# NETHINT: Monitoring Local Network Interference With Passive Measurements

Michael Welzl
*University of Oslo*
Oslo, Norway
michawe@ifi.uio.no

Safiqul Islam
*Oslo Metropolitan University*
Oslo, Norway
safiqul.islam@oslomet.no

Kristjon Ciko
*University of Oslo*
Oslo, Norway
kristjoc@ifi.uio.no

Petter Barhaugen
*Blank AS*
Oslo, Norway
petter.barhaugen@gmail.com

Oskar Haukebøe
*University of Oslo*
Oslo, Norway
oskah@ifi.uio.no

***Abstract*—The "quality" of a home Internet connection is hard to quantify, as it is a subjective measure encompassing many factors. One of them, the detection of local (as opposed to remote) traffic interference, is not provided by existing tools. We address this with NETHINT, a tool to detect whether users and their applications disturb each other in a home network. NETHINT identifies traffic correlations via passive online latency monitoring. We explain the inner workings of NETHINT, and show how it can be successfully applied for live online root cause analysis of delay in a local network.**

***Index Terms*—Measurement, monitoring, shared bottleneck**

## I. Introduction

The "best effort" service model has played an important role for the success of the Internet [1], [2]. However, it has the downside that problems can spontaneously occur: conversations can suddenly be interrupted for a few seconds, become unintelligible, or heavily delayed; players of online games may lose points due to network lag; web pages can sometimes load very slowly [3], [4].

When problems do happen, the network's complexity makes it hard to understand *why*—in particular, these reasons are often not transparent to end users [5]. Would a home network truly become "faster" (i.e., show less noticeable latency) when upgrading the capacity of the access link? Would a work video conference improve if all children in the household would stop playing games? Latency can have many different reasons—application delays, congestion in large buffers, having to wait for a transmission opportunity at a wireless link, and local retransmissions due to link noise, to name but a few [6].

It is not practical to provide users or network administrators with all of the details regarding the composition of end-to-end latency; however, it may be enough to offer information that can be acted upon. Specifically, it matters whether traffic interference is (or was) **remote** or **local**. *Remote* problems are hard to circumvent: they *might* occur within the network of the Internet Service Provider (ISP), in which case it could be worthwhile to inform the ISP or change the contract. *Local* problems, however, can sometimes be solved: if certain types of parallel Internet usage have been known to cause problems in the past, this may be a reason to avoid such parallel usage in the future. On the other hand, if problems can be identified as remote, there is no indication of a need to avoid such parallel Internet usage. Such information could be valuable to a user who shares her Internet access with others in a household. A residential ISP could potentially also benefit, to enhance the management of dispute calls in its call center.

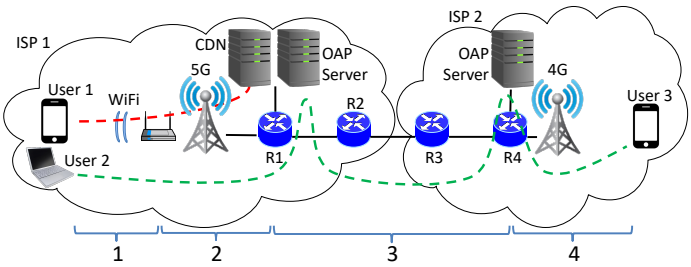


Fig. 1: Example scenario where problems are hard to diagnose. The blue brackets indicate path segments 1-4.

Figure 1 shows an example scenario. Here, user 1 and user 2 live in the same household, and they access the Internet via the same WiFi Access Point (AP) and 5G Fixed Wireless Access or 5G Mobile Access. User 3, who is remote, accesses the Internet via a 4G cellular connection. User 3 has a different ISP (ISP 2) than users 1 and 2 (ISP 1). User 1 interacts with a social media application (video download and upload from/to a Content Distribution Network (CDN) server), while user 2 has a video call with user 3. The data transfer from/to user 1 traverses path segments 1 and 2, whereas the data transfer from/to user 2 traverses all path segments 1-4, including two servers of an Online Application Provider (OAP).

**Let us assume that user 2 in this scenario experiences quality fluctuations and latency. *Why could this happen?*** Possible reasons include:

1) Congestion, from interaction with user 1 in path segment 1;
2) Other disturbances in path segment 1 (interference from overlapping channels, signal noise, ...);
3) Congestion, from interaction with user 1 in path segment 2;
4) Other disturbances in segment 2 (5G channel fluctuations or uplink competition with other endpoints);
5) Congestion in path segment 3 (this is not very likely in well configured backbone networks, but nevertheless possible [7]), or overload in one of the OAP servers;
6) Disturbances in path segment 4 (signal noise, or uplink competition with multiple other endpoints).

