Snatch: Online Streaming Analytics at the Network Edge

Yunming Xiao[1], Yibo Zhao[2], Sen Lin[1], Aleksandar Kuzmanovic[1]
[1] Northwestern University, [2] Boston University
{yunming.xiao,sen.lin,akuzma}@northwestern.edu,yibozhao@bu.edu

Abstract

In recent years, we have witnessed a growing trend of content hyper-giants deploying server infrastructure and services close to end-users, in "eyeball" networks. Still, one of the services that remained largely unaffected by this trend is online streaming analytics. This is despite the fact that most of the "big data" is received in real time and is most valuable at the time of arrival. The inability to process requests at the network edge is caused by a common setting where user profiles, necessary for analytics, are stored deep in the data center back-ends. This setting also carries privacy concerns as such user profiles are individually identifiable, yet the users are almost blind to what data is associated with their identities and how the data is analyzed. In this paper, we revise this arrangement, and plant encrypted *semantic cookies* at the user end. Without altering any of the existing protocols, this enables capturing and analytically pre-processing user requests soon after they are generated, at edge ISPs or content providers' off-nets. In addition, it ensures user anonymity perseverance during the analytics. We design and implement *Snatch*, a QUIC-based streaming analytics prototype, and demonstrate that it speeds up user analytics by up to 200x, and by 10-30x in the common case.

CCS Concepts: • **Networks** → **Network services**; **Public Internet**; **Cross-layer protocols**; • **Security and privacy**;

Keywords: Online Streaming Analytics; Network Edge

ACM Reference Format:
Yunming Xiao[1], Yibo Zhao[2], Sen Lin[1], Aleksandar Kuzmanovic[1]. 2024. Snatch: Online Streaming Analytics at the Network Edge. In *European Conference on Computer Systems (EuroSys '24), April 22–25, 2024, Athens, Greece.* ACM, New York, NY, USA, 21 pages. https://doi.org/10.1145/3627703.3629577

EuroSys '24, April 22–25, 2024, Athens, Greece

ACM ISBN 979-8-4007-0437-6/24/04…$15.00
https://doi.org/10.1145/3627703.3629577

1 Introduction

The ability to extract user analytics in a timely manner, *i.e.*, as quickly as possible, is of critical importance for numerous online applications [68]. An ad provider can more promptly adjust its ad layout to capture more clicks based on the user analytics extracted over short time scales. Many online services are utilizing machine learning systems to "learn on the fly" and either adjust content presentation (*e.g.*, return search results tailored towards a given user profile) or optimize system performance. Still, such machine learning systems fundamentally depend on analytics "triggers," which again, if available sooner or over short timescales, are more valuable.

Currently, the streaming analytics "machinery" typically resides in data centers. On the one hand, the analytics servers are fed by streams from web server clusters, which typically serve tens of thousands of clicks arriving on average every second. On the other hand, given that user web requests alone are *semantic-oblivious*, *i.e.*, carrying no direct information about users, such information first needs to be obtained from associated user-profile databases. The analytics servers thus aggregate data streams from the web servers and user databases to provide advanced analytics.

This approach, however, suffers from two main drawbacks. The first drawback comes from the trend of infrastructure migration towards the network edge. In particular, content and service providers have been continuously pushing their systems and content to the users, from the content delivery networks (CDNs) to the off-nets – servers outside their own autonomous systems (ASes) – which have become a common approach to expanding the footprint of content hypergiants [60]. Nevertheless, the semantic-oblivious requests cannot be analyzed before they reach the data centers that are distant from these edge systems/contents.

The second drawback is disrespect for user privacy, which has raised increasing attention and concerns in recent years [34, 48, 95]. More concretely, the semantic-oblivious requests, while simple in design and hence commonly adopted, carry individually identifiable information, *e.g.,* user IDs. The user IDs have allowed the service providers to record any information about the individual users as much as they can for an indefinite duration as long as the users do not actively clean up – and most users are not aware of it at all.

In this paper, we explore the potential of catching and preprocessing user clicks early, much sooner than when they reach the data centers while preserving user privacy to the

largest extent. In particular, we look at the content providers' network and off-nets, as well as edge ISPs. Our goal is to design a system to make early click catching, in-network processing, and anonymity preserving analytics possible, and to quantify the achievable performance benefits.

To enable this approach, we propose *semantic cookies*, encrypted data structures set by the server and then kept at the user. Contrary to widely-used state-of-the-art HTTP cookies, which are effectively pointers to semantic user databases (typically hosted at data center back-ends), we plant semantic user information that is not individually identifiable directly into the cookies themselves. This enables collaborating edge components, mostly edge servers but also switches, to analytically process the user requests. Importantly, semantic cookies can be seamlessly deployed without altering any of the existing protocols.

We design and implement Snatch, the first prototype of our edge-network analytics system. We explore two designs. The first one places semantic cookies at the application layer, HTTPS, and processes them at the off-net's or CDNs' edge servers. The second one places semantic cookies at the transport layer, QUIC, which enables processing them at ISP switches. The underlying trade-off is that application-layer semantic cookies provide high flexibility in terms of the number of user features, while transport-layer cookies provide faster analytics. Luckily, both types of cookies could be utilized simultaneously, when needed.

Snatch is a two-tier analytics system. The first tier consists of either edge servers, which handle application-layer semantic cookies, or LarkSwitches, which handle transport-layer semantic cookies. The second tier consists of AggSwitches, which inspects all the incoming packets to the analytics server. The first-tier devices early re-direct semantic data to the state-of-the-art analytics servers. Optionally, the two tiers coordinate to enable in-network analytics. The number of supported operations available at switches is considerable (see Appendix C); hence, it provides valuable in-network analytics support. Snatch augments existing analytics systems in a fully cooperative manner.

We implement Snatch and evaluate it in a testbed. To fully understand the performance gains that Snatch can achieve on the Internet, we conduct a large-scale measurement study. In particular, Snatch involves several components: the edge server, ISP switch, web server, and analytics server. To study the performance of these entities in practice, we host HTTPS websites using AWS EC2 instances. In addition, we purchase CDN services from Cloudflare and AWS CloudFront. Finally, we utilize over 2,000 residential nodes from the Mysterium VPN, spread around the world, as users. These measurements enable us to accurately estimate network latencies among users, edge ISPs, off-nets and CDNs, and data centers, and evaluate performance gains achievable by Snatch.

We find that Snatch brings significant speed-ups, particularly in scenarios when all calculations can be done in the network. Specifically, in-network analytics reduces latency by 5x relative to the scenario when only redirections are enabled. Processing semantic cookies at the transport layer is 3-8x faster than at the application layer. When users are spread across the world, Snatch manages to speed up user analytics by 10x compared to existing analytics systems, while the speedup climbs to 30x when users are located on a single continent, *e.g.*, North America.

2 Background And Motivation

2.1 Streaming Analytics

Data streaming analytics targets enormous data that arrive continuously in time. Efficient data streaming analytics is essential to many important real-time applications, *e.g.,* social networks [44], ad campaigns [46], and beyond [50]. Early streaming analytics systems use dataflow models [32, 42, 43]. With the increasing demand for streaming analytics, the last decade has witnessed a thriving of proposals: MillWheel [30], Storm [91], Heron [70], Puma [44], Spark Streaming [11, 31, 102], Apache Flink [8], and more. Among them, Spark Streaming [102] started to aggregate the streaming data over a short interval and perform batch analytics in a Map-Reduce fashion [51]. The state-of-the-art streaming analytics produce results at a timescale of ∼1 to ∼10 seconds [21, 46, 102].

The above-mentioned work focuses on streaming analytics in a single cluster environment, leaving the *arrival of data* out of scope. In this paper, we make the arrival of data a central topic of our research. For example, message queues [9, 10, 15, 23] are usually adopted in real-world production to link the data ingestion pipeline and the streaming analytics systems [46].We include the message queues when discussing the streaming analytics systems in this paper.

Yet, the message queues and the streaming analytics systems do not depict the whole picture. While some applications analyze only internal data, *i.e.,* stored or generated *inside* the data center, many applications analyze data from outside the data center, *e.g.,* the users' requests, generated from end-user networks scattered around the world. Further, in online applications scenarios, the application-level streaming data is typically sent to the analytics servers only after it reaches web server endpoints in data centers. Hence, rather significant latency can be added to the user requests after they are generated by end users.

The time cost incurred before data arrives at the analytics server is nontrivial (see § 2.3), however, it is often disregarded. To depict a comprehensive picture, we consider an entire online streaming analytics *cycle*. The cycle includes the streaming data generation and transmission, *i.e.,* users send requests to the servers and the servers process the requests, data processing, *i.e.,* by message queues and event processors [21]. Finally, the cycle terminates with a traditionally-defined streaming analytics system, *e.g.,* Spark Streaming [11].

2.2 Anonymity Preserving Analytics

Anonymity preserving analytics refers to computational analytics over aggregated results from the users without revealing the individual identities, and hence provides strong privacy guarantees [54, 100]. Unfortunately, widely-adopted Web cookies present a significant privacy-leaking vertical, even at the network level [95]. A single identity leak in one application opens up unforeseen tracking opportunities.

With the growing public attention and concerns about individual privacy, anonymity preserving analytics has been supported by legislators [2]. The related studies have also become a hot topic in the security and privacy academic community [48, 56]. Complying with the trend, hyper-giants have also introduced their own data collection and analytics systems that preserve user anonymity, for instance, Google [34, 57], Apple [90], Microsoft [52], and more.

2.3 Opportunities

Migrating infrastructure towards the edge. Content and service providers are continuously pushing their systems closer to the users. Content delivery networks (CDNs) allow the content providers to place static content at servers nearby users, and thus improve their experience. CDNs have thus become one of the most crucial components of the Internet today, serving billions of users across the world. In fact, more than half of the Internet traffic originates from several top CDN providers [60]. In addition to building their own data centers and backbone networks [63, 69], the major content providers also deployed off-net servers [60]. Such servers are placed in the eyeball, end-user, networks. The deployment of these edge servers further reduces the latency between the user and the content, thereby improving the user experience.

In parallel with this trend, the service providers are also pushing computation closer to the users. Hence, many distributed streaming analytics systems are proposed, aiming at working with limited resources available at the edge [40, 41, 59, 79, 83]. While helpful in certain specialized scenarios, most applications still require *centralized* streaming analytics with data from users scattered all around the globe. In this paper, we focus on centralized streaming analytics.

Importantly, with the introduction of edge servers, the overall architecture of online streaming analytics systems has changed. The streaming analytics server (cluster) is usually not placed in the same region as the edge servers, which is where the users are directed first. This results in complications of security and privacy issues, *e.g.,* a third entity has access to the cookies or sensitive content [74, 97], as well as a rather substantial increment of delay between the components of the online streaming analytics systems, as we will demonstrate below.

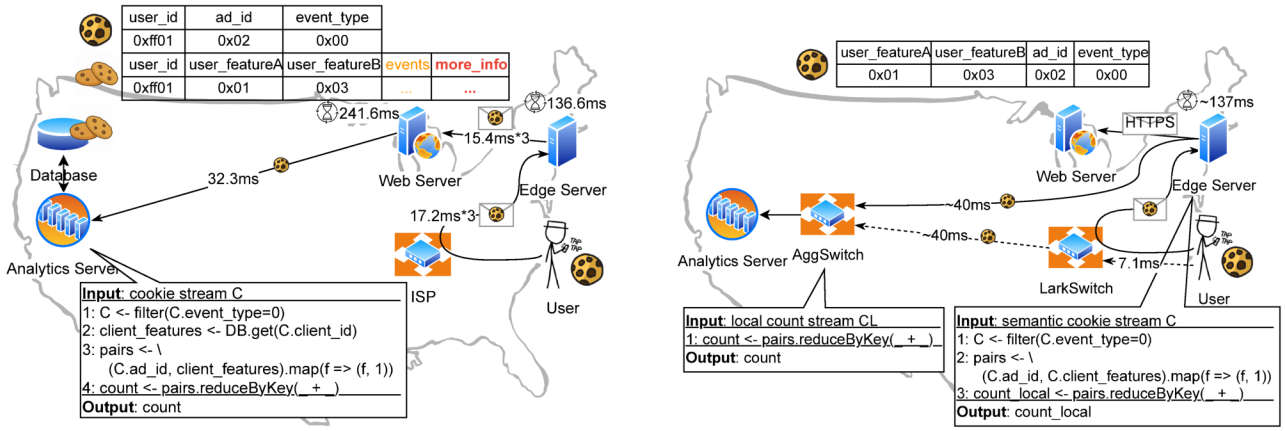
Case Study. The above infrastructure migration to the edge affects many online applications. A first example is that an advertisement provider may want to receive aggregated results of its ongoing advertisement campaign in real-time to make decisions, *e.g.,* the offering in the following advertisement auctions, based on them [46]. A second example is that real-time crowd analytics, a technique crucial to many businesses [27], needs to aggregate results about information in a particular region. A third example is the needs for faster response to users' resource demands. Today, cloud platforms have become the go-to solutions for many companies because of their capability to scale up/down in a timely manner. Nevertheless, the service scaling (where containers are usually used) needs to deploy before they become available. Hence, faster response to the demand and hence earlier provoking service deployment changes are crucial to the user experience for various online applications [1].

