Dissecting Mellowtel: Bandwidth-as-a-Service Through Browser Extensions

HyunSeok Daniel Jang
Northwestern University
Evanston, IL, USA
daniel.jang@u.northwestern.edu

Ayush Pandey
NYU Abu Dhabi
Abu Dhabi, UAE
ayush.pandey@nyu.edu

Matteo Varvello
Nokia Bell Labs
Murray Hill, NJ, USA
matteo.varvello@nokia.com

Fabian Bustamante
Northwestern University
Evanston, IL, USA
fabianb@cs.northwestern.edu

Yasir Zaki
NYU Abu Dhabi
Abu Dhabi, UAE
yz48@nyu.edu

Abstract

Browser extensions are widely used, but developers still face limited options for monetization. Mellowtel recently introduced a new approach: *bandwidth-as-a-service*. By embedding its SDK, developers earn revenue from opted-in users who share idle bandwidth to fetch web data for AI labs and startups. We combine large-scale Chrome Web Store measurements, controlled experiments across six AWS regions, and an in-the-wild deployment with 34 students spanning 25 countries to reveal how the SDK operates and the extent of its adoption. We find that while hundreds of extensions bundle the Mellowtel SDK, active monetization is rare: only 16 of 116 Mellowtel-equipped extensions actively inject crawl iframes during a browsing session, collectively reaching far fewer users than the millions Mellowtel claims, making extension counts an upper bound on real participation. Crawling intensity varies sharply by geography, with US-based nodes receiving up to 8.7× more crawl tasks than other regions. Mellowtel's crawl assignments split into two distinct workloads: broad and shallow scraping of a long tail of domains, and repeated structured extraction from a small set of high-demand targets (e.g., Google search, CouponFollow, Reddit). Finally, we identify concrete security concerns: crawl target lists include domains flagged as associated with known malware families, and some users received crawl requests for content banned in their countries. More critically, a dynamic code loading mechanism allows Mellowtel to execute remotely fetched JavaScript inside the user's browser at runtime, bypassing Chrome Web Store review entirely.

CCS Concepts

• **Networks** → *Network measurement*; *Network measurement*; • **Information systems** → *Web mining*.

Keywords

browser extensions, bandwidth monetization, web crawling

IMC '26, Karlsruhe, Germany

ACM ISBN 979-8-4007-2327-8/2026/10
https://doi.org/10.1145/3777912.3839816

ACM Reference Format:
HyunSeok Daniel Jang, Ayush Pandey, Matteo Varvello, Fabian Bustamante, and Yasir Zaki. 2026. Dissecting Mellowtel: Bandwidth-as-a-Service Through Browser Extensions. In *Proceedings of the 2026 ACM Internet Measurement Conference (IMC '26), October 12–16, 2026, Karlsruhe, Germany*. ACM, New York, NY, USA, 17 pages. https://doi.org/10.1145/3777912.3839816

1 Introduction

Browser extensions are a popular way for developers to reach millions of users, but monetization options remain limited; prior approaches have monetized the user's display surface via ad injection [58] or affiliate redirects [27], or the user's compute via cryptomining [53]. Mellowtel recently introduced a new model: *bandwidth-as-a-service* [24]. Developers integrating the Mellowtel SDK can earn revenue by asking users to share unused bandwidth. In practice, this means that opted-in browsers fetch public web content on behalf of paying customers such as AI labs and data startups. Revenue is split between extension developers (55%) and Mellowtel (45%), with projected earnings of about $50 per month for every 1,000 active participants.

Mellowtel's approach raises both opportunities and concerns. Technically, it works by injecting hidden `iframes` into pages a user visits, loading third-party websites, and returning their HTML to background service workers. To enable this, the SDK circumvents browser security headers such as Content-Security-Policy and X-Frame-Options through `declarativeNetRequest` permissions. Security researchers have criticized this approach as turning browsers into an "unwitting botnet" [6, 19] with implications for both privacy and security, though Mellowtel argues its system is privacy-compliant, opt-in, open source, and GDPR-aligned [3].

Mellowtel's model is a browser-extension instance of a broader, increasingly scrutinized ecosystem of Residential IP Proxy (RESIP) services, which route their customers' traffic through the IP addresses of end-user devices. RESIP services leverage a vast, distributed, and dynamic pool of residential IP addresses, making their traffic difficult to distinguish from that of normal users, allowing customers to circumvent detection, geo-restrictions, or IP-based blocking measures. While they could be tools for legitimate use cases (e.g., ad verification, price comparison, application testing), prior measurement studies report RESIP traffic put to malicious activities such as illicit SEO and phishing [44], ad fraud [45], cryptojacking and botnets [65], and phishing and account fraud [28]. Enforcement action has also followed at industrial scale, most notably

by Google, which disrupted IPIDEA, a residential proxy network whose infrastructure had been leveraged by over 550 distinct threat groups [21]. Unlike such classical RESIP services, which typically turn the residential node into a general-purpose relay in which the customer opens a connection and controls the destination and protocol at request time, Mellowtel is purpose-built for web scraping, and its own backend dispatches specific crawl tasks to the browser. This narrower interface nonetheless introduces its own class of risk, which motivates the empirical measurement study we present here.

We take a multi-pronged approach to provide the first systematic characterization of Mellowtel. First, we monitor the Chrome Web Store [39] (CWS) from February to April 2026 to detect extensions bundling the Mellowtel SDK, measuring adoption and user reach. Second, we build a headless experimental tool to analyze Mellowtel's behavior under controlled conditions, deploying the most popular Mellowtel-powered extensions across six geographically distributed AWS regions and running continuous experiments from November 2025 through February 2026, collecting 857k Mellowtel-triggered network requests.

To capture real-world behavior, we complement our controlled measurements with a custom Chrome extension that detects injected iframes carrying the `mllwtl` identifier and intercepts related network requests. We deploy it on laptops used by 34 university students traveling from campus to their homes during the 2025 winter break, collecting approximately 3 GB of logs spanning 2.8 million requests across 25 countries and 5 continents. The natural mobility of participants during travel provided organic geographic diversity across both developed and developing regions, residential broadband, and public Wi-Fi networks, enabling a direct comparison of Mellowtel's behavior under controlled versus real-world conditions.

Our key findings are summarized below and center on two themes: adoption is far more limited than SDK presence alone suggests, and the small share of extensions with Mellowtel actively enabled exposes users to concrete risks.

Lack of monetization. Despite Mellowtel being integrated into hundreds of extensions, active monetization remains rare. Of 116 Mellowtel-equipped extensions flagged by our static detector, only 16 actively inject crawl iframes during a 15-minute browsing session. The remainder show no Mellowtel activity at all, suggesting that SDK presence in source code is a poor proxy for runtime participation. We therefore treat our detection counts as an upper bound on active monetization.

In-the-wild measurements validate controlled experiments and expose geo-blocking. Our in-the-wild deployment across 25 countries confirms that crawling intensity is driven by geolocation, not network conditions. North Virginia stands out as the most active region, with over 26,000 crawl attempts, 5.7× to 8.7× more than any other region. In-the-wild data extends this picture to more countries, with the US, Canada, and Pakistan showing high activity (medians of 35, 24, and 19 MB per 30-minute window, respectively) while Qatar, Nepal, and Spain remain <1 MB. At the extreme, some countries such as Egypt, Saudi Arabia, Bangladesh, Ghana, and the Philippines exhibit zero crawling throughout the entire measurement period, suggesting either deliberate geofencing by Mellowtel or local blocking of its infrastructure.

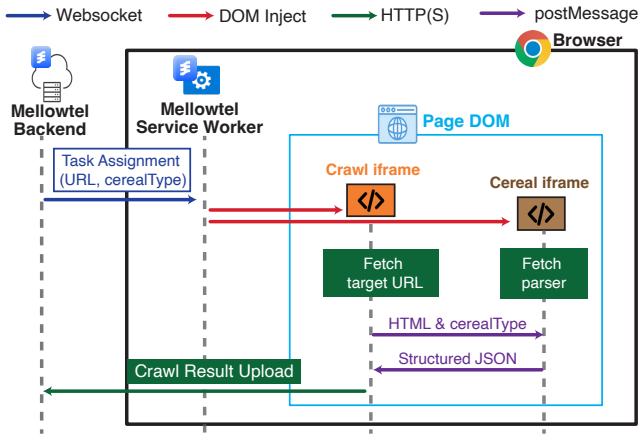


Figure 1: Timeline of Mellowtel's crawling session

Distinct scraping workloads. Across our controlled experiments, 93% of the 12,177 unique domains observed are scraped at only a single cloud vantage point. However, a small set of 43 domains scraped at all six vantage points accounts for more unique URLs (13,892) than domains scraped at only one vantage point combined (13,800), with nearly 65% of those URLs invoking a structured parser that extracts specific fields (e.g., Google search results, coupon metadata). This workload mirrors the offering of commercial residential proxy services, where structured records rather than raw HTML are the product.

Sensitive crawl targets. Across 66,910 unique domains targeted across our in-the-wild and controlled measurements, Mellowtel's crawl lists skew heavily toward business, technology, and e-commerce content, but also include legally sensitive categories: 148 gambling, 52 adult content, and 23 pornography domains, along with a credential phishing site impersonating a Turkish bank. Notably, the UAE vantage point received gambling and adult content targets despite both being prohibited under local law, suggesting Mellowtel's scheduler is geolocation-aware for payout purposes but indifferent to jurisdictional constraints when dispatching targets.

Runtime code injection bypasses extension review. Beyond target selection, Mellowtel fetches and executes parser code from AWS Lambda at runtime in roughly 0.33% of crawl sessions. This dynamic code loading bypasses CWS review entirely, meaning a compromised update to that endpoint would enable arbitrary JavaScript execution in the user's browser with full extension privileges and no user visibility.

2 Mellowtel Under-the-Hood

Mellowtel is a commercial SDK [24] that developers embed in their Chrome extensions to monetize their user base through bandwidth sharing. When a user installs a Mellowtel-enabled extension and opts in, their browser becomes a node in Mellowtel's distributed crawling network, fetching web content on behalf of paying customers such as AI labs and startups. Mellowtel passes 55% of the resulting revenue back to the extension developers.

We describe here the runtime architecture of the SDK, which we reconstructed by instrumenting extensions that bundle it and

observing their DOM modifications and network activity at runtime. Figure 1 shows the typical timeline of activities that Mellowtel performs while a user browses a website. In the following, we walk through each phase in the order it takes place.

Task Assignment. When the extension is first enabled, its background service worker establishes a WebSocket connection to Mellowtel's API endpoint at `ws.mellow.tel`, hosted on AWS API Gateway. Through this channel, the extension periodically receives task assignments, each of which carries a set of parameters that specify a crawling session, including the target URL and (optionally) a `cerealType` identifier indicating the parsing logic to apply to the fetched HTML. Hence, Mellowtel's backend retains full control over what each participating browser fetches, when it fetches, and how results are parsed. The extension maintains this WebSocket session for the duration of the browser session, receiving new tasks as they become available.

Iframe Injection. Once the extension receives a task assignment, it immediately begins execution by injecting iframes into the currently visited page. In our experiments, the first task typically arrived no sooner than 60 seconds after page load, presumably to ensure the host page has fully rendered.

We find that Mellowtel injects two distinct types of iframes, each serving a different role. *Crawl* iframes fetch target websites by setting their `src` attribute to the URL specified from the task assignment, scrape the loaded content, and transmit the results back to Mellowtel's backend. They appear in two variants — single and batch – distinguished by their `data_id` attributes. A single crawl iframe targets one URL, whereas a batch crawl iframe spawns multiple child iframes, each with its own src pointing to one of the target URLs in the assignment, fetching them in parallel. *Cereal* iframes are persistent background iframes, and always load the same URL (hosted on AWS Amplify) [5]. They load a JavaScript bundle (`cereal.bundle.js`, approximately 900KB) that serves as a centralized parsing engine. Both crawl and cereal iframes are injected in every crawl session, though as we explain below, the cereal iframe is not always actively used.

