Network Microscope: Real-time Monitoring and Inference for Access Networks

Abstract

We design, implement, deploy, and evaluate Network Microscope, a new system that enables data-driven network management of access networks. Network Microscope's main innovation is recognizing that, while network measurement forms the crux of many performance and security-related questions in access networks, today's techniques for measuring these networks are not well-suited to the higher-level questions that the operators of ISPs and platforms need to answer. We posit, however, that many of the network management tasks in access networks can be framed as statistical inference questions that rely on a common set of network features that can be efficiently collected at the network edge. Based on this insight, we design and build a system to collect these features, deploy it in more than 60 homes, and show how Network Microscope can be used to study performance and security questions across the deployment. In our deployment, we focus extensively on inferring streaming video quality of experience, given the dominance of this application in residential broadband access networks. Yet, we also show that Network Microscope is modular and extensible enough to answer a broad array of questions, including those related to smart home security and privacy.

1 Introduction

"You can't improve what you don't measure."

Managing access networks generally involves many tasks, two of which are improving performance and defending against attacks. Both tasks require collecting data and using that data to make inferences—typically as quickly as possible. In residential broadband access networks, this challenge is particularly acute, for several reasons. First, security and performance are paramount. The edge of the network has become the front line of Internet-scale attacks, as users increasingly connect insecure "smart home" devices to their home networks [9]; and, the home network is where consumers experience many performance problems relating to dominant applications, such as streaming video. Second, ISPs lack visibility into the home network; in addition, trends towards increased privacy, such as Europe's General Data Protection Regulation (GDPR), which require explicit user opt in, and trends towards end-to-end encrypted transport, including DNS (*e.g.*, DNS over HTTPS), are creating increased technical and legal hurdles for network management that relies on inspecting any traffic on the access network [15, 22].

As broadband access network speeds increase and user applications proliferate, the status quo for managing residential broadband access networks has become brittle and much less effective. Take, for example, the problem of network performance measurement. For decades, the gold standard of performance measurement has been the "Internet speed test" [39,42], which amounts to attempting to transfer as many bytes between two points on the network as quickly as possible. Such "active" measurement made sense when the access link was the main performance bottleneck, but faster access speeds have made performance measurement an increasingly complex problem, as network performance bottlenecks have shifted to many different network locations from the home network to the content delivery network, and as the performance of the *applications* are limited by factors that are increasingly distinct from measurement of network throughput along an essentially arbitrary path. Or, consider the security problem of actively scanning a network for vulnerabilities, which is becoming increasingly challenging given the proliferation of IPv6 (which is far too large to scan comprehensively) and Network Address Translation (which completely obscures large portions of the network).

These developments lead us to ask whether *passive collection and analysis of network traffic in a user's home network* can provide better solutions to these problems. The answer might seem obvious—-after all, passive measurement is well-established. And yet, the existing *general* passive measurement techniques that are designed for high-speed backbone networks, such as packet capture or flow-based monitoring (*e.g.*, IPFIX, NetFlow) are not well-suited for performance analysis and monitoring attacks. Flow-based monitoring is too coarse to enable the analysis that could help network operators answer even basic questions about the performance of specific applications, such as video streaming, and for identifying attacks. On the other hand, collecting packet traces gathers data that is too voluminous. To strike a balance between these two extremes, more recent methods rely on streaming algorithms [8, 27, 28, 47, 48, 50] that extract relevant information or filter packets in real-time to reduce the volume of per-packet monitoring while enabling more fine-grained analysis.

Still, because these approaches are designed to operate in the scale of enterprises, data centers, or network backbones, the resulting feature sets are both too coarse for detailed analysis and too heavyweight for real-time collection and analysis on resource-constrained embedded devices deployable within home networks.

In this paper, we present a new approach to real-time inference for network management that relies on *jointly* designing the collection methods and the corresponding inference algorithms: (1) tailoring the machine learning models using features that network devices can capture at line rate; and (2) designing the network monitoring systems in ways that can collect the sets of features that are both necessary and sufficient to inform the models. Such an approach ensures that collection remains efficient (and private), while still yielding features that can facilitate the management of these networks.

We realize our research in a system called Network Microscope, a system that can passively collect a corpus of fundamental network features about the streams of interest in the network (*e.g.*, all video streams) and direct those to a real-time inference algorithm that allows a network operator to determine degradations in application quality or attacks, *as they happen*. As an initial important use case, we focus on inferring streaming video quality—a problem that, while studied extensively, has yet to result in a system that can infer video quality across a wide range of videos, in real time.

Looking ahead more broadly, this work demonstrates that network management problems can and should be solved with an approach that co-designs inference and packet capture. Although we have focused on home network deployments for video quality inference, the system and approach we have developed is general. For example, we are using it to detect other performance and security-related events, including detecting denial of service attacks and tracking third-party communications from "smart" devices, as we describe in more detail in Section 6. A major takeaway from our work, aside from the video quality inference system itself, is that the same sets of features that are lightweight enough to apply to a real-time inference model are also typically *general enough* to be used in models for other types of problems. We believe that this result points to exciting future directions for real-time network inference, where network inference algorithms are tailored for the sets of features that systems like Network Microscope can extract in real-time, and models that rely on these feature sets are developed to solve a wider range of network management problems, from performance to security.

2 Problem Context and Design Goals

In this section, we first describe why monitoring in access networks close to users is important and why existing approaches do not enable the types of monitoring and inference of the access network that ISPs need. We then outline the design goals for Network Microscope.

2.1 Monitoring in Access Networks

Internet service providers need better ways to monitor traffic in the access network close to users (*e.g.*, in the home, near the head-end). For example, an Internet service provider may want to understand the streaming video quality that a subscriber is experiencing; while sometimes this can be inferred from traffic captured in the middle of the network, this visibility can also, for example, associate this traffic with a particular device or correlate the performance with the quality of the home wireless network—both of which have historically introduced performance bottlenecks on end-to-end paths.

To date, most of the existing techniques for access network measurement have been focused on active measurement: introducing measurements into the network and measuring their effects, akin to the techniques used by Speedtest [40], the FCC's Measuring Broadband America Program [6, 42], and other similar techniques. As mentioned, from both a performance and security perspective, these measurements are most useful when they are precise and can be associated with an individual device. Conventional active measurement approaches in access networks are quickly hitting fundamental limitations as the speeds of access networks increase. Active throughput tests have difficulty saturating the access link capacity and at the same time fail to convey any information about the performance of the actual user applications. Similarly, active measurements can scan networks for vulnerabilities but do not scale for larger IPv6-based networks and cannot detect actual attacks.

These limitations point to passive measurement as a possible solution. Passive network measurement is, of course, a mature, well-established practice, and there are a wide array of deployed network monitoring systems that exist, especially in enterprise, data centers, and large backbone networks. Yet, conventional passive network measurement introduces its own set of challenges. Generally speaking, most existing systems fall into two categories. Flow-based monitoring collects coarse-grained statistics (*e.g.*, IPFIX/NetFlow collection infrastructure such as Kentik [24] and Deepfield [10]). To scale to high rates in the middle of the network, flow monitoring systems resort to sampling, which decreases even further the accuracy of inference tasks [16]. Per-packet monitoring is either focused on fairly specialized applications (*e.g.*, software such as Corelight [7] for enterprise security) or triggered on demand to capture some subset of traffic for further analysis [50]. In some cases, programmable hardware has been tailored to collect more detailed statistics about traffic flows [28], but these statistics focus on feeding network-wide inferences (*e.g.*, heavy hitter detection) and are not well-suited to perform more complex monitoring tasks for performance or security (*e.g.*, tracking video segment interarrival time, inspecting DNS traffic).

Additionally, there is often a need to perform analysis and inference in the home network, for visibility beyond the NAT

and to abide by privacy requirements. A large Internet service provider can have tens of millions of subscribers; processing and analysis close to the edge of the network can also be critical for scaling, as network speeds, traffic rates, and the number of devices increases. To deploy a system in potentially millions of homes, the system should be able to operate on an embedded device, whether that device is a CPE, set-top box, or other embedded device that sits in the home network.

