Time-Series Clustering for CCA Classification

Michael Bryant*
Duke University
Durham, NC
michael.c.bryant@duke.edu

Manos Giannopoulos*
Duke University
Durham, NC
manos.giannopoulos@duke.edu

Majed Al Munefi*
Duke University
Durham, NC
majed.al.munefi@duke.edu

David Zhao*
Duke University
Durham, NC
david.zhao@duke.edu

ABSTRACT

The landscape of Congestion Control Algorithms (CCAs) on the Internet is continuously evolving. Every few years, the current state-of-the-art solution is shown to be outdated, and new solutions appear, only for the cycle to be repeated. Large corporations are deploying their own custom-made CCAs, while the ever-increasingly adopted QUIC allows any website admin to implement their own CCA in the user space. In this paper, we first examine the existing solutions for CCA classification and attempt to improve upon them. Noticing the limitations of this approach, we argue that their perspective is limited, as assigning class labels to custom-made algorithms is not realistic. We argue that a conceptual shift is needed to design methods that are sufficiently informative while not being confined within the task of classification. We sketch the first steps toward that direction, proposing methodologies that can characterize traces of CCAs without the need to assign a hard label.

1 INTRODUCTION

Congestion Control Algorithm (CCA) classification is a long-standing problem in the network community. The congestion control landscape affects design decisions of other components of the Internet such as buffer sizes [1]. This implication makes CCA identification essential for maintaining efficient, fair, and reliable network performance across different conditions and infrastructures. However, the rapid development of new CCA algorithms and the design concept shift from strictly TCP-friendly algorithms like CUBIC [6] to less traditional algorithms like BBR [4] has complicated this task. Moreover, the increasing adoption of QUIC [7] - a UDP protocol that provides a user-space CCA development environment, enabling personalized custom-made CCAs - necessitates the ability of CCA identification algorithms to handle previously unseen CCAs. In this paper, we qualitatively evaluate existing solutions, noting their limitations and proposing improvements to circumvent them. The paper is structured as follows: In section 2 we perform an extensive literature review, commenting on the strengths and weaknesses of existing solutions. In section 3, we present the methodology used to gather the necessary data and engineer our solution. In section 4 we present our preliminary results. Finally, in section 5, we discuss the limitations of our work, connect them to the limitations of existing work, and identify potential future directions of CCA Classification research.

2 RELATED WORK

Gordon. Gordon [10] emulates a local bottleneck, changing the available bandwidth and artificially inducing packet losses to probe the CCA's behavior under congestion events. In that way, it creates an expressive enough CWND trace to be able to classify all known Linux kernel-supported algorithms with very high accuracy. However, this aggressive packet-dropping technique is outdated; as of 2023, more than 96% of Alexa Top 20K websites implement DDoS protection that shuts Gordon down. Moreover, with Gordon, the authors handcraft a decision tree that labels a trace based on a small set of human-picked features. This design choice means that even if Gordon was able to function in the presence of DDoS protection, it would still need periodic updates to stay up-to-date with the current CCA landscape.

Inspector Gadget. Inspector Gadget [5] utilizes a measurement technique that is very similar to Gordon. Their main difference is in the way they obtain the final decision tree; Gordon's decision tree is handcrafted by the authors, whereas Inspector Gadget *learns* the decision tree, using the CART [3] algorithm. The authors show that optimal performance is obtained when setting the depth parameter to infinity. It is well known in the Machine Learning community that infinite-depth decision trees overfit to the training data, creating leaves that correspond to very obscure scenarios unlikely to be encountered in the test data. This limitation is "hidden" in the paper, as the authors train and test the accuracy of their solution using control data created in a testbed environment.

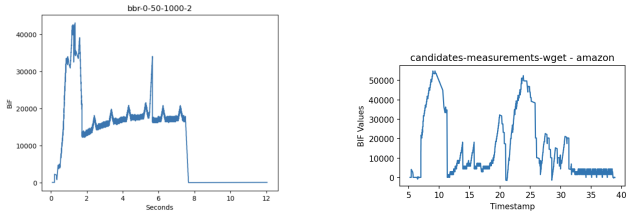
*Dept. of Computer Science

The hardness of evaluation on real data is a recurrent problem in the area of CCA classification and is further discussed in section 2.

