Inferring Netflix User Experience from Broadband Network Measurement

Sharat Chandra Madanapalli, Hassan Habibi Gharakhieli, Vijay Sivaraman
Electrical Engineering and Telecommunications, UNSW Sydney, Australia
Emails: sharat.madanapalli@student.unsw.edu.au, h.habibi@unsw.edu.au, vijay@unsw.edu.au

***Abstract*—Netflix is the largest video-streaming provider in the world today, with over 148 million subscribers and accounting for over 20% of broadband traffic in most developed countries. Internet Service Providers (ISPs) are acutely aware of the need to provide good video streaming experience to viewers, but are poorly equipped to measure and monitor per-stream quality. In this paper, we measure and analyze Netflix playback data from multiple households, develop a practical and scalable method to correlate network activity with client playback behavior, and provide a means for ISPs to infer per-stream Netflix experience from broadband traffic patterns. Our specific contributions are: (1) We develop a measurement tool for collecting network flow activity and client playback metrics, deploy it in 9 households and our lab to gather data for about 8000 Netflix video streams under various network conditions, and release the data to the public; (2) We analyze our data to highlight correlation between active flows and video playback phase, and between network chunk transfers and playback buffer health, during both regular-play and trick-play of video; (3) We develop a method for the ISP to infer Netflix user experience in terms of buffer fill-time, video bitrate and throughput, and detect playback buffer depletion and quality degradation events. ISPs can use our methods to measure, monitor, and manage Netflix user experience in real-time.**

I. Introduction

Streaming video continues to grow, accounting for about 58% of downstream traffic on the Internet according to the Sandvine 2018 report [1]. Further, Netflix is the top web application used in the Americas, and in the top-10 in every region of the world, generating 15% of global Internet traffic to serve over 148 million subscribers world-wide. With this kind of reach and scale, it is no wonder that ISPs are keen to ensure that their subscribers experience good Netflix streaming quality over their broadband networks, so they can better retain existing customers and attract new ones.

However, ISPs are operating blind on Netflix user experience. Netflix publishes a per-country monthly ranking of ISPs by prime-time Netflix speeds, but this is of limited value to ISPs since: (a) it is averaged across (a potentially large) user-base and does not give information on specific subscribers or streams; (b) it is retrospective and therefore not rectifiable by immediate action; and (c) it is at best an indicator of video resolution (bit-rate), with no insights into variation of quality during playback or video start-up delays that are central to user experience. With such limited knowledge, ISPs may attempt to install CDN servers, or as the last resort increase their network capacity with the hope for a better user experience. However these instrumentations can not only be prohibitively expensive, but also do not quantify the improved user experience.

Existing methods for inferring streaming video experience are not usable by ISPs for Netflix. These methods either require to extract statistics from the packet traces and/or HTTP logs, or visibility into encrypted traffic (that carry URLs and manifest files), neither of which are easy for an ISP to achieve for Netflix. While some prior works have studied video streaming in the mobile context, the behavior in broadband networks is different, and moreover mechanisms employed by Netflix in terms of using HTTPS, non-discretized bitrates, encrypted manifest files and urls, render earlier studies obsolete. Our aim in this work is to develop a method that an ISP can easily deploy into their existing network infrastructure to gain real-time visibility into per-stream Netflix user experience at scale.

In this paper, we collect Netflix data from multiple households, build models that deduce client playback behavior using just network flow activity, and evaluate the ability of our models to determine user experience from network measurements obtained by ISPs. Our specific contributions are as follows: (a) We build a tool that collects client playback metrics (like buffer health and bitrate) and flow-level network activity (byte counts and packet counts), deploy it in real households, and make public our dataset comprising over 750 hours of video playback with 8000 streaming sessions; (b) We analyze our dataset to understand different phases of video streaming, audio and video content transfer mechanisms, correlation between network activity (chunks transfers, number of active flows) and corresponding client behavior (buffer state, bitrate switches, trickplay), and the challenges posed by the sophisticated Netflix video client; and (c) We develop machine learning and statistical methods to infer user experience in terms of video quality, buffer fill time, and available bandwidth, and additionally deduce per-stream events like max-bitrate playback, buffer depletion, and quality degradation during both regular play and trickplay of the video.

II. Related Work

Prior works on measurement of Netflix video streaming studied bitrate adaptation models [2], [3], content distribution strategies and methods [4], bandwidth consumption and congestion control mechanisms [5], and prediction of the movie/title using fine-grained traffic patterns [6]. To our best knowledge, we are the first to analyze the behavior of Neflix video streaming to infer the user experience from network measurements for broadband ISPs who cater to home networks. Existing approaches for estimating streaming video

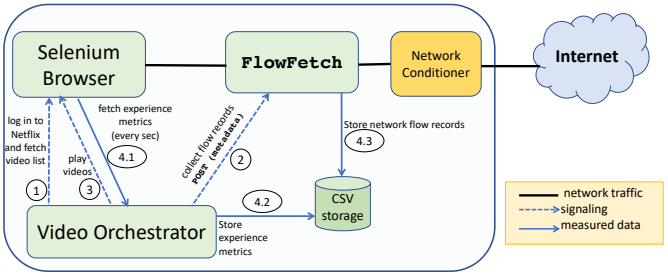


Fig. 1: Architecture of FlixMon.