Below, we analyze the first example of the advertisement campaign in detail. Here, the data is generated when a user clicks on an ad link. It follows that a request is sent to an edge server, *e.g.,* in the case of a CDN, with the user ID embedded in the HTTPS cookie and the ad ID included in the HTTPS URL. The edge server then passes the cookie to the web server in the closest data center. Next, the web server processes the cookie and delivers the data to the (centralized) analytics system which is potentially at another data center. Message queues are usually adopted to deliver the data. If the user semantic information is needed, *e.g.,* demographic or other information, the analytics server needs to first fetch this data from a database before being able to perform further analytics operations.

In this example, we assume that the application developer, who owns the web server, has control over all the cookies, meaning that they are all first-party cookies. As a result, the cookies are initially sent to the web server, and any ad broker either resides at the analytics server or receives information from it. It is important to note that in the current Web, this assumption may not hold true, as users may send separate requests to ad broker URLs along with third-party cookies. However, the use of these third-party cookies contradicts the prevailing trend of enhancing privacy and has already been banned by some major browsers [4, 5, 28] and is expected to be banned by the remaining browsers in the near future [3]. Consequently, we focus on first-party cookies in this study.

Drawbacks and Opportunities. We conduct a large-scale measurement study to comprehensively quantify the latency inflation cost (details are provided in § 5). Figure 1(a) illustrates an example of the analytics time cost breakdown of one data point. It starts from the request generated by the user in New York[1]. The closest edge server, which caches static content, is selected and is in New York. The web server that provides dynamic content and handles cookies is hosted at

[1]Here we assume QUIC is adopted as it is becoming popular, and its handshake is simpler than TCP+TLS. With TCP+TLS handshakes, the time cost of communication in Figure 1(a) will be more than doubled as it is now.



(a) Scenario without semantic cookies.

(b) Scenario with semantic cookies.

Figure 1. Breakdown of time cost in a simple application of advertisement campaign.

AWS's *us-east-1* region. The server for global streaming analytics is, however, located in California. All QUIC connection handshakes take 97.8 ms in total. Adding up to the processing time costs at both the edge and web servers, which take 378.2 ms in total, as well as the delay of 32.3 ms from the web server to the analytics server[2], the time cost before the cookie arrives at the analytics server is 508.3 ms, which occupies more than 50% of the total time cost assuming 500 ms is needed for the analytics itself[3].

Besides the latency inflation highlighted above, we also notice that in the current online streaming analytics systems, the analytics operations are performed only after they arrive at the analytics server. For instance, traversing through the path of the data in Figure 1(a), we find that the data is held by the edge server and web server for more than 300 ms in total while they respectively handle the static and dynamic web request – unrelated operations to the data analytics – and is left untouched before it arrives at the analytics server.

We now move to privacy issues. In short, the above request submits a user activity, *i.e.,* clicking an ad, along with the user ID to both the edge server controlled by a CDN provider, and the analytics server controlled by the ad provider. After that, the ad provider can perform any analytics as it wishes, or save this event associated with the user ID in its database for further analysis. Yet, this might lead to potentially serious privacy violations. This is because as long as the user does not clean up the cookie, all her activities will be logged among potentially other individual information that is obtained through other sources, as illustrated by the tables in Figure 1(a). An attacker, *e.g.,* a malicious data owner or a third-party attacker who gets access to the database of the ad provider, or network traffic, might then be able to impose danger to such individual users by splicing all the information pieces [95].

[2]We assume no handshake is needed because a persistent connection is established by the message queues, or otherwise the time cost will be greater.
[3]The default computing time interval of Spark is 1s [25]. If the data arrives evenly, the average time of analytics is 500 ms.

3 System Design

In this section, we first present the overview of our system design and illustrate the benefits using the same example as in Section 2.3. Next, we present our security and privacy threat model. We then present more design details and benefit quantification of the semantic cookie as well as in-network streaming analytics. Last, we touch on the functionalities of Snatch's controller.

3.1 Overview

At a high level, we propose to forward and (pre-)process the data much earlier than the current online streaming analytics systems do – at the edge server, or even at the ISP switch, thanks to the programmable data plane [35]. One critical obstacle for the early data forwarding and preprocessing is that the cookies are semantic-oblivious, *i.e.,* no information about the user but only a reference to the information is included. This is because at the time when the cookies are assigned, the server knows nothing about users. We instead propose *semantic cookies*. Like regular application-level cookies, they are generated by servers, and kept by users. The difference is that semantic cookies hold encrypted application-level user data, and more importantly, include no individually identifiable information. Typically, once the information about the user is collected, *e.g.,* when a user clicks a specific web page, the web server should push semantic information into the user cookie itself. To the best of our knowledge, we are the first to seize this opportunity, given the nature of our system.

We explain the procedure of Snatch with Figure 1(b), focusing on the same application as the case study in § 2. A previous benchmark study [46] evaluated the streaming analytics engines by operating a join operation to obtain the

number of users who viewed the ads. But in practice, more advanced analysis of the composition of the users may be needed to allow the ad providers to make decisions based on the results. Thus, we assume that the analytics server wants to analyze the composition (by their demographic categories) of users who viewed a particular ad in an instant windowed time. This can be achieved with three operations as shown in Figure 1(a): (*i*) filtering the arriving cookie streams by event type, *e.g.,* a user viewed an ad (L1); (*ii*) then requesting the database for the user features (demographic information) by the user ID embedded in the cookies (L2); and (*iii*) counting the number of users for every user feature (L3-4).

In Snatch, the web servers should set the semantic cookies as a replacement of the user ID, as shown in Figure 1(b), after the first connection with the user and the information of the user becomes available. It is noteworthy that the first connection cannot be accelerated and is not depicted in Figure 1; all the results we present in this paper focus on subsequent connections after the initial one. The semantic cookies should be kept by the user, similar to the current design. What is different is that the ad provider should not store any user information. From then on, the user sends requests with the semantic cookies. The semantic cookies can be recognized and processed by the edge server. As shown in Figure 1(b), the edge server filters the cookies by the event type (right L1). It also counts locally the number of users who viewed a particular ad for every user feature (right L2-3). The processed data can be forwarded directly to the analytics server. Before it arrives at the analytics server, a programmable switch close to the analytics server named AggSwitch aggregates the local counts from all the edge servers (left L1) before delivering it to the analytics server.

As there are many devices and parties involved in Snatch, a controller (not shown in Figure 1) is present to coordinate all the participants. As shown in Figure 2, Snatch controller is run by a trusted party. It accepts analytics tasks from application developers and distributes the associated instructions to different devices held by different parties.

In this example, all the analytics have been completed on the way to the analytics server while no user ID is present. The time costs on analytics operations (~500 ms) are reduced to <1 ms given that (*i*) each web server only handles a small number of requests and hence has minimal costs and (*ii*) the line-rate processing ability of the programmable switches. It follows that the total latency from when the data is generated to when the decision can be made based on the data is reduced by ~80% from 1008.3 ms to 228.6 ms. This demonstrates the feasibility and benefits of processing the data early.

Moreover, the semantic cookies in many scenarios are constant. For instance, in the second example of real-time crowd analytics, what needs to be aggregated and analyzed is the user's information, *e.g.,* demographic or interests; in the third example of faster response to users' resource demands, what needs to be aggregated and analyzed is the typical demand of the users. These information can be kept at the user's side and sent without knowing what the user's requests are. Thus, we further propose to encode encrypted semantic cookies in the transport layer. With the programmable switch's capability to read and parse packet headers, the semantic transport-layer cookies can be acted upon as soon as the user requests reach the edge ISPs, as the dashed lines in Figure 1(b) illustrate. In particular, semantic cookies could (optionally) be pre-processed, and forwarded by the LarkSwitch, as shown in the figure. This further cuts analytics latency to around 48 ms – a ~95% reduction in the total delay.

3.2 Threat Model

We assume a third-party attacker who can monitor and collect network packets from a limited geolocation range. We also assume an attacker who may join the system as a user to receive the semantic cookies from the web servers. The attacker may try to decode the format of either the application-layer or transport-layer semantic cookies by examining across the collected packets. Nevertheless, the attacker should be computationally bounded and not be capable of decrypting ciphertexts that are encrypted using advanced cryptography algorithms, such as AES and TLS.

In general, being able to decode the semantic cookie would allow the third-party attacker to intercept user information from network eavesdropping, or send fake data to distort the application developers' analytics results.

Moreover, we assume an honest-but-curious edge node, *i.e.,* edge server or LarkSwitch in Figure 1(b), who follows the protocol but may try to understand the application-layer purposes of the semantic cookies, and hence steal the user information for commercial purposes. On the other hand, we assume a malicious application developer who may try to insert individually identifiable information into semantic cookies while using our system.

3.3 Semantic Cookie

Contrary to "traditional" cookies, which are used as pointers to a back-end database of user attributes, semantic cookies enable web servers to directly encode user attributes and push them to the end-users. The semantic cookies cannot be in plaintext but need to be encoded and encrypted because they will be stored at the users' side.

Application-Layer Semantic Cookie. Because the edge server is the endpoint of the users' TLS connections, it has the access to all the application-level information in the users' requests, including the application-level cookies as required by our system. For instance, if users are sending an HTTPS request, then the edge server is able to access the headers, cookies, and payload of the HTTPS request.

Therefore, leveraging edge servers for early forwarding the application-layer cookies is straightforward to implement. Most current edge services, *e.g.*, Cloudflare's CDN, allow the user to set custom page rules to adjust caching levels, forward requests, modify headers, *etc* [12, 14]. What is needed in Snatch is to decrypt the cookies, match semantic cookies' names and values, and send the extracted data to a custom destination (analytics server) – if possible – in a custom format. The additional computational cost is minimal as it is similar to existing header-related operations.

The benefits of application-layer cookies are three-fold: First, it can support semantic cookies with as many sub-cookies (user features) as needed by the applications. Second, it does not require any modification on the user's side. Third, it is fully compatible with the current HTTPS request design and simply needs to include semantic cookies. In addition, the cookies can be easily kept across different connections between the user and the server over time, regardless of the underlying protocol, *e.g.*, TCP, UDP, QUIC, TLS, *etc.*

To better quantify the benefits of using semantic versus non-semantic application-layer cookies, we aim to quantify the speedup. It is defined as the ratio of the expected latency in two scenarios, *i.e.*, non-semantic vs semantic. Hence, speedup is ≥ 1. Denote user by C, edge server by E, web server by W, and analytics server by A. Then, d_{CE} is the delay between the user and the edge server, and so on. Let T_{trans} be the transmission duration of the request.

We further denote by T_E, T_W, and T_A the time costs for processing requests at the edge server, web server (including database communication), and analytics server (including message queues), respectively. Then, for HTTPS request on top of QUIC 1-RTT, the speedup is

$$S_{app-https} = \frac{3d_{CE} + 3d_{EW} + d_{WA} + T_{trans} + T_E + T_W + T_A}{3d_{CE} + d_{EA} + T'_E + T'_A}, \quad (1)$$

where T'_E and T'_A are the time costs when Snatch is involved. Because of the minimal additional cost from processing application-layer cookies at the edge server, we consider $T'_E = T_E$. Further, coefficient 3 in the equation comes from the QUIC 1-RTT handshake process. More analysis on QUIC 0-RTT and TCP protocols is available in Appendix B.1.

Transport-Layer Semantic Cookie. Transport-layer cookies are semantic cookies that are encrypted and encoded in the transport-layer protocol. As identified in a previous study [33], cookies can be encoded in three protocols without requiring any modifications on the users' machines: IPv6, TCP, and QUIC. In this paper we only consider QUIC. Discussions for IPv6 and TCP and more details for QUIC are provided in Appendix B.2.

Snatch fully utilizes the features of QUIC. We consider all the connections between a user and an edge server except the first one – at least one connection is needed before semantic cookies are available. If the user uses QUIC 0-RTT, she repeats the connection ID from the last connection where transport-layer cookies are encoded. LarkSwitch then will be able to decode the transport-layer cookies and forward them to the analytics server. This requires no modification on the user's side. If the user uses QUIC 1-RTT, a slight modification of the code in userspace is needed to allow the QUIC 1-RTT to keep the transport-layer cookie in the new connection, *i.e.,* QUIC should remember the connection ID from last connection but re-generate a subset of the bits without tweaking the transport-layer cookies. In summary, both QUIC 0-RTT and 1-RTT fit our vision and work for Snatch.

We further quantify the benefits of transport-layer semantic cookies. Let I denote ISP. Hence d_{CI} is the delay from user to ISP. Similar to the analysis for application-layer cookies, the speedup of the streaming analytics for QUIC 0-RTT is

$$S_{trans-0rtt} = \frac{d_{CE} + d_{EW} + d_{WA} + T_{trans} + T_E + T_W + T_A}{d_{CI} + d_{IA} + T'_A}. \quad (2)$$

For QUIC 1-RTT, its handshake needs 1 RTT and hence the coefficients for d_{CE} and d_{EW} become 3, while the denominator keeps the same as the transport-layer cookie is included in the first packet header. The speedup is

$$S_{trans-1rtt} = \frac{3d_{CE} + 3d_{EW} + d_{WA} + T_{trans} + T_E + T_W + T_A}{d_{CI} + d_{IA} + T'_A}. \quad (3)$$

3.4 In-Network Streaming Analytics

Snatch further seizes the opportunity to accelerate streaming analytics by leveraging the in-network computation: the programmable switch performs computation at line rate, much faster than the servers [82]. For transport-layer cookies, streaming analytics can be completed in the data plane – via the cooperation of LarkSwitches and the AggSwitch. The LarkSwitch decodes the transport-layer cookies, pre-processes the data, and send them to the analytics server. On the last hop to the analytics server, an AggSwitch extracts and aggregates the data from all LarkSwitches. Note that for application-layer cookies, the analytics can be done in the network as well. It only requires the edge server to forward the application-level data in a format agreed in advance, which allows AggSwitch to decode and aggregate the data.

