Using Gaming Footage as a Source of Internet Latency Information

Catalina Alvarez
EPFL
Switzerland
catalina.alvarezinostroza@epfl.ch

Katerina Argyraki
EPFL
Switzerland
katerina.argyraki@epfl.ch

ABSTRACT

Keeping track of Internet latency is a classic measurement problem. Open measurement platforms like RIPE Atlas are a great solution, but they also face challenges: preventing network overload that may result from uncontrolled active measurements, and maintaining the involved devices, which are typically contributed by volunteers and non-profit organizations, and tend to lag behind the state of the art in terms of features and performance. We explore gaming footage as a new source of real-time, publicly available, passive latency measurements, which have the potential to complement open measurement platforms. We show that it is feasible to mine this source of information by presenting Tero, a system that continuously downloads gaming footage from the Twitch streaming platform, extracts latency measurements from it, and converts them to latency distributions per geographical location. Our data-sets and source code are publicly available at https://nal-epfl.github.io/tero-project.

CCS CONCEPTS

• **Networks** → **Network measurement**; *Social media networks.*

KEYWORDS

Internet Performance; Passive Measurement; Social Media

ACM Reference Format:
Catalina Alvarez and Katerina Argyraki. 2023. Using Gaming Footage as a Source of Internet Latency Information. In *Proceedings of the 2023 ACM Internet Measurement Conference (IMC '23), October 24–26, 2023, Montreal, QC, Canada.* ACM, New York, NY, USA, 21 pages. https://doi.org/10.1145/3618257.3624816

1 INTRODUCTION

Keeping track of the Internet latency[1] experienced by end-users around the world is a classic network-measurement problem [17, 19, 25]. Significant progress was made with the creation of open measurement platforms like RIPE Atlas [3] which provide geographically distributed, publicly accessible devices that can carry out active measurements. One challenge faced by such platforms is preventing network overload that may result from uncontrolled active measurements, typically with strict quota on the number and rate of measurements initiated by a single user [20]. Another challenge is the overhead of maintaining the involved devices, which are typically contributed by volunteers and non-profit organizations, and tend to lag behind the state of the art in terms of features and performance [52]. When a company needs to track Internet latency, it avoids these challenges by implementing proprietary passive measurements, which leverage the company's existing infrastructure that needs to be maintained anyway as part of its business [17, 48]. This work is part of a quest for measurements that combine features from both of these worlds: they are passive and do not require the maintenance of special measurement infrastructure; but they are also publicly available, hence can be used to complement existing open measurement platforms and contribute to the transparency of Internet performance.

We explore gaming footage as a source of Internet latency information. We think that this is a promising approach for three reasons: (1) Players care about latency [38], so they have an incentive to use latency-optimized devices that contribute minimal overhead to end-to-end latency. (2) Many online video games display on screen the latency between play-station and game server. (3) Live game streaming is on the rise, yielding plenty of publicly available gaming footage that contains latency numbers.

We have built a system, called Tero, which periodically downloads such footage, extracts the latency numbers, and produces an almost-real-time analysis of Internet latency per geographical location. Our data-sets and source code are publicly available at https://nal-epfl.github.io/tero-project/. The current prototype downloads footage from the Twitch[2] streaming platform, chosen for its ubiquity [42], easy-to-access data, and focus on live content. Following Twitch's Terms of Service[3], we neither access video streams nor crawl streamer profiles (or any other web pages); we obtain gaming footage through Twitch's Content Distribution Network (CDN)[4] and related metadata through its Developer Application Programming Interface (API)[5]. We deployed the first version of Tero on May 25th, 2021, and since then we have collected latency measurements from 150 thousand users streaming 9 different video games.

What was challenging about designing Tero?

(1) Associating footage with geographic location: Gaming footage does not include location, and Twitch makes it hard for a streamer to disclose their location in their streaming account. To overcome this, Tero tries to identify each streamer's social-media profile, by

[1] The round-trip time (RTT) between an end-user's device and a server that responds to the end-user's requests.

IMC '23, October 24–26, 2023, Montreal, QC, Canada

ACM ISBN 979-8-4007-0382-9/23/10…$15.00
https://doi.org/10.1145/3618257.3624816

[2] https://www.twitch.tv/
[3] https://www.twitch.tv/p/en/legal/terms-of-service/
[4] https://dev.twitch.tv/docs/api/reference/#get-streams
[5] https://dev.twitch.tv/docs/api/

leveraging the voluntary connections that a streamer often draws between their profile and streaming account; it then uses natural language processing (NLP) to extract location information from the social-media profile; and maps each piece of footage to a broad geographical location (city, region, or country).

(2) Interpreting the latency numbers: When a player experiences a latency increase, this may be due to a network or end-point problem, or because the player changed their location (potentially without updating their social-media profile), or they started playing on a different server (to interact with a different group of players). To interpret a latency increase, Tero relies on player habits, the characteristics of the latency increase itself, and whether it overlaps with increases experienced by other streamers mapped to the same geographical location. For example, League-of-Legends players from the UK tend to play either on the Western European (EUW) or on the North American (NA) server; so, if a streamer that has been mapped to London transitions from a period of stable latency to another period of stable-but-higher latency *and* the latency increase is consistent with switching from the EUW to NA server, then Tero interprets this latency increase is a likely server change.

(3) To a large extent, Tero is a carefully designed combination of existing techniques and tools. Yet several problems that we expected to be trivial turned out to be challenging in our context. For example, we expected the extraction of latency numbers from gaming footage to be a trivial application of Optical Character Recognition (OCR). However, games typically display latency in low resolution (75 dpi on average), and we found state-of-the-art OCR to experience word-level error rates as high as 23.97% (consistent with [16]). In the end, to achieve reasonable accuracy, Tero needs to complement OCR with knowledge of each game's user interface (e.g., it is unlikely that latency is displayed in the middle of the screen) and conservative assumptions about the latency itself (e.g., it is unlikely that latency dropped from 110ms to 10ms; it is more likely that a drop-down menu covered the most significant digit).

The rest of the paper is organized as follows: We start with background and limitations (§2). Then we describe Tero (§3), evaluate its accuracy (§4), and summarize our findings (§5). Finally, we discuss a future direction (§6) and related work (§8), and we conclude (§9).

Ethical considerations. Our work does raise ethical issues, because it uses data posted on streaming and social-media platforms. We went through the formal evaluation process of our institution's Ethical Review Board and obtained their approval. We discuss the relevant privacy risks and legal considerations in §7.

2 BACKGROUND AND LIMITATIONS

2.1 Video Games and Streaming

We now summarize the elements of online video games and streaming that are relevant to this work.

Latency sensitivity. Latency can significantly affect gaming experience: In virtual reality (VR) games, the minimum latency increase that players can perceive is in the range of 8 to 15ms [32]. In traditional non-VR First Person Shooter games, players display similar latency sensitivity, while the minimum latency increase that affects performance is 25ms, with more skilled players suffering more [30].

Latency-driven design. Due to their latency sensitivity, games are typically designed to minimize latency overhead: they follow the client-server model, with the client (the play-station) doing most of the computation and rendering, sending only small updates to the server. Given that players care about their latency [38], many games now display on-screen measurements of the latency between client and server; this latency is measured at the server (in a proprietary manner, presumably at the application layer), and it is meant to capture the round-trip time (RTT) between client and server. A game provider typically deploys several servers around the world, and players may interact only if they play on the same server. Moreover, a game provider typically divides the world in game-regions, and it assigns players from each game-region to the "closest" server (the one that is expected to minimize their latency). Hence, two players playing the same game from the same game-region typically play on the same server. However, this is not guaranteed to be the case, because a player may occasionally join a server other than the one assigned to their game-region (in order to interact with a particular player crowd, even if the resulting latency is sub-optimal).

Twitch thumbnails. Many players are also "streamers," i.e., they broadcast video of themselves playing online video games, typically in real-time. The most popular streaming platform is Twitch, with an average of 1.2 million active daily users, covering more than 70% of the video-game streaming market [42]. Twitch makes gaming footage publicly available and download-able from their CDN in the form of thumbnails, i.e., images extracted from the videos. For each streaming session, a new thumbnail is generated **every 5 minutes** (with variations that can reach up to a minute). These thumbnails, obtained through Twitch's CDN, are the source of latency information that we study in this paper.

2.2 Limitations

Our work is subject to the following limitations—all fundamental, except for the last one, which is due to a Twitch-specific policy:

Lack of latency definition. We do not know how each game server computes the latency number that it displays to a gamer, e.g., whether it is measured at the application or network layer, or whether it is a sample RTT or a moving average. We can make educated guesses (by reverse-engineering the measurements we observe when we play), but we have no ground truth.

Lack of control over measurement points. All measurements are performed at game servers, whose location and numbers are outside our control.

Lack of topology information. At best, we geolocate each streamer at the granularity of a city. Hence, we do not know the networks or technologies underlying each measurement.

Susceptibility to false descriptions. To geolocate a streamer, we rely on the streamer's own description of their location; if that is false, then our conclusion will also be incorrect.

Streamer bias. Tero's latency information comes from streamers, which makes it biased in two ways: it comes from geographic locations where streaming is popular; it comes from end-users with a stronger incentive for good Internet connectivity than the average player, not to mention the average end-user. Hence, we cannot use

Tero to reason about Internet latency in general; we can only use it to reason about lower bounds.

Sparse per-streamer data. Even though streamers publicly broadcast their game, Twitch forbids viewers from storing the videos (they make available only one thumbnail per streamer every 5 minutes). In principle, we can overcome this, e.g., by extracting the latency information directly from the broadcasted stream (as opposed to the thumbnails). We have not taken this step yet, because we wanted to be absolutely certain that we violated neither the letter nor the spirit of Twitch's Terms of Service. However, we are optimistic that we will find a legal way to obtain denser per-streamer data. For the moment, even though we have sparse data per streamer, we compensate by combining data from multiple streamers (playing from the same geographical location).

3 SYSTEM DESIGN

Tero consists of a *download* module (App. §A) that continuously downloads thumbnails from Twitch's CDN; a *location* module (§3.1) that groups streamers per geographical location; an *image-processing* module (§3.2) that extracts latency measurements from thumbnails; and a *data-analysis* module (§3.3) that cleans the data and computes latency statistics. We describe only the aspects of these modules that we think are non-obvious, and we provide additional information needed for reproducing our results in the Appendix. We describe our implementation in App. §B and list the games currently processed in App. §C.

3.1 Location

The location module takes as input a streamer's profile (obtained through Twitch's Developer API), and it outputs the streamer's location in the form of a {*city*, *region*, *country*} tuple. Both the input and output consist solely of publicly available information.

For a small fraction (0.97%) of the streamers, Tero extracts their (likely) location directly from their profile: A Twitch profile includes a field called "description," which is unstructured text that the streamer uses to introduce themselves to their audience. Some streamers embed location information in their description, e.g., "Join us in Detroit!"

For the rest, Tero tries to find a social profile, by leveraging the voluntary connections that a streamer often draws between their social profile and streamer account: (1) A streamer typically adds to their social profile an explicit link to their streamer account, as part of advertising. (2) Some streamers use the same username across social-media and streaming platforms. The latter may be just human habit, but there is also an economic incentive: a streamer's username is part of their brand, so it makes sense to create a social presence with that username. Tero leverages these connections as follows: (1) Given a streamer account A, it looks for a social profile with the same username as A. (2) If it finds such a profile P, it checks whether P includes an explicit link to A; if yes, it associates P and A. Our current prototype considers Twitter and Steam[6] social profiles, because, to the best of our knowledge, they are the most popular ones with streamers.

Given a Twitch description or a social profile, Tero tries to extract a location tuple, using a combination of the most popular publicly available NLP tools (Table 3). CLIFF[13], Xponents[57], and Mordecai[18] are *geocoding* tools: they take as input unstructured text, extract from it the parts that describe location, and map each location description to an actual location (this last step is called *geoparsing*). The remaining tools perform only geoparsing, but have higher geoparsing accuracy than the geocoding tools. We experimentally found that Tero achieves higher accuracy by combining all these tools than using any subset of them. In particular, Tero accepts location L when (1) the output of a tool passes a conservative filter (see App. D.1), (2) at least two of the underlying tools output L, or (3) at least one tool outputs L and at least one more tool outputs a more general location that is compatible with L (e.g. one tool outputs "Los Angeles, USA" and another outputs "California, USA"). We describe location extraction in more detail in App. §D. We decided against state-of-the-art solutions like [1], because they are specifically trained for English and do not typically release their models (meaning that we would have to train them from scratch).

3.1.1 Streamers with multiple locations. Occasionally, Tero extracts different location tuples for the same streamer (at different points in time). At first, we thought that this was an error; however, in all such cases that have occurred so far, upon manual inspection, it turned out that the streamer was indeed advertising a new location (presumably because they moved), and a location change was consistent with their latency measurements. Hence, Tero may maintain multiple locations per streamer, in which case it treats each {streamer, location} tuple as a distinct end-point during data analysis.

3.1.2 Errors. The location module may extract an incorrect location for a streamer for two reasons: the underlying geocoding/geoparsing tools make a mistake, or the streamer themselves advertise an incorrect location. We estimate the error rate due to the former to $1.46 \pm 0.07\%$ (§4.2), but we have no way of estimating the latter. However, the latency measurements of streamers playing from the same location tend to fall into clusters (§3.3.3). Hence, one approach to reducing these errors would be to reject latency measurements that fall outside the clusters for the corresponding location. We do not take this step, but the users of our data-set have all the information needed to take it.

3.2 Image Processing

The image-processing module takes as input a thumbnail (a streamer's screenshot), and it outputs the latency displayed in the thumbnail. For a complete description, please see App. §E.

Our first attempt was to use Optical Character Recognition (OCR) out of the box. An OCR engine takes as input an image and identifies all the alphanumeric characters in it. So, we simply passed the thumbnail as input to an OCR engine and post-processed its output to extract numbers. We tried the three most popular publicly available OCR engines: Tesseract[7], EasyOCR[8], and PaddleOCR[9].