Thus, there exists a gap in capabilities for what is most applicable for monitoring access networks—devices and systems that can operate at moderately high speeds (*i.e.*, 1 Gbps), but trade off some amount of processing rate capabilities for the ability to perform more flexible monitoring and more sophisticated inference in real-time. We aim to develop a system that fills this gap.

2.2 Design Goals

To support real-time performance characterization and monitoring for network traffic in access networks, an effective system must meet the following requirements.

1. Support flexible and extensible data analysis pipelines for evolving applications. Internet applications are rapidly and constantly evolving. As such, a measurement platform that aims to characterize application traffic must offer the capability for users to customize measurements to applications of interest. Collecting all possible traffic features for all services simply would not scale, especially if a feature requires the system to hold persistent state information. At the same time, passive inference requires an in-depth understanding of application behavior in order to translate network traffic to application performance. A passive system must allow the user a means to specify which features to monitor for each service, and it should support different traffic analysis techniques based on the application and traffic data. This requires allowing easily extensible processing routines, while minimizing performance degradation.

2. Support increasing access network speeds. Access network throughput is rapidly increasing. For example, a one-minute packet capture of a saturated 1 Gbps link generates 7.5 gigabytes of raw data. Fortunately, packet capturing and processing at line rate in software has become more and more feasible thanks to the introduction of tools such as PF_RING [11] and DPDK [14], which enable network traffic to reach userspace processes at high speeds. Yet mere capture is not sufficient, as inference models generally require computing a collection of features at line rate. Non-network bottlenecks also arise, such as limitations on disk I/O and CPU processing power. Optimizing packet capture and processing given the available resources is key. Scaling across this diverse set of environments imposes on the platform the need to flexibly adapt to different hardware architectures. For example, if deployed inside a household, the system would have to be compelling for the user hosting it. To do so, the platform would need to be inexpensive and non-intrusive, possibly deployed using low-cost, commodity devices (*e.g.*, multipurpose boards [32, 35]). On the other hand, ISP-managed access aggregation points could exploit dedicated, custom hardware (*e.g.*, hardware-supported in-band network telemetry) [38] to assist with the collection and export of statistics about traffic flows, even on higher-speed links.

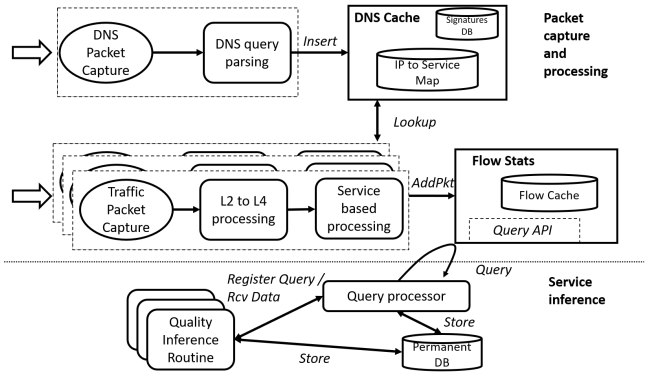


Figure 1: *Network Microscope system overview.*

3. Respect user privacy. By passively measuring traffic close to users (and, often in the home network itself) the system raises privacy concerns as it collects traffic that can be attributed to individual users and devices. With access to raw traffic, the system can observe and collect potentially private information (*e.g.*, Alice visited YouTube). Although web traffic is increasingly sent using encrypted transport, even the collection of DNS queries and traffic volumes can often reveal private information about the websites that users visit, as well as information about how they interact with various "smart" devices in their homes. As such, the system should be as conservative as possible concerning any detailed traffic statistics that it exports from the local network. In addition to limiting the volume and granularity of what is *collected*, the system aims to perform as much *inference* as possible on the local device itself, both for the scalability reasons mentioned above and to avoid collecting fine-grained or private information at a central location.

3 Network Microscope

Network Microscope has two distinct components: (1) a *packet processing* module that collects network flow statistics and tracks their state at line rate; moreover, this block implements a short-term cache used to store flow state information; (2) an *inference* module that queries the flow cache to obtain features and statistics about each traffic flow and derives higher-level features concerning the applications of interest (*e.g.*, quality of experience metrics for video streaming, such as startup time and rebuffering events). Figure 1 summarizes the system design.

3.1 Packet Processing Module

Network Microscope's packet processing module is implemented in Go (`https://golang.org/`). Processing network traffic in software is more achievable than it has been in the past; yet, coupling passive network performance measurement and inference involves developing new efficient algorithms and data structures for traffic collection and analysis. For example, to minimize processing and state for packets and flows that are irrelevant to the inference task, the packet processing module selects only the traffic flows that are relevant to applications of interest and subsequently categorizes each of these flows by application and service. For example, for the inference of video quality we will discuss in the next section, Network Microscope automatically identifies video traffic and classifies it according to whether it is Facebook, YouTube, Netflix, Hulu, Amazon, or Twitch traffic. Network Microscope's packet processing module has five components:

1. Packet capture. To support fast (and increasing) access network speeds, Network Microscope integrates a packet capture interface based on PF_RING [11] and the gopacket *DecodingLayerParser* library [20]. We design Network Microscope to allow for a pool of parallel traffic processing worker processes, which allows the system to scale capacity and take advantage of multicore CPUs. As such, we use flow clustering (in software or hardware depending on the available resources) to guarantee that packets belonging to the same flow are delivered to the same worker process. For compatibility and reproducibility purposes, Network Microscope also supports *libpcap* based packet capture. This allows the support of a higher number of device architectures and reproduce analysis on recorded network traces.

2. Signature-based traffic categorization. Real-time inference requires efficiently isolating traffic to and from various services at line rate without breaking encryption. To do so, Network Microscope maintains a short-term cache to map remote IP addresses to services accessed by local clients. A separate process maps flows to services by tracking DNS queries and responses. When Network Microscope captures DNS packets, it inspects the hostname in DNS queries and matches these lookups against a corpus of regular expressions for domain names that we have derived for those corresponding services. For example, `(.+?\.)?nflxvideo\.net` captures a set of domain names corresponding to Netflix's video caches. For further flexibility, Network Microscope supports specifying matches between services and IP prefixes, which assists with mapping when DNS lookups are cached (which we notice with some native implementations of video players that initiate flows without the initial DNS resolution).

As with any signature-based matching method, the challenge is to keep the list of signatures complete and up-to-date. It is possible to semi-automatically generate signatures by capturing traffic of the target video service in controlled settings to identify the set of domain names and IP prefixes to monitor. Additionally, many publicly available domain lists map domains to applications to allow network administrators to implement blocking and filtering in corporate networks. For our initial release, we distributed a database of signatures of a number of known over-the-top video services.

Relying on DNS to map traffic to applications and services may prove challenging in the future, as DNS becomes increasingly sent over encrypted transport (*e.g.*, DNS-over-HTTPS). In such situations, Network Microscope could still operate on path between the clients and its respective DNS resolver (*e.g.*, at the home network gateway), whereby a Network Microscope could proxy the DNS traffic, even if it is encrypted end-to-end.

3. Short-term flow cache. Upon receiving a packet belonging to a new traffic flow, the processing module creates the required data structure and stores it in a short-term cache. The flow data structure includes a service identifier, indicating whether the flow is associated with an application of interest as specified by the signature database. This data structure allows the system to maintain unique features and statistics for different applications in a single cache. The cache supports millions of concurrent read and write operations per second; this performance allows Network Microscope to read and update state at high packet rates. The cache purges entries for flows that have been idle for a configurable amount of time. In our configuration the timeout is 1 day. Each cache entry is stored using a general flow data structure, which allows for different flow types, and differing underlying content, to be stored in the single cache.

4. Service-driven packet processing. A separate worker pool processes all non-DNS packets. Each worker has a dedicated capture interface to read incoming packets. The first step parses MAC, network, and application headers, which yields useful information such as the direction of the traffic flow, the protocols, and the addresses and ports of the traffic. The system then performs additional operations on the packet depending on the service category assigned to the packet by inspecting the flow's service identifier in the short-term cache. Network Microscope uses convenient flow abstraction interfaces to allow for quick implementation of collection methods for different statistics. In particular, Network Microscope obviates the need for developers to re-implement functions to compute basic statistics or store the collected state. A developer solely needs to implement the function that processes service-specific state collection upon receipt of new packets belonging to the flow. Based on this specification, Network Microscope stores the flow cache over time and periodically exports the information. The output period is configurable, in our testing we set the system to output every 5-seconds. Section 4 describes how we tailor the packet processing for streaming video quality inference.