experience are either statistical modeling-based or machine learning-based. Recent attempts such as eMIMIC [7] and BUFFEST [8] employ statistical models, using packet traces and HTTP requests respectively, to quantify users quality of experience (QoE). Machine Learning (ML) approaches [9]–[11] used recently, attempt at measuring QoE by predicting categorical estimates of experience metrics like *low, med, high* bitrates [10], [11] or *low, high* probabilty of bitrate switches and rebuffering [10], [11]. To collect the ground-truth, both [9] and [11] used client-side instrumentation on mobile devices and, [10] used HTTP logs and metrics exported to content provider. However, all of them used packet traces to derive fine-grained attributes such as RTT, packet losses, ACKs, retransmissions which are expensive to compute. Further, study in [9] was limited to progressive video streaming and did not consider HTTP-based Adaptive Streaming (HAS). None of the these methods can be directly used for Netflix because: first, it uses adaptive streaming over HTTPS to stream video traffic which makes HTTP logging infeasible and it encodes its URL requests and manifest files thus collection of parameters such as bitrate and content type from request logs becomes infeasible (even if MITM proxies were used). Further, existing works perform a post-facto estimation of QoE by deriving features from packet traces or logs. We account these characteristics of Netflix and propose a methodology to compute the metrics in real-time and being independent of existing logging mechanisms or tools used by the ISP.

In a parallel work [12], authors attempt to classify buffer states (*i.e.,* filling, maintaining, depleting) for Youtube videos using attributes derived from aggregate network profile. In [13], authors use low-level packet features to detect startup delay and re-buffering events in real-time for Youtube traffic but do not report the quality of video playback or its variation. We instead analyze network activity of a set of TCP flows, compute per-flow attributes, correlate them with client behavior of Netflix, and infer quality of video playback in addition to classifying the phase of video playback. Further, most of the works which measure metrics like stalls, initial delay consider mobile network scenarios. We argue that these are not enough to comment on the experience of broadband users as they fail to quantitatively compare the experiences among users with no stalls and minimal delay. We thus introduce new metrics to infer experience of Netflix users on broadband network, and offer a method to ISPs for measuring users experience of Netflix on a per-stream basis in real-time.

III. FlixMon: Our Tool for Measuring Netflix Playback Performance

To construct an accurate profile of Netflix video streams, we have developed a tool – *FlixMon* – which automatically plays videos, measures their network activity profile along with client playback metrics, and stores measured records into a pair of CSV files.

A. FlixMon Architecture:

FlixMon has three main components, each packaged into a separate docker container: a custom-built network measurement app called *FlowFetch*, a *selenium browser* instance, and a *video orchestrator* application which signals the browser to play videos. There is also an optional network conditioner which uses `tc` linux tool to shape traffic by synthetically changing network conditions in software. Containerizing applications eases deployment of the FlixMon. A shared virtual network interface among the containers ensures that packets flowing through FlowFetch originate solely from the browser, eliminating other traffic on the machine where FlixMon runs.

FlowFetch is a tool that we built in *Golang* to record flow-level activity by capturing packets from a network interface. By a flow, we mean a transport-level TCP connection or UDP stream identified by a unique 5-tuple consisting of *source IP*, *source port*, *destination IP*, *destination port* and *protocol*. For a TCP/UDP flow, the tool records (at a configurable granularity) cumulative byte and packet counts (more practical and storage-friendly than packet traces) into a CSV. FlowFetch is also able to filter flows of interest DNS queries specific to certain providers (*e.g.,* Netflix). In this work, the tool is configured to log flow records every 100ms and a DNS-based filter is employed to isolate network activity of flows from `nflxvideo.net` – the primary domain responsible for delivery of Netflix video content.

For the video orchestrator, we have used Selenium client library in Python to interact with a remote Selenium browser instance (*i.e.,* server) for loading and playing Netflix videos. At the beginning of each measurement session, a browser instance (*i.e.,* Firefox or Chrome) is spawned with no cache or cookies saved which loads the Netflix web-page and logs in to the user account by entering credentials (shown by step (1) in Fig. 1). The tool can be configured in either of two ways to generate a video list: (a) from a fixed set of Netflix videos specified in a config file, or (b) by fetching the URLs of recommended videos on the Netflix homepage that are updated regularly. Given the list, FlixMon starts playing videos sequentially. Prior to playback of each video, the player module signals the FlowFetch to start measuring network activity (shown by step (2) in Fig. 1). Then, the orchestrator signals the browser to load the video and collects the playback metrics (shown by step (3) and (4.1) respectively in Fig. 1) – Netflix player offers a series of hidden menus that allow us to view our streaming quality stats, and diagnose any potential issues. The real-time metrics (refresh every second) for audio and video media include the buffering/playing bitrates, buffer health (in seconds and bytes), and the CDN from which the

TABLE I: Summary of instances in our dataset.

	List	# streams	# titles	Stream dur.	Data resol.
Households	Rec.	1720	787	5-min	100 ms
	Fix.	919	11	5-min	100 ms
Lab	Rec.	5408	1842	2-min	500 ms
	Fix.	30	10	5-min	100 ms

stream is sourced. Additionally, position and duration of the playback, frame statistics (*e.g.,* frame rate and frame drops), and throughput are also provided. The orchestrator stores client playback metrics (every second) into a CSV file (step (4.2)) that exists in storage – a shared volume among the orchestrator and Flowfetch containers. Simultaneously, Flowfetch stores the network activity (byte and packet count measured every 100ms) into another co-located CSV (step (4.3)) when the total volume of a TCP/UDP flow crosses an export threshold (*e.g.,* 2MB) since last export.

We deployed the FlixMon both in our university lab and home networks (of 9 members of our research group). For home networks, we deployed our tool without the network conditioning module and played both fixed set and recommended set of videos. In our lab, given the high bandwidth available in our university campus network we enabled the network conditioner to synthetically impose bandwidth caps ranging from 500Kbps to 100Mbps. We would like to emphasize that the tool is needed to collect data for training models in the lab (as described in §5). Subsequently in the field, ISPs will deploy just the Flowfetch component to obtain real-time in-network flow-level measurements, and derive QoE metrics using trained models.