CCAnalyzer. [11] CCAnalyzer captures queue occupancy traces to create time-series and classifies them using a distance-based approach. Specifically, it uses control flows as points of reference, and employs a 1-Nearest-Neighbor approach, assigning each test time-series the class of its nearest neighbor in the reference group only if the distance to that nearest neighbor is less than some parameter d. The distance metric used is Dynamic Time Warping [2], a standard metric for time-series classification that aligns two different time-series by matching their points in a way that minimizes the Euclidean distance. However, CCAnalyzer only manages to classify approximately 30% of the Top 10K Alexa websites, failing to label the other 70%. We conjecture that a significant factor for the failure of CCAnalyzer is the "Curse of Dimensionality", a well-known problem in the Machine Learning community when trying to use distance-based clustering on high-dimensional data. In short, the Curse of Dimensionality states that if we fix a distance threshold d from a point of reference, the probability of a test data point being within d of that point decreases exponentially with respect to the number of dimensions of the points. The dimensionality of a time-series constructed by CCAnalyzer is in the thousands, leading to a high number of test data points having no nearest-neighbor within a hypersphere of radius d. As such, it is natural that the DTW approach that CCAnalyzer employs is ineffective at classifying most of the test flows that it encounters.

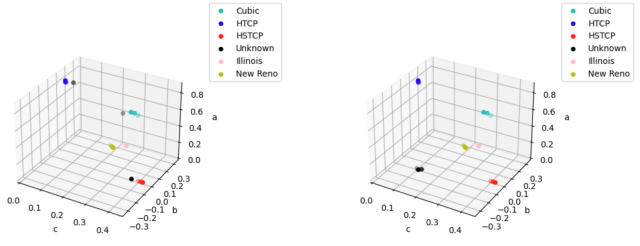
Nebby. Nebby [9] is the current state-of-the-art solution for CCA identification, tackling all of the above limitations. However, it has different drawbacks by design. Nebby uses a trail of Bytes-in-Flight (BiF) measurements to produce a signal, which it then feeds into a decision tree. The decision tree either classifies it as some version of BBR or classifies it as a loss-based CCA. If a signal is classified as loss-based, it is then converted to a high-degree polynomial. The coefficients of each polynomial-signal are matched to those of known CCAs - if no such matching exists, the signal is labeled as "Unknown". The above approach requires constructing a cluster of potential polynomial coefficients for each known CCA, which Nebby achieves by setting up servers around the globe and capturing control flows. This naturally introduces the problem of *distribution shifts*; the control flows that Nebby uses to create these clusters are significantly different to the actual flows that are produced by measuring the Alexa Top 20K websites. This is clearly depicted in Figure 1. A tangential problem that stems from this approach is the hardness of evaluation. Nebby's authors can only evaluate the accuracy of their approach by doing a train-test split on their control data; obtaining the true labels of test flows in the wild is not feasible, and can only be achieved at a small scale by asking the websites' administrators or crawling tech posts that share this information.

Moreover, Nebby's "Unknown" labeling discards valuable information. To showcase that, we adversarially simulated BiF flows that would be produced by slightly tweaked versions of known CCAs, as well as BiF flows of our own custom New Reno-like CCA that is clearly distinguishable from every other CCA supported by the Linux kernel because of our unusual choice of parameters. Nebby classifies every single one of those implementations as "Unknown", providing no additional information about them. Thus, as seen in Figure 2, a researcher cannot differentiate between Unknown measurements forming a clear cluster, being random noise, or being implementations of known algorithms that are slightly off the set of acceptable parameters constructed from the control flows. To partially circumvent this limitation, the authors of Nebby propose manually inspecting the flows that are labeled as Unknown and creating handcrafted classifiers that are plugged into the decision tree (between BBR classification and polynomial fitting). However, it is clear that this solution is not optimal and leaves an open problem.



