Real-Time Congestion Control Algorithm Identification with P4 Programmable Switches

Andrés García-López*, Elie F. Kfoury†,
Jose Gomez‡, Jorge Crichigno†, Jaime Galán-Jiménez*
*Dept. of Computer Systems and Telematics Engineering, University of Extremadura, Spain.
†Integrated Information Technology Department, University of South Carolina, USA.
‡Katz School of Business, Fort Lewis College, USA.
andresgl@unex.es, ekfoury@email.sc.edu, jagomez@fortlewis.edu, jcrichigno@cec.sc.edu, jaime@unex.es

***Abstract*—The classification of Congestion Control Algorithms (CCAs) is vital in current networks, where increasing traffic demands and dynamic conditions challenge their stability and performance. CCAs play a fundamental role in managing congestion, balancing throughput, minimizing latency, and reducing packet loss to ensure reliable data transmission across diverse scenarios. Despite their importance, accurately identifying the CCA in use remains a challenging task; critical for optimizing resource allocation and enhancing Quality of Service (QoS). This paper presents a framework that leverages P4-programmable switches for real-time extraction of key traffic metrics, including queuing delay, interarrival time, queue depth, RTT, and sending rate. These metrics are analyzed using a Random Forest classifier to predict the CCA in use with high accuracy. Extensive experiments in a controlled network environment, featuring a bottleneck link and flows utilizing CCAs such as Cubic, Reno, BBR, and Vegas, validate the effectiveness of our approach.**

***Index Terms*—TCP, Congestion control algorithm, Programmable data plane, P4, Random Forest**

I. INTRODUCTION

Transmission Control Protocol (TCP) is the cornerstone of modern networks, ensuring reliable data transmission across diverse and evolving network environments. It serves as the backbone for countless internet applications, from web browsing and video streaming to cloud services and real-time communication. A critical aspect of how TCP works is its congestion control algorithms (CCAs), which play a pivotal role in managing network congestion and optimizing overall performance. CCAs are designed to dynamically adapt to changing network conditions, employing unique methodologies to balance key performance metrics such as throughput, latency, and packet loss. By doing so, they ensure that networks remain efficient and stable, enabling smooth operation even under heavy traffic loads [1] [2].

Over the years, several CCAs have been developed, each tailored to address specific challenges. Traditional algorithms like Reno focus on simplicity and fairness, while more modern ones like Cubic emphasize scalability in high-bandwidth environments. Meanwhile, algorithms such as BBR take a fundamentally different approach by targeting bottleneck bandwidth and minimizing queuing delays. These varied strategies reflect the complexity of network behavior and highlight the critical role CCAs play in achieving optimal functionality across diverse scenarios [3] [4].

Despite the significant advancements in CCA design, identifying the specific algorithm in use for a given TCP flow remains a complex challenge. Accurate identification of the CCA can provide valuable insight for network operators, allowing them to optimize resource allocation, enhance Quality of Service (QoS), and implement tailored network policies. For instance, understanding the CCA behavior could facilitate better traffic engineering strategies, improve congestion management in Software-Defined Networking (SDN) environments, and aid in troubleshooting performance anomalies.

The difficulty of this task is further compounded by the dynamic and heterogeneous nature of modern networks. Network conditions, such as bandwidth availability, latency, and packet loss, can vary significantly over time and across different paths. Additionally, the coexistence of multiple CCAs within the same network adds to the complexity, as each algorithm responds differently to the same set of conditions. These challenges underscore the need for advanced techniques to identify and classify CCAs based on observable traffic patterns.

Analyzing traffic metrics offers a promising method for detecting CCAs by capturing crucial aspects of network behavior and providing insights into TCP flow dynamics. Metrics such as queuing delay, interarrival time, queue depth, and sending rate are key in understanding interactions among senders, networks, and receivers. A notable advancement in this study is using a P4 programmable switch to gather these metrics directly from the data plane [5]. The precision and detailed monitoring enabled by P4 allow for real-time assessment of traffic conditions with minimal overhead, thereby improving the accuracy and efficiency in identifying CCA.