The modern programmable switch is able to perform AES encryption/decryption [45] and calculate most of the common statistics [65, 76]. We limit the pre-processing to the supported operations, and leave more complex ones to the analytics servers. When all the operations of a target analysis are supported by the switches, Snatch reduces all the time costs of Pub/Sub services and the analytics process.

We consider two types of forwarding schemes: per-packet and periodical forwarding. Per-packet forwarding satisfies the needs of applications that require very low latency and immediate knowledge of the streaming data. When all the operations of a target analysis are supported, Snatch provides a huge speedup, *i.e.,* $T'_A < 1$ ms because the programmable

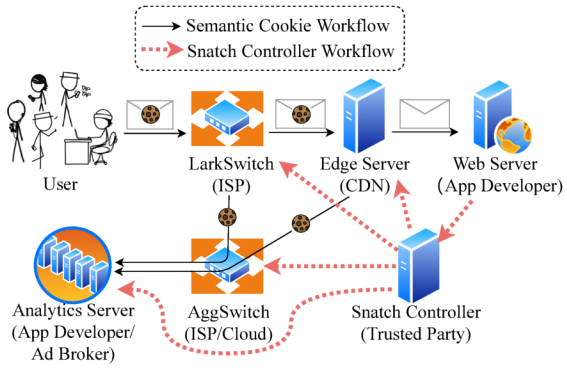


Figure 2. Snatch controller workflow.

switch operates at line rate. On the other hand, periodical forwarding targets applications that have slightly loose requirements on latency. During each period, the programmable switches updates the statistics based on incoming packets. By the end of each period, LarkSwitch and AggSwitch cooperate to calculate statistics and forward them to the analytics server. Compared to per-packet forwarding, periodical forwarding saves bandwidth resources while sacrificing latency. We explore this trade-off experimentally in § 5.2.

3.5 Controller

Snatch includes many components that spread across the current Internet infrastructure. It is not practical for any single party to possess or control all Snatch components. Instead, Snatch should leverage the existing Internet infrastructure to the largest extent and builds on top of it. To realize that, Snatch needs a controller to coordinate all the other system components. Snatch controller should be run by a trusted party that builds up commercial relationships with the application developers, edge ISPs, content providers, and optionally cloud providers. As illustrated in Figure 2, the application developer submits the analytics tasks to the Snatch controller, which then parses the tasks and distributes the instructions to different devices controlled by different parties, including the LarkSwitch by ISPs, edge servers by content providers, AggSwitch by ISP or cloud providers, and analytics server by the application developer or ad broker. It also returns the format of processed cookies to the analytics server which is controlled by the application developer.

At a high level, Snatch controller provides the following APIs to the application developers: (1) Add or remove applications. In our design, the system supports multiple applications at the same time. Different applications are distinguished by *application ID* in the cookies (either at transport or application layer). (2) Add or remove cookies. For each application, Snatch supports multiple user features, or sub-cookies. A transport-layer cookie is preferred if there is enough space. When the space is scarce for all the sub-cookies, the developer should decide which sub-cookies are encoded at the application layer based on the needs. (3) Change feature type and valid ranges. Snatch supports two types of data: *class* and *number*. For a different feature type, Snatch supports different pre-processing functions. Any data that is not in the valid feature range will be aborted. (4) Change the forwarding scheme, either per-packet or periodically.

3.6 Security and Privacy

Snatch provides security and privacy guarantees. Following our threat model, this guarantee needs to hold against three potential attackers as detailed below.

Malicious Third-Party Attackers. First, we need to protect the cookies from being understood or tempered by third-party attackers who monitor and collect network packets. To achieve this, we propose to encrypt the transport-layer semantic cookies with AES-128 (see § 4), and use HTTPS when accessing the Web protecting the application-layer semantic cookies with secure communication. In this way, third-party attackers cannot decrypt or learn the format or the content of the semantic cookies. The AES encryption keys should be set differently in different regions and changed regularly to strengthen security protection.

Honest-But-Curious Edge. Next, we need to prevent the edge nodes from being able to understand the application-layer purposes of the semantic cookies – unfortunately, they can do so now. To achieve that, the app developer should avoid using semantic names and, if possible, add transformations to the values, *e.g.*, performing reversible mathematical operations before pushing semantic cookies to the users and recovering them after receiving aggregated results from Snatch. This process renders plaintext semantic cookies semantically incomprehensible. Further, app developers can set multiple correlated cookies to substantially raise the bar for interpreting them. For example, they can set two cookies to represent the same purpose, but each time only update either one of them, hence confusing the edge nodes.

More importantly, full protection can be achieved using differential privacy (DP) [55], which secures data privacy by introducing noise to it. A naive example is that if the app developer intends to increase a cookie value by 1, they could instead increase it by 2 with a probability of 75% and decrease it by 2 with a probability of 25%. Such noises can also be applied to all fields – including those that are not undergoing actual changes – as well as their initial values, to better disguise the sensitive data changes from specific user groups or operations. Consequently, while individual data may not be entirely precise, the aggregated analytical results such as calculated statistics from all users remain accurate.

Note that the actual DP model will be adaptive and more complex than the above example. Striking the right balance between accuracy and the level of added noise is crucial when implementing DP. But deriving specific DP models is beyond the scope of this study as they depend on the

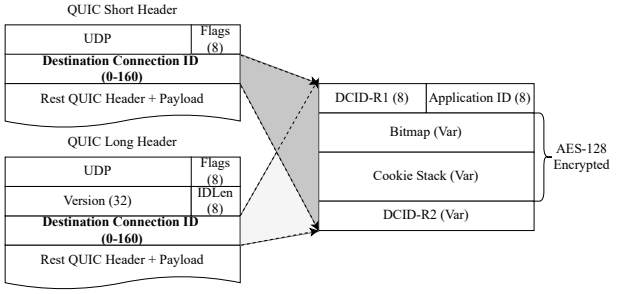


Figure 3. Transport-layer cookie design (QUIC).

specific user data range and analytics accuracy requirements following business models.

In general, Snatch may require the app developer to redesign cookies mostly because of the discarding of individually identifiable information. Further, the app developer should leverage cookie encodings, correlation, and potentially DP, and maybe employ multiple edge providers to prevent them from learning the semantic cookies. Beyond cookie redesigns, it does not rely on any particular constraints on app developers or ad brokers.

Malicious Application Developer. The last concern is that it is possible for the application developer to include individually identifiable information in a non-semantic cookie, *i.e.,* not processed by Snatch, during the communication between the web server and the users. This is prohibited by Snatch's policy and penalties will be applied once discovered. We leave the technical enforcement of excluding individually identifiable information in this scenario as future work. It is noteworthy that while such technical enforcement is not included in this paper, Snatch has made it possible to regulate the usage of individual identifiers in the cookies by providing an alternative system that works well – or even better – without such identifiers.

4 Implementation

4.1 Cookies and Programmable Switch

We implemented a prototype of LarkSwitch and AggSwitch based on an Intel Tofino switch. We first present the cookie and packet design. Then, we introduce the switch logic. Further, we introduce our implementation and discuss the scope of analytics with programmable switches.

Transport Cookie Design. We choose QUIC protocol as the carrier of transport-layer cookies because it fully meets the requirement of Snatch (see § 3.3). We encode the transport-layer cookies in the up-to-160-bit `connection-ID` field of QUIC headers. As shown in Figure 3, we split the connection-ID into four parts: (1) 8-bit destination connection ID (`DCID`), (2) 8-bit `application-ID`, (3) `bitmap` of variable length, and (4) `cookie-stack` of variable length. DCID is randomly generated for connection identification. The application-ID is used for distinguishing from normal QUIC packets and specifying the format of the remaining bits. Because of the limited space, the format of bitmap and cookie-stack are not fixed but application-dependent. Assuming there are N sub-cookies used by an application, corresponding to N features, then the bitmap has N bits where each bit denotes whether this sub-cookie is present. The cookie-stack includes N sub-cookies and the length of each sub-cookie is pre-defined by the controller. N is bounded by the memory and stage limitation of the switch. The remaining bits (if any), unoccupied by the bitmap and cookie-stack, marked as DCID-R2 in the figure, are also randomly generated for connection identification.

Next, we create a custom packet header on top of UDP to carry early-forwarded cookies or pre-processed data (either by LarkSwitch or edge server) for AggSwitch. For more details please refer to Appendix B.3.

To prevent the cookies and data from being hacked or tempered by the users or attackers, the transport-layer cookies after application-ID is encrypted using AES-128. The AES-128 key is only known to the application developer and the edge nodes, *i.e.,* edge server or LarkSwitch/AggSwitch. It is noteworthy that encrypting or decrypting the up-to-160-bit transport-layer semantic cookies using AES-128 only adds ~0.1 ms delay with a modern Tofino switch [45].

Switch Logic. When a new application is registered at a LarkSwitch or AggSwitch, its parameters – including the application-ID, the format of bitmap and cookie-stack, and the AES key – are stored in the switches' match-action table entries. LarkSwitch will try to match the application-ID for all the incoming QUIC packets. When a packet is matched, the switch decrypts and decodes the available cookies/data following the parameters of the corresponding application. the switch then performs counting or other statistical operations on the decoded cookies/data. For per-packet forwarding cookies or for periodical forwarding cookies when the period ends, the switch creates a new custom packet (see appendix B.3) and sends it and associated statistics to the analytics server. To do so, we make the switch `clone` the original packet. The original packet is still forwarded to the web server to keep the original communication. Meanwhile, the cloned packet header will be rewritten and its payload will be removed before being sent to the analytics server.

Statistics Calculation. Both the LarkSwitch and the AggSwitch involve statistics calculation when they process the cookies and data. The match-action pipeline design makes programmable switches naturally classifiers and counters. In our prototype, we have implemented the basic statistics which satisfy the (partial) needs of most streaming applications. For data type *class*, we implement counting by matching value. For data type *number*, we implement sum, min, max, and average calculations.

We further discuss the scope of applications supported by Snatch's in-network streaming analytics. P4 switches support most of the streaming analytics operations. A detailed example is provided in Appendix C where we explore the P4

switches' support for Spark Streaming APIs. One limitation is that complex operands, *e.g.,* modulo and logarithm, are not supported by most P4 devices. Nevertheless, this can be resolved by using FPGA-based devices [93], redesigning the algorithms [101], or using P4's `digest` to complete the operations with the help of the control plane [20]. Further, machine learning algorithms or their pre-processing can also be completed in the programmable data plane [87, 98].

To sum up, despite the limitations, programmable switches' ability to process data at a high speed and low power cost is a great asset to boost up the performance of Snatch.

4.2 Clients and Servers

We target minimal client modification. For QUIC 0-RTT, the client does not need any modification. For QUIC 1-RTT, a minor change in userspace is needed so that the transport-layer cookies from the last connection are stored and repeated in the next connection, while the rest of the connection ID is randomly re-generated. We implement a Snatch client based on quic-go [6]. We realize the transport-layer cookie support for QUIC 1-RTT by modifying only <50 lines of code. We further implement the Snatch-enabled edge- and web-server, also based on the quic-go repository.

4.3 Controller

Functionality. Snatch controller takes inputs from the application developers. It then generates a random byte as the application ID and a random AES-128 key for semantic cookie encryption. Then, it updates the components in the following order: AggSwitch, LarkSwitches, and the edge servers. With corresponding programs pre-installed at all the rest components, Snatch controller only needs to update the parameters, *e.g.,* altering the table entries in LarkSwitches and AggSwitches so they can recognize new applications and send results to new destinations, through RPCs to the corresponding control plane. The update frequency is overall low, *e.g.,* days or weeks, because updates only happen when new applications are added or AES keys need to be updated.

Consistency. When a controller updates an application, inconsistency issues might arise because of the delay between the controller and other components. For instance, some edge servers might change the format of transport-layer cookies before a LarkSwitch, or a LarkSwitch changes the recognition of the cookie-stack before changes are made. They may result in missing or incorrect results being reported.

We solve the inconsistency issue by adopting a version control scheme. When an update instruction is received by the controller, it generates a new application version with a new application identifier, *i.e.,* the same application has different application IDs for different versions. It then updates the components in order: AggSwitch, LarkSwitches, and the edge servers. After a period of time (possibly days), the controller deletes the old application ID and associated rules, *i.e.,* revokes the corresponding rules on the AggSwitch and LarkSwitches. In this way, Snatch ensures that consistency is preserved when updating the applications.

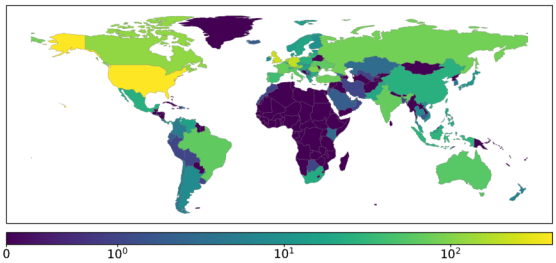


Figure 4. Overview of measurement sites.

5 Evaluation

In this section, we first present results from our global measurement study on understanding the performance of data streaming from normal Internet users. We then simulate and evaluate the benefits of our approach with our testbed that simulates real-world environments.

