# Towards Accurate TCP FlightSize Estimation: A History-Aware Learning Approach

Mingrui Zhang
*University of Nebraska-Lincoln, Lincoln, NE*
mzhang23@huskers.unl.edu

Jianlin Zhang
*Capital Normal University, Beijing, China*
3862@cnu.edu.cn

***Abstract*—The FlightSize of TCP is the amount of outstanding data in the network, and it affects TCP performance. Existing FlightSize estimation methods rely on instantaneous data and a limited set of variables and can misinterpret network events and leading to throughput degradation. This work proposes a history-aware learning approach that predicts TCP FlightSize by learning from historical data and leveraging a richer feature set, such as round-trip time (RTT) and the congestion control state. To validate our approach, we integrated the lightweight model into the Linux kernel. Experiments show that using the model-predicted FlightSize during TCP recovery improves application goodput by up to 40%, confirming the significant real-world benefits of our approach.**

***Index Terms*—TCP, machine learning, FlightSize**

## I. Introduction

The Transmission Control Protocol (TCP) is fundamental to the overall performance of the Internet. A key TCP parameter, the FlightSize [1], represents the amount of outstanding data in the network and is used to control sending decisions to optimize throughput while preventing congestion.

Accurately estimating the FlightSize is a persistent challenge, as the sender must infer the state of in-flight data using only indirect and often ambiguous network signals. Several limitations exist in the conventional estimation method, such as the one used by the Linux kernel. **First**, it is stateless, operating only on an instantaneous snapshot of network variables while ignoring valuable historical context. This makes it highly susceptible to misinterpreting network events, which leads to unnecessary throughput reductions. **Second**, it uses a limited set of features, primarily counters for sent and acknowledged packets, while overlooking other predictive signals like round-trip time (RTT) and the TCP congestion control state that could provide a more nuanced view of network conditions.

To overcome these FlightSize estimation limitations, we propose a history-aware, learning-based method for TCP FlightSize prediction. Our approach addresses the shortcomings of the conventional method by **1)** learning from historical data sequences, allowing the model to understand the temporal dynamics of the connection, and **2)** leveraging a richer feature set, enabling it to make more informed and context-aware predictions. This method facilitates a more adaptive and accurate estimation of TCP FlightSize, one that is better suited to diverse and changing network patterns.

Our contributions are summarized below:

1) We are the first to frame TCP FlightSize estimation as a network state interpretation problem, addressing it with a history-aware, learning-based methodology that moves beyond traditional heuristic-based approaches.
2) We identify predictive features that are currently not utilized in FlightSize estimation, such as smoothed round-trip time (srtt_us) and the congestion control state (icsk_ca_state), and validate their importance through ablation studies.
3) We develop and implement a practical prototype, integrating inference results of our lightweight prediction model into the Linux kernel to assist existing TCP schemes. Our evaluation demonstrates that this system improves application goodput by up to 40%.

## II. Related work

### A. FlightSize prediction

The Linux TCP stack uses a hand-crafted formula to calculate FlightSize, which utilizes an instantaneous snapshot of network variables. To the best of our knowledge, there is no prior work on TCP FlightSize prediction.

### B. Learning-Based Congestion Control

The proposed work is complementary to learning-based congestion control and can be combined to improve TCP performance.

Machine learning (ML) and Deep Reinforcement Learning (DRL) based Congestion Control Algorithms (CCAs) [2]–[4] learn from historical data and manage sending decisions to achieve better performance. Our approach has a similar goal to those learning-based CCAs, but uses the historical data from another angle: to accurately predict the FlightSize.

[5], [6] focused on dynamically switching between different CCAs already present in the Linux kernel to adapt to changing network conditions. Our work is complementary to theirs. We treat FlightSize itself as the learning target, positing that providing a more accurate value will directly facilitate and improve the performance of these established CCAs.

### C. ML approach for network prediction

ML models have been successfully applied to various network prediction tasks, including forecasting packet loss [7], [8], estimating throughput [9], and performing traffic classification [10], [11]. While valuable, these efforts focus

on predicting network outcomes or classifying traffic types. Our work differs by targeting the FlightSize itself, which we posit is a more complex and fundamental signal governing TCP behavior. We demonstrate that by isolating and accurately predicting FlightSize as a learnable signal, we can directly improve overall TCP performance.

