Published: 25 November 2025

RESEARCH-ARTICLE

VCCAnalyzer: Identifying Congestion Control Algorithms used by Video Streaming Services

DARSHIL D KANERIA, Carnegie Mellon University, Pittsburgh, PA, United States

RANYSHA WARE, Carnegie Mellon University, Pittsburgh, PA, United States

SRINIVASAN SESHAN, Carnegie Mellon University, Pittsburgh, PA, United States

Proceedings of the ACM on Networking, Volume 3, Issue CoNEXT4 (November 2025)
https://doi.org/10.1145/3768979
EISSN: 2834-5509

VCCAnalyzer: Identifying Congestion Control Algorithms used by Video Streaming Services

DARSHIL D. KANERIA, Carnegie Mellon University, USA
RANYSHA WARE, Carnegie Mellon University, USA
SRINIVASAN SESHAN, Carnegie Mellon University, USA

CCAnalyzer is a recent proposal for a congestion control algorithm (CCA) classifier to measure the deployment of congestion control algorithms in the Internet today by creating a controlled bottleneck and monitoring bottleneck queue occupancy between third-party websites and a controlled receiver. However, this work was designed for web traffic, whereas most internet traffic today is video streaming. Video streaming applications pose additional challenges in CCA classification due to the interactions between adaptive bitrate (ABR) algorithms and CCAs. In this work, we present VCCAnalyzer, which addresses these challenges by carefully selecting bottleneck bandwidth rates to minimize ABR interference and applying interpolation and smoothing techniques to create classifiable queue occupancy traces. VCCAnalyzer achieves 100% accuracy, higher accuracy than other approaches in video classification. To demonstrate the efficacy of VCCAnalyzer, we conduct a measurement study examining the CCA deployment across major streaming platforms, including Disney+, Hulu, Twitch, and other commonly visited websites. Our findings reveal some variations in congestion control strategies between these services. Our results demonstrate that VCCAnalyzer can effectively classify CCAs in video environments.

CCS Concepts: • **Networks** → **Network measurement**.

Additional Key Words and Phrases: congestion control, video streaming, network analysis

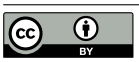
ACM Reference Format:
Darshil D. Kaneria, Ranysha Ware, and Srinivasan Seshan. 2025. VCCAnalyzer: Identifying Congestion Control Algorithms used by Video Streaming Services. *Proc. ACM Netw.* 3, CoNEXT4, Article 32 (December 2025), 11 pages. https://doi.org/10.1145/3768979

1 Introduction

The Internet relies on the stable interaction of deployed congestion control algorithms (CCAs) to ensure fairness, efficiency, and stability. This, combined with the rapid development of novel CCAs over the past few years, has created significant interest in understanding what CCAs are deployed on the Internet today. Recent work on congestion control classifiers, for identifying the CCA of a given website [3–5, 7] has largely focused on techniques designed for identifying the CCA used by web traffic with certain assumptions that one can find a large enough find that can saturate the link. However, the Internet is dominated by video streaming, accounting for 82% of global IP traffic [2], and existing techniques work poorly in classifying such traffic.

In particular, video streaming traffic has two key features that confound existing classifiers:

Authors' Contact Information: Darshil D. Kaneria, darshilkaneria@gmail.com, Carnegie Mellon University, Pittsburgh, Pennsylvania, USA; Ranysha Ware, rwjanee@gmail.com, Carnegie Mellon University, Pittsburgh, Pennsylvania, USA; Srinivasan Seshan, srini@cs.cmu.edu, Carnegie Mellon University, Pittsburgh, Pennsylvania, USA.



ACM 2834-5509/2025/12-ART32
https://doi.org/10.1145/3768979

- **Adaptive Bitrate (ABR) Control.** All popular video streaming applications use an ABR algorithm that determines the video encoding bit rate of the requested content. Typically, the ABR algorithm dynamically adjusts the video quality based on the playback buffer levels and observed network throughput. As a result, unlike bulk data transfers, the offered transmission load of a video streaming application is determined by a combination of the CCA and ABR algorithms.
- **Chunked Downloads.** Video content is not streamed in one giant file, but rather is divided into different segments of the video, called chunks. The timing and requested quality of each chunk request is determined by the ABR algorithm. Each chunk is downloaded independently, creating frequent pauses (off-periods) in the transmission behavior of the application.

These features challenge existing classification tools because the patterns in transmission behavior that existing techniques use to identify different CCAs are either modified or obscured by the combination of ABR and chunking. For example, passive techniques like CCAnalyzer, which achieved success by analyzing the distinctive queue occupancy patterns generated by CCAs under controlled bottleneck conditions, rely on observing the CCA's relatively uninterrupted behavior. ABR and chunking effects–pausing downloads when buffers are full or quickly changing requested bitrates–obscure the underlying CCA signature, which renders the queue occupancy trace difficult, if not impossible, to classify reliably. The core problem is that the observed patterns are no longer solely governed by the sender's CCA responding to intermediate network signals, but are modulated by the receiver's ABR logic and chunked transfer behavior.

