Bottleneck Type Detection with Machine Learning

Fatih Berkay Sarpkaya
Department of Electrical and Computer Engineering
NYU Tandon School of Engineering
Brooklyn, NY 11201
fbs6417@nyu.edu

***Abstract*—Active Queue Management (AQM) schemes are widely deployed in the Internet to manage congestion at network bottlenecks. However, they differ significantly in behavior, and not all TCP Congestion Control Algorithms (CCAs) perform effectively with every AQM type. In the Internet, the lack of bottleneck type awareness on the sender side presents challenges for optimizing sender strategies and can lead to performance degradation. This study explores the feasibility of using machine learning (ML) models to predict the AQM type at the bottleneck. Data was collected from a controlled experimental environment, measuring key metrics such as Round-Trip Time (RTT) and Congestion Window (CWND) over time, under varying bottleneck capacities, CCAs, number of flows, and base RTTs. A Random Forest and a Neural Network model were trained and evaluated to classify the bottleneck queue management scheme as either FIFO or PIE. Results show that the AQM type could be predicted with up to 90% accuracy in our setup, demonstrating the potential of ML-based AQM prediction to improve congestion control algorithms and optimize performance, particularly in networks lacking AQM awareness.**

***Index Terms*—Congestion Control (CC), Active Queue Management (AQM), FIFO, PIE, Machine Learning (ML), Neural Network, Random Forest**

I. Introduction

Congestion occurs in a network when the incoming data rate exceeds the network's capacity. In such cases, queuing occurs, and packets must wait in the queue until they can be transmitted. This is a significant issue because, if not properly managed, it can lead to high delays (due to long queues in buffers) and packet losses (caused by buffer overflows) [1]. These effects are particularly problematic for low-latency applications, such as online gaming, remote control, and video conferencing [2]. To address congestion, various mechanisms—both traditional and machine learning (ML) based—have been developed at the end-host and router levels. At end-hosts, TCP Congestion Control Algorithms (CCAs) dynamically adjust transmission rates based on network feedback [3], [4]. At routers, Active Queue Management (AQM) techniques manage congestion by controlling queue lengths and signaling network conditions to senders [5]. These techniques, along with their advantages and disadvantages, have been discussed in detail in the interim report [6].

The primary motivation of this paper comes from the observation that current CCAs and AQM schemes do not always interact effectively. For example, Low Latency, Low Loss, Scalable Throughput (L4S) flows, such as TCP Prague [7], designed for low-latency communication, do not coexist well with traditional CCAs like CUBIC [8] in a single-queue Explicit Congestion Notification (ECN) [9] bottleneck, where L4S flows tend to dominate, leading to performance degradation or even starvation for the other flows [10]. Another example is that real-time communication (RTC) applications using delay-based CCAs do not react to loss signals from AQMs, resulting in unnecessary loss recovery and performance degradation [11].

Such mismatches between sender-side algorithms and router-side algorithms, and their implications for performance, highlight the importance of understanding the behavior of AQMs from the sender's perspective. A sender that understands the bottleneck's queue management scheme can adjust its strategy accordingly, leading to improved overall performance and better coexistence between flows. Possible applications will be given in detail in Section V.

Despite the potential benefits, predicting the AQM type at a bottleneck is challenging due to the diverse range of AQMs, each with unique behaviors and configurations. Examples include, but are not limited to FIFO, RED [12], CoDel [13], PIE [14], FQ-CoDel [15], and DualPI2 [16]. While FIFO (First In, First Out) is not inherently an AQM, as it does not actively manage queues, we will treat it as an AQM type in this paper for simplicity. While these algorithms share some similarities, they also exhibit distinct behaviors, making it difficult to identify all of them. Additionally, external factors, such as background traffic, queue size, and varying network conditions, can hide AQM characteristics, making prediction even more challenging. Traditional techniques for bottleneck characterization often rely on heuristic-based approaches or active measurements, which can be less adaptable to diverse network environments [17], [18]. ML could be a promising approach for this problem by leveraging observed network metrics to identify patterns that distinguish between AQM types.

This paper explores the feasibility of using ML models to predict the AQM type at the bottleneck. Training and evaluation data were collected in a controlled experimental environment with varying parameters such as bottleneck capacities, CCAs, number of flows, and base Round-Trip Times (RTTs). Key network metrics, including RTT and Congestion Window (CWND), were measured over time and used to train and evaluate two ML models: Random Forest and Neural

Network. The results demonstrate that ML-based approaches have significant potential in accurately classifying AQMs as FIFO or PIE, providing insights that can improve CCAs and optimize network performance.

The rest of this paper is organized as follows: Section II discusses the AQM prediction problem in networks, emphasizing its importance, outlining the challenges involved, and reviewing related work from the literature. Section III describes the proposed solution including network description, data collection process, feature engineering, training, evaluation, and results. Section IV presents an extension idea and its results on top of the initial evaluation. Section V discusses the possible applications of our AQM prediction scheme in networking. Finally, Section VI concludes the paper and highlights directions for future work. Our artifacts for data collection and ML implementation are publicly available, allowing others to build upon and validate our research[1].

II. Background and Motivation

A. Problem Overview and Importance: Mismatch between CCAs and AQMs

In modern networks, congestion is managed through a combination of sender-side CCAs and router-side AQM implementations. However, the interactions between these two mechanisms are not always flawless, leading to performance degradation. While AQMs aim to mitigate congestion by actively managing queue lengths and signaling congestion to senders, their effectiveness depends heavily on the sender's ability to correctly interpret these signals. Similarly, the performance of a sender's CCA relies on the bottleneck's behavior, which is determined by the underlying AQM scheme. Without knowledge of the bottleneck type along the connection path, a sender cannot ensure that its CCA is optimally suited to the network conditions, resulting in suboptimal performance.