B. Dataset:

We collected a total of 8077 data instances for Netflix video streams. Each instance consists of a pair of CSV files (*i.e.,* one for network activity and one for client playback behavior). A summary of our dataset is shown in Table I. For households, our data includes profiles for 1720 streams of 787 unique recommend titles and 919 streams of 11 unique titles from a fixed list. Each video stream in the households dataset played for a duration of 5 minutes and corresponding network activity was recorded at every 100 ms. For lab, our data is relatively larger with 5408 streams of recommended titles along with 30 streams from the fixed list. Note that the lab data of recommended titles were collected for a duration of 2 minutes with the resolution of 500 ms – this was our first set of data collected prior to households measurements for which we increased both duration and resolution.

We release our datasets (spanning more than 750 hours of Netflix video playback) via `https://telescope-data.sdn.unsw.edu.au/netflix`. The data is organized in two main branches namely households and lab, each containing several zip files. Each zip file, corresponding to a household (or lab), has a folder denoting browser type (*i.e.,* Firefox and/or Chrome), in which are sub-folders of unique movie titles which contain instances of playback in a timestamped directory. There are two files corresponding to each instance of a video stream: (a) "`flows.csv`" (*i.e.,* network activity), and (b) "`netflixstats.csv`" (*i.e.,* client playback metrics). Each record of `flows.csv` represents the measurements (at the resolution of 100ms or 500ms) of individual TCP flows associated with a Netflix video stream comprising *timestampExport*, *timestampFlowMeasure*, *flowID*, *5-tuple*, *threshold* of flow volume at which the FlowFetch exports fine-grained flow profile measurements: cumulative *volume* (Bytes), cumulative *packetCount*, and *duration* (ms). Also, each record of `netflixstats.csv` represents the real-time measurements (*i.e.,* one row per second) of all client playback metrics provided by the Netflix player comprising *timestamp*, *movieID*, *CDNaudio*, *CDNvideo*, playback *position* (seconds), movie *duration* (seconds), *playing-bitrate-audio/video* (kbps), *buffering-bitrate-audio/video* (kbps), *buffer-size-bytes-audio/video*, *buffer-size-seconds-audio/video*, *throughput* (Kbps), etc.

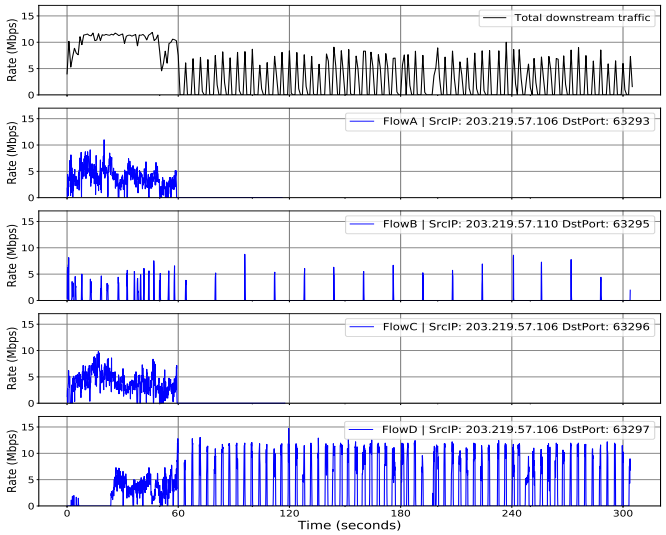


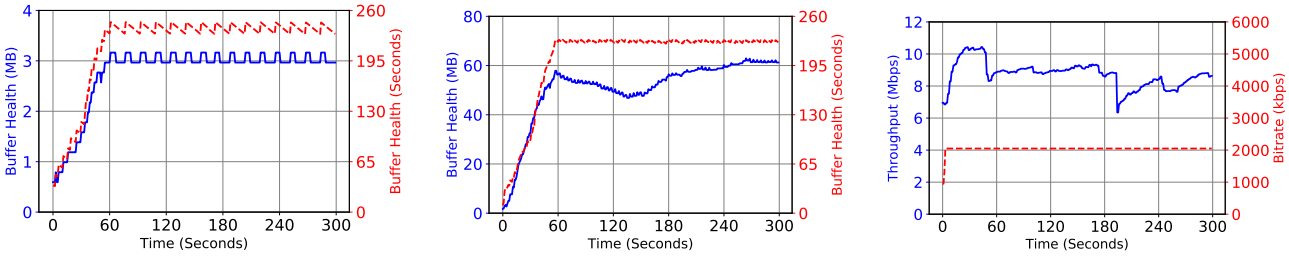
Fig. 2: Network profile of flows in a typical Netflix video stream.

IV. Netflix Streaming: Analysis and Insights

In this section, we analyze our data to highlight behavior of Netflix video streaming at client and on network .

A. Profile of a Typical Netflix Stream

Fig. 2 illustrates a time-trace of network activity measured for a representative Netflix video stream played for 5 minutes with no interruption. The top subplot shows in black lines the total downstream traffic profile for this stream, and the four subplots below in blue lines show downstream traffic profile of each TCP flow associated with this stream. We observe that the Netflix client established four parallel TCP flows to start the video, three of them come from Netflix server 203.219.57.106 and one from 203.219.57.110. All four TCP flows actively transferred content for first 60 seconds. Thereafter, two flows (A,C) became inactive (*i.e.,* idle) for a minute before being terminated by the client (*i.e.,* TCP FIN). It is seen that the remaining two active flows (B,D) changed their pattern of



(a) Audio buffer health. (b) Video buffer health . (c) Throughput and buffering-bitrate of video.