5. Query API. Network Microscope exposes an API that provides access to the information in the cache. Queries can

Network Layer	Transport Layer
○ throughput up/down	○ # flags up/down (ack/syn/rst/push/urgent)
○ throughput down difference	○ receive window size up/down
○ packet count up/down	○ idle time up/down
○ byte count up/down	○ goodput up/down
○ packet inter arrivals up/down	○ bytes per packet up/down
○ # of parallel flows	● round trip time
	● bytes in flight up/down
	● # retransmissions up/down
	● # packets out of order up/down

Table 1: *Features that Network Microscope extracts; annotations indicate the complexity of state required: Simple state (○), Complex state (●).*

be constructed based on either an application (*e.g.*, Netflix), or on a given device IP address. The current version of the system implements a module to periodically query the API to dump all collected statistics to a temporary file in the system.

3.2 Common Traffic Features

Network Microscope provides a set of common features for all network flows to allow for straightforward statistical analysis. Features are not limited to this common set, as users are able to define additional, custom (*e.g.*, application-layer) features for collection. For all traffic flows, we can track the set of common metrics summarized in Table 1. We also loosely characterize the cost of computing these features, which is roughly correlated with the amount of state required to collect and compute the associated feature.

Network Layer. For each network flow, we compute the upstream and downstream average throughput as well as average packet and byte counts per second. We also compute the difference of the average downstream throughput of a flow between consecutive time slots. We calculate the upstream and downstream average throughput for all traffic. Finally, we track interarrival times for both upstream and downstream packets as well as the number of parallel flows between hosts. The state required for network layer features is low; modern hardware can compute each of these features in-line with a single register (*e.g.*, to store counters for throughput or timestamps for packet time arrivals).

Transport Layer. We extract transport-layer features from information in TCP/UDP headers, computing each feature separately for upstream and downstream traffic. We calculate the features by analyzing individual packets, and their relative state in the lifetime of the flow. Exclusively for TCP, we extract retransmissions, window sizes, out-of-order packets, and TCP flag information. All other transport features in the table are computed for UDP, as well. We compute summary statistics such as mean, median, maximum, minimum, standard deviation, kurtosis, and skewness for features over the lifetime of the flow. The cost of computing each feature depends on whether the full state of the flow is required: features such as goodput and TCP flag counters (in the upper part of the table) require simple state and can be computed in-line similarly to the network-layer features. The remaining features require more complex mechanisms to inspect each packet and track state across packets belonging to the flow, as shown in the bottom of the table.

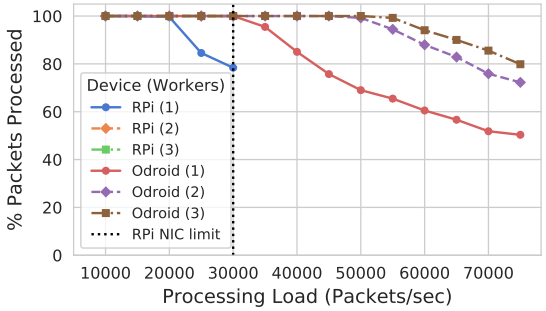


Figure 2: *System processing capacity on Odroid and Raspberry Pi platforms.*

3.3 Performance Benchmarks

In this section, we evaluate the performance of Network Microscope's packet processing module. In particular, we examine the traffic processing capacity of Network Microscope on different low-cost hardware platforms. We alter traffic load as well as the number of parallel traffic processing functions. In our testbed, we use `tcpreplay` to write network traffic from a pcap file onto a desktop computer's network interface that is directly connected to our Network Microscope hardware devices. The pcap is a five-minute capture from a busy private network's access point containing 139 unique local IPs on the network [44]. The average packet rate is 2,638 packets per second, the data rate is 9,477 Kbps, and the average packet size is roughly 450 bytes. `Tcpreplay` allows us to adjust the rate at which the traffic in the pcap is sent on the wire. The traffic load of this packet capture is relatively high compared to a traditional home network, where we envision the low-cost Raspberry Pi or Odroid platforms would be installed. We configured Network Microscope to collect the common network and transport layer features for all flows.

Figure 2 shows the results: as expected, both platforms maintain higher processing performance when using more than one traffic parser workers. The Raspberry Pi platform with two or three workers processes 100% of packets when sending at rates up to 30,000 packets per second, roughly equal to the full throughput capacity of the 100 Mbps onboard NIC when replaying the trace pcap. The Odroid with three workers processes 100% of packets up to 55,000 packets per second (roughly 190 Mbps using the trace pcap) before experiencing drops. Note that we do not test more than 3 workers as both hardware platforms feature quad-core CPUs. We leave the remaining available core for the DNS processor and query engine (Figure 1). We found that additional workers resulted in decreased capacity due to context switching.

Although the Odroid would likely be unable to fully process a saturated 1 Gbps link, the scenario we evaluate represents a

network that is much busier than a typical home. Small access networks do not typically experience the high sustained traffic loads that we tested. For comparison purposes, we monitored a university /18 subnet that serves 88 buildings including dormitories and other campus residences. The peak traffic load for this network during the evening hours was steady at roughly 100 Mbps with short peaks around 200 Mbps. Given the results in Figure 2, we are confident that even a low-cost device such as an Odroid could serve this busy network. In busier networks, higher-performance hardware could be used. For example, in testing we found that a laptop with eight cores could process 900 Mbps with no packet loss.

Signature performance. Our processing capacity experiments illuminated the signature database as an important factor for Network Microscope's performance. Recall from Section 3.1 that each DNS query is matched against the signature cache in order to categorize flows as they begin. Large signature caches will result in delays as DNS queries must search the entire signature cache for a match. For example, we ran our experiments with long signature lists containing domains for ads and trackers. One ad list included 2,700 domains and another included roughly 46,000 domains. When we enabled categorization using the ad lists in the cache we saw drops begin at 45,000 packets per second on the Odroid with 3 workers compared with loss beginning at 55,000 in Figure 2 without ads enabled. Therefore, care must be taken to optimize the DNS cache list in environments where the expected traffic load is at or near the capabilities of the hardware. For example, placing the most popular signatures at the top of list can help reduce the signature matching time for the majority of flows.

4 Case Study: Video Quality Inference

In this section, we describe how we applied Network Microscope to infer the quality of streaming video, for four major streaming video services: Netflix, YouTube, Amazon Instant Video, and Twitch. In our own previous work, we developed a general model to infer higher-level characteristics of the streaming video from the lower-level features that are evident in encrypted traffic. In testing and deployment, the Network Microscope operates as shown in Figure 1.

Designing Network Microscopeto operate for real-time streaming video inference requires tailoring packet capture to efficiently focus only on the aspects of the network traffic that are relevant to inferring video quality. Here, the aspects of co-designing collection and inference become paramount: Network Microscope must ideally capture *only* the aspects and features in network traffic that are relevant to video quality inference. This entails three steps:

1. Identifying traffic corresponding to video sessions. (Section 4.1)
2. Within this traffic, dividing the traffic into video segments. (Section 4.2)
3. Based on the resulting video segments, extracting features for an inference model. (Section 4.3)

The rest of this section describes each of these steps.

4.1 Identifying Video Sessions

Network Microscope must first identify that a video session is ongoing. Every ten seconds, Network Microscope scans the statistics collected by the pipeline to determine whether a new session has started. We build on the specific approach presented by Dimopoulos *et al.* [12], which observes that there is a spike in traffic to specific YouTube domains in the beginning of a video session and a silent period at the end. We design a session identification method that generalizes this intuition across video services.

Our analysis of video sessions across different services confirms that at the beginning of each video session, there is a spike in the volume of traffic that comes from servers that are associated with the video services but are distinct from the servers that deliver video traffic. This activity can correspond to, for example, the download of the player code or thumbnail image downloads as users browse the catalog.