Content Parsing. Whether the cereal iframe participates in a given crawl session depends on the target site being fetched. For certain target domains, the crawl iframe sends the raw HTML along with a `cerealType` identifier to the cereal iframe. The `cerealType` parameter specifies which parser to apply, and the cereal iframe returns the extracted fields as structured JSON back to the crawl iframe. The specific fields depend on the target: for example, Google search pages yield organic results, featured snippets, and AI generated overviews, while pages from a coupon marketplace (e.g., CouponFollow) yield coupon codes, discount descriptions, and expiration dates (see example in Appendix Figure 12). The bundle also implements a dynamic parser loading mechanism that fetches additional parsing logic from a remote AWS Lambda endpoint, allowing operators to deploy or update parsers without modifying the browser extension. Note that this poses a potential security risk, as it enables arbitrary JavaScript execution in the browser (see Section 7.1).

Result Upload. Regardless of whether parsing took place, the crawl iframe packages the result and uploads it to Mellowtel's coordination server at `request.mellow.tel` via an HTTPS POST.

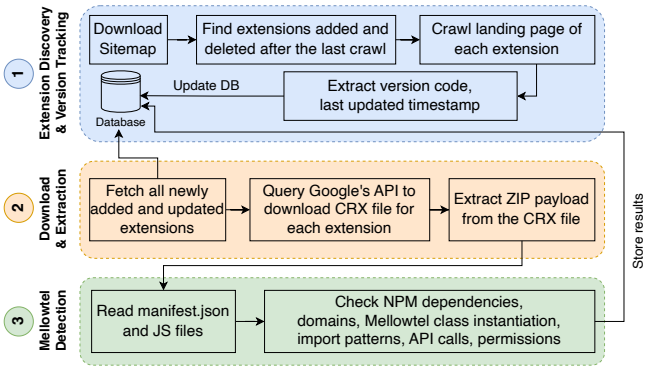


Figure 2: Chrome extension discovery and Mellowtel detection pipeline.

When parsing was performed, the payload contains both the raw HTML and the extracted JSON fields; otherwise, it contains the raw HTML alone. After upload, the crawl iframe is removed from the DOM, and the extension returns to an idle state, awaiting the next task assignment over the WebSocket channel.

3 Mellowtel Detection Pipeline

To systematically investigate the adoption and evolution of Mellowtel, we develop an automated detection and analysis pipeline. Figure 2 shows the pipeline which consists of three main components: 1) extension discovery and version tracking, 2) download and extraction, and 3) Mellowtel detection. We implement the pipeline in Python (v3.8.10). On the initial run, we download and process the full set of extensions available on the Chrome Web Store (CWS) at that time, a one-time bulk operation covering more than 230,000 extensions. Once this baseline snapshot is established, the pipeline runs daily, comparing the current set of extensions against the previous day snapshot to identify newly published extensions, newly released versions, and removed extensions.

To handle the scale of the daily crawl, we run the pipeline with 32 parallel processes on a single multi-threaded machine. We also avoid heavy browser automation tools such as `Selenium` and `Puppeteer`, and instead extract metadata directly from the HTML of each extension's landing page. We also design the crawler to handle transient scraping failures while respecting CWS rate limits. When requests fail because of temporary access restrictions or IP blocking, the crawler retries them by switching to a new VPN location. These events occur only occasionally and do not materially affect our coverage.

3.1 Extension Discovery and Version Tracking

Google periodically updates the CWS sitemap [2], which provides search engines with a list of currently available extensions for indexing. Our scraper uses this sitemap to extract extension IDs (a unique 32-character string) and queries the corresponding CWS landing page[1] for each extension.

For each daily crawl, we first compare the current sitemap with the previous day's sitemap to identify which extension IDs were added and which were removed. We then fetch the landing page

[1] https://chromewebstore.google.com/detail/EXTENSION_ID

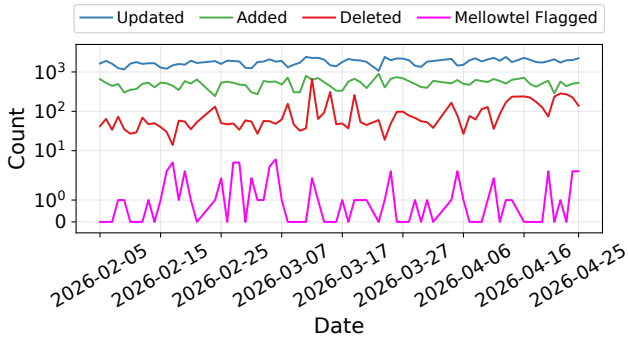


Figure 3: Number of extensions added, updated, deleted, and flagged as "potentially" containing Mellowtel within the added and updated extensions from Feb 5 to Apr 25, 2026.

of each listed extension and extract version-related metadata, including the current version string and the timestamp of the most recent update. Figure 3 shows the daily counts of extensions added to, updated in, and removed from the CWS between February 5 and April 25, 2026, together with the number of extensions flagged by our pipeline as potentially containing Mellowtel. At the median, about 514 extensions are added to the store each day and about 1,787 are updated. About 50 extensions are removed per day. These removals can result from being voluntarily unpublished by their developers, taken down by Google for violating CWS policies, or classified as malware [16].

3.2 Download and Extraction

Chrome extensions are distributed as CRX files [11, 18], a proprietary archive format used by the CWS. A CRX file is essentially a ZIP archive with an additional header prepended at the beginning. This header contains metadata used for packaging and signature verification, while the actual contents remain in the embedded ZIP payload. Current Chrome extensions are distributed in the CRX3 [10] format.

We first retrieve all newly added and updated extension IDs from our database. For each such extension, we query Google's API[2] to download the corresponding CRX file. We then validate that the downloaded file matches the expected CRX format to avoid analyzing empty files or files corrupted in transit. In the CRX3 format, the file begins with a 4-byte magic number "Cr24", followed by a 4-byte format version field, a 4-byte little-endian header length, the serialized CRX3 header, and finally the ZIP archive. We therefore parse the CRX header to determine where the ZIP payload begins. Specifically, for CRX3, we read the header length at byte offset 8 and treat all bytes from offset 12+header_size onward as the ZIP archive. We then use the AdmZip library [14] to extract this ZIP payload, yielding the full extension source code, including JavaScript files, the manifest, HTML pages, and static assets.

[2]https://clients2.google.com/service/update2/crx?response=redirect&prodversion=120.0&acceptformat=crx3&x=id%3D\{EXTENSION_ID\}%26installsource%3Dondemand%26uc

3.3 Mellowtel Detection

The core of our Mellowtel detection pipeline is a multi-stage code analysis engine that applies several detection heuristics to find extensions "potentially" integrating Mellowtel. Table 1 summarizes the nine detection indicators we use. We construct these indicators using both static and dynamic analysis. Specifically, we inspect the SDK's source code, monitor its runtime network and API requests, and analyze a sample of established extensions known to integrate Mellowtel, including Support with Mellowtel [41] and Idle Forest [1]. We then use these fingerprints to detect Mellowtel integrations in the extensions collected by our pipeline. Note that although this fingerprint-construction methodology is applicable to other SDKs, the resulting fingerprints are SDK-specific because initialization logic, code structure, permission requirements, and network behavior differ across SDKs.

We first look at the `manifest.json` file, which is required in all Chrome extensions. It declares permissions, background scripts, content scripts, and other configuration details, useful for identifying permission patterns associated with Mellowtel integration. We particularly look at three suspicious permission combination: 1) `storage` or `unlimitedStorage` (for state management), 2) `declarativeNetRequest` (for request manipulation), and 3) `<all_urls>` (for broader access to all URLs). While this combination is not definitive on its own, it becomes a useful signal when it is paired alongside other indicators. We also check whether `@mellowtel` or `mellowtel` is included as NPM dependency in `package.json`, since extensions built with modern JavaScript tooling often include this file.

We then recursively traverse the extracted extension directory and inspect JavaScript source files for the remaining indicators. We search for import/require statements that reference the Mellowtel library. We further search for direct instantiations of the Mellowtel class, *i.e.,* `new Mellowtel()`. We also look for method calls associated with the Mellowtel API, such as `mellowtel.setOptIn()` and `.optOut()`. In addition, we check for Mellowtel-specific storage keys (starting with "mellowtel_" or "mwt_") used by extensions containing Mellowtel for state management via Chrome's storage APIs. Finally, we search for hardcoded references to Mellowtel-related domains (e.g., *(api.)mellowtel.com/net/io*) and generic string occurrences of the term "mellowtel".

An extension is marked as "potentially" containing Mellowtel if any indicator is found. In practice, extensions that genuinely integrate Mellowtel trigger multiple indicators. Figure 3 shows the timeline of number of such flagged extensions from Feb 5 to April 25, 2026. The median number of Mellowtel-flagged extensions is 1 per day, with a maximum reaching up to 6 in our observation window. For each such extension, we record what indicators were found if there were any, and further manually examine Mellowtel traffic on the browser to avoid false positives. This check verifies whether, within 15 minutes, a candidate extension sends any requests to Mellowtel and injects at least one iframe into the DOM of the active Chrome tab. The 15-minute duration is an upper bound and is 15× longer than the iframe injection time we observed in active Mellowtel extensions, *i.e.,* the first 60 s. This duration can be increased for greater confidence. For example, less active extensions

Indicator	Pattern
Permissions	`declarativeNetRequest`, `storage`, `unlimitedStorage`, `<all_urls>`
NPM dependency	`mellowtel`, `@mellowtel`
Class instantiation	`new Mellowtel()`
Import statement	`import/require('mellowtel')`
Configuration key	`configuration_key: "..."`
Method calls	`.start()`, `.optOut()`, etc.
Storage keys	`"mellowtel_*"`, `"mwt_*"`
Domain references	`mellowtel.com`, `api.mellowtel.*`
Generic strings	`"mellowtel"` (substring)

Table 1: Mellowtel detection patterns.

can be left running for an entire day or week to monitor longer-term injection patterns.

3.4 Validation and Limitations

Our pipeline detects extensions that ship the Mellowtel SDK, but shipping the SDK does not imply that an extension actively monetizes user bandwidth at runtime. We find that 116 extensions trigger more than one indicator mentioned in Table 1. 96 were found in our initial crawl and 20 were found later among added and updated extensions. We use these 116 extensions as a sample to determine which merely ship the Mellowtel SDK and which actively inject Mellowtel iframes at runtime.

We instrumented each one with the headless tool described in Section 4.1. Specifically, we installed each extension in a clean profile, opted in where prompted, and recorded two signals over a 15-min browsing session: whether the background service worker established a WebSocket session with Mellowtel's backend to report client telemetry (e.g., device identifier, SDK version), and whether the extension injected crawl iframes carrying the `mllwtl` identifier.

We observed three qualitatively different behaviors. A first group, which includes Idle Forest [1] and Support With Mellowtel [41], opens the WebSocket and proceeds to inject crawl iframes throughout the session, fetching target sites and uploading results as expected. A second group also opens the WebSocket and registers with the backend, but does not perform any crawling within the 15-min interval, despite explicit opt-in from the client. LinkedIn ConnectEz [47] and NES Emulator [23] exhibit this pattern. A third group carries the SDK in its source but exhibits no Mellowtel-related network activity at all, possibly due to incomplete integration or build time inclusion without runtime initialization.

Among the 116 extensions we manually investigated, only 16 (13.7%) fell into the first group, 3 (2.6%) fell into the second group, and 97 (83.6%) fell into the third group. We therefore treat extensions reported by the Mellowtel detection pipeline as an upper bound, and we focus our controlled measurements (Section 5) on extensions confirmed to inject crawl iframes, since they are the ones that produce the network behavior of interest. We plan to continue monitoring these extensions over time to track whether actual crawling activity increases. As adoption grows, automating this process becomes challenging, particularly in the presence of opt-in settings and other extension-specific behaviors that are difficult to observe without manual intervention. We left such automation as future work.

Finally, our pipeline targets the CWS, while Mellowtel already supports Edge and has announced Firefox and Safari support [40]. The CWS is by far the largest browser extension marketplace, so our adoption counts are likely an upper bound on overall Mellowtel reach across stores. Importantly, our detection methodology is store-agnostic: the same SDK fingerprints and network indicators apply regardless of the distribution channel. Extending the pipeline to the Edge Add-ons store, and to Firefox and Safari once Mellowtel support matures, is a natural direction for future work.

4 Measuring Mellowtel

To study Mellowtel's network behavior, we design two complementary methodologies. The first is a *controlled*, containerized experimental platform that automates browser sessions in a reproducible cloud environment. The second is a lightweight Chrome extension deployable to real users, enabling *in-the-wild* measurements across diverse, uncontrolled browsing scenarios.

