Mass Surveillance of VoIP Calls in the Data Plane

Ege Cem Kirci
ETH Zürich

Maria Apostolaki
Princeton University

Roland Meier
ETH Zürich

Ankit Singla
ETH Zürich

Laurent Vanbever
ETH Zürich

ABSTRACT

Over the last decade, programmable data planes have enabled highly customizable and efficient packet processing in commercial off-the-shelf hardware. Although researchers have demonstrated various use cases of this technology, its potential misuse has gained much less traction. This work investigates a typical surveillance scenario, VoIP call identification and monitoring, through a tailored data-plane attack.

We introduce DELTA, a network-level side-channel attack that can efficiently identify VoIP calls *and* their hosting services. DELTA achieves this by tracking the inherent network footprint of VoIP services in the data plane. Specifically, DELTA stores the user addresses recently connected to VoIP services and links potential call flows with these addresses.

We implement DELTA on existing hardware and conduct high-throughput tests based on representative traffic. DELTA can simultaneously store around 100 000 VoIP connections per service and identify call streams in-path, at line-rate, inside terabits of Internet traffic per second, immediately revealing users' communication patterns.

CCS CONCEPTS

• **Networks** → **Programmable networks**.

KEYWORDS

VoIP, In-network monitoring, Internet surveillance

ACM Reference Format:
Ege Cem Kirci, Maria Apostolaki, Roland Meier, Ankit Singla, and Laurent Vanbever. 2022. Mass Surveillance of VoIP Calls in the Data Plane. In *Symposium on SDN Research (SOSR 22), October 19–20, 2022, Virtual Event, USA.* ACM, New York, NY, USA, 17 pages. https://doi.org/10.1145/3563647.3563649

SOSR 22, October 19–20, 2022, Virtual Event, USA

ACM ISBN 978-1-4503-9892-3/22/10…$15.00
https://doi.org/10.1145/3563647.3563649

1 INTRODUCTION

Since its inception in 2014, data-plane programmability [16] and P4 language [15] have powered innovation in various networking areas, including security and privacy [10, 29, 45]. Researchers have already demonstrated different use cases where commercial off-the-shelf programmable switches can potentially replace or surpass proprietary network middleboxes [38, 55, 101] while leading Internet providers such as Deutsche Telekom and AT&T have already been deploying them in their networks [18, 89]. Now that the entry barrier to in-network active intelligence is more accessible than ever, it is time to study the other side of the coin: how does in-line programmability facilitate potentially oppressive activity?

This work studies VoIP call identification, a well-known example of mass surveillance and censorship [30, 39, 61]. Past investigations reveal that NSA's PRISM program explicitly targeted calls for several popular VoIP applications [40, 51, 71], and various countries had been spying on their residents' calls [57, 80, 81]. Indeed, the news indicates that even the identification of the hosting service is critical as certain dissident groups choose specific services to hold their internal communication [2, 26, 60, 64, 87].

Given this interest, it is unsurprising that vendors supply equipment capable of monitoring VoIP calls for on-demand policing, blocking, and recording. This task is non-trivial since modern VoIP applications encrypt calls end-to-end [83–85], preventing the agencies from directly intercepting the calls. However, end-to-end encryption does not protect the calls' metadata, such as the IP addresses of the communicating parties,[1] packet sizes, inter-packet delays, or flow durations. Middleboxes such as Sandvine's [75] analyze this metadata to eventually "show who is talking to whom, for how long, and try to discover online anonymous identities" [35].

While these proprietary middleboxes are available today, they tend to be expensive, at $50 per Gbps [74], compared to a programmable switch which costs around $2 per Gbps [11]. A closer look into the technicalities of these middleboxes justifies their high cost. First, they are resource-intensive since they analyze traffic in their CPU clusters off-path [73]. Second, they require traffic mirroring or advanced state distribution to synchronize in the face of asymmetric routing [73].

[1] VoIP applications, by default, establish direct connections between the caller and the callee for better performance [83–85].

Third, they necessitate continual vendor support since traffic analysis depends on statistical learning and, thus, regular model updates [73]. However, this complexity has a silver lining in malicious use cases: it keeps the functionality relatively out of reach, as vendor dependence, dedicated middleboxes, and monetary costs are all limiting factors. We argue that programmable switches challenge these assumptions and demand a reassessment of this technology's possible malicious use cases. Although programmable switches are heavily restricted in their computation and storage capabilities, we show they are highly customizable for tailored attacks.