(a) (b)

Figure 1: (a) Control BBR flow - (b) BBR flow in the wild from amazon.com



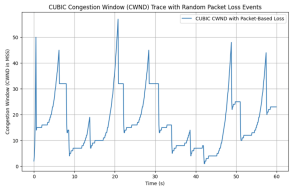
(a) (b)

Figure 2: (a) Nebby clusters with simulations of slightly tweaked known algorithms - (b) Nebby clusters with simulations of our own custom CCA

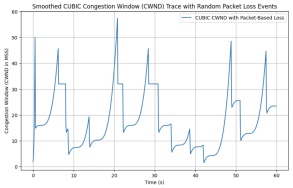
3 METHODOLOGY

3.1 Data Collection

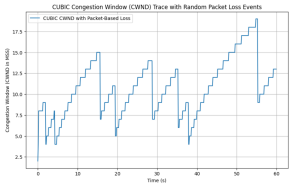
3.1.1 Control Data. Training Machine Learning models requires a clean dataset. To generate such a dataset, we used the open-source network simulator ns-3, running simulations for several CCAs that are supported in the Linux kernel. The simulations included a fixed bandwidth and RTT, and stochastic packet loss with a fixed percentage that was set according to the bandwidth and RTT to produce realistic CWND waveforms. The full set of parameters for each simulation were time length, bandwidth, RTT, and packet loss percentage. Several sets of these parameters were chosen in order to create a diverse dataset that can represent potential test time data that is collected in the wild. For the traces, we opted to get both a raw CWND trace with integer values, as well as a smoothed version to facilitate the polyfit methodology that is described in section 3.3.3. Two examples including a TCP Cubic simulation and a TCP New Reno simulation, and their respective smoothed versions can be seen in Figure 3. However, running Nebby's source code proved difficult due to unclear documentation, as well as challenges with dependencies and running it locally.



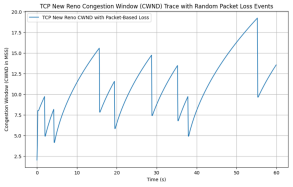
(a) CUBIC simulation example.



(b) Smoothened CUBIC simulation example.



(c) New Reno simulation example.



(d) Smoothened New Reno simulation example.

Figure 3: Examples of CWND traces simulated in ns-3.

3.1.2 Available data from Nebby [9]. Initially, we planned to collect our own flows for Alexa Top 20K websites using the scripts from the repository of Nebby[1]. However, poor documentation in the repository made this challenging. While the documentation described how to run the code, it gave little information about running the necessary background processes for the program. Moreover, the build files provided in the repository are missing several dependencies, and the helper scripts seem to have been modified. Despite our best efforts, we decided to move on from running the Nebby code locally. This decision was also motivated by the presence of a limited dataset committed to the repository which we used in the initial stage of our analysis. The repository contains data dumps of 74 flows that we ended up using as test data.

3.1.3 Available data from Gordon [10]. The repository of Gordon[2] contains CWND flows from every Alexa Top 20K website. While CWND flows are not enough to differentiate between different BBR versions, they work very well as testing flows for our loss-based CCA classifier.

3.2 Feature Engineering

Before performing any classification tasks for our traces, we need to perform some data cleaning and feature extraction. Raw measurements are noisy from cross-traffic on the internet, so we use a rolling average with a suitable window size given the size of our trace to smooth out the data. Smoothed data removes noise and can highlight important features in the data that we need for classification tasks like polynomial fitting, gradient calculations, and time-series-based clustering. A potential area of future research is to use a more sophisticated technique for smoothing, such as a Fourier transform or a Kalman filter, to react faster to changes.