This yielded insufficient accuracy: First, games display latency in low resolution (75 dpi on average), and this causes the OCR engines to make mistakes, e.g., mistake 8 for "B" or "S", 0 for "O", 4 for

[6] https://store.steampowered.com/

[7] https://github.com/tesseract-ocr/tesseract
[8] https://github.com/JaidedAI/EasyOCR
[9] https://github.com/PaddlePaddle/PaddleOCR

"A". Second, it occasionally happens that the displayed latency is partially or fully covered, e.g., by an open menu, in which case OCR may work perfectly and still produce output that is incorrect in our context. Because of these two issues, the most accurate OCR engine extracted a correct latency number from 86.32% of the thumbnails we tested it with, while the three engines were complementary (they made mistakes on partially overlapping sets of thumbnails).

To improve accuracy, Tero leverages knowledge of each game's user interface as follows: (1) It considers the area where each game typically displays latency (e.g., top right corner) and crops around it to create an image that is significantly smaller than the original thumbnail and is most likely to include the displayed latency. (2) It passes this smaller image as input to all three OCR engines. (3) It post-processes the output of each OCR engine, using game-specific heuristics, to extract the latency. For example, if multiple characters are extracted from a region where we expected a single latency digit, Tero tries to determine which one looks most like a latency digit, and which one(s) look most like other elements that the specific game is likely to display at that region (e.g., "ms" right after the latency digits, or "ping" or "latency" right before them). (4) It compares the (post-processed) output of the three OCR engines; if at least two of them agree, their output is accepted as the "primary" latency for the corresponding thumbnail; if exactly two engines agree, Tero keeps the third engine's output as an "alternative." In the data-analysis phase, if the primary latency is incompatible with the rest of the data, and the alternative is not, we use the alternative (§3.3.2).

3.2.1 Errors. The image-processing module may extract incorrect latency in two ways: through *digit confusion* (e.g., "42ms" is read as "12ms") and through *digit drop* (e.g., "41ms" is read as "1ms"). We evaluate it in detail in §4.2, but, in summary, it produces incorrect latency for 3.7% of the thumbnails, and 68.42% of these errors are digit drops.

3.3 Data Analysis

The data-analysis module continuously reads the output of the location and image-processing modules and performs the following processing: (1) It organizes its input into "streams" (§3.3.1). (2) It identifies and corrects or discards "anomalies," i.e., transient changes in a streamer's latency measurements that cannot be explained as location or server changes, hence need to be treated as noise (§3.3.2). (3) It computes a latency distribution for each {*location*, *game*} tuple, which is meant to capture the latency experienced when playing *game* from *location*, on the primary server for that location (§3.3.3).

We initially sought to extract useful information from anomalies, e.g., use them to detect network problems in real-time. We still think that that is a promising direction, however, it requires more information about the underlying network topology than Tero currently has. Hence, for the moment, Tero either corrects or discards anomalies.

Table 1 states Tero's configurable parameters, and we show how their values affect the results in App. §I.

3.3.1 Streams. A *stream* is a sequence of {timestamp, latency} tuples that represent the latency experienced by one streamer, playing one game. The first (resp. last) timestamp of a stream corresponds roughly to the moment when the streamer comes online (resp. goes offline). Two consecutive timestamps are at least 5 min apart (because Twitch posts thumbnails from a streamer's video every 5 min); they may be more than 5 min apart, when the streamer is online but not streaming (e.g., taking a break or interacting with their audience). We assume that a streamer may switch servers but not location mid-stream; they may change location in between streams.

StableLen	Minimum time one must play on the same server before switching
LatGap	Perceivable latency difference threshold
MaxSpikes	Maximum proportion of spikes/points allowed

Table 1: Tero's configurable parameters.

Each stream is further divided into *same-QoE segments* (or simply *segments*): subsets of the stream, whose latency measurements are within *LatGap*—a configurable parameter, equal to the minimum latency difference that is perceivable by human users. In this work, we use *LatGap* = 15ms (unless otherwise noted), which is the upper bound of the latency difference that is currently perceivable by players [32]. A single, long-lasting segment indicates a streamer playing from one location, on one server, and without any technical problem that significantly affected latency.

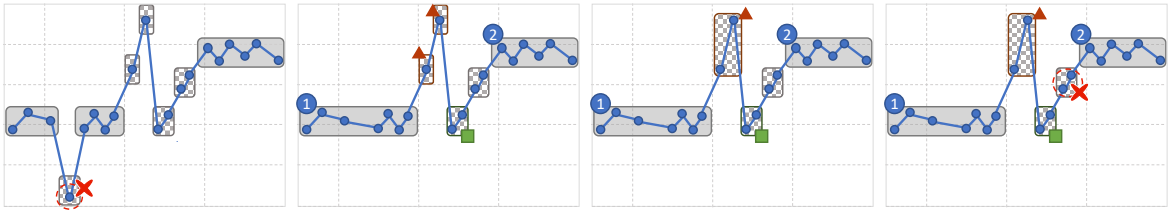
Each segment S is classified as *stable* if S includes at least *StableLen* points, and as *unstable* otherwise. Fig. 1 illustrates stable (solid gray boxes) and unstable (checkered boxes) segments. *StableLen* is a configurable, game-dependent parameter, equal to the minimum amount of time a player must play on a single server before switching to another one.

A stream yields a first picture of the streamer's experience: fewer segments indicate fewer changes in perceivable latency; more stable segments indicate longer play without perceivable latency changes. If a streamer has experienced *only* unstable segments, Tero discards all their data, because this is likely a streamer with a problematic play-station and/or Internet connection.

3.3.2 Anomalies. In summary, to detect anomalies, Tero stitches together all the same-QoE segments experienced by one streamer playing one game; and it looks for *unstable* segments with significantly higher or lower latency measurements than their *stable* neighbors.

We did try to use state-of-the-art anomaly detection and changepoint detection out of the box, but they did not work well on our data. The former did not allow us to reason about "explainable" anomalies, like server and location changes (App. §J). We also tried the PELT changepoint detection algorithm [26], but it did not complete in useful time. We think the reason is that incorrect latency measurements resulting from image-processing errors create weird probability distributions that throw the algorithm off. However, we should note that Tero's anomaly-detection algorithm is a simple form of changepoint detection with extra steps for detecting and fixing image-processing errors.

Tero distinguishes between *glitches* (latency decreases) and *spikes* (latency increases), and it uses slightly different detection algorithms for the two. The reason is that they typically occur for different reasons and have different features: Glitches are typically the result of image-processing error and, in particular, digit drop.



(a) Glitch detection. (b) Spike detection. (c) Spike merging. (d) Cleanup.

Figure 1: Stream division into stable (solid) and unstable (checkered) segments. Glitch and spike detection.

For example, a streamer's true displayed latency is "45ms," but the "4" is hidden by an open menu, causing OCR to read "5ms". Hence, a glitch typically takes the form of a sharp latency drop followed by a sharp increase. In contrast, spikes typically consist of correctly extracted latency measurements that are the result of the streamer being affected by some technical problem, e.g., network congestion or server overload. Hence, a spike typically takes the form of a relatively smoother increase followed by a similar decrease. Of course, it is possible that a spike is also the result of image-processing error, e.g., "15ms" is misread as "75ms."

Tero flags an unstable segment as a glitch when its maximum latency is lower by at least *LatGap* than the minimum latency of its two closest stable segments on each side (Fig. 1a).

Tero detects spikes iteratively: In the first iteration, it flags an unstable segment as a spike if its minimum latency exceeds by at least *LatGap* the maximum latency of its two closest stable segments on each side (second red triangle in Fig. 1b). In the second iteration, it flags an unstable segment as a spike if its minimum latency exceeds by at least *LatGap* the maximum latency of one neighbor, while the other neighbor was previously flagged as a spike (first red triangle in Fig. 1b). It repeats these iterations until no new spike is found, at which point it merges consecutive spikes (Fig. 1c). Finally, it cleans up: it revisits each unstable segment S that has not been flagged as a spike; if S's latency measurements are within *LatGap* from the measurements of the closest stable segment (on either side), S is left as is and un-flagged (green square in Fig. 1d); otherwise, S is discarded (red cross in Fig. 1d).

The very last step of this process—the discarding—may seem unnecessary, so, we explain: If glitches did not exist, an unstable segment would be either a spike or the result of a stable segment being interrupted by a spike. E.g., in Fig. 1d, stable segment 1 is interrupted by a spike (red triangle) and that yields an unstable segment (green square) that essentially "belongs" with stable segment 1. However, glitches introduce a third kind of unstable segment. E.g., in Fig. 1d, the segment marked with a red cross would have been stable if a glitch had not caused the removal of latency measurements, reducing its length and making it unstable. Any segment of this kind—that is neither a spike nor the result of a spike interrupting a stable segment—may be the result of a glitch, hence, most likely an image-processing error.

Tero tries to correct each segment S that has been flagged as a glitch or spike by replacing its latency measurements with their alternative values (§3.2). If S remains unstable after the correction, or if the correction is not possible (because S's latency measurements contain no alternative values), then S is discarded.

Tero also looks for *shared anomalies*: overlapping spikes that affected streamers playing from the same geographical location, hence may indicate an underlying technical problem in shared infrastructure. To do so, it adapts the statistical test from Schulman et al. [41]: it identifies candidate sets of spikes, and it determines that they form one shared anomaly if they were too many to have been independent with any reasonable probability. We describe how we adapt the test to our context in App. §F.

One difference from Schulman et al. [41] is that they consider end-users with known IP addresses, so they have higher-quality information for grouping end-users in relevant aggregates. In contrast, we know only a {*city*, *region*, *country*} tuple per streamer. Our best bet is to group streamers per game and region, because streamers from the same region typically play on the same server, and because these streamers are likely to share some network infrastructure. Recall that a "region" is the largest sub-division in a country, e.g., a US state, a Swiss canton, or a French province. So, it is reasonable to expect that streamers playing from the same region (e.g., Alabama, or the canton of Geneva) and on the same server (e.g., a server in Chicago, or Amsterdam) share some network infrastructure. If this expectation is wrong, Tero will fail to identify any true shared anomalies, but it will not identify any false shared anomalies—not if the statistical test is correct.

3.3.3 Latency Distributions. In summary, to compute latency distributions, Tero looks for streamers that yield "high-quality" (see next paragraph) latency information; and considers their latency when playing on a primary server and from a single location.

A streamer yields "high-quality" latency information when less than a fraction *MaxSpikes* of their latency measurements belong to spikes. This is a heuristic that resulted from experience: We found that streamers with an unusual number of spikes typically engage in game mislabeling (changing games without changing labels, leading the image-processing module to read data from the wrong part of the screen); or add custom elements to the screen (e.g., a clock or a subscriber counter) that are easily confused with latency values. The results we show in this paper were obtained for *MaxSpikes* = 50%, derived as follows: We look for streamers who appear to experience technical problems that do not affect other streamers. More specifically, we look for streamers that, if eliminated from the data-set, minimize (in their location) the number of spikes that are *not* shared.

Tero identifies possible location and server changes as follows:

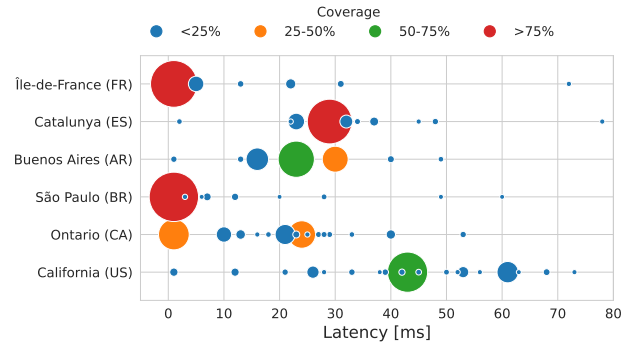


Figure 2: Examples of latency clusters. Each circle's size is proportional to the percentage of streamers inside the corresponding cluster.

(1) It considers only high-quality streamers, and it clusters each streamer's stable segments into similar-latency clusters: (a) It discards the streamer's spikes. (b) It clusters the remaining segments, such that: two segments belong to different clusters only if all their latency measurements differ by at least *LatGap*. (c) It annotates a cluster with weight w when it includes w% of the streamer's latency measurements.

(2) It classifies each streamer as *static* or *mobile*, depending on how many clusters they have: a streamer is *static* if they have one cluster with weight at least $MinWeight = 80\%$; otherwise, the streamer is *mobile*, which means that they may have played from multiple locations and/or on multiple servers.

(3) It uses the static streamers to identify similar-latency clusters for each {*location*, *game*} tuple. In particular, it considers only the highest-weight cluster from each streamer who has been located to *location* and is playing *game*; and merges these clusters such that: two clusters are *not* merged only if all their latency measurements differ by at least *LatGap*. Fig. 2 shows examples of latency clusters for various locations. We observe that most locations have only one or two clusters that are heavier than 10%. In the appendix (App. §G, Fig. 14), we show additional examples that illustrate how clusters change when using different merging criteria.

(4) It considers each mobile streamer who has been located to *location* and is playing *game*; if a pair of subsequent stable segments belong to different latency clusters (identified in Step 3), it marks the transition from one to the other as an end-point change. When an end-point change happens within the same stream, it is considered a server change (because we assume that a streamer will not change locations mid-stream). When an end-point change spans two subsequent streams, it is considered a possible location change.

To compute a latency distribution for {*location*, *game*}, Tero considers only streamers located in *location* and with no possible location changes. More specifically, it considers: (a) The static streamers. (b) From each mobile streamer, the latency measurements that belong to the highest-weight latency cluster for *location* and *game*.

Finally, for each resulting latency distribution, Tero also computes a version that is normalized by the corrected distance [44] (see next paragraph) between *location* and the corresponding "primary server," i.e., the one where users from *location* are most likely to play *game*. We obtain all the server locations at the granularity of city or region (App. §C). We do so based on developer reports [24, 37, 53], community information [2, 36], and articles posts from gaming-specific news media [31, 34].

