SnailLoad: Exploiting Remote Network Latency Measurements without JavaScript
Stefan Gast, Roland Czerny, Jonas Juffinger, Fabian Rauscher, Simone Franza, and Daniel Gruss, Graz University of Technology
https://www.usenix.org/conference/usenixsecurity24/presentation/gast
SnailLoad: Exploiting Remote Network Latency Measurements without JavaScript

Stefan Gast, Roland Czerny, Jonas Juffinger, Fabian Rauscher, Simone Franza, Daniel Gruss
Graz University of Technology

Abstract
Inferring user activities on a computer from network traffic is a well-studied attack vector. Previous work has shown that they can infer websites visited, videos watched, and even user actions within specific applications. However, all of these attacks require a scenario where the attacker can observe the (possibly encrypted) network traffic, e.g., through a person-in-the-middle (PITM) attack or sitting in physical proximity to monitor WiFi packets.

In this paper, we present SnailLoad, a new side-channel attack where the victim loads an asset, e.g., a file or an image, from an attacker-controlled server, exploiting the victim’s network latency as a side channel tied to activities on the victim system, e.g., watching videos or websites. SnailLoad requires no JavaScript, no form of code execution on the victim system, and no user interaction but only a constant exchange of network packets, e.g., a network connection in the background. SnailLoad measures the latency to the victim system and infers the network activity on the victim system from the latency variations. We demonstrate SnailLoad in a non-PITM video-fingerprinting attack, where we use a single SnailLoad trace to infer what video a victim user is watching momentarily. For our evaluation, we focused on a set of 10 YouTube videos the victim watches, and show that SnailLoad reaches classification $F_1$ scores of up to 98%. We also evaluated SnailLoad in an open-world top 100 website fingerprinting attack, resulting in an $F_1$ score of 62.8%. This shows that numerous prior works, based on network traffic observations in PITM attack scenarios, could potentially be lifted to non-PITM remote attack scenarios.

1 Introduction
Side channels can leak information from implementations that are functionally correct and free of implementation errors [40]. Software-based side channels have gained an increasing amount of attention in the system security community, in particular cache attacks, which exploit timing differences induced by the caching infrastructure of a system [52, 79]. Software-based timing attacks, in particular on the cache, have been demonstrated on cryptographic algorithms [8], on ASLR (address-space layout randomization) [23, 31], secure enclaves [61], from websites [51], and to build covert channels [45, 59]. Most of these works study local scenarios where an attacker can execute native code [31], sandboxed code [77], or code run in an interpreter [62].

Other works study remote scenarios, typically with implementation-specific operations, such as encryption interfaces [10] or special hardware interfaces [42]. A more extensive set of works focuses on fingerprinting [65] using remote side channels, e.g., via passive traffic analysis from within the same network [7, 47, 70, 71]. Alexander and Crandall [4] focused on a more active attack scenario with spoofed SYN packets, triggering interaction between victim and server that the attacker can passively observe. Most works in this space, however, focus on person-in-the-middle scenarios to fingerprint, e.g., applications [69, 74] or websites [9, 58]. That is, they assume that the attacker controls at least one gateway or router in the path between server and victim. Naturally, this allows to monitor precise transmission times of packets, as well as their types and sizes [2, 16, 17, 26, 29, 38, 39, 41, 54, 66, 67, 72]. Passive remote attacks have so far only been demonstrated in the context of the Tor anonymity network [46, 48]. Murdoch and Danezis [48] showed that they can reconstruct the activity of a Tor relay node using passive latency measurements.

Given this prior work we ask the following question: Can a passive remote attacker use latency measurements as a side channel for browsing activity on a victim system? Can such a side-channel attack be mounted over the Internet?

In this paper, we answer both questions in the affirmative: We show that neither specific code on the victim machine nor direct observation of the possibly encrypted network traffic is required to infer browsing activity on the victim system. We show that these attacks are possible from arbitrary Internet servers, with distances of more than 8 hops to the victim, and with only minimal network activity. SnailLoad exploits the subtle variations in the round-trip times (RTTs) of net-
work packets that carry a side-channel signal influenced by activity of the victim. This enables an attacker to perform network side-channel attacks, that so far required a person-in-the-middle (PITM) scenario, from a fully passive and fully remote scenario off the transport path between victim and server.

We investigate the latency variations by analyzing the RTTs for different internet connections. Our analysis shows that the root cause of SnailLoad are bandwidth differences in the transport path between victim and server. As the server cannot know how fast the victim can receive the packets, it sends a certain amount (i.e., a burst) of data in a short time frame. If there is a node on the transport path that hands over data from a higher bandwidth channel to a lower bandwidth channel, it has to buffer packets to transmit them sequentially over the lower bandwidth channel. Extreme cases where buffering leads to unusually high latencies are known as the bufferbloat phenomenon. While bufferbloat has only been considered a quality-of-service issue so far, SnailLoad shows that the timing differences induced by this buffering are exploitable by any attacker that can send packets through the same network node to the victim. Importantly, this node is typically the last node connecting the victim exclusively to the internet, i.e., all packets to and from a victim pass through this node.

We evaluate SnailLoad in the first fully remote, non-person-in-the-middle web activity fingerprinting attack. To remain stealthy and to evade widely-used firewall restrictions (e.g., ICMP ping messages being dropped), we masquerade our SnailLoad latency measurements as a slow HTTP transfer from an attacker-controlled web server. This HTTP transfer can be a background connection, e.g., for a messenger app, or a slow website component embedded (e.g., an advertisement, image, font, or style sheet) in an otherwise benign and responsive website. Importantly, SnailLoad runs no code, e.g., no JavaScript, on the victim system. We show that a single SnailLoad trace from this scenario suffices to infer what video a victim user is watching momentarily. For our evaluation, we focused on a set of 10 YouTube videos the victim watches, and reach classification $F_1$ scores of up to 98%. We also evaluated SnailLoad in an open-world website fingerprinting attack, resulting in a macro-averaged $F_1$ score of 62.8%. When using SnailLoad across connections, with training data from one connection applied to another, we still achieve an $F_1$ score of 40% in a top-10 closed-world fingerprinting scenario.

We evaluate the general applicability of SnailLoad on 10 home internet connections using ADSL, FTTH, FTTB, LTE, and cable access technologies. With training data from the victim’s internet connection over the top 10 YouTube videos, played for 90s with Firefox, we achieve $F_1$ scores between 37% on an FTTB connection and 98% on an FTTH connection. Our evaluation shows variations in the accuracy between internet access technologies, depending on whether the last-mile transport medium is shared among multiple subscribers or not. We evaluate the attack performance for probing frequencies between 20 Hz to 5 000 Hz, resulting in 0.4 kB/s to 320 kB/s of network traffic, in relation to differently-sized network activity of the victim ranging from 512 B to 8 MB. Our results show that with a higher probing frequency, we can observe also smaller-sized network activity of the victim. However, below a size of 512 kB, we find no probing frequency that leaks the network activity reliably. This is most likely due to not causing sufficient traffic for a visible amount of contention on the packet queue before the last mile, resulting in no visible delay for the attacker’s packets.