The integration of these metrics with machine learning techniques opens new frontiers for addressing the longstanding challenge of CCA identification. Machine learning algorithms, particularly classification models, excel at uncovering patterns and correlations in complex datasets, making them well-suited for distinguishing between CCAs based on traffic behavior. By training models on labeled data representing various algorithms, it becomes possible to develop predictive systems capable of accurately identifying the CCA in real-time.

This approach has significant potential for improving modern network management. By accurately identifying CCAs in an automated way, network operators can make better decisions about traffic prioritization, congestion control, and policy enforcement. Additionally, the ability to dynamically classify CCAs can enhance the adaptability of SDN controllers and optimize resource allocation.

To address this challenge, this study explores an approach to predicting the CCA of a TCP flow based on traffic metrics. By leveraging machine learning techniques, specifically a Random Forest classifier [6], we analyze a comprehensive set of connection metrics. These metrics are captured using a P4 switch, which facilitates real-time data collection and provides unparalleled insight into the behavior of TCP flows under varying network conditions [7], [8].

The proposed system is validated through extensive experimentation in a controlled network environment, featuring a P4 switch and multiple TCP flows utilizing different CCAs. Our results demonstrate the effectiveness of this approach, achieving high classification accuracy while providing actionable insights into the behavior of TCP flows.

This paper is organized as follows: Section II reviews related work on CCA analysis and classification. Section III details the proposed system architecture and methodology. Section IV presents experimental results and discusses the findings. Finally, Section V concludes the paper and outlines directions for future research.

II. RELATED WORK

Identifying CCAs is a crucial challenge for optimizing network performance and ensuring fairness between data flows. Several approaches have been proposed, ranging from passive methods and machine learning-based solutions to systems that take advantage of the flexibility of programmable data planes.

A significant contribution is the work by Ware et al. [9]. In their paper, the authors propose CCAnalyzer, a nearly-passive system for identifying CCAs. By leveraging passive traffic collection and a limited set of features extracted from packet traces, CCAnalyzer achieves efficient and accurate classification of CCAs. Their approach demonstrates the potential of traffic analysis in identifying CCAs with minimal overhead, making it suitable for deployment in real-world networks.

Sander et al. introduce DeePCCI [10], a framework based on deep learning designed for the passive identification of CCAs. Passive identification refers to the process of analyzing traffic patterns without actively injecting or modifying network traffic, instead relying solely on observations of existing flows. By utilizing neural networks, the framework captures the subtle behavioral signatures left by different CCAs within observed traffic. The study demonstrates that this approach achieves high accuracy and scalability, showcasing the potential of deep learning for non-intrusive network analysis.

Sawada et al. [11] propose a method to estimate CCAs using deep recurrent neural networks. Their research focuses on real-world applications, particularly web servers on the internet, and evaluates the impact of accurate CCA estimation on improving server-side performance. The work underscores the value of leveraging temporal patterns in traffic for precise algorithm identification.

Carmel and Keslassy [12] contribute by introducing Dragonfly, a system designed for in-flight CCA identification. Their approach uses packet-level analysis to identify CCAs dynamically during active network flows, demonstrating its utility in scenarios where quick and reliable classification is essential. Dragonfly addresses the need for low-latency and high-accuracy identification in real-time networking environments.

Kfoury et al. [13] present P4CCI, an online system for identifying CCAs, utilizing the flexibility of the P4 programming language. Their approach focuses on separating TCP traffic based on the underlying CCA to enable optimized network performance. P4CCI emphasizes the power of P4 in implementing efficient, real-time traffic classification systems, showcasing its applicability in dynamic and heterogeneous environments.

Turkovic and Kuipers [14] propose P4air, a framework implemented using the P4 programming language to enhance fairness among competing CCAs. The system dynamically adapts its behavior to improve the distribution of resources between flows using different CCAs, preventing bias in heterogeneous environments. P4air highlights the role of programmable data planes in achieving adaptive and fair network behavior.