## III. Background

### A. TCP FlightSize Definition

According to RFC 5681 [1], the TCP FlightSize is defined as the amount of "data that has been sent but not yet cumulatively acknowledged". The term "cumulative" signifies that a single acknowledgment (ACK) from the receiver confirms that it has successfully obtained all data packets with sequence numbers preceding that of the ACK.

This definition has two key implications. First, the FlightSize is calculated exclusively on the sender's side, using its record of sent packets and the ACK information it receives. Second, an individual packet contributes to the FlightSize for the entire duration it is in transit. That is, from the moment it is sent by the sender until an ACK confirming its successful delivery is received.

### B. Notations

To formalize the definition from the previous section, we denote the TCP FlightSize at time $t$ as $F_{TCP}(t)$. It is calculated using the following equation:

$$F_{TCP}(t) = SND(t) - RCV_{acked}(t) - Lost(t) \quad (1)$$

where $SND(t)$ is the total number of data packets sent by time $t$; $RCV_{acked}(t)$ is the total number of data packets cumulatively acknowledged by time$t$ ; and $Lost(t)$ is the total number of data packets lost by time $t$.

This notation is subtly different from the one we used in [12], which studies the FlightSize accuracy from the perspective of a network observer that has all information in the network. The Equation (1) stands from the TCP sender's aspect to study the accurate FlightSize value.

### C. Current Linux Kernel TCP FlightSize estimation

The current Linux kernel calculates the FlightSize using the expression below, which aggregates several variables from the TCP socket structure (tcp_sock). This calculation is intuitively understood as "the total number of packets sent and retransmitted", minus "the sum of packets that have been selectively acknowledged (sacked_out) or marked as lost (lost_out)".

$$\begin{aligned} &FlightSize = packets\_out + retrans\_out - \\ &(sacked\_out + lost\_out); \end{aligned} \quad (2)$$

### D. FlightSize inaccuracy

From the conclusion in our previous work [12], several factors contribute to inaccuracies in TCP's FlightSize estimation.

*1) Network delay:* The network delay alone will cause the TCP FlightSize to be overestimated.

*2) Packet Loss:* The packet loss alone will cause the TCP FlightSize to be overestimated, as the TCP sender needs time to realize the loss.

*3) Packet Reordering:* Packet reordering refers to packets arriving at the receiver in a different sequence than they were sent. The packet reordering alone will cause the TCP FlightSize to be underestimated, as the TCP sender may misinterpret the reorder event as packet loss.

### E. TCP FlightSize impact on throughput

The accuracy of the FlightSize estimation directly and significantly impacts TCP throughput for three primary reasons. **1)** FlightSize works in tandem with the Congestion Window (CWND) to govern the data sending rate; specifically, the Linux kernel permits data transmission only when the FlightSize is less than the CWND. **2)** During the critical recovery phase after a loss event, the CWND and slow-start threshold are reset based on the current FlightSize value, thereby recalibrating the sending rate and preventing further congestion [1]. **3)** Many widely deployed Congestion Control Algorithms, including CUBIC and BBR, rely on the FlightSize parameter to function correctly. Given these dependencies, an inaccurate or misleading FlightSize value causes TCP to make suboptimal decisions, which inevitably degrade throughput performance.

## IV. Challenges

To achieve data-driven learning methods to predict FlightSize, several challenges need to be solved.

1) Establishing a ground-truth prediction target.
2) Balancing feature richness with practicality.
3) Integrating learning-based inference without harming TCP performance.

## V. A History-Aware Learning Approach for FlightSize Prediction

To overcome the limitations of heuristic-based FlightSize estimation, we propose a novel history-aware learning approach. Our approach redefines the problem by establishing a more accurate prediction target and leveraging a richer set of network state features available within the TCP socket.