For example, delay-based CCAs like Copa [19] and model-based CCAs like BBR [20] rely primarily on measured delay information and are less sensitive to packet loss. In contrast, many AQMs use packet-dropping strategies to signal congestion to the sender, creating a mismatch in understanding between the sender and the AQM. This mismatch can lead to excessive queuing at the bottleneck, which is particularly problematic for low-latency applications.

Another example involves TCP Prague, a CCA specifically designed for low-latency communication. If a sender using TCP Prague does not observe the expected low latency, the cause might be the absence of an appropriate AQM at the bottleneck. For instance, a simple FIFO queue would not provide low latency, whereas DualPI2 is specifically designed to support such traffic. If a sender can identify or estimate the bottleneck type, it can better diagnose performance issues and adapt its strategy accordingly, ultimately improving overall efficiency and flow coexistence.

[1]Artifacts are available at: https://github.com/fatihsarpkaya/bottleneck-prediction

B. Challenges in Solving the Problem

Estimating the bottleneck type in a network is a complex task due to the diversity of AQMs deployed across networks. AQMs such as FIFO, RED, CoDel, PIE, FQ-CoDel, and DualPI2 vary significantly in their configurations and operational behaviors. Each implements a distinct congestion management strategy, ranging from drop-based methods to ECN-based signaling. This diversity makes it challenging for a sender to universally adapt its strategy without specific knowledge of the bottleneck's AQM type.

Traditional methods for bottleneck characterization, such as active probing and heuristic-based approaches, also have limitations. While theoretical models may differentiate and classify AQM behaviors under controlled conditions, real-world networks introduce complications such as background traffic, dynamic network conditions, and varying congestion levels. These factors can hide the observable patterns of AQMs, making accurate identification significantly more difficult.

An additional challenge arises in dynamic network configurations where multiple routers with different AQMs exist along the path, and the bottleneck location shifts between them. For instance, consider a network with two routers: one using FIFO and the other using PIE. If the bottleneck alternates between these two routers due to traffic fluctuations or changes in flow dynamics, identifying the active AQM becomes difficult. Such scenarios indicates the need for robust, adaptive methods capable of operating in highly dynamic and complex environments.

C. Related Work

In the literature there are limited number of studies to estimate the bottleneck types. In [18], the authors propose *TADA*, an active measurement tool that detects whether a bottleneck router has an AQM by analyzing queue delay and packet loss trends. However, it does not classify different AQM types and its accuracy decreases significantly in the presence of high background traffic or complex network conditions, limiting its applicability in real-world scenarios. In [11], the authors analyze the temporal behaviors of packet loss trends to infer AQM types, categorizing them into three groups (Spike, Cliff, Hill) based on distinct loss patterns. While their method achieves high accuracy under simulated conditions, it relies on rule-based inference and linear fitting, making it less effective in handling diverse real-world scenarios with highly dynamic network environments. In [17], the authors propose a heuristic mechanism to detect the presence of a single queue, ECN-capable AQM, which poses challenges when TCP Prague coexists with other TCP CCAs. The detection relies primarily on RTT variation measurements. On identifying this type of queue, TCP Prague fallbacks to Reno-like behavior to ensure compatibility. While this approach offers a lightweight and practical solution for identifying such problematic AQMs, it lacks the capability to differentiate between various AQM types, limiting its broader applicability in diverse network scenarios.

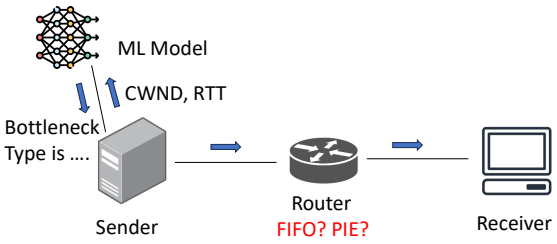


Fig. 1: Illustration of the problem and proposed mechanism. ML model on the sender predicts the AQM type on the bottleneck.

Given these challenges, there is a need for advanced and scalable approaches to reliably identify AQM types in diverse and dynamic network environments. ML emerges as a promising solution, leveraging network metrics to predict the bottleneck type and improve congestion control strategies.

III. Proposed Solution and Results

Our approach leverages ML to predict the AQM type at the bottleneck. This project is inspired by the study in [21], where we implemented the core idea and introduced an extension, detailed in Section IV. Similar to the referenced study, our system consists of a sender equipped with an ML model, with Random Forest and Neural Network implementations in our case. During the training phase, the sender collects network metrics, such as CWND and RTT, over time. After feature engineering, the ML model is trained on data collected from various experimental samples and subsequently evaluated on data from a separate test environment.

The primary objective of the sender is to classify the bottleneck AQM as either FIFO or PIE. We formulated this task as a binary classification problem for simplicity, providing a baseline to evaluate ML's effectiveness in identifying AQM types under the given network conditions. The mechanism of the system is illustrated in Figure 1.

A. Network Description

This problem arises in the context of modern Internet environments, where bottleneck routers may implement diverse queuing mechanisms such as AQMs or traditional drop-tail. The analysis applies to both wired and wireless networks, as both have challenges from varying bottleneck behaviors.

For this study, we focus on a wired network environment, assuming a bottleneck link with varying buffer sizes, capacities, base RTTs, and the number of flows. This controlled setup allows us to isolate and analyze the performance of our proposed ML-based approach under different network configurations.

B. Data Collection & Experiment Methodology

The data is collected on the sender side and does not require additional hardware, as metrics like congestion window and RTT are readily available within standard TCP implementations. To gather data for both training and evaluation, we conducted a series of systematic experiments. In these experiments, one sender implements an ML model to predict the AQM type, while other senders generate background traffic without using an ML model. This section describes our experimental methodology and the data collection process in detail.

