Measuring Service Congestion, Sharing, and Pacing in a Residential Access Network

Shuyue Yu
sy3011@columbia.edu
Columbia University
New York, United States

Bruce Spang
bspang@netflix.com
Netflix
Los Gatos, United States

Carson Garland
ctg2137@columbia.edu
Columbia University
New York, United States

Kevin Chen
kvnc@netflix.com
YouTube / Netflix
Cambridge, United States

Ramki Gummadi
gsrk@google.com
Google DeepMind
Mountain View, United States

Gil Zussman
gil@ee.columbia.edu
Columbia University
New York, United States

Ethan Katz-Bassett
ethan@ee.columbia.edu
Columbia University / Google
New York, United States

Te-Yuan Huang
thuang@netflix.com
Netflix
Los Gatos, United States

Renata Teixeira
rteixeira@netflix.com
Netflix
New York, United States

Abstract

Residential access networks today often appear well provisioned on paper, with high access speeds, modern Wi-Fi, and content served from nearby caches. We investigate how such networks actually behave using packet-level traces from 33 residential buildings owned and managed by Columbia University, hosting over 1,500 units. Our methodology infers TCP loss and access RTTs to detect congestion and estimates per-packet in-flight intervals to identify when packets from different services overlap on shared buffers. Even with low average utilization, we find that short traffic bursts can cause substantial loss and queueing delay, and that services often share capacity primarily with their own traffic. Pacing mitigates this burst-induced congestion by adding delay between packets and has been studied by YouTube and Netflix separately, but the vantage point of a single service cannot observe the effects of pacing on other services. We evaluate pacing through a real-world experiment in which YouTube and Netflix alternately enable and disable pacing for users in this network. Under pacing, each service roughly halves its own median loss rate and reduces median access queueing delay by about 30–40%, while sustaining similar traffic volumes. For other services whose packets directly overlap paced video, median loss and queueing delay also decrease, even though pacing slightly increases the probability of such overlaps. Our results provide the first in-the-wild evidence that pacing is friendly in such a well-provisioned network.

CCS Concepts

• **Networks** → **Network measurement**; *Network performance analysis*; Network experimentation.

IMC '26, Karlsruhe, Germany

ACM ISBN 979-8-4007-2327-8/2026/10
https://doi.org/10.1145/3777912.3839796

Keywords

Residential Access Networks, Congestion, Network Resource Sharing, Pacing

ACM Reference Format:
Shuyue Yu, Bruce Spang, Carson Garland, Kevin Chen, Ramki Gummadi, Gil Zussman, Ethan Katz-Bassett, Te-Yuan Huang, and Renata Teixeira. 2026. Measuring Service Congestion, Sharing, and Pacing in a Residential Access Network. In *Proceedings of the 2026 ACM Internet Measurement Conference (IMC '26), October 12–16, 2026, Karlsruhe, Germany*. ACM, New York, NY, USA, 20 pages. https://doi.org/10.1145/3777912.3839796

1 Introduction

In many markets, residential Internet connectivity is increasingly viewed as "good enough." Even a decade ago, measurements showed that individual homes often used far less than their provisioned access capacity [38, 46]. Since then, both access link rates and in-home Wi-Fi have increased significantly [7, 42], and content providers now serve much of their traffic from caches and servers deployed close to users [36]. Taken together, these trends suggest that, on average, many residential access networks now offer ample capacity for typical workloads, and one might expect congestion to be largely a thing of the past for such networks.

To investigate what congestion looks like in practice, we study one such well-provisioned network in detail. We collect traces of all packets sent to and received from 33 off-campus residential buildings owned and managed by Columbia University, with over 1,500 units housing graduate students, faculty, and their families (§3.1). This collection extends the residential access dataset introduced in our prior work, which was based on 4 hours of traffic per day from approximately 1,000 units [75]. This paper introduces a more comprehensive view, with 24/7 collection across more than 1,500 units. Following the same data-sharing policy as our prior work, the dataset used in this study—and 24/7 data from the 1,500 units going forward in perpetuity—is also available upon request through the dataset website [1]. The traces are collected at a shared aggregation point, and give us visibility into many homes and many services at once—a vantage that prior residential studies have lacked. From

these traces we infer per-packet TCP loss, access RTTs, and overlapping in-flight intervals as proxies for congestion and queue sharing in the access network (§3).

With this in hand, the picture turns out to be more nuanced than aggregate capacity suggests. Although hourly building utilization is below 13% for 95% of building-hour pairs, traffic can still experience considerable packet loss and queueing delay: hourly TCP loss rates exceed 2% in 8% of hours, and 11% of packets experience more than 50 ms of access queueing delay (§4). An explanation is that modern traffic is bursty. Video streaming often downloads chunks in short bursts separated by idle periods, while web transfers are dominated by short flows that send their data quickly and complete before reaching a steady sending rate. Multiple devices and applications in a home can generate these bursts at the same time. Transport behavior can further amplify variability: connection startup, changes in congestion window, and imperfect pacing all create transient bursts, even when congestion control aims to be ACK-clocked or explicitly paced. When such bursts exceed network capacity—whether on a home Wi-Fi link or a shared access or aggregation link—they can lead to queueing delay or packet loss.

Prior work, especially in data centers, has identified burstiness as a central challenge for congestion management and load balancing [2, 4–6, 34, 35]. In residential networks, however, we still lack a clear picture of how congestion episodes are shared across flows, services, and homes. In our study, we find that congestion in this network is overwhelmingly local and self-induced. Congestion is driven more by sharing within a single residential unit than across units in a building, and the great majority of a packet's in-flight neighbors come from the same connection to the same service, rather than from competing connections or services. These patterns suggest that interventions targeting individual services can have meaningful impact (§5).

Recently, Netflix and YouTube (among the largest sources of traffic on the Internet [58] and in this network) have been exploring *pacing* to reduce the congestion caused by burstiness. Pacing intentionally adds delay between packets. It is typically implemented at the transport layer, where it is agnostic to application needs [18, 31]. Netflix and YouTube have instead explored pacing traffic based on application demands, at a rate closer to the video bitrate. This reduces throughput well below available bandwidth while still maintaining video quality [33, 63]. YouTube has paced all its video traffic as of May 2025, while Netflix deployed pacing to nearly all models of TVs, STBs, and game consoles starting in November 2025.

With large traffic sources exploring and adopting pacing, it is crucial to understand how it performs in the real world. Yet all prior real-world pacing experiments share a key limitation: the vantage point of a single service. A service can measure how pacing affects its own traffic, but not how pacing affects *other* services sharing the same access network. For services whose packets share queues with video, pacing might make a service more "friendly" by reducing large bursts that cause congestion. But if the on-period of video traffic is normally very short, smoothing it out could *create* overlaps with neighboring services that would not otherwise have occurred. Without visibility into an actual access network and the traffic from multiple services it is delivering, it is difficult to reason about the network-wide impact of pacing.

We develop passive methods to infer congestion in the Columbia network, and we combine them with a controlled experiment to evaluate the impact of pacing (§6). During the experiment, YouTube and Netflix alternately enabled and disabled pacing to conduct a month-long switchback experiment. The experiment yields all four combinations of {YouTube paced, unpaced} × {Netflix paced, unpaced}, each sustained for a full week. This design allows us to study how pacing affects both the paced service itself and other traffic sharing the same network. This is a unique study: while providers regularly experiment with their own traffic using A/B tests, to our knowledge this is the first large-scale study that looks at the impact of any rate-control treatment from a major traffic source (or two) on other traffic sharing the same network.

Under pacing, each video service substantially reduces congestion on its own traffic—median loss rates drop by roughly half for both Netflix and YouTube, and median queueing delays fall by about 30–40%. Total traffic volume remains similar across paced and unpaced weeks, indicating that these congestion reductions are achieved without reducing the volume or bitrate of video delivered (§6.3). For neighboring services, pacing slightly increases the probability of overlapping with the paced video traffic, but it substantially reduces the intensity of those overlaps. Congestion metrics for other services are stable overall and improve significantly when they directly overlap with paced video traffic. While most packets do not overlap with video traffic, pacing provides meaningful benefits to those that do (median loss falls 13–29% and median queueing delay 16–36%) (§6.4).

In this network, our results provide concrete, in-the-wild evidence that pacing largely improves performance for the pacer at no observable cost to other services while providing improvement to neighboring services during direct overlaps. Our results suggest that other services whose users gain little from sending at unconstrained rates (*e.g.,* video traffic, software updates, cloud sync, and other bulk transfers) have a strong case for adopting pacing on the same grounds. In other well-provisioned networks, we would expect traffic to be similarly bursty and pacing to provide similar benefits for both the pacer and its neighbors.

More broadly, we see our approach of combining controlled modification of service-side configuration with packet traces from access networks as a template other studies could use when evaluating congestion control and traffic-shaping mechanisms. Prior work on algorithms such as BBR [18, 40] and service-side treatments evaluated with A/B tests [62] has shown that behavior at scale can differ markedly from what we expect from controlled settings, especially in terms of fairness and interactions with other traffic. We hope our work encourages similar studies that measure what these interactions look like in other real networks.

2 Challenges

We aim to answer: (i) to what extent congestion exists in a well-provisioned access network, (ii) how traffic shares network resources in the network, and (iii) how enabling pacing for a service changes congestion for that service and for other services that share bottlenecks with it.

To answer these questions in a fully controlled way, one would ideally have three capabilities. First, to study whether and where

congestion occurs and how it is shared, we would like direct access to all clients, servers, and buffers along the path to measure loss, queueing delay, and buffer occupancy precisely. Second, to isolate the impact of pacing on a given service, we would like the ability to toggle pacing for arbitrary services to form clear treatment and control groups. Third, to attribute differences between those groups to pacing rather than to changes in demand, we would like to expose them to identical traffic patterns.

In practice, the decentralized nature of the Internet precludes such unified control. No single operator can observe all clients, servers, and buffers, and, while individual services can toggle their own pacing, they cannot control or even see other services' traffic. Moreover, simply replaying traces under different pacing configurations does not faithfully capture the feedback between one service's pacing choices and the traffic of others. We therefore turn to a more pragmatic design that combines cooperation from major services, a rare multi-home vantage point, and inference techniques that recover congestion and sharing from packet traces. The remainder of this section outlines the main practical challenges this design must address.

Loss and latency are not directly inferable for all traffic from a single passive vantage point. It is challenging to infer loss and RTTs for non-TCP traffic, as sequence or acknowledgment numbers may be unavailable or encrypted depending on the protocol. To facilitate our pacing experiment, YouTube shifted Columbia users from QUIC to TCP sessions during the study period. Consequently, UDP accounts for 16% of traffic in the dataset we study. While we do not infer loss for the remaining UDP traffic, we estimate access RTTs, queueing delays, and in-flight intervals for UDP packets based on adjacent TCP packets for the same residential unit (see §3 for details).

Direct measurements of buffer occupancy are difficult. Router buffer occupancy is hard to observe at fine time scales. Standard monitoring tools such as SNMP have low resolution (Cisco recommends polling at 60 seconds to limit CPU overhead [20]), and high-resolution tools like P4-based monitoring are not available on commodity devices found in most access networks. Our packet traces record when packets arrive at the data collection point, but not instantaneous queue states or which packets share a queue at any given time. Simply counting packets within a fixed time interval is also insufficient, as packets can remain in buffers for varying durations, especially under congestion. To address this, we infer per-packet in-flight intervals from TCP RTTs and study how these intervals overlap, using overlap as a proxy for packets sharing limited network resources (§3).

Complex equipment and opaque home routers limit topology visibility. Modern access networks often use switches with complex queueing structures and behavior, whose details are typically not exposed to researchers. The Columbia residential access network uses chassis switches with shared packet-buffer memory, and residents also bring their own home routers, which vary in capacity, buffer sizes, and configuration and lie outside the operator's direct control. Rather than attempting to reconstruct exact buffer occupancy at each device under these constraints, we rely on packet dynamics and overlaps of in-flight intervals to infer when packets are likely to share queues (§3).

Real traffic patterns are essential, but trace replay can misrepresent pacing effects. Whereas concepts such as the friendliness of congestion control algorithms can be evaluated in controlled lab settings, understanding the nature of actual congestion and sharing requires visibility into the traffic of a real network, as it depends intimately on the behavior of users and the applications they use. Further, a static trace of such behavior is not enough to understand the impact of pacing (or other mechanisms) on congestion and sharing. Replaying traces to emulate a different service configuration can introduce bias [9], because pacing changes traffic dynamics in ways that could impact behavior of other services, but such closed-loop feedback is beyond our ability to model in replay. Instead, YouTube and Netflix change their pacing configurations in production, and we observe the resulting traffic—both their own flows and those of other services—using a dataset that continuously collects live packet headers from a residential access network owned and managed by Columbia University. Covering over 1,500 residential units, this large-scale dataset enables reliable conclusions that are difficult to achieve in a typical lab setting. We discuss the dataset and its sufficiency in further detail in §3.1.

Limited control over where and when pacing is used. In contrast to an ideal setting where pacing could be toggled for any application, production experiments require participation from service providers. Our author team includes researchers from Netflix and YouTube, both of which have studied pacing [33, 63], enabling us to run an experiment where they enable and disable pacing in a controlled manner (see Table 2). While this scope is limited to two services, they are two of the three largest traffic sources in the Columbia dataset, making them a meaningful case study for understanding pacing at scale. By experimenting with both, we measure not just the impact of pacing on their own traffic but also how network conditions change depending on whether one or both is pacing. We further justify our focus on these two services in §5.3.