These challenges lead us to a fundamental question: *How do we accurately identify congestion control algorithms in video streaming when ABR and chunking behavior constantly masks their signature patterns?* Solving this would reveal insights into performance issues and fairness concerns affecting billions of users daily.

To do this, we introduce VCCAnalyzer, which extends CCAnalyzer to classify video streams by minimizing the impact of ABR and chunked downloads. It accomplishes this with two main innovations. First, we identify what network settings to use for classification to reduce gaps between chunk requests by examining how the bottleneck link rate impacts which bitrates are chosen by the different ABR algorithms in §3.1. We find that choosing bottleneck link rates that closely match the bitrates chosen by the ABR algorithm removes large off-periods–times when the video application sends no traffic, even if the network could handle more. The size of the chunks is enough to keep the bottleneck link saturated and for there to be classifiable queue activity. However, video files being transmitted in chunks rather than one file, still introduces artifacts in the traces with off-periods between chunk requests. Thus, the second innovation of VCCAnalyzer is to apply linear interpolation and smoothing to the queue occupancy traces after collection to remove these small off-periods which we describe in §3.2. Then we are able to use the same 1-Nearest-Neighbor classifier used by CCAnalyzer to classify video streams which we describe in §3.3.

In §5.1 we evaluate VCCAnalyzer and find that we can achieve 100% accuracy classifying video streams, outperforming prior work. Notably, we are able to classify across ABR algorithms. If we use one ABR algorithm for testing and a different ABR algorithm for training, we are still able to achieve that accuracy. In addition, we can distinguish BBRv1 and BBRv3 traces. We then conduct a measurement study of some popular video streaming services and websites with video content in §5.2 and measure services using BBR, Cubic and an unknown algorithm. Lastly, we conclude our work in §6.

2 Prior Work and Challenges of Video Traffic

Researchers have tackled the problem of identifying CCAs used by remote servers for years and have developed various tools for this. Generally, these tools involve connecting to a server, generating some traffic, and analyzing the resulting network behavior. However, their effectiveness is limited when dealing with video traffic, which now constitutes the majority of Internet traffic.

Early classification tools like **Gordon**[5] and **Inspector Gadget**[3] work by deliberately interfering with the connection–introducing packet loss, adding delay to acknowledgements, or changing available bandwidth–to force the server's CCA to react in specific ways. By observing these reactions, they try to deduce the CCA's identity. However, active probing has drawbacks: it can be intrusive, consume considerable bandwidth, and might even cause servers to rate-limit or block the tool. Furthermore, estimating the congestion window (internal state in TCP) can be tricky, and these active methods are not well-suited to the dynamic interactions present in video streaming.

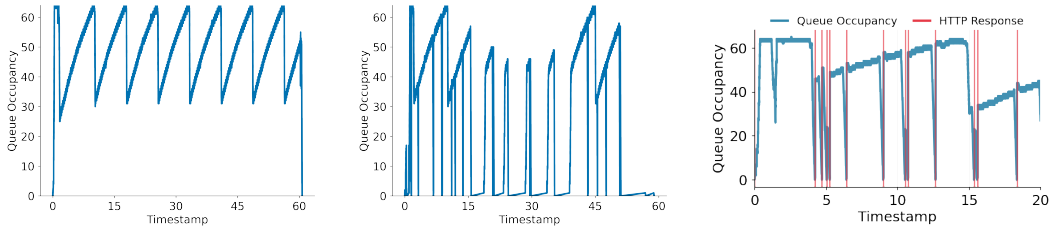
More recent, passive methods aimed to avoid these limitations. **Nebby**[4], for example, examines fine-grained packet-level details like inter-packet timing and packet sizes, searching for patterns characteristic of specific CCA behaviors (e.g., pacing intervals). While effective, Nebby sometimes requires specific knowledge about the CCA (like BBR) to interpret these patterns accurately, a limitation VCCAnalyzer does not share. As we will show in §5.1, Nebby has an accuracy of only 60% in video streams.

CCAnalyzer[7], the direct predecessor to this work, also uses a passive approach. It works by inserting a controlled bottleneck into the network path between the client and server. The key insight in CCAnalyzer is that different CCAs tend to fill and drain queues differently–compare the classic sawtooth of Reno (Figure 1a) versus the steadier, lower queue preferred by BBR. CCAnalyzer treats the queue occupancy trace as a time series and uses Dynamic Time Warping (DTW) to measure the similarity between a test trace and a library of known CCA traces. It uses a 1-Nearest Neighbor (1NN) approach, assigning the label of the most similar known trace, combined with voting and a distance threshold to identify unknowns.