We introduce DELTA, a novel network-level side-channel attack that can efficiently identify VoIP calls in-path, in real-time, and at scale. DELTA differentiates from traffic analysis approaches [62, 72] by merely tracking connections instead of examining traffic. More specifically, DELTA leverages the VoIP signaling mechanism used for peer discovery and call setup: VoIP callers and callees first connect to a publicly known central application server shortly before communicating directly with each other. This fundamental call bootstrapping mechanism, used by prominent VoIP platforms, leads to a triangular connection pattern. DELTA takes advantage of this pattern by storing the signaling connections and linking the candidate call flows to these stored addresses. More precisely, DELTA continuously keeps track of the client IP addresses in recent signaling connections. Then, for each arriving flow, DELTA checks the flow address tuple against the stored IP addresses. This mechanism reveals the participating addresses of a call and identifies the facilitating application service without examining the traffic.

DELTA's in-path continuous tracking approach is particularly compelling considering reported incidents' years-long, country-wide surveillance scenarios [5, 36, 77]. First, DELTA detects call flows with the very first TLS [68] packet in the data plane. That enables the switch to take direct actions such as rate-limiting, blocking, or logging compared to traffic analysis which requires a flow observation period for inference and possibly off-path decision loops. Second, DELTA does not require policy-based re-routing or traffic mirroring to redirect flows into dedicated policing hardware. Administrative overheads such as policy-based re-routing configuration, distributed traffic mirroring, or off-path decision-loops introduce considerable complexity to a network. Indeed, researchers recently discovered a partial failure of a similar policing application in Russia [95].

DELTA is conceptually simple yet challenging to implement in the data plane. First, DELTA identifies signaling flows by reading the Server Name Indication (SNI) extension [33] included in TLS handshakes. Since the SNI field is of variable length and its position within the packet is unfixed, it is challenging to parse it correctly. We build a multi-stage finite state machine to parse this field iteratively. Second, DELTA requires real-time global state synchronization to share the stored signaling addresses among switches. For this, we implement a data-plane message exchange protocol under the hardware limitations of programmable switches. Third, DELTA requires continuously storing global signaling IP addresses in a scalable manner under tight memory constraints. That requires a space-efficient query data structure that can also keep track of the inserted entries' *liveness*. We achieve this by implementing a data structure called Double Buffering Bloom Filter (DBBF) [24] entirely in the data plane.

We implement DELTA on an Intel Tofino switch [42] and evaluate our prototype against both real and synthesized packet traces. Our experiments show that a single switch's DBBF can hold connection states to simultaneously detect up to 100 000 unique VoIP calls for each VoIP application with virtually no hash collisions within up to 6.4 Tbps of traffic, the switch's maximum throughput. To put these numbers in perspective, AMS-IX [7] – the busiest Internet location in the world – transits around 11 Tbps of traffic on its peak [8], whereas Whatsapp, the most popular VoIP service in the world, facilitates 100 million calls per day *worldwide* [63]. In other words, a rough extrapolation suggests that even two switches would have enough capacity to process worldwide VoIP traffic, provided the traffic would cross these switches.

DELTA demonstrates that, by commodifying highly efficient and capable hardware, programmable data planes have the potential to strengthen network attacks. From a cost-benefit risk-assessment perspective, this technical shift calls for further investigation as faulty assumptions regarding the practicality and cost of attacks may lead to underestimations of potential threats. We intend this work as a wake-up call and thereby devote §6 to discussing best practices that *users can deploy today* agains DELTA.

2 OVERVIEW

DELTA is a network-level side-channel attack that identifies VoIP calls and their facilitating VoIP applications. It uncovers the IP addresses, and by that, the identities of users making these calls, violating users' privacy expectations. Detecting the communicating parties *and* the VoIP application service enables chains of attacks, including *(i)* selectively blocking specific pairs of users on target VoIP services; *(ii)* performing behavioral analysis per VoIP service; *(iii)* tracking all the communications of VoIP users in real-time; or *(iv)* building an allow-list, default-off VoIP ecosystem enabled only for VoIP providers cooperating with the attacker [6, 97].

This section first presents our attacker model (§2.1); next, we describe on a high-level how VoIP services operate (§2.2). Then, we illustrate DELTA in action (§2.3) and discuss what makes it challenging to implement entirely in the data plane.