We conclude that numerous prior works on network side channels in PITM attack scenarios could potentially be lifted to a remote non-PITM attack scenario based on SnailLoad.

**Contributions.** We summarize our contributions as follows:

1. We present SnailLoad, a new side-channel attack exploiting latency variations of the victim’s network connection measured by loading an asset from an attacker-controlled server; allowing to infer web activity of a victim.
2. We show that the root cause of the side channel is buffering in a transport path node, typically the last node before the user’s modem or router, related to a quality-of-service issue called bufferbloat.
3. We show that SnailLoad works despite the blocking of ICMP echo packets, through a low-bandwidth background connection such as a slow download, slow loading image in a website, or a connection for a chat client.
4. We evaluate SnailLoad in a user study with 10 different internet connections and 6 different connection technologies, achieving video-fingerprinting $F_1$ scores of 37% to 98%. We also evaluate SnailLoad in a top-100 open-world website-fingerprinting attack, achieving an $F_1$ score of 62.8%.

**Outline.** Section 2 provides background on internet technologies and remote fingerprinting side-channel attacks. Section 3 presents the basic latency measurements that are the foundation of SnailLoad. Section 4 examines the root cause
of SnailLoad. Section 5 presents the threat model and attack setup of SnailLoad. Section 6 evaluates SnailLoad in a video-fingerprinting scenario. We performed a user study comprising 10 different internet connections and 6 different internet connection technologies. Section 7 presents a top-100 open-world website-fingerprinting attack. We discuss the impact of SnailLoad and the relevant scenarios in Section 8. We conclude in Section 9.

2 Background

We provide background on internet access technologies typically used by end users to connect to the internet. Furthermore, we briefly introduce TCP/IP networking and dynamic adaptive streaming over HTTP. Finally, we describe video fingerprinting attacks and remote timing side channels.

2.1 Internet Access Technologies

While fast backbone internet routers deliver 100 Gbit/s per trunk line [24], with a recently ongoing shift to 400 Gbit/s technology [15], the last mile of an internet connection still forms a bandwidth bottleneck. For Q3 2023, a recent study [37] of an internet speed test provider shows a worldwide median download speed of 83.95 Mbit/s for wired connections and a worldwide median download speed of 203.04 Mbit/s for 5G connections, with high variations between different countries.

The prevalent connection types for end-users are DSL, cable, fiber optic, and mobile. In this section, we briefly describe their characteristics.

Digital Subscriber Line (DSL) DSL is an internet connection technology utilizing existing copper telephone lines. Multiple subscribers are connected to a DSL access multiplexer (DSLAM) before forwarding the connection to the internet service provider (ISP). The location of the DSLAM is an essential factor in the achievable bandwidth due to signal attenuation increasing with the length of the copper lines [21].

DSL has a low bandwidth and high latency compared to other technologies like cable and fiber optics. Download bandwidths typically range from 5 Mbit/s to 120 Mbit/s, with upload bandwidths from 1 Mbit/s to 20 Mbit/s. Since the last mile of DSL is shared between multiple participants, the bandwidth can be limited when many participants use the shared line. This also influences the latency.

Cable Similarly to DSL, cable internet also takes advantage of existing infrastructure: the coaxial cables for cable television. The Data Over Cable Service Interface Specification (DOCSIS) is a set of specifications standardizing data transmission over television cable systems [32]. In contrast to DSL, the length of the data cable, due to being a coaxial cable, is not that influential on the bandwidth.

However, the number of end users sharing the same connection strongly influences the bandwidth. The download bandwidth of cable internet is up to 10 Gbit/s with DOCSIS 4.0, and the upload bandwidth is up to 6 Gbit/s [32].

Fiber optic Fiber optic internet connections use light impulses transmitted via a fiber optic cable. They provide the highest bandwidth and lowest latency for data transmission, reaching up to 50 Gbit/s symmetric [19]. The fiber optic cable itself does not constrain this bandwidth but instead by the capabilities of the terminal equipment [56].

Fiber optic internet connections vary based on the composition of the last mile’s connectivity. In a fiber-to-the-home (FTTH) setup, the entire last mile exclusively uses fiber optic cables, resulting in a non-shared, highly efficient connection.

In different, less costly configurations, such as Fiber-to-the-Curb (FTTC) or Fiber-to-the-Building (FTTB), existing infrastructure like coaxial cables links homes to the optical fiber network. In many instances, this leads to sharing lines with other users, similar to cable and DSL connections.

Mobile In contrast to the previously described wired technologies, the router is wirelessly connected to a cellular tower for a mobile internet connection. Because they generally serve multiple clients, the last mile of the connection is inherently shared, making the bandwidth and quality of the connection highly dependent on the number of clients.

The 5G standard uses higher frequency bands to increase bandwidth and reduce latency compared to 4G. Due to the higher frequency, 5G towers can only serve a smaller area than 4G towers. This leads to fewer users sharing the last mile, improving signal quality due to reduced congestion.

The bandwidth of 4G is up to 150 Mbit/s for download and up to 50 Mbit/s for upload. 4G+ increased it to 300 Mbit/s and 150 Mbit/s respectively [55]. 5G can reach significantly higher bandwidths, with up to 20 Gbit/s for download and 10 Gbit/s for upload [33].

2.2 TCP/IP Networking

The Transmission Control Protocol (TCP) is a protocol ensuring the correct and ordered transmission of data over a network. It works closely with the Internet Protocol (IP) that enables the transmission of packets across network boundaries from a specific source to a receiver. TCP/IP, the combined suite, forms the foundation of most modern internet communications, supporting a wide array of applications like web browsing (HTTP), E-Mail, and file transfers.

Packet Acknowledgment TCP is a connection-oriented protocol, using a handshake mechanism to establish a connection between sender and receiver. Upon the arrival of a packet,
the receiver sends an acknowledgment packet (ACK) back to the sender, enabling the retransmission of lost packets.

TCP is implemented using send and receive buffers. Packets reside in the send buffer until an ACK is received. The size of the send buffer determines how many not-yet-acknowledged packets can be in transit. A receive buffer holds all received packets the application still needs to read.

### Congestion Control

Another vital feature of TCP is congestion control, which is crucial for maintaining a stable network, especially when it becomes heavily loaded. TCP employs several mechanisms to avoid or remove network congestion.

A congestion window determines the amount of data the sender may send before receiving an ACK packet. The size of the congestion window is controlled by the sender and is based on its perception of network congestion using information like lost or delayed ACK packets. The value adapts dynamically to the current load of the network.

To determine the congestion window size, typically, an algorithm known as slow start is employed. The initially small congestion window size is doubled upon each successful round trip until it reaches a threshold where packet loss is detected. Once the threshold is reached, TCP transitions from slow start to congestion avoidance. In this mode, the window size is changed more slowly to avoid congestion.

### Dynamic Adaptive Streaming over HTTP

Dynamic Adaptive Streaming over HTTP (DASH) is a standard for video streaming [35], used by multiple video streaming services, including YouTube [26, 39]. DASH servers split videos into small chunks of typically 2–6 seconds, each with multiple quality levels, allowing the client to request each chunk on demand, dynamically choosing a suitable bitrate, based on the available bandwidth. Video codecs and streaming services typically use variable-bitrate (VBR) encoding [6], enabling them to save bandwidth for scenes with only minor picture changes and to increase the bitrate for scenes containing more motion. Consequently, timing and sizes of transferred chunks differ for each video, based on the content.