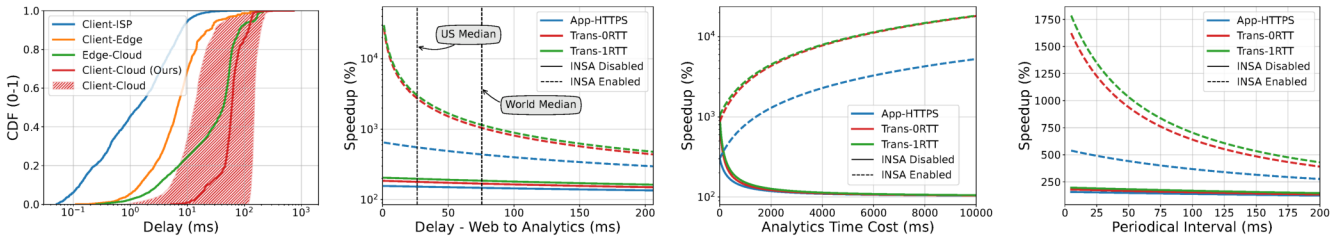
5.1 Measurement and Estimation

Methodology. Snatch involves multiple Internet components: the ISP switch, edge server, web server, and analytics server. To study the performance of these subjects in practice, we set up experiments as follows. First, we host HTTPS websites using AWS EC2 instances [7], which represent the web servers in Figure 1(b). Then, we purchase CDN services from Cloudflare [12] and AWS Cloud Front [14]. This allows us to set up the edge servers on a global scale.

Next, we need to measure the performance of regular Internet users on a global scale. We choose the decentralized VPNs (dVPNs), which have gained much popularity recently [96], as our means of measurement over academic measurement platforms [47, 53, 86, 89] for better flexibility. In dVPNs, regular Internet users from all over the world monetize their spare bandwidth by hosting a VPN proxy at their homes, providing a VPN service to the public. Thus, it provides a desirable measurement platform for our study. We select Mysterium [18] among various dVPNs since it has the largest footprint [96] – Mysterium currently holds over 5,000 dVPN nodes (proxies), among which over 2,000 are recognized as "residential," *i.e.,* hosted in regular Internet users' home networks.

We iteratively connect to all the available residential dVPN nodes as measurement sites. During each dVPN connection, we send out various packets and derive delays between Snatch components. More details on the delay derivations are provided in Appendix D.1. For all the per-site operations, we iterate 10 times and take the median for further analysis to avoid outliers resulting from unstable network conditions.

Measurement Results. We conduct our measurement over 14 days, during which we tested 2,253 sites (dVPN nodes)



(a) Inter-component delays.

(b) Speedup of per-packet forwarding as a function of d_{WA}.

(c) Speedup of per-packet forwarding as a function of T_A.

(d) Speedup of periodical forwarding as a function of interval.

Figure 5. Measurement and speedup results.

around the world. Figure 4 shows the per-country site counts. Among 87 countries we have investigated, the US has the most sites, followed by the UK and Germany. It is noteworthy that while the measured sites are not representative of billions of Internet users, they allow us to capture a glimpse of the current global WAN practice and provide a meaningful basis to estimate the potential benefits of Snatch. Also, the number of sites is not entirely proportional to the total number of Internet users per country, but they represent the user engagement to a large extent. Thus, we utilize such collected statistics to evaluate Snatch.

Figure 5(a) shows the delays between client, ISP, edge (server), and cloud (web server and analytics server), respectively. The delay from client to ISP is the smallest as expected, with a median of 1.4 ms. The delay from the client to the edge is slightly larger, with a median of 6.7 ms. This shows the success of CDN services as a means to improve Internet performance, and also Snatch's potentials from the semantic cookie early forwarding. For each site, we take the minimal delay from the off-net servers, Cloudflare CDN, and AWS Cloud Front. Appendix D.2 provides additional details.

Next, we look at the cloud performance. Figure 5(a) shows that the delays from client to the cloud (dashed red area) vary a lot – from 13.1 ms to 150.3 ms in the median – depending on the relative geolocation. Further, the median delays from the client and from the edge to our hosted EC2 machines are 60.1 ms and 43.6 ms (red and green lines), respectively. Note that the sum delays from the client to the edge and from the edge to the cloud are not always equal to the delay from the client to the cloud, because of the complex routing policies across ASes which may not assign the same path [39].

Moreover, we find that the intra-data center delays for AWS range from 0.8 ms to 4.4 ms, whereas the inter-data center delays range from 4.7 ms to 206 ms, with a median of 75.5 ms. Additional details are provided in Appendix D.3.

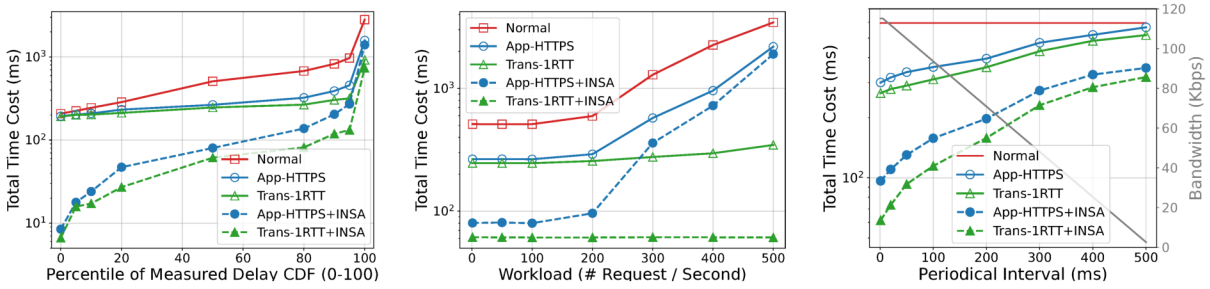
Quantifying Snatch Benefits. Here, we estimate the speedup that Snatch brings. In particular, we utilize the speedup Equations for different protocols, *i.e.,* (1), (2), and (3), combined with the above measurement results. If not otherwise indicated, we estimate based on medians: 1.4 ms for delay between client and ISP (d_{CI}), 6.7 ms for delay between client and edge server (d_{CE}), 43.6 ms for delay between edge and web server (d_{EW}), 0.8 ms for transmission time cost (T_{trans}), 136.6 ms for time cost at the edge (T_E), 241.6 ms for time cost at the web server (T_W), and 500 ms for time cost at the analytics server (T_A) – assuming default Spark parameters [25].

We first investigate the expected Snatch speedup as a function of the median delay between the web server and the analytics server, (d_{WA}). We adopt the "best practice" assumption: the client will always choose the closest edge and web servers. We further assume that *d_{CA} and d_{EA} grows proportionally as the delay d_{WA}, within their own range respectively.* Appendix D.2 provides details.

Figure 5(b) shows Snatch's speedup as a function of the delay from the web server to the analytics server (d_{WA}). The solid line represents the case when only early forwarding is enabled ($T'_A = T_A = 500$ ms) whereas the dashed line represents when in-network streaming analytics (INSA for abbreviation) is also enabled ($T'_A = 1$ ms). The figure shows that enabling Snatch's INSA feature improves the performance by a great margin, by up to two orders of magnitude, versus when INSA is disabled. Looking at various protocols, we see that the scenarios where Snatch benefits the most to least are Trans-1RTT, Trans-0RTT, and APP-HTTPS. This is expected as transport-layer cookies provide better performance than application-layer cookies.

Figure 5(b) further shows that as d_{WA} increases, hence d_{CA} and d_{EA} increase following best practice assumption, the Snatch benefits necessarily decrease. Indeed, the more distributed the users are, the network latency more significantly affects Snatch's performance. Next, we focus on two scenarios: (1) US, where end-users are located in the US and the median inter-data-center delay is 26.3 ms, and (2) worldwide, where users are dispersed around the world and the median inter-data-center delay is 75.5 ms (see Figure 9(a)). Figure 5(b) shows that QUIC 1-RTT INSA speedup is 31x in US and 12x worldwide, while App-HTTPS INSA speedup is 5.5x in US and 4.4x worldwide.

Figure 5(c) shows the speedup as a function of analytics time cost, T_A. In practice, the analytics time cost depends on many factors, including the analytics algorithms, workload, the Pub/Sub queuing delays, the settings of traditionally



(a) Delays. (b) Workload. (c) Periodical interval.

Figure 6. Testbed evaluations. Total time cost as functions of (a) delays, (b) workload, and (c) periodical interval.

defined analytics systems, *etc.* The time cost thus ranges from negligible to ~10 seconds at the hyper-giants [21]. Here we consider general tasks and hence vary T_A from 1 ms to 10s. When T_A is negligible, INSA naturally does not play an important role. But as T_A grows, the speedups diverge: they decrease when INSA is disabled but increase when INSA is enabled. Overall, Snatch always boosts up the performance of streaming analytics. For reference, when T_A is 10s, and INSA is enabled, the speedup for Trans-1RTT is 183x, for Trans-0RTT is 181x, and for App-HTTPS is 53x.

Figure 5(d) shows the speedup in the case of periodical forwarding, as a function of the period (interval) ranging from 5 ms to 200 ms. When the interval is 5 ms, the speedup is naturally closer to per-packet forwarding. As expected, the speedup decreases when the interval increases because the data takes more time to the analytics server. For reference, for interval of 5 ms, the speedup for Trans-1RTT is 18x, while for interval of 200 ms, the speedup for Trans-1RTT is 4.3x.

5.2 Testbed Experiments

Environment Setup. We set up a testbed consisting of 6 host machines and one Tofino programmable switch. Among them, three host machines represent the client (request generator), the edge server, and the web server, respectively. The analytics server is represented by a cluster of three machines, which consist of two slave nodes and one master node. The Tofino switch represents both LarkSwitch and AggSwitch. The topology follows Figure 2 where the Tofino switch connects to the client, the master node of the analytics server, and the edge server (with two different ports connecting to two different network interfaces, respectively). Each host machine is equipped with an 8-core AMD EPYC CPU with 16GB RAM. The delays between the machines are controlled via Linux `Traffic Control` module [26].

We adopt QUIC 1-RTT in our evaluation below. Once the analytics server (master node) receives an aggregation packet, meaning that INSA is enabled, it will log the timestamp; otherwise, it will submit the data to Spark Streaming which processes the data and logs the finishing timestamp.

We select the advertisement campaign analytics as the target application. Different from Yahoo Streaming Benchmark [46], which correlates the user ID and the ad campaign ID, we go further to count the user demographic information (we randomly generate gender, age, and geolocation for each user) for each ad campaign. In addition, we set the interval of Spark Streaming to be 150 ms as it is suitable for most of our tasks and environment, *i.e.*, it minimizes the time cost.

Performance Evaluation. We first evaluate the impact of delays between the components for per-packet forwarding. We adopt different delays—taking Nth percentile of delays from Figure 5(a)—in our testbed and perform 10,000 requests from the client for each experiment. We send 10 requests per second (RPS), a relatively low rate, to exclude the impact of workloads (which we explore later in the text). We adopt the same "best practice" assumption as in § 5.1.

Figure 6(a) shows the total time costs given different delay percentiles in our measurement. The total time costs are measured from when clients send requests until the results are obtained, either from Spark Streaming (solid lines and hollow markers) or from AggSwitch if INSA is enabled (dashed lines and filled markers). Overall, the results show that the total time cost increases as the delay percentile increases, *i.e.,* the clients experience worse Internet infrastructure. Still, Snatch is beneficial at all times. The shortest to longest total time cost are between Trans-1RTT and App-HTTPS with INSA, and between Trans-1RTT and App-HTTPS without INSA. One exception occurs at the 100th delay percentile where the performance of Trans-1RTT without INSA exceeds App-HTTPS with INSA because d_{CE} drastically increases.

In terms of speedups, APP-HTTPS and Trans-1RTT reduce the time cost at most by a factor of 2.1x and 5.4x without INSA (at 95th delay percentile), or by 24.5x and 31.2x with INSA (at 1st delay percentile). When INSA is enabled, the speedups slowly decrease as the delay percentile increase, *i.e.,* the clients experience worse Internet infrastructure. Yet, the speedup of Trans-1RTT with INSA is at least 3.8x at the 100th delay percentile, which brings the total time from 2,807 ms down to 735 ms. In the median case, the speedups for APP-HTTPS and Trans-1RTT are 1.9x and 2.0x without INSA, or

6.3x and 8.3x with INSA. Compared to Figure 5(b), Trans-1RTT under-performs the results from § 5.1, yet App-HTTPS over-performs the corresponding results. This is because the processing time costs at the edge server and at the analytics server in our testbed are both smaller than in § 5.1.

Next, we evaluate the impact of the workload. We take the median delays from the measurement, and adjust the workload, which we quantify as the number of requests that the clients send per second. We consider per-packet forwarding here because it consumes more bandwidth and is thus more sensitive to workload compared to periodical forwarding. Figure 6(b) shows that the total time costs are stable with the same rank as in Figure 6(a) when the workload is relatively low (<100). Later, *i.e.,* when workload >100, the total time costs increase as the workload increases for all scenarios except Trans-1RTT with INSA, demonstrating the power of in-network transport-layer switch-based processing. When the workload is equal to or greater than 300, the time costs for no-Snatch and App-HTTPS start to increase sharply (note that the y-axis of Figure 6(b) is in log scale). Likewise, App-HTTPS with INSA is less effective than Trans-1RTT without INSA. This suggests that congestion happens at the edge server and the web servers because they are overwhelmed by the high request rate.