### A. Prediction Target

Our primary prediction target is accurate FlightSize $F_{TCP}(t)$ at a time $t$. To create an accurate target, we use packet events and the actual number of lost packets to calculate $F_{TCP}(t)$. By training a model to approximate this $F_{TCP}(t)$, we move towards the accurate FlightSize value.

### B. Feature Space for Prediction

We designed two feature selection principles to solve Challenge (2). First, all features must be readily available to the TCP sender. Second, they must be generalizable across connections, excluding connection-specific identifiers like sequence or port numbers. Adhering to these principles, we source our features with 20 features exclusively from the information available in the Linux TCP socket structure.

This feature set strategically combines variables already used in the conventional Linux FlightSize calculation (e.g., packets_out and lost_out) with new features that are not currently used. These new features, such as the smoothed round-trip time and the TCP congestion control state(icsk_ca_state), are chosen for their potential to help the model more effectively distinguish between different network conditions.

### C. Machine Learning Models

To solve this prediction task, we explore two distinct categories of machine learning models. This dual approach allows us to investigate the problem from different perspectives, reflecting the temporal nature of TCP data.

**1) Non-sequential Models**: We establish a baseline using models that make predictions based only on the most recent snapshot of the network state. We implemented several such models, including Decision Trees, Random Forests, and XGBoost.

**2) Sequential Models**. Since TCP operations are inherently a sequence of events, we investigate models designed to capture these temporal dependencies. To evaluate whether historical TCP socket information can lead to more accurate predictions, we implemented three prominent sequential architectures: Recurrent Neural Networks (RNNs), Long Short-Term Memory (LSTM) networks, and Transformer models.

## VI. Implementation

### A. Training Data collection

We use iperf3 to generate TCP traffic. We collected training data using the Mahimahi [13] network emulation platform, which allowed us to replay a diverse set of cellular [13] and wired [2] network traces. To ensure our dataset covered a wide range of network behaviors, we systematically varied the network conditions for each trace. We set the bandwidth to the values provided in the traces, varied the round-trip times to range from 1 ms to 100 ms, varied the packet loss rates from 0.1% to 10%, and varied the underlying TCP CCA used both CUBIC and BBR.

During these emulations, we used a custom-built Linux kernel module to capture the necessary feature data. This module was attached to key functions in the TCP stack responsible for data sending and ACK processing, allowing us to export relevant socket information for model training. Total number of data rows: 547,755.

**Data processing**: Processing the collected data into a final training set involves three key steps to solve Challenge(1). First, to generate the ground-truth values for our prediction target($F_{TCP}$), we must accurately determine the true packet loss($Lost(t)$). We accomplish this by performing a meticulous offline analysis of the raw TCPdump packet traces. As detailed in Alg.1, this process matches every sent packet with its corresponding acknowledgment, allowing us to reliably distinguish true packet loss from other network events.

Second, once the accurate loss count ($Lost(t)$) is established, we calculate the ground-truth FlightSize target ($F_{TCP}$) for each timestamp using the formula from Equation (1).

**Algorithm 1:** Accurate Packet Loss Detection

**Data:** $tcpdumpLogFile[N]$
/*Each line of store a Send or Receive event*/;
**Result:** $Packet[N]$
**while** $i < len(tcpdumpLogFile[])$ **do**
  **if** *tcpdumpLogFile[i].action == SEND* **then**
    $seq \leftarrow tcpdumpLogFile[i].seq$;
    $tsval \leftarrow tcpdumpLogFile[i].tsval$;
    $action \leftarrow RECV$;
    **if** $\{seq, tsval, action\}$ *exist in tcpdumpLogFile[]* **then**
      $Packet[seq, tsval] = delivered$
    **end**
    $Packet[seq, tsval] = lost$
**end**

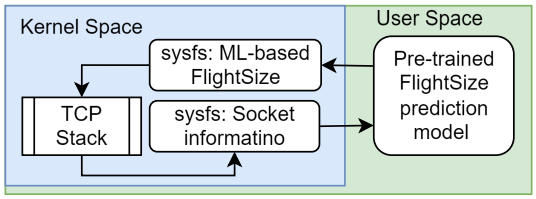


Fig. 1. Architecture of Experimental System Integration. Model predicts in user space and shares prediction results to kernel space via sysfs, eliminating inference delay.