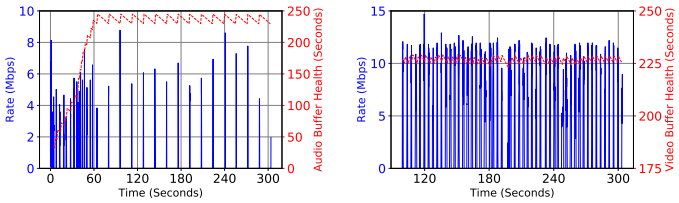
Fig. 3: Client metrics of a Netflix stream.

activity – *FlowB* has small spikes occurring every 16 seconds and *flowD* has large spikes occurring every 4 seconds.

Let us correlate this with metrics offered by the Netflix client application for the same video stream shown in Fig. 3. We show in Fig. 3(a) and 3(b) the buffer health of audio and video respectively which is measured in terms of: (a) volume in bytes (shown by solid blue lines and left y-axis) and (b) duration in seconds (shown by dashed red lines and right y-axis). We observe that the buffer health in seconds for both audio and video ramps up during the first 60 seconds of playback, till it reaches to a saturation level at 240 seconds of buffered content – thereafter, this level is consistently maintained by periodic filling. Note that the audio and video buffers are replenished every 16 and 4 seconds respectively, suggesting a direct contribution from the periodic spikes in network activity (observed in *FlowB* and *FlowD*).

Netflix client interface reports a metric called "throughput" which is an estimate of bandwidth available for the video stream. Fig. 3(c) shows the throughput (in Mbps, solid blue lines, on the left y-axis) and the buffering-bitrate of video (in Kbps, dashed red lines, on the right y-axis). We observe that the video starts at a low-quality bitrate 950Kbps, switches to higher bitrate 1330Kbps after 2 seconds, and jumps to its highest bitrate 2050Kbps after a second. Note that it stays at this highest bitrate for the rest of video playback even though far more bandwidth is available. Additionally, we note in Fig. 3(b) that the video buffer health in volume is variable while the buffer in seconds and the buffering bitrate are both consistent. This is because of variable bitrate encoding used by Netflix to process the videos where each video chunk is different in size depending on scene complexity. In contrast, buffer health volume for audio in Fig. 3(a) stays at 3MB with periodic bumps to 3.2MB – this indicates a constant bitrate encoding used for audio content and bumps occur when a new audio chunk is downloaded and an old one is discarded from the buffer. For audio, we observed (not shown in the Fig. 3(c)) a constant bitrate of 96Kbps throughout the playback.

Having analyzed streaming behavior on network and client individually, we now attempt to correlate them. We observed two distinct phases of video streaming: (a) the first 60 seconds of *buffering*, (b) followed by *stable* buffer maintenance. In the buffering phase, the client aggressively transferred contents at a maximum rate possible using four concurrent flows and then in the stable phase it transferred chunks of data periodically to replenish the buffer using only two flows.



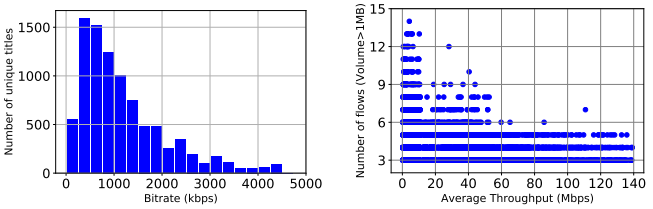
(a) Audio buffer (on client) versus corresponding flow (on network). (b) Video buffer (on client) and corresponding flow (on network).

Fig. 4: Correlation of network activity and client behavior.

Of the two flows active in stable phase, *FlowB* (with a spike periodicity of 16 seconds) displays a strong correlation between the spikes of its network activity and the replenishing audio buffer levels on the client, as shown in Fig. 4(a). This suggests that the TCP flow was used to transfer audio content right from the beginning of the stream. Isolating content chunks of this flow , we found that the average chunk size was 213KB with a standard deviation of 3KB (1.4%). Every chunk transfer corresponds to an increase of 16 seconds in the client buffer level. Considering the fact that each chunk transferred 16 seconds (indicated by both periodicity and increase in buffer level) of audio and the buffering bitrate of audio was 96Kbps, the size of audio chunk is expected to be 192KB which is very close to our computed chunk size of 213KB which includes the packet headers. Additionally, we note that for this specific flow, the server IP address differs from other flows (as shown in Fig. 2) and the Netflix client statistics also indicate that audio comes from a different CDN endpoint.

Further, *FlowD* (with a spike periodicity of 4 seconds) during the stable phase, displays a similar correlation between its network activity and the client buffer health of video, as shown in Fig. 4(b). The chunks of this flow have an average size of 1.15MB and a standard deviation of 312KB (27%). With each chunk constituting 4 seconds of video content and the video bitrate on client measured as 2050Kbps, the actual chunk size is expected to be 1.00MB which is close to the computed average chunk size while accounting for packet headers. Additionally, a high deviation in video chunks size also suggests that video is encoded using variable bitrate (in contrast, audio has a constant bitrate).

Trickplay: Having understood the streaming behavior during a normal playback (with no interruption), let us now analyze the behavior of Netflix streams during trickplay events. Trickplay occurs when the user watching the video decides to play another segment far from current seek position by



(a) Histogram of bitrate across all videos in our datset.

(b) Scatter plot of flow count versus average throughput.

Fig. 5: (a) Distribution of quality, and (b) correlation of network activity with available bandwidth.