While CCAnalyzer was accurate for standard web traffic, its reliance on observing continuous queue occupancy changes makes it vulnerable to the ABR interference inherent in video streaming. The off-periods and sudden rate changes common in video mask or distort the CCA's queue signature (as shown in Figure 1b), making reliable identification difficult for the original CCAnalyzer. Figure 1 illustrates the difference between an ideal bulk transfer trace and a video trace for the same CCA (TCP Reno). One might consider capturing only the initial phases of video streaming before ABR effects become significant. However, our experiments (Figure 1c) showed that these early connection phases do not provide enough data points for reliable classification. The queue occupancy patterns during initial buffering are too short and more likely representative of slow-start behavior rather than the steady-state needed for an accurate classification. As we will show in §5.1, CCAnalyzer has an accuracy of 82% on video streams. Given these challenges, there is a clear need for a classification technique specifically adapted for video streaming. VCCAnalyzer fills this need by modifying the passive classification approach to handle the complexities introduced by video traffic. We describe these changes in the following section.

3 Methodology

VCCAnalyzer leverages the same unique insight as CCAnalyzer, that different CCAs have distinct bottleneck queue occupancy patterns. Recognizing that raw queue traces from video streams are distorted by ABR and chunking, our methodology focuses on techniques that either minimize the



(a) Bulk transfer flow using TCP Reno on 5 Mbps link, 85ms RTT, and 64 packet queue

(b) Video streaming flow using TCP Reno on 10 Mbps link, 85ms RTT, and 64 packet queue

(c) HTTP responses overlayed on queue occupancy at 5 Mbps link, 85ms RTT, and 64 packet queue

Fig. 1. Queue Occupancy (packets) traces vs Time (s) captured at a controlled bottleneck by CCAnalyzer

impact of ABR and chunking or filter their effects from the resulting queue traces. VCCAnalyzer introduces the following measurement and processing pipeline:

(1) **Optimal Link Rate Identification:** This initial step determines suitable bottleneck link rates for performing measurements. The aim is to find a set of link rates where ABR-induced pauses and rate adjustments are minimized while making sure the CCA is actively engaged, making its behavior observable.
(2) **Trace Processing:** Choosing the right link rates alone does not eliminate all video streaming effects. VCCAnalyzer applies interpolation and smoothing of the collected queue occupancy data to clean the trace of remaining streaming artifacts without perturbing the underlying CCA transmission patterns.
(3) **Classification:** Using the processed traces, this final stage uses the same classification technique as CCAnalyzer. It leverages DTW distance and a 1-Nearest Neighbor (1NN) approach with voting across multiple settings and thresholding, to identify the most probable CCA.

The next subsections dive into more details behind each of these components.

3.1 Optimal Link Rate Identification

A key part of using queue occupancy for classification is choosing the right network settings to distinguish CCAs. In video streaming systems, the ABR algorithm and download scheduler are typically separate components. The ABR selects chunk quality while the scheduler controls request timing. However, algorithms like BOLA [6] create tighter coupling between these functions by making quality decisions based on buffer occupancy, which directly influences when chunks are requested. When BOLA maintains a high buffer level, downloads pause more frequently; when targeting lower buffers, downloads occur more continuously. Our goal is not to disable the ABR, but rather to guide it into a state where it requests data more continuously and, thus, its impact on transmission behavior is minimized. This allows the server-side CCA's response to network conditions–specifically, the queue build-up at our controlled bottleneck–to become the dominant observable pattern.

Our hypothesis is that we can choose a network setting such that the bitrate chosen by the ABR algorithm will closely match the bottleneck bandwidth. To find this favorable condition, we perform an empirical analysis. Using our measurement testbed (detailed in §4), we emulate a client watching a specific video stream governed by an ABR algorithm (e.g., BOLA) and a server using a particular CCA. We then sweep through a range of bottleneck link capacities configured on our intermediate switch, from 100 Kbps up to 30 Mbps. For each configured capacity, we measure the

average bitrate of the requested chunks over 60-second intervals. This process maps the ABR's effective video consumption rate to the network capacity it experiences.