Finally, we temporally align the captured kernel features with their corresponding ground-truth target values. This ensures that the model is trained on correctly matched feature-label pairs, completing the data processing pipeline.

### B. System Integration

To validate our approach in a realistic environment, we implemented and deployed a complete system prototype on the Mininet network emulator. We used a different platform for testing than for data collection to prevent any potential data leakage and ensure a fair evaluation.

*1) Experimental System Architecture:* To solve Challenge (3), our system employs a hybrid user-space/kernel-space architecture, as illustrated in Fig. 1. This design separates the machine learning inference from the real-time TCP stack, with communication between the two components through the sysfs virtual filesystem. This fully asynchronous communication mechanism decouples the kernel from the prediction model. The overhead is evaluated in Section VII-E.

The data flow proceeds as follows: **1)** A lightweight kernel module continuously exports relevant TCP socket variables (e.g., RTT, CWND) to a file in sysfs. **2)** In user space, our pre-trained prediction model reads these variables from the file. **3)** The model performs an inference to predict the FlightSize and writes the result back to another separate sysfs file, making it instantly available to the kernel.

*2) Kernel Patch Implementation:* We integrated our model's predictions by directly modifying the Linux kernel source code. Our approach patches the FlightSize calculation function to return our model's predicted value instead of the

standard one. This change is activated only when a TCP connection enters the Recovery state, a critical phase where the congestion window is recalculated based on the current FlightSize. This modification allows the other kernel TCP CCAs (e.g., Reno, Cubic, BBR) to utilize predicted FlightSize without changing their source code.

We opted for this direct modification instead of a more dynamic approach in [5], such as attaching an eBPF program to a Kernel Probe (Kprobe). While a Kprobe-based method would avoid a full kernel recompilation, it was not viable, as the FlightSize function is defined as an **inline function** in the kernel source. During compilation, **inline** functions are expanded, and the function address cannot be located at runtime. Consequently, a Kprobe cannot be attached, making direct source code modification our most feasible integration method. The source code is published on GitHub at github.com/zmrui/FlightSizeLearn.

## VII. Evaluation

In this section, we design and run experiments to study the following research questions (RQs) about our approach:

- **RQ1:** How accurately can machine learning models predict the FlightSize target value?
- **RQ2:** What TCP socket variables are most predictive of FlightSize?
- **RQ3:** Can the improved ML-based FlightSize approach translate into tangible TCP performance improvements in a real-world kernel implementation?

### A. Metrics

To evaluate our models, we define metrics for both predictive accuracy in an offline setting and real-world performance impact from an online, in-kernel deployment.

*1) Accuracy Metric:* We first define the accuracy as the difference between the model's predicted FlightSize($F_{pred}(t)$) and the ground-truth target ($F_{TCP}(t)$).

$$Accuracy = |F_{pred}(t) - F_{TCP}(t)| \tag{3}$$

*2) Evaluation Metric:* For offline evaluation, we use two standard statistical metrics calculated from the Accuracy values across our test dataset: Root Mean Squared Error (RMSE) and R-squared ($R^2$). A lower RMSE corresponds to a lower average prediction error, indicating a model that more accurately interprets the network state.

For the online evaluation, our primary performance indicator is goodput. It is defined as the application-level throughput measured at the receiver, which critically excludes protocol overhead and retransmitted packets. An increase in goodput directly reflects a tangible improvement in end-to-end transmission efficiency.