### Video Fingerprinting Attacks

As the timings and sizes of the transferred chunks form an individual fingerprint of each video, an attacker can infer the video played from the victim’s network traffic by matching the observed traffic metadata against prerecorded traces. Multiple prior video fingerprinting attacks monitored network traffic directly from a PITM position [16, 26, 27, 38, 39] or from a WiFi signal [16]. More recent work [57] demonstrated a video fingerprinting attack based on the detection of network interrupts on the victim machine, indirectly observing network traffic. Inferring videos from traces is a typical classification task. Prior attacks used different classification methods, such as Dynamic Time Warping [26], Minimum Variance Matching [26], Nearest Neighbor Classification [16], Support Vector Machines [16], Sequential Minimal Optimization [39] or, more recently, convolutional neural networks [38, 39, 57].

### Remote Timing Side Channels

In the context of side-channel attacks, the term remote has been used for a variety of scenarios. Some works described JavaScript-based attacks as remote [18, 25, 51, 76], other attacks focus on specific interfaces, e.g., encryption interfaces or encrypted web traffic [8, 10, 13, 14, 63], and a third category of attacks works by only sending network packets to a machine without targeting specific interfaces [42, 44, 73]. All of these attacks rely on implementation-specific operations that the victim system performs that are unrelated to the actual victim activity targeted. Contention of buffers within the local (wireless) network [75] same system [20] can also contribute to performance anomalies that SnailLoad exploits.

In the context of IoT devices, passive traffic analysis from within the same network has been demonstrated to allow derivation of privacy-related information [7, 47, 71]. This is related to a line of research fingerprinting encrypted web traffic [65]. Previous works have demonstrated application fingerprinting [69, 74], video fingerprinting [16, 68], and website fingerprinting [9, 58]. The two attack scenarios for these works are either a local network attacker, controlling a gateway in the local network far enough to spy on the traffic of a specific machine [7, 47, 70, 71], or a remote network attacker, controlling a router somewhere in the path between server and victim. More active attacks can also trigger interaction between victim and server that the attacker then measures [4]. Different features that are available to such person-in-the-middle attackers have been exploited [2, 16, 17, 26, 29, 38, 39, 72], e.g., message type [41, 67], packet length [54, 66]. Rather passive remote attacks have so far only been demonstrated in the context of the Tor network [46, 48]. Murdoch and Danezis [48] showed that they can reconstruct the activity of a Tor relay node using passive latency measurements. Finally, many network side-channel works focus on the information recovery rather than the information collection aspect [9, 16, 58, 68].

### The Latency Side Channel

In this section, we do not yet mount an SnailLoad but only study the latency side channel. We examine latency variations on internet connections depending on different activities and show that the ISP’s endpoint and the last mile have a significant influence. Our initial experiments also show that the effect can be observed by any unauthorized and unprivileged third party with a network connection to the system.

For now, we measure latencies using ICMP pings. For our SnailLoad attack later on, we have to resort to other packets as networks today commonly filter “ping packets” by default.
The ping command sends an ICMP Echo Request to the target machine, which in turn responds with an ICMP Echo Reply. The latency between sending the ICMP Echo Request and receiving the ICMP Echo Reply is recorded as the round trip time (RTT). Typical use cases of the ping command focus on the round trip time, e.g., to monitor the network status.

For our first experiment, we use a 50 Mbit/s ADSL connection, with a client machine directly connected to the internet gateway (i.e., home router) via an Ethernet cable. On the client machine, we ping the Google DNS server (IP: 8.8.8.8) with an interval of 50 ms and record the RTTs over time. Accessing google.com and amazon.com on the same machine shows RTT spikes while loading (cf. Figure 2a). We can see that this effect is quite subtle for a comparably plain website like the Google start page, while it is more noticeable for more complex websites like the Amazon start page. This is a surprising observation as either of the websites only triggers a few hundred kilobytes of traffic, nothing that should affect the network response times in a significant way.

**The latency is influenced by small degrees of victim internet activity already.**

In a second experiment, we show that the latency spikes are not caused by an effect on the client computer, e.g., kernel- or user-level packet handling, or system activity due to rendering. We repeat the experiment on the same internet connection but on a second machine directly connected to the gateway. We still ping the server from the first machine but load the websites on the second machine. This experiment results in the same latency spikes as shown in Figure 2b. This observation shows that the effect is not caused by any hardware or software component of either of the two client computers.

**The latency is influenced by internet activity on other systems in the same network.**

We also verify that the effect is not caused by the network connection between the two client machines. For this, the first client machine pings the second machine, with the second machine loading the websites. In this scenario, we do not observe any latency spikes corresponding to website loads. We repeat the same experiment with a ping interval of 10 ms, again without observing latency spikes. The average RTT in this setting is 0.91 ms ($n = 2223$), with a maximum of only 1.5 ms at the start of the measurement, unrelated to the website loads. These results indicate that the observed latency spikes are caused by either the ISP’s endpoint, the home internet gateway, or the last mile in between.

**The latency is influenced by the ISP’s endpoint and the last mile.**

### 4 Latency Side Channel Root Cause Analysis

In this section, we analyze the root cause for the latency spikes observed in Section 3. We show that these can be explained by packet buffering at the ISP’s endpoint, to which the customer’s internet gateway is connected on the last mile.

As the last mile of the user’s internet connection has a significantly lower bandwidth than backbone internet infrastructure (see Section 2.1), there is a bandwidth bottleneck at the ISP’s endpoint (i.e., DSLAM, CMTS, ONT, or cell tower). Network components, such as switches and routers, connect a high-bandwidth to a low-bandwidth network segment and buffer packets to be forwarded to the low-bandwidth segment. This prevents packet loss when the low bandwidth segment is congested. However, as packets have to be forwarded sequentially, a busy low-bandwidth connection delays the delivery of packets coming from the high-bandwidth segments.

**The root cause of the latency side channel is the buffering of packets in nodes between high-bandwidth and low-bandwidth network segments, often located directly before the last mile. Typically, all internet packets for a specific user have to pass through this node.**

We investigate the extent of this delay for 11 internet connections by comparing their baseline and congested ping round-trip times (RTTs). We directly connect a client computer to the home internet gateway (i.e., router) via a standard 1000BASE-T (ethernet) cable and ping the Google DNS server (IP address: 8.8.8.8). To measure the baseline RTT, we perform the measurement while the connection is other-
we tested. The histograms in Figure 3 show the distributions
wise idle. For the congested RTT, we perform the measure-
ment (AQM) schemes, such as CoDel [50,64] or PIE [53], we
want to emphasize that these only prevent excessive buffering
and reduce the delay caused by it. Even with AQM schemes,
the low-bandwidth bottleneck still exists, and some (reduced)
form of buffering is required to prevent excessive packet loss.

Thus, AQM reduces the difference between a congested
last mile and an idle connection. However, it cannot entirely elimi-
nate the difference. Hence, all the tested connections carry a
side-channel signal leaking usage information. Only for some of
these connections, the side-channel signal is amplified by a
heavy bufferbloat. Figure 4 illustrates the latency differences
for idle, busy, and bufferbloat-affected connections.