We plot the results of this empirical analysis of average bitrate of requested chunks vs. the bottleneck link rate in Figure 2. We notice the following:

- **Low Link Rates:** At very low configured link rates (e.g., 100-400 Kbps in the example), the achieved video bitrate closely tracks the configured rate, often lying near the ideal y=x line. In this setting, the network link itself is the primary performance bottleneck. The ABR algorithm attempts to maximize quality within this constraint, leading to relatively continuous download requests as it tries to fill its buffer. However, from a CCA classification perspective, this region is suboptimal. Because the link rate is low, the bottleneck queue rarely builds up significantly, and the queue build up appears non-contiguous due to the small congestion window which often makes it harder to distinguish the traces. Classifying based on such traces would be unreliable.
- **High Link Rates:** As the link rate increases substantially (e.g., > 10 Mbps in the example), the achieved bitrate diverges significantly from the y=x line, plateauing or even decreasing. This occurs because the network capacity now exceeds the bitrate of the higher available video quality levels. The client's playback buffer fills very quickly. Buffer-aware ABRs (like BOLA) react to this full buffer by frequently pausing downloads to prevent overflow, leading to off-periods in the traffic. Throughput-based ABRs might oscillate wildly. In this regime, the ABR logic, driven by buffer management, becomes the dominant factor controlling traffic flow. While the CCA might be active during the bursts, the frequent long pauses introduced by the ABR corrupt the queue occupancy trace and mask the CCA's signature making classification difficult.
- **Optimal Band:** Between these two extremes lies a sweet spot. Within this band, between 1-5 Mbps, the ABR is encouraged to request data more consistently to maintain its target buffer level, but the buffer doesn't fill so rapidly as to trigger excessive pausing. At the same time, the link rate is high enough to allow significant queue build-up at the bottleneck, forcing the CCA to actively manage congestion. Measurements at these rates yield queue occupancy traces with sufficient activity for classification but with significantly fewer ABR-induced off-periods compared to higher link rates.

CCA-Dependent Behavior: An interesting pattern is uncovered when this analysis is repeated with different CCAs while keeping the client-side ABR constant. We observed that different CCAs trace different curves. Notably, BBR shows a curve that diverges from the ideal y=x line much earlier and more significantly than Cubic. This suggests that BBR's interaction with the network and the ABR's perception of throughput results in requests for lower bandwidth (and, hence, lower quality) video chunks. Despite this potentially less efficient interaction in certain scenarios, BBR variants are widely deployed by major streaming services, as discussed later in our measurement study. Plotting the curves for 15 Linux CCAs with the BOLA ABR (Figure 2) shows that while most follow the general trend, their exact divergence points and plateau levels vary. However, the 1-5 Mbps band remains a reasonable region for capturing traces for many common CCAs before ABR effects become overwhelming.

ABR Algorithm Independence: To make sure this optimal rate selection strategy is not merely an artifact of a single ABR algorithm, the entire analysis was repeated using different ABR algorithms: BOLA (buffer-based), a Throughput-based algorithm, and a Dynamic algorithm. The results (Figure 2) show that while the exact divergence points varied slightly between ABR algorithms, the overall shape of the curve and the suitability of the 1-5 Mbps band as a good measurement region remained

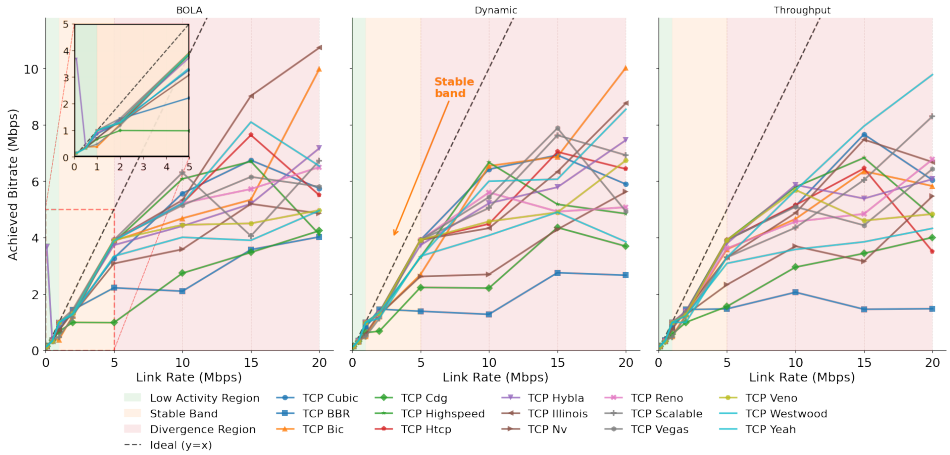


Fig. 2. Link rate vs. achieved bitrate plot for all CCAs across three ABR algorithms (BOLA, Dynamic, and Throughput).

consistent. This provides confidence that selecting a bottleneck rate in this band is a good strategy for minimizing ABR interference across different client implementations. Therefore, VCCAnalyzer uses bottleneck link rates within the 1-5 Mbps band for its trace collection. Traces captured at these rates, while significantly cleaner, may still contain minor residual artifacts from the ABR, requiring the subsequent trace processing steps.