### B. Prediction Performance

*1) Non-Sequential Models:* To answer RQ1, we first evaluated a set of non-sequential models: Decision Tree, Random Forest, and XGBoost. These models were trained to predict FlightSize using an instantaneous snapshot of richer network features at a given time, but excluding any timestamp information. The XGBoost's hyperparameters are selected from grid searches.

| Algorithm | RMSE | $R^2$ |
|---|---|---|
| Baseline(Linux) | 110.2 | 0.540 |
| Decision Tree | 108.7 | 0.553 |
| Random Forest | 109.6 | 0.545 |
| XGBoost | 75.7 | 0.783 |

TABLE I
80%-20% Split of Models

| Algorithm | RMSE | $R^2$ |
|---|---|---|
| Decision Tree | 134.86 $\pm$ 88.34 | 0.760 $\pm$ 0.243 |
| Random Forest | 136.46 $\pm$ 88.36 | 0.763 $\pm$ 0.228 |
| XGBoost | 75.45 $\pm$ 19.39 | 0.923 $\pm$ 0.051 |

TABLE II
5-fold Cross Validation of Models

We assessed performance using a standard 80%-20% train-test split and a 5-fold cross-validation. The results are presented in TABLE I and TABLE II.

We observe that the 80%-20% split results show all models have better performance than the baseline. However, the 5-fold cross-validation results reveal that the models exhibit performance degradation and instability when tested on unseen data folds. Despite this, XGBoost consistently emerges as the superior algorithm across both evaluation methods.

*2) Sequential Data on Neural Network Models:* To answer RQ1, we further evaluated a set of sequential neural network (NN) models to determine if capturing the temporal nature of TCP data could improve prediction accuracy. We implemented three architectures in TensorFlow: Recurrent Neural Networks (RNNs), Long Short-Term Memory (LSTM), and the Transformer model. Unlike the non-sequential approach, these models were trained using historical sequences of network features, including timestamps, to inform their predictions.

The performance of these models, assessed using both 80%-20% split and 5-fold cross-validation, is detailed in TABLE III and TABLE IV. We observe that while all models achieve excellent scores on the single 80%-20% test split, their average performance is substantially lower and more volatile under 5-fold cross-validation. This discrepancy suggests that while sequential models are powerful, they may be overfitting to patterns in the training set that do not generalize well to unseen data traces.

### C. Feature Importance

To answer RQ2 and identify the most predictive variables for FlightSize, we analyzed the feature importance scores

| Neural Network Architecture | RMSE | $R^2$ |
|---|---|---|
| RNN | 23.76 | 0.989 |
| LSTM | 20.89 | 0.992 |
| Transformer | 25.13 | 0.988 |

TABLE III
80%-20% Split of Neural Network Models

| Neural Network Architecture | RMSE | $R^2$ |
|---|---|---|
| RNN | 88.64 $\pm$ 75.53 | 0.920 $\pm$ 0.097 |
| LSTM | 106.93 $\pm$ 51.37 | 0.873 $\pm$ 0.184 |
| Transformer | 62.79 $\pm$ 24.28 | 0.958 $\pm$ 0.035 |

TABLE IV
5-fold Cross Validation of Neural Network Models

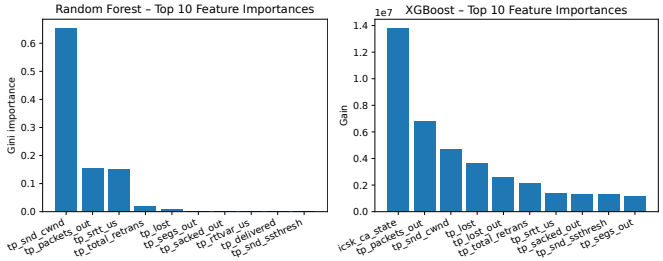


Fig. 2. Feature importance for Random Forest is measured by Gini importance, while the importance of XGBoost is the total gain across all splits. XGBoost shows the use of icsk_ca_state, which is not utilized in Linux's FlightSize estimation.

from our two best-performing non-sequential models: Random Forest and XGBoost. Then, we perform an ablation study for the XGBoost model. Due to the computation resource limit, we only perform ablation studies for the best-performing model of the non-neural network method.

*1) Model Feature Importance:* As shown in Figure 2, the analysis reveals which TCP socket variables the models found most valuable for making predictions.