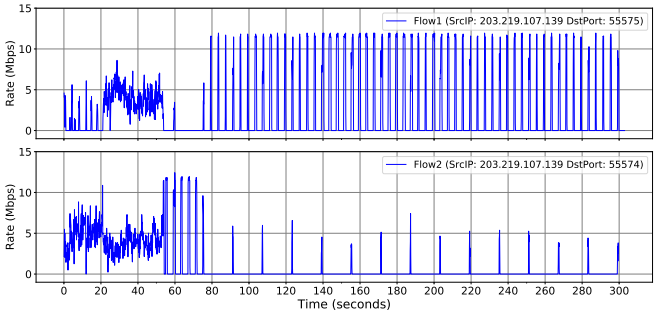


Fig. 6: Multiplexing audio and video over two TCP flows.

performing actions such as fast-forward, or rewind. A trickplay is performed either within the buffered content (*e.g.,* forward 10 seconds to skip a scene) or outside the buffered content (*e.g.,* random seek to unbuffered point). In the former case (within buffer), our observations show that the Netflix client uses existing TCP flows to fetch the additional content filling up the buffer up to 240 seconds. However, in the latter case, the client discards the current buffer and existing flows, and starts a new set of flows to fetch content from the point of trickplay. This means that trickplay outside the buffer is very similar to the start of a new video stream, making it difficult to determine whether the client has started a new video (say next episode in a series) or has performed a trickplay. For this reason, we consider a trickplay event equivalent to start of a new video stream and compute our experience metrics accordingly. Additionally, we note that for a stream in the stable phase, trickplay results in transitioning back to the buffering phase until the buffer is replenished. In §V, we will distinguish trickplay from network congestion that can cause a stream to transition into the buffering phase.

B. Analysis of Netflix Streams Across Our Dataset

Having looked at a representative video stream, we now analyze properties of Netflix streams in the dataset. Starting with the quality of streams across all instances in our dataset, we plot in Fig. 5(a) the histogram (with 20 bins) of the number of unique titles for a given video bitrate – the x-axis is capped at 5000 Kbps for readability of the plot. Note that each title is played at multiple bitrate values during a stream, as explained earlier. We make two observations: (a) Netflix videos are available in a fine granularity of bitrates in the range (*i.e.,* [80, 6100] Kbps) of bitrate – this is in contrast with [7], in which video providers employed a small discrete set of 7 bitrates. The availability of Netflix videos in many bitrates across the range, combined with variable bitrate encoding, makes it nontrivial to map a chunk size observed on the network to a particular quality bitrate, and (b) all movie titles are available at lower bitrates (*i.e.,* less than 1500Kbps), while only 517 titles in our dataset were available (or played) at a high-quality bitrate (*i.e.,* more than 3000Kbps).

Moving to correlation of active flows and network condition, we show in Fig. 5(b) the scatter plot of the total number of TCP flows (those with volume more than 1 MB) per each stream versus the average throughput (measured by Netflix client). Note that for each stream, we counted all TCP flows during both initial buffering and midstream (due to CDN switch or network congestion events). It is seen that Netflix often uses 3 to 5 TCP flows for the entire range of measured throughput – upon commencement of the stable phase only a couple of flows remain generally. We also observe that the flows count can go up to more than 12 when the available bandwidth is relatively lower (*i.e.,* less than 8 Mbps) – this is not surprising as Netflix attempts to spawn multiple flows to quickly fetch required contents for a smooth playback.

Lastly, we would like to point out certain challenges in analyzing Netflix behavior. We found that some TCP flows carry both audio and video contents (audio content is identified by chunk sizes of about 220KB and periodicity of 16 seconds in the stable phase) – both in an interleaved and alternating fashion. Also, each content type may switch TCP flows midstream – *e.g.,* we observe in Fig. 6 in the stable phase of a sample stream that Flow1 carries audio and Flow2 carries video at the beginning, but after about 20 seconds video is carried in Flow1 and audio is carried in Flow2.

Therefore, the mapping of a flow to the content it carries is nontrivial to determine. The complex and sophisticated orchestration of flows and their content type/quality makes it challenging to accurately predict all the client playback metrics purely based on network activity. In the next section, we use machine learning and statistical methods to compute a set of metrics (buffer-fill-time, average bitrate, and available throughput) per stream to infer user experience from network measurements.

V. Inferring Netflix QoE from Network

Having understood the behavior of video streaming, we now develop a method which uses just the network measurements to infer Netflix user experience (since ISPs do not have access to end clients). We first detect presence of Netflix video streams per host by using DNS, detect the phase of video playback and finally compute our metrics relevant to user experience.

A. Isolating Netflix Video Streams

Prior to video playback, the client sends a DNS query to fetch the IP address of Netflix streaming servers. To isolate flows corresponding to Netflix, we capture the `A`-type DNS response packets and inspect them for the suffix `nflxvideo.net` – if present, we mark the IP address as a Netflix streaming server. In parallel, we track five tuple flows

established to these streaming servers on a per-host basis. For example, given a user with IP address of 1.1.1.1, we track the connections from Netflix servers to this IP address in a separate data structure, and thus group all flows established by this user to the Netflix streaming server. For now, we assume that one host plays at most one video at any time – later we will describe a method to detect households with multiple parallel Netflix sessions. We note that an ISP can equivalently use any other method to isolate Netflix traffic, *e.g.,* SNI field present in server hello message sent during SSL connection establishment. We used DNS as it is simpler to capture and avoids the use of sophisticated deep packet inspection techniques required otherwise. However, we acknowledge that the DNS information may be cached in the browsers, thus every video stream may not have a corresponding query observed on the network. Nonetheless, maintaining a set of IP addresses (from previous DNS queries) will ensure that the video streams are captured.