Since most CCAs have an indistinguishable behavior in their slow start phases, we ignore these phases and attempt to extract segments from the traces that are associated with the congestion avoidance phase. We can do this by taking our smoothened data and looking for unusually high negative gradients that indicate a sharp decrease in the cwnd size and then keeping the data from then up until the next time we see unusually high positive gradients again. In the figure below, we can see that BBR, TCP Cubic, and TCP Reno all have similar slow start phases but have distinct congestion avoidance phases that can be used to help distinguish and classify them. It is important to note as well that loss-based CCAs like Cubic and BBR can exhibit periodicity because they back off periodically after buffer overflow. So if a raw trace is big enough, then it is possible to have multiple segments for a given trace.

[1]https://github.com/NUS-SNL/Nebby/tree/main

[2]https://github.com/NUS-SNL/Gordon/tree/master

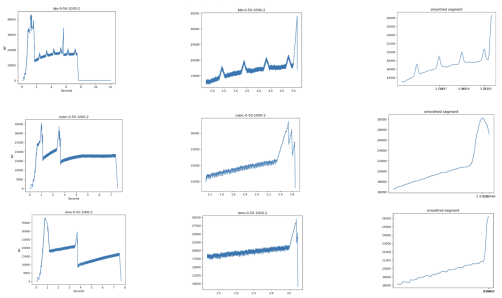


Figure 4: Examples of smoothened, segmented traces for BBR, Cubic, and Reno

Using our simulated data from NS3, we don't have as much noise in our traces, and the packet loss is very clear with immediate drops in the cwnd size. NS3 gives an option to smooth the data as well from integer cwnd sizes to float sizes using what we suspect to be a rolling average algorithm as well. We can see the feature extraction process for NS3 data below.

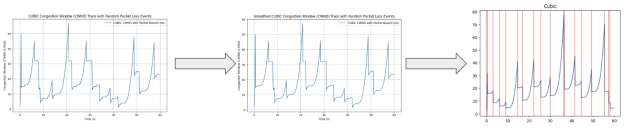


Figure 5: Feature engineering process for NS3 CUBIC trace

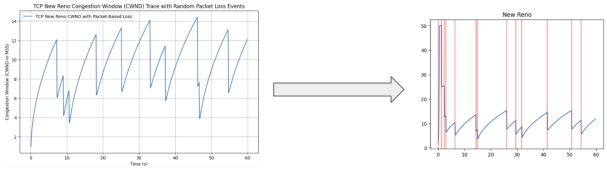


Figure 6: Feature engineering process for NS3 New Reno trace

We measured 500 unique traces of TCP CUBIC and TCP New Reno. We randomly shuffle the samples and use 50 traces of CUBIC and 50 traces of Reno for our test set. Out of our 900 flows of CUBIC and Reno, we have 9350 CUBIC segments and 5370 Reno segments. The average segment length of CUBIC is 20.77 data points and the average New Reno segment length is 11.9 data points.

3.3 Classification

3.3.1 Loss-Based vs Non-Loss-Based Classifier. BBR (Bottleneck Bandwidth and Roundtrip propagation time)is a non-loss-based congestion control algorithm designed by Google in 2016. Traditional loss-based CCAs like TCP CUBIC and TCP Reno use packet loss as an indicator for congestion, whereas non-loss based CCAs use bandwidth probing and RTT_{min} estimation to infer congestion. BBR's development was motivated by the fact that nowadays, when bottleneck buffers are large, loss-based congestion control keeps them full causing bufferbloat. Furthermore, when bottleneck buffers are small, loss-based congestion control algorithms misinterpret packet drop as a congestion signal, leading to low throughput.

BBR uses latency instead of lost packets as the primary factor in determining the sending rate. It considers how fast the network is delivering data. Given a network connection, it uses recent measurements of the network's delivery rate and round trip time to build a model that includes 1) the maximum recent bandwidth available to that connection and 2) its minimum recent round trip delay.

Because BBR doesn't have the characteristic cut-off of cwnd size after packet loss, it would be unwise to include it in our clustering where we don't have any characteristic polynomial to fit. Rather, BBR has a few unique characteristics that we can look for in the time-series to determine whether or not a CCA is BBR or not. If our BBR classifier can confidently say that an algorithm is BBR, then we don't include it in our clustering. The characteristics that we look for are as follows.