2.1 Attacker model

We consider an attacker controlling a territorial network infrastructure such as intelligence agencies or governments ruling over single or multiple Internet Service Providers. The attacker mainly uses DELTA for its in-path VoIP call and service type detection capabilities that can facilitate real-time blocking or recording of calls. The attacker cannot collude with the VoIP application service providers nor break their encryption or software systems. However, the attacker can eavesdrop on domestic networks. We believe this attacker model realistically depicts the setting observed in past call surveillance and interception incidents from various countries. Notably, these incidents were often ruled to be unlawful and illegal [70, 91].

The attacker knows the IP addresses and domain names used by the target VoIP application providers. Note that this information is publicly available. Thereby, the attacker does not compromise any VoIP infrastructure. The attacker's goal is to uncover the communicating parties' IP addresses, the facilitating application, along with the time and duration of the calls. If a caller is behind a middlebox with address translation capabilities, the attacker captures the public IP address of the middlebox. In this work, we assume the attacker has the ability to map the public IP address and port combinations generated by middleboxes to user identities. We note that DELTA can also be deployed on existing middleboxes that handle user traffic. Such an implementation, however, would require updates to existing middleboxes and, therefore, would be challenging to deploy compared to the standalone implementation proposed in the current work.

2.2 VoIP call setup

DELTA leverages the unique network-level call bootstrapping footprint that VoIP calls produce. All well-known VoIP services implement call establishment in a *signaling* and a *streaming* phase [13, 46, 53, 86]. The signaling step involves a "rendezvous server" with caller ↔ server and callee ↔ server communication typically encrypted using TLS. This call bootstrapping allows direct caller ↔ callee communication in the streaming step, typically over UDP [27, 66]. Eventually, an eavesdropping attacker can identify the triangular connection pattern even with encryption: two users communicate with a publicly known VoIP application endpoint, and shortly after, direct inter-communication over UDP flows begin.

Fig. 1 shows[2] the call setup between Alice (caller) and Bob (callee) over an arbitrary VoIP application:

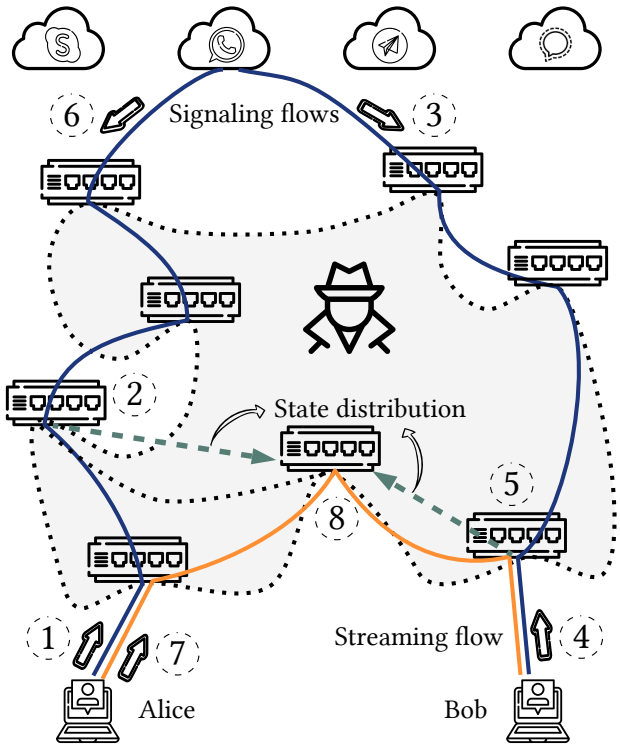


Figure 1: DELTA overview. An attacker identifies and couples VoIP signaling and streaming connections to reveal the IP addresses of the caller and the callee.

(1) Alice sends an `INVITE` message to the VoIP server, indicating the intent to call Bob.
(3) The server forwards the `INVITE` to Bob.
(4) With an `OK` message, Bob informs the VoIP server that he accepts the call.
(6) The server forwards the `OK` to Alice.
(7) Alice and Bob establish a direct UDP connection to stream call audio and video data.

Observe that the users can not establish a direct connection in some corner cases, such as when behind specific firewalls or symmetric/restricted network address translators (NATs). In such cases, the VoIP application provider falls back to a *relayed mode* of operation, which routes the call trafic over the VoIP service infrastructure. As DELTA only captures the two edges of the triangle in such cases (i.e., caller ↔ server and callee ↔ server), the attack mechanism needs modification to report these incidents. However, our experiments show that the tested VoIP services avoid the relayed mode unless necessary, as it introduces more delay and network costs for the VoIP provider.

[2]The enumerated dashed circles in the remaining text in this section stand for the dashed circles in Fig. 1.