Meanwhile, however, Trans-1RTT with INSA keeps a very stable performance – it takes 61 ms regardless of the workload. This reveals a property of Snatch: *no parallelism inflation*. The stable performance is expected because of the nature of line-rate processing of programmable switches and the design of Snatch: Trans-1RTT skips all the computation on the edge and web servers (and the analytics server if INSA is enabled) where congestion may happen at a high workload. In fact, Trans-1RTT and Trans-0RTT are able to keep the best performance as long as the throughput does not exceed the capacity of the switches, which is over 10 Tbps [16].

Finally, we evaluate the periodical forwarding. We adopt the median delays and a workload of 200 RPS. Figure 6(c) shows that as the periodical interval increases, the total time cost increases while the bandwidth consumption (grey line) between LarkSwitch/the edge server and AggSwitch decreases. Nevertheless, when the periodical interval is 500 ms, Trans-1RTT and App-HTTPS still speed up the total time cost by 1.2x and 1.1x without INSA, or 1.8x and 1.7x with INSA. The bandwidth consumption linearly decreases from ~112 Kbps to ~1 Kbps as the periodical interval increases from less than 5 ms to 500 ms.

6 Discussion

Semantic Cookie Related Issues. One question may raise on how the application developers can derive the semantic information without a user ID. In fact, we can regard the semantic cookie as a state machine: the developers have the state from the last request, update it based on the current request, and save it on the users' side for the next request.

Another issue may be the additional overhead from adopting semantic cookies. Transport-layer semantic cookies do not incur any overhead as an existing header field of QUIC with limited length is used. Application-layer semantic cookies inherit the current cookie design but *ideally* only discard the individual identifiers, which brings no overhead. Still, overhead may be introduced by the way that the developers design the application-layer semantic cookies. Currently, the developers build their own database and store as much user information as they want, *e.g.,* the complete visit history per user [92]. With the semantic cookies, the developers can only collect the visit history by appending the new visit to the semantic cookies every time the user visits the website. This will indeed bring non-trivial overhead. Nevertheless, while no hard restriction on the size of semantic cookies is applied, we argue that this is a feature rather than a defect: the semantic cookies are meant to prevent the developers from logging everything about the user, *e.g.,* complete visit history. Hence, it forces the developers to carefully re-design the cookies and only ask for the least; otherwise, they may lose customers because of bad experiences.

Alternative to Latency Inflation. One alternative to reduce latency inflation introduced in § 2.3 is to ask the users to send duplicate requests to both the web servers and the analytics servers. Yet, there are many drawbacks from this approach. First and most importantly, it does not enhance user privacy as Snatch does because individual identifiers are still present. Second, it cannot benefit from in-network computation, which may be a larger factor in performance improvement than latency inflation (see § 5). Third, it requires the users to double their bandwidth consumption and leads to a worse web experience, yet without offering any incentives to the users. In addition, exposing the analytics server to public may open the door to attacks.

View From Application Developers. With Snatch, application developers can benefit from faster online streaming analytics and hence obtain more valuable results. Meanwhile, they lose the freedom to store whatever they want from the users' activities and may fail to perform certain analytics, *e.g.,* individual profiling [92]. Nevertheless, more studies are looking into how to effectively perform anonymity-preserving analytics [34, 52, 90]. It is thus questionable how much the cost really is from discarding individual-level analytics. Moreover, developers may lose the freedom anyway as stricter privacy policies may be enforced given the public's rising privacy concerns. In addition, the developers can actually benefit from respecting user privacy: users who care about their privacy may be more inclined to websites that adopt semantic cookies compared to other competing websites. This may become an important incentive for more developers to adopt the semantic cookies, and (hopefully) eventually lead to widespread adoption of semantic cookies, similar to the history of HTTPS adoption.

Generality of Analytics. In our implementation, we pre-install programs at the edge devices and have them accept RPCs from Snatch controller to update certain parameters (§ 4.3). This would allow edge devices to recognize new applications and perform analytics accordingly. Yet, we acknowledge that our implementation only supports fixed types of aggregation analytics. While the edge servers should be able to conduct any streaming analytics, we have analyzed the capabilities as well as the limitations of the programmable switches (§ 4.1). In an ideal implementation, the controller should generate efficient and on-demand codes and push them to the edge devices. We leave this as future work.

Fault Tolerance. Snatch might fail due to various issues. For example, inconsistency might occur when the controller tries to update other components (see § 4.3). Other examples include failing to update AES keys at edge servers, or packet drops, *etc.* All these issues will result in the same outcome: the aggregated results become inaccurate. Fortunately, we can detect such failures by running the same analytics on data that is collected from the web servers and arrives at a later time. Application developers should report the result difference to the Snatch controller, which would then check and update the other components through RPCs. We leave the real-time detection and correction for future work.

In-Network Streaming Analytics Trade-offs. In the evaluation, we consider that either INSA is enabled or disabled. In practice, and for most real-world scenarios, the speedup is in between because of the complexity of queries. When more computation is offloaded to the network, the speedup is higher given the negligible time cost for the processing at the switches. Still, more computation also incurs more switch resources, *i.e.,* fewer applications can utilize the switches' support. Thus, there exists a trade-off for the ISPs: support more applications with a smaller speedup for each, or support fewer applications with a larger speedup for each. Independently, Snatch provides a considerable speedup compared to the state-of-the-art even when INSA is disabled.

7 Related Work

Streaming Analytics. In addition to streaming analytics systems discussed in § 2, JetStream [84] and AWStream [103] explore the wide-area streaming analytics whose data sources are widely distributed and propose to reduce the data rate to cope with the limited WAN bandwidth. In addition, Iridium [83] optimizes the data placement before the arrivals of queries. Sana [67] applies WAN-aware multi-query optimization. The wide-area streaming analytics assumes that the data is heading *directly* to the analytics server after it is generated. This is however different from our concerned scenarios where the data accompanies the user requests and thus makes a detour. Snatch removes this detour and enables in-network analytics via semantic cookies.

In-Network Computation. With the advent of programmable networking hardware and programming languages [35, 36, 88], researchers have proposed to leverage in-network computation to handle network management [73], caching [66], load balancers [78], deep neural network training [72, 87], *etc.* Ports *et al.* [82] summarizes a general guide of what and when to offload the computation to the network. While most work targets scenarios within data centers, Jagen targets ISP-centric defense [77]. Snatch aims to speed up online streaming analytics by leveraging the in-network computation and in cooperation with both the ISPs and the cloud.

Anonymity Perseverance. The anonymity perseverance research spans across different fields including social networks [58, 85], crowd-sourcing [64], recommendations [99], *etc.* One approach is to add structural noise to its data to report [54], and thus prevent the attackers from inspecting what each user actually sends while ensuring that the aggregated results are statistically correct. Another approach is using secure multi-party computation protocols, where a set of non-colluding servers privately perform computation over the user data [48]. However, MPC methods often come with significant overhead [38, 75, 80]. In general, the common challenges for all privacy-preserving analytics include a high cost and robustness towards malicious users and servers. A third approach is to make the users send data through an anonymizing network, *e.g.,* mix-net [37, 71] or Tor [62, 81], where the data and individual identities are decoupled. However, these methods incur a high cost [49, 94]. Our proposal instead prevents the user from sending individually identifiable information by design.

8 Conclusion

This paper presented Snatch, a system that early forwards and pre-processes the online streaming data at the network edge to speed up the online streaming analytics and preserve user anonymity. The key to enabling Snatch is the introduction of semantic cookies, which carry encrypted user information that is individually unidentifiable and directly available for analytics. We demonstrated that it is viable to encode semantic cookies in the existing application or transport protocols. Our evaluation of Snatch – based on real-world measurements – showed that when processing can be done early in-network, Snatch can speed up user analytics by 10-30x. Given the growing trend of migrating infrastructure towards the edge, such speedups along with privacy enhancements are likely to soon become a reality.

Acknowledgements

We would like to thank our shepherd Andrew Quinn and the anonymous reviewers for their insightful feedback and guidance. We also thank Chenkai Weng for his valuable discussion on privacy-related issues. This work has been funded by the NSF grant #2226107.

References

[1] Alibaba Cloud MMO Gaming Solution Architecture, 2018. https://www.alibabacloud.com/blog/alibaba-cloud-mmo-gaming-solution-architecture_593877.

[2] General Data Protection Regulation (GDPR), 2018. https://gdpr-info.eu/.

[3] Chromium Blog: Building a more private web: A path towards making third party cookies obsolete, 2020. https://blog.chromium.org/2020/01/building-more-private-web-path-towards.html.

[4] Full Third-Party Cookie Blocking and More, 2020. https://webkit.org/blog/10218/full-third-party-cookie-blocking-and-more/.

[5] How Do I Manage Cookies In Brave?, 2020. https://support.brave.com/hc/en-us/articles/360050634931-How-Do-I-Manage-Cookies-In-Brave-.

[6] A QUIC implementation in pure Go, 2021. https://github.com/lucas-clemente/quic-go.

[7] Amazon EC2, 2021. https://aws.amazon.com/ec2/.

[8] Apache Flink – Stateful Computations over Data Streams, 2021. https://flink.apache.org/.

[9] Apache Flume, 2021. https://flume.apache.org/.

[10] Apache Kafka, 2021. https://kafka.apache.org/.

[11] Apache Spark, 2021. https://spark.apache.org/.

[12] Cloudflare – The Web Performance & Security Company, 2021. https://www.cloudflare.com/.

[13] Cloudping – AWS Latency Monitoring, 2021. https://www.cloudping.co/grid.

[14] Content Delivery Network (CDN) - Amazon CloudFront, 2021. https://aws.amazon.com/cloudfront/.

[15] Google Cloud Dataflow, 2021. https://cloud.google.com/dataflow.

[16] Intel Tofino 2, 2021. https://www.intel.com/content/www/us/en/products/network-io/programmable-ethernet-switch/tofino-2-series.html.

[17] IP SLA Network Performance – Arelion, 2021. https://www.arelion.com/our-network/ip-sla-network-performance.html.

[18] Mysterium Network: Open Web Protocol, 2021. https://www.mysterium.network/.

[19] Our Global IP Network – NTT-GIN, 2021. https://www.arelion.com/our-network/ip-sla-network-performance.html.

[20] P416 Portable Switch Architecture (PSA) – Packet Digest, 2021. https://p4.org/p4-spec/docs/PSA.html#sec-packet-digest.

[21] Processing billions of events in real time at Twitter, 2021. https://blog.twitter.com/engineering/en_us/topics/infrastructure/2021/processing-billions-of-events-in-real-time-at-twitter-.

[22] pyspark.streaming.DStream – PySpark 3.2.0 Documentation, 2021. https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.streaming.DStream.html.

[23] RabbitMQ, 2021. https://www.rabbitmq.com/.

[24] SLA Performance For Global IP – Sprint, 2021. https://www.sprint.net/tools/sla-performance/sl.

[25] Spark Streaming Programming Guide, 2021. https://spark.apache.org/docs/latest/streaming-programming-guide.html.

[26] tc(8) — Linux manual page, 2021. https://man7.org/linux/man-pages/man8/tc.8.html.

[27] Mira – Real-Time Crowd Analysis, 2022. https://mira.co/solutions/real-time/.

[28] Firefox rolls out Total Cookie Protection by default to more users worldwide, 2023. https://blog.mozilla.org/en/mozilla/firefox-rolls-out-total-cookie-protection-by-default-to-all-users-worldwide/.

[29] Anubhavnidhi Abhashkumar, Jeongkeun Lee, Jean Tourrilhes, Sujata Banerjee, Wenfei Wu, Joon-Myung Kang, and Aditya Akella. P5: Policy-driven optimization of p4 pipeline. In *Proceedings of the Symposium on SDN Research*, pages 136–142, 2017.

[30] Tyler Akidau, Alex Balikov, Kaya Bekiroğlu, Slava Chernyak, Josh Haberman, Reuven Lax, Sam McVeety, Daniel Mills, Paul Nordstrom, and Sam Whittle. Millwheel: Fault-tolerant stream processing at internet scale. *Proceedings of the VLDB Endowment*, 6(11):1033–1044, 2013.

[31] Michael Armbrust, Tathagata Das, Joseph Torres, Burak Yavuz, Shixiong Zhu, Reynold Xin, Ali Ghodsi, Ion Stoica, and Matei Zaharia. Structured streaming: A declarative api for real-time applications in apache spark. In *Proceedings of the 2018 International Conference on Management of Data*, pages 601–613, 2018.

[32] Magdalena Balazinska, Hari Balakrishnan, Samuel Madden, and Michael Stonebraker. Fault-tolerance in the borealis distributed stream processing system. In *Proceedings of the 2005 ACM SIGMOD international conference on Management of data*, pages 13–24, 2005.

[33] Tom Barbette, Chen Tang, Haoran Yao, Dejan Kostić, Gerald Q Maguire Jr, Panagiotis Papadimitratos, and Marco Chiesa. A high-speed load-balancer design with guaranteed per-connection-consistency. In *17th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 20)*, pages 667–683, 2020.

[34] Andrea Bittau, Úlfar Erlingsson, Petros Maniatis, Ilya Mironov, Ananth Raghunathan, David Lie, Mitch Rudominer, Ushasree Kode, Julien Tinnés, and Bernhard Seefeld. Prochlo: Strong privacy for analytics in the crowd. In *Proceedings of the 26th Symposium on Operating Systems Principles, Shanghai, China, October 28-31, 2017*, pages 441–459. ACM, 2017.