3 Dataset and Methodology

In this section, we introduce the Columbia residential access dataset used in this study, which extends the dataset published in our prior work [1, 75], as well as our method to compute congestion metrics and infer how services share network resources. As the inbound direction carries the majority of traffic [24], for the remainder of the paper, we focus on *traffic from servers to clients.*

3.1 Columbia Residential Access Dataset

We leverage a dataset originally introduced in our prior work [75]. At the time, we collected traffic for four hours per day from approximately 1,000 off-campus residential units owned and managed by Columbia University. Since then, we have expanded the deployment to continuous 24/7 collection across more than 1,500 residential units. Following the same data-anonymization policy as our prior work, the collection pipeline anonymizes privacy-sensitive fields and generally discards layer 4 packet payloads for privacy, while retaining selected information used for service identification such as DNS domain names and TLS Server Name Indication (SNI) values. The collection and anonymization pipeline was approved by Columbia IT and formally reviewed and declared exempt by Columbia's IRB as non-human-subjects research; further details are

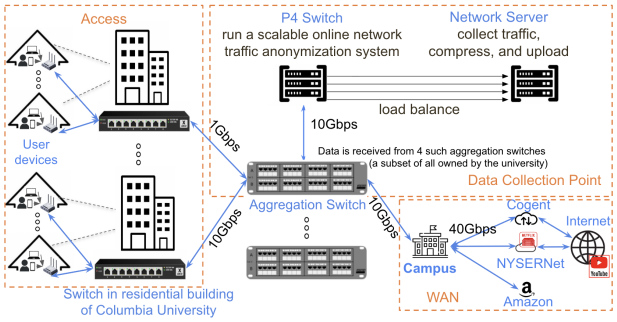


Figure 1: Network topology and data collection of the Columbia residential access dataset (reproduced from a figure on the dataset website [1]). NYSERNet is a Regional Research/Educational Network.

provided in our prior work [75]. Following the same data-sharing policy, the dataset used in this study is available upon request, with details provided on the dataset website [1].

Topology. Data comes from 33 university-affiliated *residential buildings* which house graduate students, faculty, and their families in over 1,500 *residential units*. Figure 1 shows the topology of the residential access network. Each residential unit has one Ethernet wall jack, and residents can connect devices directly via Ethernet or connect their own home routers for Wi-Fi access. As each Ethernet port has its own IP address, we can associate packets with a specific residential unit. However, a unit may contain multiple devices connected in different ways, making it challenging to reliably infer device identity and connectivity at this scale. We do not attempt to distinguish devices or human users within a unit, or to determine how individual devices connect. Ethernet ports in the same building share a common subnet, which allows us to identify packets traversing the same building. According to Columbia IT, the residential buildings use chassis switches, many of which have tens of megabytes of shared packet-buffer memory.

Packets are mirrored from four aggregation switches to the data collection servers over four 10 Gbps fiber pairs, which were deployed as part of the NSF COSMOS testbed fiber plant [19, 56]. Because the dedicated mirroring paths are lightly loaded relative to capacity and the data collection servers, co-located with COSMOS Sandbox 2, are 178 m from the aggregation switches, we assume that the additional delay from mirroring is approximately constant and negligible. We refer to the combination of the aggregation switches and collection servers as *the data collection point.*

A well-provisioned network. Columbia's network is relatively well-provisioned. Based on Netflix's internal data, the Columbia AS is in the 92nd percentile of average per-session throughput. Most residential-unit ports are provisioned at 1 Gbps, while some are provisioned at 100 Mbps. The majority of residential buildings in the dataset have a 1 Gbps uplink, while a substantial minority have a 10 Gbps uplink. Across all building-hour pairs, the 95th percentile of hourly building utilization rate, defined as the ratio of hourly throughput to building uplink capacity, is only 13%.

Discussion of generalizability. Our study focuses on a single, well-provisioned residential access network. While the residential units house graduate students, faculty, and their families, rather than the predominantly student population captured in many campus-network datasets, its network capacity and user population are specific to this setting and may differ from those of many other networks. For instance, Netflix data shows that Columbia users more often stream with browsers rather than TVs, unlike users in other networks. We acknowledge that these differences may result in different congestion and sharing dynamics. We refer readers to our prior work for a more detailed characterization of this network, including its service usage and delivery [75].

Although not representative of the Internet as a whole, this setting is a useful case study for observing real-world service interactions. It offers visibility into interactions across many homes and services that cannot be obtained from active measurements or server-side logs of individual services alone. Some of the qualitative insights from this setting may also apply to other well-provisioned access networks. Studying such a well-provisioned network is increasingly relevant given the growing deployment of fiber-to-the-home [10] and broader industry trends toward higher bandwidths [63]. For instance, the median Netflix session today has an average throughput 15× its average bitrate, up from 13× in 2023.

Dataset. For this study, we specifically use the data collected over four weeks, from September 30 to October 27, 2025. For pacing analysis in §6, we use data from all four weeks, each with a different pacing configuration. For all other analyses, we focus on the week of October 14–20 when YouTube and Netflix do not use application pacing, ensuring that our results reflect typical behavior without application pacing. We also demonstrate in Appendix A.2 that using one week of data is sufficient.

3.2 Map Flows to Services

As we are interested in understanding the performance and sharing between services, we map traffic to services (*e.g.,* YouTube, Netflix). We adopt the method from the paper that first introduced the dataset [75]: We first group packets with the same 5-tuple (*i.e.,* source IP, destination IP, protocol, source port, destination port) into *flows*. We then pair flows with corresponding DNS domains and TLS SNI. Finally, we map flows to services based on keywords in domains. For flows with no keywords found in the associated domains, we use the last two labels of the associated domains as the service. For the remaining flows with no mapped domains, we use the organization owning the server IP address as the service. The granularity of service labels depends on the observable flow information: domain-keyword mappings are more specific, while fallback mappings identify only the last two domain labels or server organization. We refer readers to our prior work for further details and discussion of the limitations of domain-based service mapping [75].

3.3 Compute Congestion Metrics

We use TCP packet loss and queueing delay as indicators of congestion and develop a method to infer them.

Recall that we refer to the aggregation switches and collection servers together as *the data collection point* (Fig. 1). An inbound packet can be lost before reaching the data collection point (*WAN loss*), at the data collection point, or between it and the end clients

(*access loss*). Since our goal is to study congestion within the residential access network, we focus on *access loss* and *access queueing delay* (*i.e.,* queueing delay occurring between the data collection point and the end clients). We also focus on *inbound packets*, as the inbound direction carries most traffic.

Identify access TCP packet loss. We focus on TCP loss because inferring non-TCP loss from transport headers is challenging. We identify TCP retransmits and loss based on sequence numbers in TCP headers. For each flow, we track the expected next sequence number of inbound packets, defined as the current packet's sequence number plus its payload length. We consider three cases. (1) If a packet arrives with a sequence number equal to the expected next sequence number, we treat it as the normal in-order case and update the value accordingly. (2) If the packet's sequence number is less than the expected value, we classify it as a retransmission. For each identified retransmission, if we previously observed packet(s) whose sequence ranges are covered by the retransmission, we label those packets as lost within the access network (access loss).[1] Otherwise, we infer that the original transmission was lost before reaching our collection point (WAN loss). (3) If the packet's sequence number is greater than the expected value, we infer that some packets covering the sequence range between the expected value and the current sequence number were lost upstream (WAN loss).[2] We define the access TCP packet loss rate as the percentage of TCP packets inferred as lost within the access network.

Infer access queueing delay. To infer access queueing delay, we first estimate access RTTs—the round-trip time between the data collection point and the end clients. Since a single ACK may acknowledge multiple packets in a batch, we estimate access RTTs accurately by identifying the most recent downstream packet covered by the ACK. Following the intuition behind Karn's Algorithm [44], we only compute RTTs for packets that were neither lost nor retransmitted, ensuring the measured RTTs correspond to the original transmission rather than a retransmission. We also incorporate SACK information when available, mapping the end of each SACK block to the corresponding packet and using the most recently updated block when multiple SACKs are present. Finally, to avoid including cases where a connection becomes idle for a long period before a delayed ACK is sent, we remove RTT samples above the 99th percentile, which effectively filters out spurious values caused by idle connections rather than network latency. We choose the 99th percentile as a conservative outlier threshold that removes only the extreme RTT tail, where such artifacts are more likely to occur; other high-percentile thresholds could serve a similar purpose. We also apply a smoothing step as per RFC 6298 [54] to further stabilize the values: $\mathrm{RTT}_{\mathrm{smoothed}} = \mathrm{RTT}_{\mathrm{smoothed}} * (1-\alpha) + \mathrm{RTT}_{\mathrm{new}} * \alpha$ with $\alpha = 1/8$. The smoothing step allows us to characterize meaningful congestion—delays that persist long enough to impact flow performance and congestion control decisions—rather than including transient spikes caused by measurement noise (*e.g.,* OS kernel scheduling jitter).

We propagate the smoothed access RTT values to nearby packets to increase coverage. By doing so, we can also infer smoothed access RTT values for UDP packets. The key assumption is that packets destined to the same unit within a short time interval should incur highly similar access RTTs. Specifically, for each building unit, we assign a packet the most recent smoothed RTT value associated with another packet to the same unit, since different units within a building may have different network configurations and so different latency. We use packets from the same unit rather than from the same device because we only know the unit identifier (destination IP address) to which the packet is destined. If a packet has no other packet within one second with a smoothed access RTT to use, we leave it unassigned and exclude it from later steps. We select this one-second RTT propagation window based on a sensitivity analysis of coverage and estimation error (Appendix A.3.4). This filtering step prevents assigning stale RTT values from temporally distant packets.

For each residential unit, we compute its *base access RTT*, defined as the minimum smoothed access RTT for that unit over the four-week measurement period; we also compute its *one-way propagation delay* as half of its base access RTT. Although access paths can be asymmetric—*e.g.,* DSL and cable links often have much lower upstream than downstream capacity, leading to larger upstream queueing delays—this network is all-fiber, and the access segment between the data-collection point and the homes has short base access RTTs (typically on the order of 1 ms). In this setting, propagation delay is small compared to the queueing delays we observe, so modest asymmetries in one-way propagation do not materially affect our results. For every packet associated with a smoothed access RTT, we define its *queueing delay* as the difference between its smoothed access RTT and the base access RTT of the unit that the packet is destined to.

Evaluation. To evaluate the accuracy of our congestion inference, we conduct 1,100 controlled experiments on a lab testbed that simulates the topology of the residential access network. We use iperf to generate TCP and UDP flows with varying sending rates. Since we lack ground-truth queueing delay and TCP packet loss, we validate our method using ground-truth TCP retransmits. Across all experiments, the R-squared values between our inferred and ground-truth retransmissions per second have a median of 0.98. The close match in TCP retransmissions provides evidence supporting the accuracy of our congestion inference, as these metrics are computed from the same TCP header information in a similar manner. We also validate our RTT propagation method using held-out RTT samples, finding a median absolute error of 0.89 ms (1% relative error) with the 1 s propagation window. More details about the testbed setup and evaluation results can be found in Appendix A.3.

Limitations. We acknowledge several limitations in our methodology. First, we do not infer loss for QUIC and other UDP packets, which account for 16% of the dataset (*e.g.,* traffic from Instagram, Apple, and Facebook), because unencrypted sequence number information is not uniformly available. Prior work has developed application-specific methods to infer loss for some UDP traffic, such

[1] This inference can overestimate access loss when TCP retransmits a packet that was not actually lost, for example following a retransmission timeout. However, since we use inferred loss as a signal of congestion, these cases still correspond to packets that TCP treated as lost, even if the original packets were not technically lost.

[2] To handle out-of-order packets, we buffer those with sequence numbers greater than the expected value for up to 3 ms before treating them as WAN loss signals. We use this 3 ms threshold following the fallback default in Wireshark [73] to balance measurement accuracy and processing complexity. We also account for TCP seq and ack number wraparound when tracking.

as RTP/WebRTC [47, 61], and QUIC-specific signals such as the optional spin bit can support passive RTT measurement [45]. These approaches require application- or protocol-specific information that is not uniformly available in our anonymized dataset. Incorporating such information to improve coverage is an interesting direction for future work. Second, our inference of UDP queueing delay relies on the presence of recent TCP traffic within the same residential unit, which leads to low RTT coverage when TCP activity is low and may introduce error when RTTs change rapidly within short intervals. We select the 1 s propagation window based on a controlled testbed evaluation (Appendix A.3.4), which may not fully reflect real traffic loads and network conditions. Extending our framework to incorporate QUIC-specific features remains an interesting direction for future work, which would require modifying the data collection pipeline to identify and store QUIC headers, as the dataset currently discards payload after layer 4 for privacy. These limitations do not affect the main pacing experiment, however, because during the experiment YouTube delivered traffic over TCP (rather than QUIC) for users of Columbia's residential network, and Netflix's video traffic also uses TCP. Third, we do not infer whether devices within a residential unit use Ethernet or Wi-Fi. For devices using Wi-Fi, wireless effects such as poor signal quality may contribute to the loss and delay we observe, and our methodology cannot distinguish these effects from congestion. Nevertheless, our congestion signals increase with more concurrent traffic (§4.2), supporting their use as indicators of congestion, and our main pacing analysis focuses on relative changes across weeks with and without pacing treatments. Last, we focus on downstream congestion within the access network, while uplink traffic may exhibit different loss and delay dynamics, particularly for applications such as video conferencing. We consider a more detailed investigation of uplink congestion as an interesting direction for future work.

3.4 Infer Sharing

We define the *in-flight interval* of a packet as the period starting at its capture timestamp at the data collection point and ending after the sum of its one-way propagation delay (half of the corresponding base RTT) and its queueing delay. As inbound traffic dominates volume and is thus the primary source of congestion in access networks [13, 24], we use this interval to represent when the packet is expected to be traversing the network path from the data collection point to the user. We then infer how packets share network resources based on overlaps between in-flight time intervals.

To quantify the degree of sharing, we count *the number of overlapping packets* for each packet. Specifically, to understand network resource sharing within a unit, for each packet, we count the number of packets destined for that unit whose in-flight time intervals overlap with its own (*unit-level overlaps*). Similarly, to understand network resource sharing within a building, for each packet, we compute the number of packets destined for the same building (same or different unit) whose in-flight time intervals overlap with its own (*building-level overlaps*). We compare sharing within and across residential units in §5.1.

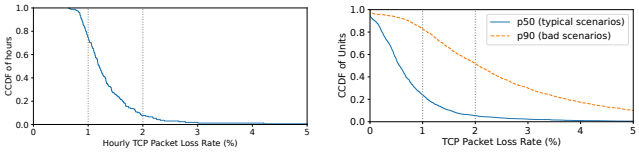
Furthermore, to understand sharing within and across services, we categorize overlaps by source. For each packet, we classify its overlapping packets as either *same-service* or *cross-service overlaps*, depending on whether those packets are from the same service. Similarly, we distinguish between *same-flow* and *cross-flow overlaps*, depending on whether those packets are from the same flow. We analyze these sharing dynamics across flows and services in §5.2.