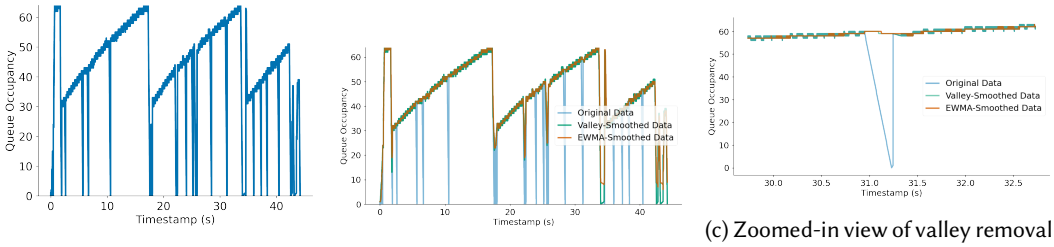
3.2 Trace Processing Techniques

Using the link bandwidth of 5 Mbps (85 ms RTT, 64 packet queue) gives us more usable traces like shown in Figure 3a, however, there are still noticeable drops in the queue occupancy traces. In Figure 1c we highlight why these drops occur. Each drop corresponds to the beginning of a new chunk being transferred. The red bar represents a response to the initial GET range request which is followed by the data for that specific video chunk.

VCCAnalyzer applies two signal processing steps sequentially to the raw queue occupancy data collected at the bottleneck to address these remaining artifacts: interpolation and smoothing.

Interpolation for Off-Period Handling: Because the video is requested in chunks, we observe intermittent gaps in queue occupancy, with each gap occurring when a video chunk finishes transmitting. This is inconsistent with typical CCA behavior (which usually involves smoother increases or decreases). VCCAnalyzer identifies these valleys by looking for periods where queue occupancy dips to zero for longer than a specified duration. To remove these artifacts, linear interpolation is applied. Conceptually, the algorithm finds the point just before the valley begins and the point just after it ends. It then replaces the data points within the valley with points lying on a straight line connecting the start and end points. This effectively bridges the gap, removing the sharp, artificial drop while preserving the overall trend of the queue occupancy before and after the ABR pause.

Choosing the duration threshold for identifying valleys is important. Too long, and we might interpolate over genuine CCA-driven queue decreases; too short, and we might miss some valleys. A threshold is identified for each captured trace by sweeping through a range of candidate threshold values (50ms - 1000ms) over which to interpolate. The value that results in the removal of the most valleys is selected and used to filter the trace. For CCAs like BBR, valley removal does remove some of the characteristic CCA behavior (e.g., its queue draining RTT probe periods). However, we find



(a) Before pre-processing (b) After pre-processing (c) Zoomed-in view of valley removal and EWMA smoothing

Fig. 3. Queue occupancy trace captured on 5 Mbps link, 85ms RTT and 64 packet queue after pre-processing stages shows a reduction in the amount of valleys making it cleaner for classification.

during evaluation that BBR (and BBRv3) traces retain enough of their characteristic behavior to have a 100% classification accuracy.

Smoothing: After interpolation removes the major valleys, the queue occupancy trace can still appear jagged or noisy on a fine timescale. This noise arises from the nature of packet capture – individual packets arriving (enqueue) and departing (dequeue) cause small fluctuations in the queue size (§4). While showing the exact queue state, this noise results in a higher DTW distance on average throughout the classification process. Smoothing out these ridges can help DTW match across points more accurately. To address this, we apply an EWMA filter. An appropriate α value is found by sweeping across a range of values (0.1 to 0.9) until we find the smallest one which smoothens the enqueue-dequeue noise without altering the characteristic appearance of the trace (Figure 3c). VCCAnalyzer uses α=0.5 for all captured traces.

3.3 Classification

VCCAnalyzer's final classification step borrows significantly from the CCAnalyzer design. VCCAnalyzer uses Dynamic Time Warping (DTW) to compare traces. DTW can match patterns even when they're stretched or shifted. This helps find similar CCA behaviors even when their timing differs slightly. For a video service under test, VCCAnalyzer collects traces for 5 different network settings within the band identified earlier: 1,2,3,4,5 Mbps link, 85ms RTT and 64 packet queue. We find this queue size and RTT work well across bandwidths. VCCAnalyzer computes the DTW distance from an individual processed trace to all reference traces collected under matching conditions. The trace is initially labeled with the CCA of its nearest neighbor—the reference trace with the minimum DTW distance. For example, Figure 5 shows all the distances between a Cubic testing sample and all the training samples in the 5 Mbps link settings. Here the closest distance is to a Cubic training sample. To determine the final CCA for the service, VCCAnalyzer performs a majority vote across the 5 traces collected for that service. This approach allows VCCAnalyzer to achieve 100% classification accuracy for known CCAs in our controlled experiments. Ties in voting are resolved by the minimal DTW distance to the closest training sample among the tied labels. Lastly, to handle classifying CCAs as unknown, VCCanalyzer sets a DTW distance threshold T for each network setting. When the DTW distance is beyond T, the trace is marked as unknown. We chose T based on the CDF for each network setting (Figure 4 shows one for a 5 Mbps link, 85ms RTT, and 64 packet queue), which shows the distribution of distances for traces of the same CCA versus a different CCA in the training set. For the network setting of 5 Mbps, 85ms RTT, and 64 packet queue, we choose a distance threshold of about 143 because 90% of traces with the same label have a DTW distance less than that.