[35] Pat Bosshart, Dan Daly, Glen Gibb, Martin Izzard, Nick McKeown, Jennifer Rexford, Cole Schlesinger, Dan Talayco, Amin Vahdat, George Varghese, et al. P4: Programming protocol-independent packet processors. *ACM SIGCOMM Computer Communication Review*, 44(3):87–95, 2014.

[36] Pat Bosshart, Glen Gibb, Hun-Seok Kim, George Varghese, Nick McKeown, Martin Izzard, Fernando Mujica, and Mark Horowitz. Forwarding metamorphosis: Fast programmable match-action processing in hardware for sdn. *ACM SIGCOMM Computer Communication Review*, 43(4):99–110, 2013.

[37] Justin Brickell and Vitaly Shmatikov. Efficient anonymity-preserving data collection. In *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 76–85, 2006.

[38] Martin Burkhart, Mario Strasser, Dilip Many, and Xenofontas Dimitropoulos. {SEPIA}:{Privacy-Preserving} aggregation of {Multi-Domain} network events and statistics. In *19th USENIX Security Symposium (USENIX Security 10)*, 2010.

[39] Matthew Caesar and Jennifer Rexford. Bgp routing policies in isp networks. *IEEE network*, 19(6):5–11, 2005.

[40] Hung Cao and Monica Wachowicz. Analytics everywhere for streaming iot data. In *2019 Sixth International Conference on Internet of Things: Systems, Management and Security (IOTSMS)*, pages 18–25. IEEE, 2019.

[41] Hung Cao and Monica Wachowicz. An edge-fog-cloud architecture of streaming analytics for internet of things applications. *Sensors*, 19(16):3594, 2019.

[42] Don Carney, Uğur Çetintemel, Mitch Cherniack, Christian Convey, Sangdon Lee, Greg Seidman, Nesime Tatbul, Stan Zdonik, and Michael Stonebraker. Monitoring streams—a new class of data management applications. In *VLDB'02: Proceedings of the 28th International Conference on Very Large Databases*, pages 215–226. Elsevier, 2002.

[43] Sirish Chandrasekaran, Owen Cooper, Amol Deshpande, Michael J Franklin, Joseph M Hellerstein, Wei Hong, Sailesh Krishnamurthy, Samuel R Madden, Fred Reiss, and Mehul A Shah. Telegraphcq: continuous dataflow processing. In *Proceedings of the 2003 ACM SIGMOD international conference on Management of data*, pages 668–668, 2003.

[44] Guoqiang Jerry Chen, Janet L Wiener, Shridhar Iyer, Anshul Jaiswal, Ran Lei, Nikhil Simha, Wei Wang, Kevin Wilfong, Tim Williamson,

and Serhat Yilmaz. Realtime data processing at facebook. In *Proceedings of the 2016 International Conference on Management of Data*, pages 1087–1098, 2016.

[45] Xiaoqi Chen. Implementing AES encryption on programmable switches via scrambled lookup tables. In Ang Chen and Laurent Vanbever, editors, *Proceedings of the 2020 ACM SIGCOMM 2020 Workshop on Secure Programmable Network Infrastructure, SPIN@SIGCOMM 2020, Virtual Event, USA, August 14, 2020*, pages 8–14. ACM, 2020.

[46] Sanket Chintapalli, Derek Dagit, Bobby Evans, Reza Farivar, Thomas Graves, Mark Holderbaugh, Zhuo Liu, Kyle Nusbaum, Kishorkumar Patil, Boyang Jerry Peng, et al. Benchmarking streaming computation engines: Storm, flink and spark streaming. In *2016 IEEE international parallel and distributed processing symposium workshops (IPDPSW)*, pages 1789–1792. IEEE, 2016.

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

[48] Henry Corrigan-Gibbs and Dan Boneh. Prio: Private, robust, and scalable computation of aggregate statistics. In Aditya Akella and Jon Howell, editors, *14th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2017, Boston, MA, USA, March 27-29, 2017*, pages 259–282. USENIX Association, 2017.

[49] Henry Corrigan-Gibbs, Dan Boneh, and David Mazières. Riposte: An anonymous messaging system handling millions of users. In *2015 IEEE Symposium on Security and Privacy*, pages 321–338. IEEE, 2015.

[50] Marcos Dias de Assuncao, Alexandre da Silva Veith, and Rajkumar Buyya. Distributed data stream processing and edge computing: A survey on resource elasticity and future directions. *Journal of Network and Computer Applications*, 103:1–17, 2018.

[51] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, 2008.

[52] Bolin Ding, Janardhan Kulkarni, and Sergey Yekhanin. Collecting telemetry data privately. In Isabelle Guyon, Ulrike von Luxburg, Samy Bengio, Hanna M. Wallach, Rob Fergus, S. V. N. Vishwanathan, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, pages 3571–3580, 2017.

[53] Constantine Dovrolis, Krishna Gummadi, Aleksandar Kuzmanovic, and Sascha D Meinrath. Measurement lab: Overview and an invitation to the research community, 2010.

[54] Cynthia Dwork, Frank McSherry, Kobbi Nissim, and Adam D. Smith. Calibrating noise to sensitivity in private data analysis. In Shai Halevi and Tal Rabin, editors, *Theory of Cryptography, Third Theory of Cryptography Conference, TCC 2006, New York, NY, USA, March 4-7, 2006, Proceedings*, volume 3876 of *Lecture Notes in Computer Science*, pages 265–284. Springer, 2006.

[55] Cynthia Dwork and Aaron Roth. The algorithmic foundations of differential privacy. *Found. Trends Theor. Comput. Sci.*, 9(3-4):211–407, 2014.

[56] Úlfar Erlingsson, Vitaly Feldman, Ilya Mironov, Ananth Raghunathan, Kunal Talwar, and Abhradeep Thakurta. Amplification by shuffling: From local to central differential privacy via anonymity. In Timothy M. Chan, editor, *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2019, San Diego, California, USA, January 6-9, 2019*, pages 2468–2479. SIAM, 2019.

[57] Úlfar Erlingsson, Vasyl Pihur, and Aleksandra Korolova. RAPPOR: randomized aggregatable privacy-preserving ordinal response. In Gail-Joon Ahn, Moti Yung, and Ninghui Li, editors, *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security, Scottsdale, AZ, USA, November 3-7, 2014*, pages 1054–1067. ACM, 2014.

[58] Keith B. Frikken and Philippe Golle. Private social network analysis: how to assemble pieces of a graph privately. In Ari Juels and Marianne Winslett, editors, *Proceedings of the 2006 ACM Workshop on Privacy in the Electronic Society, WPES 2006, Alexandria, VA, USA, October 30, 2006*, pages 89–98. ACM, 2006.

[59] Zacharias Georgiou, Moysis Symeonides, Demetris Trihinas, George Pallis, and Marios D Dikaiakos. Streamsight: A query-driven framework for streaming analytics in edge computing. In *2018 IEEE/ACM 11th International Conference on Utility and Cloud Computing (UCC)*, pages 143–152. IEEE, 2018.

[60] Petros Gigis, Matt Calder, Lefteris Manassakis, George Nomikos, Vasileios Kotronis, Xenofontas Dimitropoulos, Ethan Katz-Bassett, and Georgios Smaragdakis. Seven years in the life of hypergiants' off-nets. In *Proceedings of the 2021 ACM SIGCOMM 2021 Conference*, pages 516–533, 2021.

[61] Robert M. Gray and David L. Neuhoff. Quantization. *IEEE Trans. Inf. Theory*, 44(6):2325–2383, 1998.

[62] Susan Hohenberger, Steven Myers, Rafael Pass, et al. Anonize: A large-scale anonymous survey system. In *2014 IEEE Symposium on Security and Privacy*, pages 375–389. IEEE, 2014.

[63] Chi-Yao Hong, Subhasree Mandal, Mohammad Al-Fares, Min Zhu, Richard Alimi, Chandan Bhagat, Sourabh Jain, Jay Kaimal, Shiyu Liang, Kirill Mendelev, et al. B4 and after: managing hierarchy, partitioning, and asymmetry for availability and scale in google's software-defined wan. In *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*, pages 74–87, 2018.

[64] Pei Huang, Xiaonan Zhang, Linke Guo, and Ming Li. Incentivizing crowdsensing-based noise monitoring with differentially-private locations. *IEEE Trans. Mob. Comput.*, 20(2):519–532, 2021.

[65] Nikita Ivkin, Zhuolong Yu, Vladimir Braverman, and Xin Jin. Qpipe: Quantiles sketch fully in the data plane. In *Proceedings of the 15th International Conference on Emerging Networking Experiments And Technologies*, pages 285–291, 2019.

[66] Xin Jin, Xiaozhou Li, Haoyu Zhang, Robert Soulé, Jeongkeun Lee, Nate Foster, Changhoon Kim, and Ion Stoica. Netcache: Balancing key-value stores with fast in-network caching. In *Proceedings of the 26th Symposium on Operating Systems Principles*, pages 121–136, 2017.

[67] Albert Jonathan, Abhishek Chandra, and Jon Weissman. Multi-query optimization in wide-area streaming analytics. In *Proceedings of the ACM symposium on cloud computing*, pages 412–425, 2018.

[68] Taiwo Kolajo, Olawande Daramola, and Ayodele Adebiyi. Big data stream analysis: a systematic literature review. *Journal of Big Data*, 6(1):1–30, 2019.

[69] Logan Kugler. How the internet spans the globe. *Communications of the ACM*, 63(1):14–16, 2019.

[70] Sanjeev Kulkarni, Nikunj Bhagat, Maosong Fu, Vikas Kedigehalli, Christopher Kellogg, Sailesh Mittal, Jignesh M Patel, Karthik Ramasamy, and Siddarth Taneja. Twitter heron: Stream processing at scale. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, pages 239–250, 2015.

[71] Albert Kwon, David Lazar, Srinivas Devadas, and Bryan Ford. Riffle. *Proceedings on Privacy Enhancing Technologies*, 2016(2):115–134, 2016.

[72] ChonLam Lao, Yanfang Le, Kshiteej Mahajan, Yixi Chen, Wenfei Wu, Aditya Akella, and Michael M Swift. Atp: In-network aggregation for multi-tenant learning. In *NSDI*, pages 741–761, 2021.

[73] Yuliang Li, Rui Miao, Changhoon Kim, and Minlan Yu. Flowradar: A better netflow for data centers. In *13th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 16)*, pages 311–324, 2016.

[74] Shihan Lin, Rui Xin, Aayush Goel, and Xiaowei Yang. Invicloak: An end-to-end approach to privacy and performance in web content distribution. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security, CCS 2022, Los Angeles, CA, USA, November 7-11, 2022*, pages 1947–1961. ACM, 2022.

[75] Yehuda Lindell. Fast cut-and-choose-based protocols for malicious and covert adversaries. *Journal of Cryptology*, 29(2):456–490, 2016.

[76] Zaoxing Liu, Antonis Manousis, Gregory Vorsanger, Vyas Sekar, and Vladimir Braverman. One sketch to rule them all: Rethinking network flow monitoring with univmon. In *Proceedings of the 2016 ACM SIGCOMM Conference*, pages 101–114, 2016.

[77] Zaoxing Liu, Hun Namkung, Georgios Nikolaidis, Jeongkeun Lee, Changhoon Kim, Xin Jin, Vladimir Braverman, Minlan Yu, and Vyas Sekar. Jaqen: A high-performance switch-native approach for detecting and mitigating volumetric ddos attacks with programmable switches. In *30th {USENIX} Security Symposium ({USENIX} Security 21)*, 2021.

[78] Rui Miao, Hongyi Zeng, Changhoon Kim, Jeongkeun Lee, and Minlan Yu. Silkroad: Making stateful layer-4 load balancing fast and cheap using switching asics. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, pages 15–28, 2017.

[79] Mehdi Mohammadi, Ala Al-Fuqaha, Sameh Sorour, and Mohsen Guizani. Deep learning for iot big data and streaming analytics: A survey. *IEEE Communications Surveys & Tutorials*, 20(4):2923–2960, 2018.

[80] Valeria Nikolaenko, Stratis Ioannidis, Udi Weinsberg, Marc Joye, Nina Taft, and Dan Boneh. Privacy-preserving matrix factorization. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, pages 801–812, 2013.

[81] Raluca Ada Popa, Andrew J Blumberg, Hari Balakrishnan, and Frank H Li. Privacy and accountability for location-based aggregate statistics. In *Proceedings of the 18th ACM conference on Computer and communications security*, pages 653–666, 2011.

[82] Dan RK Ports and Jacob Nelson. When should the network be the computer? In *Proceedings of the Workshop on Hot Topics in Operating Systems*, pages 209–215, 2019.

[83] Qifan Pu, Ganesh Ananthanarayanan, Peter Bodik, Srikanth Kandula, Aditya Akella, Paramvir Bahl, and Ion Stoica. Low latency geo-distributed data analytics. *ACM SIGCOMM Computer Communication Review*, 45(4):421–434, 2015.

[84] Ariel Rabkin, Matvey Arye, Siddhartha Sen, Vivek S Pai, and Michael J Freedman. Aggregation and degradation in jetstream: Streaming analytics in the wide area. In *11th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 14)*, pages 275–288, 2014.