Evaluation. We rely on the same 1,100 controlled experiments used for our congestion metrics evaluation, focusing on how well inferred overlaps track ground-truth buffer occupancy. In the testbed, the only relevant ground truth is buffer occupancy sampled at 1-second resolution. To compare against this, we estimate instantaneous buffer occupancy by counting, at each timestamp, how many packets' in-flight intervals include that time. We then sample this inferred occupancy every 0.1 seconds and take the median over each second to obtain a per-second estimate. Across all experiments, we find that the R-squared values between inferred and ground-truth buffer occupancy are above 0.82, with a median of 0.98. Because our inferred buffer occupancy and overlapping-packet counts are both derived from the same in-flight intervals, the close agreement between inferred and ground-truth buffer occupancy provides evidence for the accuracy of our overlapping-packet inference. More details can be found in Appendix A.3.

Limitations. We acknowledge several limitations in our methodology. First, because we attribute queueing delay to the inbound direction (where congestion most frequently occurs), we may overestimate the inbound in-flight time when congestion actually happens in the outbound direction. Second, our overlap metric includes both packets captured before a given packet (which may contribute to its congestion) and those captured after (which may be delayed by that packet). We use this broader definition because we do not assume that packet ordering is globally FIFO across the access path. The path includes chassis switches and heterogeneous home routers whose internal forwarding, queueing, and scheduling behavior we do not fully observe. To evaluate sensitivity to this choice, we consider an alternative definition that counts only overlapping packets captured before the packet under study. Although this alternative reduces absolute overlap counts, our main findings remain qualitatively unchanged (Appendix A.4). Developing more granular, hardware-aware reconstruction techniques is a compelling direction for future work.

4 Congestion in a Well-Provisioned Residential Access Network

At first glance, the Columbia residential network appears "good enough." The 95th percentile of hourly building utilization is just 13%, and its video throughput places it in the 92nd percentile of networks Netflix serves worldwide. These statistics suggest substantial spare capacity, although they do not rule out congestion. In this section, we examine what congestion looks like in such a well-provisioned network. We focus on a baseline week (week 3), during which both YouTube and Netflix run without their pacing treatments. We apply our methodology of §3.3 to measure access TCP packet loss and access queueing delay across all traffic. The picture that emerges differs sharply from what aggregate capacity suggests: TCP loss rates often fall in the 1–2% range, and queueing delays of tens of milliseconds are frequent.



(a) Packet loss rates per hour (b) Packet loss rates per unit

Figure 2: Access TCP packet loss rate across hours and units.

4.1 Severity of Congestion

Despite the network's low average hourly utilization, we still observe packet loss and queueing delay.

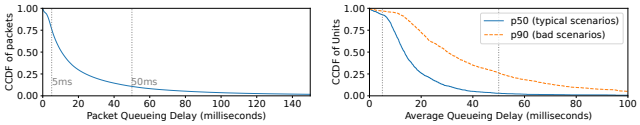
4.1.1 Access TCP packet loss rate: We measure access TCP packet loss as described in §3.3, inferring loss from TCP retransmissions and attributing it to the access network when the original packet was seen at our collection point. Some loss is expected, since many TCP congestion control algorithms rely on packet loss to adjust sending rates, so we use established benchmarks for context: ITU standards and Cisco guidelines treat 1% loss as the point where interactive applications (e.g., VoIP) degrade noticeably [21, 43], and Microsoft Teams considers 2% loss "poor" [49].

We compute the access loss rate for each hour and plot its distribution in Figure 2a. TCP loss rates are often substantial: 74% of hours exceed 1% loss, and 8% exceed 2%. These packet loss rates characterize the loss experienced by TCP traffic transmitted during each hour, rather than how frequently loss occurs over time. To examine the latter, we divide time into 1 ms intervals and find that 99% of milliseconds contain no inferred TCP loss. This loss is not confined to a few unlucky units. For each unit, we calculate the median and 90th-percentile loss rate over the week and show their distributions in Figure 2b. Most units typically experience low loss (76% have a median loss rate below 1%), but most do experience elevated loss at some times: in each unit's worst 10% of hours, 83% of units exceed 1% loss and half exceed 2%.

4.1.2 Access queueing delay. Another metric we use to measure congestion is *access queueing delay* (§3.3), defined as the excess access RTT above a unit's minimum. To contextualize these values, we borrow two thresholds from prior work [52]: AR/VR and autonomous driving require latency below 5 ms, while gaming and 360-degree video typically require latency below 50 ms.

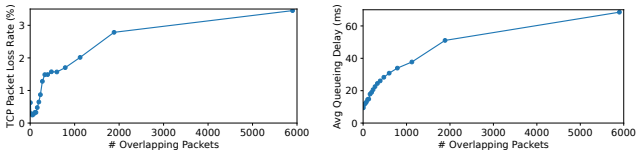
Figure 3a plots the CCDF of access queueing delay. A large fraction of packets see significant delay within the access network: 76% of packets exceed 5 ms, and 11% exceed 50 ms. While these thresholds are defined for end-to-end RTT, the access segment is only one component of the end-to-end path, so exceeding these thresholds within access alone means end-to-end delay can only be worse, which is a particular concern for latency-sensitive traffic.

As with loss, these delays are not confined to a few units. For each unit, we compute the median and 90th-percentile average queueing delay over the week; Figure 3b shows their distributions. For comparison, one-way latency across the continental US (west coast to a Columbia University IP address) is about 30 ms, yet over half of the units see access queueing delays above 30 ms in their worst 10% of hours. A widely cited study [25] reports that a 100 ms reduction in page load time can increase conversion rates by 8% for retail sites; in our measurements, more than 5% of units see over 100 ms of access queueing delay in their worst 10% of hours.



(a) Queueing delay per packet. (b) Avg. queueing delay per unit

Figure 3: CCDF of access queueing delay.



(a) Packet loss rate (b) Average queueing delay

Figure 4: Correlation between the number of unit-level overlaps per packet and congestion metrics.

4.2 Congestion increases with more traffic

We next examine how congestion varies with the amount of traffic sharing a unit's access path. For every packet (including retransmitted and lost packets), we compute its *unit-level overlap* count as defined in §3.4. Higher overlap means more packets are in flight to the same unit at the same time, so we expect congestion to increase.

To visualize this relationship, we sort all packets by overlap count and partition them into 20 equal-sized bins (quantile-based binning). For each bin, we compute the average overlap count and, over the packets in the bin, the TCP packet loss rate and average access queueing delay. Figure 4 plots the congestion metrics against average overlap. Both loss and delay rise sharply with overlap, confirming that packets with higher levels of sharing are more likely to see congestion. While this relationship is expected, verifying it provides support for using these metrics in our subsequent analysis.

4.3 Traffic is Bursty

The packet loss and queueing delay we observe occur despite the network's low average utilization. A plausible explanation is higher utilization at short timescales: even if an hour is under-utilized on average, traffic may briefly exceed capacity and contribute to the loss and queueing we see. We show examples where such short-timescale bursts occur, and in §6 we show that reducing burstiness causally reduces loss and delay in this network.

As one such example, Figure 5 illustrates a one-minute snapshot of building traffic. Over the minute, the building's 1 Gbps access link is only 17% utilized on average. At a per-second granularity, the link remains under-utilized, typically below 0.2 Gbps and never exceeding 0.85 Gbps. But at the millisecond level, 2% of all milliseconds in this example reach or exceed capacity. Utilization rates above 100% occur because we are measuring arrival rate, and bursts above capacity may be buffered or dropped.

Looking beyond this building, we see traffic that is highly bursty at short timescales and smoothed out at coarser ones. Figure 6 shows the CDF of utilization rates across (building, time-window) pairs. For 1-second windows, 90% of points have utilization below

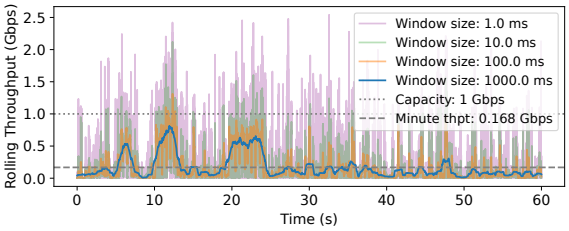


Figure 5: Example minute of building utilization at different timescales. Traffic is bursty at short timescales.

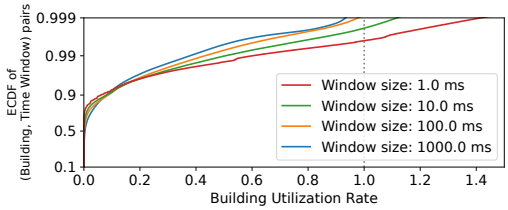


Figure 6: Building utilization rates at different timescales.

10%, confirming that the network is well provisioned, but as we consider smaller windows, an increasing fraction of periods hit or exceed link capacity.

These patterns align with datacenter studies that identify short-timescale bursts as a central challenge for congestion management and load balancing [2, 4–6, 34, 35], and with architectures such as L4S [14] that aim to separate bursty, latency-sensitive traffic from throughput-oriented flows.

5 Sharing Dynamics

The previous section showed that traffic can experience considerable loss and queueing delay in this well-provisioned access network and that short-term bursts push links to capacity, motivating our study of pacing. Prior work from YouTube [33] and Netflix [63] showed pacing reduces congestion for their own traffic, but their observation points were limited to their serving infrastructure and clients, without visibility into how their traffic interacts with other services in access networks. In this section, we characterize how network resources are shared during the baseline week (week 3), when YouTube and Netflix run without their pacing treatments; this establishes the sharing context needed to evaluate pacing's impact on neighboring traffic in §6.

5.1 Sharing Within and Across Units

We compare sharing within a residential unit to sharing across units in the same building. Using the Columbia dataset, we measure both *unit-level* and *building-level overlaps* (defined in §3.4). Figure 7 plots the CDFs of both metrics, and we observe notable differences between the two distributions. At the 50th percentile, the building-level overlap count is approximately 2.3 times the unit-level count, while at the 90th percentile, the gap narrows to about 1.6 times. Despite these differences, the magnitude of the gap remains small relative to the total number of units per building (typically in the tens), suggesting that congestion is driven disproportionately by interactions within the same unit, rather than by aggregate traffic across the building. Because of this finding, the remainder of the paper focuses on unit-level overlaps unless otherwise specified.

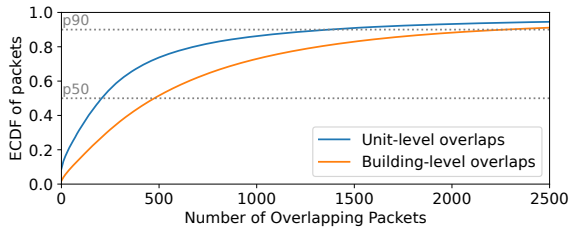
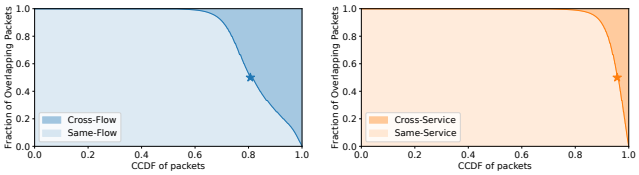


Figure 7: Distribution of unit-level and building-level overlaps over all packets.



(a) Same-flow vs. cross-flow **(b) Same- vs. cross-service**

Figure 8: CCDF of the fraction of same-flow and same-service overlaps per packet. Most sharing occurs within the same service and the same flow.

5.2 Sharing Within and Across Services

We next study sharing dynamics across flows and services. For each packet, we categorize its overlapping packets as *same-flow* or *cross-flow overlaps*, depending on whether those packets are from the same flow. For each packet, we calculate the fraction of its overlaps that are same-flow and plot this distribution in Figure 8a. We find that most sharing occurs within the same flow: for 81% of packets, over half of the overlapping packets are from the same flow (the blue marker). Since pacing is a flow-based technique, the intensity of same-flow sharing confirms that pacing has a significant opportunity to mitigate the bursts responsible for self-induced congestion. These findings further motivate our investigation into pacing in the next section.

Similarly, we classify overlapping packets as either *same-service* or *cross-service overlaps*, depending on whether they are from the same service. We calculate the fraction of same-service overlap for each packet and plot this distribution in Figure 8b. We find that most sharing occurs within the same service: for 96% of packets, more than 50% of the overlapping packets are from the same service (the orange marker). We further quantify the contribution of same- and cross-service sharing with an attribution analysis. For each packet, we attribute its observed loss or queueing delay to the packets that overlap with it, and then aggregate these attributions according to whether the overlapping packets belong to the same or a different service. We find that cross-service sharing accounts for only 5.2% of attributed loss and 8.3% of attributed queueing delay (see Appendix A.5 for details). Given the strong correlation between the number of overlapping packets and congestion (§4.2), this intense self-sharing is likely responsible for the majority of congestion experienced by a service. While this provides a clear incentive for services to adopt pacing to reduce their own congestion, it also poses a potential risk: as cross-service sharing is currently low, will spreading traffic over time via pacing increase this sharing and impact neighboring services negatively?

Service Type	Service (Ranking)	Traffic (%)	Active Unit-Hours (%)	TCP Packet Loss (%)	Avg Queueing Delay (ms)	Avg # Overlaps (Same-Service)	Avg # Overlaps (Cross-Service)
Video on Demand	YouTube (1)	13	45.5	0.58	22.5	410	6
	Netflix (3)	6.5	17.3	0.25	22.1	268	6
Game	Steam (2)	6.9	4.6	3.76	41.3	4631	13
	PlayStation (18)	1	1.3	2.29	81.8	1130	18
Cloud	iCloud (6)	2.9	75.9	1.16	23.8	474	16
	Microsoft 365 (36)	0.6	55.1	1.81	31.5	486	39
Social Media	Instagram (8)	2.7	30.2	0.04	24.5	311	14
	Facebook (27)	0.8	43.7	0.16	21.4	288	15
Real-time Streaming	Zoom (23)	0.9	20.5	0.34	15.4	183	11
	Twitch (35)	0.6	0.6	0.96	7.8	114	7

Table 1: Leading services by category (selected from top 50 by traffic volume). TCP packet loss rate and average queueing delay are defined in §4.1. Same- and cross-service overlaps are defined in §5.2; the table reports their average number per packet. Cell shading follows a gradient heatmap per column, where higher values are indicated by more intense colors.

5.3 Service-Specific Congestion and Sharing

Before diving into the effects of pacing, we summarize how the top 50 services by traffic volume experience congestion and share network resources. For the baseline week (week 3), during which both YouTube and Netflix run without their pacing treatments, we compute for each service its TCP packet loss rate, average queueing delay per packet, and the average numbers of same- and cross-service overlaps per packet. Table 1 shows the top two services in five categories (Video on Demand, Game, Cloud, Social Media, and Real-Time Streaming; the full table appears in Appendix A.6).

