A Lightweight Approach to Manifesting Responsible Parties for TCP Packet Loss

Guang Cheng
School of Computer Science, Southeast University, P.R.China
Key Laboratory of Computer Network &
Information Integration, Ministry of Education, P.R.China
email: gcheng@njnet.edu

Yongning Tang Tibor Gyires
School of Information Technology
Illinois State University, USA
email: ytang, tbgyires@ilstu.edu

***Abstract*—Troubleshooting TCP packet loss is a crucial problem for many network applications. TCP packets could be lost in different network segments for various reasons. Understanding the responsible parties for TCP loss is an important step for network operators to diagnose related problem. However, TCP is designed for end-to-end control. It is difficult for any third party to detect whether and where (even coarsely) TCP packet loss has occurred. We design *TCPBisector*, a lightweight efficient diagnosis tool to manifest responsible parties for TCP packet loss. *TCPBisector* divides the responsibility between "My" and "Other" parties or networks (denoted as Net_m and Net_o) conceptually delimited by a passive network measurement point (denoted as MP), and quantifies the responsibility by using TCP packet loss ratios on the corresponding networks. The evaluation shows that the *TCPBisector* can accurately estimate TCP packet loss ratios with estimation error rate 3.5-6.9%.**

Keywords- responsibility; performance diagnosis.

I. INTRODUCTION

TCP packets could be lost in different network segments for various reasons, including network congestion, packet corruption, faulty network components, and network misconfiguration. Understanding the responsible parties for TCP packet loss is an important step for network operators to troubleshoot the problem. However, TCP [17] is designed as an end-to-end control protocol. It is difficult for any third party to detect whether and where (even coarsely) TCP packet loss has occurred.

TCP performance monitoring and diagnosis have been extensively studied. Several sophisticated network monitoring frameworks [7][18] and intrusive active probing techniques [1][8][9] were proposed to pinpoint the root cause of TCP packet loss. Many previous work [2]-[6] also focused on comprehensively understanding TCP behaviors (e.g., including TCP window sizes, TCP retransmission and reordering) and its correlation with the actual network performance (e.g., network throughput and congestion). Maintaining accurate and complete TCP flow information is critical for this type of study, which typically requires large memory space and high computing power. Recent study [16] focused on providing real time TCP monitoring and performance diagnosis based on various flow sampling techniques, which may skip important flow information.

In this paper, we propose *TCPBisector*, a lightweight tool to help network operators to answer one critical question: "How much should I (or other parties) be responsible for TCP packet loss?"

As shown in Figure 1, *TCPBisector* divides TCP packet loss responsibility between "My" and "Other" parties or networks (denoted as Net_m and Net_o) conceptually delimited by a passive network measurement instrument (denoted as Measurement Point or MP), and quantifies their responsibilities by using TCP packet loss ratios on "My" and "Other" networks (denoted as LR_m and LR_o), respectively.

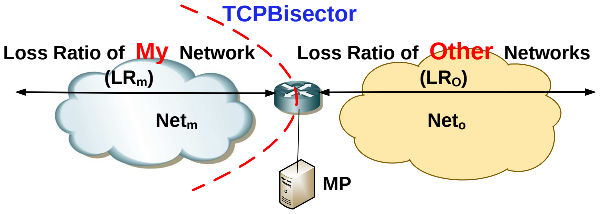


FIGURE 1: *TCPBisector*: a tool to bisecting responsible parties for TCP Packet Loss between Net_m and Net_o

Our work makes two contributions. Our first contribution is TCP behavior modeling. In the paper, we show that TCP presents different behaviors at the MP when TCP packets are lost by different parties between Net_m and Net_o. Accordingly, we model TCP behaviors by using several easy-to-track network events that allow the MP to ascribe the TCP packet loss responsibility to different parties.

Our second contribution is an efficient TCP packet loss inference algorithm. Instead of studying the causality of TCP packet loss, the *TCPBisector* only requires a small set of essential TCP related events commonly observable in various TCP packet loss scenarios. Thus, the *TCPBisector* can be used effectively and efficiently to infer occurred TCP packet loss and further identify their relative occurring locations, without suffering from the overhead of distinguishing TCP loss scenarios as shown in many related work [5][15][16]. Our inference algorithm presents the computation complexity of $O(n)$ and only requires a bounded memory space.