[85] Edo Roth, Karan Newatia, Yiping Ma, Ke Zhong, Sebastian Angel, and Andreas Haeberlen. Mycelium: Large-scale distributed graph queries with differential privacy. In Robbert van Renesse and Nickolai Zeldovich, editors, *SOSP '21: ACM SIGOPS 28th Symposium on Operating Systems Principles, Virtual Event / Koblenz, Germany, October 26-29, 2021*, pages 327–343. ACM, 2021.

[86] Mario A Sánchez, John S Otto, Zachary S Bischof, David R Choffnes, Fabián E Bustamante, Balachander Krishnamurthy, and Walter Willinger. Dasu: Pushing experiments to the internet's edge. In *10th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 13)*, pages 487–499, 2013.

[87] Amedeo Sapio, Marco Canini, Chen-Yu Ho, Jacob Nelson, Panos Kalnis, Changhoon Kim, Arvind Krishnamurthy, Masoud Moshref, Dan Ports, and Peter Richtarik. Scaling distributed machine learning with in-network aggregation. In *18th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 21)*, pages 785–808, 2021.

[88] Anirudh Sivaraman, Alvin Cheung, Mihai Budiu, Changhoon Kim, Mohammad Alizadeh, Hari Balakrishnan, George Varghese, Nick McKeown, and Steve Licking. Packet transactions: High-level programming for line-rate switches. In *Proceedings of the 2016 ACM SIGCOMM Conference*, pages 15–28, 2016.

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

[90] Apple's Differential Privacy Team. Learning Privacy at Scale, 2021. https://machinelearning.apple.com/research/learning-with-privacy-at-scale.

[91] Ankit Toshniwal, Siddarth Taneja, Amit Shukla, Karthik Ramasamy, Jignesh M Patel, Sanjeev Kulkarni, Jason Jackson, Krishna Gade, Maosong Fu, Jake Donham, et al. Storm@ twitter. In *Proceedings of the 2014 ACM SIGMOD international conference on Management of data*, pages 147–156, 2014.

[92] Michael Trusov, Liye Ma, and Zainab Jamal. Crumbs of the cookie: User profiling in customer-base analysis and behavioral targeting. *Mark. Sci.*, 35(3):405–426, 2016.

[93] Han Wang, Robert Soulé, Huynh Tu Dang, Ki Suh Lee, Vishal Shrivastav, Nate Foster, and Hakim Weatherspoon. P4fpga: A rapid prototyping framework for p4. In *Proceedings of the Symposium on SDN Research*, pages 122–135, 2017.

[94] David Isaac Wolinsky, Henry Corrigan-Gibbs, Bryan Ford, and Aaron Johnson. Dissent in numbers: Making strong anonymity scale. In *10th USENIX Symposium on Operating Systems Design and Implementation (OSDI 12)*, pages 179–182, 2012.

[95] Ning Xia, Han Hee Song, Yong Liao, Marios Iliofotou, Antonio Nucci, Zhi-Li Zhang, and Aleksandar Kuzmanovic. Mosaic: quantifying privacy leakage in mobile networks. In Dah Ming Chiu, Jia Wang, Paul Barford, and Srinivasan Seshan, editors, *ACM SIGCOMM 2013 Conference, SIGCOMM 2013, Hong Kong, August 12-16, 2013*, pages 279–290. ACM, 2013.

[96] Yunming Xiao, Matteo Varvello, and Aleksandar Kuzmanovic. Monetizing spare bandwidth: The case of distributed vpns. *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, 6(2):33:1–33:27, 2022.

[97] Rui Xin, Shihan Lin, and Xiaowei Yang. Quantifying user password exposure to third-party cdns. In *Passive and Active Measurement - 24th International Conference, PAM 2023, Virtual Event, March 21-23, 2023, Proceedings*, volume 13882 of *Lecture Notes in Computer Science*, pages 652–668. Springer, 2023.

[98] Zhaoqi Xiong and Noa Zilberman. Do switches dream of machine learning? toward in-network classification. In *Proceedings of the 18th ACM workshop on hot topics in networks*, pages 25–33, 2019.

[99] Dingqi Yang, Bingqing Qu, and Philippe Cudré-Mauroux. Privacy-preserving social media data publishing for personalized ranking-based recommendation. *IEEE Transactions on Knowledge and Data Engineering*, 31(3):507–520, 2018.

[100] Zhiqiang Yang, Sheng Zhong, and Rebecca N. Wright. Anonymity-preserving data collection. In Robert Grossman, Roberto J. Bayardo, and Kristin P. Bennett, editors, *Proceedings of the Eleventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Chicago, Illinois, USA, August 21-24, 2005*, pages 334–343. ACM, 2005.

[101] Yifan Yuan, Omar Alama, Jiawei Fei, Jacob Nelson, Dan R. K. Ports, Amedeo Sapio, Marco Canini, and Nam Sung Kim. Unlocking the power of inline floating-point operations on programmable switches. In Amar Phanishayee and Vyas Sekar, editors, *19th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2022, Renton, WA, USA, April 4-6, 2022*, pages 683–700. USENIX Association, 2022.

[102] Matei Zaharia, Tathagata Das, Haoyuan Li, Timothy Hunter, Scott Shenker, and Ion Stoica. Discretized streams: Fault-tolerant streaming computation at scale. In *Proceedings of the twenty-fourth ACM symposium on operating systems principles*, pages 423–438, 2013.

[103] Ben Zhang, Xin Jin, Sylvia Ratnasamy, John Wawrzynek, and Edward A Lee. Awstream: Adaptive wide-area streaming analytics. In *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*, pages 236–252, 2018.

A Ethical Consideration

Our measurement in Section 5.1 involves sending requests through proxies located at Internet users' home networks. However, these Internet users are selling their Internet access, and the dVPN service is publicly available. Therefore, this is no different than connecting to traditional VPNs. Further, we did not send any malicious requests or had any operations which might endanger the proxies. Thus, this work does not have any ethical concerns.

B System Design

B.1 Application-Layer Semantic Cookie

Speedup. Here we investigate the speedup for application-layer semantic cookie with QUIC 0-RTT connections. Because QUIC-0RTT send data at the very beginning, we have

$$S_{app-https-0rtt} = \frac{d_{CE} + d_{EW} + d_{WA} + T_{trans} + T_E + T_W + T_A}{d_{CE} + d_{EA} + T'_E + T'_A}. \tag{4}$$

We further look into the speedup for application-layer semantic cookies when TCP connections are adopted. For an unencrypted HTTP request, on top of TCP, the speedup S_{app} of the streaming analytics is

$$S_{app-http-tcp} = \frac{3d_{CE} + 3d_{EW} + d_{WA} + T_{trans} + T_E + T_W + T_A}{3d_{CE} + d_{EA} + T'_E + T'_A}, \tag{5}$$

where the coefficient 3 in $3d_{CE}$ and $3d_{EW}$ comes from the 1-RTT TCP handshake process during the connection establishment.

For HTTPS requests, TCP + TLS 1.2 handshakes need at least 3 RTTs to set up. Thus, the speedup is

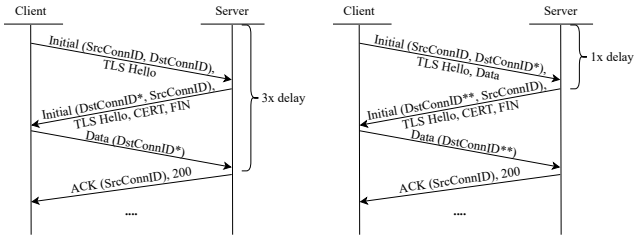
$$S_{app-https-tcp} = \frac{7d_{CE} + 7d_{EW} + d_{WA} + T_{trans} + T_E + T_W + T_A}{7d_{CE} + d_{EA} + T'_E + T'_A}. \tag{6}$$

For example, 3 RTTs needed to establish an HTTPS connection between a client and an edge servers implies 7 one-way delays, *i.e.*, 7 d_{CE}.

B.2 Transport-Layer Semantic Cookie

Transport-layer cookies are semantic cookies that are encoded in the transport-layer protocol. As identified in a previous study [33], there are three ways to encode cookies in the transport layer without requiring any modifications on the users' machines: (1) encode the cookie into the least significant bits of IPv6 addresses with a maximum of 64 bits, (2) encode the cookie into the `timestamp` option of TCP with a maximum of 32 bits, and (3) encode the cookie into the `conneciton ID` of QUIC with a maximum of 160 bits.

IPv6 – The use of IPv6 addresses requires the assumption that the MAC address is associated with the least significant bits of the IPv6 address, and thus is not appropriate in our case. We consider the other two options: via the TCP timestamp and via the QUIC connection id.



(a) QUIC 1-RTT. (b) QUIC 0-RTT.

Figure 7. QUIC handshake procedure and the time cost for the server to receive data.

TCP – When the TCP timestamp option TSP is set and used in one direction (*e.g.*, from server to client), all the packets in the reverse direction (from client to server) will attach the same TSP value automatically. However, there are several issues with this approach. First, the TSP value cannot be reused in the next TCP connection. Second, if the client wants to send the cookie in the next TCP connection proactively, it requires non-negligible modification on the client's side – access to the root privilege and modifying the outgoing packets accordingly. This breaks our vision of minimal to no client modification.

QUIC – QUIC is a transport-layer protocol implemented in the userspace on top of UDP. The QUIC connection establishment procedure is illustrated in Figure 7 (left for the 1-RTT handshake, right for the 0-RTT handshake). For QUIC 1-RTT, a long QUIC header will be used during the handshake phase. The client will send two randomly generated connection IDs *SrcConnID* and *DstConnID*. Then the server will copy *SrcConnID* but set a new *DstConnID** and return them to the client. In the following communication, a short QUIC header will be used where the client sends packets with *DstConnID** and the server sends packets with *SrcConnID*. Further, the server can reset the connection ID with version negotiation packets at any time. For QUIC 0-RTT, it is only applicable when there was a previous connection between the same end-points. The client will send the same *DstConnID** as in the last connection.

We find that the connection-id field, in particular *DstConnID**, allows the encoding of transport-layer cookies. In addition, it takes minimal effort to modify the connection-id field because QUIC is implemented in the userspace. Thus, it fits our vision of minimal (QUIC-1RTT) to no (QUIC-0RTT) client modification.

B.3 Custom Aggregation Packet

We create a custom packet header on top of UDP to carry early-forwarded cookies or pre-processed data (either by LarkSwitch or edge server) for AggSwitch. Figure 8 shows that the custom packet header includes three parts: 1) a 16-bit special string `SID`, a custom identifier for distinguishing

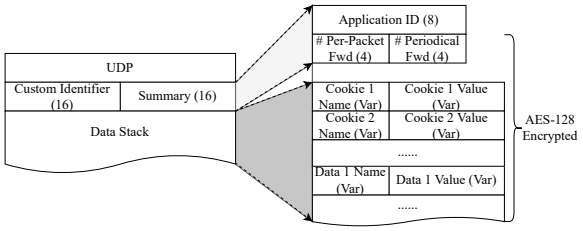


Figure 8. Custom aggregation packet design.

from regular UDP packets; 2) a 16-bit `summary` that contains application-ID and the number of sub-cookies/data for either per-packet forwarding or periodical forwarding, respectively; 3) `data-stack` that contains N sub-cookies and data. All data after the application ID are encrypted using the AES-128 algorithm.

The extracted cookies and data encoded in the custom aggregation packet from LarkSwitch and edge server to AggSwitch may be lost because UDP is used. We argue that the benefits of using UDP overtake the loss. The loss here is that less than 0.01%, *i.e.,* the packet drop rate in today's WAN [17, 19, 24], of the cookies or data will be lost. In comparison, there are two major benefits. First, for short-term analysis, which is the target for Snatch, the value of the data is much higher when the data is available sooner. Dropping one data point out of tens or hundreds of thousands will not make a large difference to the distribution of the data, and thus to the results. At the same time, the data is not lost forever. For a long-term analysis, full and accurate results can be obtained by syncing up the records at the web servers or related databases. Second, implementing a retransmission mechanism on programmable switches is non-trivial and consumes scarce DRAM resources to keep the status. Instead, the resources can be used to offload more computation and thus provide better speedup or support more applications. In conclusion, it is the best choice for Snatch to adopt UDP for the custom aggregation packet.

B.4 Repeated Counting

One potential issue with periodical forwarding is that one user might send multiple requests to the web server within one period and thus cause repeated counting. While repeated counting is needed in some scenarios, it can be avoided by implementing a hash table or a Bloom filter, which is widely used in projects involving programmable switches [66, 73, 78].

C Scope of Snatch Applications

Here, we take Spark Streaming as a comparison to illustrate what can be done for the in-network streaming analytics (INSA). Indeed, INSA is not as flexible as Spark Streaming because of the constraint on the programming model and computational and storage resources. Our goal for INSA is to assist with the streaming analytics and potentially complete

Table 1. Supported operations and related application with in-network streaming analytics. N/A for not applicable, N for not supported, Y for supported, and Y* for supported with limitation.