It is imaginable that user 2 would attempt to "fix the problem"

979-8-3315-2096-0/25/$31.00 ©2025 IEEE

by telling user 1 to stop using the social media application, but this would only help in two out of the above six cases (cases 1 and 3); **which case really matches the current situation is unknown.** Knowing whether Internet delay is caused by local interference between applications could therefore be helpful in practical real-life situations. This paper addresses this need with the following contributions:

- We introduce and explain the *NETHINT (NETwork Home INTerference)* tool which, from passive online latency monitoring, produces suitable inputs to identify whether local or remote interference has occurred. NETHINT is open source and available online.[1]
- We show how to carry out a correlation analysis using the data from NETHINT with a simple and efficient (applicable to live monitoring) approach.
- We report test results from a network testbed with a variety of traffic profiles, to identify whether a bottleneck is shared or not, on the wired or wireless segment of our network topology. We find that the correlations can be quite clear, and that they can therefore serve as input for management decisions in a home network (taking corrective action).

After discussing related work in the next section, we will describe NETHINT in Section III, and evaluate its usage with correlation analysis in live monitoring in Section IV. Section V concludes.

## II. Related Work

Today, to the best of our knowledge, no open source passive online monitoring approach exists that was shown to provide the needed inputs for the correlation analysis that we have initially discussed—to not just offer a measure of latency, but give an indication of interference between applications. Related work can be categorized as i) commercial tools that give hints about the local network, or a path's connection "speed" or latency (but not reasons for latency experienced by specific applications), and ii) academic work on network measurements.

The scope of commercial measurement techniques does not usually extend further than the red line in Figure 1. There are tools that operate locally (segment 1 in the figure), e.g., sniffing tools such as Wireshark, which provide an in-depth look into traffic visible to the network interface, or WiFi analyzers, e.g., the "Wireless Diagnostics" tool that is shipped in macOS. However, it is not always clear that the local network is indeed the bottleneck for a certain application, and hence such tools draw an incomplete picture.

Then, there are tools that measure end-to-end throughput, and, more recently, also latency, between the user equipment (UE) and a measurement server. This server is typically placed close to the user in order to attain a meaningful result—i.e., a path like the red line in Figure 1 is being measured. Well-known examples of the latter are speedtest.net, fast.com by Netflix and dslreports.com. Due to their limited scope, these tools also draw an incomplete picture. Latency is identified, but its root cause is not determined. It is now widely known that Internet latency is often caused by large unmanaged buffers [4], but this is not always the case (e.g., on wireless links, delay can occur for various other reasons).

Various ways to report the "quality" of a network connection in relation to latency are currently under discussion in the Internet Engineering Task Force (IETF), e.g. by describing an expected delay distribution [8], [9]. The recent metric called RPM (RTTs Per Minute) is designed to increase user comprehension of delay times with an intuitive "bigger is better" benchmark which seeks to measure latency *under load*, i.e. when a "bloated" buffer would supposedly be filled [10]. These are steps in the direction of identifying the root cause of delays. However, even when the root cause is known, it is important to understand *where* along an end-to-end path a bloated buffer is located (the scope of NETHINT).

The academic network measurement community has always relied on a wealth of possibilities to obtain as much data as possible. It is not uncommon for tools to run for months, actively probing the Alexa Top 1 Million web pages, or sending probes towards all announced BGP prefixes, from tens or even hundreds of vantage points. Such work usually aims to draw general conclusions about the Internet, often with worldwide scope. NETHINT's goal is live monitoring, with no active traffic generation, and being able to pinpoint the origin of latency. This means that only a subset of academic work is related—tools that are passive, and fast enough for monitoring feedback (i.e., not operating at a timescale of days, for example). We briefly discuss three categories of such work:

*1) Shared Bottleneck Detection (SBD):* Correlations between performance measurements (one-way delay [11], or several summary statistics [12]) are determined, in order to identify if traffic has traversed a common network bottleneck. The Round-Trip Time (RTT) can also be used [13], and this input can be obtained from passive traffic measurements (e.g., using PPing[2] or ePPing [14]). On this basis, flows are then grouped—in [11], using an extended version of the "Chinese Whispers" clustering algorithm.

SmartSBD [15], an approach within the SBD category, leverages supervised learning algorithms to detect whether Multipath TCP (MPTCP) subflows share the same bottleneck link. During the training phase, SmartSBD collects system logs from MPTCP hosts in real-world heterogeneous networks and trains a classifier which, in the runtime phase, predicts the sharing bottleneck condition of MPTCP subflows. While SmartSBD maximizes the throughput and improves fairness, it is tightly coupled with the need of MPTCP congestion control and also needs to keep record of state variables such as RTT samples, congestion state and ACK sequences for all ACK packets for each flow, which can result in a significant amount of data in high-speed networks with a large amount of flows.

FlowBot [16] is another learning-based solution in the realm of SBD algorithms which aims to improve the co-

[1]https://github.com/petternett/NETHINT

[2]https://github.com/pollere/pping

bottlenecks detection for large-scale applications in Content Delivery Networks (CDNs).

Various SBD algorithms utilize other signals such as inter-packet arrival time [17], congestion interval variance [18], and ECN [19] to detect shared bottlenecks in the network.

NETHINT is not an SBD mechanism per se, but it is complementary in that it collects the raw data that can be fed into an SBD mechanism. In Section IV, we discuss a simple correlation analysis of NETHINT's output using the Pearson correlation coefficient [20]—such analysis could benefit from other, more sophisticated SBD methods.