Games	Genshin Impact, League of Legends
Bottleneck bandwidth	1Gbps, 100 Mbps
Bottleneck queue size	50, 500, 1000, 5000 packets
Traffic sources	2 UDP flows (50% BD each), 8 TCP flows (10% BD each, staggered by 5sec)
Experiment duration	5 minutes (2 with traffic)
Repetitions	5 per condition

Table 2: Parameters for experimental evaluation.

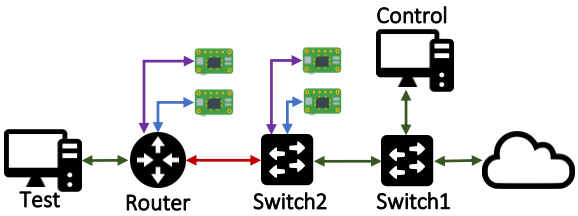


Figure 3: Testbed for experimental evaluation.

In most cases, the choice of the primary server is straightforward. E.g., there is one League of Legends server in Europe (in Amsterdam), and all players from Europe are supposed to play there. There are a few cases, however, where the choice is ambiguous. E.g., Call of Duty has 10 servers in North America and 8 servers in Europe, and players are assigned to servers based on measurements performed when they join the game. In these cases, we pick the server with the smallest corrected distance from *location*.

The "corrected distance" between a streamer and a server consists of two components: (1) The geodesic distance between the geometric centers of the streamer and server locations. E.g., if a streamer located in Athens is playing on an Amsterdam server, the first component is the geodesic distance between the geometric centers of Athens and Amsterdam. (2) The average distance of any point in the streamer's location from the location's geometric center. E.g., for a streamer located in Athens, the second component is the average distance of any point in Athens from Athens' geometric center. The second component is especially important when streamer and server are in the same location. E.g., if a streamer located in Amsterdam is playing on an Amsterdam server, then considering only the first component would yield a distance of 0. Ideally, the second component would consider how streamers are geographically distributed within Amsterdam, but we don't have this kind of information; considering the average distance to Amsterdam's geometric center is our best alternative.

4 EVALUATION

In this section, we answer two questions: how well does the latency reported by games follow the network-layer latency between the corresponding client and server? how often does Tero extract a wrong location or latency?

4.1 Gaming vs Network Latency

We use the term *gaming latency* to refer to the latency displayed as part of a game (which is meant to capture the application-layer RTT

between game server and play-station). We use the term *network latency* to refer to the network-layer RTT between two devices.

In summary, we find that the gaming latency closely follows network latency; when the network is severely congested, gaming latency tends to lag behind by a few seconds, i.e., when network latency increases, gaming latency takes a few seconds to reflect the increase.

When designing the experiments that led to these conclusions, we had to address two challenges: (a) We could not risk disturbing other players and potentially breaking competitive integrity, hence we restricted ourselves to 2 of the games supported by Tero (Table 2), which are single-player or have a single-player practice/-training mode. (b) Game servers do not typically respond to any type of probing packet, hence we could not directly measure network latency. Moreover, a game server typically communicates with a play-station through periodic updates, hence we could not rely on the timing of the server's packet arrivals to infer information about latency.

We used the topology in Fig. 3: We connected two play-stations, "Control" and "Test," to the same game server. The two play-stations share a common path to the game server, except that the path from Test includes an additional network bottleneck (between Router and Switch2) that is under our control. To create background traffic for the bottleneck, we used generator/sink devices connected directly to Router and Switch2 and iperf3[10]. The two play-stations are desktop computers with similar specifications: Test has an RTX 3070 GPU, 16 GB of RAM, and an AMD Ryzen 7 5800 processor, while Control has the same GPU, 32 GB of RAM, and an AMD Ryzen 9 3900X processor. The switches are off-the-shelf TP-Link 1Gb switches. The router is a MikroTik RouterBoard RB750GR3 hEX running RouterOS.

We experimented with a variety of network conditions, causing the bottleneck's network latency to vary from 0.4ms to 590ms. More specifically, for each game, we ran 8 experiments, repeating each experiment 5 times. Each experiment lasts 5 minutes: 2 minutes of "start-up," without background traffic; 1 minute with UDP background traffic; 1 minute with mixed UDP+TCP background traffic; and 1 minute of "die-down," again without background traffic. During start-up, we observe whether Control and Test see the same gaming latency; if not, we abort the experiment. Die-down allows the gaming latency to stabilize again after congestion ends. Across experiments, we varied the bottleneck bandwidth, the bottleneck queue size, and the volume of background traffic as specified in Table 2. During each experiment, we collected (5 times per second) the gaming latency displayed at each play-station, and we measured the network latency of the bottleneck link.

We assessed the difference between gaming and network latency as follows: First, we computed an *adjusted* gaming latency as the gaming latency displayed at Test minus the gaming latency displayed at Control at time t; and we compared it to the network latency of the bottleneck *also measured at time t*. Fig. 4 shows the difference between these two quantities, per experiment and per game; the x-axis represents experiment index, and experiments are sorted by the worst network latency they created. We see that, at worst, the 95th percentile of the difference between network

[10]https://iperf.fr/iperf-download.php

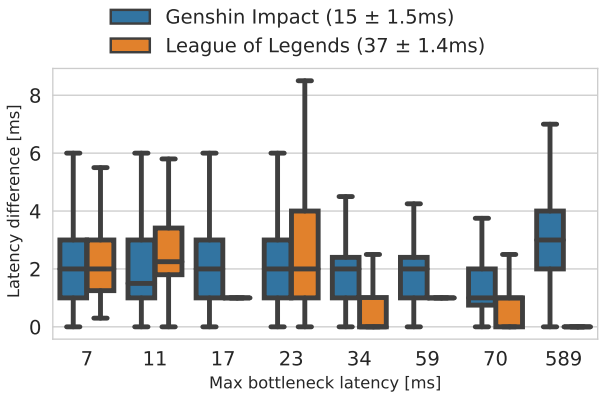


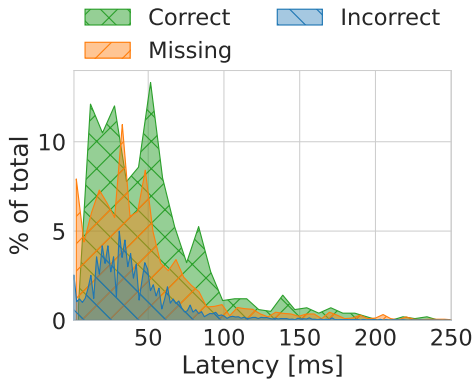
Figure 4: Difference between gaming and network latency. At the top, next to each game name, in parentheses, the average and standard deviation of the gaming latency displayed at Control.

and gaming latency was 8.5ms. Next, we focused on the specific moments in time when the difference was worse (above 4ms): They all corresponded to the beginning or end of background traffic, i.e., when network latency sharply increased or decreased. In all cases, the difference went back to 4ms or less within a few seconds. We posit that this "lag" between network and gaming latency is because gaming latency is computed as an average over a window of a few seconds, hence takes as much time to reflect a sharp change in network latency.

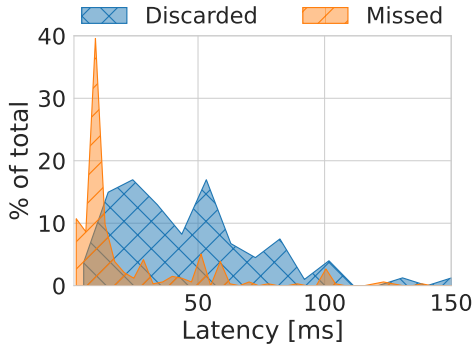
We also assessed whether the number of players on a server significantly affects the difference between gaming and network latency; we did not find any evidence that this is the case. We do not know the number of active players on a server, hence we used the number of active streamers playing on that server as a proxy. We focused on California, US, which is the region with the highest number of streamers in our data-set. We observed how the latency distribution computed by Tero for California and for different games changes as a function of the number of active streamers. In general, more active streamers does *not* imply higher gaming latency. E.g., for League of Legends, on a typical day, gaming latency is highest (average: 61ms, 95th percentile: 78ms) at 16:00 UTC, when the number of active streamers is 1,700, and it is lowest (average: 57ms, 95th percentile: 78ms) at 21:00 UTC, when the number of active streamers is two times higher (3,530). We hypothesize that gaming latency is higher during the day simply because the network is more loaded.

4.2 Error Rates

4.2.1 Location. Tero identified a location for 2.77% (722 thousand out of 26 million) of the streamers that were active during the considered time period; of the identified locations, we estimate that $1.46 \pm 0.07\%$ are incorrect due to errors of the underlying tools or incorrect mapping between Twitch accounts and Twitter profiles. To estimate this error rate, as well as the error rates of the underlying techniques, we inspected manually hundreds of input/output pairs from each technique. The most common reason for not locating a streamer or locating them incorrectly is that they advertise their



(a) Latency measurements that are correctly extracted, incorrectly extracted, and missed by image-processing.



(b) Incorrect latency measurements that are discarded and missed by data-analysis.

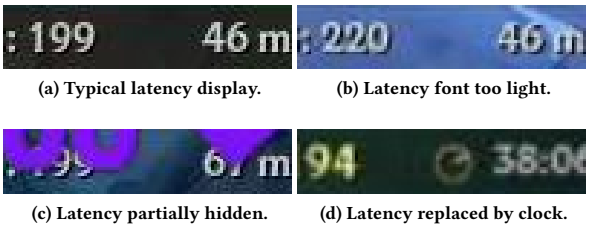
Figure 5: Image-processing and data-analysis errors.

	% extracted	Error rate
CLIFF	0.44%	33.4 ± 1.04%
Xponents	3.55%	36.27 ± 0.96%
Mordecai	0.81%	23 ± 0.50%
CLIFF++	63.99%	3.6 ± 0.31%
Xponents++	41.85%	2.87 ± 0.29%
Mordecai++	17.94%	2.43 ± 0.12%
Twitch Comb.	1.91%	3.47 ± 0.27%
Twitter-Twitch mapping	1.96%	1.6 ± 0.33%
Nominatim	70.83%	7.93 ± 1.05%
Geonames	69.55%	11.87 ± 0.47%
Twitter Comb.	70.77%	1.91 ± 0.29%
Tero	2.5 %	1.46 ± 0.07%

Table 3: Extraction and error rates of location techniques. "*Tool*++" indicates *Tool* augmented with a conservative filter that we designed (App. §D.1) to improve extraction and error rates. "Comb." indicates the combination of tools used on Twitch descriptions (App. §D.2) and Twitter locations (App. §D.3), respectively.

	Measurements not extracted	Incorrect measurements
EasyOCR	5.75 ± 0.38%	8.31 ± 0.73%
PaddleOCR	5.84 ± 0.33%	9.96 ± 0.75%
Tesseract	15.52 ± 0.57%	8.77 ± 0.26%
Tero	28.37 ± 0.47%	3.7 ± 0.40%

Table 4: Miss and error rates of OCR engines and their combination.



(a) Typical latency display. (b) Latency font too light.

(c) Latency partially hidden. (d) Latency replaced by clock.

Figure 6: Examples of OCR input.

location in an informal way that confuses geocoding/geoparsing tools (e.g., "I live in Denmarkian but have roots in Iran"). Table 3 summarizes the results. We describe our estimation method in App. §H.1.

4.2.2 Image Processing. We estimate that Tero fails to extract a latency measurement from 28.37% ± 0.47% of the downloaded thumbnails (that include a visible measurement); we estimate that 3.7% ± 0.4% of the measurements it extracts are incorrect. To estimate these error rates, as well as the error rate of the underlying OCR engines, we inspected manually thousands of thumbnails and the measurements extracted from them. Table 4 summarizes the results. We describe our estimation method in App. §H.2.

A valid concern about both missed (not extracted) and incorrect latency measurements is that they may introduce bias. E.g., one could imagine that Tero fails to extract correct measurements due to particularly low image resolution resulting from network problems, in which case missing/incorrect measurements would be correlated with high-latency measurements. To address this concern, we compared the estimated distribution of both missed and incorrect latency measurements against the estimated distribution of the correctly extracted latency measurements (Fig. 5a), and we found no indication of such bias.

In fact, the most common reason for missed measurements is that their font color is too close to that of the background (Fig. 6b), while the most common reason for incorrect measurements is digit drop as a result of the measurement being partially hidden by on-screen elements (Fig. 6c). Side note: The trickiest error we encountered was a streamer who modified their UI and made it display the current time exactly at the place where it normally displays latency (Fig. 6d).

4.2.3 Data Analysis. Of the incorrect latency measurements produced by image-processing, some are detected and corrected/discarded during data-analysis, as part of anomaly detection (§3.3.2); the question is how many escape. We assess that anomaly detection misses about 30% of the incorrect latency measurements. However, the reason these are missed is that they are close enough to their neighbors (within *LatGap*) that they do not constitute an anomaly

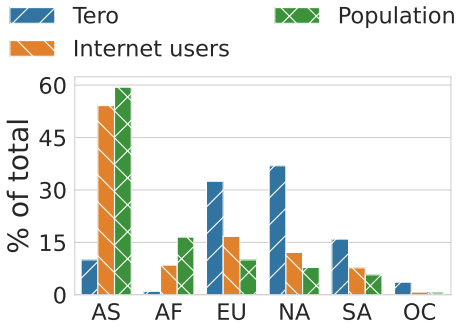


Figure 7: Distribution of Tero's users, Internet users, and global population by continent [5].

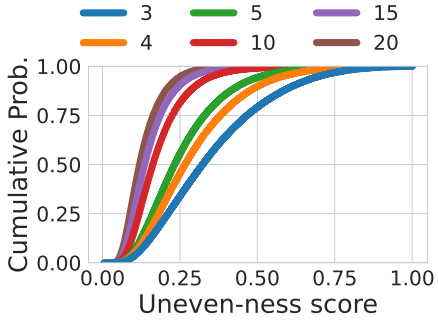


Figure 8: Uneven-ness score depending on the number of streamers per 5-minute interval.