2.3 DELTA in action

DELTA leverages programmable data planes to detect VoIP calls in-path, in real-time, and at scale. In a nutshell, the switches analyze each passing flow to identify *(i)* signaling flows between VoIP clients and a VoIP service and *(ii)* call flows between two VoIP clients.

If a switch detects a signaling flow, it records the IP address of the client and notifies the other switches of the discovered connection for state synchronization. Thereby, the switches' data planes maintain a synchronized set of active client addresses and match UDP flows with them to detect the flows established between these addresses.

DELTA detects a call initiated between Alice and Bob in a four-step process, as also illustrated in Fig. 1:

(1) The attacker obtains the IP addresses of VoIP applications and their domain names to configure the switches with the signaling information.
(2) The switches identify connections to VoIP servers and store/ distribute the source IP addresses in the data plane. The registered addresses are candidates for upcoming calls as the detected flows such as ⟨$\underline{\tilde{1}}$⟩ and ⟨$\underline{\tilde{4}}$⟩ represent potential signaling connections.
(3) The switches maintain a synchronized global state of registered IP addresses. The distribution of signaling IP addresses occurs entirely in the data plane, as indicated by the green dashed arrows at ⟨$\underline{\tilde{2}}$⟩ and ⟨$\underline{\tilde{5}}$⟩.
(4) The switches locally check the UDP traffic and match the corresponding IP addresses against those registered in the previous step. Thereby, the attacker identifies VoIP calls in-path, as portrayed in ⟨$\underline{\tilde{8}}$⟩.

We leverage programmable data planes in three novel ways:

First, DELTA inspects the `Client-Hello` packets [68] – the first packet in the TLS handshake – destined for VoIP application addresses to check on the VoIP domain names. Since the SNI field is of variable length and its position within the `Client-Hello` packet is unfixed, we iteratively parse the packet and decode the domain name with programmable match-action tables over multiple pipeline stages.

Second, since DELTA requires state distribution for signaling addresses, we implement a data-plane broadcasting mechanism to distribute signaling events throughout the network. Programmable packet parsers enable us to develop our custom dissemination protocol and update the network before the call has already begun.

Third, DELTA overcomes the scalability challenges posed by the global state storage by implementing a probabilistic data structure called Double Buffering Bloom Filter. DBBF maintains the recently seen IP addresses in a scalable manner, *decays* the aged elements, and allows queries at constant processing time. Programmable arithmetic logic units and random-access registers enable DBBF's implementation.

3 DESIGN AND IMPLEMENTATION

In this section, we first review the high-level system design for DELTA (§3.1). Then, we elaborate on the technical challenges of implementing DELTA under the constraints and limitations of programmable switches (§3.2). In the remaining, we delve into the low-level details of identifying signaling users (§3.3), distributing the identified addresses (§3.4), and storing the addresses continuously and scalably (§3.5) – all in the data plane. Finally, we explain how the switches detect the VoIP call flows in-path and in real-time (§3.6).

3.1 High-level architecture

A DELTA switch essentially has two distinct workflows: *(i)* identifying and distributing signaling connections and *(ii)* detecting call flows. Namely, an IP packet has two pipeline routines, as illustrated in Fig. 2 with the orange-colored and the blue-colored process paths:

The orange-colored path illustrates the signaling connection discovery and dissemination of these addresses over the network. Since it is common for VoIP applications to run on virtual-hosting services such as Microsoft Azure [12], they share IP addresses with other applications. Thereby, DELTA applies three filters to match beyond their IP addresses. First, the switch checks whether a packet's destination IP address matches those of target VoIP servers. Second, the switch opportunistically inspects the TLS packet type to identify the `Client-Hello` packets. Third, if the packet is a `Client-Hello` packet, the switch examines the packet content to match the string representing the domain name in the SNI field. In case the packet contains a domain name that belongs to a VoIP service, the switch classifies the packet as part of a signaling connection. In this case, the switch stores and disseminates the packet's source IP address as a potential caller or callee with the destination VoIP service. The storage is done by insertion to the DBBF. The dissemination to other switches is done by cloning the original packet and multicasting the clone in the data-plane broadcast topology.

The blue-colored path illustrates the call flow detection. The switch queries every UDP packet against the DBBF to see if the packet's source and destination IP addresses have both been previously registered in the DBBF as potential signaling connections. If so, the packet is identified as a streaming packet. Thus, the detection is complete: the source and the destination establish a direct call stream, and the switch takes the necessary action.