*2) Time-Series Latency Probing (TSLP):* This has been used in, for example, [7] and [21] to try to identify the location of Internet queuing delay. Fundamentally, all of these approaches determine delay on a link by sending TCP probes with different Time-To-Live (TTL) values, where probes with TTL values N and N+1 reflect the delay difference between the "near end" and the "far end" of the Internet link at TTL=N.

*3) Available bandwidth and bottleneck capacity estimation:* These are mechanisms to examine the effect of the network bottleneck on a given sequences of packets—packet pairs, "trains", or a pattern called a "chirp"—to deduce the bottleneck's capacity or its available bandwidth (see, for example, [22] and references therein). Such an approach could be used to make NETHINT more reliable. For example, if an extended NETHINT would find that a particular bottleneck's capacity is underutilized, flows traversing it would probably not influence each other on the same bottleneck. If a latency correlation analysis (or any other SBD method) says otherwise, it is probably a false positive.

*Summary:* There is a disconnect between commercial tools on the one hand, which provide "live" information but do typically not fit our envisioned scope, and most academic work on the other hand, where richer information is obtained, but the goal is long-term data collection.

While PPing and ePPing passively measure latency between two end hosts, NETHINT determines whether a bottleneck exists locally or somewhere else on the network path by comparing flows originating from different end hosts. Also, different from NETHINT, the delay measurement method in PPing and ePPing is exclusively based on Timestamps for the sake of simplicity [14], which can render the output less reliable (we will discuss this in the next section).

In contrast to packet sniffing tools like Wireshark, NETHINT only collects and parses data that are required to infer knowledge about the local network. For example, by filtering only valid RTTs to be displayed, the user is presented with more accurate and easily readable data than with other similar tools that can be difficult to navigate if users don't know exactly what they are looking for.

## III. The NETHINT Tool

NETHINT is a python program that uses the `scapy` library to listen for packets and process them according to their transport protocol. It finds reliable delay sources, detects lost segments and retransmissions, presents the data visually and saves it to log files in JSON format. We have used these log files for the evaluation that is discussed in the next section.

TABLE I: Delay sources used in NETHINT

| Delay source category | Outgoing data flow | Incoming data flow |
|---|---|---|
| TCP SEQ/ACK based pairing | - RTT of data packets<br>- Conn. establ. packets<br>- Conn. term. packets | - Conn. establ. packets<br>- Conn. term. packets |
| TCP Timestamps based pairing | - RTT of retrans. data pkts | - OWD from server |
| QUIC | - Initial and handshake<br>- Spin bit [23] | - Initial and handshake<br>- Spin bit [23] |

NETHINT is primarily intended for operation on a local WiFi network. For example, in the scenario in Figure 1, it could run on the devices of users 1 or 2, the AP, or on another device in the same WiFi network (not shown in the figure) that is dedicated to measurements only. The latter may be the most practical way to operate it, as NETHINT needs to "see" all the local WiFi traffic, i.e. it must put the wireless network interface card (NIC) into monitor mode[3], which is not supported by all WiFi NICs. In relation to the other delays measured by NETHINT, the WiFi physical propagation delay is minuscule; thus, such a device will see packets at essentially the same time as if it runs on the end user devices themselves.

Passively monitoring traffic and trying to deduce latency can only be done with protocols that carry out handshakes. This rules out UDP. Latency estimations can, however, be implemented for protocols running atop UDP, such as the Real-time Transport Protocol (RTP) or QUIC. NETHINT supports TCP and QUIC. Our goal was to capture the on-path delay as accurately as possible; to this end, NETHINT uses a mix of delay sources, which are shown in Table I.

**TCP SYN/FIN packets (Table I line 1):** For TCP, we can rely on a host to immediately reply to packets with a set SYN or FIN flag, which allows us to use all such packets for RTT estimation. We do this by monitoring the sequence (SEQ) and acknowledgment (ACK) numbers of packets.

**TCP original outgoing data packets (Table I line 1):** We treat *original* outgoing data packets similar to packets that carry the SYN or FIN flag. We note, however, that it is inevitable that some of these measurements will include the delay of the "delayed ACK" timer on the remote side. Also, we require these packets to contain data. Outgoing TCP packets that neither carry a SYN or FIN flag nor contain data do not provoke an acknowledgment from the peer (such packets are "pure ACKs", and there are no "ACKs-of-ACKs" in TCP).

**TCP retransmissions of outgoing data packets (Table I line 2, column 2):** We cannot use SEQ-ACK pairing on retransmitted data packets due to the well known "retransmission ambiguity" problem of TCP (when a packet is retransmitted, it is not clear whether a following matching ACK was caused by the retransmission or by the original data packet). This prob-

[3]Promiscuous mode is not enough, as it only allows access to all traffic to/from the NIC.