These works collectively emphasize the importance of identifying and managing CCAs to optimize network performance and ensure fairness. From passive methods such as CCAnalyzer to machine learning approaches such as DeePCCI and the work of Sawada et al., as well as real-time systems such as Dragonfly and programmable frameworks such as P4CCI and P4air, each study addresses unique challenges in CCA identification and optimization.

However, existing methods often have limitations: passive systems may lack real-time capabilities, deep learning solutions can be computationally intensive, and fairness-oriented systems like P4air do not explicitly focus on CCA identification. The proposed system integrates the real-time metric collection capabilities of P4 with the efficiency and interpretability of a Random Forest classifier. By leveraging these technologies, the system provides a scalable solution for identifying CCAs, while also capturing essential metrics like queue delay, interarrival time, and sending rate. Additionally, it lays the groundwork for extending fairness-based mechanisms, offering potential improvements in multi-CCA environments. This dual focus on identification and actionable insights positions the system as a practical tool for modern network management.

III. SYSTEM MODEL

The proposed system is designed to classify CCAs by leveraging the advanced capabilities of P4-programmable data planes and the robustness of Random Forest machine learning. This architecture integrates real-time metric extraction with data-driven classification, providing a comprehensive framework for analyzing network behavior under various traffic conditions. The system architecture is shown in Fig. 1, and it can be divided into four main steps: metric calculation, metric collection and storage, dataset preparation and classification.

A. Metric Assessment in the P4 Switch

At the core of the proposed system is the P4 switch, which plays a pivotal role in metric computation. Unlike traditional approaches that rely on external monitoring tools, the P4 switch processes packets directly in the data plane, enabling high-speed and low-latency operations. As packets cross the switch, key traffic metrics are computed, including queue delay, interarrival time, queue depth, sending rate, and data sent.

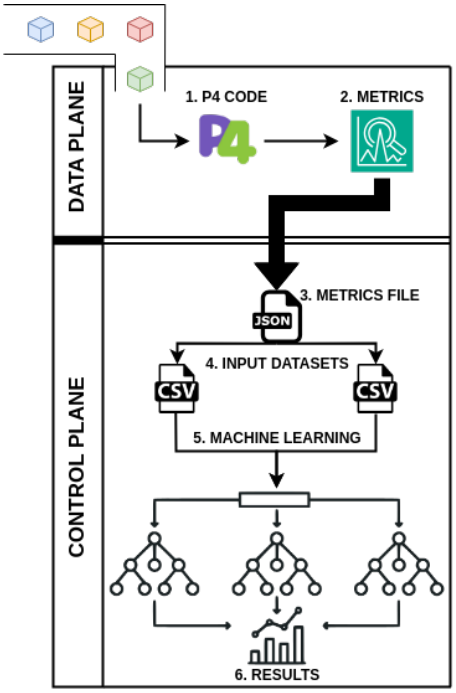

Fig. 1: Proposed system architecture.

The calculation of these metrics is performed in real time, leveraging the programmable capabilities of the switch. Queue delay is determined as the difference between the egress and ingress timestamps of a packet, calculated directly in the egress pipeline. This value provides insight into the time a packet spends traversing the internal buffers in the switch. Queue depth is extracted from the standard metadata *enq_qdepth* field, which represents the number of packets queued at the moment of packet enqueue.

The interarrival time is computed in the ingress pipeline using a hash-based register. Each flow is uniquely identified by hashing its source and destination IP addresses along with the transport-layer ports. The difference between the current timestamp of the packet and the last timestamp stored in the register gives the interarrival time, representing the spacing between consecutive packets of the same flow.

To calculate the sending rate, the ingress pipeline uses registers to track the total bytes transmitted for each flow and the time elapsed since the last packet was seen. By dividing the total bytes by the time difference, the switch computes the sending rate in bits per second. Similarly, the data sent metric is updated by accumulating the total size of packets (in bytes) for each flow.

The computed values are embedded into custom fields within the packet headers, ensuring that the metrics travel alongside the packets as they traverse the network. This inline computation and embedding approach eliminates the need for additional probes or out-of-band mechanisms, reducing overhead and enhancing scalability.