Both tools share a common detection primitive. A DOM observer (MutationObserver) watches for injected iframes whose `id` or `data-id` attribute contains "*mllwtl*". When such an iframe is detected, the tool extracts the hostname from its src (e.g., `example.com`) and adds it to an in-memory set of tracked domains; subsequent HTTP requests to any tracked domain or any of its subdomains (e.g., `api.example.com`) are logged. Independently of iframe detection, both tools always monitor traffic to `request.mellow.tel`, the endpoint to which crawl iframes upload scraped results. This domain-scoped filtering is what allows us to separate Mellowtel-attributable traffic from the unrelated third-party requests a typical page generates, such as analytics, advertising, and CDN resources.

4.1 Headless Experimental Tool

The headless tool is designed for deployment across geographically distributed cloud vantage points, packaged as Docker [15] image. The system combines Chrome automation through Selenium WebDriver [55] with selenium-wire to intercept network traffic, running atop X Virtual Frame Buffer (Xvfb) [64] to provide a virtual display environment without requiring physical graphics hardware.

This data collection tool revolves around a Python controller script that orchestrates the experimental workflow. For each run, the script launches a fresh Chrome profile with the target extension preloaded and sequentially visits websites from a configurable list. While staying at each website for 5 minutes, it monitors for Mellowtel iframe injection by polling the DOM every 2 seconds. Detected iframes are tracked via the shared primitive described above, and requests matching the tracked domains (or directed at `request.mellow.tel`) are captured. Network requests are aggregated in memory by iframe URL and written to disk when the iframe disappears from the DOM. Each run produces three artifacts organized under a timestamped directory: a log of all intercepted network requests attributed to Mellowtel iframes, metadata recording the lifetime of each injected iframe (e.g., appearance and disappearance timestamps, `src`), and the raw payloads posted to Mellowtel's backend.

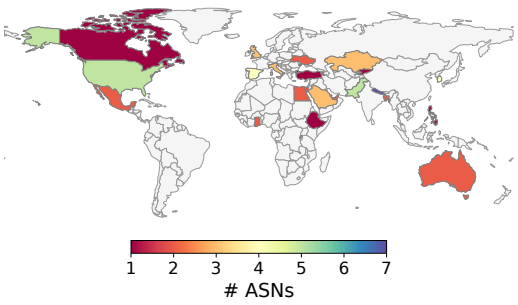


Figure 4: Number of ASNs by country observed in our in-the-wild measurements worldwide.

Vantage Point	# Runs	First Run	Last Run
Cape Town	1,583	2025-11-19	2026-02-09
Frankfurt	1,611	2025-11-19	2026-02-09
London	1,732	2025-11-19	2026-02-09
Tokyo	1,581	2025-11-19	2026-02-09
North Virginia	870	2025-12-20	2026-02-09
UAE	839	2025-12-22	2026-02-09

Table 2: Number of experiment runs and measurement window per AWS vantage point.

Finally, to avoid Mellowtel treating the session as automation, the Python controller disables Selenium's automation flags, randomizes visit order across runs, and periodically scrolls each page to simulate natural user activity.

4.2 Monitoring Extension

The monitoring extension is our in-the-wild counterpart to the headless tool. Rather than driving a scripted browsing session from a cloud vantage point, it runs inside a user's regular Chrome profile and records Mellowtel-attributable network activity during ordinary browsing. The extension follows the Chrome Manifest V3 architecture [20], splitting functionality between a background service worker and page-level content scripts. A *content script* implementing the shared iframe detection primitive (described above) is injected into every webpage; it extracts the iframe's source domain and notifies the background service to begin tracking that domain. The service worker then records browser requests whose destination hostname matches this domain or `request.mellow.tel` using Chrome's `webRequest` API. It captures four stages of HTTP requests: request initiation, outgoing headers, incoming response headers, and final completion or failure.

The extension requires the following permissions: `webRequest`, `declarativeNetRequestWithHostAccess`, `host_permissions` for `<all_urls>`. These permissions enable comprehensive network monitoring, but also raise privacy considerations, which we discuss in Appendix A.1.

Data Collection. For each intercepted request to a tracked domain, the extension records request metadata such as URL, method, and type; request headers and bodies when available; response status and headers; and completion or error information. Due to browser security restrictions, response bodies cannot be captured.

5 Deployment and Data Collection

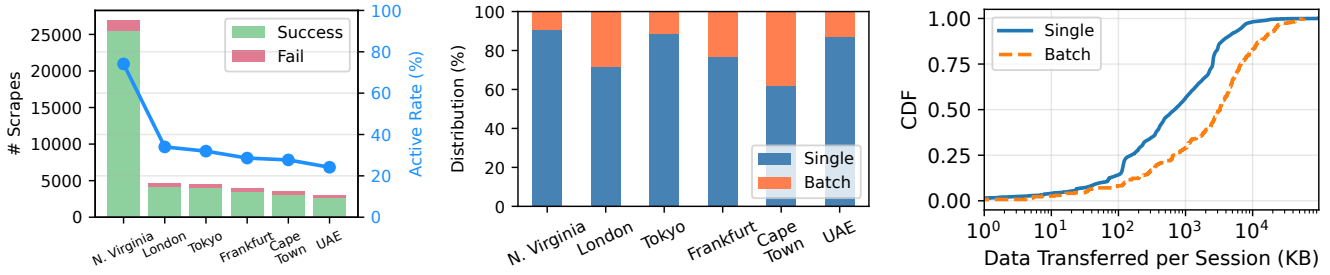
5.1 Controlled Experiments

To characterize Mellowtel's network behavior under controlled conditions, we deploy our headless experimental tool (Section 4.1) across six geographically distributed AWS regions: Cape Town (af-south-1), Frankfurt (eu-central-1), London (eu-west-2), North Virginia (us-east-1), Tokyo (ap-northeast-1), and UAE (me-central-1). We deploy a t3.large instance (2 vCPUs, 8 GB memory) in North Virginia and t3.medium instances (2 vCPUs, 4 GB memory) in the remaining five regions. While both instance types provide identical network performance (5 Gbps baseline and burst bandwidth), t3.large offers 30% higher CPU performance, which is necessary for North Virginia due to significantly higher network activity observed in preliminary tests that caused frequent crashes on the t3.medium configuration.

Each experimental browser instance is configured either with `Support With Mellowtel` [41] or `Idle Forest` [1], two extensions reputedly developed by members of the Mellowtel team [6]. We select these extensions because their direct connection to Mellowtel developers makes them unlikely to have the SDK removed in future updates, ensuring measurement continuity throughout our study period. Note that while our controlled measurements characterize Mellowtel's behavior as observed specifically through the two extensions, actual task assignment and monetization may vary across different extensions depending on the specific business relationship between Mellowtel and the extension developers (Section 3.4).

Experimental Protocol. Each experiment run visits a curated list of eight websites sequentially, spending five minutes per site for a total runtime of approximately 40 minutes per run. Our website selection prioritizes popular news outlets that exhibit dynamic landing pages without authentication requirements: *BBC*, *CNN*, *New York Times*, *Washington Post*, *Associated Press*, *NBC News*, and *The Verge*. These sites are predominantly served through global CDN providers (Cloudflare and Fastly), with the exception of NBC News, which is hosted on AWS EC2 instances in Oregon and North Virginia. We also include a personal website (anonymized) served via Fastly CDN with minimal organic traffic to examine whether website popularity impacts Mellowtel's scraping behavior. Finally, before each experiment run, we measure baseline network conditions using Ookla Speedtest [48] and TCP ping measurements to `request.mellow.tel`, Mellowtel's server endpoint.

Data Collection Period. Each AWS instance executes one experiment run per hour. We conducted measurements from November 19, 2025 through February 9, 2026, yielding 8,216 experiments in total across six vantage points (VPs). Table 2 summarizes the run counts per region, which are not uniformly distributed for two reasons. First, North Virginia and UAE began collection later than the other four regions (on December 20th and 22nd, 2025, respectively). Second, sporadic system failures, such as Selenium crashes and EC2 instance reboots, interrupted individual runs at each VP; we adjusted the experiment frequency at affected regions as needed to recover from these failures and maintain continuous collection. This sustained measurement period captured temporal variations



(a) (b) (c)

Figure 5: (a) Crawl attempts (bars, left axis) and Active Rate (lines, right axis) across vantage points. (b) Distribution of Single vs. Batch crawl sessions per vantage point. (c) CDF of data transferred per crawl session, broken down by Single and Batch modes.

in Mellowtel task assignment and potential regional differences in bandwidth utilization patterns.

5.2 In the Wild Experiments

We deployed the monitoring extension along with the "Support with Mellowtel" extension [41] to 34 university students during the December 2025 holiday break. Each participant received: 1) installation instructions for Chrome on their desktops; 2) a consent form explaining that the extension monitors network requests associated with iframes injected by mellowtel, unrelated to their traffic; 3) a unique participant ID embedded in the extension for identifying log sources.

Students were asked to browse the Web normally during the holiday period, which typically involved 2-3 weeks of travel from university campuses to home locations across different countries and regions. This natural mobility provided geographic diversity without requiring explicit coordination. Figure 4 highlights the number of ASNs observed in our in-the-wild measurements across 25 countries in 5 continents, including both developed regions (US, Canada, Australia, Italy, etc.) and developing regions (Pakistan, Nepal, Bangladesh, Egypt, etc.). Network types spanned residential broadband and public Wi-Fi. Because students were often traveling within and across countries, the access networks from which they connected changed over time, leading to realistic usage observations from multiple ASNs.

The extension was configured to buffer logs locally and upload them periodically to our collection server via HTTPS. Logs included timestamps, participant IDs and current IP address, detected iframe URLs, network request/response data associated with iframe URLs, data POSTed back to Mellowtel servers. In addition, the extension made a heartbeat API call every 7 minutes to indicate that the participant's browser was active and the extension was running. We collected approximately 3 GB of logs over the deployment period, spanning 2.8 million Mellowtel-triggered network requests.

6 Dissecting Mellowtel

6.1 Crawling Intensity

Geolocation. We begin by examining how aggressively Mellowtel performs web crawling across geographic regions. We define the *active rate* as the fraction of (five-minute) website visits during which Mellowtel injected at least one crawl iframe (single or batch). Intuitively, this metric captures the probability that Mellowtel initiates scraping activity while a user visits a given website. The blue lines in Figure 5a (right y-axis) show the active rate for each of our six AWS vantage points. North Virginia stands out with an active rate exceeding 74%, meaning Mellowtel triggered crawling in the vast majority of website visits. The remaining five regions range between 24% and 34%, roughly a factor of 2.5× lower. One plausible explanation is Mellowtel's tiered pricing policy, which advertises higher payouts for traffic originating in the US, France, Italy, Spain, and China [42]. Under such policy, Mellowtel's task scheduler would have a stronger incentive to assign crawl tasks to nodes in premium regions, consistent with the disparity we observe.

The bar plots in Figure 5a (left y-axis) show the number of crawl attempts, defined as instances in which a crawl iframe reported a scrape to Mellowtel's server. To ensure a robust comparison across regions with differing experiment counts, we restrict this analysis to the time windows during which all six vantage points were simultaneously active, retaining 81.2% of all crawl attempts. We find that North Virginia recorded 26,931 attempts, 5.7× to 8.7× more than any other vantage point, consistent with its elevated active rate. Not all attempts succeed: the pink segments indicate failures, predominantly HTTP 403 (authorization) and 502 (server-side) errors. The failure rates ranged from 5% in North Virginia to 13% in Cape Town.

Network Conditions. Mellowtel states that its SDK avoids degrading the host page's performance by rate-limiting crawl activity to the client's available bandwidth [43], which suggested network conditions as a plausible driver of the active-rate disparity: either because Mellowtel's scheduler adapts its own throttling to a client's link quality, or because slower links simply fail to complete assigned tasks within our measurement window. To test this hypothesis, each vantage point recorded two measurements before every experiment run: available downlink and uplink bandwidth via Ookla Speedtest [48], which selects its own nearest measurement server, and round-trip latency to Mellowtel's own endpoint, `request.mellow.tel`, via TCP ping.