TCPBisector requires only one passive network measurement instrument (i.e., MP) as shown in Figure 1. The MP can be deployed arbitrarily on network depending on how

Copyright (c) IARIA, 2015. **ISBN: 978-1-61208-398-8**

a responsibility scope defined under different monitoring strategies. Essentially, Net_m represents the scope of "my" responsibility, and Net_o shows the boundary of others. Depending on the different deployment strategy, the Net_m can be an enterprise network using cloud services, or a data center providing cloud services.

The rest of the paper is organized as the following. Section II describes the related work, and Section III introduces the TCP behavior modeling. Section IV presents the modularized *TCPBisector* as a system and discusses the algorithm for inferring LR_m and LR_o. We show the validation of our system via both emulations and experiments on a Tier-1 network in Section V. Section VI concludes our work.

II. RELATED WORK

Numerous measurement studies have investigated the characteristics of TCP connections in the Internet, either via actively measured end-to-end properties (e.g., loss, delay, throughput) of TCP connections, or passively characterized a connection's aggregate properties (e.g., size, duration, throughput). Various TCP measurement methodologies and metrics have also been proposed [10]-[13] to monitor TCP performance via a set of important TCP parameters (e.g., RTT[14]).

Among various TCP related parameters, TCP packet loss is one of the most important metrics. Many scholars have proposed a variety of methods for TCP packet loss ratio estimation. Sommers et al., [1] proposed to send probe packets by the sender, and view the number of probe packets at the receiver that arrives to estimate end-to-end packet loss ratio. Benko and Veres [2] proposed to use the observed TCP sequence numbers to estimate TCP packet loss. Ohta and Miyazaki [3] explored a passive monitoring technique for packet loss estimation relying on hash-based packet identification. Friedl et al. [4] compared flows with sender and receiver for computing the packet loss,

Jaiswal et al. [5][15] presented a passive measurement methodology that observes the TCP packets in a TCP connection and infers/tracks the time evolution of two critical sender variables: the sender's congestion window (cwnd) and the connection round trip time (RTT). Allman et al. [6] estimated the packet loss ratio by observing the sender's retransmit packets. Nguyen et al. [7] built a model called the HSMM to analyze the packet loss. Zhang et al. [8] analyzed packet loss, delay and bandwidth from the random packet of the entrance and packet loss. *STA* [18] developed an efficient packet classification technique which is used to infer the loss and reorder rates of individual TCP flows.

Recent research has studied how to diagnose TCP performance issues in clouds. Ghasemi et al. [16] proposed a heuristic inference algorithm to infer several important TCP parameters (e.g., congestion-window size and the TCP state) from sampled TCP related statistics (e.g., RTT).

TCPBisector proposed in this paper is to coarsely bisect TCP packet loss responsibility between interior and exterior networks. *TCPBisector* is designed based on the fact that observable TCP behaviors could be different on different network segments along the same end-to-end path under the same network condition. *TCPBisector* aims at providing a practical, lightweight, and real-time tool for both cloud users and service providers understand network conditions between Net_m and Net_o.

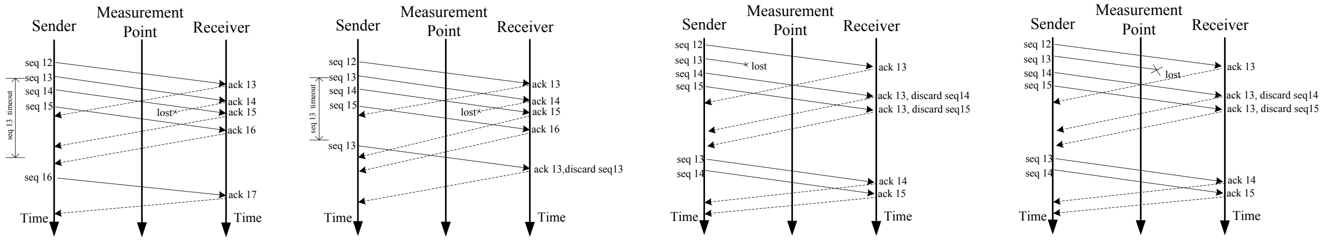
III. TCP BEHAVIOR MODELING

Although TCP is designed as an end-to-end control protocol, we show that the MP in the middle still can discern differences on the corresponding TCP behaviors when packet loss occurred on the different responsible parties (i.e., Net_m and Net_o). In the following, we will first illustrate several representative TCP packet loss scenarios. Then we will define two TCP behaviors distinguishable by the MP so as to ascribe the TCP packet loss responsibility to Net_m or Net_o.