- BBR has a probing behavior, where for every 8 RTTS, it increases the sending rate by 25%, we can look for periodic spikes in the time series of the extracted segments. If we see more than 3 spikes with respect to the derivative, then this requirement is satisfied
- BBR also backs off every 10 seconds to estimate the minimum RTT of the path. So in our segments, if we see a drop in sending, ie flat line behavior at the end of our probing then we can conclude the CCA is BBR.

The behavior of BBR can be seen clearly in the control data shown below.

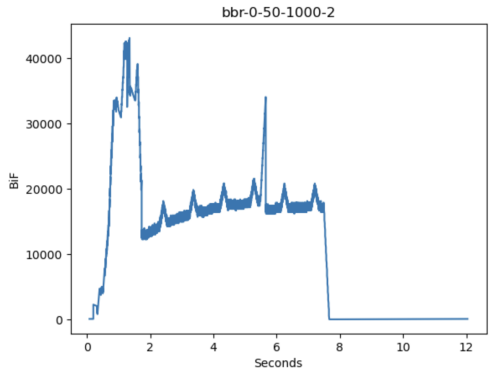


Figure 7: BBR control data

We segment the traces to only look at the period of congestion avoidance and then smooth the data using a running average to eliminate noise and make the calculations of the gradient clearer as shown below.

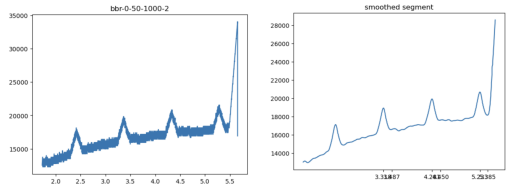


Figure 8: BBR control data segmented and smoothened

To classify BBR, we look for these 2 characteristic features in our respective traces and if they are satisfied, then we can classify it as BBR. To do so, we use a sliding window technique where the window size is determined by the RTT, and we look at the gradients inside the current window. If we see a 25% increase in the gradient in our window, then we can increment a counter. If we have at least 3 increases that meet our threshold, then the first characteristic is satisfied. The second requirement can be met by looking at the end of the trace and seeing if there is a dropoff in sending where we have 0 Bytes-in-Flight sent after the characteristic bandwidth probing.

3.3.2 Time Series K-Means with Dynamic Time Warping. We use a time series implementation of K-Means in order to determine CCA trace centroids. Like traditional K-Means, this approach generates representative traces and compares new data points to classify using Euclidean distance. We use a generalized approach that applies to time series data.

Using such an approach requires a degree of pre-processing. We use the Dynamic Time Warping metric which, as described above, allows comparison for waveforms that occur at different speeds. This is important as we are searching for a fundamental shape and do not want to differentiate between different occurrences.

Additionally, we standardize our segmented data on a few heuristics. First, we resample the data to have the same length. We also mean adjust and shift so all data starts at a congestion window of zero.

After data standardization, we fit our K-Means based on the known classes of the segments.

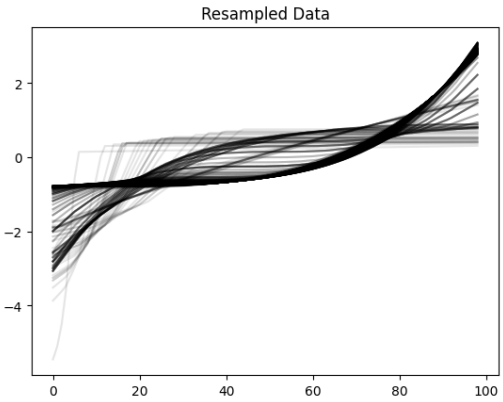


Figure 9: Resampled data of two classes superimposed (Cubic, TCP New Reno). The different classes can be visually differentiated. Plotted over sample index.