Table 1: Tested internet connections and their round trip times (idle and saturated), as measured with ICMP Echo Requests.

<table>
<thead>
<tr>
<th>Connection</th>
<th>Bandwidth [Mbit/s]</th>
<th>Idle RTT [ms]</th>
<th>Congested RTT [ms]</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>max.1</td>
<td>min.</td>
<td>avg.</td>
</tr>
<tr>
<td>ADSL-1²</td>
<td>50</td>
<td>47</td>
<td>13.939</td>
</tr>
<tr>
<td>ADSL-2²</td>
<td>8</td>
<td>3.8</td>
<td>14.754</td>
</tr>
<tr>
<td>FTTH-1²</td>
<td>80</td>
<td>73.4</td>
<td>7.200</td>
</tr>
<tr>
<td>FTTH-2³</td>
<td>80</td>
<td>76.2</td>
<td>8.10.538</td>
</tr>
<tr>
<td>FTTH-3³</td>
<td>300</td>
<td>292.8</td>
<td>7.093</td>
</tr>
<tr>
<td>FTTB-1⁴</td>
<td>250</td>
<td>249</td>
<td>34.467</td>
</tr>
<tr>
<td>FTTB-2²</td>
<td>300</td>
<td>291.2</td>
<td>12.067</td>
</tr>
<tr>
<td>FTTB-3²</td>
<td>150</td>
<td>236.8</td>
<td>14.437</td>
</tr>
<tr>
<td>LTE²</td>
<td>75</td>
<td>78</td>
<td>49.683</td>
</tr>
<tr>
<td>LTE⁺²</td>
<td>200</td>
<td>120</td>
<td>30.515</td>
</tr>
<tr>
<td>Cable²</td>
<td>250</td>
<td>261.3</td>
<td>20.622</td>
</tr>
</tbody>
</table>

¹ according to ISP, ² n = 5000 for RTT measurements, ³ n = 1000 for RTT measurements, ⁴ n = 100 for RTT measurements

Figure 3: Histograms of the round trip times of multiple internet connections, idle and with a download running in parallel.
Connection idle: packet forwarded immediately.
(b) Connection busy: packet buffered for a short delay.
(c) Bufferbloat: Excessive packets lead to long delays.

Figure 4: The effect of congestion on network latencies. The forwarding delay of the measurement packet correlates with the occupancy of the buffer in the router.

In extreme cases, the latency is amplified by the bufferbloat phenomenon.

While our analysis provides a good understanding of the latency side channel and the root cause of the latency variations and spikes, we now move on to the SnailLoad attack.

5 SnailLoad: Video-Fingerprinting Attack

In this section, we describe the SnailLoad attack in detail, starting with the threat model and attack overview. Because we cannot use ping to measure RTTs, we show a new accurate method to measure RTTs using TCP ACK packets. This method uses only 400 B/s of network traffic. Using it, we record network latency traces of streamed videos and classify them using a convolutional neural network. The average classification accuracy of 10 videos over all 10 tested internet connections is 66.8%.

5.1 Threat Model

The attacker wants to know which video the victim user is watching. The attacker runs a native code TCP server and causes the victim to initiate a TCP connection with a download from the server while watching the video. As any background TCP transfer is enough to conduct SnailLoad, there are numerous ways to deploy it, e.g., download mirrors for legitimate benign software, (third-party) website assets like images, and several other scenarios (see Section 8). Importantly, the attacker cannot run any code on the victim machine, i.e., the client application accessing the web server does not execute JavaScript or WebAssembly originating from the attacker’s web server. We furthermore do not assume ICMP Echo Requests and ICMP Echo Responses to be forwarded between the victim and the attacker, i.e., ping is blocked, a standard router configuration today [60].

Algorithm 1: Measuring round trip times using TCP Acknowledgment messages from a Linux process

Input: A socket descriptor sock with the TCP_NODELAY option set
Input: A single byte b to be sent to the client
Result: The measured round trip time rtt

1. begin
2. acked ← false;
3. start ← get_current_time();
4. send(sock, b, 1, 0);
5. repeat
6. if ioctl(sock, SIOCOUTQ) = 0 then
7. acked ← true;
8. end
9. until acked;
10. end ← get_current_time();
11. return end − start;
12. end

5.2 Attack Overview

Our video-stream fingerprinting attack consists of an online recording phase and an offline post-processing phase. During the recording phase, the attacking server records the network latency trace of the victim watching the first 90 seconds of a video, measuring the round trip times of single-byte TCP packets and their corresponding acknowledgment messages. In the post-processing phase, we infer which video was played from the recorded trace using a convolutional neural network trained with traces from an identical network setup.

5.3 Network Latency from TCP ACKs

In the previous experiments, we measured network round trip times using the ping command that was running on the victim machine. However, in our threat model, the attacker does not have control over the victim machine. We implement a similar measurement with code only running on the attacker-controlled server. While, in principle, it is possible to ping the public IP address of the victim machine, the attacker might not know the victim’s public IP address in advance, and the internet gateway of the victim might be configured to drop ICMP echo requests for security reasons [60]. In this section, therefore, we exploit the TCP acknowledgment mechanism to measure round trip times, i.e., any open TCP connection between victim and server is sufficient to record round trip times periodically.

5.3.1 TCP Packet Acknowledgments

To ensure reliable transmission, TCP requires the recipient to acknowledge each segment back to the server. For this, the receiver sends back a TCP message to the sender with the ACK
flag and the acknowledgment number set in the TCP header (see Section 2.2). Measuring the time between sending some data and receiving the corresponding TCP acknowledgment again results in the round trip time, similar to using ICMP Echo Requests and Replies as before. However, unlike ICMP Echo Requests and Echo Responses, TCP acknowledgments are required for every TCP connection, and, therefore, they cannot be blocked globally.

A single, low-traffic TCP connection is sufficient to mount SnailLoad. Blocking ICMP Echo messages does not prevent SnailLoad.

5.3.2  TCP Packet Acknowledgment Detection

Linux does not provide a standardized way to inform a user space application of packet acknowledgments. However, by probing the number of packets in the send buffer, we can see when a packet was acknowledged because it is removed from the send buffer. The equivalent information can also be obtained using libpcap, as we show in Section 7.

Algorithm 1 shows how such a measurement is implemented in a Linux server process. After obtaining the start time (Line 3), we invoke the send system call to enqueue a single byte to be transmitted to the victim machine (Line 4). As send returns immediately after placing the data to be sent in the send buffer, only timing the send system call is not sufficient. However, as described in Section 2.2, sent data is removed from the send buffer as soon as the server receives an acknowledgment for it. Consequently, in Line 5 until Line 9, we wait until the send buffer becomes empty again, using the SIOCOUTQ socket ioctl call to retrieve the number of bytes currently in the send buffer (Line 6). As soon as that number is 0 again, we leave the loop and obtain the end time in Line 10, yielding the round trip time as the difference between the end time and the start time (Line 11). Alternatively, an equivalent approach with libpcap can be used to match outgoing packets and incoming ACKs (cf. Section 7). For probing frequencies that may exceed the round-trip time of the connection, we resort to libpcap, and use the send buffer for lower frequency attacks (e.g., Section 6).