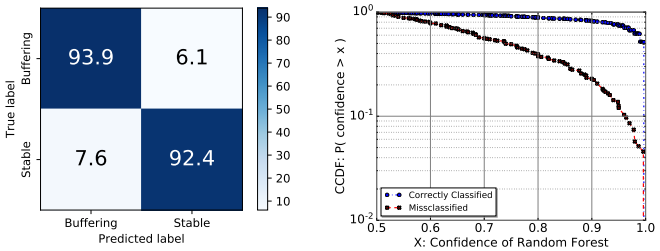
B. Streaming Phase Classification

Having isolated the TCP flows of a stream, we now build machine learning-based model to classify the phase (*i.e.,* buffering or stable) of a video streaming playback by using several waveform attributes (explained next).

Data Labeling: Each video streaming instance in our Netflix dataset is broken into separate windows of each 1-minute duration. We label a window of individual TCP flows associated with a stream using the client buffer health (in seconds) of that stream. For each window, we consider three measures namely the average, the first, and the last value of buffer health in that window. If both the average and last buffer values are greater than 220 seconds, then we label it as "stable". If both the average and the last buffer values are less than 220 seconds but greater than the first buffer value, then we label the window as "buffering". Otherwise (*e.g.,* transition between phases), we discard the window and do not use for training of our model.

Attributes: For each flow active during a window, we compute two sets of attributes. Our first set of attributes include: (a) **totalVolume** – relatively high during buffering phase; (b) **burstiness** (*i.e.,* μ/σ) of flow rate – captures the spike patterns (high during stable phase); (c) **zeroFrac**, fraction of time the flow is idle (*i.e.,* transferred zero bytes) – this attribute is expected to be smaller in the buffering phase; (d) **zeroCross**, count of zero crossing in the zero-mean flow profile (*i.e.,* [x-μ]) – this attribute is expected to be high in the buffering phase due to high activity of flows; and (e) **maxZeroRun**, maximum duration of being continuously idle – this attribute is relatively higher for certain flows (*e.g.,* aging out or waiting for next transfer) in the buffering phase.

Our next set of attributes are computed by isolating chunks of transfers from the flow profile. Each chunk in a flow is isolated by three successive data points of zero (*i.e.,* 300 ms idle after a transfer). Our five attributes computed from chunks are: (f) **chunksCount**; (g,h) **average** and **standard-deviation** of chunk sizes; (i,j) **average** and **mode** of chunks



(a) Confusion matrix. (b) CCDF of confidence-level.

Fig. 7: Performance of phase classification: (a) confusion matrix, and (b) CCDF of confidence-level.

inter-arrival time. In the buffering phase, the flow would have less chunks, lower inter-chunk time, and higher volume in each chunk compared to the stable phase. In total, for each flow in a window, we have 10 attributes computed (considering just the waveform profile, independent of available bandwidth) for each training instance (*i.e.,* 1-min window of a TCP flow).

Classification Results: We used the RandomForest ML algorithm available in Python scikit-learn library. We configured our model to use 100 estimators which are used to predict the output along with a confidence-level of the model. We split our labeled data of 12,340 instances into training (80%) and testing (20%) sets. We evalued the performance of our classifier using the testing set and obtained a total accuracy of 93.15%, precision of 94.5% and recall of 92.5%. We show in Fig. 7(a) the confusion matrix of our classifier. It is seen that 93.9% of buffering and 92.4% of stable instances are correctly classified. Fig. 7(b) illustrates the CCDF of the model confidence for both correctly and incorrectly classified instances. The average confidence of our model is greater 94% for correct classification while it is less than 75% for incorrect classification – setting a threshold of 80% on the confidence-level would improve the performance of our classification.

Use of Classification: For each TCP flow associated with a streaming session, we call our trained model to predict its phase of video playback. As explained earlier in §IV, multiple flows are expected especially at the beginning of a stream. We perform majority voting on outputs of the classifier for individual flows to determine phase of the video stream. In case we have a tie, we pick the phase with maximum sum confidence of the model. In addition to the classification output, the count of flows in the stable phase (*i.e.,* two flows) can be used to check (validate) the phase detection. This cross-check method also helps detect the presence of concurrent video streams for a household to discount them out of the analysis – having more than two Netflix flows for a household IP address, while the model indicates the stable phase (with a high confidence), likely suggests parallel playback streams.

C. Computing User Experience Metrics

We now identify three key metrics that together help us infer Netflix user experience. The first two are metrics directly related to experience, and the third one is used to deduce events affecting experience.

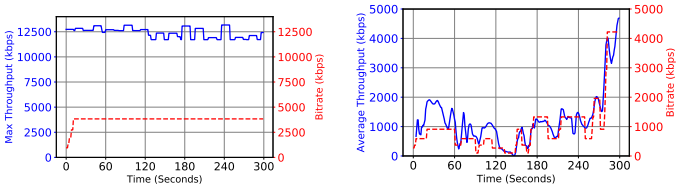
1) Buffer Fill-Time: As explained in §IV (by Fig. 3(a) and 3(b)), Netflix streams tend to fill up to 240 seconds worth of audio and video to enter into the stable phase – a shorter buffer fill-time implies a better network condition and hence a good user experience. Once the stream starts its stable phase, we begin by measuring *bufferingStartTime* when the first TCP flow of the stream was established. We then identify *bufferingOnly* flows – those that were active only during the buffering phase, go inactive upon the completion of buffering, and are terminated after one minute of inactivity (*FlowA* and *FlowC* shown in Fig. 2). We, next, compute *bufferingEndTime* as the latest time when any bufferingOnly flow was last seen active (ignoring activity during connection termination (*e.g.,* TCP FIN)). Lastly, the buffer fill-time is obtained by subtracting bufferingEndTime and bufferingStartTime.