All vantage points sustained over 1 Gbps in both downlink and uplink, while latency measurements ranged from <1 ms in North Virginia (co-located in us-east-1) to 224 ms in Cape Town (see Appendix Figure 13). To test whether this latency differential alone could explain the disparity, we applied synthetic network conditions

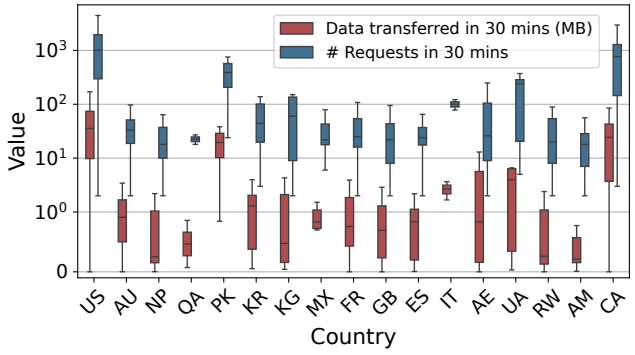


Figure 6: Box plots of the distribution of total data transferred and number of outbound requests within a 30-min window, separated by country.

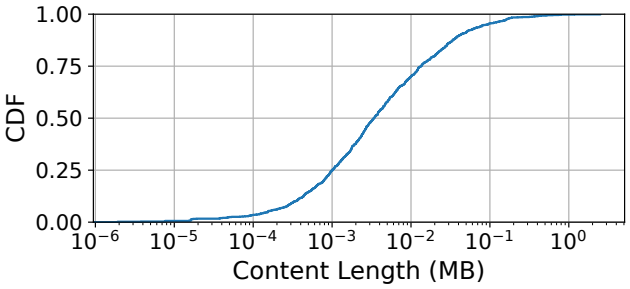


Figure 7: CDF of response content lengths of 1,017 requests made within 30 mins in the US.

on a random subset of North Virginia experiments using Linux `tc` [37], emulating the higher latency observed at other vantage points (100 ms, 200 ms) and ISP's aggressive bandwidth throttling (2 Mbps) [35]. None of the emulated conditions reproduced the 2.5× gap seen across regions (see Appendix A.2).

A related concern is that Mellowtel's scheduler may treat AWS IP space differently from non-cloud addresses, biasing our cross-region comparison. To rule this out, we replicated the UAE experiment on a Linux desktop attached to campus Ethernet in Abu Dhabi, the same city that hosts the AWS me-central-1 region, for 15 days in January 2026. The two deployments produced near identical active rates (20.1% AWS vs 19.3% campus) and single versus batch splits (92.7% vs 92.4% single); see Appendix A.3. Taken together, these controls indicate that Mellowtel's task scheduling is primarily driven by client geolocation rather than by network performance, and is not an artifact of measuring from AWS.

In-the-wild Validation. To validate this pattern beyond our controlled setup, we examine Mellowtel's activity across 25 countries in the wild. Because students use their devices at irregular intervals (unlike the controlled experiment with a fixed window), we report data transferred and outbound request counts per 30-minute window rather than active rate. We consider only intervals in which we consistently received iframe injections no more than 5 minutes apart. This threshold distinguishes periods of active monitoring from those in which the participant's device or Chrome browser–and consequently our monitoring extension–was likely inactive. Excluding these inactive periods prevents intervals without measurements from artificially lowering the observed request counts and data transfer. The box plots in Figure 6 show the distributions of total data transferred and the number of outbound network requests generated across time windows. The figure only shows 17 out of the 25 countries since five did not receive any iframe injections: Egypt, Saudi Arabia, Bangladesh, Ghana, and the Philippines. Two others, Ethiopia and Kazakhstan, received iframe injections, but at a comparatively lower frequency, that is, not within 5 minutes of one another, so we omit them from the plot. The last remaining country, Turkey, did not receive enough heartbeat API calls for statistical analysis.

The spread of the box plots in Figure 6 indicates that Mellowtel's activity is bursty rather than constant: some windows contain little to no traffic, while others show sharp increases in both transferred bytes and number of requests. US stands out as the most active region, with a median of 35 MB and peaks reaching up to 1.4 GB of data transfer in 30 mins. Similarly, Canada and Pakistan show higher activity than most other regions. In Canada, median bandwidth consumption remained at 24 MB and its peak reached 10× higher at 240 MB. Unlike US and Canada, Pakistan shows a smaller spread with a median of 19 MB and a peak of 38 MB. By contrast, countries such as Qatar, Nepal, Spain, and Armenia remain consistently low, with typical transferred volume below 1 MB and request counts below 25 per 30-min window.

Request counts follow a similar pattern. The US, Canada and Pakistan consistently generate more requests than the other countries. In the US, number of requests reaches about 1,000 at the median and nearly 10,000 at the peak within a 30-min window. Despite this high request volume, the median bandwidth remains only 35 MB, which indicates that many of these requests return small response bodies. One such example is shown in Figure 7 as a CDF of response content lengths of 1,017 requests made in the US within 30 mins. Overall, the measurements in the wild are consistent with the controlled experiments and show that Mellowtel's network activity depends strongly on geographic location.

6.2 Crawling Session Structure

We next examine the structure of individual crawl tasks. Recall from Section 2 that each crawl iframe operates in one of two modes: single (targeting one URL) or batch (spawning child iframes to scrape multiple URLs in parallel). We define a *crawl session* as the full set of network activity generated by one crawl iframe instance, encompassing resource fetches for the target site(s) and the subsequent upload of raw HTML alongside structured fields extracted by the cereal iframe to Mellowtel's server.

Figure 5b compares the share of single versus batch sessions across vantage points. In North Virginia, Tokyo, and the UAE, single sessions dominate, with batch sessions accounting for only 9% to 13% of the total. In contrast, London, Frankfurt, and Cape Town exhibit a notably higher share of batch sessions (23% to 38%). One possible explanation is that Mellowtel's scheduler batches multiple targets per iframe in regions where demand is lower, amortizing the overhead of activating a node. In high-demand regions such

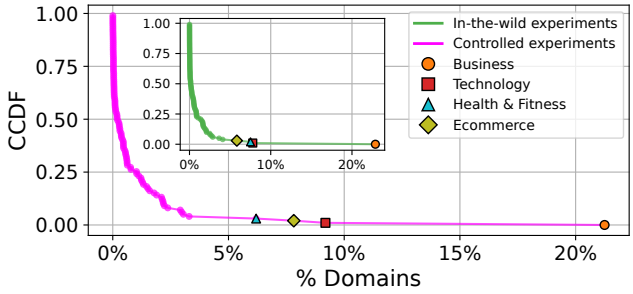


Figure 8: Complementary CDF of the percentage of domains per content category, as classified by Cloudflare's Domain Intelligence API.

as North Virginia, the scheduler could dispatch single-target tasks more frequently. However, we note that this cannot be confirmed without visibility into Mellowtel's scheduling logic.

To quantify the bandwidth consumed by Mellowtel per crawl session, we sum the bytes downloaded by the crawl iframe and, for batch sessions, its child iframes. Download bytes come from the `Content-Length` response header for 2xx responses. 26.2% of 2xx responses (29.2% of unique URLs) lacked a `Content-Length` header, predominantly HTTP/2 streams for which the header is optional; chunked transfers (`Transfer-Encoding: chunked`) only accounted for 0.4% of tracked responses. We recover the missing sizes by re-fetching each URL with headers matching the original Chrome request (e.g., User-Agent, Accept) and recording the compressed on-wire response size, successfully retrieving 84.9% of them. To bound the error from temporal drift, we re-fetched a 1,500-response sample with known `Content-Length` values, allocated proportionally across
`Content-Type` categories; sizes matched near exactly (see Appendix A.4). Note that because HTTP/2 separates header delivery from body completion, re-fetched sizes upper-bound the bytes actually delivered for streamed responses, in the conservative direction for our bandwidth characterization.

The volume of data transferred per session differs substantially between single and batch modes. Figure 5c plots the CDF of per-session download. Single sessions have a median of roughly 687 KB, while batch sessions are roughly 5× larger at 3.3 MB, reflecting the multiple target pages loaded in parallel. The distributions exhibit heavy tails, with transfers exceeding 10 MB occurring in 1.8% of single sessions and roughly 17% of batch sessions. The largest session we observed downloaded 96.9 MB at the Frankfurt vantage point while scraping a website of an animation studio [3], of which 549 of 573 requests were images. A comparable session in North Virginia downloaded 89.7 MB while scraping a Nike sitemap composed of four large XML files. These outliers underscore that bandwidth consumption can be substantial in extreme cases, driven by media-rich pages or bulk-data targets.

6.3 Crawl Targets

Domain Classification. Throughout the controlled experiments, Mellowtel targeted 23,253 unique domains. In the wild, it targeted 46,896 unique domains. To categorize these domains by content type, we queried Cloudflare's Domain Intelligence API [13], which returned classifications for 56,070 (83.8%) of the 66,910 unique domains queried. We selected this API for its breadth of coverage; it aggregates signals from Cloudflare's global network, various third-party intelligence feeds, proprietary ML classifiers, and community feedback [12], making it one of the most comprehensive domain classification sources publicly accessible. Note that the mapping is not strictly one-to-one, since a single domain can belong to multiple categories (e.g., `wsj.com` is labeled both "News and Media" and "Economy and Finance"); 21,101 domains (38.1%) carried two or more category labels.

Figure 8 plots the complementary CDF (CCDF) of the percentage of domains per category in both controlled and in-the-wild experiments. The distribution is heavily skewed: roughly 70% of categories contain fewer than 100 domains, while four categories stand out as clear outliers: "Ecommerce" (4,177), "Health & Fitness" (4,356), "Technology" (5,044), and "Business" (13,784). Beyond mainstream content, we also find that Mellowtel's crawl targets include domains flagged under sensitive or questionable categories: 148 domains classified as "Gambling", 52 as "Adult Themes", and 23 as "Pornography". These domains may be subject to legal restrictions in certain jurisdictions, with implications we revisit in Section 7.

Scraping Taxonomy. We next conduct a finer-grained analysis of the URLs scraped at each vantage point (VP) to understand whether Mellowtel's crawl assignments are partitioned across regions and what kinds of websites are targeted. We restrict to the controlled experiments during the time window when all six VPs were simultaneously active to ensure a fair comparison across regions, yielding 12,177 unique domains. To characterize their geographic footprint, we first attribute each domain to a country using its ccTLD when available, and otherwise the country of its hosting IP, obtained by resolving the domain name. Across VPs, the resulting country mix is similar, dominated by US, Germany, and Netherlands (see Appendix A.5 for a detailed breakdown). We note, however, that this procedure excludes 54% of domains hosted on global CDNs (up to 63.5% in Tokyo).

The specific domains targeted at each VP, however, differ sharply. Figure 9a shows the number of unique domains (blue bars) and unique URLs (orange bars) scraped at exactly k VPs, for k from 1 to 6. Of all 12,177 unique domains, 93.0% (11,322) appear at exactly one VP, and the count drops sharply with k, reaching only 43 at all six. However, the 43 domains scraped at all six VPs account for more unique URLs (13,892) than the much larger set of domains scraped at only one (13,800). Figure 9b provides a closer look at this asymmetry, plotting the distribution of unique URLs per domain, broken down by k. As k grows, the distributions shift toward higher URL counts, indicating that domains scraped from more VPs are also scraped over many more URLs.

These two patterns correspond to qualitatively different scraping behaviors. The domains observed from only one VP are consistent with broad and shallow web crawling: many distinct domains are fetched only a few times each, over a small number of URLs. Because many such tasks circulate through the network, any given task may be handled by whichever node is available, with little repetition across regions. The collected raw HTML is uploaded to Mellowtel

[3] https://www.aranca.com/practices/investment-research

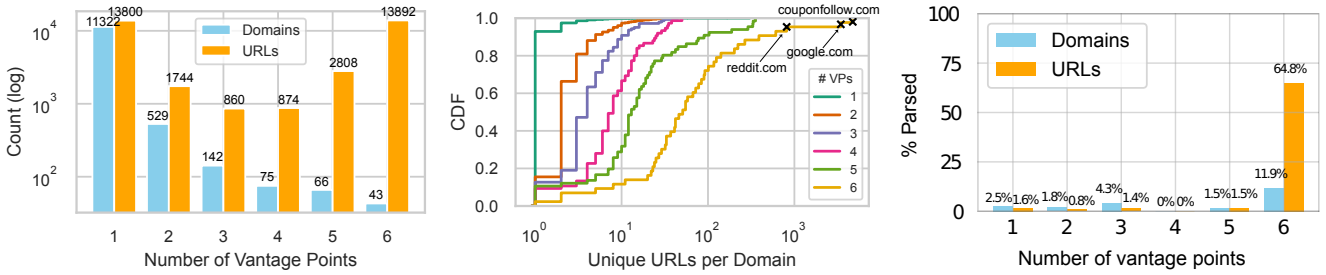


Figure 9: (a) Number of unique domains (blue) and unique URLs (orange) scraped at exactly k AWS vantage points, for k from 1 to 6. (b) Distribution of unique URLs per domain, grouped by the number of vantage points at which each domain was scraped. (c) Percentage of domains (blue) and URLs (orange) parsed by a cerealType parser, broken down by the number of VPs at which the domain was scraped.

backend without any parsing, plausibly bound for bulk web content collection.