A. TCP Loss Scenarios

TCP behaves differently in response to varying network condition. More importantly, TCP presents different observable behaviors at the MP when TCP packets loss occurred in Net_m or in Net_o. In the following, we illustrate our ideas via several representative TCP packet loss scenarios as shown in Figure 2. In all the scenarios, we assume the sender is from Net_m and the receiver is located within Net_o.

- ACK loss: Figure 2(a) & 2(b) show that a data packet from the sender has successfully delivered to the receiver. However, one acknowledgement packet (i.e., ACK_{14}) from the receiver was lost. In the scenario shown in Figure 2(a), since the following ACKs (i.e., ACK_{14} and ACK_{15}) arrived before a retransmission timeout event triggered at the sender, no retransmission occurred. Otherwise, the sender retransmitted the unacknowledged packet (i.e., Seq_{13}) as shown in Figure 2(b). Apparently, it appeared to the MP as if no packet loss in the first loss scenario (shown in Figure 2(a)). In this scenario, only one ACK lost in Net_o before passing the MP. However, the following ACKs successfully arrived at the sender, which took over the responsibility of the lost ACK. Thus, considering this scenario the same as no TCP loss makes sense practically. In the scenario shown in Figure 2(b), the MP could observe the occurrence of data retransmission.
- Single packet loss with 3-ACK: Figure 2(c) and Figure 2(d) show a type of common TCP loss scenarios, in which one data packet (i.e., Seq_{13}) was lost. Consequently, the sender received three duplicate ACKs (denoted as 3-ACK). Depending on where the data packet lost, the MP may only observe three consecutive ACKs as shown in Figure 2(c) if the loss occurred in Net_m; or observed duplicated data

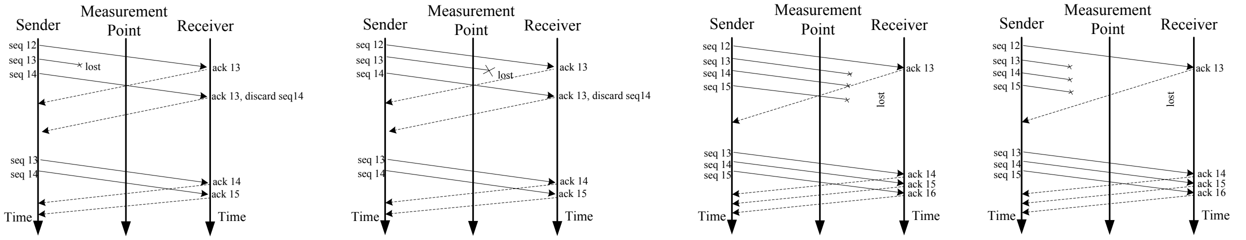


(a) Single packet lost w/o retransmission

(b) Single ACK lost w/ retransmission

(c) Single packet lost in Net_m w/ 3-ACK

(d) Single packet lost in Net_o w/ 3-ACK



(e) Single packet lost in Net_m

(f) Single packet lost in Net_o

(g) Mass packet lost in Net_o

(h) Mass packet lost in Net_o

FIGURE 2: The Illustration of Representative TCP Packet Loss Scenarios

packets in addition to three consecutive ACKs as shown in Figure 2(d) if the loss occurred in Net_o.