By setting the TCP_NODELAY socket option, we disable Nagle’s Algorithm [34,49] to ensure that each of the single bytes is sent out immediately in separate TCP packets without merging multiple bytes into a single TCP packet. Consequently, each single byte is acknowledged separately by the victim machine. Therefore, each single-byte transmission enables the server to measure the round trip time to the victim. Furthermore, sending each byte in a separate TCP packet prevents measurement influence from changing TCP transmission window sizes.

It is possible to detect TCP packet acknowledgments with user privileges by polling the TCP send buffer.

Figure 5: Averaged RTT traces from a client watching two different videos twice. There are features to distinguish the two videos and match the two traces of the identical videos.

5.3.3  Web Server Implementation

While SnailLoad works with any TCP connection, we implement a web server offering a file via HTTP to record network latency traces due to the large compatibility with existing client applications, i.e., web browsers. The victim connects to the web server and requests the file. After sending back the HTTP header, the server performs latency measurements while transferring the file content. Every 50 ms, the server sends a single byte to the client and measures the round trip time by polling the send buffer, as previously described. We choose 50 ms as the time slice, since we want to have a high temporal resolution, yet we do not want to exceed the regular round trip time of an idle connection. This results in a low transfer rate of only $\frac{1B}{50ms} = 20B/s$ visible to the user, which gives SnailLoad its name. Effectively, due to the overhead of TCP/IP, this results in 400 B/s of network traffic, which is still very unsuspicious given that internet connections today can commonly handle multiple megabytes per second. Figure 5 shows the traces of one client watching two different videos two times. We can recognize the clear similarity between the
same videos (traces 1 and 2) and see distinctive features for the other video (traces 3 and 4).

5.4 Attack Parameter Sweep

To determine the conditions under which the side channel works, we perform a parameter sweep, shown in Figure 6, with different download sizes and sample rates. The evaluation was done on the ADSL-1 (50 Mbit/s) connection. We evaluated sampling rates specified as window sizes between 100 µs, corresponding to 100 kHz, to 51 200 µs, corresponding to 19.5 Hz. Translating the connection capacity to these window sizes we obtain 336 kB per 51 200 µs and 655 B per 100 µs. Thus, in theory, transmitting as much data within this time frame should lead to contention in the buffer before the last mile and thus to a visible latency difference. However, as we see in Figure 6, even for higher sampling rates, we cannot reliably detect downloads of less than 512 kB. The reason for this may be that slow-start or congestion-avoidance mechanisms successfully prevent the negative impact of the download’s packets on other packets. Furthermore, at higher sampling rates the TCP stack also starts grouping packets, effectively keeping the sampling rate higher than attempted. This observation can also be the reason for lower accuracies for some websites in our website-fingerprinting attack in Section 7.1, i.e., due to their relatively low size.

5.5 Classifying Network Latency Traces with a Convolutional Neural Network

In order to efficiently classify the network latency traces, we employ a method typically used in signal classification [12, 30, 57, 78]. To classify a network latency trace, we first apply a Short-Time Fourier Transform (STFT) to it. An STFT performs multiple Fourier Transforms on short time windows of the network trace. The output of the STFT consists of two dimensions, with one dimension being the time and a second dimension being individual Fourier Transforms on the time slices, corresponding to the change in frequency over time. These two dimensions provide us with frequency information from the network trace while partially preserving the time domain. The 2D STFT allows us to perform convolutions on the data, which would not be directly possible with the initial one-dimensional network latency traces. We feed the output of the STFT into a convolutional neural network (CNN) for classification. We use KERAS (Tensorflow) on an Intel(R) Core(TM) i5-10210U CPU, with a runtime of 5 min to 10 min. The layout of the CNN is shown in Table 2. The CNN consists of three convolutional layers followed by three dense layers. The final layer outputs the likelihood that the input corresponds to a specific label for each possible label.

6 SnailLoad: Video-Fingerprinting Evaluation

In this section, we show that SnailLoad leaks privacy-relevant information about which video a victim is playing. We evaluate SnailLoad on 10 different home internet connections. For a set of 10 YouTube videos, played for 90 s in Full HD resolution, our evaluation shows an accuracy of up to 98%.

We select the videos in two steps: From the videos trending in the USA between October 2022 and September 2023, we first select the videos with at least 70 million views, yielding 23 videos. In the second step, we randomly select 10 videos out of this subset.

For each internet connection, we repeat the following three steps: First, we record 50 traces of the first 90 s of each of the 10 videos (Section 6.1). In the second step, we reserve a randomly selected subset of 10 traces per video as a test set and use the remaining 40 traces to train our CNN-based classifier (Section 6.2). Finally, we evaluate the trained model on the test set (Section 6.3).
6.1 Recording the Network Latency Traces

For each connection, we successively record traces for the 10 videos and repeat the recording in 2 batches with 25 iterations. Recording the videos successively during each iteration prevents correlations between daytime-dependent noise and the played video. For each trace, the client starts the SnailLoad HTTP file transfer and, after waiting for 3 s, also starts watching the video. After 90 s, the client aborts the file transfer and closes the video. After a pause of 3 s, the client continues with the next video. In total, recording all 500 traces for a single connection takes approximately 13.5 h.

Each recording is performed on one of two hosted servers, both of them connected to the Internet via a 1 Gbit/s downlink. Both servers are more than 8 hops away from the client machine, as tested with the `traceroute` command.

On the client machine, we use Firefox for both the HTTP file transfer and for watching the video. However, as the network-latency side channel is independent of the browser and operating system being used, we can expect similar results using another client configuration. We also tested this in smaller experiments but focused on this specific setup for the large-scale evaluation. As our traces shall only contain data from videos requested by the user, without any advertisements, we use uBlock Origin [28] as an ad blocker. Using an ad blocker is not strictly required for the attack because advertisements are effectively separate videos, and prior work [6] has shown that each video starts with a characteristic burst of network traffic, allowing the attacker to separate the advertisements from the video. However, it would increase the time until the video starts, prolonging the time the participants of our user study need to run the measurement. Hence, we opted for ad blocking to reduce the time for the participants.

6.2 Training the Classifier

After recording, we split the 50 traces of the 10 videos into training, validation and test sets. Following best practices [1], we use a training set of 36 traces per video to fit the model, a validation set of 4 traces per video to evaluate the model during training and a test set of 10 traces for the final evaluation of the trained model. For each internet connection, the individual hyperparameters for the CNN were empirically chosen to achieve good generalization against the validation set.

6.3 Results

After training the classifier for the internet connection, we evaluate the model on the test set. Figure 7 shows the results for all the 10 internet connections we tested. Each cell shows the probability that the classifier assigns the label indicated by the column to the videos indicated by the row. White or light gray cells indicate a low probability, whereas black or dark gray cells indicate a high probability. We also provide the $F_1$ scores we obtained for each connection.

For all the internet connections we tested, we obtain $F_1$ scores significantly better than random guessing (which would be $F_1 = 10\%$). The high probabilities along the diagonals in the confusion matrices also support that our attack works...
on all the tested connections. Overall, our $F_1$ scores range from 98% for the FTTH-1 connection to 37% for the FTTH-3 connection.