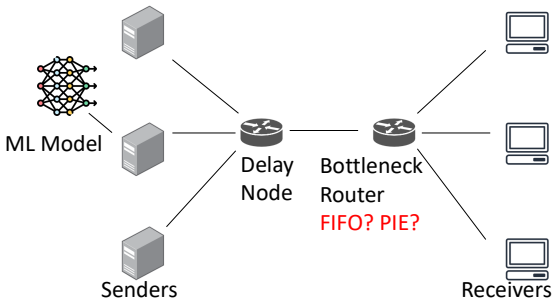


Fig. 2: Experiment Topology for Training Data

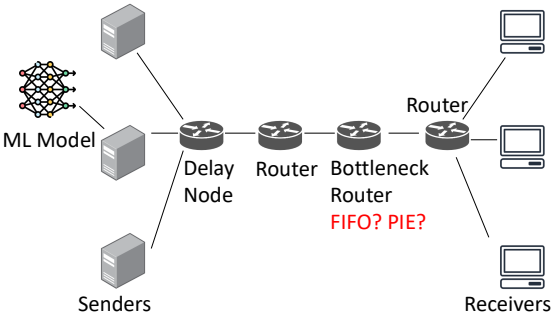


Fig. 3: Experiment Topology for Testing Data. Its network parameters differ from those used in the training topology to ensure diverse evaluation conditions.

Experiment Platform: We conduct experiments on FABRIC [22], a national scale experimental networking testbed. Each node in our experiment is a virtual machine running Ubuntu 22.04, with 4 cores and 32 GB RAM.

Topology for Training Data: The network topology for training data consists of three senders and three receivers, connected through a delay node and a bottleneck router, as illustrated in Figure 2. The delay node employs `netem` to emulate a base RTT, applying half of the delay in each direction. The bottleneck router is configured with a token bucket filter (`tc-htb`) to set the bandwidth and buffer size.

Topology for Evaluation Data: For evaluation, we extended the topology to include three routers with a single bottleneck, using different parameters for network configurations. This setup is shown in Figure 3.

Implemented AQM Types: We have implemented FIFO and PIE at the bottleneck for classification purposes. Detailed explanations of these AQM types are provided below:

- **FIFO:** the simplest queue management approach, using a single drop-tail queue without ECN support. It is implemented using `tc-bfifo`. An illustration of this mechanism is shown in Figure 4.
- **PIE:** a single-queue mechanism that operates in three main steps. First, it calculates the current queue delay by comparing the enqueue and dequeue times of packets. Next, it computes a drop probability (`drop_prob`) based

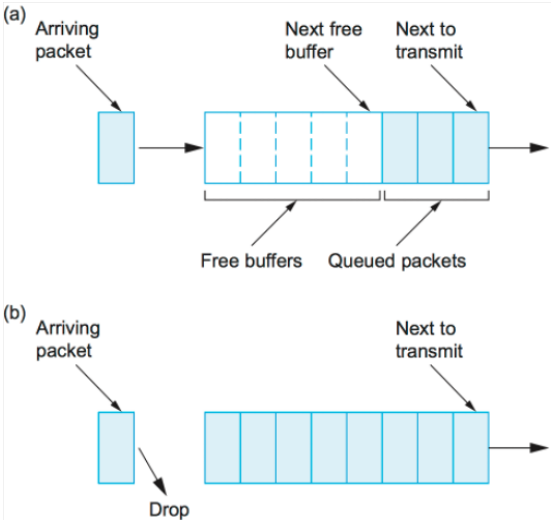


Fig. 4: FIFO - Tail Drop Structure [1].

on the difference between the current and target delays, as well as the change in delay since the last update. Finally, it decides whether to drop or enqueue a packet based on the computed probability. We use `tc-pie` with ECN disabled. An illustration of this mechanism is provided in Figure 5.

Network Setting: We used different network parameters for the training and testing environments. The parameters are listed below:

- **Bottleneck Buffer Size (n x BDP):**
  - Training: [2]
  - Testing: [1, 3]
- **Bottleneck Capacity (Mbps):**
  - Training: [20, 50, 100, 200, 500, 1000]
  - Testing: [40, 150]
- **Base RTT (ms):**
  - Training: [5, 10, 20, 50, 100]
  - Testing: [15, 60]
- **PIE target (ms):**
  - Training: [10]
  - Testing: [5, 15]
- **TCP CCA:**
  - Training: [Cubic, Reno]
  - Testing: [Cubic, Reno, BBR]
- **Number of flows from each sender:**
  - Training: [5, 10, 25]
  - Testing: [2, 10, 20]

Flow Generation: For each network configuration, we generate a single TCP flow from the main sender, which incorporates the ML model, and multiple TCP flows from the other senders using the `iperf3` utility. Each experiment runs for a duration of 20 seconds. During this time, we record the output of the `ss` command on the main sender to collect time-series data for CWND and RTT values.

By using this methodology, we collected data from 960 experiments for training and 198 experiments for testing.

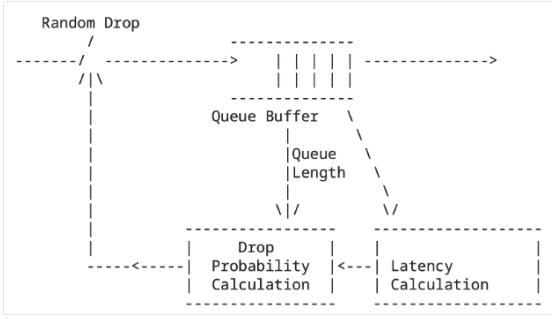


Fig. 5: PIE Structure [14].

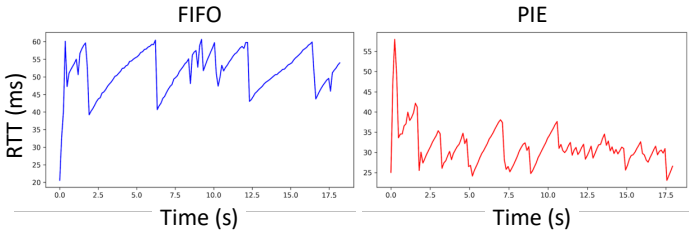


Fig. 6: CUBIC flow RTT vs Time Plots for FIFO and PIE under 100 Mbps Bottleneck Capacity, 20 ms base RTT, 2BDP buffer size, 10 ms PIE target, 1 CUBIC and 1 RENO flow background traffic.