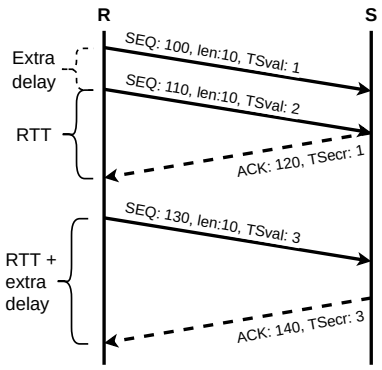


Fig. 2: For the first RTT sample, delayed ACKs cause extra delay only when relying on timestamps (hence we use sequence numbers when possible). For the second RTT sample, extra delay from delayed ACKs is inevitable. $R$: Receiver, $S$: Sender.

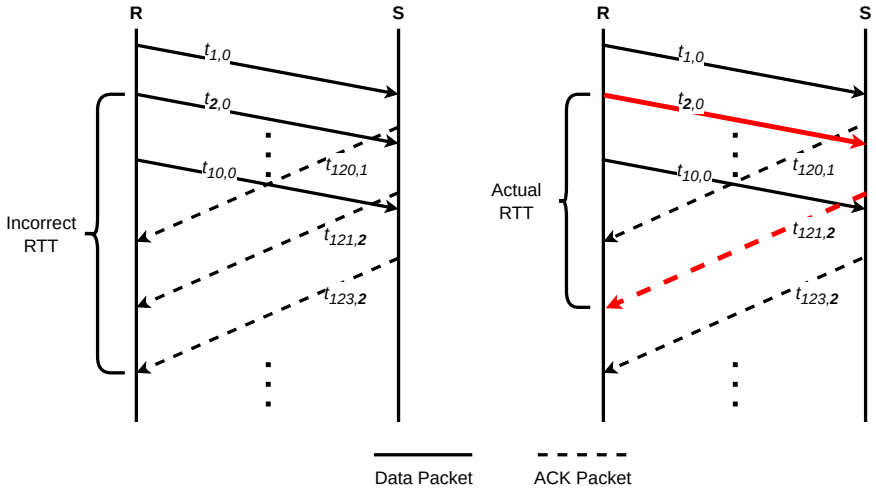


Fig. 3: When returning "Timestamp Echo Reply" (TSecr) Timestamp values are equal, the first must be taken to ensure correctness. Timestamps are written in the format: $t_{TimestampValue(TSval),TSecr}$. $R$: Receiver, $S$: Sender.

lem has been addressed with the TCP Timestamps option [24], which therefore also lends itself for passive delay estimation.

Relying on this option comes with three major limitations. First, it is not enabled in all hosts, which limits its applicability. Second, since the RTT estimate is used to calculate the retransmission timeout in TCP, Timestamps *always* include the delayed ACK timer. As Figure 2 illustrates, for some samples, this renders the outcome less precise for the "on-the-wire" delay that we care about. NETHINT therefore relies on SEQ/ACK pairing for original outgoing data packets, using timestamps only for retransmissions (for which SEQ/ACK pairing can be incorrect). In contrast, PPing and ePPing only rely on timestamps for the sake of simplicity, but the result can be less precise; this is documented in the appendix of [14].

Third, the update frequency for the timestamp value is limited, which means that we can unambiguously identify more packets with SEQ/ACK pairing than with Timestamps pairing. Due to the limited timer granularity, it is possible for multiple similar "Echo Reply" (TSecr) values to arrive at the monitoring host that passively determines the RTT. Thus, *when* we use Timestamps, we only utilize the first new timestamp in such situations. Figure 3 shows that this ensures correctness of the RTT estimate. This approach is also used in ePPing.

**TCP incoming data packets (Table I line 2, column 3):** Data packets from the remote host can be arbitrarily delayed by the remote application; thus, we cannot apply SEQ-ACK pairing on such packets. Instead, when the Timestamps option is in use, it is possible to determine the One-Way Delay (OWD) from the remote host to NETHINT as *local time - timestamp in packet* (again only sampling packets with the first new timestamp value in case of consecutive similar ones). The resulting value is incorrect in absolute terms unless clocks are synchronized, but this does not matter since we primarily care about the change of this value as an indicator of (jointly) growing or diminishing congestion. OWD values are used in the same way in the widely deployed Low Extra Delay Background Transport (LEDBAT) mechanism [25] (e.g., LEDBAT is used in BitTorrent, and it has been reported to be in use for Microsoft Windows and Apple operating system updates [26]). This OWD estimate is also prone to error from clock skew in long connections, but this is typically very small [27] and hence probably negligible for interference detection.

**QUIC packets (Table I line 3):** After the handshake, QUIC prevents monitoring by encrypting all but a small readable part of the packet header. The spin bit, which is available in this readable part, has been specifically designed to make RTT measurements possible [23]; it is contained in packets following the handshake, since the handshake itself can also be monitored (prior to encryption establishment). NETHINT implements spin bit monitoring as specified in [28]; the present version of NETHINT does not parse the the QUIC handshake.