(e.g., 101 is misread as 107). Hence, they do not significantly impact our regional latency analysis (which excludes anomalies).

We also assess that 25.87% ± 0.67 of the detected glitches do not actually include incorrect latency measurements, hence may be "false positives" (correct latency measurements that should not have been discarded). We think that these are true latency decreases due to a location or server change, but they ended up in unstable segments because the streamer interrupted their game. Our estimation method is in App. §H.3.

Finally, we would like to relate one instance where we were able to map shared anomalies to a probable cause. During 5 particular days (starting Nov 16, 2022), we detected an extraordinary number of shared spikes (669) in many geographical locations, and they all concerned one game. It turned out that a new version of the game was released exactly on Nov 16 [22], which makes it possible that the game servers themselves or their network connections were overloaded.

5 RESULTS

5.1 Basic Data Properties

Volume. Between May 25th, 2021 and May 1st, 2023, Tero processed 205 million thumbnails from 9 online video games. It extracted 64.6 million latency measurements. After filtering out anomalies, it retained 58.03 million measurements for further analysis, distributed across 150 thousand users from 195 countries, and spanning 3,903,121 streams.

Coverage. The geographical distribution of our streamers closely follows the distribution of Twitch users [59], which is concentrated in the Americas, Europe, and particular areas of Asia such as South Korea and Japan. If we exclude Asia, this distribution follows the one of Internet users (Fig.7). Our coverage of the Asian continent suffers despite the fact that gaming is popular there, because Twitch faces significant competition from Chinese and Indian streaming platforms [23, 55, 56]; we hope to improve by incorporating them.

The skew-ness of the geographic distribution is reflected in temporal coverage: In the locations where Twitch users are concentrated, we have data points as much as 70% of the time (this number is from California, US). However, there are also locations with only a few sporadic data points. In total, if we separate the data per game, we have enough data (at least 50 users) to compute latency distributions in 74 countries, 143 regions, and 90 cities.

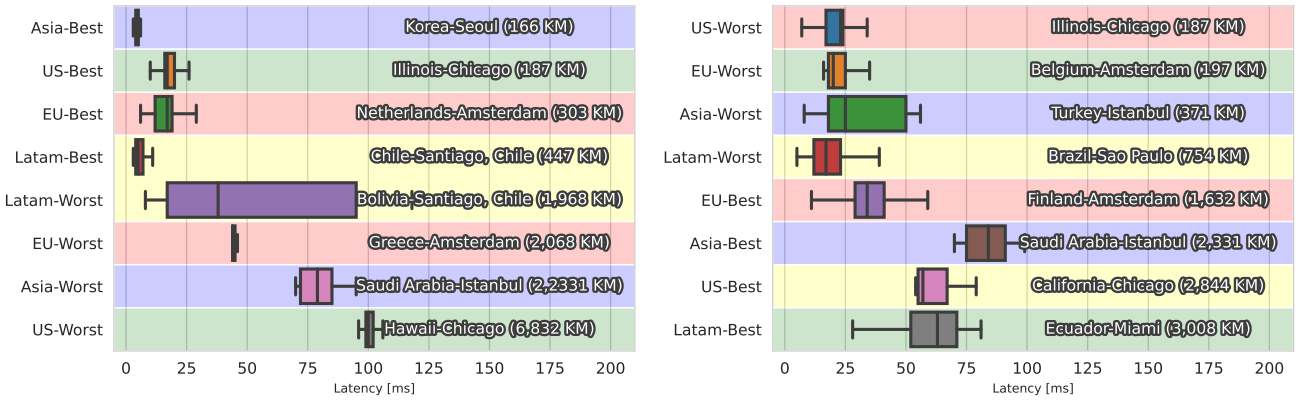
When multiple streamers play from the same location, their latency measurements are randomly distributed over time (it is *not* the case that Twitch produces thumbnails in bursts). To confirm this, we grouped latency measurements per location and 5-minute interval, and computed an "uneven-ness" score for each group: the Wasserstein distance [27] between the uniform distribution and the actual distribution of points across time, normalized by the Wasserstein distance between the uniform distribution and the most uneven distribution (all points happening at the same time). Fig. 8 shows the resulting uneven-ness CDFs; each curve/color corresponds to a different number of streamers active per 5-minute interval. We see that, even when we have 3 active streamers per interval, uneven-ness leans toward a uniform distribution 80% of the time.

5.2 Regional Latency

We start by comparing the latency distributions that we obtain for different geographical locations and the same game. Such a comparison makes sense in our context, even if the corresponding physical distances differ significantly: it shows the latency experienced by some of the most latency optimized Internet users around the world, when they access the same service (play the same game on their closest server).

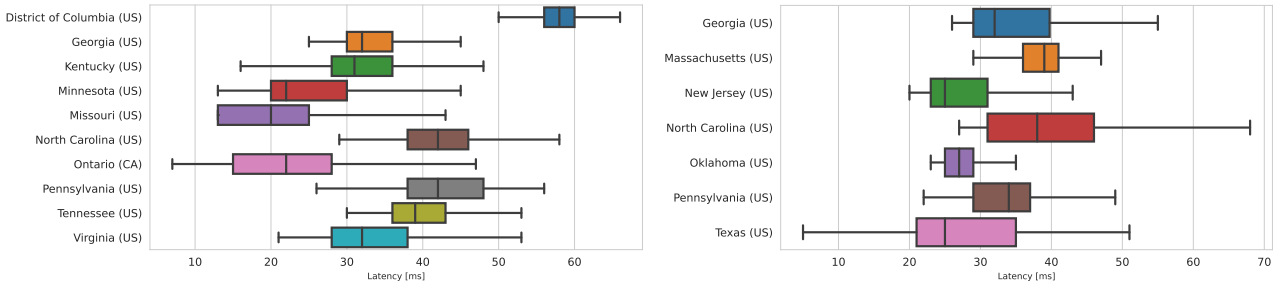
We plot each latency distribution as a boxplot that marks the 5th, 25th, 50th, 75th, and 95th percentiles. This differs slightly from a traditional boxplot, which marks the minimum and maximum values (excluding outliers) instead of the 5th and 95th percentiles. We chose this representation, because we expect up to 3.7% of the data points in a latency distribution to be incorrect due to image-processing error (§4.2.2). Hence, we want to conservatively exclude at least the 3.7% lowest and 3.7% highest data points from each distribution.

Fig. 9 shows the latency distributions for League of Legends and a set of geographic locations selected as follows: 9a includes the locations with the best and worst median latency from the US, Latin America, Europe, and Asia; 9b includes the locations with the best and worst median *distance-normalized* latency from the same places. To ensure that the distributions are comparable, we produced them using latency measurements from the same number of streamers (50 per location); for locations where we have data from more streamers, we randomly sampled 50. In both graphs, the x-axis represents absolute latency in ms. Each box is annotated with the name of the location, the location of the primary server,



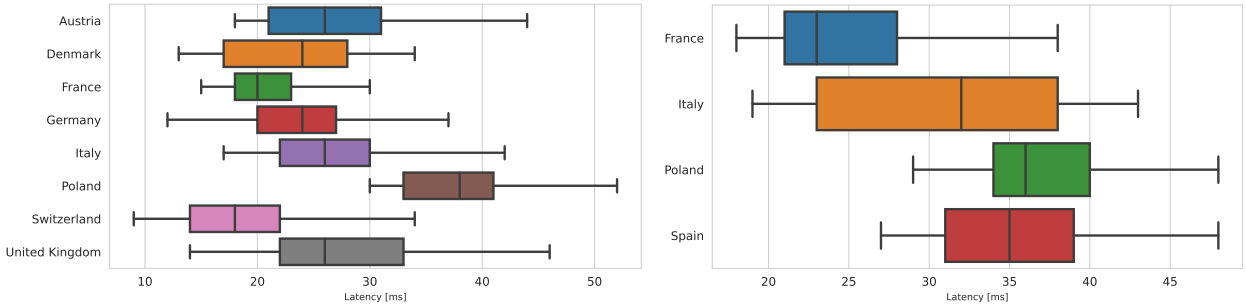
(a) Locations with best and worst absolute latency. **(b) Locations with best and worst distance-normalized latency.**

Figure 9: League-of-Legends absolute latency for different geographical locations.



(a) 500 KM < X < 1,000 KM **(b) 1,000 KM < X < 1,500 KM**

Figure 10: League-of-Legends absolute latency for US states in the same "doughnut".



(a) 500 KM < X < 1,000 KM **(b) 1,000 KM < X < 1,500 KM**

Figure 11: League-of-Legends absolute latency for EU countries in the same "doughnut".

and the average corrected distance (§3.3.3) between the server and the streamers (whose latency measurements contributed to the distribution). E.g., when we write "Turkey-Istanbul (371 km)," 371 km is the average corrected distance between the server's location (Istanbul) and the locations of all the streamers playing from Turkey.

There is a strong correlation between better latency and shorter physical distance to the game server (as expected), however, there are also significant differences in latency that cannot be justified by distance. Not surprisingly, the best latency corresponds to 4 locations that are less than 500 km away from their primary servers (the top 4 in Fig. 9a). However: the average corrected distance between Turkey (3rd from the top in Fig. 9b) and its primary server in Istanbul is only 371 km, yet its 75th percentile latency is, at 25ms, the same as Brazil's (4th in Fig. 9b), which is at double the distance from its primary server in Sao Paulo. Bolivia (4th from the bottom in Fig. 9a) is 1,900 km away from its primary server in Santiago, yet its 75th percentile latency is, at 100ms, the same as Hawaii's (bottom in Fig. 9a), which is 6,800 km away from its primary server in Chicago.

Greece and Saudi Arabia (2nd and 3rd from the bottom in Fig. 9a) have similar physical distances from their primary servers, yet their 75th percentile latency differs by around 25 ms (a significant difference for latency-sensitive applications).

We observed the most surprising latency differences in North America. Fig. 10 shows the latency distributions of different US states[11] that fall within the same 500-km-thick "doughnut," centered at the primary server in Chicago. Consider first Fig. 10a, which covers states 500–1000 km away from the primary server: the highest 75th percentile latency is around 60ms (District of Columbia), while the lowest is around 15ms (Missouri and Ontario, Canada). Now consider Fig. 10b, which covers states 1000–1500 km away from the primary server: the highest 75th percentile latency again exceeds 45ms (North Carolina), while the lowest is around 21ms (Texas). So: states that fall in the same 500-km-thick doughnut differ by as much as 30ms in their 75th percentile—again, a significant difference for latency-sensitive applications.

We observed smaller latency differences in Europe. Fig. 11 shows the latency distributions of different EU countries[12] that fall within the same 500-km-thick doughnut, centered at the primary server in Amsterdam. In Fig. 11a, the highest 75th percentile latency exceeds 40ms (Poland), while the lowest is 15ms (Switzerland). At the same time, some countries exhibit significantly bigger latency gaps across streamers than others. E.g., in Fig. 11b, for Italy, the gap between 25th and 75th percentile latency exceeds 15ms, while for France the same gap is around 5ms.

To consider possible explanations for these differences, one needs the following context: League-of-Legends is hosted by the game provider's private cloud, which has points of presence in all the major Internet eXchange Points (IXPs) and (at least in North America and Europe) peers with all the major eyeball ISPs. Hence, traffic from a player to a League-of-Legends game server typically crosses the player's ISP to the closest peering point with the game provider, then the game provider's private backbone to the server. Hence, when two states/countries or two streamers from the same country experience significantly different latency to the same server, the difference must come either from (a) their eyeball ISPs or from (b) the game provider's backbone. We do not have data to assign probabilities, but we posit that (b) is unlikely, especially when the two states have similar distances to the game server, given that game-provider backbones are particularly optimized for low latency.

We think that this kind of information can be useful to the entities responsible for Internet infrastructure, as, at the very least, it indicates states that may suffer from relatively poor residential connectivity.

As a final note, Tero obtains latency distributions for two countries with no RIPE probes or any other kind of publicly available measurements, to the best of our knowledge: El Salvador and Jamaica. Fig. 12 shows their latency distributions for League of Legends, compared against those of other locations that have similar distance (±200 km) from the primary game server in Miami.

[11] Georgia and North Carolina include streamers located in both doughnuts. For each of these states, we compute two different latency distributions, one per doughnut.
[12] France, Italy, and Poland have streamers in both doughnuts. For each of them, we compute two different latency distributions.

6 DISCUSSION: ANALYZING USER BEHAVIOR

We think that extracting publicly available latency measurements from latency-sensitive applications is an interesting direction because it opens the door (to all researchers) for real-time analysis of user behavior.

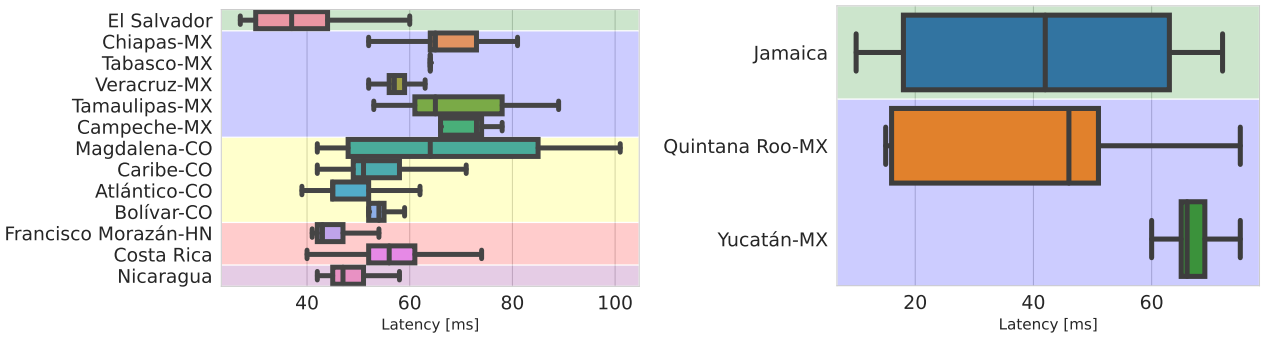
We take a small first step in this direction by studying how latency spikes impact the attitude of the affected players: we study the hypothesis that latency spikes (and the resulting degradation in quality of experience) could be a reason why players decide to change servers or abandon the game they are playing altogether.