B. Metrics Collection and Storage

Once the packets reach the destination host, the metric values stored in the headers are extracted and processed. A dedicated control plane application running on the destination host parses the received packets to retrieve these metrics. Each test scenario generates a distinct set of data, which is stored in individual JSON files.

This structured approach to data collection ensures that each dataset is isolated and easily accessible for further processing. Additionally, using JSON as the storage format facilitates compatibility with a wide range of tools and programming languages, making the system adaptable to different environments and research needs.

C. Dataset Curation

After collecting the metrics, the individual JSON files are consolidated into a single CSV file. This consolidation process serves two purposes, to unify the data from multiple test scenarios into a single data set for analysis and to prepare the data for machine learning by organizing it into a tabular format.

The consolidated dataset is then divided into two subsets. The first subset is used to train the Random Forest classifier, where the decision trees are built based on the input features. The second subset is reserved for testing and validating the model to assess its accuracy and ability to generalize to unseen data.

D. Random Forest Classification

The final step involves using the Random Forest algorithm to classify the CCA employed during each test scenario. Random Forest, a widely used ensemble learning method, combines the outputs of multiple decision trees to improve classification accuracy and robustness.

To optimize the performance of the classifier, hyperparameter tuning was performed to determine the best combination of parameters. The final model was configured with the following hyperparameters:

- `random_state=42`: Ensures reproducibility of results by controlling the randomization process during the creation of decision trees.
- `max_depth=30`: Limits the maximum depth of each tree, preventing overfitting by constraining the growth of trees and focusing on more general patterns in the data.
- `max_features=None`: Allows each tree to consider all available features when searching for the best split, maximizing the potential to identify the most informative splits.
- `min_samples_leaf=3`: Specifies the minimum number of samples required to form a leaf node, ensuring that individual leaf nodes are not too small, which reduces the risk of overfitting.
- `min_samples_split=11`: Defines the minimum number of samples required to split an internal node, controlling the growth of the tree and ensuring splits are meaningful.
- `n_estimators=145`: Sets the number of decision trees in the forest. This value was chosen to strike a balance between computational cost and model accuracy, with enough trees to reduce variance without unnecessary redundancy.

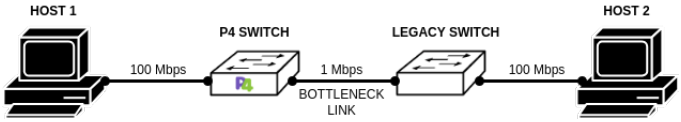

Fig. 2: Topology used to test the proposed system.

These hyperparameters were carefully selected through an extensive search process, using validation techniques to evaluate multiple configurations and identify the combination that yielded the best performance.

The classifier utilizes metrics calculated in the P4 switch as input features to capture traffic flow dynamics under various CCAs. By assessing these metrics, it achieves precise CCA predictions. Performance metrics, including precision, recall, F1-score, and accuracy, are computed to measure classification efficacy. These findings offer insights into the proficiency of the system in identifying CCAs across different network conditions, showcasing the optimized effectiveness of the Random Forest model.

IV. RESULTS

To evaluate the performance of the proposed system, a series of experiments were conducted using the network topology shown in Fig. 2. The experiments were performed in a virtual environment using Mininet. The topology consists of two hosts, h1 and h2; and two switches: a P4 programmable switch on the left side of the topology and a legacy switch on the right side of the topology. The hosts are connected using 100 Mbps links, except for the link connecting the two switches, generating a bottleneck with a bandwidth of 1 Mbps and a delay of 1 second.

The P4 switch was configured to allow seamless communication between hosts by installing the necessary forwarding rules. In addition, it was programmed to calculate and embed traffic metrics, such as queuing delay, queue depth, sending rate, data sent, and interarrival time, in the packet headers in real time. These metrics provide valuable insights into the behavior of the traffic flows, enabling accurate classification of CCAs.