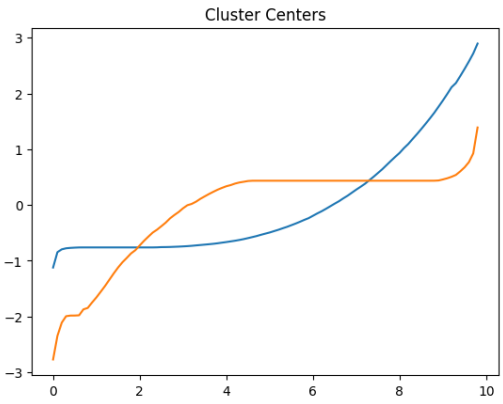


Figure 10: Computed cluster centers for the above two-CCA example. Plotted over duration in seconds.

Determination of a CCA as unknown is interpreted as a learned hyperparameter of the Euclidean distance to nearest centroid.

3.3.3 Polynomial Fitting for Loss Based CCAs. For loss-based CCAs, we attempt to fit a third-degree polynomial using Python's numpy.polyfit function. We offset our segments

to all start from a 0 cwnd size, so we can get consistent coefficients when looking at different traces. We find that fitting a third-degree polynomial is suitable for characterizing our CCAs as it can capture the shape of more complex segments like those generated from TCP Cubic. Using a threshold on if the magnitude of the coefficient for the cubic polynomial is large enough, we can classify a trace to be TCP CUBIC or TCP New Reno.

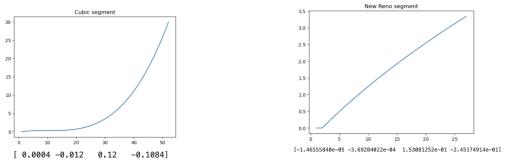


Figure 11: Examples of CUBIC and New Reno Classification using np.polyfit

The coefficients seen from our polynomial fitting confirm the Additive Increase Multiplicative Decrease (AIMD) behavior of New Reno where the window size is increased by one segment every RTT. For TCP CUBIC, the coefficients confirm the more aggressive window-increasing behavior in the congestion avoidance phase.

4 RESULTS

4.1 Loss Based vs Non-Loss Based Classification

Out of 77 websites from our Nebby dataset, we classify 37 out of 77 websites to be a Loss-based CCA like BBRv1. This includes websites like Hulu, Amazon, Prime Video, YouTube, and TikTok. It is interesting to note that most websites classified as BBRv1 are video streaming platforms and that Youtube (a Google-owned platform) is classified as BBRv1. As we don't have the ability to confirm what CCAs or variants of well-known CCAs that companies are using, we cannot have an accurate performance-based metric to further evaluate (which is another reason why our simulated NS3 data is valuable). We cross-reference our results from popular websites like YouTube, Amazon, TikTok, etc. with the paper "Keeping an Eye on Congestion Control in the Wild with Nebby" along with other sources and find that our results agree with those presented by the other authors.

4.2 Time Series K-Means

The result of our testing of the total CCA identification yielded 70.1% accuracy. However, it is worth nothing that this is purely on a segment basis.

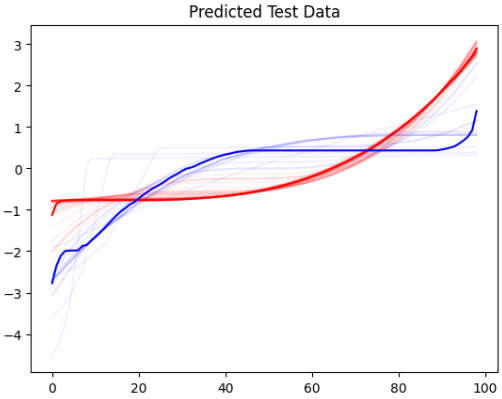


Figure 12: Two-CCA example with cluster centers in dark blue and red, along with test data classifications in light red and blue.

The main point of inaccuracy comes from the high number of false-positive CUBIC labels. This likely stems from the scaling algorithm we employed, and its resulting uneven scaling of the cubic nature before and after the saturation point.