Fill-Time Results: To quantify the accuracy of computing buffer fill-time, we use our client data of video buffer health (in seconds) as ground-truth. Results show that our method achieved 10% relative error for 75% of streams in our dataset – the average error for all streams was 20%. We observe that in some cases a TCP flow starts in the buffering phase and (unexpectedly) continues carrying traffic in the stable phase for some time after which it goes idle and terminates. This will result is our predictions of buffer fill-time to be larger than its true value thereby underestimating the user experience.

2) Bitrate: A video playing at a higher bitrate brings a better experience to the user. We estimate the average bitrate of Netflix streams using the following heuristics. During the stable phase, Netflix replaces the playback buffer by periodically fetching the video and audio chunks. This means that over a sufficiently large window (say, 30 seconds), the total volume transferred on the network would be equal to the playback buffer of the window size (*i.e.,* 30 seconds) since the client tends to maintain the buffer at a constant value (*i.e.,* 240 seconds). Therefore, the average bitrate of the stable stream is computed by divinding the volume transferred over the window by the window length. During the buffering phase, Netflix client downloads data for the buffer-fill-time and an additional 240 seconds (*i.e.,* the level maintained during the stable phase). Thus, the average bitrate of the buffering stream is computed by dividing total volume downloaded by sum of buffer fill-time and 240 seconds.

By tracking the average bitrate, we are able to determine the bitrate switches (*i.e.,* rising or falling bitrate) in the stable phase. As discussed earlier, there are a range of bitrates available for each video. For example, title "Eternal Love" was sequentially played at 490, 750, 1100, 1620, 2370, and 3480Kbps during a session in our dataset. We note that Netflix makes bitrates available in a non-linear fashion – bitrate values step up/down by a factor of ∼1.5 to their next/previous level (*e.g.,* 490×1.5 approximately indicates the next bitrate level 750). We use this pattern to detect a bitrate switch if the measured average bitrate changes by a factor of 1.5 or more.

Bitrate Results: We evaluated the accuracy of our bitrate estimation using the client data as ground-truth. For the average bitrate in buffering phase, our estimation resulted a mean absolute error of 158Kbps and an average relative error of 10%. The estimation errors for average bitrate in stable phase, were 297Kbps and 18% respectively. These errors arise mainly due to the fact that Netflix client seems to report an average bitrate of the movie but due to variable bitrate encoding, each scene is transfered in different sizes of chunks, hence a slightly different bitrate is measured on the network. Nonetheless, we note that detection of bitrate switch events will be accurate since the average bitrate would change by more than a factor of 1.5 in case of bitrate upgrade/downgrade.



(a) Good expeirence (bitrate saturates while more bandwidth available).

(b) Bad expeirence (bitrate follows stream throughput closely).

Fig. 8: Inferring user experience considering throughput.

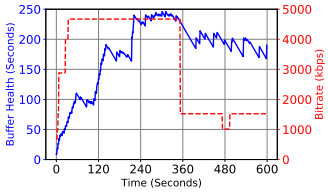
3) Throughput: We use the aggregate throughput measurements of the stream (obtained by summing up the throughput of all TCP flows involved) to detect experience events listed below. To do so, we derive two signals over a sliding window (say, 5 seconds) of the aggregate throughput: (a) max throughput, and (b) average throughput – note that the flow throughput is measured every 100ms.

Max bitrate playback. For a video stream, if the gap between the max throughput and the computed average bitrate is significantly high (say, twice the bitrate being played), then it implies that the client is not using the available bandwidth as it is currently playing at its maximum possible bitrate, as shown in Fig. 8(a), indicating a good experience.

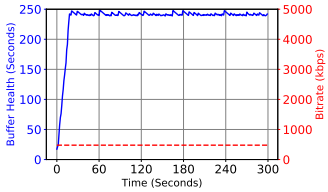
Bitrate variations during buffering. If the max throughput measured is relatively close to the bitrate ranges of Netflix (up to 5000 kbps) and is highly varying, it indicates possible bitrate switching events. In this case, the actual bitrate strongly correlates with the average throughput signal, as shown in Fig. 8(b). The fluctuating average throughput with high standard deviation (*i.e.,* $\geq$ 20%) causes the stream to switch bitrates and becomes unstable, indicating a bad experience.

D. Detecting Buffer Depletion and Quality Degradation

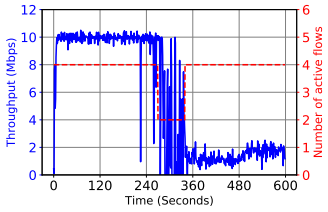
We now detect bad experiences in terms of buffer health and video quality using the metrics described above. To illustrate our method, we conducted an experiment in our lab whereby the available network bandwidth was capped at 10 Mbps. We first played a Netflix video on a machine, and one minute after the video went into the stable phase (*i.e.,* 240 seconds of buffer filled on client) we introduced UDP downstream traffic (*i.e.,* CBR at 8Mbps using `iperf` tool) to congest the link. For videos, we chose two Netflix movies – Season 3 Episode 2 of "Deadly 60" with high quality bitrate available up to 4672Kbps (Video1), and Season 1 Episode 1 of "How I Met Your Mother" with a maximum bitrate of 478Kbps (Video2).



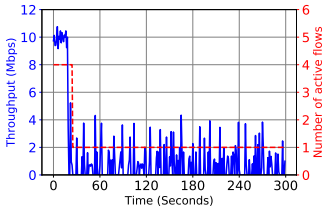
(a) Quality dropped due to congestion (client behavior of Video1).



(b) Quality maintained even with congestion (client behavior of Video2).



(c) Quality dropped due to congestion (network activity of Video1).