The system classifies four widely used congestion control algorithms: Reno, Cubic, BBR, and Vegas. These algorithms were chosen for their distinct congestion control strategies and relevance in various networking scenarios. Reno and Cubic represent traditional loss-based approaches, with Reno serving as a foundational algorithm and Cubic enhancing performance in high-bandwidth networks. BBR takes a proactive approach by estimating available bandwidth and round-trip time, making it suitable for low-latency applications. Vegas, in contrast, is a delay-based algorithm that adjusts sending rates based on precise RTT measurements. The inclusion of these algorithms ensures the system is evaluated across diverse traffic patterns and congestion control methods, demonstrating its classification effectiveness.

To further align the configuration of the P4 switch with the bottleneck link constraints, the parameters *set_queue_rate* and *set_queue_depth* were computed using precise mathematical models. The *set_queue_rate* parameter was derived using the formula:

$$q_{\text{rate}} = \frac{\text{Maximum rate [bits/s]}}{\text{Packet size [bits/packet]}}$$

where the maximum rate corresponds to the bottleneck bandwidth of 1 Mbps (1,000,000 bits/s) and the packet size represents the Maximum Transmission Unit (MTU) of 1500 bytes, equivalent to 12,000 bits per packet. Substituting these values, the calculated *q_rate* is:

$$q_{\text{rate}} = \frac{1,000,000}{12,000} \approx 83 \,\text{packets/s}.$$

Similarly, the *set_queue_depth* parameter was configured to match the Bandwidth-Delay Product (BDP) of the bottleneck link, computed as:

$$\text{BDP} = BW \times \text{delay}$$

where BW is the bottleneck bandwidth of 1 Mbps (1,000,000 bits/s) and the delay is set to 1 second. Using these values, the BDP is calculated as:

$$\text{BDP} = 1,000,000 \times 1 = 1,000,000 \,\text{bits}.$$

Converting the BDP into packets by dividing by the packet size (12,000 bits per packet), the resulting *set_queue_depth* is:

$$q_{\text{depth}} = \frac{1,000,000}{12,000} \approx 83 \,\text{packets}.$$

This configuration ensures that the bottleneck behavior is accurately represented, enabling precise calculation and embedding of metrics such as queuing delay and interarrival time while maintaining realistic congestion scenarios.

We evaluated the performance of the classification system through multiple experiments, first assessing accuracy at the packet level and then at the flow level by aggregating packets into progressively larger time windows. Data used for the experiments was synthetically generated with iperf3 to emulate a real-world network behavior. Traffic patterns were carefully designed to reflect a variety of conditions, ensuring a comprehensive evaluation of the ability of the system to classify congestion control algorithms under diverse network dynamics.

All transmissions lasted for 180 seconds, during which bandwidths were varied from 0.1 Mbps to 2 Mbps, and the number of concurrent flows ranged from 1 to 45. These parameters were chosen to simulate realistic network conditions, including moments of limited resource availability and high contention. By generating synthetic data with such variability, the experiments provided a robust testbed for evaluating the accuracy of the classification system and adaptability across a wide range of scenarios.

The confusion matrix in Fig. 3 provides a comprehensive view of the packet-level classification results. The matrix shows the actual labels on the y-axis and the predicted labels on the x-axis for the four CCAs under study: BBR, CUBIC, Reno, and Vegas. The system achieved a high overall accuracy of 97.01%, as shown in the figure. These results highlight the robustness of the system, particularly in distinguishing between Reno and Vegas, which are often challenging due to their similar responses to congestion.

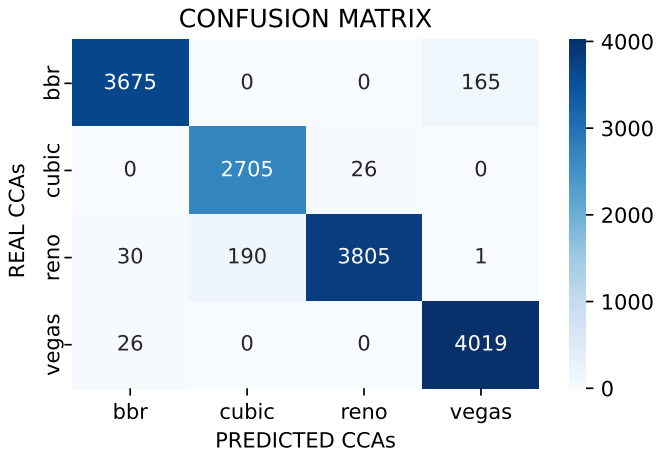

Fig. 3: Confusion matrix to evaluate the prediction by packets.