C. Feature Engineering

After the experiments and data collection process, we obtained time-series CWND and RTT data for each experiment.

In Figure 6, an example of RTT versus time is shown for both FIFO and PIE. As observed in this figure, under FIFO, the RTT reaches the full buffer size latency (20 ms base RTT + 40 ms buffer size). In contrast, with PIE, the RTT remains around its target delay on average (20 ms base RTT + 10 ms target delay). This behavior is due to the different queue management strategies: FIFO uses a tail-drop strategy, meaning it does not drop packets until the buffer is full, while PIE uses a probabilistic early-drop strategy to maintain the delay below its target.

From Figure 6, it may appear straightforward to differentiate between the two bottleneck types by observing their RTT patterns. However, this is not always the case. In Figure 7, the differences between FIFO and PIE are less distinct, and many similar scenarios can be generated. Therefore, it can be claimed that the two bottleneck types exhibit both distinct and similar behaviors depending on the network environment, making identification challenging.

Considering this observation, 72 features (36 for CWND and 36 for RTT) are generated from the collected time-series data, following the methodology in the study [21]. Examples of these features include the first and second-order gradients and their mean, variance, and maximum; the number of local minima and maxima; total variation; and the mean and variance of the local maxima, among others.

After the feature engineering process, each experiment is represented by a single data point consisting of 72 features and one label (0 for FIFO or 1 for PIE).

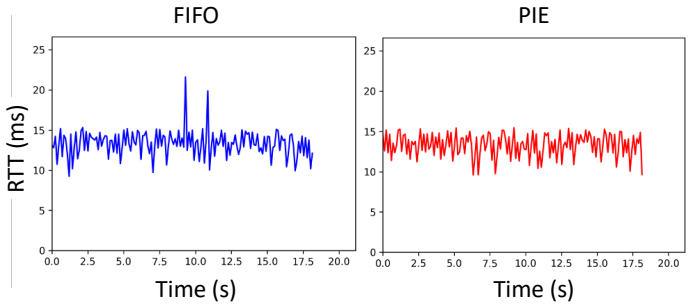


Fig. 7: CUBIC flow RTT vs Time Plots for FIFO and PIE under 100 Mbps Bottleneck Capacity, 5 ms base RTT, 2BDP buffer size, 10 ms PIE target, 1 CUBIC and 1 RENO flow background traffic. It is not easy to differentiate FIFO and PIE in this case.

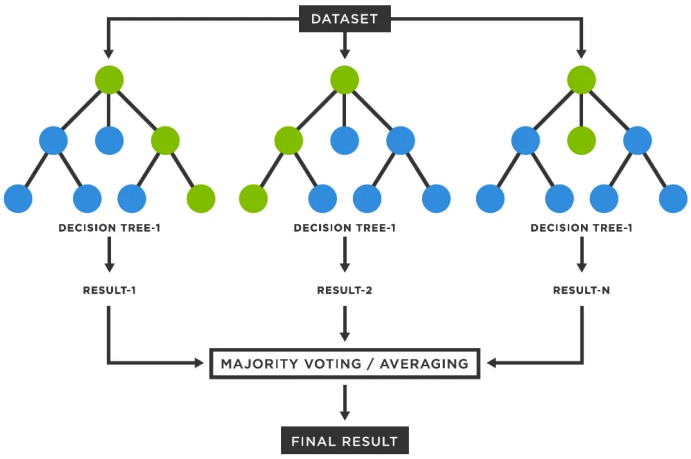


Fig. 8: Structure of the Random Forest Model. Multiple decision trees work together to classify bottleneck types [24].

D. Training

With the data collected from our experiments, we prepared it for training and evaluation of machine learning models as explained above. Then, we trained and tested Random Forest and Neural Network models to classify bottleneck types.

1) Random Forest Model: The Random Forest classifier is a simple but powerful ensemble learning algorithm that operates by constructing multiple decision trees during training and outputting the class label that is the mode of the classes (classification) or mean prediction (regression) of the individual trees [23]. The structure of a Random Forest model is illustrated in Figure 8.

We chose this model primarily for its simplicity and minimal need for complex hyperparameter tuning. To optimize its performance, we tuned the number of trees (`n_estimators`) using cross-validation. Other hyperparameters, such as the maximum depth of trees (`max_depth`), were kept at their default values.

2) Neural Network Model: A Neural Network (NN) is a powerful model capable of capturing non-linear relationships and complex dependencies among features, making it suitable for tasks where traditional models like Random Forest may struggle [25]. The structure of the Neural Network is shown in Figure 9. We selected this model because our features (e.g., gradients, maxima, local minima) may exhibit complex interactions that are challenging for simpler models to capture. The NN's ability to learn and identify these complex patterns makes it a strong candidate for this classification task.

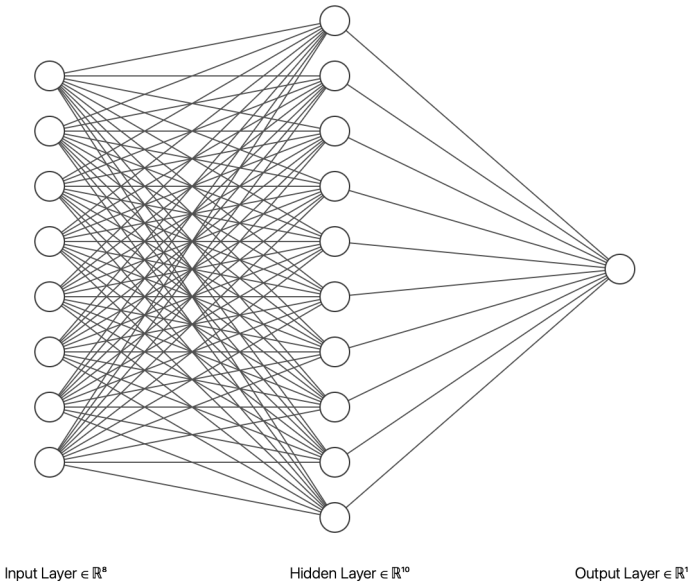


Fig. 9: Structure of the Neural Network Model [26].