## IV. Evaluation

### A. Test setup: topology and scenarios

We evaluated the accuracy of interference measurements with our NETHINT tool in our TEACUP [29] testbed. Figure 4 illustrates the network topology and how the traffic flows between the devices. The servers were utilised to generate traffic, while the clients acted as traffic sinks. This setup models a scenario where Client 2 consumes VoIP traffic from Server 2, while Client 1 consumes bandwidth for different activities from Server 1. The objective is to determine whether NETHINT, running at Router 1, is able to discern if the traffic between Server 1 and Client 1 interferes with the traffic between Server 2 and Client 2.

The logical topology presented in Figure 4 demonstrates how the traffic between Server 1 and Client 1 passes through both Routers 2 and 1, whereas the traffic between Server 2 and

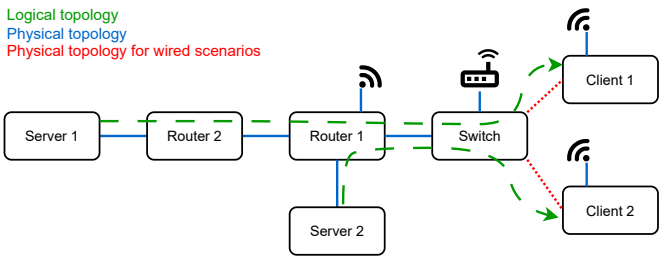


Fig. 4: TEACUP testbed experimental setup.

Client 2 just traverses Router 1. Such traffic patterns enable us to either create a shared bottleneck at Router 1 or the WiFi AP connected to it on one hand, or a non-shared bottleneck on Router 2 on the other. Our WiFi AP was a NETGEAR N300 Wireless Router with External Antennas (JWNR2010) that we set to follow the IEEE 802.11 b/g/n 2.4GHz standard, to obtain a 54 Mbit/s limit. We emulated scenarios with and without a common bottleneck using the following three configurations (the wired connections were 100 Mbit/s Ethernet):

1) **No common bottleneck:** The outgoing (right, in Figure 4) interface of Router 2 limited traffic from Server 1 to 15 Mbit/s, The WiFi AP limited traffic from Server 2 to 54 Mbit/s.
2) **Common bottleneck, wireless:** The outgoing interface of Router 2 was set to 70 Mbit/s, The WiFi AP limited all traffic to 54 Mbit/s.
3) **Common wired bottleneck:** The outgoing interface of Router 2 was set to 70 Mbit/s, the outgoing (right) interface of router 1 limited all traffic to 15 Mbit/s.

The third scenario was included to see how strongly the results depend on the wireless behavior and on the specific configured capacity. Since both of these aspects were different (wired instead of wireless, and a different capacity), seeing similar results would indicate that these factors do not play a major role—and this is desirable.

To somewhat realistically emulate a home network where interference with neighbors is common, we ran these tests at daytime on workdays in the office environment of the Informatics department at the University of Oslo, where multiple "foreign" WiFi SSIDs were visible, with "regular" Wifi usage.

### B. Test setup: network traffic

We used TCP traffic to allow NETHINT to collect meaningful information (we dismissed the idea of evaluating QUIC after finding that, in tests with Chromium v109.0.5417.74, QUIC's "spin bit" was not enabled for Google.com or any other websites that we tried). Therefore, in our experiments, we employed a single VoIP flow[4] between Client 2 and Server 2 while varying traffic types between Server 1 and Client 1 as outlined below, using the TCP variants available in our system (all hosts ran Linux kernel version 6.1.0):

1) 3 TCP NewReno flows
2) 3 TCP BBR flows
3) 3 TCP Cubic flows
4) 1 TCP BBR, 1 TCP NewReno, 1 TCP Cubic flow
5) 1 VoIP, 3 TCP NewReno flows
6) 1 VoIP, 3 TCP Cubic flows
7) 1 VoIP, 3 TCP Cubic flows
8) 1 VoIP, 1 TCP BBR, 1 TCP NewReno, 1 TCP Cubic flow

These are 8 different types of traffic that were run on two different delay configurations (10 and 50 ms) as well as 4 different queue length configurations (0.5, 1, 1.5, and 2 times the Bandwidth×Delay Product (BDP)). This yields 64 different tests that were run for each of the three bottleneck configurations. After removing 13 failed tests, we had data from a total of 179 valid test runs. To ensure at least 20 TCP sawteeth in the slowest configuration (maximum BDP, largest queue and NewReno), each test ran for 300 seconds.

### C. Correlating NETHINT outputs

For all tests, we determined the correlation by taking OWD values (i.e., delay from the Server 1 or Server 2 to Router 1) from NETHINT's output and calculating the Pearson correlation coefficient (PCC) [20]. We did not differentiate the TCP streams between Server 1 and Client 1, but instead looked at all data going towards Client 1 as a whole.

Our first approach was to separately calculate the PCC for each test's raw data, and then obtain the Cumulative Distribution Function (CDF) of all tests for each of the three scenarios (no common bottleneck, common wireless bottleneck, common wired bottleneck). Since the PCC requires the inputs to have the exact same length, we had to enforce this property by interpolating missing values. We did this using the `numpy` library's `numpy.interp` function, which implements a piecewise linear interpolation. We note that this introduces artificial data points, which could, in principle, provoke false results. This does however not diminish the credibility of our evaluation, since we compare the correlation output against ground truth.