- Single packet loss with timeout: Figure 2(e) and Figure 2(f) show another type of common TCP loss scenarios, in which the sender didn't receive three duplicate ACKs. Instead, a timeout event was triggered at the sender for retransmission. Figure 2(e) shows the case when TCP loss occurred in Net_m, in which the MP could observe out-of-order IPID. More specifically, the IPID in the TCP packet with Seq_{13} is larger than the IPID in the TCP packet with Seq_{14}. Figure 2(f) shows the case when TCP loss occurred in Net_o, in which the MP could observe duplicated packets in addition to out-of-order on IPID.
- Mass packet loss: Figure 2(h) and Figure 2(g) show the scenarios when multiple consecutive transferred packets were lost. If the packet loss occurred in Net_m (Figure 2(h)), the MP observed abnormal time gap between transferred data. If the packet loss occurred in Net_o (Figure 2(g)), the MP observed duplicated data transfer in addition to abnormal time gap between transferred data.

By no means, we try to list all possible TCP packet loss scenarios. Instead, we would like to point out from these illustrating examples that (1) TCP behaves differently when TCP packet loss occurs in Net_m or in Net_o; and (2) such TCP behavior differences can be characterized via a small set of easy-to-check TCP related network events. We will show in TABLE I that all the scenarios shown in Figure 2 can be identified in our proposed TCP Behavior Model (TBM).

B. Characterizing TCP Behavior

We want to characterize TCP behaviors so that the MP can effectively detect TCP packet loss and identify the corresponding occurring locations based on the observed TCP behaviors.

In the following, we first define several TCP related parameters, and then use them to specify four easy-to-check TCP events that can be used by the *TCPBisector* to detect TCP packet loss and further ascribe the responsibility for TCP packet loss to Net_m or Net_o.

For i^{th} observed TCP packet (denoted as p_i) at the MP, we denote by I_i and Q_i the corresponding IPID and TCP sequence number, respectively. Let $T_{i,j}$ be the time interval between p_i and p_j $(i < j)$ in the same TCP flow, and let T_{f_k} denote the estimated sender's retransmission timeout for TCP flow k.

We introduce four easy-to-check TCP events as below. Each event is denoted by a binary variable $e_i (i = 1, 2, 3, 4)$, and we say e_i is $True$ if the associated network condition is detected. More specifically, we have:

- e_1 (timeout event): When the condition $T_{i,j} > T_{f_k}$ $(p_i, p_j \in f_k)$ is observed, $e_1 = True$.
- e_2 (3-ACK event): When the condition $I_j - I_i \geq 3$ is observed, $e_2 = True$.
- e_3 (reordering event): When the condition $Q_i > Q_j$ is observed, $e_3 = True$.
- e_4 (retransmission event): When the condition $Q_i = Q_j$ is observed, $e_4 = True$.

In our TCP behavior model or TBM, e_1 and e_2 are called triggering events because either event indicates the occurrence of TCP packet loss. e_3 and e_4 are called distinguishing events because e_3 should be observed if the packet lost in Net_m; otherwise e_4 should be observed.

Next, we are going to define two distinguishable TCP behaviors. Each TCP behavior should be observed by the

MP to infer the associated occurring location of TCP packets loss.

Definition 1. We define *Behavior I* as the observable TCP behavior when TCP packets are lost before the MP (i.e., in Net_m), denoted by a binary variable B_I . More specifically, B_I is True when the condition $(e_1 \vee e_2) \wedge e_3$ is satisfied, namely, $B_I = (e_1 \vee e_2) \wedge e_3$.

Definition 2. We define *Behavior II* as the observable TCP behavior when TCP packets are lost after the MP (i.e., in Net_o), denoted by a binary variable B_{II}. More specifically, B_{II} is True when the condition $(e_1 \vee e_2) \wedge e_4$ is satisfied, namely, $B_{II} = (e_1 \vee e_2) \wedge e_4$.

Following up the previously discussed TCP loss scenarios (as shown in Figure 2), now we can characterize them using TBM as shown in TABLE I.