Gaming and cloud services stand out with high loss and large same-service overlap, consistent with their traffic being dominated by bulk transfers such as game downloads and file synchronization. In these cases, congestion is largely self-induced and self-shared within the service. In contrast, real-time streaming services such as Zoom and Twitch exhibit much lower queueing delays despite similar loss rates, likely reflecting their latency-sensitive designs [39]. Video on Demand services sit between these extremes, with moderate loss and queueing delay driven by bursty behavior. Overall, beyond sharing metrics, service-specific communication patterns and congestion-control choices play a major role in performance. We next return to one such mechanism, pacing.

6 Pacing Effects

YouTube and Netflix account for 13% and 7% of downstream traffic in the Columbia dataset, respectively, which is consistent with industry reports where they are the two largest video sources [58]. At the time we designed the experiment, both services were exploring pacing [33, 63] and were willing to toggle it for Columbia users on a controlled schedule. This gave us a rare chance to observe, from a vantage neither operator owns, how pacing by one or both of them affects congestion on a real residential network—both for the paced service itself and for others. Both YouTube and Netflix have since deployed pacing in production.

Pacing is a natural lever to study. Short-term bursts can cause considerable loss and queueing delay despite low average utilization (§4), and most sharing occurs within a single flow or service (§5.2). By smoothing bursts, pacing should thus ease self-induced congestion. The impact on other services is less obvious: pacing could shrink bursts that occasionally spill into neighbors, but because cross-service sharing is rare and bursts are short, it might also create overlaps with traffic that would otherwise run on an idle link—something prior single-service studies cannot see.

Across four weeks of switchback experimentation, we corroborate the substantial self-impact reported in those prior single-service studies: median loss rates and queueing delays for the paced service drop sharply (§6.3). For neighboring services, pacing slightly raises the probability of overlapping with the paced traffic but substantially reduces the intensity of those overlaps; congestion metrics for other services are stable overall and improve when packets directly overlap paced video (§6.4). Together these results show that in this well-provisioned network, pacing is good for the pacer *and* modestly beneficial for its neighbors.

6.1 Background on Pacing

Pacing mitigates burst-induced congestion by rate-limiting the sender to smooth traffic. The pacing rate can be determined through several strategies:

Fixed-rate pacing: A server enforces a fixed max transmission rate regardless of application or network conditions.

Transport-informed pacing: The rate is set by the congestion control algorithm. For example, BBR [17] paces based on its estimate of the bottleneck bandwidth, which may far exceed actual application requirements.

Application-informed pacing: The rate is set based on the specific requirements of the content being delivered. For example, YouTube's Trickle dynamically bounds the congestion window using target streaming rate and the current round-trip time (RTT), reducing loss by 43% and RTT by 28% [33]. Netflix's Sammy integrates adaptive bitrate (ABR) with rate control, allowing the system to select both video bitrate and pacing rate based on network conditions, reducing retransmissions by 36% and RTT by 14%, with slight quality of experience (QoE) gains [63].

6.2 Experiment Setup and Validation

We conduct a four-week pacing experiment in October 2025 in which YouTube and Netflix alternately enabled and disabled their pacing treatments for all users in Columbia's network, while keeping their congestion control algorithms unchanged.

	Date Ranges	Netflix	YouTube
Week 1	2025-09-30 ~ 2025-10-06	Paced	Paced
Week 2	2025-10-07 ~ 2025-10-13	Unpaced	Paced
Week 3	2025-10-14 ~ 2025-10-20	Unpaced	Unpaced
Week 4	2025-10-21 ~ 2025-10-27	Paced	Unpaced

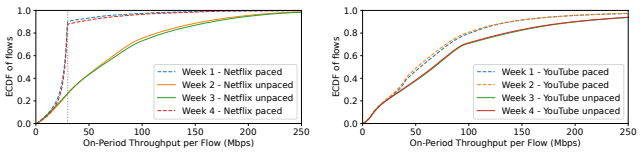
Table 2: Pacing experiment schedule.

Throughout this section, we use "paced" and "unpaced" to refer to whether these pacing treatments are enabled. Following the switchback experimental design detailed in Table 2, this experiment yields all four combinations of {YouTube paced, unpaced} × {Netflix paced, unpaced}, each sustained for a full week. We use data from week 3—during which neither YouTube nor Netflix uses its pacing treatment—to study unpaced congestion and sharing dynamics (§4, §5). We use data across all four weeks to study pacing effects (§6).

Both providers made changes to support the experiment. While our method infers RTTs for UDP traffic, we do not infer UDP packet loss, as QUIC/UDP do not directly expose sequence or acknowledgment numbers needed by our loss-inference method. To facilitate our inference, YouTube switched Columbia users from QUIC to TCP sessions for the duration of the experiment. YouTube uses BBR as the TCP congestion control algorithm, and Netflix uses its custom version of TCP NewReno. YouTube recently implemented a new application-informed pacing algorithm (distinct from its prior publication [33] and inspired by Sammy [63]) and used this algorithm during the experiment. Netflix has focused its application pacing deployment on TVs, STBs, and game consoles. To maximize the power of this experiment, Netflix instead used a simple fixed pacing rate of 30 Mbps that applied to all devices in the network (including browsers, tablets, and phones). The 30 Mbps rate was chosen to resemble the behavior of the production pacing algorithm for the particular bitrates in this network.

To quantify the pacing effects observed, we use the metric of *on-period throughput* per flow, defined as the number of bytes transmitted by the flow divided by the duration of the union of all in-flight intervals for that flow. We calculate this metric only for flows larger than 1 MB to focus on sustained video playback. This filtering excludes only about 2% of the corresponding traffic. The excluded small flows are likely associated with service webpage loading or other auxiliary traffic. Based on our operational experience, such small transfers are also typically excluded from provider-side analyses of video user experience. Our validation shows a distinct shift in the distribution of on-period throughput per flow during paced weeks, as shown in Figure 9. Specifically, for both YouTube and Netflix, the CDF of on-period throughput reveals a significant reduction in the upper percentiles. As Netflix employs a fixed-rate pacing of 30 Mbps for this experiment, the on-period throughput for its flows exhibits a much more substantial reduction than YouTube's. The median of Netflix's on-period throughput decreases by 50%, from 60 to 30 Mbps. In contrast, YouTube's pacing is more conservative, decreasing median throughput by 25%, from roughly 67 to 50 Mbps.

We also verify that traffic volume remains similar across the four weeks, indicating that observed congestion differences are likely driven by changes in pacing behavior rather than user viewing behavior. We computed the overall and Netflix/YouTube hourly traffic volume for each residential unit and then plotted the CDF of



(a) Netflix Throughput **(b) YouTube Throughput**

Figure 9: On-period throughput per flow.

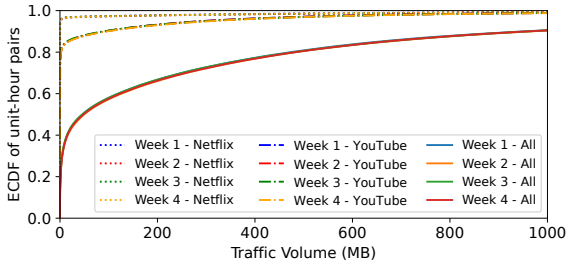


Figure 10: Total hourly traffic volume per unit for all 4 weeks.

the unit-hour traffic volumes across the four experimental weeks (Fig. 10). The curves for each week are similar, with mean unit-hour traffic volumes ranging from 26–27 MB for Netflix, 52–54 MB for YouTube, and 384–403 MB overall across the four weeks. This suggests that base traffic demand did not change significantly during the experiment.

6.3 Improved Performance for the Paced Services

Previous papers from both YouTube and Netflix have shown that pacing improves their own congestion metrics without sacrificing their own user experience [33, 63]. Our experiment results corroborate these congestion improvements in an independent network.

To avoid unreliable metric values when a unit-hour contains too few packets, we restrict our analysis in this section to unit-hours containing at least 10,000 packets for the corresponding service. This is a relatively low threshold—10,000 packets represents approximately two minutes of SD 480p video, based on a 1 Mbps bitrate and a 1500-byte MTU [74]. This filtering excludes less than 1% of the corresponding traffic.

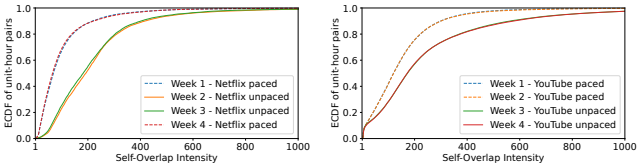
We first examine how pacing reduces the self-induced bursts generated by the paced services. Specifically, for a packet sharing network resources with traffic from its own service, we measure how pacing reduces the number of same-service overlaps on average. To quantify this, we define the *self-overlap intensity for a given service* as the expected number of same-service overlaps per packet, conditioned on the occurrence of at least one overlap. This condition allows us to focus on packets where sharing happens.

Mathematically,

$$\text{Overlap Intensity}(V_1, V_2) = \mathbb{E}[X_p^{V_2} \mid p \in \text{Service } V_1, X_p^{V_2} > 0]$$

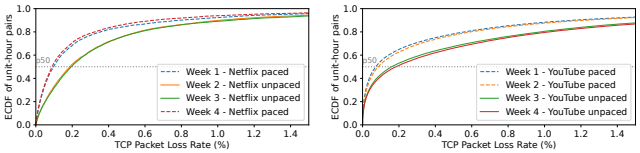
where $X_p^{V_2}$ represents the number of Service V_2 packets that a packet p belonging to Service V_1 overlaps with. In particular, self-overlap intensity is the overlap intensity when $V_1 = V_2$.

We observe that pacing leads to significant reductions in Netflix's and YouTube's self-overlap intensity. Figure 11 plots the CDFs



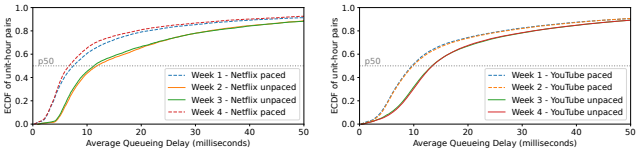
(a) Netflix (b) YouTube

Figure 11: Self-overlap intensity of Netflix and YouTube traffic, over all 4 weeks.



(a) Netflix (b) YouTube

Figure 12: Access TCP packet loss rate of Netflix and YouTube traffic, per unit per hour, over all 4 weeks.



(a) Netflix (b) YouTube

Figure 13: Average access queueing delay of Netflix and YouTube traffic, per unit per hour, over all 4 weeks.

of self-overlap intensity over unit-hour pairs, with each line representing a week. By analyzing the metric across unit-hours, we capture the distribution over varying times of day and network conditions, ensuring the results are not skewed by a few heavy users and reflecting the traffic dynamics encountered by users. Pacing decreases the median self-overlap intensity by 65% for Netflix (from ~185 during unpaced weeks to ~65 during paced weeks) and 30% for YouTube (from ~170 during unpaced weeks to ~120 during paced weeks). At the 90th percentile, the reductions are similarly substantial, with Netflix decreasing by 46% and YouTube by 48%.

These reductions yield clear improvements in congestion metrics. We compute the access TCP packet loss rate and the average access queueing delay per unit-hour. Figure 12 and Figure 13 plot the CDFs of these metrics across unit-hours. During paced weeks, the median loss rate decreased by 51% for Netflix and 55% for YouTube (Fig. 12). The average queueing delay followed a similar trajectory, decreasing by 27% for Netflix and 39% for YouTube (Fig. 13). Although YouTube had a much smaller reduction in on-period throughput (Fig. 9), both services observed similar improvements in packet loss rate and queueing delay, perhaps because both treatments reduced on-period throughput, especially on the tail.

All observed differences are statistically significant, as determined by the Kolmogorov-Smirnov (KS) test. The full test statistics are reported in Appendix A.7.

Regarding the impact on end users, both Netflix and YouTube indicate that, based on their standard QoE metrics, they did not observe meaningful changes during the experiment, consistent with their previous findings that pacing can reduce congestion without sacrificing user experience [33, 63]. This is also consistent with our expectation, as the network is well provisioned, such that reducing traffic burstiness through pacing need not degrade user experience. The underlying provider-side QoE data and analyses are internal to Netflix and YouTube, so we do not report the individual metrics or detailed results here. We do not analyze the fine-grained impact of cross-service sharing on QoE, because QoE is difficult to infer reliably from network traces and the anonymized dataset prevents Netflix and YouTube from reliably identifying sessions with substantial overlap with other services. We leave such a QoE analysis to future work.

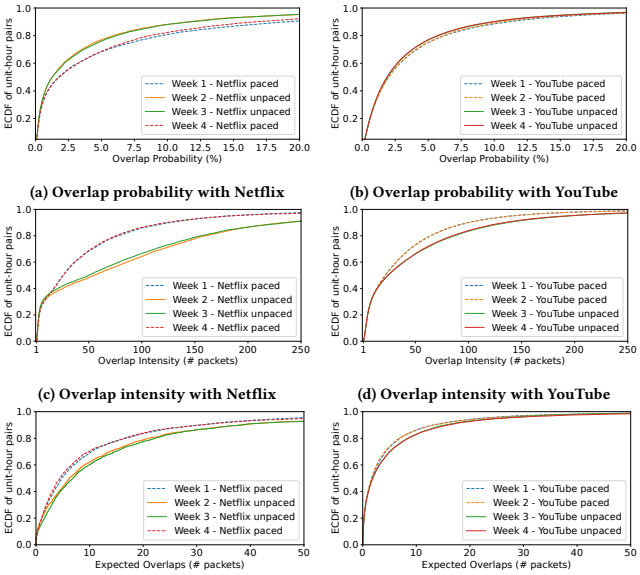
6.4 Neutral to Beneficial Impact on Neighboring Services

Our results (§6.3) and prior studies [33, 63] show that pacing benefits the paced services, motivating it as a way to make a service more "friendly" when services share the network. However, pacing can also spread traffic over a longer period of time, inducing overlaps with other services that would not have occurred otherwise. Studying the impact of pacing on neighboring services in the wild therefore remains important.

To quantify these cross-service sharing dynamics, we introduce two new metrics. First, we define *overlap probability* as the likelihood of non-Netflix (non-YouTube) packets overlapping with at least one Netflix (YouTube) packet. Second, we define *expected overlaps* as the expected number of Netflix (YouTube) packets that a packet belonging to another service overlaps with, which is also the product of overlap probability and overlap intensity. Intuitively, expected overlap is the average number of concurrent packets a single packet "sees" during its time in flight.