We start from server changes: out of 196,019 {*streamer*, *game*} tuples, 6,110 (3.12%) experience at least one server change (as detected in §3.3.3). We limit our analysis to these tuples as they represent scenarios where a player is *able and willing to change servers*. In general, a player may be unable to change servers (e.g., due to their location giving them access only to one server) or unwilling to do so (e.g., to avoid changing playing partners). We consider each game separately because it has been shown that different types of video games elicit different responses to latency [10].

We prepare our data for analysis as follows: (1) We consider all streams from the selected streamers. (2) We discard any stream that is shorter than the minimum time that a player must play until they are allowed to change servers. (3) We consider streams with at least one server change, and measure the median time to the first change; then we truncate streams *without* a server change to this median time. This step is necessary, because streams with server changes are statistically longer than streams without server changes; and we need them to be of comparable length in order to test on them the hypothesis that latency spikes cause server changes. (4) We annotate each stream-without-a-change with its number of spikes, and each stream-with-a-change with its number of spikes before the first change.

We rely on Probit regression models [21], which are a standard statistics tool for assessing the effect of a "treatment" (in our case, latency spikes) on a binary outcome (in our case, server change or not). To assess the effect of different latency increases, we group spikes by their size, starting from 8ms (the minimum latency change that has been shown to be perceptible by human users [32]). Then we fit a Probit model per game and spike size (in each case, our "independent/predicting variable" is the number of spikes associated with each stream, and the "dependent variable" is whether the stream includes a server change or not). Each model is a prediction function that gives the probability of a server change given the number of spikes. A typical way to summarize a model is the average "marginal effect," which is the slope of the prediction function, i.e., how the probability of the binary outcome changes when one extra unit of the predicting variable is added.

Table. 5, at the top, shows the average marginal effect of the number of spikes on the probability of a server change. E.g., for League of Legends, adding one spike of 25ms or more increases the probability of a player changing server by 0.48%. For each game, we mark the highest marginal effect in bold. All effects are statistically significant at 1%, except those marked with a *, which are significant at 10%; empty cells indicate no statistically significant correlation. "Significant at p%" means that: if server changes were independent



(a) El Salvador. **(b) Jamaica.**

Figure 12: League-of-Legends absolute latency for locations at a similar distance (±200 km) as El Salvador (left) and Jamaica (right) from the Miami game server.

	Server changes								
Game	N obs	>8ms	>10ms	>15ms	>20ms	>25ms	>30ms	>35ms	>40ms
League of Legends	16,587	0.0039	0.0043	0.0044	0.0046	**0.0048**	0.0047	**0.0048**	**0.0048**
Call of Duty Warzone	95,416	0.0025	0.0029	0.0041	0.0051	0.0061	0.0069	0.0076	**0.0079**
Genshin Impact	14,946	0.0059	0.0066	0.0069	**0.0074**	0.0069	0.0069	0.0067	0.0068
Teamfight Tactics	8,174	0.0053	0.0056	0.0074	0.0077	0.0082	**0.0086**	**0.0086**	0.0081
Dota 2	8,860	0.0038	0.0042	0.0049	0.0060	0.0063	**0.0069**	0.0063	0.0074
Among Us	1,351	0.0121	0.0130	0.0130	0.0103*	**0.0164**	-	-	-
Lost Ark	684	0.0041	0.0036	0.0044	**0.0159**	0.0156	0.0155	-	-
	Game changes								
Game	N obs	>8ms	>10ms	>15ms	>20ms	>25ms	>30ms	>35ms	>40ms
League of Legends	1,483,918	0.0229	0.0255	**0.0280**	0.0274	0.0271	0.0262	0.0251	0.0245
CoD Warzone	724,157	0.0092	0.0129	0.0174	0.0217	0.0245	0.0266	0.0281	**0.0290**
Genshin Impact	268,491	0.0358	0.0372	0.0405	0.0406	**0.0414**	0.0410	0.0405	0.0402
Teamfight Tactics	98,470	0.0218	0.0222	**0.0252**	0.0247	0.0247	0.0218	0.0212	0.0205
Dota 2	130,742	0.0132	0.0140	0.0162	0.0177	0.0178	**0.0187**	0.0184	0.0180
Among Us	9,816	0.0358	0.0382	0.0421	0.0434	0.0409	**0.0458**	0.0452	**0.0458**
Lost Ark	41,072	0.0129	0.0129	0.0171	0.0329	0.0337	**0.0343**	0.0302	0.0269

Table 5: Average marginal effects of the number of spikes on the presence of a server or game change.

of spikes, the probability of observing what we observed (the given spikes and server changes) would be p%.

So: The effect of spikes on server changes is small, but statistically significant (a small but clear correlation does exist). E.g., for League of Legends, one extra spike of 15ms or more in a stream yields around 0.44% increase in the probability of the player changing server. For some games, spike size has a bigger effect, e.g., for Call of Duty Warzone, a player is 68% more likely to change server (a change in probability from 0.41% to 0.69%) if the spikes are >30ms instead of >15ms.

Then we consider game changes, i.e., we study the hypothesis that spikes may cause players to change games, looking for a better quality of experience. To determine *when* streamers change games, every 30 minutes we fetch from Twitch all information about ongoing streams (including the game currently being streamed). We cross-reference this data with our latency measurements and obtain relevant game changes: for all streamers included in our data-set, we look for instances in which the streamer changes from (or to) a game included in our data-set. Out of all the streams included in our data-set (4,280,654), 2,274,761 (53.14%) contain a game change with the required characteristics. We perform a similar analysis as for server changes and show the results in Table. 5, at the bottom.

We see that the effect of spikes on game changes is an order of magnitude higher than on server changes. In our opinion, the reason behind this difference is that it is significantly easier to change games than servers: First, depending on the player's location, a server change may be impossible or inconvenient due to language or cultural barriers. Second, changing servers typically requires menu navigation, while changing games is as easy as starting a new game.

The magnitude of the effects we observe is consistent with state-of-the-art research on the effect of latency on user behavior in general. E.g., Nam et al. [33] studied why users stop watching videos and found relationships comparable to ours, e.g., the number of rebuffering events has a marginal effect on the abandonment rate of 2.46%, which is close to the marginal effect of spikes on game changes. Krishnan et al. [28] found that a delay of 1 second at the start of a video increases the abandonment rate by 5.8%. Finally,

Google and Bing reported in a joint conference presentation that an added delay of 200ms in response times yields a decrease of 0.4% in user satisfaction.

Our preliminary study shows the potential of Tero—or any system that collects measurements directly from network-sensitive applications—to tie network metrics to user behavior. And while it is reasonable to assume that latency spikes affect game retention, we think it is interesting to put specific numbers on retention rate as a function of latency.

7 DISCUSSION: ETHICAL CONSIDERATIONS

Tero touches three kinds of entities: the streamers whose latency we extract; the streaming platforms that make the gaming footage publicly available; and the social-media platforms that host the streamers' profiles.

We take extreme care to not even look for information that a streamer did not clearly intend to share publicly. In particular, we never try to localize a streamer at a granularity finer than a city. To minimize privacy risks for our streamers, we take the following measures: (1) We store the minimum information required for our system: approximate geographic location and latency measurements from gaming footage. (2) We use consistent hashing to map each streamer ID to a randomly generated ID; this is because we need to remember that a location and a set of measurements belong to the same streamer, but we do not need to remember the streamer's actual ID. (3) We delete any intermediate information—the actual streamer IDs, the downloaded profiles, and the downloaded thumbnails—as soon as we process it.

We also take extreme care to comply with the Terms of Service of each streaming and/or social-media platform. In particular: As mentioned in §1, to comply with Twitch's Terms of Service, we neither access video streams nor crawl streamer profiles; we obtain gaming footage through Twitch's CDN and related metadata through its Developer API. Twitter allows[13] to draw connections between platforms that someone would "reasonably expect" for us to make; we interpret this strictly and look only for explicit links left by a user themselves from their own Twitter and/or Steam account to their own Twitch account.

8 RELATED WORK

To the best of our knowledge, our work is the first to extract latency measurements from gaming footage. Closest in spirit is the line of work that passively measures the transport-layer latency of active TCP connections: Chen et al. [7] conduct their measurements on a programmable switch, enabling real-time reaction by the corresponding provider. Ruru [11] and Veal et al [58] conduct similar measurements by leveraging certain parts of a TCP connection, e.g., the SYN/SYN-ACK handshake and packet-ACK pairs that share timestamps. Another related line of work gains insights on Internet latency by leveraging proprietary passive measurements [15, 17, 25, 43]. Choffnes et al. [8] develop a BitTorrent extension that enables them to monitor the BitTorrrent performance experienced by end-users and detect interesting events. With all this work, we share the goals of reasoning about latency in the face of noise; however, each line of work faces distinct challenges (including different kinds of noise). Our unique challenges are mapping our streamers to geographical locations, extracting measurements from their thumbnails, and handling the peculiar pattern of the resulting data.

We view Tero as complementary to active measurement platforms like RIPE Atlas[51], BISmark [52], Speedchecker[14], SamKnows[15], and Mlab-hosted projects like NDT [14]. These enable precise, controlled network-layer measurements, conducted on dedicated servers and potentially on the devices of volunteering end-users. In comparison, Tero yields significantly noisier measurements; its contribution is that it reveals the application-layer latency actually experienced by tens of thousands of streamers around the world, without the need for dedicated, publicly available, or volunteered measurement infrastructure. We should also mention Microsoft's Odin [6] and Facebook's platform described by Schlinker et al [48], which are great examples of precise, controlled, albeit closed active-measurement platforms.

Many research projects leverage open active measurement platforms like ours: Dang et al. [12] leverage Speedchecker data to study end-user connectivity to different cloud providers, as well as end-user latency as a function of data-center deployment and choice of networking infrastructure. Swati et al. [46] leverage BISmark data to identify and analyze correlated latency anomalies in access networks (we should clarify that they operate at a different scale, they consider average latency per day). Høiland-Jørgensen et al. [19] leverage NDT data to study the evolution of latency over years. Schulman et al. [50] leverage PlanetLab [9] to deploy Thunderping, which measures the connectivity of residential Internet hosts before, during, and after periods of severe weather. Padmanabhan et al. [41] leverage the Thunderping data-set to identify correlated instances of failures that are likely due to severe weather; as already stated, we borrow their technique for identifying shared anomalies.

9 CONCLUSION

Gaming footage is a source of real-time, publicly available, passive latency measurements that are directly contributed by tens of thousands of some of the most latency-sensitive and latency-optimized Internet users; we showed that it is feasible to mine this source of information to complement the picture of Internet latency provided by open measurement platforms.

ACKNOWLEDGMENTS

We would like to thank Ankit Singla for his valuable input and support during the early stages of the work. We would also like to thank our shepherd, Oliver Hohlfeld, and the anonymous reviewers for their constructive feedback.

[13] Off-Twitter matching https://developer.twitter.com/en/developer-terms/agreement-and-policy

[14] https://www.speedchecker.com/

[15] https://www.samknows.com/

REFERENCES

[1] Beatrice Alex, Clare Llewellyn, Claire Grover, Jon Oberlander, and Richard Tobin. 2016. Homing in on Twitter users: evaluating an enhanced geoparser for user profile locations. In *Proceedings of the Tenth International Conference on Language Resources and Evaluation (LREC'16)*. 3936–3944.

[2] What are the dota servers and can I ping them? [n. d.]. https://www.reddit.com/r/learndota2/comments/9zk8o3/what_are_the_dota_servers_and_can_i_ping_them/. Accessed: 2023-05-23.

[3] RIPE Atlas. [n. d.]. https://atlas.ripe.net/. Accessed: 2023-08-31.

[4] Markus M Breunig, Hans-Peter Kriegel, Raymond T Ng, and Jörg Sander. 2000. LOF: identifying density-based local outliers. In *Proceedings of the 2000 ACM SIGMOD international conference on Management of data*. 93–104.

[5] Internet Users by Country (2016). [n. d.]. https://www.internetlivestats.com/internet-users-by-country/. Accessed: 2022-09-19.

[6] Matt Calder, Ryan Gao, Manuel Schröder, Ryan Stewart, Jitendra Padhye, Ratul Mahajan, Ganesh Ananthanarayanan, and Ethan Katz-Bassett. 2018. Odin:{Microsoft's} Scalable {Fault-Tolerant} {CDN} Measurement System. In *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18)*. 501–517.

[7] Xiaoqi Chen, Hyojoon Kim, Javed M Aman, Willie Chang, Mack Lee, and Jennifer Rexford. 2020. Measuring TCP round-trip time in the data plane. In *Proceedings of the Workshop on Secure Programmable Network Infrastructure*. 35–41.

[8] David R Choffnes, Fabián E Bustamante, and Zihui Ge. 2010. Crowdsourcing service-level network event monitoring. In *Proceedings of the ACM SIGCOMM 2010 Conference*. 387–398.

[9] Brent Chun, David Culler, Timothy Roscoe, Andy Bavier, Larry Peterson, Mike Wawrzoniak, and Mic Bowman. 2003. Planetlab: an overlay testbed for broad-coverage services. *ACM SIGCOMM Computer Communication Review* 33, 3 (2003), 3–12.

[10] Mark Claypool and Kajal Claypool. 2010. Latency can kill: precision and deadline in online games. In *Proceedings of the first annual ACM SIGMM conference on Multimedia systems*. 215–222.

[11] Richard Cziva, Christopher Lorier, and Dimitrios P Pezaros. 2017. Ruru: High-speed, flow-level latency measurement and visualization of live internet traffic. In *Proceedings of the SIGCOMM Posters and Demos*. 46–47.