Our implementation employs a fully connected feed-forward architecture with one hidden layer. The input layer consists of 72 neurons, corresponding to the total number of features. The hidden layer uses the ReLU (Rectified Linear Unit) activation function to introduce non-linearity while maintaining computational efficiency. The output layer contains a single neuron with a sigmoid activation function for binary classification. The ADAM optimizer is selected for training, and the binary cross-entropy loss function is used, as it is well-suited for binary classification tasks.

To optimize the model's performance, we tuned several parameters through cross-validation, including the number of neurons in the hidden layer, the number of training epochs, the regularization parameter, and the batch size. The best-performing configuration from cross-validation was applied to the test dataset for evaluation. For instance, in our experiments, the optimal configuration included 8 hidden layer neurons, 100 epochs, a regularization parameter (α) of 10^{-4}, and a batch size of 24.

E. Evaluation

After the training process, we evaluated our classifiers on the test dataset. The results for the Random Forest and Neural Network models are presented in Figure 10. The overall accuracy of the classifiers was 83% for the Random Forest and 90% for the Neural Network, demonstrating strong performance in predicting AQM types. Additionally, the F1 scores for the Random Forest and Neural Network models were 0.83 and 0.90, respectively, further showing their classification effectiveness.

As noted earlier, the test environment has different network parameters from the training environment, representing unseen conditions. Despite this, both models achieved high accuracy, highlighting their generalizability to varied network scenarios. While these results are promising, we acknowledge that further testing on more diverse and complex environments is necessary to fully evaluate the robustness of the models. Nevertheless, our findings show the potential of machine learning approaches for AQM classification.

Among the two models, the Neural Network outperformed the Random Forest in both accuracy and F1 score. This suggests that the Neural Network's ability to capture complex, non-linear relationships among features provides a clear advantage for this task.

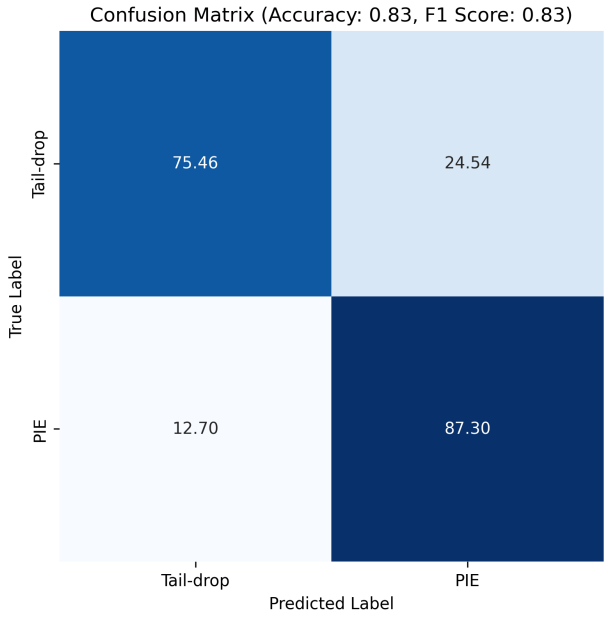
A closer examination of the results reveals that most misclassifications occurred in the Tail-Drop class for both models. This could indicate that PIE's probabilistic early-dropping mechanism produces more distinct and recognizable patterns, making it easier to classify compared to the simpler Tail-Drop mechanism. This insight could be valuable for refining the models further, especially for scenarios involving less distinct queuing behaviors.

1) Feature Importance: As noted, we have 72 features for each data point, 36 related to CWND and 36 related to RTT. After the Random Forest evaluation, we obtained feature importance values to understand which features significantly influence the classification and which features have relatively little impact. The top 10 most important features and the bottom 10 least important features are shown in Figure 11.

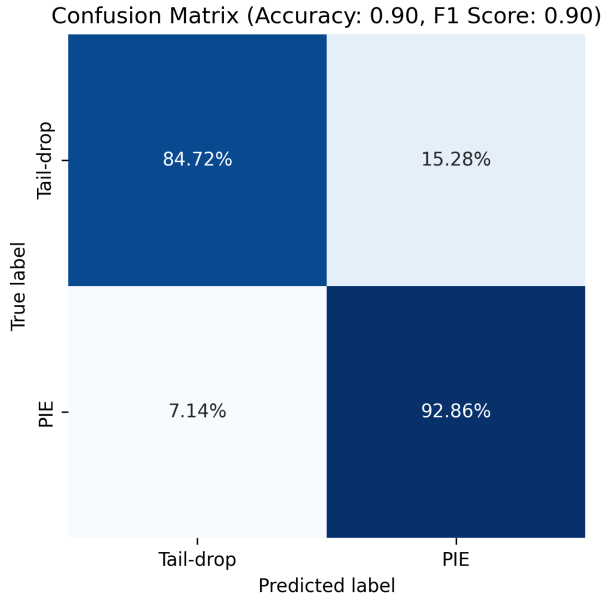
From our evaluation, RTT-related features dominate the top 10 most important features, such as mean of RTT gradient (RTT-mean-grad), standard deviation of RTT at local minima (RTT-std-localMin), and mean RTT value (RTT-mean-value). These features provide insights into the variations in RTT trends and local minima or maxima, which align well with the behaviors of FIFO and PIE AQMs. PIE's early-dropping mechanism aims to control delay, making RTT-based features more indicative of the bottleneck's AQM type.

In contrast, CWND-related features are dominant in the bottom 10 least important features, such as minimum sum of gradients (CWND-min-subgrad), L2-norm squared total variation of CWND values (CWND-totalVar-value), and sum of gradients (CWND-sum-grad). These features indicate congestion window behavior, which, while informative for general congestion control, appears less indicative of the distinction between FIFO and PIE in this specific environment. This could be due to both AQMs generating similar congestion window patterns when interacting with senders, especially in our test setup.