When comparing the FTTH connections, we notice a difference in accuracy between the 80 Mbit/s (FTTH-1, FTTH-2) and the 300 Mbit/s connection (FTTH-3). A lower bandwidth increases the transfer duration of the video segments, resulting in signal features that are easier to detect for our classifier. Thus, on the lower bandwidth FTTH-1 and FTTH-2 connections, we achieved accuracies of 98% and 94%, respectively, whereas on the higher bandwidth FTTH-3 connection, the accuracy was 57%.

When comparing the results from the fiber-based connections, we notice a difference in accuracy between the FTTH connections and the FTTB connections. The accuracy with the FTTB-1 and FTTB-3 connections was significantly lower, with $F_1$ scores of only 41% and 37%, respectively. This might be caused by noise from other customers using the same shared fiber cable of the FTTB connection simultaneously. Especially for the FTTB-3 connection, when measuring the RTTs for the congested case in Section 4, we observed significant fluctuations in the measured latencies, which have a standard deviation of $\sigma = 12.595$ ms ($n = 5000$). This is relatively high when compared to the average RTT difference between the idle and congested case of only 13.014 ms, and thus makes detecting relevant features more challenging for the classifier.

Compared to the FTTB connections, we achieved higher accuracies on the cable and mobile connections. With the Cable connection, the accuracy was $F_1 = 66\%$. For the mobile LTE and LTE+ connection, we achieved similar $F_1$ scores of 57% and 56%, respectively. These comparatively high accuracies are surprising, as for both cable and mobile connections, the last mile is shared among multiple customers, similar to FTTB. However, both cable [32] and LTE [55] divide the frequency band used for transmissions into separate channels, possibly reducing the sharing between customers.

On both ADSL connections, we also achieved high accuracies of $F_1 = 89\%$ (ADSL-1) and $F_1 = 73\%$ (ADSL-2). The relatively low bandwidth contributes to prominent signal features (as shown in Figure 5) that improve the accuracy.

Overall, as with other fingerprinting attacks, the accuracy can be improved with more training data, i.e., more or longer traces per video. However, we aimed to cover a wide variety of internet connections and recruited volunteers for recording the traces. While our recording time was limited for practical reasons, we assume that a dedicated attacker has the resources available to record more training data and thus to improve the accuracy of the attack even further.

We repeated the attack on the ADSL-1 connection, with the video quality reduced to 720p. On that connection, we have not observed any significant degradation in accuracy, even with the lower playback quality.

7 SnailLoad: Website-Fingerprinting

In this section, we present an open-world website-fingerprinting attack on the top 100 websites from the Alexa top 1 million list [3]. We use an other class for randomly sampled websites not in the top 100.

7.1 Evaluation

We evaluate our SnailLoad on in an open-world scenario. We collect 30 traces per classified website (3,000 in total). Since we evaluate an open-world scenario, we collect an additional 750 traces from randomly sampled from the Alexa top 1 million list [3], which are not in the top 100. With 10 seconds per trace and about 6.5 seconds to switch between websites, this yields a data collection runtime of about 17 hours. We randomly split the data for each class into 5 equally large parts and perform a 5-fold cross-validation, while making sure that the test set never overlaps with the training set in the same run. Consequently, in the open-world scenario, test set traces of the other class belong to websites that the model has never seen during training. We train our CNN with a validation split of 10% of the training set.

Our classifier achieved a macro-averaged $F_1$ score of 62.8%. The confusion matrix is shown in Figure 8 and Table 3 in the appendix shows the results for all websites in detail. The color of each cell indicates the prediction probability
Prediction Website
0 10 20 30 40 50 60 70 80 90 100
10 20 30 40 50 60 70 80 90 100
≥ 50

Figure 8: The confusion matrix for our top-100 open-world fingerprinting attack. The macro-averaged $F_1$ score is 62.8%.

Figure 9: Training our neural network on connection ADSL-1 and applying the model to a trace from connection FTTH-1 yields a substantially higher accuracy than random-guessing, as visible with on the diagonal.

for each website, with a diagonal showing the high accuracy of our classifier across the vast majority of websites. The classification did not work well for 10 websites, resulting in $F_1$ scores below 20%. These 10 websites include wikipedia.org, t.co, yahoo.co.jp, jianshu.com, which transfer less than 120 kB and thus are harder to detect for SnailLoad (see Figure 6). For alipay.com and amazonaws.com, we observe relatively slow loading times which may have moved the relevant features outside of our trace, which was restricted to only 10 seconds. All of the above websites had an $F_1$ score below 10%. Four further websites had an $F_1$ score below 20%, where taobao.com, tmall.com, and cnki.net were likely loading too slowly and google.com.hk was mostly misclassified as other Google domains. Excluding these websites, the macro-averaged $F_1$ score increases to 68.5%. For 76 classes, including the other class, the $F_1$ score is above 50%. The low performance for some websites is not surprising as different domains lead to the same actual page, e.g., Google domains are often misclassified as other Google domains.

Figure 10: Detecting that two persons had a video call on Discord with each other. At around 155 s, the video call was started, resulting in an increase of RTTs on both connections. When the call ended at 205 s, the RTTs on both connections returned back to the lower baseline level.

In the cross-connection scenario, we apply a model trained on one connection (ADSL-1) to another connection (FTTH-1). We only performed a test on the top-10 websites in a closed-world scenario to minimize the time involvement for the participants of our study. In this scenario, we still achieve an $F_1$ score of 40% (see Figure 9), which is significantly higher than random guessing.

8 Limitations and Discussion

While SnailLoad shows that obtaining a network latency trace of a victim system from only a TCP connection is feasible, inferring the specific web activity of the victim can be more challenging in practice. For our fingerprinting attacks, we used data from the same network connection for training set and test set. While we demonstrated in Section 7.1 that applying a model trained on one connection to a trace from another connection can yield a significantly higher accuracy than random-guessing, other scenarios may offer the possibility to obtain training and test data through the same network connection. This might be realistic if an attacker has access to the same network connection but in this situation the attacker likely already has other means than SnailLoad to observe the network latency, or even packet transmissions, of the victim. The attacker could also run the fingerprinting through a website, where one iframe shows the website of video to fingerprint, whereas another resource in the website is used to record a latency trace using SnailLoad. Consequently, the attacker has different means to obtain traces and mount an attack that can be adapted to the concrete setup and attacker capabilities. More generally, once SnailLoad leaked traces from a victim, an attacker can take arbitrary time to analyze or store the data until enough information is available to infer videos, websites, or other information contained in the trace.