Similar to the previous subsection, to avoid unreliable metric values when a unit-hour contains too few packets, we restrict our Netflix (or YouTube) analysis in this section to unit-hours containing at least 10,000 Netflix (or YouTube, respectively) packets and at least 10,000 concurrent packets from other services. We apply this same threshold as previously justified. We compute overlap probability, overlap intensity, and expected overlaps per unit-hour.

When a service enables pacing, we observe a notable distributional shift in overlap probability, overlap intensity, and expected overlaps of other services with the paced service. Because spreading traffic over time makes the paced service more likely to be encountered by neighboring packets, pacing does slightly increase the overlap probability, particularly at the tail: the 90th percentile of overlap probability with Netflix increases from roughly 12% to 18% (Fig. 14a), while for YouTube it increases from roughly 10% to 11% (Fig. 14b). Conversely, because pacing reduces large bursts, the reduction in overlap intensity is significant: the 90th percentile for Netflix decreases from roughly 235 packets to 120 (Fig. 14c), and for YouTube from roughly 135 packets to 100 (Fig. 14d). Ultimately, the reduction in overlap intensity offsets the increased overlap probability, resulting in a slight decrease in expected overlaps: the 90th percentile of expected overlaps with Netflix decreases from roughly 38 packets to 31 (Fig. 14e), and for YouTube from roughly 16 packets to 13 (Fig. 14f). These changes are less pronounced for YouTube, consistent with the smaller reductions in



(a) Overlap probability with Netflix

(b) Overlap probability with YouTube

(c) Overlap intensity with Netflix

(d) Overlap intensity with YouTube

(e) Expected overlaps with Netflix

(f) Expected overlaps with YouTube

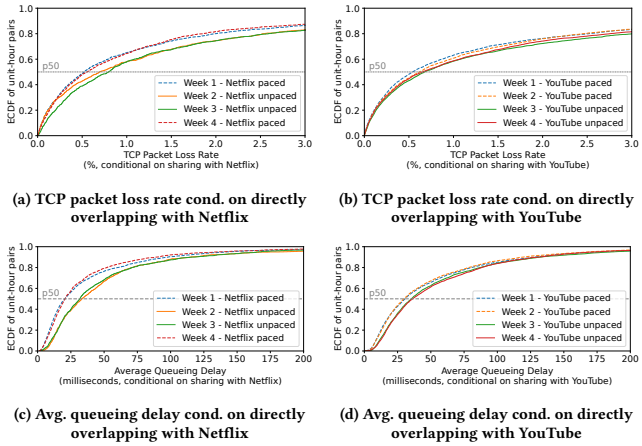
Figure 14: Overlap probability, overlap intensity, and expected overlaps of non-Netflix (or non-YouTube) packets, over all 4 weeks. The reduction in overlap intensity offsets the increased overlap probability, resulting in a slight decrease in expected overlaps.

on-period throughput observed for YouTube (§6.2). This is likely because YouTube's application-informed pacing reduces its sending rate less aggressively than the fixed-rate pacing employed by Netflix for the experiment, and is applied on top of BBR's existing transport-informed pacing. These results suggest that, in the well-provisioned network we study, smoothing traffic is slightly more beneficial for neighboring services than allowing high-intensity bursts, even if the probability of overlapping increases.

We then investigate how these sharing dynamics translate into congestion. The dual effect of pacing (increased overlap probability and decreased overlap intensity) prompts us to study congestion in two scenarios: **when sharing with the paced service actually occurs** and **when sharing with the paced service can potentially happen**. Similar to the previous subsection, we analyze congestion metrics across unit-hours to capture the congestion experienced by users while preventing heavy users from skewing the results.

First, to analyze congestion *during actual sharing episodes*, we compute the access TCP packet loss rate and queueing delay *for non-Netflix (non-YouTube) packets that directly overlap with Netflix (YouTube)*. Figure 15 plots the CDFs of these metrics across unit-hour pairs. Pacing results in a clear reduction in congestion. For the packets directly overlapping with the paced service, the median access TCP loss rate decreases by 29% for Netflix (Fig. 15a) and 13% for YouTube (Fig. 15b). Similarly, the median average access queueing delay decreases by 36% for Netflix (Fig. 15c) and 16% for YouTube (Fig. 15d). As with the overlap metrics, these changes are less pronounced for YouTube, consistent with its smaller reductions in on-period throughput (§6.2).

Looking only at packets that actually overlap with the paced service is insufficient, because it does not capture pacing's impact



(a) TCP packet loss rate cond. on directly overlapping with Netflix

(b) TCP packet loss rate cond. on directly overlapping with YouTube

(c) Avg. queueing delay cond. on directly overlapping with Netflix

(d) Avg. queueing delay cond. on directly overlapping with YouTube

Figure 15: Access TCP packet loss rate and average queueing delay of other packets that directly overlap with Netflix (or YouTube), over all 4 weeks. Pacing reduces congestion for packets that directly overlap with the paced service.

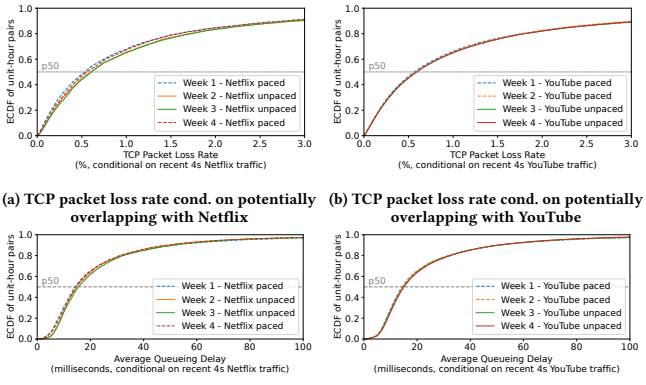
on how many packets overlap. To address this, we study congestion *when sharing could potentially happen* by introducing the concept of *potentially overlapping*. A packet is potentially overlapping with the paced service if it is within a specific time window of packets from the paced service. This definition allows us to exclude packets that are unlikely to be affected by the paced service, such as those sent while a video is paused. We use a 4-second window, consistent with the recommended MPEG-DASH segment duration [23, 53]. As shown in Figure 16, congestion metrics for potentially overlapping packets show only small changes, with minimal improvements in some cases. We also evaluate a larger 300-second window (default TCP idle timeouts for many servers [22, 48]) and obtain similar results (omitted for brevity).

All comparisons between the paced and unpaced weeks are statistically significant, as determined by the Kolmogorov-Smirnov (KS) test. The full KS statistics and p-values are reported in Appendix A.7. We also observe the same qualitative conclusions for tail queueing delay, showing that these effects are not limited to average delay (Appendix A.8).

In summary, our analysis of cross-service pacing effects complements prior studies from Netflix and YouTube, which focused on the impact of pacing on themselves. For the network under study, pacing produces a dual effect on sharing dynamics, resulting in improved congestion metrics when direct overlaps occur while maintaining at least stable congestion metrics overall. As our experiment is limited to Netflix and YouTube in this well-provisioned network, we consider studying pacing in other network and service settings an important direction for future work.

7 Related Work

Residential and access-network measurement. Residential and access-network performance has been studied with active and passive measurements [12, 26, 27, 29, 32, 38, 46, 50, 59, 60, 64–67]. These methods are well suited to capacity, bottleneck localization, and per-home utilization. Other work has used packet traces at home gateways or access points to infer application-layer performance,



(a) TCP packet loss rate cond. on potentially overlapping with Netflix

(b) TCP packet loss rate cond. on potentially overlapping with YouTube

(c) Avg. queueing delay cond. on potentially overlapping with Netflix

(d) Avg. queueing delay cond. on potentially overlapping with YouTube

Figure 16: Access TCP packet loss rate and average queueing delay of other packets potentially overlapping with Netflix (or YouTube) traffic, over all 4 weeks. Congestion for those potentially overlapping packets from other services changes little across paced and unpaced weeks, with slight improvements in some cases.

such as streaming video QoE [15]. To our knowledge, no prior residential measurement study has (i) had a packet-level vantage on traffic from many homes behind shared access infrastructure, (ii) studied per-packet congestion and how different services share a residential queue as in §4.1 and §5, nor (iii) coordinated with content providers to toggle a rate-control mechanism in production and observed the effect on the network as a whole.

The dataset we use is characterized in a prior paper [75], which describes service usage, CDN delivery, and infrastructure for the Columbia residential network. We focus instead on congestion, queue sharing, and an in-the-wild experiment with pacing.

Burst-driven congestion in the datacenter. The bursts we see are reminiscent of those in datacenters. Recent measurement work characterizes the same short-timescale, application-shaped congestion in production RDMA fabrics [35] and high-degree incast [16]. Designs such as DCTCP [5], pFabric [6], CONGA [4], DRILL [34], and Vertigo [2] attack the same underlying problem with controlled fabrics and in-network mechanisms. The residential setting has none of those affordances: no controlled fabric, no in-network telemetry, and a much more heterogeneous traffic mix.

Application-informed pacing. We study *application-informed* pacing which deliberately caps sending rate well below network capacity. This was proposed in Trickle and Sammy [33, 63], though neither is the mechanism in our experiment. Application-informed pacing is qualitatively different from classical TCP pacing [3, 37, 41, 51, 57, 68, 71, 72] or the pacing built into BBR [17, 18], which smooths packets within an existing sending window.

Congestion control comparison, fairness, and in-the-wild experiments. Much of what we know about CCA and service competition comes from controlled lab studies of CCA fairness [8, 28, 69, 70] and of service-level friendliness [55]. We instead observe many services in the wild on a real residential network, and find that service-level behavior is diverse. Steam packets, for instance, see roughly 17× more same-service overlaps than Netflix (§5.3, Table 1).

A complementary line of work evaluates rate-control mechanisms through controlled production experiments, from inside individual services: BBR at Google [18] (later complemented by lab evaluations of its interaction with Cubic [40]), Trickle and Sammy [33, 63], and Ben-Ameur et al.'s recent L4S field study at Comcast [11]. These production experiments are typically run as randomized A/B tests: allocating flows randomly to treatment and control. This is natural from a single-service vantage but is biased in congested networks by exactly the queue-sharing dynamics we measure [9, 62]. Our multi-service vantage lets us use a switchback design with week-long blocks instead, avoiding this bias and measuring cross-service effects directly. The same setup also lets us study properties like cross-CCA fairness directly on residential traffic, rather than separately in the lab and at one provider.

Abagnale [30] synthesizes algorithm-level behavior from packet traces, which would further help investigate the service-level diversity we observe in §5 and tie it to specific CCA behavior. We defer this to future work.

8 Conclusion

We present a comprehensive study on congestion, sharing, and pacing in a residential access network. We develop a method to infer congestion metrics and measure network resource sharing by identifying packets with overlapping in-flight time intervals. Even in the well-provisioned network we study, bursty traffic causes considerable congestion, and services often share capacity primarily with their own traffic. We further show that, in this network, pacing effectively reduces congestion for both the paced service and other services when sharing occurs, suggesting that pacing can be a practical strategy for improving performance in similar well-provisioned access networks.

More broadly, this paper shows that combining packet-level visibility across many homes, a methodology for inferring service-level sharing, and experimental control over a rate-control mechanism is feasible in an operational residential network. The dataset is available upon request [1], the methodology is documented in §3, and the pacing experiment was run with content-provider cooperation that other partnerships can replicate. We hope other researchers and network operators will run analogous experiments, so that the community can accumulate evidence about the network-wide effects of rate-control changes.

Acknowledgements

We would like to thank our shepherd and the anonymous reviewers for their valuable feedback. We are grateful for the support of Anthony Robert Acosta, Jarred Lee Buford O'Brien, Joel L. Rosenblatt, Thomas Rom, Yosef Gunsburg, the security, privacy, and networking teams of Columbia University Information Technology (CUIT), and the NSF COSMOS testbed. This work is supported in part by NSF grants CNS-2212479, CNS-2450567, and CNS-2148128.

References

[1] 2025. Columbia University Residential Datasets. https://wimnet.github.io/CUResidential/.

[2] Sepehr Abdous, Erfan Sharafzadeh, and Soudeh Ghorbani. 2021. Burst-tolerant datacenter networks with Vertigo. In *Proceedings of the 17th International Conference on emerging Networking EXperiments and Technologies (CoNEXT)*. 178–192.

[3] Amit Aggarwal, Stefan Savage, and Thomas E. Anderson. 2000. Understanding the Performance of TCP Pacing. In *Proc. IEEE INFOCOM*, Vol. 3. 1157–1165. doi:10.1109/INFCOM.2000.832483

[4] Mohammad Alizadeh, Tom Edsall, Sarang Dharmapurikar, Ramanan Vaidyanathan, Kevin Chu, Andy Fingerhut, Vinh The Lam, Francis Matus, Rong Pan, Navindra Yadav, and George Varghese. 2014. CONGA: Distributed congestion-aware load balancing for datacenters. In *Proceedings of the 2014 ACM Conference on SIGCOMM*. 503–514.

[5] Mohammad Alizadeh, Albert Greenberg, David A Maltz, Jitendra Padhye, Parveen Patel, Balaji Prabhakar, Sudipta Sengupta, and Murari Sridharan. 2010. Data Center TCP (DCTCP). In *Proceedings of the 2010 ACM Conference on SIGCOMM*. 63–74.

[6] Mohammad Alizadeh, Shuang Yang, Milad Sharif, Sachin Katti, Nick McKeown, Balaji Prabhakar, and Scott Shenker. 2013. pFabric: Minimal near-optimal datacenter transport. In *Proceedings of the 2013 ACM Conference on SIGCOMM*. 435–446.

[7] Martin Armstrong. 2023. https://www.statista.com/chart/31075/global-average-internet-download-speed/.

[8] Venkat Arun and Hari Balakrishnan. 2018. Copa: Practical delay-based congestion control for the internet. In *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18)*. 329–342.

[9] Mihovil Bartulovic, Junchen Jiang, Sivaraman Balakrishnan, Vyas Sekar, and Bruno Sinopoli. 2017. Biases in data-driven networking, and what to do about them. In *Proceedings of the 16th ACM Workshop on Hot Topics in Networks*. 192–198.

[10] Pete Bell. 2025. The Rise of Fiber. https://blog.telegeography.com/the-rise-of-fiber.