Two other features are useful for identifying session boundaries. First, in addition to generating more traffic to other servers, video players also generate new network flows to download audio and video content when moving from one video to another. Second, considering the buffer-based approach of video services, most sessions have no video traffic at the end, when the player is exclusively rendering the buffered content without having to further retrieve new segments.

Network Microscope identifies a new session under the following conditions:

1. The amount of non-video service traffic exceeds a threshold α or its rate over the entirety of the service traffic exceeds a threshold β.
2. The time elapsed since the last video traffic activity exceeds a threshold γ or the rate of new flows initiated exceeds a threshold δ.

We set the thresholds based on the general behavior of the studied protocols. In particular, we set α to 50 kbps and β to 5% following the distribution of sessions collected in the dataset described in [1]; both are relatively conservative, but we rely on the second condition of our detection method to further increase the accuracy. γ is set to 0.5; this value offers a good trade-off between YouTube, that for most sessions uses only a single flow dedicated to video retrieval, and Netflix, which, in contrast, uses a large number of parallel flows. Finally, δ is set to 30 seconds which corresponds to a small playback buffer with respect to the max buffer size of these services (for example, Netflix uses up to 220 seconds worth of buffered video). We anticipate that in cases where other video services have drastically different traffic profiles, per-service tuning of the thresholds may be necessary.

Validation. Figure 3 shows the cumulative distribution function of the error in video session start time; specifically,

the figure shows the time between the detected session start minus the ground truth session start as seen from the web history (taking the timestamp of the first video or audio segment request) for all video sessions. The results show that the estimate for startup delay is within a few seconds of the actual start time for most sessions; in some cases, the technique infers a shorter startup delay, perhaps as a result of non-video traffic that Network Microscope mistakenly attributes to video traffic. Additionally, Network Microscope extracts and reports features at a fixed period (*e.g.*, every n seconds), which makes it impractical to identify an exact start time; previous approaches such as Dimopoulos *et al.* [12] perform inference *per session*, as opposed to continuously at fixed time intervals. Based on the results of our validation, we modified Network Microscope inference model for startup delay to rely on the first 60 seconds of the video stream.

4.2 Identifying Video Segments

The next step in video quality inference is identifying individual segments (and segment boundaries) within a video streaming session. Krishnamoorthi *et al.*. [25] mentioned, without formalizing, that individual video segment information can be extracted by observing patterns in upstream packets. The key idea behind this method is to use the times of upstream requests to break down the stream of downstream packets into video segments. Although newer versions of HTTP permit pipelining of multiple requests over a single connection, adaptive video plays one video segment after the other and hence sequential request-response is natural. The key challenge is how to identify the times of requests for video segments within the stream of packets from the client to the server. Fortunately, upstream packets are either requests for video segments or acknowledgments, so we can identify requests based on packet size.

We implement this segment identification mechanism Network Microscope by tracking the last seen upstream packet with a non-zero payload and counting the subsequent downstream traffic to determine segment sizes. The system then stores information of the detected segments in its short-term cache. Every ten seconds,Network Microscope uses the query API to write to file the collected statistics. The described behavior is similar for QUIC, except that QUIC signaling packets have non-zero payload size. We then say that a QUIC packet contains a request if the UDP payload size is greater than a threshold. We set this threshold to 150 bytes based on the current QUIC specification [36] and the empirical distribution of QUIC upstream packet sizes in a dataset of QUIC YouTube sessions we collect.

Validation. We validate the techniques used to infer segment boundaries. We use a dataset of about 10,000 sessions from three major video services: Netflix, Amazon, and YouTube We validated our video segment identification technique using ground truth browsing history, collected via a browser plugin, from the sessions to build a time series of downloaded video/audio segments during each experiment. We isolate individual segment requests by inspecting the URL of the web requests.

We characterize video segments in terms of both the number of segments and segment sizes in a given video session. Figures 4a and 4b show the accuracy of this approach. The method works similarly across all three services tested; it slightly underestimates the number of segments in a video session. We suspect two possible reasons for this underestimation: (1) session detection may indicate that a session start time is slightly later than the actual start time, thus causing detection to miss a few segments at the beginning of a session; and (2) some services generate a number of empty requests at the beginning of a session, possibly to test the path to their video caches, which may cause the small error shown in Figure 4a. Evaluation of segment size estimation (Figure 4b) shows that we accurately infer segment size for all three services in the general case, with an error within 1% of the actual segment size in the majority of cases. Overestimates may be caused by double-counting retransmissions and the fact that some services periodically issue empty requests.

4.3 Inferring Video Quality

After Network Microscope identifies video sessions and divides the stream into segments, the next step is to infer the quality of each segment. Video quality inference is well-studied: Previous work has demonstrated the feasibility of using flow statistics obtained from passive measurement to infer the quality of a video session from encrypted traffic [1, 12, 25, 25, 30, 33]. Subsequently, we evaluated all of these models and discovered that none of them generalize across all video services—they are typically designed to operate on only a single video service. Additionally, many of them require features and processing that must be performed offline, precluding the ability to perform inference in real-time on embedded devices at the edge of the network, which is our goal.

We thus extended previous work, developing a new general model that can infer startup delay and resolution across all services. The models we develop operate on the features that we summarized in Table 1 as well as some custom application-layer features that we include, such as segment inter-arrival time. The models—including the evaluation of their accuracy—are discussed at length in an accompanying technical report [1]. All of the models that Network Microscope uses rely on lightweight features (*e.g.*, upstream and downstream throughput, inferred video segment sizes) and outperform the accuracy of those that require maintaining more complex per-flow state (*e.g.*, bytes in flight, number of retransmissions).

[1]We were unable to obtain ground truth for segment sizes for Twitch because the application uses chunked transfer encoding, where the size of the segment is inline with the HTTP payload, as opposed to in a separate header.

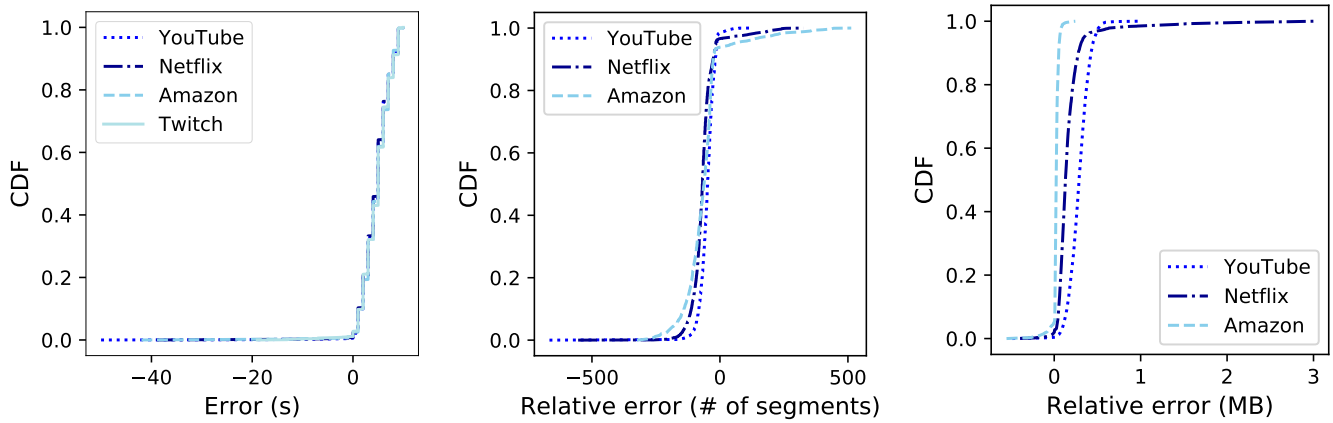


Figure 3: *Absolute error in segment start time estimation.*

(a) *Number of segments.*

(b) *Segment sizes.*

Figure 4: *Relative error in segment detection.*[1]