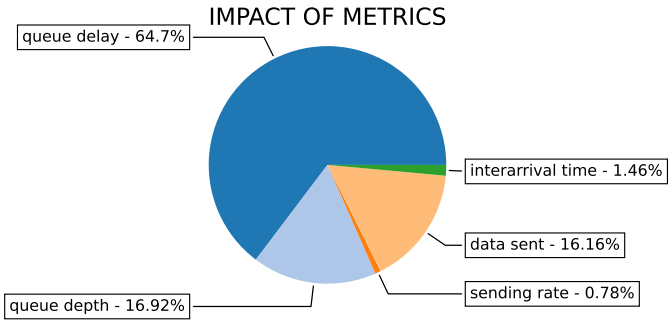

Fig. 4: Impact of metrics on the decision making.

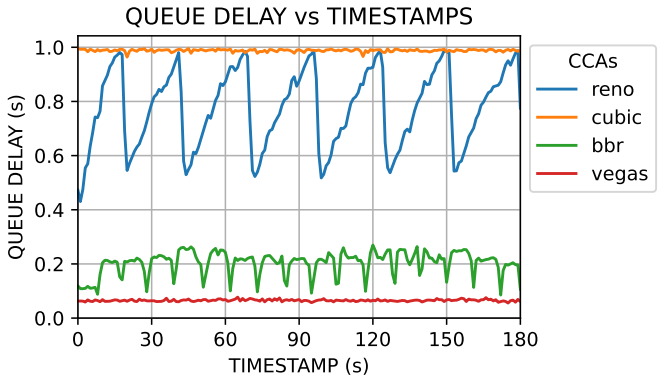

Fig. 5: Behavior of queue delay for the different CCAs.

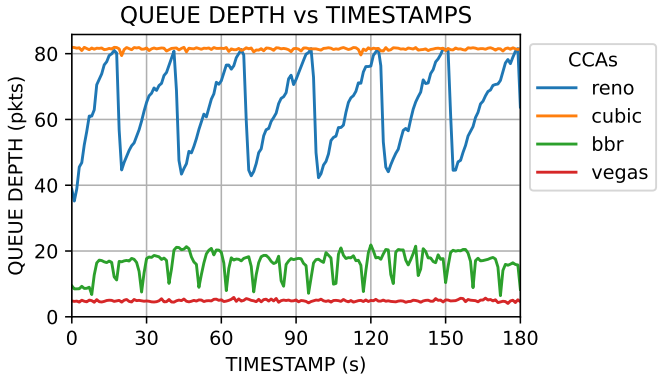

Fig. 6: Behavior of queue depth for the different CCAs.

To further analyze the contribution of individual traffic metrics to the classification process, the importance of each metric was assessed and visualized in Fig. 4. The pie chart reveals that queue delay and queue depth are the most significant factors. The remaining metrics also contribute but to a lesser extent. These findings underscore the critical role of queue-based metrics in identifying the behavior of different CCAs under congestion.

In addition, we analyze the performance of each CCA on the metrics that most impact flow classification under different congestion conditions, namely queue delay, queue depth, and data sent. The figures present the metric values for each CCA based on a separate execution per algorithm, but they are displayed in the same graph to ensure a coherent and consistent comparison. As can be seen in Fig. 5, the queue delay metric reveals clear differences between the algorithms. BBR maintains systematically low but variable queue delays due to its proactive approach to congestion control, which avoids excessive queuing by accurately estimating the available bandwidth and maintaining a constant data flow. In contrast, Reno and CUBIC show higher delays due to their reactive strategies, where forwarding rate adjustments are only triggered after packet losses are detected. Vegas shows a stable but slightly increasing queuing delay.

Regarding the queue depth metric in Fig. 6, the same behavior as for queue delay can be observed. Furthermore, it can be confirmed that Reno reduces its congestion window by half.