[11] Kyraan Ben-Ameur, Francesco Bronzino, Paul Schmitt, and Nick Feamster. 2026. Measuring Low Latency at Scale: A Field Study of L4S in Residential Broadband. In *Proc. Passive and Active Measurement (PAM) (LNCS, Vol. 16477)*. Springer, 168–181. doi:10.1007/978-3-032-18268-5_8

[12] Zachary S Bischof, Fabian E Bustamante, and Nick Feamster. 2018. Characterizing and Improving the Reliability of Broadband Internet Access. *Telecommunications Policy Research Conference (TPRC)* (2018).

[13] Body of European Regulators for Electronic Communications (BEREC). 2024. *BEREC Report on the IP Interconnection Ecosystem*. Technical Report BoR (24) 177. BEREC. https://www.berec.europa.eu/en/all-documents/berec/reports/berec-report-on-the-ip-interconnection-ecosystem Accessed: 2026-04-27.

[14] Bob Briscoe, Koen De Schepper, Marcelo Bagnulo, and Greg White. 2023. Low Latency, Low Loss, and Scalable Throughput (L4S) Internet Service: Architecture. RFC 9330. doi:10.17487/RFC9330

[15] Francesco Bronzino, Paul Schmitt, Sara Ayoubi, Guilherme Martins, Renata Teixeira, and Nick Feamster. 2020. Inferring streaming video quality from encrypted traffic: Practical models and deployment experience. *ACM SIGMETRICS Performance Evaluation Review* 48, 1 (2020), 27–28.

[16] Christopher Canel, Balasubramanian Madhavan, Srikanth Sundaresan, Neil Spring, Prashanth Kannan, Ying Zhang, Kevin Lin, and Srinivasan Seshan. 2024. Understanding Incast Bursts in Modern Datacenters. In *Proc. ACM Internet Measurement Conference (IMC)*. 674–680. doi:10.1145/3646547.3689028

[17] Neal Cardwell, Yuchung Cheng, C Stephen Gunn, Soheil Hassas Yeganeh, and Van Jacobson. 2016. BBR: Congestion-based congestion control: Measuring bottleneck bandwidth and round-trip propagation time. *Queue* 14, 5 (2016), 20–53.

[18] Neal Cardwell, Yuchung Cheng, C Stephen Gunn, Soheil Hassas Yeganeh, and Van Jacobson. 2017. BBR: Congestion-based congestion control. *Commun. ACM* 60, 2 (2017), 58–66.

[19] Tingjun Chen, Jiakai Yu, Artur Minakhmetov, Craig Gutterman, Michael Sherman, Shengxiang Zhu, Steven Santaniello, A Biswas, Ivan Seskar, Gil Zussman, and Dan Kilper. 2022. A software-defined programmable testbed for beyond-5G optical-wireless experimentation at city-scale. *IEEE Network* 36, 2 (April 2022), 90–99.

[20] Cisco Systems. [n. d.]. *Troubleshoot High CPU Utilization due to SNMP on Catalyst 9800 Wireless Controllers*. Cisco Systems, Inc. https://www.cisco.com/c/en/us/support/docs/wireless/catalyst-9800-wireless-controllers-cloud/222237-troubleshoot-high-cpu-utilization-due-to.html

[21] Cisco Systems, Inc. [n. d.]. *Quality of Service for Voice over IP*. Solutions Document. Cisco Systems, Inc. https://www.cisco.com/c/en/us/td/docs/ios/solutions_docs/qos_solutions/QoSVoIP/QoSVoIP.html Accessed: 2024-05-22.

[22] CloudBlue. 2020. *Handling Idle TCP Connections*. CloudBlue. https://docs.cloudblue.com/cbc/20.4/premium/content/Firewall-Configuration/Handling-Idle-TCP-Connections.htm CloudBlue Commerce 20.4 Documentation. Accessed: 2024-05-22.

[23] Cloudflare. [n. d.]. *What is MPEG-DASH? | HLS vs. DASH*. https://www.cloudflare.com/learning/video/what-is-mpeg-dash/

[24] Columbia University. [n. d.]. Columbia University Internet Traffic Statistics. https://www.columbia.edu/acis/networks/bandwidth.html. Accessed: 2026-04-30.

[25] Deloitte Digital. 2020. Milliseconds Make Millions: Why Mobile Speed is the Next Frontier for Brand Growth. https://www.deloitte.com/ie/en/services/consulting/research/milliseconds-make-millions.html.

[26] Lucas DiCioccio, Renata Teixeira, and Catherine Rosenberg. 2013. Measuring home networks with homenet profiler. In *International Conference on Passive and Active Network Measurement*. Springer, 176–186.

[27] Marcel Dischinger, Andreas Haeberlen, Krishna P Gummadi, and Stefan Saroiu. 2007. Characterizing residential broadband networks. In *Proceedings of the 7th ACM SIGCOMM conference on Internet measurement*. 43–56.

[28] Mo Dong, Tong Meng, Doron Zarchy, Engin Arslan, Yossi Gilad, Brighten Godfrey, and Michael Schapira. 2018. PCC vivace: online-learning congestion control. In *15th USENIX symposium on networked systems design and implementation (NSDI 18)*. 343–356.

[29] Nick Feamster. 2016. Revealing Utilization at Internet Interconnection Points. *Telecommunications Policy Research Conference (TPRC)* (2016).

[30] Margarida Ferreira, Ranysha Ware, Yash Kothari, Inês Lynce, Ruben Martins, Akshay Narayan, and Justine Sherry. 2024. Reverse-Engineering Congestion Control Algorithm Behavior. In *Proc. ACM Internet Measurement Conference (IMC)*. doi:10.1145/3646547.3688443

[31] Tobias Flach, Pavlos Papageorge, Andreas Terzis, Luis Pedrosa, Yuchung Cheng, Tayeb Karim, Ethan Katz-Bassett, and Ramesh Govindan. 2016. An internet-wide analysis of traffic policing. In *Proceedings of the 2016 ACM SIGCOMM Conference*. 468–482.

[32] Romain Fontugne, Anant Shah, and Kenjiro Cho. 2020. Persistent Last-mile Congestion: Not so Uncommon. In *Proc. ACM Internet Measurement Conference (IMC)*. 420–427. doi:10.1145/3419394.3423648

[33] Monia Ghobadi, Yuchung Cheng, Ankur Jain, and Matt Mathis. 2012. Trickle: Rate limiting YouTube video streaming. In *2012 USENIX Annual Technical Conference (USENIX ATC 12)*. 191–196.

[34] Soudeh Ghorbani, Zibin Yang, P Brighten Godfrey, Yashar Ganjali, and Amin Firoozshahian. 2017. DRILL: Micro load balancing for low-latency data center networks. In *Proceedings of the 2017 ACM Conference on SIGCOMM*. 225–238.

[35] Soudeh Ghorbani, Yimeng Zhao, Srikanth Sundaresan, Ying Zhang, Yijing Zeng, Abhigyan Sharma, Prashanth Kannan, and Cristian Lumezanu. 2025. Congestion Patterns in a Large-scale RDMA Datacenter. In *Proceedings of the 2025 ACM Internet Measurement Conference*. 944–951.

[36] Petros Gigis, Matt Calder, Lefteris Manassakis, George Nomikos, Vasileios Kotronis, Xenofontas Dimitropoulos, Ethan Katz-Bassett, and Georgios Smaragdakis. 2021. Seven Years in the Life of Hypergiants' Off-Nets. In *Proceedings of the ACM SIGCOMM Conference 2021*.

[37] Carlo Augusto Grazia, Martin Klapez, and Maurizio Casoni. 2021. The New TCP Modules on the Block: A Performance Evaluation of TCP Pacing and TCP Small Queues. *IEEE Access* 9 (2021), 129329–129336. doi:10.1109/ACCESS.2021.3113891

[38] Sarthak Grover, Mi Seon Park, Srikanth Sundaresan, Sam Burnett, Hyojoon Kim, Bharath Ravi, and Nick Feamster. 2013. Peeking Behind the NAT: An Empirical Study of Home Networks. In *Proceedings of the ACM Internet Measurement Conference (IMC) 2013*.

[39] Kai Hayashi, Angelo Paxinos, and Yueshi Shen. 2021. Low Latency, High Reach: Creating an Unparalleled Live Video Streaming Network at Twitch. Twitch Blog. https://blog.twitch.tv/en/2021/10/25/low-latency-high-reach-creating-an-unparalleled-live-video-streaming-network-at-twitch/ Accessed: 2026-08-19.

[40] Mario Hock, Roland Bless, and Martina Zitterbart. 2017. Experimental Evaluation of BBR Congestion Control. In *Proc. IEEE 25th Int. Conf. on Network Protocols (ICNP)*. 1–10. doi:10.1109/ICNP.2017.8117540

[41] Janey C. Hoe. 1995. *Start-up Dynamics of TCP's Congestion Control and Avoidance Schemes*. Master's thesis. Massachusetts Institute of Technology. https://dspace.mit.edu/handle/1721.1/36971

[42] Intel Corporation. 2021. Different Wi-Fi Protocols and Data Rates. https://www.intel.com/content/www/us/en/support/articles/000005725/wireless/legacy-intel-wireless-products.html.

[43] International Telecommunication Union. 2003. *Recommendation ITU-R M.1079-2: Performance and quality of service requirements for International Mobile Telecommunications-2000 (IMT-2000)*. Technical Report. ITU-R. http://www.itu.int/rec/R-REC-M.1079-2-200306-I/en Accessed: 2024-05-22.

[44] Phil Karn and Craig Partridge. 1987. Improving round-trip time estimates in reliable transport protocols. *ACM SIGCOMM Computer Communication Review* 17, 5 (1987), 2–7.

[45] Ike Kunze, Constantin Sander, and Klaus Wehrle. 2023. Does It Spin? On the Adoption and Use of QUIC's Spin Bit. In *Proceedings of the 2023 ACM on Internet Measurement Conference*. 554–560.

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

[47] Oliver Michel, Satadal Sengupta, Hyojoon Kim, Ravi Netravali, and Jennifer Rexford. 2022. Enabling passive measurement of zoom performance in production networks. In *Proceedings of the 22nd ACM internet measurement conference*. 244–260.

[48] Microsoft. 2024. *Azure Firewall TCP session behavior*. Microsoft Learn. https://learn.microsoft.com/en-us/azure/firewall/tcp-session-behavior Accessed: 2024-05-22.

[49] Microsoft. 2024. Monitor call and meeting quality in Microsoft Teams. https://support.microsoft.com/en-us/office/monitor-call-and-meeting-quality-in-microsoft-teams-7bb1747c-d91a-4fbb-84f6-ad3f48e73511. Accessed: 2024-05-22.
[50] Katsiaryna Mirylenka, Vassilis Christophides, Themis Palpanas, Ioannis Pefkianakis, and Martin May. 2016. Characterizing home device usage from wireless traffic time series. In *19th International Conference on Extending Database Technology (EDBT)*.
[51] Radhika Mittal, Vinh The Lam, Nandita Dukkipati, Emily Blem, Hassan Wassel, Monia Ghobadi, Amin Vahdat, Yaogong Wang, David Wetherall, and David Zats. 2015. TIMELY: RTT-based Congestion Control for the Datacenter. In *Proc. ACM SIGCOMM*. 537–550. doi:10.1145/2785956.2787510
[52] Nitinder Mohan, Lorenzo Corneo, Aleksandr Zavodovski, Suzan Bayhan, Walter Wong, and Jussi Kangasharju. 2020. Pruning Edge Research with Latency Shears. In *Proceedings of the ACM Workshop on Hot Topics in Networks (HotNets) 2020*.
[53] Owais. 2026. *MPEG-DASH Streaming: Complete Guide to Adaptive Video Delivery over HTTP*. https://antmedia.io/mpeg-dash-streaming-protocol/
[54] Vern Paxson, Mark Allman, Jerry Chu, and Matt Sargent. 2011. RFC 6298: Computing TCP's retransmission timer.
[55] Adithya Abraham Philip, Rukshani Athapathu, Ranysha Ware, Fabian Francis Mkocheko, Alexis Schlomer, Mengrou Shou, Zili Meng, Srinivasan Seshan, and Justine Sherry. 2024. Prudentia: findings of an Internet fairness watchdog. In *Proceedings of the ACM SIGCOMM 2024 Conference*. 506–520.
[56] Dipankar Raychaudhuri, Ivan Seskar, Gil Zussman, Thanasis Korakis, Dan Kilper, Tingjun Chen, Jakub Kolodziejski, Michael Sherman, Zoran Kostic, Xiaoxiong Gu, Harish Krishnaswamy, Sumit Maheshwari, Panagiotis Skrimponis, and Craig Gutterman. 2020. Challenge: COSMOS: A city-scale programmable testbed for experimentation with advanced wireless. In *Proc. ACM MOBICOM'20*.
[57] Ahmed Saeed, Nandita Dukkipati, Vytautas Valancius, Vinh The Lam, Carlo Contavalli, and Amin Vahdat. 2017. Carousel: Scalable Traffic Shaping at End Hosts. In *Proc. ACM SIGCOMM*. 404–417. doi:10.1145/3098822.3098852
[58] Sandvine. 2026. https://www.applogicnetworks.com/gipr-2026
[59] Matthew Sargent and Mark Allman. 2014. Performance within a fiber-to-the-home network. *ACM SIGCOMM Computer Communication Review* 44, 3 (2014), 22–30.
[60] Ranya Sharma, Nick Feamster, and Marc Richardson. 2024. A longitudinal study of the prevalence of wifi bottlenecks in home access networks. In *Proceedings of the 2024 ACM on Internet Measurement Conference*. 44–50.
[61] Taveesh Sharma, Tarun Mangla, Arpit Gupta, Junchen Jiang, and Nick Feamster. 2023. Estimating WebRTC Video QoE Metrics Without Using Application Headers. In *Proceedings of the ACM Internet Measurement Conference (IMC) 2023*.
[62] Bruce Spang, Veronica Hannan, Shravya Kunamalla, Te-Yuan Huang, Nick McKeown, and Ramesh Johari. 2021. Unbiased Experiments in Congested Networks. In *Proc. ACM Internet Measurement Conference (IMC)*. 80–95. doi:10.1145/3487552.3487851
[63] Bruce Spang, Shravya Kunamalla, Renata Teixeira, Te-Yuan Huang, Grenville Armitage, Ramesh Johari, and Nick McKeown. 2023. Sammy: Smoothing Video Traffic to Be a Friendly Internet Neighbor. In *Proceedings of the ACM SIGCOMM Conference 2023*.
[64] Srikanth Sundaresan, Walter De Donato, Nick Feamster, Renata Teixeira, Sam Crawford, and Antonio Pescapè. 2011. Broadband internet performance: a view from the gateway. *ACM SIGCOMM computer communication review* 41, 4 (2011), 134–145.
[65] Srikanth Sundaresan, Nick Feamster, and Renata Teixeira. 2015. Measuring the Performance of User Traffic in Home Wireless Networks. In *Proceedings of the Passive and Active Measurement Conference (PAM) 2015*.
[66] Srikanth Sundaresan, Nick Feamster, and Renata Teixeira. 2016. Home network or access link? locating last-mile downstream throughput bottlenecks. In *International Conference on Passive and Active Network Measurement*. Springer, 111–123.
[67] Simon Sundberg, Anna Brunstrom, Simone Ferlin-Reiter, Toke Høiland-Jørgensen, and Robert Chacón. 2024. Measuring Network Latency from a Wireless ISP: Variations Within and Across Subnets. In *Proc. ACM Internet Measurement Conference (IMC)*. 29–43. doi:10.1145/3646547.3688438
[68] Linus Torvalds. 2021. tcp_input.c – Linux kernel (v5.11-rc5). https://github.com/torvalds/linux/blob/2ab38c17aac10bf55ab3efde4c4db3893d8691d2/net/ipv4/tcp_input.c.
[69] Belma Turkovic, Fernando A Kuipers, and Steve Uhlig. 2019. Fifty shades of congestion control: A performance and interactions evaluation. *arXiv preprint arXiv:1903.03852* (2019).
[70] Ranysha Ware, Matthew K Mukerjee, Srinivasan Seshan, and Justine Sherry. 2019. Modeling BBR's interactions with loss-based congestion control. In *Proceedings of the internet measurement conference*. 137–143.
[71] David X. Wei, Pei Cao, and Steven H. Low. 2006. TCP Pacing Revisited. In *Proc. IEEE INFOCOM*.
[72] Michael Welzl, Wesley Eddy, Vidhi Goel, and Michael Tüxen. 2026. Pacing in Transport Protocols. Internet-Draft draft-irtf-iccrg-pacing-01, IRTF. https://datatracker.ietf.org/doc/draft-irtf-iccrg-pacing/
[73] Wireshark Foundation. 2024. Wireshark TCP Analysis: TCP Out-of-Order. https://www.wireshark.org/docs/wsug_html_chunked/ChAdvTCPAnalysis.html. Accessed: 2024-05-22.
[74] YouTube. 2026. System requirements and supported devices for YouTube. https://support.google.com/youtube/answer/78358?hl=en.
[75] Shuyue Yu, Thomas Koch, Ilgar Mammadov, Hangpu Cao, Gil Zussman, and Ethan Katz-Bassett. 2025. Internet Service Usage and Delivery As Seen From a Residential Network. *Proc. ACM Meas. Anal. Comput. Syst.* 9, 2, Article 41 (2025). doi:10.1145/3727133