The result is shown in Figure 5. While we can see that the wired and wireless behavior with a shared bottleneck is indeed quite similar, and altogether different from the case of no common bottleneck, the plot is also a little unsatisfactory, as it would be hard to draw a clear boundary (vertical line in the plot) that can separate the case of no common bottleneck from the other cases.

In the absence of a common bottleneck, any correlations are misleading, as they are just caused by traffic pattern similarities. The result could be improved with one of the more sophisticated SBD methods described in Section II; instead, we use the PCC for simplicity, and to keep the discussion more focused on the raw data that NETHINT obtains.

In Figure 5, all the tests where the correlation coefficient in the "no common bottleneck" scenario was above 0.7 contained VoIP traffic between Server 1 and Client 1 (traffic scenarios

[4]We sent 20 TCP packets per second, each with a packet size of 100 bytes to emulate VoIP traffic (this mimics Skype, which will use TCP when UDP does not work and was found to send at roughly this rate and packet size with occasional outliers [30]).

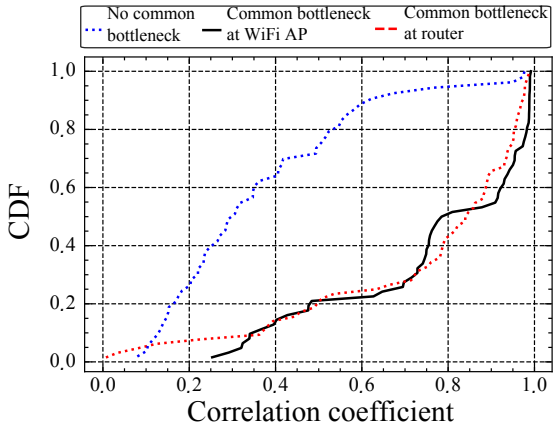


Fig. 5: Correlations for all tests, obtained by calculating: 1) the correlation per test, 2) the CDF of all tests, for each of the three shown cases.

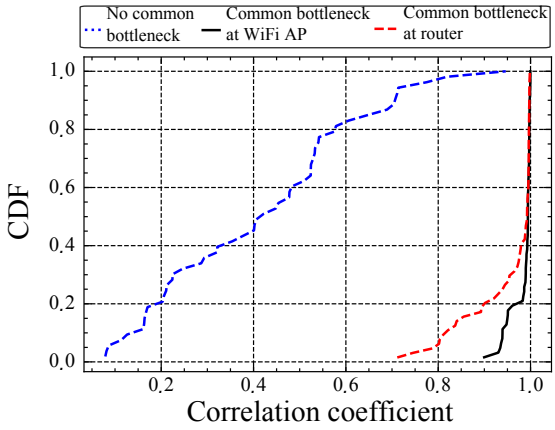


Fig. 7: Correlations for all tests, obtained by calculating: 1) the CDF per test, 2) the correlation per test, 3) a CDF of all tests, for each of the three shown cases.

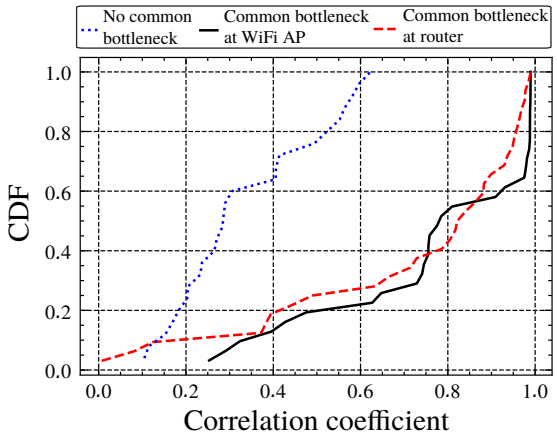


Fig. 6: Correlations for all tests without VoIP traffic, obtained by calculating: 1) the correlation per test, 2) the CDF of all tests, for each of the three shown cases.

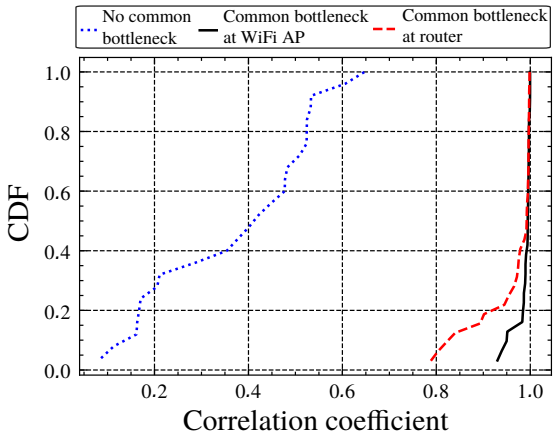


Fig. 8: Correlations for all tests without VoIP traffic, obtained by calculating: 1) the CDF per test, 2) the correlation per test, 3) a CDF of all tests, for each of the three shown cases.