[12] The Khang Dang, Nitinder Mohan, Lorenzo Corneo, Aleksandr Zavodovski, Jörg Ott, and Jussi Kangasharju. 2021. Cloudy with a chance of short RTTs: analyzing cloud connectivity in the internet. In *Proceedings of the 21st ACM Internet Measurement Conference*. 62–79.

[13] Catherine D'Ignazio, Rahul Bhargava, Ethan Zuckerman, and Luisa Beck. 2014. Cliff-clavin: Determining geographic focus for news articles. NewsKDD: Data Science for News Publishing, at KDD 2014.

[14] Constantine Dovrolis, Krishna Gummadi, Aleksandar Kuzmanovic, and Sascha D Meinrath. 2010. Measurement lab: Overview and an invitation to the research community. *ACM SIGCOMM Computer Communication Review* 40, 3 (2010), 53–56.

[15] Ramakrishnan Durairajan, Sathiya Kumaran Mani, Joel Sommers, and Paul Barford. 2015. Time's forgotten: Using ntp to understand internet latency. In *Proceedings of the 14th ACM Workshop on Hot Topics in Networks*. 1–7.

[16] Julian D Gilbey and Carola-Bibiane Schönlieb. 2021. An end-to-end Optical Character Recognition approach for ultra-low-resolution printed text images. *arXiv preprint arXiv:2105.04515* (2021).

[17] Vasilis Giotsas and Marwan Fayed. 2021. "Look, Ma, no probes!" — Characterizing CDNs' latencies with passive measurement. https://blog.cloudflare.com/cdn-latency-passive-measurement/.

[18] Andrew Halterman. 2017. Mordecai: Full Text Geoparsing and Event Geocoding. *The Journal of Open Source Software* 2, 9 (2017).

[19] Toke Høiland-Jørgensen, Bengt Ahlgren, Per Hurtig, and Anna Brunstrom. 2016. Measuring latency variation in the internet. In *Proceedings of the 12th International on Conference on emerging Networking EXperiments and Technologies*. 473–480.

[20] Thomas Holterbach, Cristel Pelsser, Randy Bush, and Laurent Vanbever. 2015. Quantifying interference between measurements on the RIPE Atlas platform. In *Proceedings of the 2015 Internet Measurement Conference*. 437–443.

[21] Nick Huntington-Klein. 2021. *The effect: An introduction to research design and causality*. CRC Press.

[22] Call Of Duty: Modern Warfare II and including DMZ Call Of Duty: Warzone 2.0 Season 01: Everything you need to know. [n. d.]. https://www.callofduty.com/blog/2022/11/call-of-duty-modern-warfare-II-warzone-2-0-season-01-overview-battle-pass-dmz. Accessed: 2023-05-27.

[23] For Gamers in India YouTube Is the Ultimate Live Streaming Platform. [n. d.]. https://gadgets360.com/games/features/youtube-gaming-live-video-game-streaming-india-gamers-ines-cha-2147975. Accessed: 2022-09-11.

[24] Server Issues and a new update! [n. d.]. https://www.innersloth.com/server-issues-and-a-new-update/. Accessed: 2023-05-23.

[25] Yuchen Jin, Sundararajan Renganathan, Ganesh Ananthanarayanan, Junchen Jiang, Venkata N Padmanabhan, Manuel Schroder, Matt Calder, and Arvind Krishnamurthy. 2019. Zooming in on wide-area latencies to a global cloud provider. In *Proceedings of the ACM Special Interest Group on Data Communication*. 104–116.

[26] Rebecca Killick, Paul Fearnhead, and Idris A Eckley. 2012. Optimal detection of changepoints with a linear computational cost. *J. Amer. Statist. Assoc.* 107, 500 (2012), 1590–1598.

[27] Soheil Kolouri, Se Rim Park, Matthew Thorpe, Dejan Slepcev, and Gustavo K Rohde. 2017. Optimal mass transport: Signal processing and machine-learning applications. *IEEE signal processing magazine* 34, 4 (2017), 43–59.

[28] S Shunmuga Krishnan and Ramesh K Sitaraman. 2012. Video stream quality impacts viewer behavior: inferring causality using quasi-experimental designs. In *Proceedings of the 2012 Internet Measurement Conference*. 211–224.

[29] Fei Tony Liu, Kai Ming Ting, and Zhi-Hua Zhou. 2008. Isolation Forest. In *Proceedings of the 2008 Eighth IEEE International Conference on Data Mining*. 413–422.

[30] Shengmei Liu, Mark Claypool, Atsuo Kuwahara, James Scovell, and Jamie Sherman. 2021. L33t or N00b? How Player Skill Alters the Effects of Network Latency on First Person Shooter Game Players. In *Proceedings of the Workshop on Game Systems (GameSys' 21)*. 1–6.

[31] Apex Legends Server Locations. [n. d.]. https://netduma.com/blog/apex-legends-server-locations/. Accessed: 2023-05-23.

[32] Katerina Mania, Bernard D Adelstein, Stephen R Ellis, and Michael I Hill. 2004. Perceptual sensitivity to head tracking latency in virtual environments with varying degrees of scene complexity. In *Proceedings of the 1st Symposium on Applied Perception in Graphics and Visualization*. 39–47.

[33] Hyunwoo Nam, Henning Schulzrinne, Hyunwoo Nam, Kyung-Hwa Kim, Henning Schulzrinne, Martín Varela, Hyunwoo Nam, Henning Schulzrinne, Toni Mäki, Hyunwoo Nam, et al. 2016. Youslow: What influences user abandonment behavior for internet video? *Columbia University Rcp* (2016).

[34] Call of Duty: Warzone 2.0 Server Locations. [n. d.]. https://netduma.com/blog/call-of-duty-warzone-2-0-server-locations/. Accessed: 2023-05-23.

[35] Game Durations League of Legends. 2023. https://www.leagueofgraphs.com/stats/game-durations. Accessed: 2023-04-26.

[36] Servers: League of Legends. [n. d.]. https://leagueoflegends.fandom.com/wiki/Servers. Accessed: 2023-05-23.

[37] League of Legends Regional Servers. [n. d.]. https://support-leagueoflegends.riotgames.com/hc/en-us/articles/201751684-League-of-Legends-Regional-Servers. Accessed: 2023-05-23.

[38] Manuel Oliveira and Tristan Henderson. 2003. What online gamers really think of the Internet?. In *Proceedings of the 2nd workshop on Network and system support for games*. 185–193.

[39] Warzone 2 Game Times Will Be Much Longer Than The Original. [n. d.]. https://www.ggrecon.com/articles/warzone-2-game-times-will-be-much-longer-than-the-original/. Accessed: 2023-04-26.

[40] Nobuyuki Otsu. 1979. A threshold selection method from gray-level histograms. *IEEE transactions on systems, man, and cybernetics* 9, 1 (1979), 62–66.

[41] Ramakrishna Padmanabhan, Aaron Schulman, Alberto Dainotti, Dave Levin, and Neil Spring. 2019. How to find correlated internet failures. In *International Conference on Passive and Active Network Measurement*. 210–227.

[42] Streamlabs & Stream Hatchet Q1 2021 Live Streaming Industry Report. [n. d.]. https://streamlabs.com/content-hub/post/streamlabs-and-stream-hatchet-q1-2021-live-streaming-industry-report. Accessed: 2022-05-13.

[43] Philipp Richter, Ramakrishna Padmanabhan, Neil Spring, Arthur Berger, and David Clark. 2018. Advancing the art of internet edge outage detection. In *Proceedings of the Internet Measurement Conference 2018*. 350–363.

[44] A Rodriguez-Bachiller. 1983. Errors in the measurement of spatial distances between discrete regions. *Environment and Planning A* 15, 6 (1983), 781–799.

[45] Peter J Rousseeuw and Katrien Van Driessen. 1999. A fast algorithm for the minimum covariance determinant estimator. *Technometrics* 41, 3 (1999), 212–223.

[46] Swati Roy and Nick Feamster. 2013. Characterizing correlated latency anomalies in broadband access networks. In *Proceedings of the ACM SIGCOMM 2013 conference on SIGCOMM*. 525–526.

[47] Amazon EC2 Auto Scaling. [n. d.]. https://aws.amazon.com/ec2/autoscaling/. Accessed: 2022-05-18.

[48] Brandon Schlinker, Italo Cunha, Yi-Ching Chiu, Srikanth Sundaresan, and Ethan Katz-Bassett. 2019. Internet performance from facebook's edge. In *Proceedings of the Internet Measurement Conference*. 179–194.

[49] Sebastian Schmidl, Phillip Wenig, and Thorsten Papenbrock. 2022. Anomaly detection in time series: a comprehensive evaluation. *Proceedings of the VLDB Endowment* 15, 9 (2022), 1779–1797.

[50] Aaron Schulman and Neil Spring. 2011. Pingin'in the rain. In *Proceedings of the 2011 ACM SIGCOMM conference on Internet measurement conference*. 19–28.

[51] RIPE NCC Staff. 2015. Ripe atlas: A global internet measurement network. *Internet Protocol Journal* 18, 3 (2015).

[52] Srikanth Sundaresan, Sam Burnett, Nick Feamster, and Walter De Donato. 2014. {BISmark}: A Testbed for Deploying Measurements and Applications in Broadband Access Networks. In *2014 USENIX Annual Technical Conference (USENIX ATC 14)*. 383–394.

[53] About the Among Us servers. [n. d.]. https://innersloth.zendesk.com/hc/en-us/articles/9686064498580-About-the-Among-Us-servers. Accessed: 2023-05-23.
[54] Improving the quality of the output. [n. d.]. https://tesseract-ocr.github.io/tessdoc/ImproveQuality.html. Accessed: 2022-05-18.
[55] Guide to Live Streaming in China 2022. [n. d.]. https://seoagencychina.com/live-streaming-marketing-guide-china/. Accessed: 2022-09-11.
[56] China Bans Twitch. [n. d.]. https://www.businessinsider.com/china-bans-twitch-2018-9. Accessed: 2022-09-11.
[57] Marc Ubaldino. 2019. OpenSextant Xponents: Geotagging Toolkit for World-wide Geography. https://opensextant.github.io/Xponents/. Accessed: 2023-03-15.
[58] Bryan Veal, Kang Li, and David Lowenthal. 2005. New methods for passive estimation of TCP round-trip times. In *International workshop on passive and active network measurement*. Springer, 121–134.
[59] Which 15 Countries Have The Most Twitch Viewers? [n. d.]. https://visualsbyimpulse.com/countries-most-twitch-viewers-top-15/. Accessed: 2022-09-11.

J APPENDIXES

A Download Module

The downloading module consists of a *coordinator* and multiple parallel *downloaders*; the former detects when streamers start broadcasting, while the latter download thumbnails from broadcasting streamers. The reason for the split between the coordinator and downloader roles is that downloading thumbnails is time-sensitive: each streamer is assigned a URL, where their latest thumbnail is posted every 5 minutes, overwriting the previous one; if Tero fails to download a thumbnail before it is overwritten, it simply loses the thumbnail. For this reason, we keep the downloaders as lean as possible, delegating all tasks—state keeping, failure recovery, etc.—but plain downloading to the coordinator.

The coordinator keeps a list of streamers and the last time it checked each streamer's status; and it periodically queries the Twitch API to identify streamers who have started broadcasting; given that Twitch rate-limits access to its API, the coordinator issues these queries in a way that respects the rate limit. When the coordinator detects that a streamer has started broadcasting, it writes the corresponding URL (where the streamer's thumbnails are posted) in a Key-Value store for the downloaders to retrieve. Also, the coordinator periodically checks the Key-Value store for signals written by the downloaders that a previously active streamer has gone offline. Finally, in case of a system crash, the coordinator is in charge of recovering all the necessary state from the Key-Value store.

A downloader retrieves URLs, written by the coordinator, from the Key-Value store and periodically downloads thumbnails from each URL. To do this, it keeps a list of all the URLs (that this downloader is in charge of) and the time when a new thumbnail will become available at each one. When it is time to download a thumbnail, the downloader first issues a HEAD request, in order to obtain the time when the next thumbnail will become available; it stores the response in its list, downloads the thumbnail, and stores it in an object store. If the streamer has gone offline, the URL redirects to a generic offline URL; in that case, the downloader removes the URL from its list and writes the streamer's name in the Key-Value store for the coordinator to retrieve.

To scale as the number of streamers grows, we use a simple load-balancing approach: a downloader takes on a new streamer whenever it becomes idle (there is no new thumbnail available from any of its current streamers). A more sophisticated approach would be to use CPU or memory utilization to identify overloaded downloaders and split their load as, for example, AWS offers with their Auto Scaling capability [47]. However, we found thumbnail size—hence download time—to be so unpredictable, that Tero would not benefit from such techniques.

B Implementation

We run Tero on one machine with 2 Intel Xeon E5-2680 v3 CPUs (with 12 cores, 24 hyper-threads each), 256 GB of RAM, 2 Nvidia Titan X Maxwell GPUs, and a 10Gbps connection. Our implementation follows a micro-service architecture to ensure easy migration of parts—or potentially all—of the system, in case at some point the resources available in our machine do not suffice anymore. Hence, each module is a set of processes, each containerized and periodically invoked by the PM2 process manager[16]. We use three storage systems: (1) the Redis[17] Key-Value store for inter-process communication and to store streamer-location information; (2) an S3-like distributed object store based on Ceph[18] to store the thumbnails and the intermediate products of image-processing; and (3) a MongoDB[19] document store for the latency measurements and analysis. For implementation details, see App. §B.