Regarding the data sent in Fig. 7, BBR ensures steady and efficient data transmission, capitalizing on its bandwidth estimation mechanism, while Cubic and Reno exhibit periodic fluctuations linked to congestion loss recovery cycles. Vegas transmits data at a relatively constant pace, prioritizing stability over aggressive throughput.

These observations highlight the distinctive strategies employed by each CCA and validate the use of these metrics in capturing the nuances of traffic dynamics and congestion management.

As packet-level classification, although informative, is not always practical, a more realistic approach was adopted by aggregating predictions over time windows. Predictions were first made for a 1 second window, then for several 5 and 10 second windows, progressively increasing the window size. Within these windows, flows were classified based on the cumulative predictions of packets, ensuring more stable and reliable classification while minimizing individual misclassification effects. Fig. 8 shows each congestion control algorithm separately but within the same figure to allow for a coherent and consistent comparison. The system achieved 100% accuracy for all algorithms, confirming its reliability in scenarios where decisions are based on aggregated intervals and reinforcing its applicability in network management.

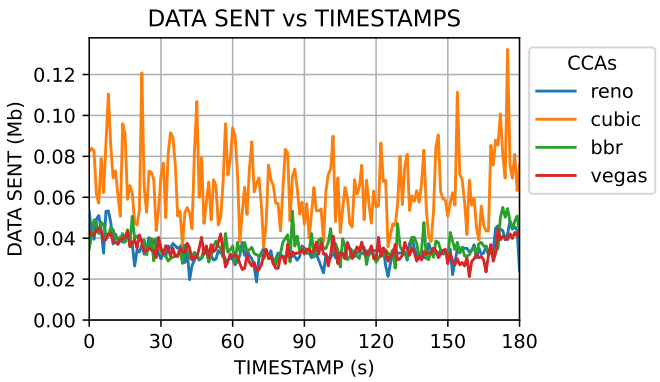

Fig. 7: Behavior of data sent for the different CCAs.

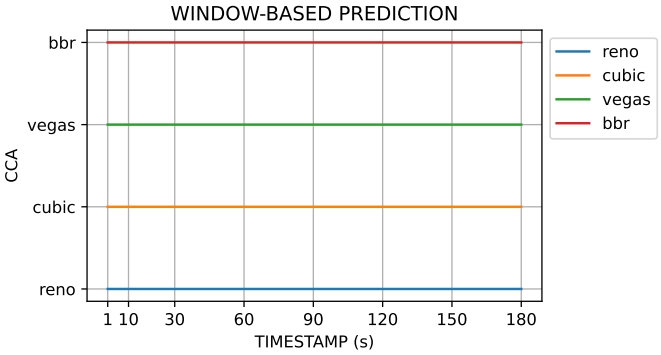

Fig. 8: Classification of CCAs by windows.

V. CONCLUSION AND FUTURE WORKS

This paper presented a system for classifying CCAs using P4-programmable switches, where traffic metrics such as queue delay, interarrival time, queue depth, sending rate, and data sent are collected directly in the data plane. This approach eliminates the need for external monitoring tools to capture network data, reducing overhead while enabling real-time metric extraction. These metrics are embedded into packet headers and subsequently processed by a Random Forest classifier running on a separate server, which analyzes the information to accurately identify CCAs such as Cubic, Reno, BBR, and Vegas. Experimental results validate the effectiveness of this system in bandwidth-constrained scenarios, demonstrating its ability to provide valuable insights into CCA behavior under diverse network conditions. This work highlights the potential of integrating programmable networking with machine learning to enhance network monitoring and performance analysis.

Future research will focus on extending the system by incorporating additional CCAs to evaluate scalability and adaptability. Exploring new traffic metrics, such as retransmission rates or bandwidth utilization, could improve classification accuracy. To further enhance precision and adaptability, future efforts will investigate model retraining strategies that adjust to evolving network conditions. A key direction is migrating the classification logic entirely to the data plane, enabling real-time CCA identification and reducing reliance on control plane processing. This shift, along with potential on-device ML inference within the P4 switch, would minimize external computation, reduce latency, and improve applicability in high-speed, large-scale networks, paving the way for more efficient and adaptive management strategies.