5-8). This traffic has a very distinct pattern, and it was also visible in the traces whenever all other TCP flows backed off, emptying the queue. The cases including VoIP from Server 1 to Client 1 are therefore particularly unfavorable scenarios for our system, where both users use an application that has the same, quite distinct, traffic pattern. Removing the traffic cases with VoIP (5-8) yielded Figure 6, which shows the "no common bottleneck" cases ending at a PCC value that is slightly above 0.6. This means that a management system could give out a **"local traffic interference!"** warning when the PCC exceeds a value such as 0.7: clearly, all such cases *do* share a bottleneck. On the negative side, as we can see from the figure, approximately 30% of interference (common bottleneck) cases would not be detected by such a system.

Another way to compare the traffic is to first calculate the CDF from the raw OWD values in a test. This allows for comparing the distribution of OWD values instead of how the OWD values change over time. To align these OWD value distributions, we shift them such that the lowest OWD value for each client becomes 0. This means that all other OWD values represent how much longer the OWD was compared to the lowest OWD, and that the calculated correlation represents how similar this distribution is between the two clients.

As can be seen in Figure 7, this did not significantly affect the CDF for the "no common bottleneck" case, except for decreasing the highest values slightly compared to the plot in Figure 5. However, the values for both configurations with a common bottleneck now appear significantly more correlated. With this calculation, the highest correlation coefficient for no common bottleneck is below 0.95, while the same value only accounts for only about 20% of the tests with a common bottleneck. In other words, if a management system would give out a **"local traffic interference!"** warning at this value, it would be correct, and it would miss only 20% of the cases where such interference really happened. Alternatively, a PCC value of 0.8 would mis-classify both "common bottleneck" and 'no common bottleneck" cases, but only very few. Altogether, this result is significantly better than what we had before

(Figure 5 and Figure 6). When doing the calculation in this way, the difference between including the tests with VoIP traffic and not including them was less significant—but now, in Figure 8, we can see that it is possible to make a 100% correct distinction between the cases of sharing or not sharing a bottleneck with a correlation coefficient of 0.7.

## V. Conclusion

We have shown that NETHINT allows to clarify if network problems are caused by interference between applications on a home network. We believe that this is an important first step towards a management approach for home networks that ensures good network quality for its users. Possible applications of NETHINT range from raising alarms when the specified network needs of applications are not met to a distributed solution for prioritization, and better usage of multiple APs in a mesh network.

A number of important features are missing for practical real-life use of NETHINT, and they offer interesting opportunities for future work: NETHINT does not currently decrypt traffic. Our testbed used an unencrypted WiFi network, which sufficed to show the tool's ability to detect traffic correlations, but this is not how a practical network should be operated. It is slow, causing it to miss many packets—similar to ePPing, NETHINT version 2 could leverage eBPF for higher speed. Some of the more sophisticated correlation methods for SBD from Section II could be used instead of the Pearson correlation coefficient.

The relationship between inputs and outputs produced by an SBD approach could be used as training data for a Machine Learning (ML) system, which could then be used with online inference for a network monitoring and management system aimed at intelligent decision-making. More direct information on wireless interference that a NIC in monitor mode provides could also be fed into such an ML system (per-frame signal strength etc.). We caution, however, that such outputs can never be 100% reliable, as they depend on the traffic pattern: as we have seen with our emulated VoIP traffic, very regular or otherwise peculiar traffic patterns may always produce correlations in the raw delay data even when there are no common bottlenecks.

All in all, we see several exciting possibilities for future systems that can address the goal of informing users about the reasons for any network problems they may face—and we expect NETHINT to be a good basis for such developments.

## References


[1] S. Floyd and M. Allman, "Comments on the Usefulness of Simple Best-Effort Traffic." RFC 5290, July 2008.

[2] E. Lear, "Report from the IAB Workshop on Internet Technology Adoption and Transition (ITAT)." RFC 7305, July 2014.

[3] D. Damjanovic, P. Gschwandtner, and M. Welzl, "Why is this web page coming up so slow? investigating the loss of syn packets," in *NETWORKING 2009* (L. Fratta, H. Schulzrinne, Y. Takahashi, and O. Spaniol, eds.), Springer, 2009.

[4] J. Gettys, "Bufferbloat: Dark buffers in the internet," *IEEE Internet Computing*, vol. 15, no. 3, pp. 96–96, 2011.

[5] K. Park, "The Internet as a Complex System," in *The Internet as a Large-Scale Complex System*, Oxford University Press, 06 2005.

[6] B. Briscoe, A. Brunstrom, A. Petlund, D. Hayes, D. Ros, I. Tsang, S. Gjessing, G. Fairhurst, C. Griwodz, and M. Welzl, "Reducing internet latency: A survey of techniques and their merits," *IEEE Communications Surveys Tutorials*, vol. 18, pp. 2149–2196, thirdquarter 2016.

[7] A. Dhamdhere, D. Clark, A. Gamero-Garrido, M. Luckie, R. Mok, G. Akiwate, K. Gogia, V. Bajpai, A. Snoeren, and k. claffy, "Inferring Persistent Interdomain Congestion," in *ACM SIGCOMM*, Aug 2018.