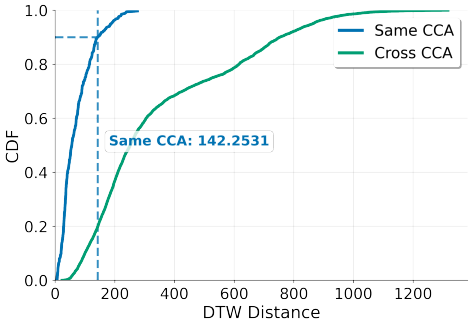


Fig. 4. Distances between traces of the same CCA and between traces with different CCAs at 5 Mbps bottleneck link.

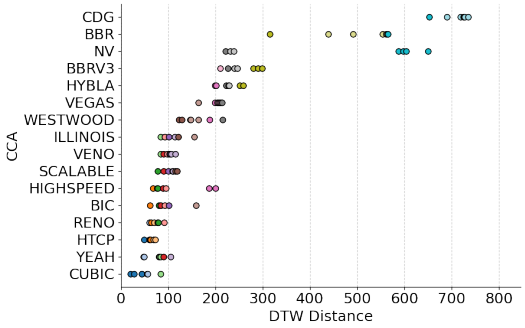


Fig. 5. A Cubic trace's distance computed to all traces in the training set.

4 Experimental Setup

We evaluate VCCAnalyzer using a controlled testbed designed to emulate video streaming over configurable network conditions and to capture precise bottleneck queue dynamics. Our setup, built on Cloudlab using Ubuntu 22.04, consists of client and server machines with an intermediate Linux-based bottleneck switch. This switch creates a bottleneck link. We use Linux Traffic Control (tc) with Hierarchical Token Bucket (HTB) for rate limiting and Network Emulator (netem) for adding delay and setting queue limits. Using this we set the bottleneck link rate (within the 1-5 Mbps band), round-trip time (RTT, 85ms in our tests), and queue capacity (64 packets) for every flow passing through the switch. The choice of 85ms RTT was influenced by analysis in previous work [7], which found that 87% of the 10,000 websites tested in their survey have baseline RTTs within 85ms. This makes sure that the added delay exceeds the baseline RTT to most websites (making our controlled bottleneck the limiting factor), while also remaining small enough to generate adequate queue build-up for classification. Traffic from the server to the client passes through this configured bottleneck. To accurately capture queue occupancy over time, we attach an eBPF program to kernel tracepoints associated with packet queueing (`qdisc_enqueue` and `qdisc_dequeue` events on the bottleneck interface) and capture every packet enqueue and dequeue event.

To generate training data, we stream a 10-minute video clip (Big Buck Bunny [1]) encoded at multiple quality levels (from 144p up to 4K) from a server which is in a different cloudlab region than the switch and client. On the client-side, we emulate various ABR algorithms including BOLA (buffer-based), a throughput-based ABR, and a dynamic ABR to make sure VCCAnalyzer is robust to different client behaviors. On the server-side, we configure known Linux CCAs (e.g., Cubic, Reno, BBR, BBRv3, and others from the standard kernel set) to generate training queue occupancy traces. All training samples are collected using the same link rate band (1–5 Mbps, using discrete bandwidths of 1, 2, 3, 4, and 5 Mbps), network settings, and eBPF-based capture method, and then undergo the same trace processing steps as traces from third-party servers.

5 Results and Analysis

We evaluate the effectiveness of this framework through several lenses. First, we check its classification accuracy on video streams generated with known server-side CCAs while emphasizing the importance of the trace processing techniques. Second, we compare with current state-of-the-art approaches CCAnalyzer and Nebby. Finally, we show VCCAnalyzer's practical utility by applying it to classify CCAs used by several major third-party video streaming services.

