 2, 3, and 4 in Section V.

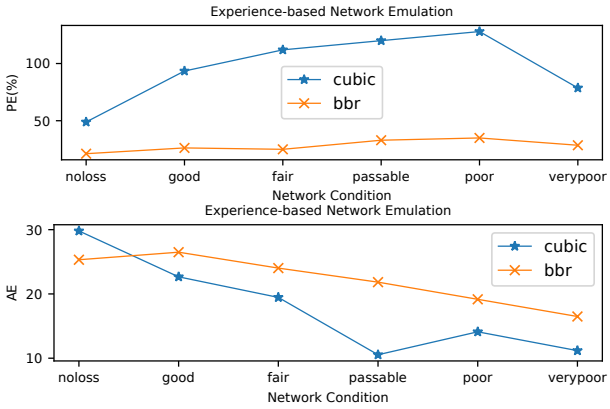

Fig. 6. The offline evaluation platform can be used to evaluate TCP FlightSize estimation in dynamic cellular networks. Both BBR and CUBIC overall overestimate FlightSize. They have similar absolute errors but different percentage errors due to their different throughputs.

RQ4: Do different TCP congestion control algorithms have similar trends of estimation error? Yes, they have similar trends of estimation errors because they use the same TCP FlightSize estimation method. They may have different absolute or percentage estimation errors because they have different throughputs determined by their congestion control algorithms.

VIII. Conclusion

In this paper, we proposed two complementary platforms to evaluate the accuracy of TCP FlightSize estimation. Our future work is to design a more accurate FlightSize estimation method for TCP based on our findings.

IX. Acknowledgement

This work was supported in part by NSF CCF-2124116.

References

[1] M. Allman, V. Paxson, and E. Blanton. TCP congestion control. *RFC 5681*, September 2009. I, II, III

[2] An Instant Virtual Network on your Laptop (or other PC). http://mininet.org//. VI-C, VI-D

[3] V. Arun, M. Arashloo, A. Saeed, M. Alizadeh, and H. Balakrishnan. Toward formally verifying congestion control behavior. In *Proceedings of ACM SIGCOMM*, page 1–16, Virtual Event, 2021. II

[4] N. Cardwell, Y. Cheng, C. Gunn, S. Yeganeh, and V. Jacobson. BBR: Congestion-based congestion control. *Communications of the ACM*, 60(2):pp. 58–66, February 2017. I, II

[5] cell-emulation-util:A script based on Linux TC netem to emulate the latency, loss, and bandwidth of a real-world cellular network. https://github.com//akamai//cell-emulation-util. VII-C

[6] S. Ha, I. Rhee, and L. Xu. CUBIC: A new TCP-friendly high-speed TCP variant. *SIGOPS Oper. Syst. Rev.*, 42(5):64–74, jul 2008. I, II

[7] M. Mellia, A. Carpani, and R. Lo Cigno. Measuring IP and TCP behavior on edge nodes. In *Proceedings of IEEE Global Telecommunications Conference (GLOBECOM)*, volume 3, pages 2533–2537, 2002. II

[8] A. Mishra, L. Rastogi, R. Joshi, and B. Leong. Keeping an eye on congestion control in the wild with nebby. In *Proceedings of ACM SIGCOMM*, page 136–150, 2024. II, VII-A

[9] B. Nechaev, M. Allman, V. Paxson, and A.V Gurtov. A preliminary analysis of TCP performance in an enterprise network. In *Proceedings of INM/WREN*, volume 10, 2010. II

[10] Network Simulator 3. https://www.nsnam.org/. VI-C

[11] F. Qian, A. Gerber, Z. Mao, S. Sen, O. Spatscheck, and W. Willinger. TCP revisited: a fresh look at TCP in the wild. In *Proceedings of ACM IMC*, Chicago, IL, November 2009. II

[12] H. Tazaki, F. Uarbani, E. Mancini, M. Lacage, D. Camara, T. Turletti, and W. Dabbous. Direct code execution: revisiting library OS architecture for reproducible network experiments. In *Proceedings of ACM CoNEXT*, Santa Barbara, CA, December 2013. VI-C