4.3 Polynomial Fitting for Loss-Based CCAs

Out of 7960 segments, we correctly classify 6966 traces, giving us an accuracy of 87.5%. Out of the 994 incorrect segments: 198 were predicted to be TCP New Reno but were actually TCP CUBIC, and 796 segments were predicted to be TCP CUBIC but were actually TCP New Reno.

4.4 Segment vs Flow

We also propose leveraging the relationship between segments and flows as part of our algorithm. Specifically, since segments are subsections of flows, we can aggregate segments from flows to make a stronger prediction. While we do recognize that dynamically changing CCAs could be deployed, we determined this to be a negligible subset.

Our approach concerning flow identification from segment classifications is as follows. If the classifications for all segments are unknown or a single classification, and there is at least one non-unknown classification, the flow is classified as the non-unknown classification. However, a flow would be classified as unknown if all segments are unknown or if multiple classifications are present beyond a trained hyperparameter.

Using this approach theoretically increases flow identification accuracy exponentially with respect to the number of segments recorded. Our experimentation yielded a resultant accuracy of greater than 90%.

5 DISCUSSION

5.1 Limitations

Data Collection. We tried to include a variety of ns-3 simulations to gather a diverse training dataset. However, the scope of our work was limited and we believe significant improvements can be made by constructing the training dataset more rigorously. Moreover, as we were unable to run our own scripts to capture traces from the Alexa Top 20K websites, we relied on test measurements existing in several repositories, which were taken under different conditions. Thus, our test data distribution is a mixture of distributions that is unlikely to be representative of a present real-world scenario. However, the above caveats can only impact our results quantitatively; conceptually the ideas we present in this paper should not be affected by the quality of the gathered test data.

Hardness of Evaluation. As we have argued above (see section 2), the evaluation of CCA classification algorithms using real-world data as opposed to testbed data is a recurrent problem in the area that is present in every work in the literature. The reason behind this issue is simple: it is infeasible to obtain a "hard label" of the actual CCA used in every single website in the wild. Therefore, authors result to running testbed simulations to evaluate their solutions on artificially created test data. No matter how diverse these simulations are, they will always capture only a small subset of real-world scenarios, meaning that the evaluation of any solution will be quite inaccurate. We further discuss this issue in section 5.2, raising conceptual questions about what constitutes an appropriate evaluation of a CCA identification solution.

Curse of Dimensionality. Our proposed DTW method has the same limitation as CCAnalyzer, failing to work in cases where the BiF trace is sufficiently high-dimensional. We were unable to provide an appropriate solution to this problem in the timeframe of this project, but we sketched a proposed approach in section 5.2, delegating it as future work.

5.2 Future Work

Handling the Curse of Dimensionality. In order for distance-based clustering methods to work well, we need to incorporate some type of dimensionality reduction into our solution. We propose to include an encoder early in the pipeline, that maps each time-series to a feature-rich low-dimensional latent space. This encoder can be trained using standard contrastive learning, an often-used technique in the machine learning community to handle unsupervised learning problems. For further reading toward that direction, we cite [8, 12].

Conceptual issues with CCA Classification. We believe our work points out a larger underlying problem in the area of CCA Classification; the objective is not well-defined. With the ever-accelerating expansion of custom-made CCAs, trying to assign hard labels to flows captured in the wild might not be realistic in the near future, as their true labels will actually be unknown. We conjecture that a more appropriate objective would be to transform the problem to CCA *Characterization*, using clustering and soft-assigning each test flow to one or more clusters. Each cluster could then represent some characteristics of the CCA algorithms it encompasses such as steepness of backoff, probing behavior, polynomial degree, etc. This concept opens the door to a variety of problem formulations and design pathways, allowing researchers to explore novel methods that better capture the complicated behaviors of modern CCAs. We hope that our work not only highlights this shift in perspective but also serves as an inspiration for advancing research in this promising direction.

A APPENDIX

A.1 Code Repository

All of our source code is publicly available on GitHub. The repository and associated documentation can be found at https://github.com/dazh5587/CS514?tab=readme-ov-file.