TABLE I: TCP PACKET LOSS SCENARIO IN TBM

TCP packet loss scenario	e_1	e_2	e_3	e_4	B_I	B_{II}
Figure 2(a)						
Figure 2(b)	√			√		√
Figure 2(c)		√	√		√	
Figure 2(d)		√		√		√
Figure 2(e)	√		√		√	
Figure 2(f)	√			√		√
Figure 2(h)	√		√		√	
Figure 2(g)	√			√		√

As we discussed earlier, TCP packet loss can occur in various scenarios. Instead of studying the causality of TCP packet loss, we adopt into our model (i.e., TBM) the essential common events observable in various TCP packet loss scenarios. Thus, the TBM can be used effectively and efficiently identify TCP packet loss, without suffering from the overhead of distinguishing TCP loss scenarios as in many related work. For instance, TCP load balancing, as a misleading scenario discussed in [2], will not trigger any events from $e_1 \sim e_4$ in the TBM if no TCP packet loss occurred.

IV. THE SYSTEM

The *TCPBisector* consists of three modules as shown in Figure 3: (1) data processing module (DPM), (2) inference engine module (IEM), and (3) reporting & querying module (RQM). The *TCPBisector* can be run directly on the MP or installed on a different server. The *TCPBisector* receives from the MP all captured TCP packets, and reports aggregated and flow-based TCP packet loss ratios on Net_m and Net_o, respectively.

DPM collects all TCP packets passing through the MP, and classifies them into TCP flows based on five-tuple (i.e., source and destination IP addresses, source and destination ports, protocol number). The memory location of each recorded TCP flow is stored in a hash table for fast retrieval. For each TCP flow, the *TCPBisector* only needs to keep a fixed number (i.e., 25 as discussed later) of TCP packets in order to accurately estimate TCP packet loss ratios. The corresponding record for each TCP packet in flow k includes its IPID, TCP sequence number, its arrival time, and the estimated retransmission timeout per flow, the TCP packet loss ratios (LR_m^k and LR_o^k). For each flow k, the *TCPBisector* counts the total number of traversed TCP packets denoted as N_k in both directions. We denote by LN_k^I and LN_k^E as the total number of lost TCP packets in Net_m and Net_o, respectively. Accordingly, we have $LR_m^k = LN_k^m/N_k$ and $LR_o^k = LN_k^o/N_k$. The *TCPBisector* also aggregates the flow statistics to provide the aggregated LR_m and LR_o for all observed active TCP flows.

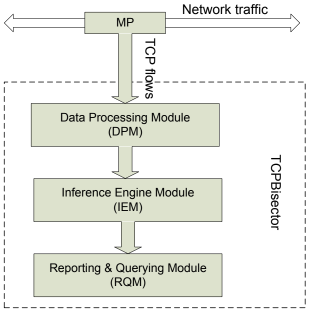


FIGURE 3: *TCPBisector* System

IEM is essentially an event handler. If a triggering event (e_1 or e_2) detected for TCP flow k, IEM verifies the occurrence of distinguishing events (e_3 or e_4) in the recorded flow data structure in order to ascribe the packet loss to the corresponding responsible party (i.e., Net_m or Net_o).

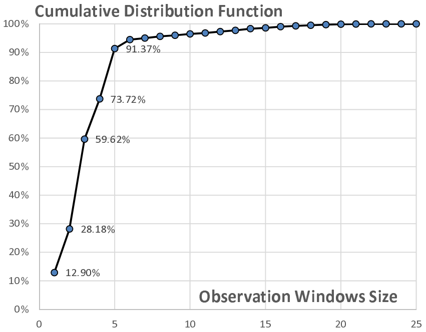
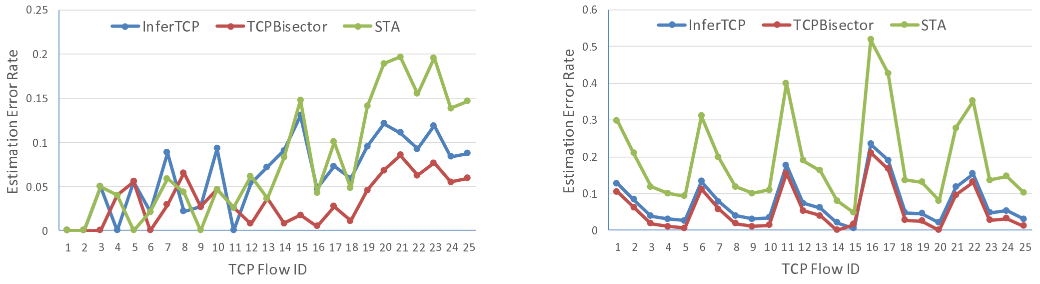