In contrast to the remote timing side channels discussed in Section 2.5, which focus on JavaScript-based attacks or
special network interfaces, SnailLoad is a remote attack in the sense that we do not rely on the victim system’s own hardware or software implementation but contention of buffers that are in the transport path to the victim (close to the victim). SnailLoad is more similar to passive traffic analysis [7, 47, 71] but, in contrast to these works, does not require a person-in-the-middle attack scenario. SnailLoad instead can target any machine it can reach with a network connection. Gong et al. [22] already observed that ICMP ping times correlate with network activity on a DSL connection in 2010. However, ICMP ping packets are commonly blocked by default today [60]. In contrast to ICMP ping packets, the TCP ACKs used by SnailLoad are fundamental for reliable data transmission and cannot be blocked. Murdoch and Danezis [48] used passive latency measurements to estimate traffic on a Tor relay node. We go significantly beyond their attack and show that SnailLoad can infer specific videos a user watches on arbitrary network connections. In contrast to prior work focusing on information recovery, e.g., with deep-learning techniques [9, 16, 58, 68], SnailLoad focuses on the information channel itself. Hence, SnailLoad could be combined with any of these works for generic non-person-in-the-middle information recovery, significantly amplifying the impact of these attacks.

To provide an example of such a use case of SnailLoad, we show that a scenario similar to the one explored by Li et al. [43] is also possible with SnailLoad. Instead of an instant messenger, we evaluate a video call scenario, where the attacker obtains latency traces of multiple victim systems and tries to infer which of these systems are interacting with each other, i.e., which ones are having a joint video call. Figure 10 shows this for two persons having a short video call with each other on Discord, while the attacker records latency traces of both of them, as described in Section 5.3. The video call runs from $t = 155s$ to $t = 205s$. The attacker observes a clear increase in RTTs for both victims in this specific time frame. Consequently, the attacker can deduce that they had a video call with each other in this time frame, again without requiring a PTM setting or the installation of stalkerware [11].

**Mitigation of SnailLoad.** Mitigating SnailLoad is not trivial as the root cause, the different bandwidths of channels in the transport path, cannot be eliminated. To provide adequate performance to multiple users simultaneously, the backbone network infrastructure has to have a higher bandwidth than the user’s connection. Thus, the critical bottleneck is typically close to the user (i.e., the last mile), and thus, the buffering occurs in a node that also handles the attacker’s packets, even if packets have different priorities, e.g., due to quality-of-service measures. Dropping packets to avoid the bottleneck would not close the side channel, as the attacker could instead measure the share of dropped packets.

Similar as other side channels, SnailLoad is affected by noise and adding additional noise can hinder attacks. Noise is added to contention side channels by adding spurious random contention. In our case, the contention comes from the last-mile bottleneck. Thus, we have to establish random traffic with an external server. We implemented a simple proof-of-concept that for every 5-second interval occupies the network connection randomly between 0% to 100%. To evaluate the effect of this noise on SnailLoad, we run a top-10 website-fingerprinting attack. Without noise, we achieve an $F_1$ score of 77%, with a very clear diagonal visible in the confusion matrix in Figure 11. With noise, the $F_1$ score drops to 15%, when using noisy traces for training and tests. This is just barely above the random-guessing probability and the diagonal does not stand out anymore. When using clean traces for the training, and noisy traces for the tests, the $F_1$ score drops even lower, to 10%, which is the apriori probability of random guessing. This shows that noise can effectively mitigate SnailLoad. However, the noise also introduces an inconvenience for the user as it occupies part of the network connection.

**Ethical Considerations.** For our user study, we approached local undergraduate and graduate students who volunteered to run our measurement script. We asked the students not to perform any operations on the network that could expose personal information while running our script. Furthermore, it is essential to note that we only search for correlations between our measurements and the YouTube videos our measurements script opened. Thus, no personal information of the students was exposed to information leakage at any point. For the remaining possibility that students still used their network during the measurement time frame in a sensitive way, we are planning to destroy the traces after the publication of the paper and provide students with the option to ask us for the deletion of their traces and exclusion of their results from the paper at any point.

**Responsible Disclosure.** We demonstrated our video-fingerprinting attack on YouTube and, therefore, reported our attack to Google on March 9. Google acknowledged the sever-
ity and that it is a generic, unaddressed problem. For YouTube, they are investigating server-side mitigations.

9 Conclusion

While a substantial amount of research has studied person-in-the-middle side-channel attacks on encrypted web traffic, the possibility of such attacks in non-person-in-the-middle scenarios remained unclear. In this paper, we presented a novel attack, SnailLoad, that exploits the network packet latency as a side channel for activities on a victim system. Our root cause analysis indicates that buffering in a transport path node close to the victim is the root of the latency variations, as well as their connection to the bufferbloat quality-of-service phenomenon. We demonstrated SnailLoad in a scenario where the attacker hides the latency measurement in a network connection of the victim to retrieve an asset, e.g., a file or an image, from a seemingly benign attacker-controlled server. SnailLoad requires no JavaScript, no form of code execution on the victim system, no user interaction but only a constant exchange of network packets, e.g., a network connection in the background. We evaluated SnailLoad with a video-fingerprinting attack, inferring what video a user is watching from a single SnailLoad trace. Our user study comprised 10 internet connections with 6 different internet connection technologies. Over a set of the top 10 YouTube videos, we obtain classification $F_1$ scores between 37% and 98%. In a top-100 open-world website-fingerprinting attack, we achieve an $F_1$ score of 62.8%. SnailLoad highlights that numerous prior works on network side channels could potentially be lifted to a non-PITM remote attack scenario.

Acknowledgments

We would like to thank our anonymous reviewers and our shepherd for their valuable and timely feedback. We furthermore thank David Bom, Lukas Maar, Sebastian Felix, Andreas Kogler, Lorenz Schumm, and Martin Glasner. This research was supported in part by the European Research Council (ERC project FSSec 101076409) and the Austrian Science Fund (FWF SFB project SPyCoDe 10.55776/F85 and FWF project NeRAM I6054). Additional funding was provided by generous gifts from Red Hat, Google, and Intel. Any opinions, findings, and conclusions or recommendations expressed in this paper are those of the authors and do not necessarily reflect the views of the funding parties.

References


[59] Gururaj Saileshwar, Christopher W Fletcher, and Moinuddin Qureshi. Streamline: a fast, flushless cache covert-channel attack by enabling asynchronous collusion. In


**Appendix**

Table 3 shows the $F_1$ score for each website in the top-100 open-world fingerprinting attack from Section 7.
Table 3: Websites used in the top-100 open-world fingerprinting attack and their corresponding $F_1$ scores.