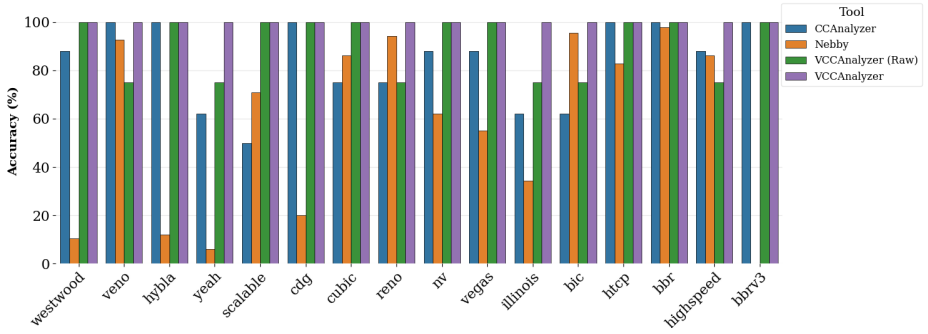


Fig. 6. We find that the overall classification accuracy for the CCAs in our training set is 100%. CCAnalyzer and Nebby underperform in several cases leading to an overall accuracy of 82.3% and 60.41% respectively.

5.1 Classification Accuracy

Higher classification accuracy in classifying CCAs used by video streaming platforms is the big goal of VCCAnalyzer. To quantify its performance, we streamed video content using multiple ABRs from a server which cycled through all 16 CCAs in the Linux kernel (which includes some of the common ones like Cubic, Reno, BBR, BBRv3, etc.).

Across all tested CCAs, VCCAnalyzer achieved an overall classification accuracy of **100%**, a breakdown of which is shown in Figure 6. This indicates that the combination of identifying an optimal link rate to capture video and applying appropriate processing techniques can significantly improve the trace to a degree where the underlying CCA can be reliably identified in a vast majority of cases.

We tested two previous approaches, CCAnalyzer and Nebby (Figure 6), by collecting traces at their recommended link rates. We modified Nebby to assign a dedicated bottleneck link to each flow. The primary reason for the lower accuracy of CCAnalyzer is the off-periods in the trace due to the involvement of the ABR algorithm. Traces captured using Nebby showed similar effects, and their shape-based classifier performed poorly on video data, which was also confirmed by the original authors in private communication. To show the effectiveness of our pre-processing approach, we also evaluated VCCAnalyzer with raw traces and achieved an accuracy of 90.6%. In comparison, CCAnalyzer achieved 82.3% accuracy and Nebby's shape-based classifier achieved 60.41% accuracy on video traffic. We also tested the classifier's ability to generalize by incorporating traces from different client-side ABR algorithms into the training set. Even when training and testing on data from mismatched ABRs, the classification accuracy remained 100% and the traces captured using different ABR algorithms visually represent the underlying CCA (Figure 7) which strengthened our confidence in the current approach's effectiveness across different client-side conditions.

5.2 Application to Third-Party Services

To demonstrate the real-world capabilities of this framework, we deployed it against popular Over-The-Top (OTT) video streaming services and popular websites that serve video content. For each video service, we initiated a video playback on our client (using the Selenium tool) and routed traffic through our controlled bottleneck to capture the queue occupancy changes over time. The results reveal interesting deployment trends.

The results strongly suggest the prevalence of BBR-family CCAs among major video platforms. Hulu, CNN, Dailymotion, and Disney+ were classified as using a BBR variant. The low DTW distance indicates a confident match to the general BBR signature. Notably, VCCAnalyzer identified

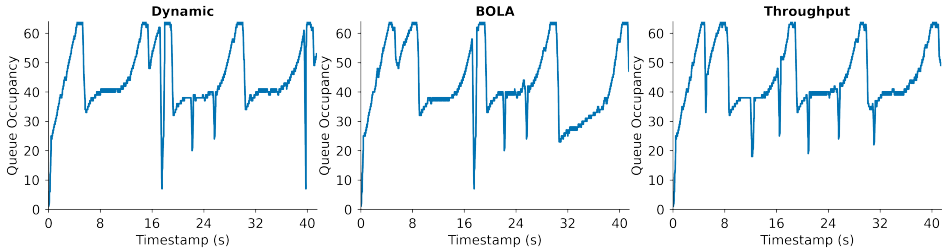
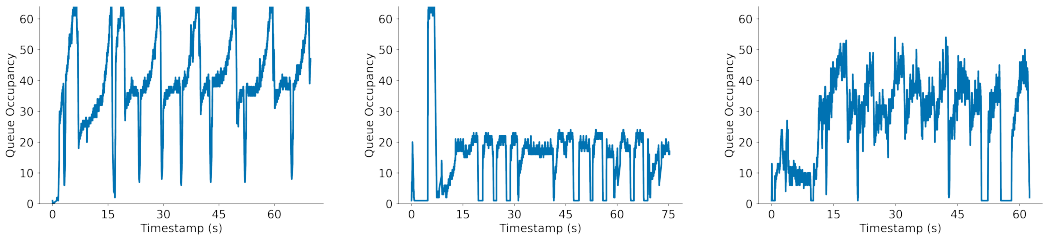


Fig. 7. Queue occupancy traces captured at 5 Mbps link, 85ms RTT, and 64 packet queue for Cubic when data is requested using three different ABR algorithms.