FIGURE 4: CDF of Observation Window Size and Packet Loss Estimation Accuracy

The core function in the *TCPBisector* is TCP loss ratio estimation, which requires to efficiently detect the relevant network events based on captured TCP packets per flow. One significant difference between the *TCPBisector* and the related work [5][15][16][18] is that the *TCPBisector* does not need maintain complete TCP flow information. As shown in our empirical study, the *TCPBisector* only needs to keep track of small number of TCP packets per flow to accurately estimate TCP packet loss ratios. Such a desirable feature in the *TCPBisector* results from the TCP



(a) TCP loss in Net_m

(b) TCP loss in Net_o

FIGURE 5: Comparison between InferTCP, STA and TCPBisector (a) when TCP loss in Net_m, (b) when TCP loss in Net_o

behavior modeled, which only relies on several easy-to-check TCP events (i.e., $e_1 \sim e_4$). Such a difference makes the *TCPBisector* a lightweight and efficient tool with much lower requirement on the system's computing power and memory space,

Based on the network traces collected from both our emulation and real network experiments, we empirically study the relationship between the estimation accuracy and the size of observation window measured by the number of required TCP packets per flow. The statistics results are shown in Figure 4, where the horizontal axis shows the varying sizes of observation window per TCP flow, and the vertical axis is the CDF of the observation window size measured by trackable number of TCP packets for each flow. Based on Figure 4, we can clearly see that 91.373% of all TCP loss cases, the gap between a lost packet and its retransmitted packet is less than or is equal to 5. We can track almost all TCP packet loss if we record 25 TCP packets per flow. Thus, the time complexity of the inference engine is $O(n)$, where n is the total number of interested TCP flows. Since for each TCP packet, we only keep 20-byte IP header and 20-byte TCP header, the total memory space is bounded by the number of interested TCP flows. For the purpose of cloud application monitoring, the number of flows should be limited.

Finally, RQM updates the per-flow and aggregated statistics of TCP packet loss ratios. RQM also provides query interface such that collaborative parties can correlate their *TCPBisector* reports on the commonly interested TCP flows in order to present a finer-grained view on their network.

V. EVALUATION

We validate the *TCPBisector* using both emulations in a controlled environment and experiments in a Tier-1 network. We also compared the performance of the *TCPBisector* to two related work [15][18].

A. Emulation

We validate the correctness of our methodology used in the *TCPBisector* via a emulation, in which we can obtain the ground truth of various TCP related parameters and packet loss ratios on different network segments.

TABLE II: ACCURACY VERIFICATION VIA EMULATION

LR_m			LR_o		
Actual	**Estimate**	**Error Rate**	**Actual**	**Estimate**	**Error Rate**
0.437%	0.437%	0	0.521%	0.576%	0.106%
0.648%	0.648%	0.000%	5.321%	5.374%	0.010%
0.450%	0.426%	0.053%	8.547%	8.599%	0.006%
0.954%	0.929%	0.026%	5.389%	5.443%	0.010%
1.064%	0.967%	0.091%	8.982%	9.109%	0.014%
3.421%	3.244%	0.052%	1.035%	1.090%	0.053%
3.069%	2.798%	0.088%	4.919%	4.920%	0.0002%
2.778%	2.425%	0.127%	9.590%	9.443%	0.015%
5.631%	5.218%	0.073%	0.817%	0.954%	0.168%
4.931%	4.336%	0.121%	8.207%	8.210%	0.003%
5.413%	4.900%	0.095%	5.307%	5.438%	0.025%
9.071%	8.311%	0.084%	4.884%	5.042%	0.032%
8.819%	8.049%	0.087%	9.249%	9.351%	0.011%
Average Error Rate		0.069%	**Average Error Rate**		0.035%