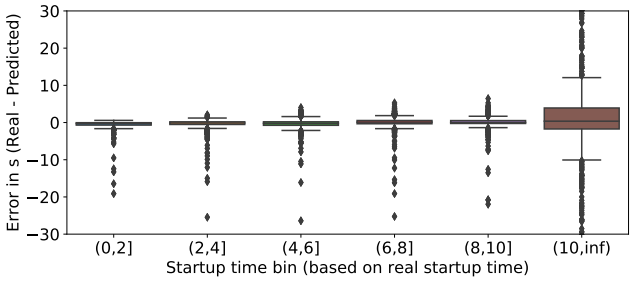
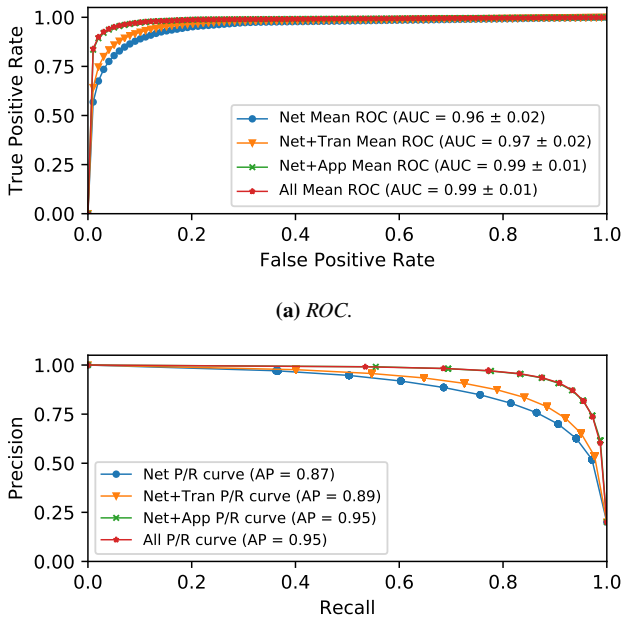


Figure 5: *Relative error in startup delay inference (by levels) using Net+App features cut at [-30, 30] on the y axis (For all four video services).*



(a) *ROC.*

(b) *Precision-recall.*

Figure 6: *Resolution inference using different feature sets (For all four video services).*

Validation. In contrast to models in previous work which make discrete predictions about startup delay (*i.e.*, "good" vs."bad", based on arbitrary thresholds), Network Microscope predicts startup delay as a *continuous value* (*i.e.*, seconds). This granularity of prediction also allows us to study the magnitude of errors in our model prediction. Figure 5 shows the distribution of the relative errors for inferred startup delays split into two-second bins. The model performs particularly well for startup delays of less than ten seconds with errors mostly within one second.

In some cases, startup delay prediction has larger absolute error. These errors result from two reasons: (1) 91.2% of all sessions in our data have startup delays of less than ten seconds, meaning that the number of samples in the greater than ten-second range is small; and (2) as startup delays increase beyond ten seconds, precision decreases because the range of possible values is larger. Previous work has shown that startup delays above ten seconds can cause a rate of abandonment above 50% [26]. In a practical sense, detecting that the startup delay exceeds such a large value should be sufficient for an operator to identify a video session as poor quality.

To validate the model for resolution inference, we divided each video session into ten-second time intervals and inferred the resolution for each interval. Network Microscope uses a random forest multi-class classifier for each set of features and service. Each class of the classifier corresponds to one of the following resolution values: 240p, 360p, 480p, 720p, and 1080p. Figure 6a and (Figure 6b) show the receiver operating characteristic and precision-recall curves for the models, respectively. The models trained with application layer features consistently achieve the best performance with both precision

Speed [mbps]	Homes	Devices	# Video Sessions			
			Netflix	YouTube	Amazon	Twitch
(0,50]	20	329	21,774	65,448	2,612	2,584
(50,100]	23	288	11,788	50,178	3,273	3,345
(100,500]	19	277	13,201	38,691	2,030	197
(500,1000]	4	38	523	442	86	1
Total	66	932	47,286	154,759	8,001	6,127

Table 2: *Number of video sessions per speed tier.*

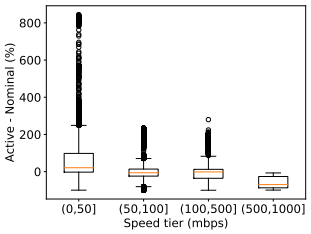


Figure 7: *Speed tiers vs estimated capacity over time for the homes in the deployment.*

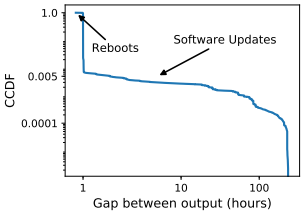


Figure 8: *Deployment uptime.*

and recall reaching 91% for a 4% false positive rate. Any model not including application features reduces precision by at least 8%, while doubling the false-positive rate.

Overall, these results demonstrate the utility of co-designing collection and inference: A model that is based on the common features outlined in Table 1 (Section 3) is still quite accurate "out of the box", yet the additional extensibility and programmability of Network Microscope makes it possible to tailor collection and inference to collect additional service-specific lightweight features for specific applications. In Section 6, we perform a preliminary exploration of some additional use cases.

5 One-Year Field Deployment

To evaluate Network Microscope in real networks, we developed and deployed a prototype implementation of Network Microscope and used it to study the quality of streaming video as part of an ongoing collaboration with a team of investigative journalists from a major news organization, with whom we are working to understand how application performance in home networks relates to the speed tiers that ISPs sell to subscribers.

5.1 Deployment

We have deployed Network Microscope using the Raspberry Pi [35] and Odroid [32] platforms, and we currently manage a deployment in the homes of volunteer users. We began deploying devices in late 2017; the initial versions of the software were experimental and changed considerably. In this paper, we only analyze data from the production version of Network Microscope deployed on January 28, 2018. As of March 2019, we have installed approximately 60 devices in homes in the United States; an additional six devices are deployed in France. Downstream throughputs from these networks range from 18 Mbps to 1 Gbps.

Thus far, we have recorded a total of 216,173 video sessions from four of the major video service providers: Netflix, YouTube, Amazon, and Twitch. Table 2 presents an overview of the deployment, including the total number of video sessions collected for each video service provider grouped by the homes' nominal Internet speed. We have also integrated Bismark [41] measurement suite with Network Microscope to periodically measure throughput to the last mile. Figure 7 shows the collected measurements compared against the nominal speed tiers that the users subscribe to (as collected during our recruitment of study participants).

Figure 8 shows a CCDF reflecting device uptimes in the deployment, as measured by the time gaps between output files (we expect an output file every hour). Most output files have a gap of one hour, meaning that the system was up and functioning during these time periods. The distribution contains two distinct tails. First, a small number of gaps are less than one hour, which we suspect are due to system reboots. The other tail includes gaps longer than one hour, which we believe are attributable to software upgrades, the user's local network configuration, or other hardware problems. In some of these cases, we had to go to the participant's home to bring the device back online.

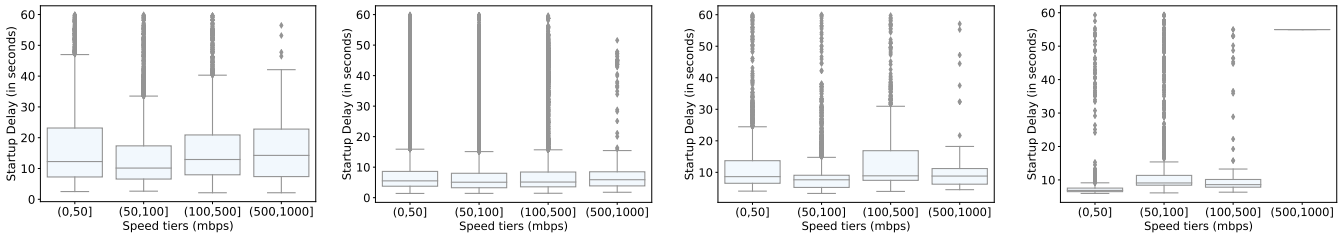
5.2 Streaming Video Quality Analysis

We ran the video inference models from Section 4.3 on all video sessions collected throughout the lifetime of the deployment. In this section, we study two video quality metrics: startup delay and resolution. We filter out all video sessions with startup delay larger than 60 seconds, as these sessions most likely correspond to instances when the video never played. The total number of discarded sessions is 4.7% of the entire dataset.