ACKNOWLEDGMENT

This work has been partially funded by MICIU/AEI/10.13039/50100011033 and by "European Union NextGenerationEU/PRTR" (projects TED2021-130913B-I00 and PDC2022-133465-I00), by the project PID2021-124054OB-C31 and the grant CAS21/00057 (MICIU/AEI/FEDER, UE), and by the Regional Ministry of Economy, Science and Digital Agenda of the Regional Government of Extremadura (GR21133).

REFERENCES

[1] J. Postel, "Rfc0793: Transmission control protocol," 1981.

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

[3] R. Al-Saadi, G. Armitage, J. But, and P. Branch, "A survey of delay-based and hybrid tcp congestion control algorithms," *IEEE Communications Surveys & Tutorials*, vol. 21, no. 4, pp. 3609–3638, 2019.

[4] H. Jamal and K. Sultan, "Performance analysis of tcp congestion control algorithms," *International journal of computers and communications*, vol. 2, no. 1, pp. 30–38, 2008.

[5] E. F. Kfoury, J. Crichigno, and E. Bou-Harb, "An exhaustive survey on p4 programmable data plane switches: Taxonomy, applications, challenges, and future trends," *IEEE access*, vol. 9, pp. 87 094–87 155, 2021.

[6] A. Parmar, R. Katariya, and V. Patel, "A review on random forest: An ensemble classifier," in *International Conference on Intelligent Data Communication Technologies and Internet of Things (ICICI) 2018*, J. Hemanth, X. Fernando, P. Lafata, and Z. Baig, Eds. Cham: Springer International Publishing, 2019, pp. 758–763.

[7] E. Kfoury, J. Crichigno, and E. Bou-Harb, "P4bs: Leveraging passive measurements from p4 switches to dynamically modify a router's buffer size," *IEEE Transactions on Network and Service Management*, vol. 21, no. 1, pp. 1082–1099, 2024.

[8] E. Kfoury, J. Crichigno, E. Bou-Harb, and G. Srivastava, "Dynamic router's buffer sizing using passive measurements and p4 programmable switches," in *2021 IEEE Global Communications Conference (GLOBECOM)*, 2021, pp. 01–06.

[9] R. Ware, A. A. Philip, N. Hungria, Y. Kothari, J. Sherry, and S. Seshan, "Ccanalyzer: An efficient and nearly-passive congestion control classifier," in *Proceedings of the ACM SIGCOMM 2024 Conference*, ser. ACM SIGCOMM '24. New York, NY, USA: Association for Computing Machinery, 2024, p. 181–196. [Online]. Available: https://doi.org/10.1145/3651890.3672255

[10] C. Sander, J. Rüth, O. Hohlfeld, and K. Wehrle, "Deepcci: Deep learning-based passive congestion control identification," in *Proceedings of the 2019 Workshop on Network Meets AI & ML*, ser. NetAI'19. New York, NY, USA: Association for Computing Machinery, 2019, p. 37–43. [Online]. Available: https://doi.org/10.1145/3341216.3342211

[11] T. Sawada, R. Yamamoto, S. Ohzahata, and T. Kato, "Tcp congestion control algorithm estimation by deep recurrent neural network and its application to web servers on internet," *International Journal on Advances in Networks and Services Volume 16, Number 1 & 2, 2023*, 2023.

[12] D. Carmel and I. Keslassy, "Dragonfly: In-flight cca identification," in *2023 IFIP Networking Conference (IFIP Networking)*, 2023, pp. 1–9.

[13] E. Kfoury, J. Crichigno, and E. Bou-Harb, "P4cci: P4-based online tcp congestion control algorithm identification for traffic separation," in *ICC 2023 - IEEE International Conference on Communications*, 2023, pp. 4007–4012.

[14] B. Turkovic and F. Kuipers, "P4air: Increasing fairness among competing congestion control algorithms," in *2020 IEEE 28th International Conference on Network Protocols (ICNP)*, 2020, pp. 1–12.