Table 1. Prediction results for video streaming services and websites along with lowest DTW distance

Filename	Predicted Label	Distance
Hulu	bbr	27.3461
CNN	bbr	22.75
Dailymotion	bbr	39.65625
YouTube	bbrv3	34.4424
NYT	cubic	78.5
Disney+	bbr	44.0
Tiktok	bbrv3	108.0
Twitch	bbrv3	72.3125
Vimeo	unknown	160.25
HBO Max	bbrv3	44.296875



(a) New York Times (b) Max (HBO) (c) Vimeo

Fig. 8. Queue occupancy traces for websites captured on 5 Mbps link, 85ms RTT, and 64 packet queue

YouTube, TikTok, Twitch, and HBO Max (Figure 8b) as using BBRv3. This aligns with Google's known deployment on YouTube and suggests potential adoption by CDNs or other infrastructure used by different services. The DTW distance for these, while higher than the BBR matches, were clearly closer to BBRv3 and well below the threshold for being classified as unknown. This shows VCCAnalyzer's ability to distinguish between versions of the same base algorithm when their queue signatures differ sufficiently. The classification of New York Times video service (Figure 8a) as Cubic confirms that traditional loss-based algorithms remain in use for video delivery, even if less common.

Vimeo's (Figure 8c) classification as unknown is significant. The reported DTW distance (160.25) exceeds the thresholds established during training, indicating that its trace did not closely resemble

any of the known CCAs in our reference library. This successfully shows the open-set capabilities of VCCAnalyzer and suggests Vimeo might be using a significantly modified variant of an existing CCA or a proprietary algorithm altogether.

6 Conclusion

This paper addressed the critical question: *How can we identify congestion control algorithms in video streaming environments where ABR behavior masks their signatures?* We presented VCCAnalyzer, a framework to classify CCAs used by video streaming platforms. VCCAnalyzer incorporates two key techniques to improve CCA classification for video. First, VCCAnalyzer analyzes the relationship between bottleneck link capacity and achieved video bitrate. This identifies conditions (link rates) that minimize ABR-induced off-periods while ensuring the CCA behavior remains observable. Then, VCCAnalyzer processes captured queue occupancy traces to remove remaining video-related artifacts. This is done using linear interpolation to remove short off-periods and EWMA smoothing to reduce noise. These steps clean the captured queue occupancy traces and make the underlying CCA signature more apparent. Finally, VCCAnalyzer applies a DTW-1NN classification method to these processed traces. We show that this design gives high accuracy (100%) in identifying known CCAs in controlled video streaming experiments across various ABR algorithms. We also demonstrate the practical utility of VCCAnalyzer by applying it to several major third-party video streaming platforms. Our findings revealed a landscape dominated by BBR variants, including the identification of BBRv3 on services like YouTube, TikTok, Twitch, and Max. We also confirm the continued presence of Cubic and identified one service (Vimeo) as using an unknown CCA, showing the framework's open-set classification capability. In essence, this work validates the statement that by selecting measurement conditions to stabilize ABR behavior and using trace processing techniques to filter ABR-induced artifacts, the principles of passive, queue-based CCA classification can be extended to the domain of video streaming traffic.

Ethics Statement: This work does not raise any ethical issues.

Acknowledgements

This work was supported in part by NSF grant CNS-2212390.

References

[1] Blender Foundation. 2008. Big Buck Bunny. https://peach.blender.org/about/.

[2] Cisco Systems. 2018. cisco.com. https://www.cisco.com/c/dam/m/en_us/solutions/service-provider/vni-forecast-highlights/pdf/Global_Device_Growth_Traffic_Profiles.pdf. [Accessed 22-04-2025].

[3] Sishuai Gong, Usama Naseer, and Theophilus A. Benson. 2020. Inspector Gadget: A Framework for Inferring TCP Congestion Control Algorithms and Protocol Configurations. In *2020 IFIP Networking Conference*. IEEE. doi:10.23919/IFIPNetworking49063.2020.9142132

[4] 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* (Sydney, NSW, Australia) *(ACM SIGCOMM '24)*. Association for Computing Machinery, New York, NY, USA, 136–150. doi:10.1145/3651890.3672223

[5] 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. doi:10.1145/3366693

[6] Kevin Spiteri, Rahul Urgaonkar, and Ramesh K. Sitaraman. 2020. BOLA: Near-Optimal Bitrate Adaptation for Online Videos. *IEEE/ACM Transactions on Networking* 28, 4 (2020), 1698–1711. doi:10.1109/TNET.2020.2996964

[7] 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, 181–196. doi:10.1145/3651890.3672255

Received June 2025; accepted September 2025