While the idea is conceptually simple, implementing this procedure at scale is challenging because it needs to *(i)* process all packets to distinguish those that belong to VoIP calls and *(ii)* store state across packets and among switches to recognize potential callers and callees.

3.2 Implementation challenges

We implement DELTA in an Intel Tofino switch [42] in around 1500 lines of P4 code.[3] We briefly introduce programmable data planes and summarize the constraints and limitations that impact our design.

A primer on programmable data planes. Programmable data planes with PISA architecture contain five main components: parser, ingress pipeline, traffic manager, egress pipeline, and deparser [16]. Once a packet arrives, the switch parses the packet headers and streams the parsed representation through the pipeline. Afterward, the packet traverses an ordered series of pipeline stages with match-action tables, memory registers, and arithmetic logic units.

The match-action tables can match (read) the parsed representation of packets and additional metadata such as the packet's ingress port number and apply actions on top of them. The actions can modify the parsed headers and packet metadata and read from or write to stateful memory *registers*.

Typically, each packet streams through the pipeline exactly once. However, it is possible to circulate packets within the switch for additional processing, referred to as *recirculation*. The recirculated packets pass through a dedicated port with fixed throughput, called a *recirculation port*. Lastly, it is possible to duplicate a packet to create a *clone* of a packet and process the clone independently.

Hardware constraints. While programmable data planes offer functional flexibility, they still need to process packets at the line rate of their throughput. That, combined with the limited resources, results in some constraints. More specifically, the following restrictions impact our design decisions:

- ***R1*** The number of pipeline stages limits the number of non-parallelizable actions available per packet.
- ***R2*** The available memory per stage is fixed and limits the amount of data storage.
- ***R3*** A packet can only access a memory block once and only when it is at the stage containing that block, and that restricts the feasibility of stateful algorithms.

In the following, we mention these three restrictions for the places where they appeared as design challenges.

3.3 Identifying signaling packets

As the orange-colored flow in Fig. 2 illustrates, packets go through several filters before the switch identifies them as signaling connections. First, switches filter signaling packets by their transport protocol type (TCP). Second, they filter the destination addresses for the VoIP applications' hosting addresses. Since these addresses can host multiple applications

[3] P4 [15] is a domain-specific language for programming packet forwarding platforms.

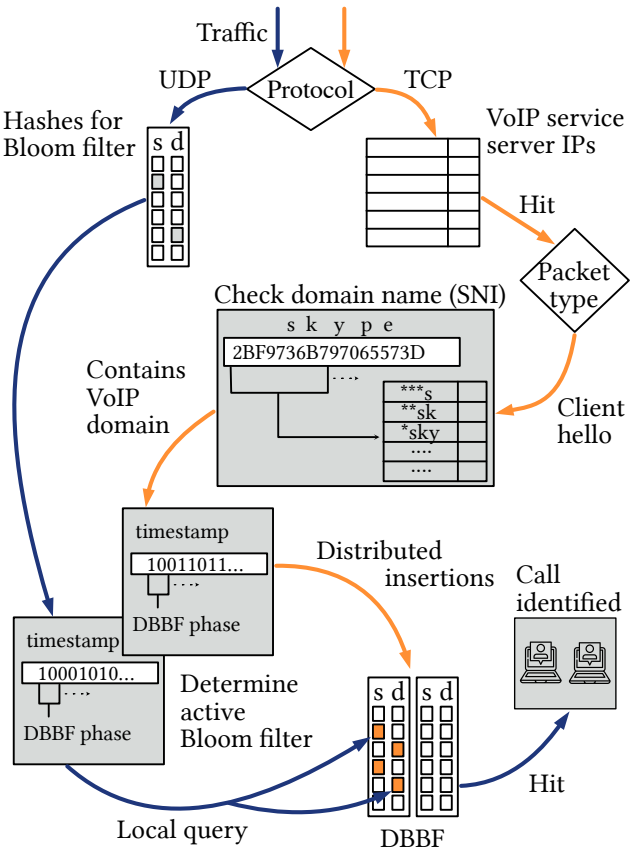


Figure 2: DELTA workflow. The orange-colored flow identifies and stores the addresses undergoing a signaling routine, whereas the blue-colored flow checks potential call flows against the stored addresses.

virtually (e.g., Signal uses Microsoft Azure [12]), switches also inspect the domain name to identify the applications.

We identify signaling packets by checking three properties (see Fig. 2): *(i)* the destination is a known VoIP service's server; *(ii)* the packet contains a TLS `Client-Hello` message; and *(iii)* the requested domain name corresponds to the targeted VoIP service.