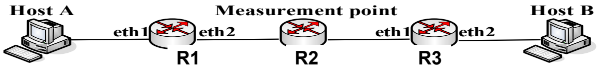


FIGURE 6: Emulation Environment

In our emulation as shown in Figure 6, a 5-node network is constructed, including two end hosts connected through three routers (i.e., R1, R2, and R3). The TCP packet loss ratios on different router ports are controlled by *Netem* [19]. The error rate is calculated as $Err(LR_m) = |LN^m_{actual} - LN^m_{TCPBisector}|/LN^m_{actual}$ and $Err(LR_o) = |LN^o_{actual} - LN^m_{TCPBisector}|/LN^m_{actual}$ for LR_m and LR_o error rates, respectively. The emulation results as in TABLE II showed that the error on estimating Internal Loss Ratio (LR_m) is 0.069, and the error on estimating External Loss Ratio (LR_o) is 0.035. The result shows that the *TCPBisector* achieves high accuracy on loss ratio estimations for both LR_m and LR_o.

B. Comparison via Emulation

We compare the performance of *TCPBisector* to the two closest related work referred to as *InferTCP* [15] and *STA* [18]. *InferTCP* kept track of the values of two important variables: the senders congestion window (cwnd) and the connection round trip time (RTT) to diagnose end-user-perceived network performance. *STA* [18] developed an efficient packet classification technique which is used to infer the loss and reorder rates of individual TCP flows.

We adopt the same emulation environment as used in *InferTCP* [15] to compare *InferTCP* and *STA* with *TCPBisector*. We generated 25 TCP flows, and each flow has $3.600 \sim 4,500$ packets. We control the loss ratio is between 0.5% and 10% for each TCP flow. As shown in Figure 5, *TCPBisector* outperformed both *InferTCP* [15] and *STA* [18], and achieved $3 \sim 10\%$ lower estimation error rate on both LR_m and LR_o.

C. CERNET Traces

The traces have been collected at different time from a Tier-1 backbone network CERNET. The MP is placed between the border router in Jiangsu CERNET and the national backbone router. In this paper, we analyze three 5-minute traces collected at properly selected times: 23:55:15, 12, Apr, 2014 (trace 1), 13:55:16, 20, Apr, 2014 (trace 2), 15:55:16, 21, Apr, 2014 (trace 3), representing low, peak, average traffic periods, respectively. The traffic is also classified into forward flows (FF) if destined to Net_o and backward flows (BF) if destined to Net_m.

TABLE III: ACCURACY VERIFICATION VIA EXPERIMENTS

Metrics		**Trace 1**	**Trace 2**	**Trace 3**
# of detected flows		$5,504$	$10,274$	$9,876$
FF	# of Packets	$7,872,722$	$13,441,114$	$13,437,532$
	# of Bytes	4.72 GB	8.14 GB	8.11 GB
	Avg Reordering Ratio	4.012%	3.498%	3.949%
	Avg LR_m	1.686%	1.533%	1.661%
	Avg LR_o	2.705%	2.443%	2.571%
BF	# of Packets	$9,650,460$	$16,584,584$	$16,578.534$
	# of Bytes	7.22 GB	12.79 GB	12.73 GB
	Avg Reordering Ratio	1.494%	2.555%	3.117%
	Avg LR_m	0.836%	1.338%	2.001%
	Avg LR_o	1.220%	1.749%	1.981%

We use the three traces to evaluate the algorithm. The ground truth is hard to obtain in a real network environment with uncontrollable networks. We assume that the performance on the same network remains relatively stable within a short time window (i.e., 15 minutes). Accordingly, we conducted active TCP probing within the 15-minute window after each trace passively collected. Comparing the error between the active and passive measurements for both LR_m and LR_o as shown in TABLE III, the difference is very similar to the results reported in our emulation (5.7% error rate for LR_m and 4.1% for LR_o).

VI. CONCLUSION

In this paper, we propose a lightweight passive monitoring system called *TCPBisector*, in which TCP packet loss responsibility is divided between an internal and external networks conceptually delimited by a network monitor, and quantified using LR_m and LR_o. Using our proposed TCP behavior modeling, the inference algorithm in the *TCPBisector* could accurately and efficiently estimate TCP packet loss ratios with estimation error rate $3.5 \sim 6.9\%$, but only presents computation complexity of $O(n)$ and requires a bounded memory space.