We derived two insights from the results. **First**, the models largely confirmed the logic of the existing kernel implementation. Both Random Forest and XGBoost assigned high importance to features corresponding to the four variable types used by the standard formula, such as tp_packets_out, and tp_sacked_out. This indicates that the models successfully learned the most intuitive relationships from the data. **Second**, the models discovered new, predictive signals that are currently underutilized by the Linux kernel. 1) Both models identified the smoothed round-trip time as a key feature, which is absent from the kernel's static calculation. 2) The XGBoost placed significant weight on the congestion control state. This variable, which indicates whether the TCP stack is in an open, recovery, or loss state, is not used in the existing Linux FlightSize implementation.

The models' reliance on these additional features suggests that the potential of calculating FlightSize depends on network RTT and the connection's current state.

*2) Ablation Study:* To quantify the impact of specific variables on prediction accuracy, we conducted a series of ablation studies on our best-performing non-sequential model, XGBoost. We employed two strategies: removing each of the top-ranked features one by one, and cumulatively removing the top-k features.

The results of change ($\Delta$) in RMSE and $R^2$ are shown in TABLE V and TABLE VI. We observe that the one-by-one removal results in TABLE V confirm that most of the top-ranked features are indeed beneficial to the model's performance. The removal of tp_packets_out caused the most significant degradation, underscoring its role as the single most critical feature. We also observe that the importance of the new signals, the removal of the Smoothed RTT (tp_srtt_us), and the congestion avoidance state (icsk_ca_state) led to substantial performance drops. Interestingly, the removal of tp_snd_ssthresh resulted in a performance improvement, suggesting it may be a redundant or noisy feature for this task.

| Feature Removed | RMSE Change ($\Delta$) | $R^2$ Change ($\Delta$) |
|---|---|---|
| icsk_ca_state | +14.53 | -0.026 |
| tp_packets_out | +71.40 | -0.185 |
| tp_snd_cwnd | +2.24 | -0.001 |
| tp_lost | +17.12 | -0.026 |
| tp_lost_out | +7.03 | -0.002 |
| tp_total_retrans | +12.29 | -0.019 |
| tp_srtt_us | +23.79 | -0.062 |
| tp_sacked_out | +8.04 | -0.005 |
| tp_snd_ssthresh | -4.14 | -0.006 |
| tp_segs_out | +16.64 | -0.027 |
| tp_reordering | +16.68 | -0.026 |
| tp_rttvar_us | +11.19 | -0.010 |

TABLE V
TOP-10 IMPORTANCE FEATURES ONE-BY-ONE REMOVAL ABLATION STUDY FOR XGBOOST: THE RTT AND CONGESTION CONTROL STATE SIGNALS ARE IMPORTANT TO THE MODEL'S ACCURACY.

| Top-k Feature Removed | RMSE Change ($\Delta$) | $R^2$ Change ($\Delta$) |
|---|---|---|
| Top 1 | +14.53 | -0.026 |
| Top 2 | +49.71 | -0.118 |
| Top 3 | +73.64 | -0.317 |
| Top 5 | +70.86 | -0.300 |
| Top 10 | +229.91 | -2.572 |

TABLE VI
TOP-K REMOVAL ABLATION STUDY FOR XGBOOST: MODEL'S PERFORMANCE DECREASED AS MORE FEATURES WERE REMOVED.

*3) Discussion:* Our data-driven models effectively validated the logic of the kernel's existing hand-crafted formula. The models independently learned to assign high importance to the same four core variables that the kernel uses, and tp_packets_out is proving to be the most critical feature in both the standard implementation and our predictive model.

The XGBoost model demonstrated the value of incorporating signals that the kernel currently ignores. The model's reliance on the congestion control state is particularly insightful. This highlights the primary advantage of a learning-based approach: the ability to discover and leverage such state-dependent nuances that a static formula cannot capture, and provide a promising improvement direction for the hand-crafted FlightSize calculation formula.