5.2.1 Startup Delay

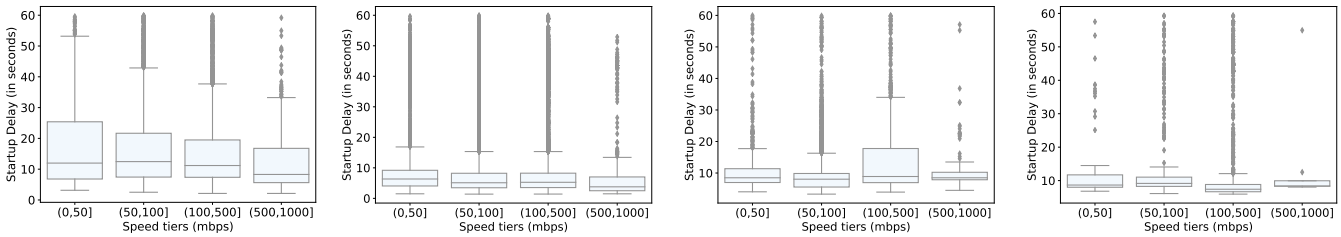
We use Network Microscope's video quality inference module to infer the startup delay for each video session in the dataset. We ask the question: how does startup delay relates to access capacity? Answering this question allows us to understand the benefits of paying for higher access capacity for getting improved video streaming. We use two metrics of "capacity": the nominal download speed that users reported for each home, and the 95th percentile of active throughput measurements that Network Microscope collected. Figure 9 presents the box plot of startup delay versus nominal speeds, whereas Figure 10 presents startup delay versus the 95th percentile of throughput measurements. The number of samples for the highest tier for both Amazon and Twitch is too small to draw conclusions (see Table 2), so we focus on the first three tiers for these services.

Figure 9 exhibits a mostly stable trend for the median startup delay across nominal speed for all services. For example, YouTube consistently achieves a median of 5 seconds startup delay across all tiers and Amazon a constant median



(a) *Netflix.* **(b)** *YouTube.* **(c)** *Amazon.* **(d)** *Twitch.*

Figure 9: *Startup Delay Inference vs. Nominal Speed Tier.*



(a) *Netflix.* **(b)** *YouTube.* **(c)** *Amazon.* **(d)** *Twitch.*

Figure 10: *Startup Delay Inference vs. Active Throughput Measurements (95th Percentile).*

startup delay of 8 seconds. Netflix and Twitch's startup delay differs by at most $\pm$ 2 seconds across tiers, but these plots exhibit no trend of decreasing startup delay as nominal speeds increases as one would expect. There are several possible explanations for this result. One possibility is that actual speeds can vary considerably over time due to variations in available capacity along end-to-end paths due (for example) to diurnal traffic demands.

Figure 10 shows that startup delay decreases as measured throughput increases for Netflix, YouTube, and Twitch. For Netflix, the difference in startup delay between the highest and lowest tier is four seconds, this difference is three seconds for YouTube. This figure also highlights the difference in startup delay across services, given similar network conditions. These difference reflect the fact that different services select different design tradeoffs that best fit their business model and content. For example, Netflix has the highest startup delay of all the services. Because Netflix content is mostly movies and long-form television shows—which are relatively long—having higher startup delays is acceptable. As we see in the next section, our inference reflects the expected trends that Netflix trades off higher startup delay to achieve higher resolution video streams.

5.2.2 Resolution

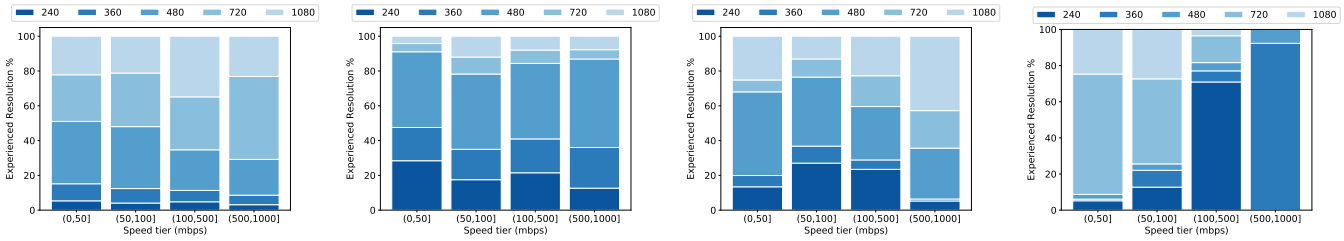
Next, we use Network Microscope's video quality inference module to infer the resolution in each ten-second time bin of a video session. As with the analysis of startup delay, we study the correlation between resolution and measured access capacity. For this analysis, we omit the first 60 seconds of video sessions, since most Adaptive Bitrate (ABR) algorithms begin with a low resolution to slowly ramp up as they obtain a better estimate of the end-to-end throughput. Network Microscope's video quality inference module outputs one of five resolution classes: 240, 360, 480, 720, or 1080.

Figure 11 presents resolution versus the speed tier, whereas Figure 12 presents resolution versus the 95th percentile of active throughput measurements. As with startup delay, we observe that resolution increases with measured throughput (Figure 12); the trend is less clear for speed tiers (Figure 11). Recall that the number of samples in the highest speed bar for Amazon and Twitch is small; focusing on the other three tiers shows a clear trend of increased resolution with measured speeds. For example, Netflix and YouTube in the highest speed tier achieve about 40% more 1080p than in the lowest speed tier. In general, YouTube streams at lower resolution for the same network conditions than other services.

Video resolution depends on factors other than the network conditions. First, some videos are only available in SD and thus will stream at 480p regardless of network conditions. Second, services tailor to the device type. We can identify the device type for some of the devices in the deployment based on the MAC address. Of 1,290,130 YouTube sessions for which we have device types, 616,042 are to smartphones, indicating that device type may be a confounder.

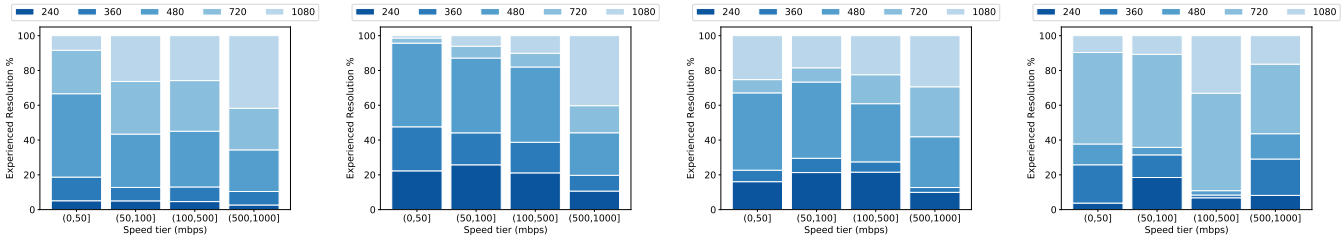
6 Other Use Cases in Access Networks

In this section, we describe how we extended Network Microscope to support additional proof-of-concept applications.



(a) *Netflix.* **(b)** *YouTube.* **(c)** *Amazon.* **(d)** *Twitch.*

Figure 11: *Resolution vs Nominal Speed Tier*



(a) *Netflix.* **(b)** *YouTube.* **(c)** *Amazon.* **(d)** *Twitch.*

Figure 12: *Resolution vs. Active Throughput Measurements (95th Percentile).*

6.1 Security: IoT DDoS Detection

Doshi *et al.* performed real-time detection of Distributed Deninal of Service (DDoS) traffic from Internet of Things (IoT) devices using five machine learning algorithms: k-nearest neighbors "KDTree" algorithm, support vector machine with linear kernel, decision tree, random forest, and neural network [13]. This work shows that traffic features such as packet size, inter-packet interval, and protocol are important for detecting whether a smart home IoT device is participating in a DDoS attack. The work finds that: (1) normal packet sizes vary between 100 and 1,200 bytes, while over 90% of attack packets are under 100 bytes; (2) normal IoT traffic has limited bursts while attack traffic has close to zero inter-packet intervals; and (3) UDP traffic dominates TCP traffic by almost a factor of three in normal traffic due to UDP-based video streaming, whereas this ratio is less during attack.