The *TCPBisector* is designed as a coarse-grained TCP performance diagnosis tool. However, *TCPBisector* provides flow based TCP loss ratio estimation. If multiple collaborating parties (e.g., between a cloud user and her service provider) deploy the *TCPBisector* systems, combining the *TCPBisector* reports from both sides on TCP packet loss in a cloud application flows will provide finer-grained view to narrow down the scope of the responsible party.

REFERENCES

[1] J. Sommers, P. Barford, N. Duffield, and A. Ron. “Improving accuracy in end-to-end packet loss measurement.” *ACM SIGCOMM 2005*, pages 157-168, 2005

[2] P. Benko and A. Veres. “A Passive Method for Estimating End-to-End TCP Packet Loss.” *IEEE Globecom 2002*, pages 2609-2613, 2002

[3] S. Ohta and T. Miyazaki. “Passive packet loss monitoring that employs the hash-based identification technique.” *9th IFIP/IEEE International symposium on Integrated Network Management, pages 2-9, 2005*

[4] A. Friedl, S. Ubik, A. Kapravelos, M. Polychronakis, and E. P. Markatos. “Realistic Passive Packet Loss Measurement for High-Speed Networks.” *Computer Science*, Volume 5537/2009, pages 1-7, 2009

[5] S. Jaiswal, G. Iannaccone, C. Diot, J. Kurose, and D.Towsley. “Measurement and classification of out-of-sequence packets in Tier-1 IP backbone.” *IEEE/ACM Transactions on networking*, Vol.15, NO.1, pages 1199-1209, Feb. 2007

[6] M. Allman, W. M.Eddy, and S. Ostermann. “Estimating Loss Rates With TCP.” *ACM Performance Evaluation Review*, pages 12-24, Dec. 2003

[7] H. Nguyen, M. Roughan. “Rigorous statistical analysis of internet loss measurements.” *IEEE/ACM Transactions on Networking*, Volume 21 Issue 3, pages 734-745, June 2013

[8] D. Zhang and D. Ionescu “Online Packet Loss Measurement and Estimation for VPN-Based Services.” *IEEE Transactions on Instrumentation and Measurement*, pages 2154-2166, Aug. 2010

[9] L. Gharai, C. Perkins, and T. Lehman. “Packet reordering, high speed networks and transport protocol performance.” *Proc. IEEE 13th Intl Conf. On Computer Comm, and Networks, ICCCN 2004*, pages 73-78, Oct. 2004

[10] A. Morton, L. Ciavattone, G. Ramachandran, S. Shalunov, and J. Perser. “Packet reordering metrics.” *RFC 4737*, 2006

[11] A. Jayasumana, N. Piratla, T. Banka, A. Bare, and R. Whitner. “Improved Packet Reordering Metrics.” *RFC5236*, 2008

[12] G. Almes, S. Kalidindi, and M. Zekauskas. “A One-way Packet Loss Metric for IPPM.” *RFC2680*, 1999

[13] R Koodli and R Ravikanth. “One-way Loss Pattern Sample Metrics.” *RFC3357*, 2002

[14] B. Veral, K. Li, and D. Lowenthal. “New Methods for Passive Estimation of TCP Round-Trip Times.” *Passive and Active Network Measurement*, pages 121-134, 2005

[15] S. Jaiswal, G. Iannaccone, C. Diot, J. Kurose, and D. Towsley “Inferring TCP connection characteristics through passive measurements.” *In the Proceedings of Infocom*, pages 1582-1592, 2004

[16] M. Ghasemi, T. Benson, and J. Rexford. “Real-time diagnosis of TCP performance in clouds.” *In the Proceedings of CoNEXT Student Workshop*, pages 57-58, Dec. 2013

[17] Information Sciences Institute, University of Southern California "Transmission Control Protocol." *RFC793*, 1981

[18] E. Brosh , G. Lubetzky-sharon, and Y. Shavitt "Spatial-temporal analysis of passive TCP measurements." *In the Proceedings of Infocom*, pages 949-959, 2005

[19] A. Jurgelionis, J. Laulajainen, M. Hirvonen, and A. I. Wang. "An Empirical Study of NetEm Network Emulation Functionalities." *In the 2011 Proceedings of ICCCN*, pages 1-6, 2011