### D. Goodput Performance Evaluation

In this section, we deploy the offline-trained XGBoost FlightSize prediction model from the previous RQs and implement related user-kernel-space communication infrastructure as illustrated in Fig. 1. In each run, we obtain goodput under various network conditions. We repeat each network condition 100 times and summarize its median and mean values.

| Network parameter | ML-Predicted FlightSize | | Original FlightSize | |
|---|---|---|---|---|
| | Median | Mean | Median | Mean |
| 10Mbps, 10ms RTT | 10.13 | 10.1365 | 10.1 | 10.0984 |
| 10Mbps, 50ms RTT | 9.36 | 9.3425 | 9.06 | 8.9749 |
| 10Mbps, 100ms RTT | 7.725 | 7.7619 | 7.015 | 7.0405 |
| 50Mbps, 10ms RTT | 49.59 | 48.8848 | 49.23 | 49.2103 |
| 50Mbps, 50ms RTT | 24.61 | 25.0994 | 19.855 | 20.027 |
| 50Mbps, 100ms RTT | 14.585 | 15.2902 | 10.715 | 10.8827 |
| 100Mbps, 10ms RTT | 94.875 | 93.7094 | 94.45 | 94.3815 |
| 100Mbps, 50ms RTT | 33.545 | 33.5748 | 22.485 | 22.8714 |
| 100Mbps, 100ms RTT | 16.595 | 18.0752 | 11.44 | 11.6461 |

TABLE VII
COMPARISON OF GOODPUT (IN MBPS) BETWEEN ML-BASED AND ORIGINAL FLIGHTSIZE ACROSS VARIOUS NETWORK CONDITIONS USING THE BBR ALGORITHM

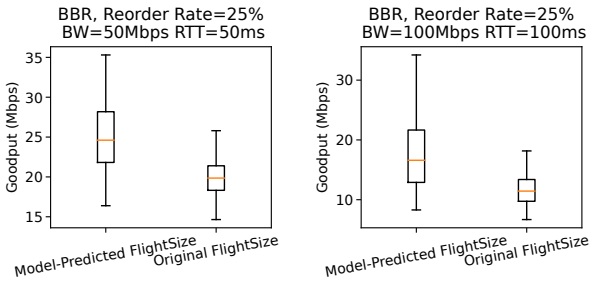


Fig. 3. Boxplot goodput based on 100 results shows model-predicted FlightSize improved TCP goodput in 50Mbps, 50ms RTT, and 100Mbps, 100ms RTT network using TCP BBR algorithm. The higher the better.

| Measurements | Proposed Interval | | Naive Polling | |
|---|---|---|---|---|
| | Avg | Max | Avg | Max |
| Inference Time(ms) | 3.0414 | 11.6208 | 4.3335 | 19.6899 |
| Communication Latency(ms) | 3.6803 | 12.7220 | 4.9457 | 20.6715 |
| CPU Overhead(%) | 9.06 | 26.20 | 390.93 | 5084.80 |

TABLE VIII
DEPLOYMENT OVERHEAD

TABLE VII shows goodput comparison between using model-predicted FlightSize and original FlightSize. Fig. 3 intuitively shows goodput improvements in two representative networks: moderate and long-haul networks. Each row in TABLE VII can be visualized with a boxplot as Fig. 3. Due to the page-length limit, we only show two plots here.

This empirical evidence confirms that the model's improved FlightSize accuracy in interpreting network state directly translates into significant, tangible improvements in TCP performance, especially in challenging high-reorder networks.

### *E. Deployment Overhead*

In this section, we evaluate the overhead of our system. We implemented and tested the interval-based deployment strategy, which performs inference only when socket information changes. For comparison, we also measured a naive continuous polling baseline to quantify the efficiency gains of our approach. We measure the Communication Latency as the time count from reading data, making a prediction, until writing back data; CPU overhead is measured as the CPU utilization of the whole Python program. We have the following measurements on a desktop with an i7-4790 CPU and 16 GB of memory, as shown in TABLE VIII. We observe that the end-to-end communication and inference latency are always in milliseconds. We also observed that the baseline method, while ensuring maximum data freshness, incurred an impractical average CPU overhead of 390.93%. In contrast, our optimized interval strategy dramatically reduces the average CPU overhead to a manageable 9.06%. We argue that this interval deployment is an acceptable trade-off between timely information and CPU overhead for many server-side deployments.

### *F. Discussion, Limitations and Future works*

Our live deployment experiments demonstrate that using a model-predicted FlightSize, particularly during TCP recovery, can yield significant goodput improvements over the standard kernel implementation. This confirms our core hypothesis that a more accurate, state-aware FlightSize leads to tangible performance benefits.