In contrast, domains scraped across multiple regions reflect targeted scraping with narrow and deep coverage: a small set of high-demand domains scraped repeatedly across many distinct URLs. One example is search engine result harvesting: Google accounts for 3,445 unique URLs of which 99% are search result pages of the form "/search?q=", parsed into organic results, featured snippets, and AI overview. CouponFollow was scraped across 4,734 URLs (85% fetched exactly once, with a small periodically refreshed subset), with a dedicated parser to extract specific coupon metadata (see Appendix Figure 12 for an example). Similarly, Reddit (814 URLs) invokes a parser that extracts specific posts and comments. Figure 9c quantifies this concentration; for $k \leq 5$, less than 2% of URLs invoke a parser and raw HTML is uploaded as is. At $k = 6$, 64.8% of URLs from just 12% of domains invoke a parser, indicating that a small set of target domains account for the bulk of parsed web content. This resembles the workloads served by commercial residential proxy networks [9, 44], where structured data from specific websites are the product rather than raw HTML at scale.

It is also worth noting that the regional distribution within these recurring domains is uneven. Of the 66 domains present at five of six vantage points, 35 are missing specifically from North Virginia, even though Mellowtel is most active there. The list skews toward US government and educational domains (e.g., war.gov, rural.gov, fau.edu, tech.cornell.edu, police.uci.edu), alongside several major commercial sites (e.g., eventbrite.com, redbubble.com); see Appendix Table 4 for a full breakdown. One plausible reading is that Mellowtel's scheduler avoids dispatching such targets to US-based IPs, perhaps because they are more likely to be flagged or blocked by these sites. Another potential explanation is that these gaps reflect customer-specified targeting, for example a paying customer excluding US vantage points from a particular crawl list. Confirming either explanation would require visibility into Mellowtel's task assignment logic or acting as a paying customer ourselves, neither of which was available to us.

7 Implications

Having characterized Mellowtel's behavior at runtime, we now turn to what its design implies for the parties involved in its distributed scraping operation. We organize this discussion around two stakeholders: the participants who opt in to bandwidth sharing, and the third-party sites whose content is fetched on their behalf.

7.1 Impact on Users

Absent Threat Intelligence Filtering. We first examine whether Mellowtel's crawl targets include domains with known security risks and malware. We again refer to Cloudflare's Domain Intelligence API for threat intelligence signals associated with each domain in the combined set of crawl targets from controlled and in-the-wild measurements. Cloudflare flagged 174 (0.3%) unique domains under "suspected malware family" label, predominantly under families based on domain generation algorithms (DGAs) such as Locky and Dyre. To cross-validate these flags, we queried VirusTotal [61], a service that aggregates verdicts from ~60 antivirus engines and URL scanners, and applied the conventional consensus threshold of three or more engines [36, 51]. Only one domain, `yapikronline.com`, met the threshold, flagged by six engines as *malicious* and a seventh as *suspicious*. Manual inspection reveals that `yapikronline.com` mimics the branding of Yapı Kredi, a major Turkish bank, while operating on an unrelated domain, a pattern characteristic of credential phishing campaigns targeting online banking customers. While the absolute fraction of flagged domains is small, the fact that they appear at all suggests that the crawl lists submitted by paying customers undergo limited (if any) filtering against threat intelligence sources.

Dynamic Code Loading. As described in Section 2, the cereal iframe serves as a parsing engine that converts raw HTML from crawl targets into structured data, with the crawl iframe specifying which built-in parser to apply via a `cerealType` parameter. Parsing is invoked in only 22% of crawl sessions in our controlled experiments; the remaining 78% transmit raw HTML directly to Mellowtel's server. When parsing is invoked but no `cerealType` is provided, the `cereal.bundle.js` bundle falls back to a dynamic loading mechanism, fetching parser code from a remote AWS Lambda

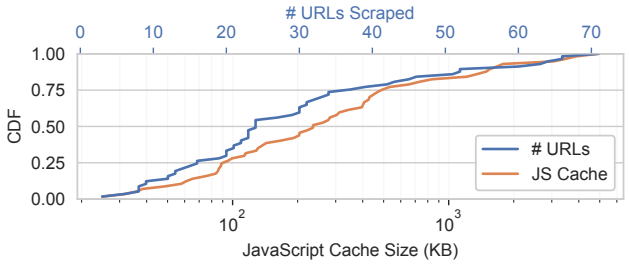


Figure 10: CDF of compiled JavaScript added to the V8 code cache (orange, bottom axis) and number of URLs scraped (blue, top axis) per experiment run.

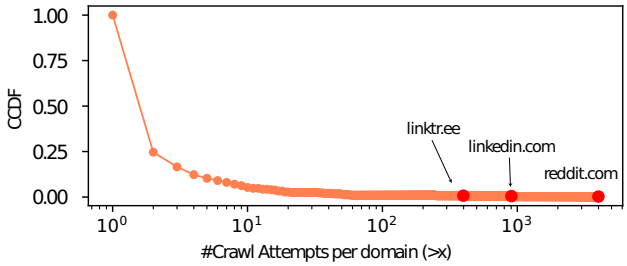


Figure 11: CCDF of the number of Mellowtel's crawl attempts for domains whose robots.txt disallows scraping under the wildcard user agent.

endpoint and executing it at runtime (Appendix Figure 17). We observed this fallback in 1.5% of cereal iframe invocations, corresponding to roughly 0.33% of crawl sessions. In every observed instance, the fetched code performed standard HTML parsing, extracting structured fields such as titles, links, and metadata from the target page without exhibiting behavior beyond its stated parsing role.

The security risk lies in the mechanism itself rather than in the benign behavior we observed. Because the remotely fetched code runs with the same privileges as the extension itself, a compromise of the Lambda endpoint or a malicious update to the hosted parser would enable arbitrary JavaScript execution within the user's browser. Crucially, this execution occurs without any code review by the browser extension store, since the parser logic is retrieved at runtime rather than bundled in the reviewed extension package. The user has no visibility into what code is fetched or executed, and the extension's permissions (including <all_urls> host access and declarativeNetRequest) amplify the potential impact of any injected payload.

Jurisdictional Mismatches. Mellowtel's scheduler uses geolocation for payout purposes (Section 6.1), but does not appear to account for jurisdictional constraints when assigning target sites to crawl, potentially exposing users to regulatory risk. For instance, across our cloud and in-the-wild experiments, our UAE vantage points crawled 10 gambling and 11 pornographic websites (classified per Cloudflare's domain intelligence API), despite both categories being explicitly restricted under local law. Mellowtel exposes no interface for participants to filter or preview assigned targets, so the URL is fetched before the user has any opportunity to inspect it. Whether these targets are intentionally requested by customers or arise from broad crawl lists is unclear, but their presence raises further questions about the degree of curation applied to scraping tasks.

Client-side Footprint. Beyond network activity, Mellowtel's crawling leaves persistent traces on the participant's local disk. To characterize this, we ran the North Virginia vantage point continuously for five days with *Support With Mellowtel* [41] enabled and inspected the Chrome user data directory at the end of the run (each run uses a fresh Chrome profile; Section 4.1). We find that HTML of websites scraped by Mellowtel is not retained in Chrome's HTTP cache (`Default/Cache/`). However, V8 code cache (`Default/Code Cache/Js`), which stores compiled bytecode keyed by script URL to accelerate subsequent script loads [57], is populated with JavaScript drawn from the third-party pages fetched by crawl iframes. Figure 10 shows the per-run distributions of code cache contribution (orange line, bottom x-axis) and the number of URLs scraped (blue line, top x-axis). The median run contributed 235 KB of compiled JavaScript to the code cache, and runs that scraped more URLs generally contributed more cache, as expected since each additional crawl target brings its own first- and third-party scripts into V8's compilation path.

Ads Recommendations. To test whether the traffic injected by Mellowtel impacts user profiling and ad delivery, we run a fresh Chrome profile with *Support With Mellowtel* enabled for four days, recording both injected traffic and the ad recommendations received. We measure ad recommendations on 10 popular US news websites from SimilarWeb [56] that show visible ads on their landing pages. We use news websites because they commonly include display ads, making them suitable for repeated measurements. Every 30 minutes, we visit the 10 selected sites, locate ad placements using EasyList [17], and capture screenshots of those ad elements.

We use GPT-4o and Gemini 2.5 Pro to classify both observed ads and the HTML content of injected pages. In both cases, the models use the same 14-category taxonomy from Amazon Ads [4], following [34], so that ad and page classifications are directly comparable. We retain only labels on which both models agree. Across the experiment, we classify 18,198 injected pages and collect 6,870 ads. We remove 962 empty ads (14%) because each contained only a single pixel value throughout the image. We further exclude cases where the models disagree, which account for 19.2% of the remaining ads. We then compute Pearson correlations between injected-page categories and observed-ad categories at four offsets: same hour, hour + 1, hour + 2, and hour + 3. We observe only one case with a correlation of 0.38 between injected pages labeled "Hospitality" and ads labeled "Hospitality" during the same hour window; all other same-category pairs exhibit no correlation. This suggests no strong relationship between Mellowtel-injected traffic and users' ad recommendations.

7.2 Impact on Crawled Websites

We identify two ways in which Mellowtel places itself at odds with established crawling norms.

Stealth Crawling Behavior. RFC 9309 [31] establishes that automated agents should identify themselves via a distinct identifier

in the User-Agent header so that site operators can set crawler-specific policies. Mellowtel instead inherits the browser's default User-Agent string, rendering its requests indistinguishable from organic user traffic. Consequently, site operators have no reliable way to identify or restrict Mellowtel's traffic. First, any bot-specific directives a site has defined are silently circumvented, since Mellowtel's requests do not match any identifier a site operator could target. Second, operators wishing to restrict Mellowtel cannot do so without also blocking legitimate users of the same browser.

Robots.txt Compliance. We checked robots.txt compliance for all unique domains that Mellowtel attempted to scrape during our controlled experiments. Of the 18,311 domains that served a parseable policy, 393 (2.15%) explicitly disallowed scraping under the wildcard ("*") user agent. Figure 11 shows the distribution of scraping attempts across these 393 disallowed domains. The vast majority saw low attempt accounts, with 365 domains (93%) receiving ≤10 attempts, yet the distribution exhibits a long tail driven by a few heavily targeted sites: reddit.com (4,022), linkedin.com (974) and linktr.ee (397). Both Reddit and LinkedIn maintain restrictive robots.txt policies precisely because they are frequent targets of large-scale scraping, making these violations particularly conspicuous. Combined with the absence of a distinct User-Agent, this indicates that Mellowtel's crawling neither announces itself to site operators nor honors the access policies they publish.

8 Related Work

To the best of our knowledge, no previous scientific paper has investigated Mellowtel. Since our work sits at the intersection of browser extension security, extension monetization, and web scraping, we here cover the main works in these areas.

Browser Extension Security and Privacy. Browser extensions have been a significant security concern since their introduction. Early work by Barth et al. [7] analyzed Chrome's extension security architecture, highlighting risks of over-privileged extensions. Carlini et al. [8] found that many extensions request excessive permissions beyond their advertised functionality. For detection, Kapravelos et al. [27] used dynamic analysis to identify malicious behaviors such as credential theft and cookie stealing at runtime. Jagpal et al. [26] reported three years of operational experience with Google's WebEval, identifying ad injection and data exfiltration as the most common malicious behaviors. Pantelaios et al. [49] detected extensions that turn malicious post-vetting by analyzing JavaScript additions.