A Appendix

A.1 Ethics

This work uses a published dataset and does not attempt to de-anonymize the dataset or understand behavior of individual users. During the experiment, YouTube temporarily shifted Columbia users from UDP to TCP sessions. In this well-provisioned network with short RTTs to YouTube servers, YouTube did not expect this shift to cause noticeable performance differences for users and was comfortable enabling it for the duration of the experiment. We believe that the real-world pacing experiment also raises no ethical issues, since previous studies show that pacing does not harm user experience [33, 63]. While the previous work demonstrates that pacing improves congestion metrics for the service adopting it, the impact on other services has not been studied due to limited visibility. Understanding this broader effect is essential for deployment, motivating the need for our study.

A.2 Dataset Sufficiency

To demonstrate that the dataset we study provides sufficient coverage and that a week-long time window can capture typical service usage behavior, we evaluate how service usage distributions estimated from limited data differ from those computed over the full four-week dataset. For example, the distribution for the first day might be 30% Netflix and 70% YouTube, while the full four-week distribution might be 40% Netflix and 60% YouTube (if there are only two services). Specifically, we randomly select X consecutive hours, compute the service usage distribution over those hours, and compare it to the distribution computed using all hours. We vary X from small values up to four weeks and quantify the difference using the Bhattacharyya distance, a standard measure of similarity between distributions. This process is repeated over 20 random selections of consecutive hours.

In Figure 17, the blue line shows the median distance across random selections as the dataset grows, and the red shading represents one standard deviation. We find that the median distance decreases rapidly as X increases and exhibits diminishing reductions beyond approximately 100 consecutive hours. This indicates that additional data beyond this point yields only marginal changes in the estimated service usage distribution, suggesting that roughly one week (168 hours) of data is sufficient to capture stable and representative usage patterns in the network under study. This is consistent with the operational experience of streaming services, which often compare week-over-week traffic to detect anomalies.

A.3 A Testbed Evaluation of Our Methodology

A.3.1 Testbed. While obtaining ground truths from the residential network is challenging due to limited visibility, we set up a testbed that simulates the topology of the residential access network and

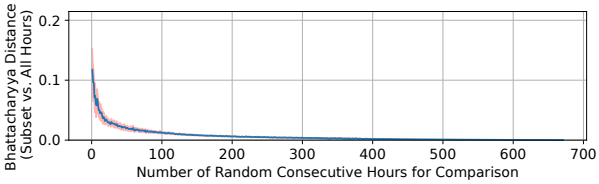


Figure 17: Convergence of service usage distributions with increasing observation time.

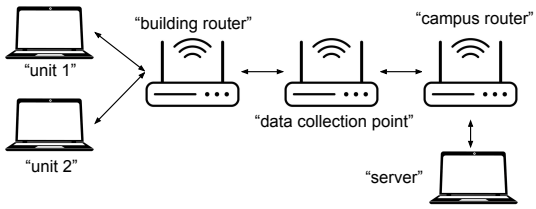


Figure 18: Topology of the controlled testbed. Labels indicate the role of each device in this simulated residential network.

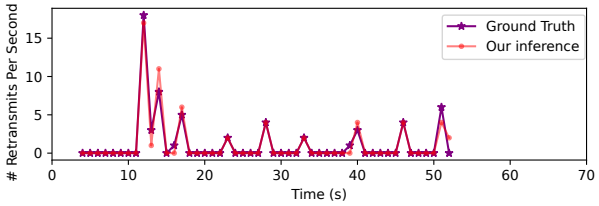


Figure 19: Our inference of TCP retransmits closely matches the ground truth.

conduct controlled experiments to evaluate the accuracy of our methodology described in §3.

As illustrated in Figure 18 (from right to left), we have a router that represents *"a campus router"* connecting the residential network to the Internet; a laptop is connected to this router and acts as *"a server"*; a second router is connected to the first router and simulates *"the data collection point"* (the aggregation switch plus the data collection servers); a third router is connected to the second router and acts as *"a building router"*; and two laptops are connected to the building router and represent two different *"units"*. Because the routers are physically close, we introduce a 10 ms bidirectional delay at the building router to better approximate realistic network settings. We use OpenWrt routers so that we can ssh into them and run commands. We also use 20 ms as the base RTT when inferring performance and overlapping packets.

In each experiment, the *"server"* sends data to the client using iperf. The *"data collection point"* captures packet traces using tcpdump. To configure the building's maximum network capacity, the *"building router"* throttles traffic to 10 Mbps and maintains a first-in, first-out queue of 100 packets using Linux's `tc`. We also use `tc` to monitor the queue status, taking a snapshot every second. Our testbed uses a lower-bandwidth link setting due to hardware constraints of our commodity home routers, which make reliable packet capture at multi-Gbps speeds challenging. We acknowledge that traffic patterns in the real residential access network differ from those in this testbed and that real services behave differently from iperf traffic. Nevertheless, the controlled setting provides ground truth for evaluating our inference method across a range of network utilization levels.

To create different scenarios representing varying levels of building network utilization, the server sends X Mbps to one *"unit"* using UDP, and sends TCP Reno traffic to the other *"unit"*, with its sending rate determined by TCP Reno's congestion control. We conduct 100 trials at each UDP iperf throughput: $X = 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11$ Mbps. In summary, we run a total of 1,100 experiments. For each experiment, we collect the packet traces from the *"data collection point"*, the tc queue logs from the *"building router"*, and the iperf logs from the *"unit"* sending the TCP request. We apply our method (§3) to the packet traces and compare our inference results with the ground truths from the other two logs. To preserve the integrity of our metric evaluation, we excluded 172 experimental trials where RTT observations were absent for more than 25% of the experiment duration. We impose this threshold because insufficient data points preclude the reliable calculation of R-squared scores. This filtering applies exclusively to the two highest UDP rates ($X = 10, 11$ Mbps), where the UDP traffic saturates the bottleneck and starves the TCP traffic used by our inference methodology.

A.3.2 Our inference of TCP retransmits matches the ground truth.
Ground truth of retransmits: The TCP iperf log records the amount of transferred data, bitrate, the number of retransmissions, and the congestion window at one-second intervals. We use the number of retransmissions per second as the ground truth.

Our inference of retransmits: Applying the method described in §3.3 to the packet traces collected from *"the data collection point"* in the testbed, we identify the TCP packets that are retransmitted and compute the number of TCP retransmits per second.

Across all experiments, the R-squared values between our inferred and ground-truth retransmissions per second have a median of 0.98. In Figure 19, we show our inferred TCP retransmits (red line) and the ground truth (purple line) for a single experiment (with UDP iperf sending traffic at 8 Mbps). The plotted lines start at time > 0 because we issue the TCP iperf request a few seconds after the beginning of packet trace collection. The two lines align closely, and the R-squared value between them is 0.94, indicating that our inference closely matches the ground truth for this experiment.

Although we can observe the number of dropped packets using tc at *"the building router"*, these drops include UDP packet losses. Since our method does not infer UDP packet loss from transport headers as it does for TCP, we do not make this comparison because it would not provide meaningful results. In addition, we do not have ground truth for queueing delays. However, we believe that the validation of TCP retransmits suggests that our congestion-related metrics are likely accurate as they are computed using the same TCP header information in a similar manner.

A.3.3 Our inference of overlapping packets matches the ground truth.
Ground truth of buffer occupancy: Since our OpenWrt routers only support second-level time precision, we use tc at each second to show the number of packets in the queue of *"the building router"*. The value reflects the number of packets that could impact a packet arriving at that moment.

Our inference of overlapping packets: As described in §3.4, we infer access RTTs and in-flight time intervals for packets in the collected packet traces. For each packet, we can compute the number of overlapping packets based on in-flight time intervals.

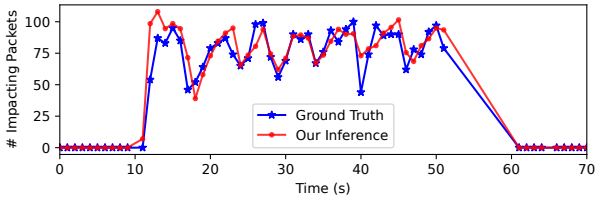


Figure 20: Our inference of overlapping packets aligns with the ground truth.

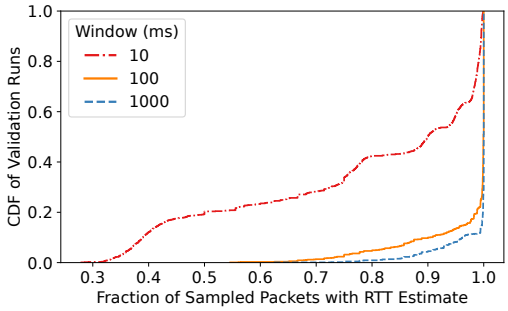


Figure 21: CDF of RTT coverage across validation runs for different propagation windows.

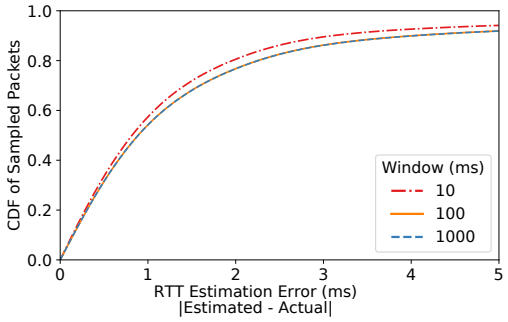


Figure 22: CDF of RTT estimation error (|Estimated – Actual|) across packets for different propagation windows.

Because the OpenWrt routers provide queue snapshots with second-level resolution, we cannot directly compare our inference of overlapping packets with ground truths of buffer occupancy. To address this, we infer the *buffer occupancy at a specific timestamp* as the number of packets whose in-flight intervals include that timestamp. To generate a comparable per-second value, we sample this inferred occupancy at 0.1 s intervals (10 samples per second) and use the median of these samples as the value for that second.

Across all experiments, the R-squared values between our inferred and ground-truth buffer occupancy per second exceed 0.82, with a median of 0.98. In Figure 20, we compare our inferred overlapping packets (red line) with the ground truth (blue line) for a single experiment (with UDP iperf sending at 8 Mbps). The two lines align closely, and the R-squared value between them is 0.94. Since both the buffer occupancy and the overlapping packet counts are derived from the same inferred in-flight intervals, we believe that this high correlation provides evidence for the accuracy of our overlapping packet inference.

A.3.4 RTT propagation window selection and inference accuracy. **Propagation window selection.** We perform a sensitivity sweep across RTT propagation windows of 10 ms, 100 ms, and 1 s. For each validation run, we withhold a random 10% sample of packets with directly observed RTTs as a test set and treat their RTTs as missing. We then apply our RTT propagation method and compare the resulting estimates against the withheld RTTs. This held-out evaluation allows us to test the propagation procedure we use to assign RTTs to packets in our Columbia traces without direct RTT observations, including UDP packets. We evaluate both estimation error (the absolute difference between estimated and observed RTT) and coverage (the fraction of test packets with an RTT estimate).

As illustrated in Figure 21, larger propagation windows substantially improve coverage. A 10 ms window achieves a mean coverage of 79%, whereas 100 ms increases this to 97%. The 1 s window achieves 99% mean coverage. In contrast, as shown in Figure 22, estimation error changes little with the propagation window: median errors are 0.82 ms at 10 ms and 0.89 ms at both 100 ms and 1 s. The 80th-percentile error increases by only 0.29 ms from 10 ms to 1 s and changes negligibly between 100 ms and 1 s. Therefore, we use a 1 s propagation window to maximize coverage while maintaining similar accuracy.