<table>
<thead>
<tr>
<th>#</th>
<th>Website</th>
<th>$F_1$ [%]</th>
<th>#</th>
<th>Website</th>
<th>$F_1$ [%]</th>
<th>#</th>
<th>Website</th>
<th>$F_1$ [%]</th>
</tr>
</thead>
<tbody>
<tr>
<td>001</td>
<td>google.com</td>
<td>31</td>
<td>035</td>
<td>office.com</td>
<td>88</td>
<td>069</td>
<td>amazon.co.jp</td>
<td>69</td>
</tr>
<tr>
<td>002</td>
<td>youtube.com</td>
<td>95</td>
<td>036</td>
<td>t.co</td>
<td>6</td>
<td>070</td>
<td>google.co.in</td>
<td>32</td>
</tr>
<tr>
<td>003</td>
<td>baidu.com</td>
<td>31</td>
<td>037</td>
<td>naver.com</td>
<td>61</td>
<td>071</td>
<td>msn.cn</td>
<td>90</td>
</tr>
<tr>
<td>004</td>
<td>bilibili.com</td>
<td>52</td>
<td>038</td>
<td>apple.com</td>
<td>77</td>
<td>072</td>
<td>tencent.com</td>
<td>51</td>
</tr>
<tr>
<td>005</td>
<td>facebook.com</td>
<td>66</td>
<td>039</td>
<td>sina.com.cn</td>
<td>50</td>
<td>073</td>
<td>freepik.com</td>
<td>66</td>
</tr>
<tr>
<td>006</td>
<td>qq.com</td>
<td>63</td>
<td>040</td>
<td>aliexpress.com</td>
<td>85</td>
<td>074</td>
<td>etsy.com</td>
<td>71</td>
</tr>
<tr>
<td>007</td>
<td>twitter.com</td>
<td>82</td>
<td>041</td>
<td>yahoo.co.jp</td>
<td>9</td>
<td>075</td>
<td>amazon.co.uk</td>
<td>63</td>
</tr>
<tr>
<td>008</td>
<td>zhihu.com</td>
<td>85</td>
<td>042</td>
<td>xhamster.com</td>
<td>64</td>
<td>076</td>
<td>imgur.com</td>
<td>80</td>
</tr>
<tr>
<td>009</td>
<td>wikipedia.org</td>
<td>8</td>
<td>043</td>
<td>paypal.com</td>
<td>75</td>
<td>077</td>
<td>jianshu.com</td>
<td>5</td>
</tr>
<tr>
<td>010</td>
<td>amazon.com</td>
<td>54</td>
<td>044</td>
<td>spankbang.com</td>
<td>45</td>
<td>078</td>
<td>ilovepdf.com</td>
<td>68</td>
</tr>
<tr>
<td>011</td>
<td>instagram.com</td>
<td>85</td>
<td>045</td>
<td>pinterest.com</td>
<td>83</td>
<td>079</td>
<td>twitch.tv</td>
<td>80</td>
</tr>
<tr>
<td>012</td>
<td>linkedin.com</td>
<td>38</td>
<td>046</td>
<td>mail.ru</td>
<td>81</td>
<td>080</td>
<td>atlassian.net</td>
<td>87</td>
</tr>
<tr>
<td>013</td>
<td>reddit.com</td>
<td>64</td>
<td>047</td>
<td>ebay.com</td>
<td>71</td>
<td>081</td>
<td>force.com</td>
<td>58</td>
</tr>
<tr>
<td>014</td>
<td>whatsapp.com</td>
<td>72</td>
<td>048</td>
<td>douban.com</td>
<td>70</td>
<td>082</td>
<td>dropbox.com</td>
<td>83</td>
</tr>
<tr>
<td>015</td>
<td>openai.com</td>
<td>90</td>
<td>049</td>
<td>msn.com</td>
<td>94</td>
<td>083</td>
<td>office365.com</td>
<td>47</td>
</tr>
<tr>
<td>016</td>
<td>yahoo.com</td>
<td>53</td>
<td>050</td>
<td>imdb.com</td>
<td>76</td>
<td>084</td>
<td>alipay.com</td>
<td>5</td>
</tr>
<tr>
<td>017</td>
<td>bing.com</td>
<td>66</td>
<td>051</td>
<td>amazon.in</td>
<td>65</td>
<td>085</td>
<td>discord.com</td>
<td>85</td>
</tr>
<tr>
<td>018</td>
<td>taobao.com</td>
<td>14</td>
<td>052</td>
<td>netflix.com</td>
<td>83</td>
<td>086</td>
<td>namu.wiki</td>
<td>50</td>
</tr>
<tr>
<td>019</td>
<td>163.com</td>
<td>49</td>
<td>053</td>
<td>telegram.org</td>
<td>71</td>
<td>087</td>
<td>t.me</td>
<td>63</td>
</tr>
<tr>
<td>020</td>
<td>yandex.ru</td>
<td>80</td>
<td>054</td>
<td>dzen.ru</td>
<td>78</td>
<td>088</td>
<td>wordpress.com</td>
<td>64</td>
</tr>
<tr>
<td>021</td>
<td>xvideos.com</td>
<td>61</td>
<td>055</td>
<td>quora.com</td>
<td>33</td>
<td>089</td>
<td>tradingview.com</td>
<td>87</td>
</tr>
<tr>
<td>022</td>
<td>live.com</td>
<td>78</td>
<td>056</td>
<td>stackoverflow.com</td>
<td>59</td>
<td>090</td>
<td>avito.ru</td>
<td>70</td>
</tr>
<tr>
<td>023</td>
<td>pornhub.com</td>
<td>64</td>
<td>057</td>
<td>sohu.com</td>
<td>21</td>
<td>091</td>
<td>3dmgame.com</td>
<td>73</td>
</tr>
<tr>
<td>024</td>
<td>microsoft.com</td>
<td>72</td>
<td>058</td>
<td>spotify.com</td>
<td>100</td>
<td>092</td>
<td>xiaohongshu.com</td>
<td>55</td>
</tr>
<tr>
<td>025</td>
<td>vk.com</td>
<td>93</td>
<td>059</td>
<td>aliyun.com</td>
<td>93</td>
<td>093</td>
<td>instructure.com</td>
<td>88</td>
</tr>
<tr>
<td>026</td>
<td>zoom.us</td>
<td>84</td>
<td>060</td>
<td>xinxx.com</td>
<td>50</td>
<td>094</td>
<td>onlyfans.com</td>
<td>88</td>
</tr>
<tr>
<td>027</td>
<td>github.com</td>
<td>83</td>
<td>061</td>
<td>1688.com</td>
<td>47</td>
<td>095</td>
<td>amazonaws.com</td>
<td>8</td>
</tr>
<tr>
<td>028</td>
<td>jd.com</td>
<td>44</td>
<td>062</td>
<td>myshopify.com</td>
<td>54</td>
<td>096</td>
<td>flipkart.com</td>
<td>68</td>
</tr>
<tr>
<td>029</td>
<td>weibo.com</td>
<td>71</td>
<td>063</td>
<td>tmall.com</td>
<td>19</td>
<td>097</td>
<td>hao123.com</td>
<td>28</td>
</tr>
<tr>
<td>030</td>
<td>google.com.hk</td>
<td>13</td>
<td>064</td>
<td>indeed.com</td>
<td>69</td>
<td>098</td>
<td>alibaba.com</td>
<td>67</td>
</tr>
<tr>
<td>031</td>
<td>tiktok.com</td>
<td>91</td>
<td>065</td>
<td>deepl.com</td>
<td>69</td>
<td>099</td>
<td>cnki.net</td>
<td>12</td>
</tr>
<tr>
<td>032</td>
<td>canva.com</td>
<td>66</td>
<td>066</td>
<td>pixiv.net</td>
<td>92</td>
<td>100</td>
<td>mediafire.com</td>
<td>52</td>
</tr>
<tr>
<td>033</td>
<td>csdn.net</td>
<td>62</td>
<td>067</td>
<td>feishu.cn</td>
<td>66</td>
<td>101</td>
<td>other</td>
<td>53</td>
</tr>
<tr>
<td>034</td>
<td>fandom.com</td>
<td>70</td>
<td>068</td>
<td>duckduckgo.com</td>
<td>88</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>