More recent works characterize the ecosystem at scale. Hsu et al. [22] mined historical Chrome Web Store (CWS) data and find that extensions containing malware, policy-violating, and vulnerable, collectively affect nearly 350 million users. Kim et al. [30] identified 59 vulnerabilities across 40 extensions enabling privilege escalation, including universal cross-site scripting. Moreno et al. [46] combined static, dynamic, and NLP-based code-similarity analyses to evaluate the CWS vetting pipeline, reporting that only 1% of CWS-flagged malware is also flagged by third-party antimalware engines. On privacy, Xie et al. [63] used dynamic taint tracking to uncover thousands of extensions with privacy-threatening data flows. While these studies focus on overtly malicious extensions (e.g., credential stealers, ad injectors), our work examines a third-party SDK that monetizes user bandwidth—a more subtle phenomenon sitting between legitimate functionality and potentially unwanted behavior.

Extension Monetization. Prior work has examined how browser extensions enable a range of monetization schemes, from ad manipulation to cryptomining and beyond. Thomas et al. [58] found 50,870 Chrome extensions that modified ads in users' browsing sessions, affecting 5% of Google users. With the rise of in-browser cryptomining, Rüth et al. [53] measured Monero mining scripts across the web, while Wang et al. [62] monitored seven extension marketplaces and identified 186 malicious cryptocurrency-themed extensions. Beyond content modification, Papadopoulos et al. [50] proposed a framework for stealthy browser-based abuse (e.g., cryptomining, DDoS attack) that relies on standard web APIs and Service Workers.

Mellowtel introduces a qualitatively different monetization model: bandwidth sharing. Its crawling activity produces no user-visible artifacts during browsing, operates within the browser's standard resource loading path via iframes, and generates network requests that are indistinguishable from organic page subresource fetches. This combination of minimal user-facing footprint and legitimate-looking network behavior distinguishes bandwidth-as-a-service from prior monetization abuse vectors and complicates both user awareness and automated detection.

Residential Proxy Services. Mellowtel's bandwidth-sharing model is also a specific instance of the broader residential proxy ecosystem, in which end-user devices are enrolled, often through bundled SDKs, to route third-party traffic through consumer IP space [9, 44]. Relatedly, Mi et al.'s measurement of mobile proxy SDKs [45] found four commercial providers paying app developers roughly $50 per month per 1,000 monthly active users, and that a majority of the Android apps bundling these SDKs were subsequently flagged as malicious or removed from Google Play. Yang et al. [65] found that the majority of studied Chinese RESIPs were involved in malicious traffic, including cryptojacking and botnets like Mozi. A parallel industry investigation of consumer-facing bandwidth-sharing apps such as Honeygain, PacketStream, and IPRoyal Pawns found their exit nodes carrying SQL injection probes, government-website crawling, and bulk scraping of personally identifiable information, despite each service's own terms describing legitimate use cases such as market research [59]. Khan et al. [28] instrumented eight such apps directly and found traffic consistent with dating-platform account fraud and connections fingerprinted to a phishing toolkit. Enforcement activity against this class of service has also intensified, from disruption of IPIDEA [21], to actions against the botnet-based operator 911 S5 [32, 60], the backdoored Android TV botnet BADBOX 2.0 [54], and the FBI's seizure of infrastructure tied to NetNut, a residential proxy service operated by a publicly traded company [33, 52]. Mellowtel differs from these prior subjects mainly in its distribution model, an opt-in SDK bundled into browser extensions with a disclosed revenue share, rather than a standalone paid VPN, free utility app, or covertly bundled proxy client. However, our findings, including limited threat-intelligence filtering of crawl targets and jurisdictionally insensitive task assignment, suggest

that this disclosed, opt-in framing does not fully insulate Mellowtel's model from the abuse patterns documented in this broader literature.

Web Scraping Compliance and AI Crawling. Automated crawling governance has drawn renewed attention as AI companies scrape the open web for training data. While RFC 9309 [31] requires crawlers to identify themselves via a distinct User-Agent, compliance is inconsistent. Kim et al. [29] tracked 130 self-declared bots and found that compliance declines as robots.txt directives become more restrictive. Liu et al. [38] reported significant gaps in awareness, agency, and efficacy of the protective tools available to site operators and content creators. We find that Mellowtel does not self-identify at all, inheriting the browser's default User-Agent, which leaves site operators with no signal to enforce crawler-specific policies.

9 Limitations

Several of our findings rest on inference rather than direct confirmation, since Mellowtel's backend, task assignment logic, and business arrangements with extension developers are not observable from an external vantage point. Consequently, our discussion in Section 6.1 that task assignment is driven by geolocation rather than network performance rests on the network-level control experiments (described in Appendices A.2 and A.3) rather than a confirmed mechanism, and our broader characterization of targeting patterns (Sections 6 and 7) is necessarily inferred from observed behavior across vantage points rather than read directly from the assignment logic itself.

A related limitation concerns the population of extensions we study. Mellowtel's business relationships with individual extension developers are not observable externally, and may plausibly affect task assignment or monetization activity (Section 3.4). Our controlled and in-the-wild deployment experiments rely on two extensions directly affiliated with Mellowtel, and hence we cannot confirm that independently developed, third-party integrations of the SDK receive the same treatment from Mellowtel's scheduler; absence of crawling activity we observe in 100 of 116 flagged extensions may in part reflect an incomplete or administratively dormant integration on the developer side rather than the absence of a business relationship altogether.

Finally, our reported count of 16 out of 116 extensions should be interpreted as an upper bound within the flagged set, not as a bound on Mellowtel's adoption across the broader extension ecosystem. Evaluating the recall of our detection methodology would require manually auditing our full corpus of more than 2,000 extensions; we did not conduct such an audit, and the resulting absence of a recall evaluation is a limitation of our methodology.

Our in-the-wild deployment included 34 participants across 25 countries. These participants provide geographically distributed vantage points from which to observe Mellowtel's activity, but they are not intended to constitute statistically representative samples of the populations of individual countries. The limited number of observations per country reduces our confidence in attributing differences among countries exclusively to geography, since individual browsing behavior and chance may also contribute. Nevertheless, the deployment provides evidence that the observed behavior occurs across diverse geographic locations, complementing our controlled network-level experiments. Increasing the number of participants and repeated observations within each country would strengthen confidence in the geographic patterns and enable more granular country-level comparisons. Because all participants were recruited from the same university population, future deployments spanning broader demographic and institutional populations would further improve the generalizability of these findings.

10 Conclusion

This paper presents the first empirical characterization of Mellowtel, a bandwidth-as-a-service SDK that monetizes browser extensions by routing web crawl traffic through opted-in users. We combined large-scale Chrome Web Store monitoring, controlled experiments across six AWS regions, and an in-the-wild deployment with 34 students spanning 25 countries to measure adoption, behavior, and security implications. We find that while hundreds of extensions bundle the Mellowtel SDK, active monetization is rare: only 16 of the 116 SDK-bundling extensions actually inject crawl iframes during a browsing session, making install counts a poor proxy for actual adoption. Among active nodes, crawling load is geographically skewed: US-based nodes receive up to 8.7× more crawl tasks than nodes in other regions, with peak transfers reaching 1.4 GB per 30-minute window. Beyond adoption patterns, we identify concrete security concerns. Crawl target lists include domains flagged by antivirus engines and credential phishing sites, suggesting limited filtering against threat intelligence feeds. More critically, Mellowtel's dynamic code loading mechanism allows remotely fetched JavaScript to execute inside the user's browser at runtime, bypassing Chrome Web Store review entirely. Bandwidth-as-a-service is a nascent avenue for extension monetization, but its current realization through stealth iframe injection in user browsers introduces security and accountability gaps that warrant closer scrutiny from researchers, browser vendors, and extension marketplaces.

References

[1] 2025. IdleForest — Plant trees for free while browsing. https://www.idleforest.com/

[2] Google 2026. *Chrome Web Store*. Google. https://chromewebstore.google.com/sitemap Accessed: 2026-03-29.

[3] Arslan Ali. 2025. Responding to Ars Technica (Condé Nast) and 'Mellow-Drama' Articles. https://www.mellowtel.com/blog/responding-to-ars-technica-and-mellow-drama-article. Blog post; accessed 2025-11-06.

[4] Amazon Ads. 2026. Industry Solutions. https://advertising.amazon.com/solutions/industries Accessed: 2026-04-28.

[5] Amazon Web Services. 2026. AWS Amplify. https://aws.amazon.com/amplify/. Accessed: 2026-04-11.

[6] Secure Annex. 2025. "Mellow Drama". https://secureannex.com/blog/mellow-drama/. Blog post; accessed 2025-11-06.

[7] Adam Barth, Adrienne Porter Felt, Prateek Saxena, and Aaron Boodman. 2010. Protecting Browsers from Extension Vulnerabilities. In *Proceedings of the 17th Network and Distributed System Security Symposium (NDSS)*. San Diego, CA. https://www.ndss-symposium.org/ndss2010/protecting-browsers-extension-vulnerabilities/

[8] Nicholas Carlini, Adrienne Porter Felt, and David Wagner. 2012. An Evaluation of the Google Chrome Extension Security Architecture. In *Proceedings of the 21st USENIX Security Symposium (USENIX Security 12)*. USENIX Association, Bellevue, WA, 97–111. https://www.usenix.org/conference/usenixsecurity12/technical-sessions/presentation/carlini

[9] Elisa Chiapponi, Marc Dacier, and Olivier Thonnard. 2023. Inside residential ip proxies: Lessons learned from large measurement campaigns. In *2023 IEEE*

European Symposium on Security and Privacy Workshops (EuroS&PW). IEEE, 501–512.

[10] Chromium Authors. 2017. CRX3 File Format Specification (crx3.proto). https://chromium.googlesource.com/chromium/src/+/refs/tags/131.0.6765.0/components/crx_file/crx3.proto. Accessed 2025-11-06.

[11] Chromium Project. 2025. CRX File Format. https://www.dre.vanderbilt.edu/~schmidt/android/android-4.0/external/chromium/chrome/common/extensions/docs/crx.html. Accessed 2025-11-06.

[12] Cloudflare, Inc. 2025. Domain Categories - Cloudflare One Traffic Policies. https://developers.cloudflare.com/cloudflare-one/traffic-policies/domain-categories/ Accessed: 2026-03-25.

[13] Cloudflare, Inc. 2026. Cloudflare API: Intel. https://developers.cloudflare.com/api/resources/intel/ Accessed: 2026-03-25.

[14] cthackers. 2023. ADM-ZIP: A Javascript implementation of zip for NodeJS. https://github.com/cthackers/adm-zip.

[15] Docker, Inc. 2013. Docker: Accelerated Container Application Development. https://www.docker.com. Accessed: April 15, 2026.

[16] Oliver Dunk. 2023. *Bringing Safety check to the chrome://extensions page.* Chrome for Developers. https://developer.chrome.com/blog/extension-safety-hub/

[17] EasyList. 2026. EasyList. https://easylist.to/ Accessed: 2026-04-28.

[18] FileFormat.com. 2025. CRX File Format Specification. https://docs.fileformat.com/misc/crx/. Accessed 2025-11-06.

[19] Dan Goodin. 2025. Browser Extensions Turn Nearly 1 Million Browsers Into Website-Scraping Bots. https://arstechnica.com/security/2025/07/browser-extensions-turn-nearly-1-million-browsers-into-website-scraping-bots/. *Ars Technica* (July 2025). Accessed 2025-11-06.

[20] Google Chrome Developers. 2026. Manifest V3. https://developer.chrome.com/docs/extensions/develop/migrate/what-is-mv3. Accessed: April 15, 2026.

[21] Google Threat Intelligence Group. 2026. No Place Like Home Network: Disrupting the World's Largest Residential Proxy Network. https://cloud.google.com/blog/topics/threat-intelligence/disrupting-largest-residential-proxy-network. Google Cloud Blog.

[22] Sheryl Hsu, Manda Tran, and Aurore Fass. 2024. What is in the chrome web store?. In *Proceedings of the 19th ACM Asia Conference on Computer and Communications Security*. 785–798.

[23] Hunkie Peanut. 2026. NES Emulator. https://chromewebstore.google.com/detail/nes-emulator/jfhpelccbpfcomeacbikkpjpoplckmgp Chrome extension.