Inter-process communication: Processes that produce information push it into the relevant store, and processes that consume information pull it from the relevant store when they are ready. This approach makes sense because processing time varies significantly—and sometimes unpredictably—across processes. For instance, the processes of the location module invoke different external (Twitch, Twitter, and Steam) APIs, each one with a different rate limit (imposed by the corresponding terms of service) and performance; instead of pushing to each process a list of Twitch usernames to locate, the latter are written to a Key-Value store, and each location process pulls them at the rate imposed by the corresponding API service terms and performance. Similarly, the processes of the image-processing module invoke different OCR engines (Tesseract, PaddleOCR, and EasyOCR), each one with different performance sensitivity to input; so, the thumbnails are written in a Key-Value store, and each image-processing process pulls a fixed-size batch when ready. If the available thumbnails are fewer than the batch size, no process pulls them, and this allows the slower processes to make use of the shared resources and catch up.

Failure recovery: If a process crashes, when it restarts it receives an index number from the coordinator so that the process can read most of its previous state, discarding recently pulled data, from a Key created using the index.

C Server Locations

Out of the 9 games currently processed by Tero, we were able to find information about server locations for 8 of them. Tables 6-7 show server locations for these games, at the finest granularity disclosed by the game provider, as well as the area of the world that is served by each.

[16] https://pm2.keymetrics.io/
[17] https://redis.io/
[18] https://github.com/ceph/ceph
[19] https://www.mongodb.com/

Game	Server location	Area served
League of Legends, Teamfight Tactics	Amsterdam, Netherlands	Europe
	Chicago, Illinois	US, Canada
	Sao Paulo, Brazil	Brazil
	Miami, Florida	Northern South America
	Santiago, Chile	Southern South America
	Sydney, Australia	Oceania
	Istanbul, Turkey	Middle East
	Seoul, Korea	Korea
	Tokyo, Japan	Japan
Dota 2	Virginia, USA	North America
	Seattle, USA	North America
	Vienna, Austria	Europe
	Luxemburg	Europe
	Santiago, Chile	South America
	Lima, Peru	South America
	Dubai, Saudi Arabia	Middle East
	Sydney, Australia	Oceania
	Tokyo, Japan	Asia
Genshin Impact	Virginia, USA	Americas
	Frankfurt, Germany	Europe and Middle East
	Tokyo, Japan	Asia
Honkai: Star Rail	Virginia, USA	Americas
	Frankfurt, Germany	Europe and Middle East
	Tokyo, Japan	Asia
Among Us	California, USA	Americas and Oceania
	Texas, USA	Americas and Oceania
	Frankfurt, Germany	Europe and Middle East
	Tokyo, Japan	Asia

Table 6: Server locations.

Server location	Area served
Salt Lake City, USA	North America
Los Angeles, USA	North America
San Francisco, USA	North America
Dallas, USA	North America
St. Louis, USA	North America
Colombus, USA	North America
New York, USA	North America
Chicago, USA	North America
Washington, USA	North America
Atlanta, USA	North America
London, England	Europe
Frankfurt, Germany	Europe
Amsterdam, Netherlands	Europe
Brussels, Belgium	Europe
Paris, France	Europe
Madrid, Spain	Europe
Stockholm, Sweden	Europe
Rome, Italy	Europe
Santiago, Chile	South America
Lima, Peru	South America
Sao Paulo, Brazil	South America
Riyadh, Saudi Arabia	Middle East
Sydney, Australia	Oceania
Tokyo, Japan	Asia

Table 7: Server locations for Call of Duty: Warzone and Call of Duty: Modern Warfare.

D Location: Underlying Tools

The location module combines geocoding and geoparsing tools. Geocoding tools take as input generic text, identify in it location information, and map the latter to a concrete location. Geoparsing tools take as input text that describes location and map it to a concrete location. So, geocoding tools can handle broader input than geoparsing tools. However, geoparsing tools tend to perform better when the input is restricted to location information.

D.1 Conservative Filter. Tero uses the following conservative filter: accept a tool's output location as valid if the input description contains at least the *country* or *region* field of the output location. For example, consider the description "Join us in Detroit"; CLIFF extracts from it the location tuple (*United States*, *Michigan*, *Detroit*), however, the input description contains neither "United States" nor "Michigan"; hence, CLIFF's output is discarded (unnecessarily, in this case). As another example, consider the description "From Miami, Florida"; CLIFF extracts from it the location tuple (*United States*, *Florida*, *Miami*), and the input description contains "Florida"; hence, CLIFF's output is accepted.

D.2 Twitch Descriptions. To process Twitch descriptions, Tero uses CLIFF [13], Xponents [57], and Mordecai [18]. CLIFF is a tool that extracts locations from news articles, but is in principle applicable to any unstructured text. Xponents [57] is a library for general-purpose extraction of locations from text. Mordecai is similar to Xponents, however, it may output multiple results without indicating which one is likelier, making it hard to use on its own.

Tero processes each Twitch description as follows: (1) It passes the description as input to CLIFF, Xponents, and Mordecai. (2) It filters CLIFF's and Xponents' output using the conservative filter described above (§D.1). (3) For output that does not pass the previous step, if at least two of the three tools yield the same output, that output is accepted. Otherwise, (4) if a tool yields an output that subsumes that of another tool, the more complete output of the two is accepted.

We also leverage Twitch tags as a source of location. Until Feb 2023, Twitch defined a set of standardized tags[20] that streamers could use to inform viewers about their streams; among these tags, there were country-level tags included. Starting March 2022 until Twitch stopped supporting them on Feb, 28th, 2023, we gathered information about all streams on Twitch every 30 minutes and keep all streams that contain country-level tag information. We leverage these tags to determine the location of users by looking at how stable the tags are over time: for each user, we check all their streams in the selected time range and group all uninterrupted appearances of the same country-level tag. In total, we obtain country tags for 1,065,877 users out of a total of 14,079,347 (7.57%). We use the tag

[20] https://help.twitch.tv/s/article/guide-to-tags?language=en_US

information to recover some of the results discarded by the geocoding systems: we accept values even when our heuristic decides to discard them if there is tag information confirming that the country was correctly geocoded. Using Twitch tags allows us to increase the recovery rate of the geocoding systems by 9.41% while keeping the error rate under 4%.

D.3 Twitter Profiles. Extracting location information from a Twitter profile is easier (relative to a Twitch description) because the former includes an explicit location field; however, the location field itself is unstructured—allows free-style text—and this is why we still need geoparsing. We researched the publicly available geoparsing tools, but did not find any that clearly outperformed the rest (i.e., did better in extracting correct location information in the face of name coincidences, typos, and irrelevant text); in the end, we decided to combine the two most popular tools: Nominatim[21] and GeoNames[22].

Tero processes each Twitter profile as follows: (1) It extracts the content of the location field. (2) It passes the latter as input to both Nominatim and GeoNames. (3) It compares the two outputs; if they agree, or one subsumes the other, the more complete output is accepted; otherwise, Tero processes the content of the location field exactly as it processes a Twitch description. This last step works well when the location field contains non-geographic references (e.g. "Your heart, Chicago").

E Image Processing: All the Steps

The image-processing module operates in four steps:

(1) **Pre-processing** prepares the input thumbnail for Optical Character Recognition (OCR): (a) It crops around the area where the corresponding game typically displays latency. (b) It performs a set of standard tasks that render OCR more effective: converts the image to black-and-white, up-scales, applies a Gaussian filter to blur the edges and reduce noise, applies thresholding to separate foreground and background, and runs several iterations of dilating and eroding the image in order to merge disjoint regions [40, 54].

(2) **OCR** extracts characters from the pre-processed thumbnail. It uses three OCR engines: Tesseract[23], EasyOCR[24], and PaddleOCR[25].

(3) **Cleanup** filters and reconciles the OCR output: (a) It processes the output of each OCR engine separately and filters out characters that are not part of the latency measurement. (b) It compares the (post-filtering) output of the three OCR engines: if at least two of them agree, and their output is not 0, and their output has up to 3 digits, their output is accepted; if exactly two of them agree, the third engine's output is kept as an alternative; if none agree, their output is considered ambiguous and sent for "reprocessing." Side note: Some games show latency zero while the game is being prepared, or the player is waiting for their next match. This value is a placeholder, so it makes sense to discard it (which is what Tero does).

[21] https://nominatim.org/
[22] http://www.geonames.org/
[23] https://github.com/tesseract-ocr/tesseract
[24] https://github.com/JaidedAI/EasyOCR
[25] https://github.com/PaddlePaddle/PaddleOCR

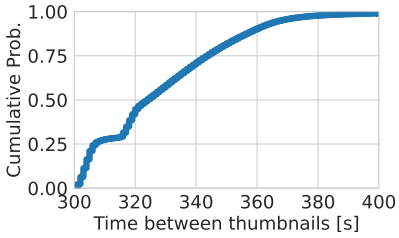


Figure 13: CDF of thumbnail inter-arrival time.

(4) **Reprocessing** repeats the OCR and cleanup steps but without the pre-processing. If the outcome is again ambiguous, the input thumbnail is discarded.

F Shared Anomalies: Statistical Test

For each {*location*, *game*} tuple, we estimate the probability that the location experiences a spike (shared or not) as

$$P_e = \frac{\#spikes}{\#measurements}, \tag{1}$$

where *#measurements* is the total number of latency measurements from *location* and *game* across all time. To ensure that our data is statistically significant, we consider only locations that fulfill the following condition [41]):

$$\#measurements \times P_e \times (1 - P_e) \geq 10. \tag{2}$$

Then, for each identified spike E: (1) We consider all the streamers with the same {*location*, *game*} tuple as the streamer who experienced spike E. (2) From these, we determine the number of streamers N who were streaming during spike E. We say that a streamer was "streaming during a spike" if we collected at least one latency measurement in the 12-minute window around the spike. The value of 12 minutes comes from the fact that the 90th percentile of time lapse between two consecutive thumbnails is 6 minutes (Fig. 13). (3) From these N streamers, we determine the number of streamers D who experienced a spike in the 12-minute window around spike E. (4) We compute the probability that D out of N streamers experienced a spike independently:

$$Pr\,[D\ spikes] = \binom{N}{D} P_e (1 - P_e)^{N-D} \tag{3}$$

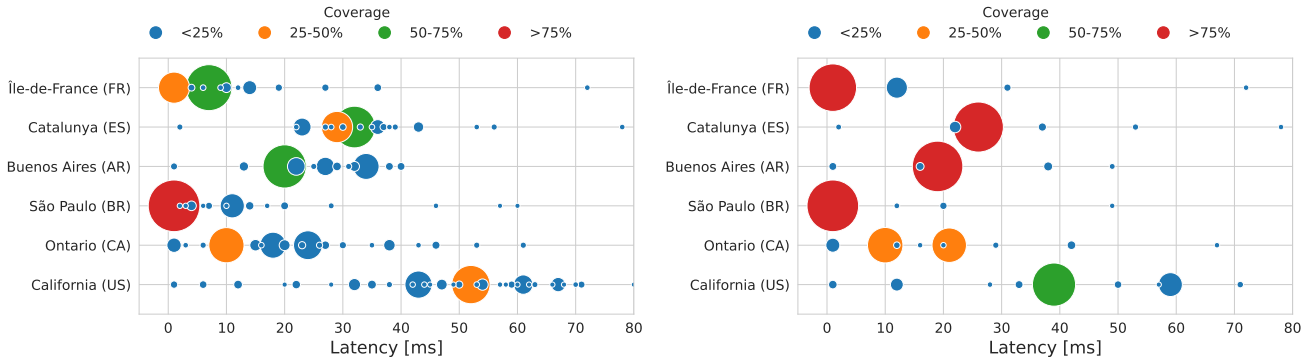
We consider the spikes to be part of the same shared anomaly if $P_D \leq 0.01\%$.

G Examples of Latency Clusters

The examples are in Fig. 14. We indicate with color and size the percentage of streamers out of the total in the location that are included in each cluster; factor indicates the minimum distance between points in order to be clustered separately; by default, it is x1 *LatGap*.

H Estimation of Error Rates

H.1 Location Errors. (1) To assess the geocoding error rate, we considered all streamers active between Nov 11th, 2022 and Mar 10th, 2023 (a pool of 9,472,272 unique users) and extracted those with a description (a total of 4,796,609 users) to process with the three geocoding systems. We randomly selected 500 results from



(a) Factor $\times 0.5$*LatGap*. **(b) Factor** $\times 1.5$*LatGap*.

Figure 14: Examples of latency clusters for different geographical locations and merging thresholds. Both size and color indicate the percentage of streamers inside each cluster. Each cluster is placed in the middle point between its border values.

each geocoding system and manually check the output; we repeated this process 3 times, and we report averages across the 3 experiments. An output is considered correct if the location extracted would be the same as the one extracted by a human; in the case of Mordecai, as it outputs more than one result, the complete output is considered correct if it contains at least one correct location. We repeat the same experiment with the output of each system after our heuristics (referred to as "SYSTEM++"). Table 3 summarizes the results, showing the extraction (proportion of the input accepted) and error rates. We note that all three systems have error rates over 20%. Finally, we repeated the experiment, selecting 500 users 3 times, but only considering the users accepted at the end of the process ("Twitch Comb."), obtaining an error rate of 3.47 ± 0.27%.

(2) To assess the Twitch/Twitter mapping algorithm error rate, we randomly selected 500 results from the pool of the 722 thousand streamers located by Tero and manually checked the mapping; we repeated this process 3 times. We see that Tero mapped to an incorrect Twitter profile 1.6 ± 0.33% of the time.

(3) We repeated the experiment from Step (2), evaluating the location from Twitter field extraction, first using each geoparsing system (Nominatim and GeoNames), and then combining their outputs. Finally, we evaluated the whole process, from mapping to location extraction, obtaining that the method ("Twitter-Twitch e2e") yields an incorrect location 1.91 ± 0.29% of the time.

(4) We combine all streamers located by both previous methods and repeat the experiment from Step (2). We obtain that Tero incorrectly located a streamer 1.46 ± 0.06% of the time.