RTT inference accuracy with the 1 s propagation window. With the selected 1 s window, the median absolute error is 0.89 ms (1% relative error), while the 80th and 90th percentile errors remain at 2.26 ms (3%) and 4.04 ms (5%), respectively. Overall, 77% of estimates are within 2 ms of the observed RTT values. These results show that our inference methodology introduces relatively small estimation error. Nevertheless, because this evaluation uses controlled testbed traffic, the coverage–accuracy tradeoff may differ under real traffic loads and network conditions. Further evaluating propagation-window choices under operational conditions is an interesting direction for future work.

A.4 Sensitivity to the Overlap Definition

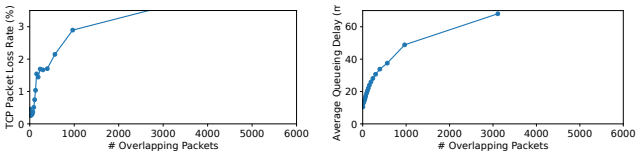
Our primary overlap metric counts packets whose in-flight intervals overlap with that of the packet under study, including packets captured both before and after it. We use this broader definition because we do not assume globally FIFO packet ordering across the access path. As discussed in §3.4, however, this definition may count some later-captured packets that did not contribute to the packet's congestion. To evaluate the sensitivity of our results to this choice, we consider a more restrictive definition that counts only overlapping packets captured before the packet under study. This alternative reduces absolute overlap counts, as expected, but our main qualitative findings remain unchanged.

Figure 23 shows the relationship between unit-level overlaps and congestion under this alternative definition. Both TCP packet loss and average access queueing delay continue to increase with unit-level overlaps, consistent with our finding in §4.2 that packets with more sharing are more likely to experience congestion.

Similarly, Figure 24 compares unit-level and building-level overlap counts. While the absolute values of both are lower under the alternative definition, their relative relationship remains similar to that under our original overlap definition. Thus, our finding in §5.1 that sharing is disproportionately concentrated within the same residential unit also remains unchanged.

Service	Cross-Service Attributed Delay (%)	Cross-Service Attributed Loss (%)
youtube	0.42	0.21
AS: apple	0.27	0.16
icloud	0.25	0.21
applecdn	0.21	0.15
instagram	0.21	0.12
2label: apple.com	0.20	0.14
2label: googleapis.com	0.18	0.10
netflix	0.17	0.06
microsoftcloud	0.16	0.06
AS: google	0.15	0.08

Table 3: Top 10 services by cross-service attributed queueing delay. Cross-service attributed queueing delay (loss) is reported as a percentage of total queueing delay (loss) across all traffic.



(a) Packet loss rate **(b) Average queueing delay**

Figure 23: Correlation between the number of unit-level overlaps per packet and congestion metrics under the alternative definition.

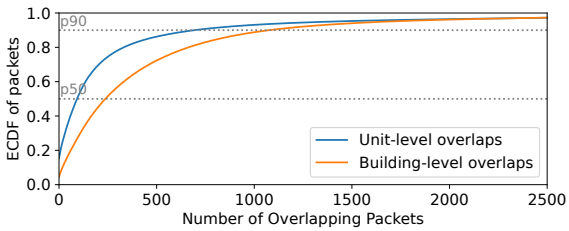


Figure 24: Distribution of unit-level and building-level overlap counts under the alternative overlap definition.

A.5 Attribute Congestion to Shared Network Resources

To better understand how congestion is associated with traffic sharing network resources, we perform an attribution analysis that distributes packet loss and queueing delay across overlapping packets. We acknowledge that this analysis is approximate, as it assumes that congestion can be distributed evenly across packets with overlapping in-flight intervals. The true contribution of individual packets may depend on factors we cannot observe, such as queue position, scheduling, and buffer dynamics. Yet it provides a quantitative way to estimate how congestion is distributed across sharing traffic.

Specifically, we attribute the loss and queueing delay experienced by each packet evenly across its overlapping packets. For example, consider packets A, B, C, X, D, E, and F, where X is a lost packet and its in-flight interval overlaps with all seven packets, including itself. We attribute one seventh of X's loss to each of these packets. We apply the same procedure to queueing delay.

For each packet, we define its *attributed loss* as the sum of the fractional loss attributed to it by all packets whose in-flight intervals overlap with it. Similarly, we define its *attributed delay* as the sum of the fractional queueing delay attributed to it by overlapping packets. Since each packet is associated with a service, we aggregate these packet-level attributions by service and distinguish between same-service and cross-service attribution. An attributed fraction is classified as same-service when the packet experiencing the loss or delay and the packet receiving the attribution belong to the same service, and as cross-service otherwise.

We find that cross-service attributions account for only 5.2% of total packet loss and 8.3% of total queueing delay. Thus, more than 90% of both attributed loss and delay is associated with packets from the same service. This result is consistent with our earlier observation that network resource sharing in this network occurs predominantly within services rather than across services.

We also examine which services account for the most cross-service attributed congestion during the baseline week (week 3). We report the top ten services by cross-service attributed queueing delay in Table 3. The percentages reported are relative to the total queueing delay and total packet loss across all traffic, rather than only to the cross-service portions of these metrics. YouTube has the largest cross-service attributed queueing delay, accounting for 0.42% of total queueing delay and 0.21% of total packet loss. Netflix accounts for 0.17% of total queueing delay and 0.06% of total packet loss through cross-service attribution. Several Apple-related services also rank highly, which may partly reflect their prevalence across residential units and therefore greater opportunities to overlap with other traffic. Overall, cross-service attribution is distributed across many services, and these cross-service attribution values reflect both a service's prevalence in the network and its traffic delivery patterns: services carrying more traffic have more opportunities to overlap with other services, while differences in burstiness and sending behavior can also affect the amount of attributed congestion.

A.6 Behavior of Top 50 Services

Table 4 provides a complete list of the top 50 services and their statistics, ranked by total traffic volume. As described in §3.2, "2label: " denotes service names derived from the last two labels of the associated domain, and "AS:" denotes the organization of the associated autonomous systems. For each service during the baseline week (week 3, when both YouTube and Netflix run without their pacing treatments), we report: its percent of overall traffic volume, its own TCP packet loss rate, the average queueing delay experienced by the service, the average number of same-service and cross-service overlaps for each of its packets, and the fraction of unit-hours containing packets from this service.

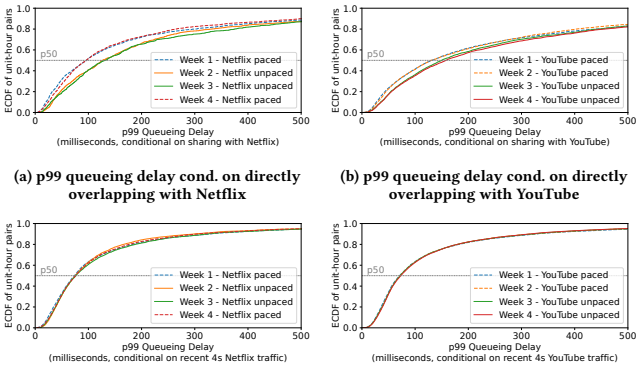
A.7 Kolmogorov-Smirnov Tests for Pacing Effects

We use two-sample Kolmogorov-Smirnov (KS) tests to assess the paced-versus-unpaced distributional differences reported in §6.3 and §6.4. For each service and metric, we compare the unit-hour observations from weeks when that service uses its pacing treatment with those from weeks when it does not, using the same filtering and conditioning as in the corresponding analyses.

Impact on the paced services. For the analysis in §6.3, we test the paced-versus-unpaced differences in self-overlap intensity,

Service (Ranking)	Traffic Volume (%)	TCP Packet Loss Rate (%)	Avg Queueing Delay (ms)	Avg # Overlaps (Same-Service)	Avg # Overlaps (Cross-Service)	Active Unit-Hours (%)
youtube (1)	13.0	0.58	22.5	410	6	45.5
steam (2)	6.9	3.76	41.3	4631	13	4.6
netflix (3)	6.5	0.25	22.1	268	6	17.3
applecdn (4)	3.6	0.95	20.6	539	15	72.0
disneyplus (5)	3.1	0.84	19.5	460	6	2.8
icloud (6)	2.9	1.16	23.8	474	16	75.9
2label: apple.com (7)	2.8	0.79	15.2	425	11	78.9
instagram (8)	2.7	0.04	24.5	311	14	30.2
primevideo (9)	2.3	0.86	23.6	427	7	9.5
applestore (10)	1.9	0.76	19.8	518	17	63.2
AS: google (11)	1.8	0.65	23.4	466	13	62.4
AS: apple (12)	1.7	0.63	24.5	311	18	74.4
cloudfront (13)	1.5	1.09	16.6	439	13	27.0
2label: akamai.net (14)	1.5	1.48	26.2	543	32	26.4
tiktok (15)	1.3	0.92	20.3	305	12	15.0
2label: pv-cdn.net (16)	1.2	0.64	22.1	512	5	1.0
peacock (17)	1.0	0.85	21.8	497	8	1.1
playstation (18)	1.0	2.29	81.8	1130	18	1.3
hbomax (19)	1.0	0.56	24.5	388	6	2.5
appletv (20)	0.9	0.68	23.7	565	5	0.6
AS: amazon.com (21)	0.9	1.06	25.9	311	14	85.3
2label: akamaized.net (22)	0.9	2.62	21.3	439	23	9.8
zoom (23)	0.9	0.34	15.4	183	11	20.5
AS: akamai international b.v (24)	0.9	1.53	23.2	454	19	32.0
2label: akadns.net (25)	0.8	0.81	19.5	463	12	67.4
AS: microsoft (26)	0.8	0.54	28.5	225	27	77.7
facebook (27)	0.8	0.16	21.4	288	15	43.7
xiaohongshu (28)	0.8	3.65	26.2	286	9	6.1
2label: fastly.net (29)	0.7	0.64	25.1	543	15	24.0
2label: real-debrid.com (30)	0.7	1.30	6.7	776	8	0.1
AS: fastly (31)	0.7	0.68	23.0	564	20	19.6
AS: cloudflare (32)	0.7	0.75	21.1	286	15	25.5
twitter (33)	0.7	0.89	16.1	215	7	10.9
reddit (34)	0.7	0.38	25.2	613	11	10.3
twitch (35)	0.6	0.96	7.8	114	7	0.6
microsoftcloud (36)	0.6	1.81	31.5	486	39	55.1
spotify (37)	0.6	1.49	25.5	391	13	27.1
googleupdates (38)	0.6	0.67	23.8	553	12	26.0
AS: datacamp (39)	0.6	0.52	23.6	395	25	11.0
AS: facebook (40)	0.6	0.11	22.6	231	19	24.4
gmail (41)	0.5	0.83	16.9	59	13	59.4
2label: akamaihd.net (42)	0.5	1.01	23.8	384	17	4.5
2label: googleapis.com (43)	0.5	0.72	24.6	325	27	74.0
bilivideo (44)	0.5	1.29	12.7	99	37	3.0
googleusercontent (45)	0.5	0.72	21.1	406	33	50.6
hulu (46)	0.4	1.12	14.1	442	9	3.1
2label: google.com (47)	0.4	0.46	25.0	325	32	71.4
newyorktimes (48)	0.4	0.70	31.9	741	15	8.7
nvidiagrid (49)	0.3	0.00	26.6	280	7	0.4
AS: clouvider (50)	0.2	0.18	22.7	341	13	1.6

Table 4: Top 50 services by traffic volume.



(a) p99 queueing delay cond. on directly overlapping with Netflix

(b) p99 queueing delay cond. on directly overlapping with YouTube

(c) p99 queueing delay cond. on potentially overlapping with Netflix

(d) p99 queueing delay cond. on potentially overlapping with YouTube

Figure 25: p99 access queueing delay of other packets that directly overlap with Netflix or YouTube (top) and potentially overlap with Netflix or YouTube (bottom), over all 4 weeks. Pacing reduces tail queueing delay when direct sharing occurs, while having little effect on potentially overlapping traffic overall.

TCP packet loss rate, and average queueing delay. The KS statistics for Netflix and YouTube are, respectively, 0.4430 and 0.1843 for self-overlap intensity (Fig. 11), 0.2037 and 0.1208 for TCP packet loss rate (Fig. 12), and 0.2780 and 0.2134 for average queueing delay (Fig. 13). All comparisons have $p < 0.0001$.

Impact on neighboring services. For the analysis in §6.4, we test the paced-versus-unpaced differences in cross-service sharing and congestion experienced by neighboring traffic. For the sharing metrics, the KS statistics for Netflix and YouTube are, respectively, 0.0823 and 0.0211 for overlap probability (Fig. 14a, Fig. 14b), 0.2182 and 0.0737 for overlap intensity (Fig. 14c, Fig. 14d), and 0.0955 and 0.0411 for expected overlaps (Fig. 14e, Fig. 14f). These comparisons have $p < 0.0001$.

For congestion among packets directly overlapping with Netflix or YouTube, the KS statistics are 0.0902 and 0.0373 for TCP packet loss rate (Fig. 15a, Fig. 15b) and 0.1867 and 0.0926 for average queueing delay (Fig. 15c, Fig. 15d), respectively. These comparisons also have $p < 0.0001$.

For potentially overlapping traffic, the differences are smaller. The KS statistics for TCP packet loss rate are 0.0342 and 0.0101 for Netflix and YouTube, respectively, with $p < 0.0001$ and $p = 0.0013$ (Fig. 16a, Fig. 16b). For average queueing delay, the corresponding KS statistics are 0.0480 and 0.0236, respectively, with $p < 0.0001$ for both comparisons (Fig. 16c, Fig. 16d).

A.8 Pacing Effects on Tail Queueing Delay

In §6.4, we use the average access queueing delay within each unit-hour to characterize congestion. We additionally examine tail queueing delay to understand whether the observed effects of pacing also hold in the tail of the queueing-delay distribution. Specifically, we repeat the analyses in Figure 15 and Figure 16 using the 99th-percentile (p99) access queueing delay within each unit-hour, with the same filtering and conditioning.

Figure 25 shows qualitatively similar results. For packets that directly overlap with Netflix or YouTube, pacing reduces p99 queueing delay, consistent with the average queueing-delay results in Figure 15. For potentially overlapping packets, p99 queueing delay changes little across paced and unpaced weeks, consistent with Figure 16. In summary, the overall takeaway remains unchanged: pacing reduces tail queueing delay when direct sharing occurs, while having little effect on potentially overlapping traffic overall. We also repeat the analysis using p90 and p95 access queueing delay and observe the same qualitative patterns (omitted for brevity).