[8] B. I. Teigen and M. Olden, "Quality of Outcome," Internet-Draft draft-ietf-ippm-qoo-00, IETF, Mar. 2024. Work in Progress.

[9] B. I. Teigen and M. Olden, "Requirements for a Network Quality Framework Useful for Applications, Users, and Operators," Internet-Draft draft-teigen-ippm-app-quality-metric-reqs-02, Internet Engineering Task Force, Oct. 2023. Work in Progress.

[10] C. Paasch, R. Meyer, S. Cheshire, and W. Hawkins, "Responsiveness under Working Conditions," Internet-Draft draft-ietf-ippm-responsiveness-04, Internet Engineering Task Force, Mar. 2024. Work in Progress.

[11] D. A. Hayes, M. Welzl, S. Ferlin, D. Ros, and S. Islam, "Online identification of groups of flows sharing a network bottleneck," *IEEE/ACM Transactions on Networking*, vol. 28, no. 5, pp. 2229–2242, 2020.

[12] D. Hayes, S. Ferlin, M. Welzl, and K. Hiorth, "Shared Bottleneck Detection for Coupled Congestion Control for RTP Media." RFC 8382, June 2018.

[13] O. Younis and S. Fahmy, "Flowmate: scalable on-line flow clustering," *IEEE/ACM Transactions on Networking*, vol. 13, no. 2, 2005.

[14] S. Sundberg, A. Brunstrom, S. Ferlin-Reiter, T. Høiland-Jørgensen, and J. D. Brouer, "Efficient continuous latency monitoring with eBPF," in *Passive and Active Network Measurement Conference*, Springer, 2023.

[15] E. Dong, P. Gao, Y. Yang, M. Xu, X. Fu, and J. Yang, "SmartSBD: smart shared bottleneck detection for efficient multipath congestion control over heterogeneous networks," *Computer Networks*, vol. 237, 2023.

[16] J. Song, B. Chen, B. Song, A. Ying, and Y. Hu, "Flowbot: A learning-based co-bottleneck flow detector for video servers," in *ICNP*, (Los Alamitos, CA, USA), pp. 1–12, IEEE Computer Society, oct 2023.

[17] S. Sundaresan, N. Feamster, and R. C. Teixeira, "Home network or access link? locating last-mile downstream throughput bottlenecks," in *Passive and Active Network Measurement Conference*, 2016.

[18] W. Wei, Y. Wang, K. Xue, D. S. L. Wei, J. Han, and P. Hong, "Shared bottleneck detection based on congestion interval variance measurement," *IEEE Communications Letters*, vol. 22, no. 12, 2018.

[19] W. Wei, K. Xue, J. Han, D. S. L. Wei, and P. Hong, "Shared bottleneck-based congestion control and packet scheduling for Multipath TCP," *IEEE/ACM Transactions on Networking*, vol. 28, no. 2, 2020.

[20] D. Freedman, R. Pisani, and R. Purves, "Statistics (international student edition)," *Pisani, R. Purves, 4th edn. WW Norton & Company, New York*, 2007.

[21] R. Barik, M. Welzl, A. Elmokashfi, T. Dreibholz, S. Islam, and S. Gjessing, "On the utility of unregulated IP DiffServ Code Point (DSCP) usage by end systems," *Performance Evaluation*, vol. 135, p. 102036, 2019.

[22] A. K. Paul, A. Tachibana, and T. Hasegawa, "An enhanced available bandwidth estimation technique for an end-to-end network path," *IEEE Trans. Network and Service Management*, vol. 13, no. 4, 2016.

[23] P. De Vaere, T. Bühler, M. Kühlewind, and B. Trammell, "Three bits suffice: Explicit support for passive measurement of internet latency in QUIC and TCP," in *Proc. IMC*, IMC '18, p. 22–28, ACM, 2018.

[24] D. Borman, R. T. Braden, V. Jacobson, and R. Scheffenegger, "TCP Extensions for High Performance." RFC 7323, Sept. 2014.

[25] M. Bagnulo and A. García-Martínez, "An experimental evaluation of LEDBAT++," *Computer Networks*, vol. 212, p. 109036, 2022.

[26] D. Ros and M. Welzl, "Less-than-best-effort service: A survey of end-to-end approaches," *IEEE Communications Surveys & Tutorials*, vol. 15, no. 2, pp. 898–908, 2013.

[27] S. Zander and G. Armitage, "Minimally-intrusive frequent round trip time measurements using synthetic packet-pairs - extended report," 2013.

[28] J. Iyengar and M. Thomson, "QUIC: A UDP-Based Multiplexed and Secure Transport." RFC 9000, May 2021.

[29] F. Jowkarishasaltaneh and J. But, "msTEACUP-running repeatable experiments with MPTCP over complex network topologies," in *IEEE Conference on Local Computer Networks (LCN)*, pp. 1–4, IEEE, 2023.

[30] M. Mazhar Rathore, A. Ahmad, A. Paul, and S. Rho, "Exploiting encrypted and tunneled multimedia calls in high-speed big data environment," vol. 77, no. 4, pp. 4959–4984.