[24] Mellowtel Inc. 2025. *mellowtel-js: Browser SDK for bandwidth-sharing.* https://github.com/mellowtel-inc/mellowtel-js Open-source (LGPL-3.0) browser SDK; retrieved 2025-11-06.

[25] Ipinfo. 2025. The trusted source for IP address data. https://ipinfo.io/developers

[26] Nav Jagpal, Eric Dingle, Jean-Philippe Gravel, Panayiotis Mavrommatis, Niels Provos, Moheeb Abu Rajab, and Kurt Thomas. 2015. Trends and Lessons from Three Years Fighting Malicious Extensions. In *Proceedings of the 24th USENIX Security Symposium (USENIX Security 15)*. USENIX Association, Washington, D.C., 579–593. https://www.usenix.org/conference/usenixsecurity15/technical-sessions/presentation/jagpal

[27] Alexandros Kapravelos, Chris Grier, Neha Chachra, Christopher Kruegel, Giovanni Vigna, and Vern Paxson. 2014. Hulk: Eliciting Malicious Behavior in Browser Extensions. In *Proceedings of the 23rd USENIX Security Symposium (USENIX Security 14)*. USENIX Association, San Diego, CA, 641–654. https://www.usenix.org/conference/usenixsecurity14/technical-sessions/presentation/kapravelos

[28] Etienne Khan, Elisa Chiapponi, Martijn Verkleij, Anna Sperotto, Roland Van Rijswijk-Deij, and Jeroen Van Der Ham-De Vos. 2024. A first look at user-installed residential proxies from a network operator's perspective. In *2024 20th International Conference on Network and Service Management (CNSM)*. IEEE, 1–9.

[29] Taein Kim, Karstan Bock, Claire Luo, Amanda Liswood, Chloe Poroslay, and Emily Wenger. 2025. Scrapers selectively respect robots. txt directives: evidence from a large-scale empirical study. In *Proceedings of the 2025 ACM Internet Measurement Conference*. 541–557.

[30] Young Min Kim and Byoungyoung Lee. 2023. Extending a hand to attackers: browser privilege escalation attacks via extensions. In *32nd usenix security symposium (usenix security 23)*. 7055–7071.

[31] Martijn Koster, Gary Illyes, Henner Zeller, and Lizzi Sassman. 2022. Robots Exclusion Protocol. RFC 9309. https://doi.org/10.17487/RFC9309

[32] Brian Krebs. 2024. Treasury Sanctions Creators of 911 S5 Proxy Botnet. https://krebsonsecurity.com/2024/05/treasury-sanctions-creators-of-911-s5-proxy-botnet/. Krebs on Security.

[33] Brian Krebs. 2026. FBI Seizes NetNut Proxy Platform, Popa Botnet. https://krebsonsecurity.com/2026/07/fbi-seizes-netnut-proxy-platform-popa-botnet/. Krebs on Security.

[34] Tu Le, Luca Baldesi, Athina Markopoulou, Carter T. Butts, and Zubair Shafiq. 2025. From Voice to Ads: Auditing Commercial Smart Speakers for Targeted Advertising based on Voice Characteristics. In *Proceedings of the 2025 ACM Internet Measurement Conference* (USA) *(IMC '25)*. Association for Computing Machinery, New York, NY, USA, 558–577. https://doi.org/10.1145/3730567.3764444

[35] Fangfan Li, Arian Akhavan Niaki, David Choffnes, Phillipa Gill, and Alan Mislove. 2019. A large-scale analysis of deployed traffic differentiation practices. In *Proceedings of the ACM Special Interest Group on Data Communication*. 130–144.

[36] Vector Guo Li, Matthew Dunn, Paul Pearce, Damon McCoy, Geoffrey M Voelker, and Stefan Savage. 2019. Reading the tea leaves: A comparative analysis of threat intelligence. In *28th USENIX security symposium (USENIX Security 19)*. 851–867.

[37] Linux man-pages project. 2024. tc(8) — Linux Traffic Control Manual Page. https://man7.org/linux/man-pages/man8/tc.8.html. Accessed: 2026-04-11.

[38] Enze Liu, Elisa Luo, Shawn Shan, Geoffrey M Voelker, Ben Y Zhao, and Stefan Savage. 2025. Somesite i used to crawl: Awareness, agency and efficacy in protecting content creators from ai crawlers. In *Proceedings of the 2025 ACM Internet Measurement Conference*. 78–99.

[39] Google LLC. 2025. *Chrome Web Store.* https://chromewebstore.google.com/ Online marketplace for Chrome browser extensions.

[40] Mellowtel. 2025. *Integrate Mellowtel in Your Browser Extension.* https://docs.mellowtel.com/browser-plugins/quickstart

[41] Mellowtel. 2025. *Support with Mellowtel.* https://chromewebstore.google.com/detail/support-with-mellowtel/jngbedjpioeongcicaomeideeompcbcc Chrome Web Store extension, updated September 11, 2025.

[42] Mellowtel. 2026. Mellowtel. https://www.mellowtel.com/ Section: "How much can I earn with Mellowtel?"; accessed 2026-03-25.

[43] Mellowtel. 2026. Mellowtel. https://www.mellowtel.com/ Section: "Will Mellowtel slow down my users' browsing?"; accessed 2026-08-15.

[44] Xianghang Mi, Xuan Feng, Xiaojing Liao, Baojun Liu, XiaoFeng Wang, Feng Qian, Zhou Li, Sumayah Alrwais, Limin Sun, and Ying Liu. 2019. Resident evil: Understanding residential ip proxy as a dark service. In *2019 IEEE symposium on security and privacy (SP)*. IEEE, 1185–1201.

[45] Xianghang Mi, Siyuan Tang, Zhengyi Li, Xiaojing Liao, Feng Qian, and XiaoFeng Wang. 2021. Your phone is my proxy: Detecting and understanding mobile proxy networks. In *Proceeding of ISOC Network and Distributed System Security Symposium (NDSS), 2021*.

[46] José Miguel Moreno, Narseo Vallina-Rodriguez, and Juan Tapiador. 2026. Did i vet you before? assessing the chrome web store vetting process through browser extension similarity. *IEEE Transactions on Services Computing* (2026).

[47] mukulgupta0014. 2026. LinkedIn ConnectEz. https://chromewebstore.google.com/detail/geehepcpfpchnndngcihpkdcjcafhmii

[48] OOKLA. 2024. Speedtest CLI, Internet connection measurement for developers. https://www.speedtest.net/apps/cli.

[49] Nikolaos Pantelaios, Nick Nikiforakis, and Alexandros Kapravelos. 2020. You've changed: Detecting malicious browser extensions through their update deltas. In *Proceedings of the 2020 ACM SIGSAC conference on computer and communications security*. 477–491.

[50] Panagiotis Papadopoulos, Panagiotis Ilia, Michalis Polychronakis, Evangelos P Markatos, Sotiris Ioannidis, and Giorgos Vasiliadis. 2018. Master of web puppets: Abusing web browsers for persistent and stealthy computation. *arXiv preprint arXiv:1810.00464* (2018).

[51] Peng Peng, Limin Yang, Linhai Song, and Gang Wang. 2019. Opening the blackbox of virustotal: Analyzing online phishing scan engines. In *Proceedings of the internet measurement conference*. 478–485.

[52] Kevin Poireault. 2026. FBI, Google Take Down NetNut Proxy Network Used by Cyber Threat Actors. https://www.infosecurity-magazine.com/news/fbi-google-take-down-netnut-proxy/. Infosecurity Magazine.

[53] Jan Rüth, Torsten Zimmermann, Konrad Wolsing, and Oliver Hohlfeld. 2018. Digging into browser-based crypto mining. In *Proceedings of the Internet Measurement Conference 2018*. 70–76.

[54] Satori Threat Intelligence and Research Team. 2025. Satori Threat Intelligence Disruption: BADBOX 2.0 Targets Consumer Devices with Multiple Fraud Schemes. https://www.humansecurity.com/learn/blog/satori-threat-intelligence-disruption-badbox-2-0/. HUMAN Security.

[55] Selenium Project. 2004. Selenium WebDriver. https://www.selenium.dev. Accessed: April 15, 2026.

[56] Similarweb Ltd. 2026. Similarweb. https://www.similarweb.com/ Accessed: 2026-04-28.

[57] Leszek Swirski. 2019. Code caching for JavaScript developers. https://v8.dev/blog/code-caching-for-devs. Accessed: 2026-04-14.

[58] Kurt Thomas, Elie Bursztein, Chris Grier, Grant Ho, Nav Jagpal, Alexandros Kapravelos, Damon McCoy, Antonio Nappa, Vern Paxson, Paul Pearce, Niels Provos, and Moheeb Abu Rajab. 2015. Ad Injection at Scale: Assessing Deceptive Advertisement Modifications. In *Proceedings of the 36th IEEE Symposium on Security and Privacy*. IEEE, San Jose, CA, 151–167.

[59] Trend Micro. 2023. Hijacking Your Bandwidth: How Proxyware Apps Open You Up to Risk. https://www.trendmicro.com/en_us/research/23/b/hijacking-your-bandwidth-how-proxyware-apps-open-you-up-to-risk.html. Trend Micro Research.

[60] U.S. Department of the Treasury. 2024. Treasury Sanctions a Cybercrime Network Associated with the 911 S5 Botnet. https://home.treasury.gov/news/press-releases/jy2375. Press Release.

[61] VirusTotal. n.d.. VirusTotal API v3 Overview and Reference Documentation. https://docs.virustotal.com/reference/overview. Describes the API for uploading files, scanning URLs, and retrieving analysis reports via HTTP JSON endpoints. Accessed: 2026-04-25.
[62] Kailong Wang, Yuxi Ling, Yanjun Zhang, Zhou Yu, Haoyu Wang, Guangdong Bai, Beng Chin Ooi, and Jin Song Dong. 2022. Characterizing cryptocurrency-themed malicious browser extensions. *Proceedings of the ACM on Measurement and Analysis of Computing Systems* 6, 3 (2022), 1–31.
[63] Qinge Xie, Manoj Vignesh Kasi Murali, Paul Pearce, and Frank Li. 2024. Arcanum: Detecting and evaluating the privacy risks of browser extensions on web pages and web content. In *33rd USENIX Security Symposium (USENIX Security 24)*. 4607–4624.
[64] X.Org Foundation. 1984. Xvfb: Virtual Framebuffer X Server. https://www.x.org/releases/X11R7.6/doc/man/man1/Xvfb.1.xhtml. Accessed: April 15, 2026.
[65] Mingshuo Yang, Yunnan Yu, Xianghang Mi, Shujun Tang, Shanqing Guo, Yilin Li, Xiaofeng Zheng, and Haixin Duan. 2022. An extensive study of residential proxies in China. In *Proceedings of the 2022 ACM SIGSAC conference on computer and communications security*. 3049–3062.

A Appendix

A.1 Ethics

Our study obtained IRB approval (HRPP-2025-292), and two authors completed CITI research-ethics training. All participants provided informed consent and were clearly informed that the extension monitors network activity. Importantly, our extension only logs requests to domains explicitly associated with Mellowtel iframes – we do *not* monitor or log general browsing activity, search queries, or visits to unrelated websites. Collected data was stored on secure servers. Participants could withdraw at any time by uninstalling the extension. While our extension requires broad network permissions similar to those used by Mellowtel itself, we took care to minimize privacy risks through domain filtering and by limiting data collection to the technical metadata necessary for understanding Mellowtel's behavior.

Because participants installed Support With Mellowtel [41] for the duration of the study, they were also exposed to the dynamic code-loading behavior described in Section 7.1, under which Mellowtel's cereal iframe can fetch and execute parser code from a remote endpoint at runtime. Our consent materials disclosed this behavior explicitly, describing that the extension can load and run code from Mellowtel's servers in addition to monitoring network requests, and participants were informed that this risk is inherent to any use of a Mellowtel-enabled extension, independent of study participation. We did not observe the fetched code exceed its declared parsing role during the deployment (Section 7.1), but we could not guarantee this in advance; our primary mitigations were transparency and voluntary participation, with participants free to uninstall the extension at any time or opt-out of the study. Our own monitoring extension does not itself execute remote code; it only observes network requests and DOM mutations.