We integrated the machine learning-based detection module into Network Microscope on the Odroid platform (running Armbian Linux) in only a few hours. First, we extended Network Microscope's packet processing pipeline (Section 3.1) to monitor the additional features needed by the model. We added approximately 100 lines of Go code to configure Network Microscope to extract packet sizes, inter-packet intervals, and protocols of the first ten packets from each source device for every five seconds; and about 60 lines of Python to the original detection code for continuously monitoring and parsing features coming from Network Microscope.

In our lab environment, we set up an Odroid platform and installed the modified Network Microscope system that extracts these features. We then requested and received two models from the original authors: random forest and neutral network. We trained both models with the training set included in their source code. We imported both models into our Odroid platform along with the modified Python script that runs the inference with a given model. We used GoldenEye [19] to simulate an attack and BigFlows from the tcpreplay website [44] to simulate normal traffic.

In our test, the random forest model showed 37% true positive rate and 33% true negative rate, while the neural network model showed a 82% true positive rate and 80% true negative rate. We believe these inaccuracies result because the models are trained with a small training set (included in the source code package); the inference quality would improve when trained with a larger and more representative dataset. The particular implementation of the neural net-based inference from the previous work was also somewhat slow: inference in Odroid takes around 100 seconds per flow. We believe that this performance is likely due to the inefficiency of the model. With a system like Network Microscope, we expect that this system and others like it could be tested and targeted for real deployments, whereas they might have previously stopped with offline trace analysis.

6.2 Privacy: Monitoring Ads and Trackers

To demonstrate the extensibility of the Network Microscope's domain signature module, we installed domain signatures (Section 3.1) for domains related to advertisements and trackers. Lists of such domains are widely available for use by network administrators at sites such as WebShrinker [46]. To extent Network Microscope to monitor these categories of

traffic, we added two lines containing the domain lists to the Network Microscope configuration.

The resulting measurements have allowed us to quantify the ad and tracker traffic in our deployment. We report some high-level statistics in this paper as a proof of concept; more detailed analysis is possible, naturally. Given our deployment, we observe that across all users, ad and tracker traffic makes up an average of 0.46% of downstream bytes, 1.28% of upstream bytes, and 7.43% of all flows. In one case, 22.5% of a single user's flows were related to ads and trackers. This simple example demonstrates that, with minimal effort, Network Microscope can provide in-depth usage information and help potentially identify a user or device in a network that may benefit for ad-blocking software.

7 Related Work

Conventional passive measurement. Passive traffic measurement is a well-established area. NetFlow [4] and IPFIX [5] capture summaries of traffic flows, whereas libpcap [43] can capture at packet-level granularity. Previous work in traffic measurement, from research platforms [17,18,29,45,50] to commercial systems [10,24] generally involves collecting some subset of these measurements at a central location. Yet, flow statistics are often too coarse to facilitate inference, and packet-level measurements are too high-volume to collect across potentially tens of millions of subscribers; packet-level measurements are also increasingly restricted by new privacy regulations, creating the need for the type of local collection and inference that Network Microscope enables.

Streaming analytics platforms. Network intrusion detection systems such as Bro [34] and Snort [37] rely on regular expressions to identify the subset of packets to inspect. Network Microscope uses a similar approach to identify services (using the DNS cache). Many systems allow operators to express queries on streaming traffic data: Gigascope [8], Chimera [2], NetQRE [49], Marple [31], and Sonata [23]. These systems do not focus on inference of higher-level properties such as video quality; they are primarily designed to collect low-level statistics on a backbone router or switch, or a programmable data-center switch, which operate at much higher speeds than where Network Microscope operates. As such, they typically support a more limited set of queries that are constrained by the hardware they are designed to support. Some of these query languages could ultimately have Network Microscope as a target platform. For example, while Sonata aims to compile queries across stream processors and P4 switches, it could just as easily compile queries to heterogeneous targets including Network Microscope deployment.

Streaming algorithms ("Sketches"). Sketches are custom data structures that support specific counting and aggregation tasks on traffic streams [27,48]. UnivMon [28] and Elastic Sketch [47] achieve high fidelity across a broader set of inference tasks. This line of work is complementary: for example, implementing these types of sketches on Network Microscope could facilitate inference tasks based on more complex features.

Traffic measurements in access networks. Past work has characterized traffic in residential access networks by collecting traffic statistics in network aggregation points [29,45]. These characterizations focused on more coarse-grained analysis of traffic properties, such as the amount of traffic that each application consumes. Network Microscope facilitates real-time analysis of performance and security at the granularity of individual flows and devices. Grover *et al.* [21] and Chetty *et al.* [3] used passive monitoring to analyze traffic in home networks using passive monitoring in a home network. Both of these studies required years of custom development on an embedded device (*i.e.*, a home router access point), Network Microscope can simplifies these types of measurements.

8 Conclusion

Managing residential access networks requires collecting measurements at scale, inside home networks whenever possible. Deploying measurement tools in this fashion introduces new challenges for both scalability (due to the number of vantage points and the resources available at the monitoring points) and privacy (due to what can be collected and stored about individuals without their consent). These challenges introduce interesting new design tradeoffs between capturing data at a granularity that can enable inference of higher-level network properties (*e.g.*, application quality of experience, security incidents) and the ability to collect and process the traffic data at scale, in real-time.

Existing data collection technologies are either too fine-grained (in the case of packet traces) or too coarse (in the case of IPFIX) to address the problems in access networks. Additionally, many of the existing applicable inference techniques for both application performance and security were not designed with deployment at the edge in mind. To address these shortcomings, we have explored a new approach, where the data collection system is co-designed with the inference algorithms, ensuring that the traffic statistics that are recorded are both lightweight enough to be collected in deployment *and* sufficient for accurate inference of higher-level properties. To demonstrate the viability of this approach, we designed, implemented, and deployed Network Microscope, a network measurement platform for residential access networks and shown how this system can provide new insights about streaming video application performance for both Internet service providers and platform providers. Although in this paper we have focused on inferring streaming video due to its dominant role in access networks, we have also begun to explore how Network Microscope might be applied a larger class of inference problems and deployed at different locations in the access network, including possibly the cable modem termination system (CMTS) or virtual service gateway (VSG).

References

[1] Anonymized for blind review. PC chairs have an anonymized copy.

[2] Kevin Borders, Jonathan Springer, and Matthew Burnside. Chimera: A declarative language for streaming network traffic analysis. In *Presented as part of the 21st USENIX Security Symposium (USENIX Security 12)*, pages 365–379, Bellevue, WA, 2012. USENIX.

[3] Marshini Chetty, Hyojoon Kim, Srikanth Sundaresan, Sam Burnett, Nick Feamster, and W. Keith Edwards. uCap: An internet data management tool for the home. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems*, CHI '15, pages 3093–3102, New York, NY, USA, 2015. ACM.

[4] Benoit Claise. Cisco systems netflow services export version 9. Technical report, 2004.

[5] Benoit Claise, Brian Trammell, and Paul Aitken. Specification of the ip flow information export (ipfix) protocol for the exchange of flow information, 2013. RFC 7011.

[6] Federal Communication Commission. Measuring Broadband America - Open Methodology. `https://www.fcc.gov/general/measuring-broadband-america-open-methodology`, 2019.

[7] Corelight. `https://corelight.com/`, 2019.

[8] Chuck Cranor, Theodore Johnson, Oliver Spataschek, and Vladislav Shkapenyuk. Gigascope: a stream database for network applications. In *Proceedings of the 2003 ACM SIGMOD international conference on Management of data*, pages 647–651. ACM, 2003.

[9] A. Cui and S. J. Stolfo. A quantitative analysis of the insecurity of embedded network devices: results of a wide-area scan. In *Proceedings of the 26th Annual Computer Security Applications Conference*, 2010.

[10] Deepfield. `https://deepfield.com/`, 2019.

[11] Luca Deri et al. Improving passive packet capture: Beyond device polling. In *Proceedings of SANE*, volume 2004, pages 85–93. Amsterdam, Netherlands, 2004.

[12] Giorgos Dimopoulos, Ilias Leontiadis, Pere Barlet-Ros, and Konstantina Papagiannaki. Measuring video QoE from encrypted traffic. In *Proceedings of the 2016 Internet Measurement Conference*, pages 513–526. ACM, 2016.