H.2 Image-Processing Errors. We start by checking the accuracy of the chosen OCR engines: we randomly select 1,500 thumbnails with latency and manually check the output from each OCR engine; we repeat this three times and report averages. We report two metrics: (1) missing values when the system fails to extract any measurement when there is one available; (2) the system extracted a measurement different in any way from the latency shown in the thumbnail. Tesseract missed measurements from images and incorrectly extracted values from 15.52 ± 0.57% and 8.77 ± 0.26%, EasyOCR from 5.67% ± 0.38% and 8.31 ± 0.73%, and PaddleOCR from 5.84% ± 0.33% and 9.96 ± 0.75%, respectively.

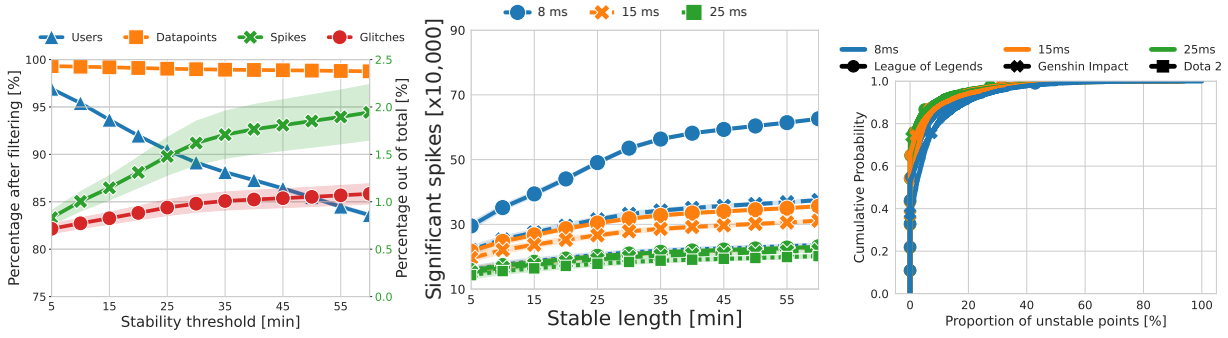
Then, we randomly selected 10,000 thumbnails processed by the image-processing module and manually checked the Tero's output before the data cleaning step; we repeated this three times, and we report averages across the three experiments: 34.97% ± 0.05% of the considered thumbnails contained a latency measurement; Tero failed to extract any measurement from 28.37% ± 0.47% of these, and it extracted an incorrect measurement from 3.7% ± 0.4%. In most cases (68.42%) where the output is incorrect, it consists of a single-digit latency measurement; this happens when the other digit(s) are blocked by on-screen elements (e.g., pointer or menus). Then, we repeated the same experiment a second time only considering images from which Tero extracted latency values, selecting 10,000 such thumbnails 3 times. Unsurprisingly, the error rate is similar at 3.3% ± 0.05%.

H.3 Data-Analysis Errors. To evaluate the performance of the glitch detection system, we use the wrong results obtained from the previous experiments. We see that 74.57% of the wrong results were identified by our method; we manually inspect the remaining glitches and plot the distribution of detected and undetected glitches in Fig. 5b. We note that more than 50% of undetected glitches are those that represent a digit confusion that only slightly changes the value (e.g. 101 is misread as 107) and hence go unnoticed by our method, as it relies on extreme changes to detect glitched values.

To study our false positive rate (correct values mislabeled as glitches), we consider 1,000 glitches and manually check their value against the labeling given by Tero: a glitch is correctly labeled if the latency value determined by Tero was incorrect. We repeat the experiment 3 times and report the average. Out of the glitches selected, 32.9% ± 1.05 correspond to zero-value points correctly detected; considering the remaining not-zero glitches, we obtain that Tero incorrectly labels as glitches correct values 25.87% ± 0.67 of the time.

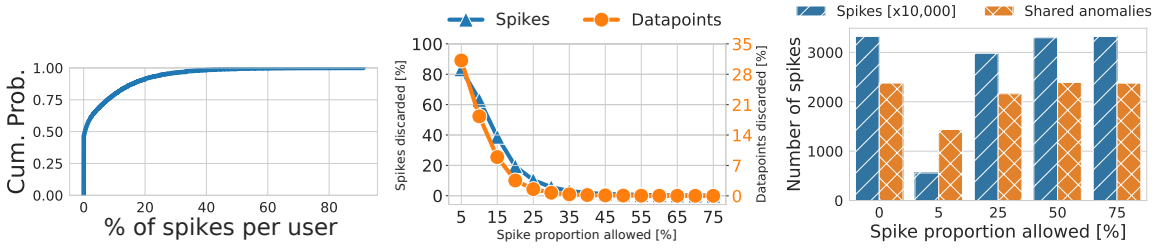
I Sensitivity to Parameters

We measured how *LatGap* and *StableLen* (§3.3) affect the results of the data cleaning step. We compared the results obtained with *LatGap* of 8, 15, and 25 ms and varying the *StableLen* from (the minimum possible) 5 to 60 minutes.



(a) (b) (c)

Figure 15: (a)(Left) Percentage of users and data points remaining after filtering depending on *StableLen*. (Right) Percentage out of the total of points of spikes and glitches detected depending on *StableLen*. (b) Number of significant spikes detected depending on *StableLen* and *LatGap*. (c) Distribution of the time users spend in an unstable (not spike) sequence depending on *LatGap*.



(a) (b) (c)

Figure 16: (a) Distribution of the spike proportion (spikes over total not-glitched points) per user. (b) (Left) Proportion of spikes discarded depending on *MaxSpikes*. (Right) Proportion of points discarded depending on *MaxSpikes*. (c) Number of spikes and shared anomalies detected depending on *MaxSpikes*: all points and spikes from users with more spikes than allowed are discarded as glitches.

We start by looking at the effect of *StableLen*. *StableLen* affects two aspects of the technique: how many users we discard during pre-processing (see §3.3) and how many consecutive points can be considered spikes or glitches. Therefore, the choice of *StableLen* presents a trade-off: short values allow us to keep more users at the expense of not detecting some spikes (or, more importantly, glitches); on the other hand, longer *StableLen* will both unnecessarily discard normal points and flag as spikes increases in latency that are normal parts of gameplay. Fig. 15a show this trade-off for League of Legends: on the left, we show the percentage of users and data points after filtering users without a stable sequence, and, on the right, the proportion of points considered spikes and glitches. As we expected, both spikes and glitches increase with *StableLen*, with spikes having a more pronounced increase. We also see that the number of users discarded quickly increases with *StableLen*, without the number of data points discarded increasing proportionally; this implies that increasing *StableLen* discards users with a low number of data points, heavily affecting the coverage of our data-set.

Then, to decide the value of *StableLen* to use, we look at the spikes detected. We say that a spike is *"significant"* (for a given threshold) if it represents an increase of at least the threshold with respect to the mean latency of the stream it is contained in. In Fig. 15b we look at the effect of *StableLen* on the number of significant spikes using three threshold values (8, 15, and 25) for two games, separated by the *LatGap* value used. We find that the number of significant spikes grows quickly for lower *StableLen*, slowing down around 25 minutes. This fact, in combination with the information that the average match length is between 25 and 35 minutes [35, 39], lead us to use 30 minutes as the *StableLen* for all games.

Then, we study the effect of varying *LatGap*. We study how *LatGap* affects, for each user, the proportion of points that are contained in "unstable" sequences that are not discarded as glitches or marked as spikes: This metric shows how often we label as unstable sequences that are likely to be normal. Fig. 15c shows the CDF of the proportion of unstable sequences per user (for three games) depending on *LatGap*; we note that for most games studied, as long as *LatGap* exceeds 15ms, the proportion of stable points is almost independent of *LatGap*.

We found that the processing time is almost independent of parameters; this makes sense given that the data-analysis module must process each latency measurement anyway, independently of parameter values.

J Standard Anomaly Detection

We compare Tero's detection algorithm to standard anomaly-detection techniques. To choose the best anomaly-detection techniques, we relied on a 2022 review [49]; due to the difficulty of obtaining ground truth to train a model with, we limited ourselves to unsupervised techniques. We selected one technique from each of the three main categories of unsupervised anomaly-detection techniques: Local Outlier Factor (LOF) from distance-based methods [4], Isolation Forests (iForests) from isolation-based methods [29], and Minimum Covariance Determinant (MCD) from distribution-based methods [45].

LOF is a distance-based method similar to K-Nearest Neighbours that compares the density (one over the average distance to the K neighbors) of a given point with the density of its K neighbors; LOF's results are highly sensitive to the value K: in practice, K controls the number of neighbors that need to be similar to a point to consider it normal. MCD estimates a Gaussian distribution over the data, but it can also be applied to unimodal, symmetric distributions. To detect anomalies, MCD assumes that the contamination factor (the proportion of anomalous points present on the data-set) is known beforehand; in practice, we simply try different values of contamination in the range [0.01, 0.5]. iForests is an isolation-based technique that works by detecting which points are farthest away from the rest of the data by iteratively partitioning the points at random. The method uses a score that measures how many partitions are required to completely isolate a point from the rest: the more separated from the rest of the data a point is, the fewer partitions are required to isolate it. As is the case with MCD, iForests requires a contamination factor; the authors of the original paper [29] suggest a methodology to automatically determine a threshold for the contamination, but in practice, we see that a threshold leads to many "false anomalies". To solve this issue we use a simple statistical rule: we only consider as real anomalies points with scores that are outliers, determined using the inter-quartile range for outlier detection; we vary K, the parameter that defines the decision range, from 0.5 to 2.0.

We evaluate the performance of the state-of-the-art techniques against our QoE-based technique, we look at the overlaps between the spikes and glitches detected by the QoE-based method and the state-of-the-art, using the same concept of "significance" (a spike/glitch is significant if it is an increase/decrease of at least a *threshold* from the mean of the stream the point belongs to); as anomaly detection has no intrinsic concept of spikes or glitches, we simply divide all anomalies across the mean. To make the comparison as fair as possible, we still consider spike (or glitch) points "fixed" by using alternative values (see §3.3.2).

Fig. 18 (Fig. 17) shows the percentage of significant spikes (glitches) separated by technique and classified into three categories: found by the technique and the QoE-based technique, found exclusively by anomaly detection, and found exclusively by QoE-based; the error bands show the difference due to each technique's parameter. We present the results obtained using a threshold of 15ms; results with other thresholds are virtually identical, only showing a difference of around 5% on the mean for all 3 techniques.

In the case of spikes, we note that all three techniques find spikes that the QoE-based technique misses (at most 20% for iForests); the rest of the time, close to 70%, our technique is as good or better than the state of the art. The results for glitches are reversed: in 2 out of 3 cases, anomaly detection marks as glitches more than 50% more glitches than our QoE-based technique. First, we examine the spikes not detected by our QoE-based technique and find that, for the spikes, between 28% (in the case of LOF) and 91% (MCD) of them, correspond to consecutive points with increased latency; these types of points are most likely *location changes* (see §3.3.3) and should not be considered as spikes. The remaining missed spikes are isolated points; to assess why the QoE-based method missed them, we measure the distance between the potential spikes and their direct neighbors: we see that for all techniques, 70% of the time, the distance is less than 1 QoE band, making the spike not "significant enough" for the QoE-based technique to detect. We repeat

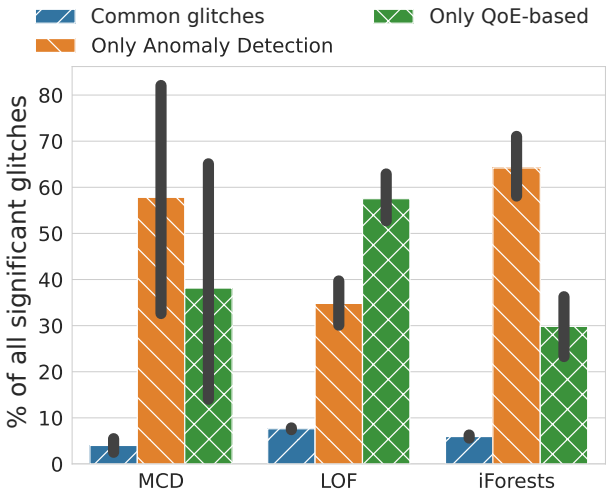


Figure 17: Percentage of significant *glitches* detected by anomaly detection techniques compared with our QoE-based technique.

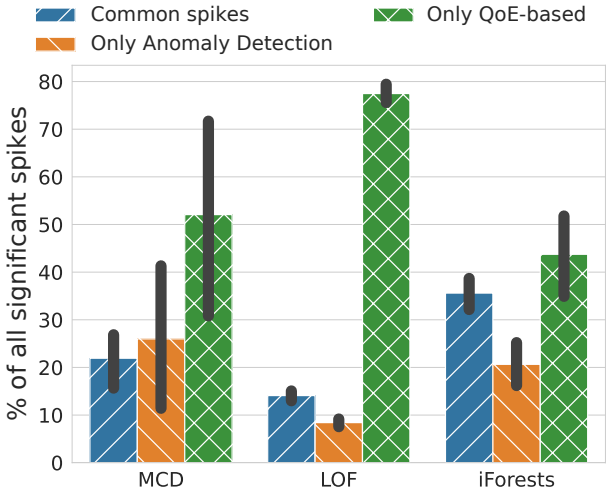


Figure 18: Percentage of significant *spikes* detected by anomaly detection techniques compared with our QoE-based technique.

the analysis for the undetected glitches and reach similar conclusions: between 23% (LOF) and 71% (MCD) of the missed glitches are consecutive points, making them likely location changes, and the 80% of the isolated points are separated from their direct neighbors by less than *LatGap*, a drop that is not significant enough for the QoE-based technique to flag as a glitch.

We explain the difference between the QoE-based technique and the state-of-the-art by making three observations: First, the state-of-the-art has no concept of "significant" changes, labeling a point as a spike even if it is just slightly different from its neighbors (e.g. a sequence of several 25ms points with one 27ms point will lead to that point being considered a spike); second, the anomaly detection techniques have no intrinsic mechanism allowing them to distinguish server/location changes from anomalies. Third, we do not have a way of knowing beforehand the number of anomalies per user, which makes choosing appropriate parameters for anomaly detection a challenging task as the same user will have a different proportion of anomalies over time.