The dominance of RTT features in importance rankings is about the working principles of FIFO and PIE. FIFO relies on a straightforward tail-drop mechanism, leading to predictable RTT increases under congestion. PIE, on the other hand, actively manages delay with early-dropping mechanisms, resulting in RTT trends that are distinguishable from FIFO.



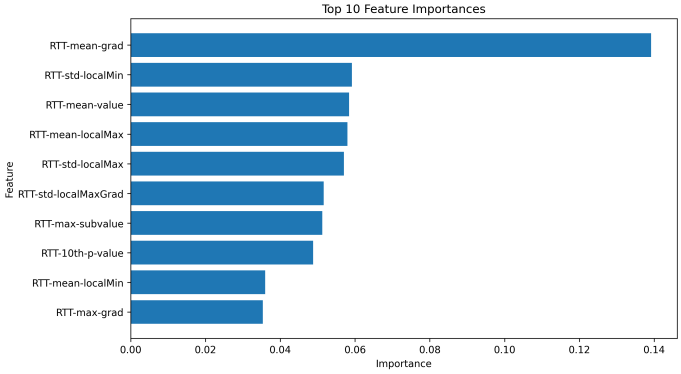
(a) Random Forest



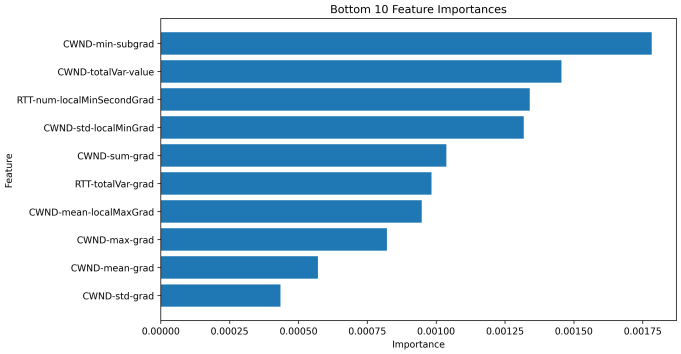
(b) Neural Network

Fig. 10: Confusion matrices for the two ML classifiers: (a) Random Forest, and (b) Neural Network.

It should be noted, however, that these results are specific to the experimental setup used in this study, including the particular AQM types (FIFO and PIE) and network configurations. In different network environments or with other AQM types, the relative importance of features may vary. Future work could explore how feature importance evolves in more diverse and complex scenarios.



(a) Top 10 features



(b) Bottom 10 features

Fig. 11: Feature importance for the two Random Forest classifier: (a) Top 10, and (b) Bottom 10.

IV. Extension: Prediction before the end of the connection

Until this point, we have implemented the work outlined in [21] using our custom topology and ML implementations. In this section, we introduce an extension to build upon this study. This project does not include a real-time implementation; however, with a simple modification, we can make some meaningful observations.

Ideally, we aim to predict the AQM type during the connection rather than after it ends. In the current implementation, the model evaluates performance using data collected over the entire connection duration. For this extension, we test the model's prediction accuracy using data collected from only the initial stages of the connection. Specifically, after training the model as described in previous sections, we evaluate its performance on test data collected from the first 1 second, 5 seconds, 10 seconds, and 20 seconds of the connection, respectively. This allows us to observe how early in the connection the model can make accurate predictions. As a reminder, each experiment in this study has a total connection duration of 20 seconds, as previously indicated. The objective of this extension is to assess whether the AQM type can be reliably predicted before the connection ends and, if so, determine the earliest point at which reasonably good prediction performance can be achieved.

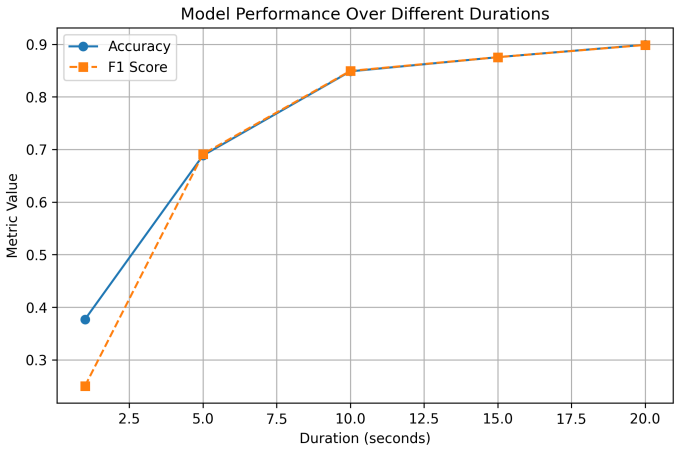


Fig. 12: Neural Network model accuracy and F1 score as a function of connection duration. The plot demonstrates the model's ability to predict AQM types at different stages of the connection, showing how performance improves as more data becomes available.

We implemented this idea using our Neural Network classifier, which demonstrated superior performance compared to the Random Forest classifier, as discussed previously. The results of this implementation are shown in Figure 12.

According to the results, both the accuracy and F1 score improve as the duration of the connection increases. For data collected within the first second, the model achieves an accuracy of approximately 40% and an F1 score of 30%, indicating limited predictive power at very early stages. However, as the duration extends to 5 seconds, both metrics significantly improve, with accuracy exceeding 70% and F1 score reaching similar levels. By the 10-second mark, the model achieves an accuracy of approximately 85% and continues to improve slightly until the full connection duration of 20 seconds, where it reaches its peak performance of around 90% for both accuracy and F1 score.

These results indicate that the Neural Network model can provide reasonably accurate predictions of the AQM type early in the connection, with substantial performance achieved as early as 5 to 10 seconds. This may suggest that near-real-time prediction is feasible, offering potential benefits for dynamic adaptation during live network conditions.