[13] Rohan Doshi, Noah Apthorpe, and Nick Feamster. Machine learning ddos detection for consumer internet of things devices. In *2018 IEEE Security and Privacy Workshops (SPW)*, pages 29–35. IEEE, 2018.

[14] DPDK, Data Plane Development Kit. `https://www.dpdk.org/`, 2018.

[15] Keith Dyer. How encryption threatens mobile operators, and what they can do about it. `https://goo.gl/Q19u9P`, January 2015.

[16] Cristian Estan and George Varghese. New directions in traffic measurement and accounting. In *Proceedings of the 2002 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, SIGCOMM '02, pages 323–336, New York, NY, USA, 2002. ACM.

[17] Anja Feldmann, Albert Greenberg, Carsten Lund, Nick Reingold, Jennifer Rexford, and Fred True. Deriving traffic demands for operational ip networks: Methodology and experience. *IEEE/ACM Trans. Netw.*, 9(3):265–280, June 2001.

[18] Chuck Fraleigh, Sue Moon, Bryan Lyles, Chase Cotton, Mujahid Khan, Deb Moll, Rob Rockell, Ted Seely, and S Christophe Diot. Packet-level traffic measurements from the sprint ip backbone. *IEEE network*, 17(6):6–16, 2003.

[19] Goldeneye code repository. [Online]. `https://github.com/jseidl/GoldenEye`, 2019.

[20] Go packet library. `https://godoc.org/github.com/google/gopacket`.

[21] Sarthak Grover, Mi Seon Park, Srikanth Sundaresan, Sam Burnett, Hyojoon Kim, Bharath Ravi, and Nick Feamster. Peeking behind the nat: An empirical study of home networks. In *Proceedings of the 2013 Conference on Internet Measurement Conference*, IMC '13, pages 377–390, New York, NY, USA, 2013. ACM.

[22] Network Management of Encrypted Traffic: Version 1.0. `https://www.gsma.com/newsroom/wp-content/uploads/WWG-04-v1-0.pdf`, February 2015. GSM Association.

[23] Arpit Gupta, Rob Harrison, Marco Canini, Nick Feamster, Jennifer Rexford, and Walter Willinger. Sonata: Query-driven streaming network telemetry. In *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*, SIGCOMM '18, pages 357–371, New York, NY, USA, 2018. ACM.

[24] Kentik. `https://kentik.com/`, 2019.

[25] Vengatanathan Krishnamoorthi, Niklas Carlsson, Emir Halepovic, and Eric Petajan. BUFFEST: Predicting buffer conditions and real-time requirements of HTTP (s) adaptive streaming clients. In *Proceedings of the 8th ACM on Multimedia Systems Conference*, pages 76–87. ACM, 2017.

[26] S Shunmuga Krishnan and Ramesh K Sitaraman. Video stream quality impacts viewer behavior: inferring causality using quasi-experimental designs. *IEEE/ACM Transactions on Networking*, 21(6):2001–2014, 2013.

[27] Abhishek Kumar, Minho Sung, Jun Jim Xu, and Jia Wang. Data streaming algorithms for efficient and accurate estimation of flow size distribution. In *ACM SIGMETRICS Performance Evaluation Review*, volume 32, pages 177–188. ACM, 2004.

[28] Zaoxing Liu, Antonis Manousis, Gregory Vorsanger, Vyas Sekar, and Vladimir Braverman. One sketch to rule them all: Rethinking network flow monitoring with univmon. In *Proceedings of the 2016 ACM SIGCOMM Conference*, pages 101–114. ACM, 2016.

[29] Gregor Maier, Anja Feldmann, Vern Paxson, and Mark Allman. On dominant characteristics of residential broadband internet traffic. In *Proceedings of the 9th ACM SIGCOMM Conference on Internet Measurement*, pages 90–102. ACM, 2009.

[30] M. Hammad Mazhar and Zubair Shafiq. Real-time video quality of experience monitoring for HTTPS and QUIC. In *INFOCOM, 2018 Proceedings IEEE*. IEEE, 2018.

[31] Srinivas Narayana, Anirudh Sivaraman, Vikram Nathan, Prateesh Goyal, Venkat Arun, Mohammad Alizadeh, Vimalkumar Jeyakumar, and Changhoon Kim. Language-directed hardware design for network performance monitoring. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, pages 85–98. ACM, 2017.

[32] ODROID-C2 Wiki. `https://wiki.odroid.com/odroid-c2/odroid-c2`, 2018.

[33] Irena Orsolic, Dario Pevec, Mirko Suznjevic, and Lea Skorin-Kapov. A machine learning approach to classifying YouTube QoE based on encrypted network traffic. *Multimedia tools and applications*, 76(21):22267–22301, 2017.

[34] Vern Paxson. Bro: a system for detecting network intruders in real-time. *Computer networks*, 31(23-24):2435–2463, 1999.

[35] Raspberry Pi 3 Model B. `https://www.raspberrypi.org/products/raspberry-pi-3-model-b/`, 2018.

[36] QUIC: A UDP-based multiplexed and secure transport. `https://www.ietf.org/id/draft-ietf-quic-transport-09.txt`, 2018.

[37] Martin Roesch et al. Snort: Lightweight intrusion detection for networks. In *Lisa*, volume 99, pages 229–238, 1999.

[38] Haoyu Song, Tianran Zhou, Zhenbin Li, Giuseppe Fioccola, Zhenqiang Li, Pedro Martinez-Julia, Laurent Ciavaglia, and Aijun Wang. Toward a network telemetry framework. Internet-Draft draft-song-ntf-02, IETF Secretariat, July 2018. `http://www.ietf.org/internet-drafts/draft-song-ntf-02.txt`.

[39] Speedtest. `https://www.speedtest.net/`, 2019.

[40] Ookla Speedtest. How does the test itself work? How is the result calculated? `https://support.speedtest.net/hc/en-us/articles/203845400-How-does-the-test-itself-work-How-is-the-result-calculated-`, 2012.

[41] Srikanth Sundaresan, Sam Burnett, Nick Feamster, and Walter De Donato. Bismark: A testbed for deploying measurements and applications in broadband access networks. In *USENIX Annual Technical Conference*, pages 383–394, 2014.

[42] Srikanth Sundaresan, Walter De Donato, Nick Feamster, Renata Teixeira, Sam Crawford, and Antonio Pescapè. Broadband internet performance: a view from the gateway. In *ACM SIGCOMM computer communication review*, volume 41, pages 134–145. ACM, 2011.

[43] tcpdump and libpcap. `https://www.tcpdump.org/`.

[44] Sample captures on tcpreplay website [Online]. `http://tcpreplay.appneta.com/wiki/captures.html`, 2019.

[45] Martino Trevisan, Danilo Giordano, Idilio Drago, Marco Mellia, and Maurizio Munafo. Five years at the edge: watching internet from the isp network. In *Proceedings of the 14th International Conference on emerging Networking EXperiments and Technologies*, pages 1–12. ACM, 2018.

[46] WebShrinker Website category and domain API. `https://docs.webshrinker.com`, 2018.

[47] Tong Yang, Jie Jiang, Peng Liu, Qun Huang, Junzhi Gong, Yang Zhou, Rui Miao, Xiaoming Li, and Steve Uhlig. Elastic sketch: Adaptive and fast network-wide measurements. In *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*, pages 561–575. ACM, 2018.

[48] Minlan Yu, Lavanya Jose, and Rui Miao. Software defined traffic measurement with opensketch. In *Presented as part of the 10th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 13)*, pages 29–42, 2013.

[49] Yifei Yuan, Dong Lin, Ankit Mishra, Sajal Marwaha, Rajeev Alur, and Boon Thau Loo. Quantitative network monitoring with NetQRE. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, SIGCOMM '17, pages 99–112, New York, NY, USA, 2017. ACM.

[50] Yibo Zhu, Nanxi Kang, Jiaxin Cao, Albert Greenberg, Guohan Lu, Ratul Mahajan, Dave Maltz, Lihua Yuan, Ming Zhang, Ben Y Zhao, et al. Packet-level telemetry in large datacenter networks. In *ACM SIGCOMM Computer Communication Review*, volume 45, pages 479–491. ACM, 2015.