However, our offline evaluations also revealed a key limitation. The performance instability observed during 5-fold cross-validation suggests that our models may be overfitting to the specific characteristics of the network traces used for training. While the models perform well on data similar to what they have seen, their ability to generalize to entirely new network patterns could be improved.

Therefore, the most critical direction for future work is to expand the training dataset. By incorporating a larger and more diverse collection of network traces from various real-world scenarios, we aim to build more robust models that can generalize effectively across a wider range of network conditions and behaviors.

## VIII. CONCLUSION

We proposed a data-driven approach to improving TCP FlightSize accuracy. We discovered useful features that are underutilized in the current Linux TCP FlightSize calculation. Our evaluations show that accurate FlightSize can improve TCP performance.

## REFERENCES

[1] E. Blanton, D. V. Paxson, and M. Allman, "TCP Congestion Control," RFC 5681, Sep. 2009. [Online]. Available: https://www.rfc-editor.org/info/rfc5681

[2] S. Abbasloo, C.-Y. Yen, and H. J. Chao, "Classic meets modern: a pragmatic learning-based congestion control for the internet," in *Proceedings of the ACM SIGCOMM*, 2020, p. 632–647.

[3] N. Jay, N. Rotman, B. Godfrey, M. Schapira, and A. Tamar, "A deep reinforcement learning perspective on internet congestion control," in *Proceedings of the 36th ICML*, vol. 97, 2019, pp. 3050–3059.

[4] F. Y. Yan, J. Ma, G. D. Hill, D. Raghavan, R. S. Wahby, P. Levis, and K. Winstein, "Pantheon: the training ground for internet congestion-control research," in *Proceedings of USENIX ATC*, 2018, pp. 731–743.

[5] J. Zhou, X. Qiu, Z. Li, G. Tyson, Q. Li, J. Duan, and Y. Wang, "Antelope: A framework for dynamic selection of congestion control algorithms," in *Proceedings of the IEEE ICNP*, 2021, pp. 1–11.

[6] C.-Y. Yen, S. Abbasloo, and H. J. Chao, "Computers can learn from the heuristic designs and master internet congestion control," in *Proceedings of ACM SIGCOMM*, New York, NY, USA, 2023, p. 255–274.

[7] A. Giannakou, D. Dwivedi, and S. Peisert, "A machine learning approach for packet loss prediction in science flows," *Future Generation Computer Systems*, vol. 102, pp. 190–197, 2020.

[8] M. Welzl, S. Islam, and M. von Stephanides, "Real-time tcp packet loss prediction using machine learning," *IEEE Access*, vol. 12, pp. 159 622–159 634, 2024.

[9] D. Minovski, N. Ögren, K. Mitra, and C. Åhlund, "Throughput prediction using machine learning in lte and 5g networks," *IEEE Transactions on Mobile Computing*, vol. 22, no. 3, pp. 1825–1840, 2023.

[10] R. Kumar, M. Swarnkar, G. Singal, and N. Kumar, "Iot network traffic classification using machine learning algorithms: An experimental analysis," *IEEE Internet of Things Journal*, vol. 9, no. 2, pp. 989–1008, 2022.

[11] I. Guarino, G. Aceto, D. Ciuonzo, A. Montieri, V. Persico, and A. Pescapé, "Fine-grained traffic prediction of communication-and-collaboration apps via deep-learning: A first look at explainability," in *Proceedings of the IEEE ICC*, 2023, pp. 1609–1615.

[12] M. Zhang, P. Ha, H. Bagheri, and L. Xu, "Accuracy evaluation of TCP FlightSize estimation: Analytical and experimental study," in *Proceedings of the International Conference on Computing, Networking and Communications*, 2025, pp. 671–677.

[13] R. Netravali, A. Sivaraman, S. Das, A. Goyal, K. Winstein, J. Mickens, and H. Balakrishnan, "Mahimahi: accurate record-and-replay for HTTP," in *Proceedings of USENIX ATC*, 2015, pp. 417–429.