REFERENCES

[1] Guido Appenzeller, Isaac Keslassy, and Nick McKeown. 2004. Sizing router buffers. *SIGCOMM Comput. Commun. Rev.* 34, 4 (Aug. 2004), 281–292. https://doi.org/10.1145/1030194.1015499

[2] R. Bellman and R. Kalaba. 1959. On adaptive control processes. *IRE Transactions on Automatic Control* 4, 2 (1959), 1–9. https://doi.org/10.1109/TAC.1959.1104847

[3] L. Breiman, J. Friedman, R. A. Olshen, and C. J. Stone. 1984. *Classification and Regression Trees (1st ed.).* Chapman and Hall/CRC.

[4] Neal Cardwell, Yuchung Cheng, C. Stephen Gunn, Soheil Hassas Yeganeh, and Van Jacobson. 2016. BBR: Congestion-Based Congestion Control. *ACM Queue* 14, September-October (2016), 20 – 53. http://queue.acm.org/detail.cfm?id=3022184

[5] Sishuai Gong, Usama Naseer, and Theophilus A. Benson. 2020. Inspector Gadget: A Framework for Inferring TCP Congestion Control Algorithms and Protocol Configurations. In *Traffic Monitoring and Analysis.* https://api.semanticscholar.org/CorpusID:219956477

[6] Sangtae Ha, Injong Rhee, and Lisong Xu. 2008. CUBIC: a new TCP-friendly high-speed TCP variant. *SIGOPS Oper. Syst. Rev.* 42, 5 (July 2008), 64–74. https://doi.org/10.1145/1400097.1400105

[7] Adam Langley, Al Riddoch, Alyssa Wilk, Antonio Vicente, Charles 'Buck' Krasic, Cherie Shi, Dan Zhang, Fan Yang, Feodor Kouranov, Ian Swett, Janardhan Iyengar, Jeff Bailey, Jeremy Christopher Dorfman, Jim Roskind, Joanna Kulik, Patrik Göran Westin, Raman Tenneti, Robbie Shade, Ryan Hamilton, Victor Vasiliev, and Wan-Teh Chang. 2017. The QUIC Transport Protocol: Design and Internet-Scale Deployment.

[8] Seunghan Lee, Taeyoung Park, and Kibok Lee. 2024. Soft Contrastive Learning for Time Series. (2024). arXiv:cs.LG/2312.16424 https://arxiv.org/abs/2312.16424

[9] Ayush Mishra, Lakshay Rastogi, Raj Joshi, and Ben Leong. 2024. Keeping an Eye on Congestion Control in the Wild with Nebby. In *Proceedings of the ACM SIGCOMM 2024 Conference (ACM SIGCOMM '24)*. Association for Computing Machinery, New York, NY, USA, 136–150. https://doi.org/10.1145/3651890.3672223

[10] Ayush Mishra, Xiangpeng Sun, Atishya Jain, Sameer Pande, Raj Joshi, and Ben Leong. 2019. The Great Internet TCP Congestion Control Census. *Proc. ACM Meas. Anal. Comput. Syst.* 3, 3, Article 45 (Dec. 2019), 24 pages. https://doi.org/10.1145/3366693

[11] Ranysha Ware, Adithya Abraham Philip, Nicholas Hungria, Yash Kothari, Justine Sherry, and Srinivasan Seshan. 2024. CCAnalyzer: An Efficient and Nearly-Passive Congestion Control Classifier. In *Proceedings of the ACM SIGCOMM 2024 Conference (ACM SIGCOMM '24)*. Association for Computing Machinery, New York, NY, USA, 181–196. https://doi.org/10.1145/3651890.3672255

[12] Xiaochen Zheng, Xingyu Chen, Manuel Schürch, Amina Mollaysa, Ahmed Allam, and Michael Krauthammer. 2024. Simple Contrastive Representation Learning for Time Series Forecasting. (2024). arXiv:cs.LG/2303.18205 https://arxiv.org/abs/2303.18205