A.2 Network Conditions at the Vantage Points

To isolate the contribution of network conditions to the Active Rate disparity reported in Section 6.1, we measured baseline network performance at each vantage point and applied synthetic network configurations to a random subset of experiments in North Virginia. Before every experiment run we recorded available bandwidth via Ookla Speedtest [48] and TCP RTT to `request.mellow.tel`. All six vantage points reported downlink and uplink capacity exceeding 1 Gbps, well above what individual crawl sessions consume. However,


```
{
    ...
    "code": "HAUL35",
    "title": "Spring Haul Sale– $35 Off w/ Minimum Spend",
    "description": "Take an extra $35 off $150 or more orders...",
    "merchant": {
        "name": "Scrubs and Beyond"
    },
    "redemptionUrl":
          "https://click.couponfollow.com/click?pwk=df1234...",
    "timestamp": "2025-12-17T21:31:29.805Z"
    ...
}
```


Figure 12: Coupon metadata returned by Cereal iFrame upon successfully parsing a target site's HTML

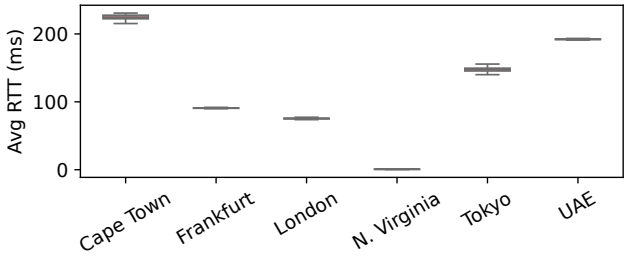


Figure 13: TCP ping latency to Mellowtel's backend per vantage point.

Figure 13 shows that RTT to Mellowtel's server varied substantially: (on average) under 1 ms in North Virginia, 75 ms in London, 90 ms in Frankfurt, 147 ms in Tokyo, 192 ms in UAE, and 224 ms in Cape Town. The sub-millisecond latency in North Virginia stems from Mellowtel's backend being co-located in the same AWS region (us-east-1).

To test whether this latency differential alone could account for the observed disparity, we injected synthetic network impairments on a random subset of North Virginia runs. Specifically, we randomly assigned each North Virginia experiment run to one of several network configurations applied via Linux `tc` [37] before the experiment run began. We tested three settings: increased RTT for outgoing packets (100 ms or 200 ms, approximating the London and Tokyo baselines), bandwidth throttling to 2 Mbps (representative of ISP enforced per client caps), and combinations of the two. Figure 14 shows the Active Rate under each configuration alongside the baseline. Increased latency alone produces no meaningful change; the 200 ms (79%) and 100 ms (74.8%) conditions generally remain consistent with the baseline (74.1%). Bandwidth throttling in isolation yields the only visible drop (62.5%), yet the effect disappears once latency is layered on top (73.8%, 80.7%). We do not have a clean explanation for this non-monotonicity, and given the smaller sample sizes in the throttled conditions we treat the bandwidth-only result as suggestive rather than conclusive. Crucially, no single configuration approaches the 2.5x Active Rate gap observed across vantage points, reinforcing that the disparity is primarily driven by Mellowtel's geolocation-based scheduling rather than network performance at the client.

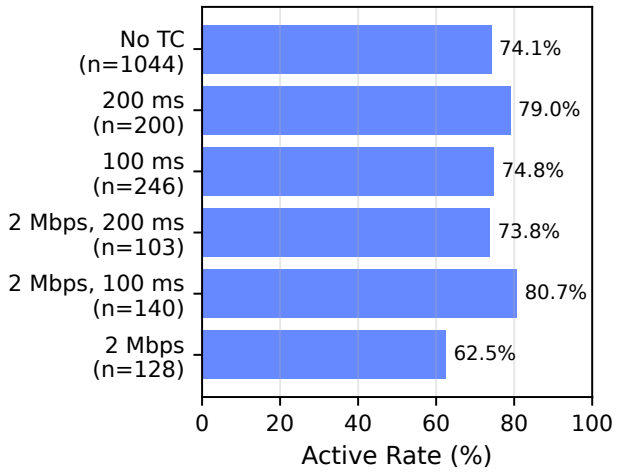


Figure 14: Mellowtel's active rate in North Virginia under different latency and bandwidth configurations. Sample sizes indicated at each y-axis label.

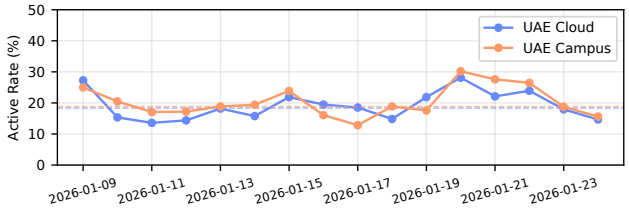


Figure 15: Daily active rate of the AWS UAE vantage point (me-central-1) and a campus Ethernet host in Abu Dhabi, January 9–24, 2026. Dashed lines denote the per deployment mean (20.1% AWS, 19.3% campus).

A.3 Cloud vs Campus Validation

The controlled experiments in Section 6.1 were deployed on AWS EC2 instances. A potential confound is that Mellowtel's scheduler may assign tasks differently based on the IP space of the client, for example by deprioritizing known cloud address ranges in favor of residential or enterprise traffic. To test this, we ran the same experimental workflow on a Linux desktop connected to campus Ethernet in Abu Dhabi, the same city that hosts the AWS me-central-1 region used by our UAE vantage point. The campus deployment ran from January 9 to January 24, 2026, overlapping fully with the AWS UAE instance over the same period. Figure 15 shows the daily active rate for both deployments. The two series track each other closely, with no systematic offset between the cloud and campus hosts; the gaps on individual days fall well within the day-to-day variation observed at a single vantage point. Averaged over the 15 days, the AWS UAE instance achieved an active rate of 20.1% and the campus host 19.3%. The session type distribution was also consistent across environments (92.7% single, 7.3% batch on AWS; 92.4% single, 7.6% batch on campus). We conclude that within a single metropolitan area Mellowtel's scheduler does not meaningfully distinguish AWS address space from campus traffic.

Content-Type	Count	Median	IQR
css	294	1.0	0.0
font	37	1.0	0.0
html	66	1.002	0.04
image	290	1.0	0.0
javascript	750	1.0	0.0
json	33	1.0	0.0
other	17	1.0	0.0
text	13	1.0	0.336

Table 3: For each `Content-Type`, the number of samples and the median and interquartile range of `refetched_size / logged_Content-Length`. A ratio of 1.0 indicates an exact match between the re-fetched size and the original.

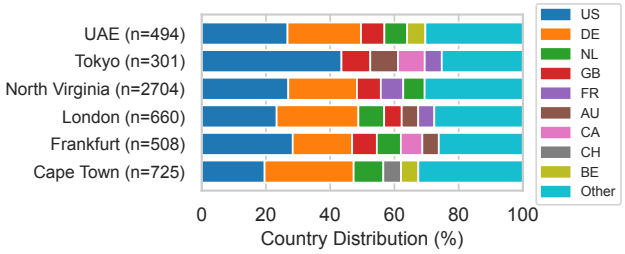


Figure 16: Country distribution of domains scraped at each vantage point, restricted to those attributable via ccTLD or non-CDN hosting IP. *n* denotes the number of attributable domains per vantage point.

A.4 Content-Length Recovery via Re-fetching

For the 26.2% of 2xx responses in our controlled experiments that lacked a Content-Length header, we re-fetched each using `curl` with HTTP/2 enabled and headers matching Chrome's original request. Responses were recorded with automatic decompression disabled so that the reported size reflects the compressed on-wire bytes, matching the semantics of Content-Length in the logged responses. To characterize the error introduced by content drift between the original crawl (November 2025 to February 2026) and the re-fetch, we applied the same procedure to a random sample of 1500 responses for which Content-Length was logged and computed the ratio `refetched_size` / `logged_content_length`. Table 3 summarizes the results per Content-Type bucket. The median ratio is close to 1.0 for static subresources (font, CSS, images) and drifts slightly for HTML documents, consistent with dynamic page content. We report per-session transfer using re-fetched sizes for missing entries and note that 15.1% of URLs could not be recovered (404, DNS failure, or timeout); these are excluded from the aggregate.

A.5 Country Attribution of Crawled Domains

To infer the operating region of a domain, we combine two heuristics. First, we attribute a domain to a country when its top-level domain matches a two-letter ISO 3166-1 country code (*e.g.,* .de, .uk, .jp), yielding 5,950 domains (26.3%). Second, for the remaining domains we perform DNS A-record lookups and map each resolved

AWS Region	Count	Unscraped Domains
Cape Town	7	brex.com, zapier.com, nucor.com, support.bigcommerce.com, legiscan.com, forestsandrangelands.gov, tripadvisor.com
Frankfurt	2	gartner.com, quickbooks.intuit.com
London	6	au.linkedin.com, experian.com, viator.com, onlinevapeshop.co.uk, superagi.com, imdb.com
Tokyo	7	forbes.com, wiz.io, lifespringhomecare.com, issuu.com, daft.ie, ovoko.fr, help.skool.com
North Virginia	35	highestplain.com, slb.com, ycds.org, clery.arizona.edu, kes.ns.ca, news.ycombinator.com, news.cornell.edu, golfergeeks.com, weaverleathersupply.com, hunkemoller.com, pilgrimacademy.org, gallery.midpac.edu, gprep.com, eventbrite.com, tech.cornell.edu, myhoneyhome.pl, redbubble.com, help.redbubble.com, war.gov, vapersonline.co.uk, dcard.cc, docs.google.com, stksteakhouse.com, vapekingz.net, telekom.com, doushesh.com, rural.gov, stopbullying.gov, janaushadhivitran.com, police.uci.edu, docs.arangodb.com, nca-i.com, stpauljackson.com, fau.edu, mvcc.edu
UAE	9	threadresearch.com, airops.com, instagram.com, researchgate.net, slashdot.org, upwork.com, aroundtownmovers.com, quora.com, sarenza.com

Table 4: Domains scraped at five of the six vantage points but not at the indicated region; North Virginia accounts for the majority of such cases.

```
var ni = "https://<hash>.lambda-url.us-east-1.on.aws/";

// Fetches parser code from AWS Lambda
fetch(`${ni}?parser_id=${parserId}`, { method: "GET" })

// Dynamically execute remotely fetched JS source
n = new Function("htmlString", "originalUrl", `
    ${fetchedCode}
    return parse(htmlString, originalUrl);
`);
```

Figure 17: Dynamic parser loading via AWS Lambda.

address to a country and ASN via IPinfo [25]. A substantial share of these, 11,667 domains (51.6%), resolve to IP space operated by global CDN providers (*e.g.,* Cloudflare, Akamai). We exclude these from the geolocation analysis. The remaining 4,386 domains (19.4%) are geolocated by their hosting IP, and 595 (2.6%) failed to resolve. We note that this process skews the sample toward smaller, regionally hosted sites by excluding domains that sit behind global CDNs; therefore the shares should be read as a conservative view of geographic footprint.

Figure 16 reports the country attributions per vantage point. Two observations stand out. First, no vantage point exhibits a pronounced bias toward its own region: Frankfurt's share of DE hosted domains (16.8%) is lower than its share of US hosted ones (25.8%), London's GB share is only 5.1%, and JP and AE do not appear in the top ten at Tokyo or UAE. Second, the dominant countries are largely consistent across regions, with the US, Germany, and the Netherlands together accounting for over 50% of attributable domains at five vantage points. Tokyo is the lone exception. Its top three countries are US (38.3%), GB (7.9%), and AU (7.6%), and the combined share of DE and NL falls to 7.0%, compared to 24-35% elsewhere. We caution against over interpreting this shift: Tokyo also has the highest fraction of CDN hosted domains (63.5%) and therefore the smallest geolocatable sample (342 domains, versus 2,885 at North Virginia), so the resolved subset is both noisier and potentially biased by which categories of sites are less likely to sit behind a CDN. Whether the apparent skew reflects genuine scheduler behavior or a sampling artifact of the CDN exclusion is not something we can disentangle without visibility into Mellowtel's task assignment logic.

A.6 Region-specific Gaps in Recurring Crawl Targets

Section 6.3 notes that 35 of the 66 domains scraped at five of six vantage points are missing specifically from North Virginia, despite North Virginia exhibiting the highest overall crawling activity. Table 4 lists the unscraped domain set for each region. The other five regions exhibit small gaps (between 2 and 9 domains), while North Virginia stands out both in count and composition.