To give context to this extension, it is worth noting that TCP connections vary widely in duration depending on the application and network conditions. For example, many web-based applications rely on short-lived TCP connections, often lasting only a few seconds, while file transfers and streaming applications involve long-lived connections that persist for extended periods. According to [27], about 40% of TCP traffic lasts less than 2 seconds, while only 1.5% of the traffic lasts longer than 15 minutes. Additionally, studies have shown that the average duration of long-lived TCP flows ranges from approximately 24 to 29 seconds, depending on the direction of the traffic and the time period studied. Short-lived TCP flows, in contrast, have an average duration of around 1.28 seconds.

In this extension, according to our results, our approach is not appropriate for many short-lived connections, because our model requires at least a few seconds to predict the AQM type with a reasonable accuracy. However, for long-lived connections, this model's ability to predict the AQM type during the early stages of the connection can provide meaningful insights and allow the sender to adapt its behavior dynamically.

V. Possible Applications

In this study, our primary focus is to predict the AQM type at the bottleneck and demonstrate the potential of ML for this task. However, simply predicting the bottleneck type does not inherently improve network performance or address a specific issue within the network. To realize performance improvements or resolve challenges that would otherwise persist without this ML implementation, the sender or network provider must take appropriate actions based on the predicted bottleneck type. In this section, we discuss the motivation for implementing an AQM prediction scheme and explore how existing applications or services could leverage this model to enhance their performance or address specific challenges.

A key example is about the L4S architecture, specifically, the interaction between TCP Prague [7] and classic TCP CCAs, such as TCP CUBIC. TCP Prague, a scalable congestion control mechanism, adjusts its CWND based on the ratio of ECN feedback from the network. In contrast, classic TCP algorithms like TCP CUBIC rely solely on the presence or absence of ECN signals and adjust their CWND at a fixed rate. This fundamental difference in behavior creates a problem in single-queue bottlenecks supporting ECN. In such scenarios, TCP Prague tends to dominate, leading to the starvation of classic TCP flows and causing severe fairness issues. This issue is briefly discussed in the traditional methods section of Section VI in the interim report [6].

To address this problem, TCP Prague needs the ability to detect single-queue bottlenecks with ECN and fallback to classic congestion control behavior instead of its scalable approach. Our proposed ML model could play a critical role in this detection process, helping to identify bottleneck types and enabling TCP Prague to adapt its behavior accordingly. By doing so, potential unfairness issues can be mitigated, ensuring more fair bandwidth sharing among flows.

Moreover, the challenges of co-existence are not limited to single-queue bottlenecks with ECN. TCP Prague exhibits distinct behaviors in various bottleneck types, and some of these scenarios can cause problematic bandwidth allocation when co-existing with classic flows. Studies such as [10] and [28] identify and discuss these issues in detail. In summary, this use case demonstrates how our model can enable TCP Prague to detect problematic bottleneck types and adapt its behavior dynamically for both improved performance and fair resource allocation across all flows sharing the same bottleneck.

Another potential use case for AQM prediction is highlighted in [11]. This study demonstrates how AQM inference can enhance RTC applications by allowing them to adapt their loss recovery mechanisms, such as forward error correction (FEC), based on the inferred AQM type. Similarly, our scheme could be used to optimize FEC strategies dynamically, with the sender adjusting redundancy rates according to the predicted AQM type.

These applications shows the broader potential of AQM prediction in addressing diverse networking challenges.

VI. Conclusion

This paper discussed the feasibility of using ML models to predict the AQM type at bottlenecks. By using metrics such as RTT and CWND, we trained and evaluated Random Forest and Neural Network classifiers to distinguish between FIFO and PIE AQMs. The results demonstrated that both models, particularly the Neural Network, achieved high accuracy and F1 scores, showing the potential for ML-based AQM classification.

Additionally, we extended the study to investigate the feasibility of early AQM prediction during the initial stages of a connection. The Neural Network model demonstrated significant prediction performance within the first 5 to 10 seconds of a connection, suggesting that near-real-time AQM inference is achievable for long-lived TCP connections. However, short-lived connections remain challenging due to the limited time available for data collection and inference.

The proposed AQM prediction model has broad implications for improving network performance and addressing some challenges in diverse use cases. For example, it can enable adaptive behaviors in CCAs like TCP Prague to improve fairness with other CCAs. Furthermore, it can enhance real-time communication applications by optimizing FEC strategies based on predicted AQM types.

While this study shows the potential of ML-based AQM classification, further research is required to generalize these findings across a broader range of AQMs and more complex network scenarios. Future work could also focus on integrating these ML models into real-time systems, addressing challenges such as dynamic bottleneck shifting and multi-hop scenarios. Additionally, after predicting the AQM type, the sender's action could be applied according to the selected use case, and the resulting overall network performance improvements could be analyzed. This extension would provide a more comprehensive understanding of the practical impact of AQM prediction on real-world network performance.

In summary, this work demonstrates the potential of ML for addressing the problem of bottleneck type detection and presents example use cases illustrating how this approach can enhance network performance and address specific challenges.

References

[1] L. L. Peterson and B. S. Davie, *Computer Networks, Fifth Edition: A Systems Approach*, 5th ed. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2011.

[2] G. Carlucci, L. De Cicco, S. Holmer, and S. Mascolo, "Congestion control for web real-time communication," *IEEE/ACM Transactions on Networking*, vol. 25, no. 5, pp. 2629–2642, 2017.

[3] S. Sunassee, A. Mungur, S. Armoogum, and S. Pudaruth, "A comprehensive review on congestion control techniques in networking," in *2021 5th International Conference on Computing Methodologies and Communication (ICCMC)*, 2021, pp. 305–312.

[4] A. Ogola, "A survey paper on tcp congestion control algorithms," 04 2024.

[5] J. Chung and M. Claypool, "Analysis of active queue management," in *Second IEEE International Symposium on Network Computing and Applications, 2003. NCA 2003.*, 2003, pp. 359–366.