DStream Method	INSA	Category
cache()	N/A	DStream-specific
checkpoint(interval)	N/A	DStream-specific
cogroup(other[, numPartitions])	Y*	partition, table-join
combineByKey(createCombiner, mergeValue, ...)	Y*	foreach
context()	N/A	DStream-specific
count()	Y	reduce
countByValue()	Y	reduce
countByValueAndWindow(windowDuration, ...[, ...])	Y	window, reduce
countByWindow(windowDuration, slideDuration)	Y	window, reduce
filter(func)	Y*	foreach
flatMap(func[, preservesPartitioning])	Y*	partition, foreach
flatMapValues(func)	Y*	foreach,
foreachRDD(func)	Y*	foreach
fullOuterJoin(other[, numPartitions])	Y*	partition, table-join
glom()	N/A	DStream-specific
groupByKey([numPartitions])	Y	partition, reduce
groupByKeyAndWindow(windowDuration, ...[, ...])	Y	partition, window, reduce
join(other[, numPartitions])	Y*	partition, table-join
leftOuterJoin(other[, numPartitions])	Y*	partition, table-join
map(func[, preservesPartitioning])	Y*	partition, foreach
mapPartitions(func[, preservesPartitioning])	Y*	partition, foreach
mapPartitionsWithIndex(func[, ...])	Y*	partition, foreach
mapValues(func)	Y*	foreach
partitionBy(numPartitions[, partitionFunc])	N	partition
persist(storageLevel)	N/A	DStream-specific
pprint([num])	N/A	DStream-specific
reduce(func)	Y*	reduce
reduceByKey(func[, numPartitions])	Y*	partition, reduce
reduceByKeyAndWindow(func, invFunc, ...[, ...])	Y*	partition, window, reduce
reduceByWindow(reduceFunc, invReduceFunc, ...)	Y*	window, reduce
repartition(numPartitions)	N	partition
rightOuterJoin(other[, numPartitions])	Y*	partition, table-join
saveAsTextFiles(prefix[, suffix])	N/A	DStream-specific
slice(begin, end)	Y	window
transform(func)	Y*	foreach
transformWith(func, other[, keepSerializer])	Y*	foreach
union(other)	Y*	table-join
updateStateByKey(updateFunc[, ...])	Y*	foreach
window(windowDuration[, slideDuration])	Y	window

relatively simple tasks alone, but not to entirely replace Spark Streaming.

While Snatch handles multiple tasks, here we focus on the "depth" of each task and hence assume to support only one task. In the discussion of the feasibility to achieve a function, we consider that modifications can be made at either the compiling phase, *i.e.,* modifying the P4 code, or Snatch application submission phase, *i.e.,* the application developer encodes the cookies and sets up the corresponding receiver at the analytics server.

In addition, because today's P4 model only supports partial integer operation (see "Statistics Calculation" in Section 4.1), we limit the following discussion in the scope of integer operations. Yet, it is noteworthy that it is possible to perform counting operations for strings: The application developer can either encode the string to integer or use a dictionary when the value possibility is limited. In this way, counting can be done by matching the hash value or the keyword. Still, other string functions such as concatenate are not supported. It is also noteworthy that the latest study has demonstrated that it is viable to perform float operation with programmable switches by carefully rescheduling the computation procedure [101]. An alternative is to leverage float number quantization [61].

A Spark Streaming program often executes a series of DStream methods [22], *e.g.,* map, reduce, etc, to a DStream object, *i.e.,* the data within an interval. For the sake of convenience of discussion, we classify the DStream methods into several categories: DStream-specific, partition, foreach, window, table-join, and reduce. A method may belong to multiple categories at the same time. For instance, `reduceByKeyAndWindow` belongs to three categories: partition, window, and reduce. Table 1 lists all the DStream methods, whether they can be done with INSA, and their categories. Indeed, the complexity of some DStream methods heavily depend on the input functions, and whether INSA supports such a DStream method depends on the input function, *i.e.,* when the operands in the input function are supported by programmable switches, the DStream method is supported by P4, and vice versa. Moreover, the total number of DStream methods that are operated on a DStream object is restricted by the limited number of pipeline stages of the programmable switches [29]. Below, we discuss the methods in detail by category.

DStream-specific methods include `cache`, `checkpoint`, `context`, `glom`, `persist`, `pprint`, and `saveAsText-Files`. They are not applicable to INSA because they are specific for assisting the Spark programming model but not computation-related operations. Related discussion involves fault tolerance, where more details are available in Appendix B.3.

Direct partition methods include `partitionBy` and `repartition`, whereas indirect partition methods, *i.e.,* where partition number is an optional input parameter, include methods in foreach, window, table join, and reduce categories. To investigate these methods, we first need to understand more about the underlying data model of Spark Streaming. Resilient Distributed Dataset (RDD) includes all the streaming data within a batch interval from all partitions, which refers to the data stored at one Spark node and is the basic operable unit in Spark. In Snatch, each edge node, *i.e.,* ISP switch or edge server, can be regarded as a partition where data is stored. But unlike Spark, the data in each partition depends on client location and activities, and cannot be moved or reassigned in Snatch. Therefore, partitionBy and repartition are not supported by INSA. However, operations on the partition are possible: AggSwitch can set up a match table for each edge node and perform different actions accordingly. The modifications should be made at the compiling phase.

Foreach methods include `combineByKey`, `filter`, `foreachRDD`, `map`, `flatMap`, `flatMapValues`, `mapPartitions`, `mapValues`, `mapPartitionsWithIndex`, `transform`, `transformWith`, and `updateStateByKey`. The main purpose of these methods is to allow operations at a finer granularity, *i.e.,* at per data point level. In INSA, the programmable switch is processing at per-packet granularity. Therefore, foreach methods are naturally supported by INSA while subjected to input function, *i.e.,* as long as the input function is supported by INSA, the foreach methods are supported by INSA.

Direct window methods include `slice` and `window` whereas indirect window methods, *i.e.,* where window settings are optional input parameters, include methods in reduce categories. Method window provides flexibility by allowing the user to extract a new windowed DStream based on the existing DStream but with a different interval. Method slice is similar but only needs aggregated data within one interval. The periodical forwarding in Snatch is similar to window methods as it returns data on windowed packets. In the same spirit, Snatch is able to realize both direct and indirect window methods by achieving another periodical forwarding with a second time counter registers. The modifications should be made at the compiling phase.

Reduce methods include `count`, `countByValue`, `countByValueAndWindow`, `countByWindow`, `groupByKey`, `groupByKeyAndWindow`, `reduce`, `reduceByKey`, `reduceByKeyAndWindow`, and `reduceByWindow`. Among them, count and groupByKey and their associated methods can be regarded as special cases for reduce and associated methods, and they have been implemented in our Snatch prototype. Reduce and associated methods fit in the match and action programming model, and thus should be supported by INSA as long as the input function is supported by INSA. The modifications should be made at the compiling phase.

Finally, table-join methods include `cogroup`, `join`, `fullOuterJoin`, `leftOuterJoin`, `rightOuterJoin`, and `union`.

These methods correspond to SQL join clauses which combine the columns from one or more tables. Snatch's cookie/data-stack has very similar data structure from tables, and technically it is possible to perform the join method at AggSwitch by storing all the cookie/data from periodical aggregation packets (representing DStreams) in the switch and then construct another custom packet as a result of join and deliver it to the analytics server. For instance, we take the `fullOuterJoin` as an example. Stream 1 has cookies A, B, C whereas Stream 2 has cookies A, D, E. AggSwitch reserves a register space for a table with columns A, B, C, D, E. When collecting periodical aggregation packets from LarkSwitches, what AggSwitch needs to do is simply fill in the registers according to the value in cookie A. Thus, when all the periodical aggregation packets are received, AggSwitch has a full table of the result of `fullOuterJoin` on Stream 1 and 2. Other table-join operations can be done in a similar spirit. Here, the modifications should be made at both the compiling phase and the application submission phase.

Note that table-join methods might be beneficial when applying to two separate applications per the developers' agreement, which is a topic we plan to explore in future work. Otherwise, it is a bad practice since it costs too much of the switch storage resources and a better design of the cookie/data-stack will remove the necessity for the join operation.

D More Measurement Results

D.1 Methodology and Delay Derivation

Snatch involves several components: the ISP switch, edge server, web server, and analytics server. To study the performance of these subjects in practice, we set up experiments as follows. First, we host two HTTPS websites with separate domains using AWS EC2 instances [7], which represent the web servers in Figure 1(b). Then, we purchase CDN services from Cloudflare [12] and AWS Cloud Front [14], respectively for each domain. This allows us to set up the edge servers on a global scale.

Next, we adopt the decentralized VPNs (dVPNs) as our means of measurement. We iteratively connect to all the available residential dVPN nodes. For each connection, we perform measurements as follows. First, we perform `traceroute` to our hosted domains, which retrieves the hops along the path to the destination with RTTs to each hop. A dVPN connection creates a VPN tunnel between the client and the proxy, which all the packets traverse through. Thus, the first hop will be the dVPN proxy itself, meaning that the delay to the first hop is the delay between our machine and the dVPN proxy. Because we want to measure the delays between the destinations and the dVPN proxy, we accordingly subtract the delay of the first hop for all the other measured delays.

We then investigate the next hops in increasing order. When the hop's IP is not private (judged by the prefix of IP address) for the first time, we consider it to be the first hop reaching the ISP, and record the associated delays. When we do not find the ISP in the first 10 hops, either because all are private IPs or the hop is not available to traceroute ("*" is returned), we consider the host not to be residential, *i.e.,* miscategorized by Mysterium, and discard the associated results.

Next, we investigate the delays between the dVPN proxy and the edge servers and cloud by performing `ping` to corresponding destinations. For edge servers, we perform ping to our domains. Because of the CDN services, the packets will be directed to Cloudflare CDN servers or Amazon edge servers instead of our EC2 instance in the cloud. In addition, we look up for, and ping, off-net servers that are in the same AS as the proxy, using the recently published database [60]. We record all the associated delays. For clouds, we perform ping to the IPs of our EC2 instances, as well as public servers in every AWS cloud region. We also measure the AWS inter-cloud delays following cloudping [13].

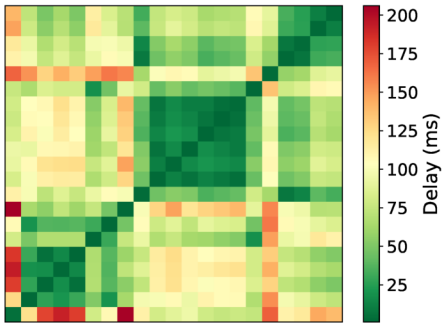
Further, we perform HTTPS GET and POST requests to our domains and the IP addresses of our EC2 instances. For POST requests to our domains, the CDN service will forward them to web servers by default. Along with the delays we measured using ping, we can infer the time cost of handling GET and POST requests by the edge and web servers, as well as the delay from the edge server to the cloud.

For all the per-site operations mentioned above, we iterate 10 times and take the median for further analysis to avoid outliers resulting from unstable network conditions.

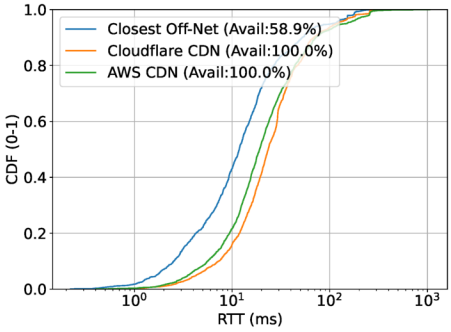
D.2 Best-Practice Assumption

When the web server and the analytics server are not in the same data center, the delays from the client and from the edge to them (d_{CA}, d_{EW}, and d_{EA}) change as d_{WA} changes. For the sake of simplicity, we assume the best setup practice: edge servers are set up globally for caching static content; the web servers are set up in all data centers for serving dynamic content; and one centralized analytics server is located at one data center. In this way, the client will always choose the closest edge and web server. In particular, the delay from the edge to the web server (d_{EW}) is approximated by taking the difference between the delays from the client to the closest cloud and from the client to the edge server, whereas the delay from the edge to the analytics server (d_{EA}) is represented by the "Edge-Cloud" curve in Figure 5(a).

In Section 5.1, we consider the delays from the client to edge (d_{CE}) and from the edge to the web server (d_{EW}) to be constant as the median values. In Section 5.2 where the Nth percentile of delay is adopted, we take the Nth percentile of d_{CE} and d_{EW} as well. Importantly, we further assume that the *delay from the client and from the edge to the analytics server* (d_{CA} and d_{EA}) *grows proportionally as the delay from the web server to the analytics server* (d_{WA}) *grows*, within their own



(a) The delay between AWS data centers.



(b) Delays from Internet client to different edge servers.

Figure 9. Measurement results.

range respectively. For instance, d_{EA} grows from 0.2 ms to 249.5 ms when d_{WA} grows from 0.8 ms to 206 ms.

D.3 Measurement Results

Figure 9(a) shows the matrix of intra- and inter-data center delays of the AWS cloud. The delays range from 0.8 ms (within the same data center) to 206 ms (from *ap-southeast-2* region to the *af-south-1* region). The inter-data center median delay is 75.5 ms. The inter-data center delays represent the communication cost from the web server to the analytics server, which may reside in different data centers as explained in Section 2.

Figure 9(b) shows the delays from client to different edge servers. The results show that the off-net servers are much closer to the clients compared to regular CDN services, though they cover only 57.9% clients in our measurement. Moreover, Amazon CloudFront outperforms Cloudflare CDN in our measurement. In our analysis in Section 5, we take the minimal delay among all the edge servers for each client, *i.e.*, if the off-net servers are present and outperform Amazon CloudFront and Cloudflare CDN, then the delay is for the off-net server; otherwise, the delay is the minimum between Amazon CloudFront and Cloudflare CDN.