(d) Quality maintained even with congestion (network activity of Video2).

Fig. 9: Detecting quality degradation for users.

Fig. 9 shows client behavior (top plots) and network activity (bottom plots) for the two videos.

Considering Fig. 9(a) for Video1, it is seen that the stream started at 679Kbps bitrate (dashed red lines), quickly switched up, and reached to the highest possible value 4672Kbps in 30 seconds. It continued to play at this bitrate and entered into the stable phase (at second 270) where only two flows remained active, as shown in Fig. 9(c), and the buffer health (solid blue lines) reached to its peak value of 240 seconds. Upon commencement of congestion (at second 340), we observe that the buffer started depleting followed by a bitrate drop to 1523Kbps. Moving to the network activity in Fig. 9(c), we observe that two new flows spawned, the stream went to the buffering phase, and the network throughput fell below 2Mbps. The change of phase, combined with a drop in throughput, indicates that the client experiences a buffer depletion – a bad experience. Using our method, we detected a phase transition (into buffering) at second 360 and deduced bitrate from the average throughput (as explained earlier in Fig. 8(b)), ranging from 900Kbps to 2160Kbps. This estimate shows a significant drop (*i.e.,* more than a factor of 1.5) from the previously measured average stable bitrate (*i.e.,* 3955Kbps). Additionally during the second buffering phase, we observe a varying average throughput with the mean 1.48Mbps and the standard-deviation 512Kbps (*i.e.,* 35% of mean) indicating a fluctuating bitrate on the client. We note that although a transition from stable to buffering can result from a trickplay (discussed in §IV) we do not detect a bad experience since no change in max throughput is observed.

Moving to Fig. 9(b) and 9(d) for Video2, the stream played consistently at the bitrate 478Kbps and quickly transitioned into the stable phase within about 20 seconds. It started with 4 active flows with aggregate throughput of 10 Mbps, but only one flow remained active after entering into the stable phase – this flow was responsible for both audio and video contents. Upon arrival of UDP traffic (at second 80), no change is observed in the playback. Employing our method for experience metrics, we estimated a buffer fill time of 17.5 seconds, average buffering bitrate of 652Kbps, and correctly predicted the stream to be in the stable phase with bitrate reported every minute as 661, 697, 658, 588Kbps. Additionally, the max throughput was accurately predicted to drop from 10Mbps to 4Mbps. We note that even though the bitrate and throughput are relative low during the stable phase, the playback is smooth and the experience is not bad. We believe that our stream phase detection, combined with estimation of bitrate and throughput, enables us to distinguish a good experience from a bad experience which could arise due to quality bitrate degradation and buffer depletion events.

VI. Conclusion

Netflix is a widely-used video streaming application and network operators are seeking visibility into its user experience. In this paper, we presented a practical method to infer Netflix user experience from broadband network measurements in real-time. We developed a measurement tool and collected network activity and client behavior data for 8000 Netflix streams. We made our dataset publicly available. We then highlighted the correlation between network flows activity and client buffer health. Finally, we developed a model to predict the streaming phase and inferred Netflix user experience in terms of buffer-fill time, average video bitrate, and available bandwidth to the stream.

References

[1] Sandvine. (2018, Oct) The Global Internet Phenomena Report. [Online]. Available: https://bit.ly/2zNZBde

[2] T.-Y. Huang *et al.*, "A buffer-based approach to rate adaptation: Evidence from a large video streaming service," *ACM SIGCOMM CCR*, vol. 44, no. 4, pp. 187–198, 2015.

[3] X. Yin *et al.*, "A control-theoretic approach for dynamic adaptive video streaming over http," in *ACM SIGCOMM CCR*, vol. 45, no. 4, 2015, pp. 325–338.

[4] V. K. Adhikari *et al.*, "Unreeling netflix: Understanding and improving multi-cdn movie delivery," in *Proc. IEEE INFOCOM*, Orlando, FL, USA, March 2012.

[5] J. Martin *et al.*, "Characterizing netflix bandwidth consumption." in *IEEE CCNC*, 2013, pp. 230–235.

[6] A. Reed *et al.*, "Identifying https-protected netflix videos in real-time," in *Proc. ACM CODASPY*, Scottsdale, Arizona, USA, March 2017, pp. 361–368.

[7] T. Mangla *et al.*, "emimic: Estimating http-based video qoe metrics from encrypted network traffic," in *IEEE TMA*, Vienna,Austria, June 2018.

[8] V. Krishnamoorthi *et al.*, "Buffest: Predicting buffer conditions and real-time requirements of http (s) adaptive streaming clients," in *Proc. ACM MMSys*, Taipei, Taiwan, June 2017.

[9] V. Aggarwal *et al.*, "Prometheus: Toward quality-of-experience estimation for mobile apps from passive network measurements," in *Proc. ACM HotMobile*, Santa Barbara, CA, USA, Feb 2014.

[10] G. Dimopoulos *et al.*, "Measuring video qoe from encrypted traffic," in *Proc. ACM IMC*, Santa Monica, CA, USA, March 2016.

[11] I. Orsolic *et al.*, "A machine learning approach to classifying youtube qoe based on encrypted network traffic," *Springer, Multimedia tools and applications*, vol. 76, no. 21, pp. 22 267–22 301, 2017.

[12] D. Tsilimantos *et al.*, "Classifying flows and buffer state for youtube's http adaptive streaming service in mobile networks," in *Proc. ACM MMSys*, Amsterdam, Netherlands, June 2018.

[13] M. H. Mazhar *et al.*, "Real-time video quality of experience monitoring for https and quic," in *IEEE INFOCOM 2018-IEEE Conference on Computer Communications*. IEEE, 2018, pp. 1331–1339.