[6] F. B. Sarpkaya, "Machine learning for congestion control. is it a boon or a bane?" https://drive.google.com/file/d/1e5ltLkso8hguaxCzAekMDMkKcXYvAjvJ.

[7] B. Briscoe, K. D. Schepper, M. Bagnulo, and G. White, "Low Latency, Low Loss, and Scalable Throughput (L4S) Internet Service: Architecture," RFC 9330, Jan. 2023. [Online]. Available: https://www.rfc-editor.org/info/rfc9330

[8] S. Ha, I. Rhee, and L. Xu, "Cubic: a new tcp-friendly high-speed tcp variant," *SIGOPS Oper. Syst. Rev.*, vol. 42, no. 5, p. 64–74, Jul. 2008. [Online]. Available: https://doi.org/10.1145/1400097.1400105

[9] S. Floyd, D. K. K. Ramakrishnan, and D. L. Black, "The Addition of Explicit Congestion Notification (ECN) to IP," RFC 3168, Sep. 2001. [Online]. Available: https://www.rfc-editor.org/info/rfc3168

[10] F. B. Sarpkaya, A. Srivastava, F. Fund, and S. Panwar, "To switch or not to switch to tcp prague? incentives for adoption in a partial l4s deployment," in *Proceedings of the 2024 Applied Networking Research Workshop*, ser. ANRW '24. New York, NY, USA: Association for Computing Machinery, 2024, p. 45–52. [Online]. Available: https://doi.org/10.1145/3673422.3674896

[11] Y. Guo, Z. Meng, B. Wang, and M. Xu, "Inferring in-network queue management from end hosts in real-time communications," in *ICC 2024 - IEEE International Conference on Communications*, 2024, pp. 3389–3395.

[12] D. Lin and R. Morris, "Dynamics of random early detection," in *Proceedings of the ACM SIGCOMM '97 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, ser. SIGCOMM '97. New York, NY, USA: Association for Computing Machinery, 1997, p. 127–137. [Online]. Available: https://doi.org/10.1145/263105.263154

[13] K. Nichols, V. Jacobson, A. McGregor, and J. Iyengar, "Controlled Delay Active Queue Management," RFC 8289, Jan. 2018. [Online]. Available: https://www.rfc-editor.org/info/rfc8289

[14] R. Pan, P. Natarajan, F. Baker, and G. White, "Proportional Integral Controller Enhanced (PIE): A Lightweight Control Scheme to Address the Bufferbloat Problem," RFC 8033, Feb. 2017. [Online]. Available: https://www.rfc-editor.org/info/rfc8033

[15] T. Høiland-Jørgensen, P. McKenney, dave.taht@gmail.com, J. Gettys, and E. Dumazet, "The Flow Queue CoDel Packet Scheduler and Active Queue Management Algorithm," RFC 8290, Jan. 2018. [Online]. Available: https://www.rfc-editor.org/info/rfc8290

[16] K. D. Schepper, B. Briscoe, and G. White, "Dual-Queue Coupled Active Queue Management (AQM) for Low Latency, Low Loss, and Scalable Throughput (L4S)," RFC 9332, Jan. 2023. [Online]. Available: https://www.rfc-editor.org/info/rfc9332

[17] B. Briscoe and A. S. Ahmed, "Tcp prague fall-back on detection of a classic ecn aqm," 2021. [Online]. Available: https://arxiv.org/abs/1911.00710

[18] M. K. Bideh, A. Petlund, and I. Ahmed, "Demonstration of tada: A tool for automatic detection of aqm," in *2016 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, 2016, pp. 997–998.

[19] V. Arun and H. Balakrishnan, "Copa: Practical Delay-Based congestion control for the internet," in *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18)*. Renton, WA: USENIX Association, Apr. 2018, pp. 329–342. [Online]. Available: https://www.usenix.org/conference/nsdi18/presentation/arun

[20] N. Cardwell, Y. Cheng, C. S. Gunn, S. H. Yeganeh, and V. Jacobson, "Bbr: congestion-based congestion control," *Commun. ACM*, vol. 60, no. 2, p. 58–66, Jan. 2017. [Online]. Available: https://doi.org/10.1145/3009824

[21] C. Baykal, W. Schwarting, and A. Wallar, "Detection of aqm on paths using machine learning methods," 2017. [Online]. Available: https://arxiv.org/abs/1707.02386

[22] I. Baldin, A. Nikolich, J. Griffioen, I. I. S. Monga, K.-C. Wang, T. Lehman, and P. Ruth, "FABRIC: A National-Scale Programmable Experimental Network Infrastructure," *IEEE Internet Computing*, vol. 23, no. 6, pp. 38–47, 2019.

[23] IBM, "What is random forest?" 2024, accessed: 2024-12-07. [Online]. Available: https://www.ibm.com/topics/random-forest

[24] D. Gunay, "Random forest," 2024, accessed: 2024-12-07. [Online]. Available: https://medium.com/@denizgunay/random-forest-af5bde5d7e1e

[25] IBM, "What is a neural network?" 2024, accessed: 2024-12-07. [Online]. Available: https://www.ibm.com/topics/neural-networks

[26] A. LeNail, "Nn-svg: Publication-ready neural network architecture schematics," *Journal of Open Source Software*, vol. 4, no. 33, p. 747, 2019. [Online]. Available: https://doi.org/10.21105/joss.00747

[27] X. Cui, X. Xu, Q. ShenLi, and X. Liang, "Analysis of tcp flow characteristics," in *Proceedings of the 2019 International Conference on Computer, Network, Communication and Information Systems (CNCI 2019)*. Atlantis Press, 2019/05, pp. 112–118. [Online]. Available: https://doi.org/10.2991/cnci-19.2019.15

[28] F. B. Sarpkaya, F. Fund, and S. Panwar, "To adopt or not to adopt l4s-compatible congestion control? understanding performance in a partial l4s deployment," 2024. [Online]. Available: https://arxiv.org/abs/2411.10952