Signaling packets have known destinations. In the first filtering step, we look at the packet's destination IP address and check whether it belongs to a known VoIP service's set of IP addresses. However, via virtual hosting, VoIP services such as Skype [88] and Signal [12] share these IP addresses with other applications. Therefore, merely checking these hosting infrastructures' IP addresses will falsely identify connections to other services. As we describe, DELTA examines the TLS `Client-Hello` packet headers to verify the domain name of the sought VoIP applications.

Signaling flows begin with Client-Hello. Signaling connections belonging to any VoIP service are TLS-encrypted. We leverage that the first TLS packet is a `Client-Hello` handshake message in the second filtering step. Since virtual-hosting front-end servers need to read the application domain name to know how to process it, these packets carry the domain name in their SNI fields. After parsing the IP and TCP headers, the switch opportunistically parses the TLS record header to check if a packet carries a TLS `Client-Hello` message. If the location corresponding to the record header type is "0x16" and the handshake type is "0x01", the switch deduces the packet contains a `Client-Hello` message.

Client-Hellos' contents reveal the service. In the last step, we analyze the content of packets that pass the previous steps and contain `Client-Hello` messages. We leverage that a `Client-Hello` of a signaling packet includes the VoIP application's domain name. For example, Skype connections carry hostnames such as "*.broadcast.skype.com" in the SNI field of the `Client-Hello` packets [88].

Parsing domain names is challenging in data planes because they parse packets as fixed-length bitstrings. However, the TLS header consists of several optional *extensions* of variable length and in no particular order [33]. Therefore, strings in extension fields, precisely the SNI field, can occur at any offset within the packet. We use the technique developed by Jepsen et al. [44] to perform string matching only on the `Client-Hello` packets that passed the previous two filters. Namely, we convert string patterns into deterministic finite automatons (DFAs) and store the DFA state transitions in match-action tables. In our case, the state machine works at the byte-level granularity, operates on ASCII characters as the SNI field is encoded in plain text, and maps on to the consecutive stages of the pipeline in match-action tables.

The string search process works as follows: we convert patterns representing domain names into p-stride DFAs, where p is the number of characters matched per transition. We then map this DFA into a match-action table. We replicate the transition table on multiple stages to achieve multiple transitions per pipeline pass. In each pipeline pass, the transition tables can search a fixed number of bytes: p times the number of pipeline stages equipped with DFA tables. As the number of stages is limited (***R2***), we need multiple pipeline passes to search deeper into the packet. Therefore, the switch initially clones the packet and forwards the original packet, as usual, to not delay its transmission. Meanwhile, the cloned packet recirculates within the switch to emulate multiple pipeline passes. The switch drops the searched blocks from the cloned packet at the end of each pipeline pass and attaches the current state of the DFA to the remaining packet for its next pass. In the next pass, the switch can parse deeper into the packet as the packet gets trimmed down in each pass. If the string search succeeds, the switch goes on with the dissemination and storing procedures.

3.4 Disseminating signaling packets

Once a switch successfully classifies a cloned packet as belonging to a signaling flow (§3.3), it disseminates the event throughout the network for each switch to store the source address and the service type. We assume the network topology is static and known in advance, but we do not consider a specific network topology. Therefore, we choose "broadcasting on a spanning tree" to provide a general approach to our dissemination mechanism.

Broadcasting on a spanning tree. We assign a root-ID to each DELTA switch that runs the detection algorithm. Each switch maintains spanning trees by a match-action table that maps root-IDs of all switches to multicasting groups. This table forwards the incoming messenger packets to children switches entirely in the data plane. If a switch detects a signaling event, the messenger packet travels through the specific spanning tree that maps to this switch's root-ID with the root-ID attached to the packet.

As an asynchronous broadcast algorithm, this mechanism has a message complexity of $n - 1$ (n being the number of switches) and a time complexity of d, assuming a rooted spanning tree with depth d. Since we do not consider a specific network topology, we can not make precise statements on how long the packet propagation takes. However, in §4.2, we measure the data-plane processing and transmission delays to show that a data plane algorithm takes negligible time compared to a software procedure (i.e., VoIP protocols in our case) regardless of d. Likewise, we show the message complexity is also negligible compared to the maximum number of signaling events DELTA can store in the data plane. Thereby, the two metrics are not bottlenecks for our design.

3.5 Storing the signaling addresses

After detecting signaling packets, we need to store their source IP addresses in a data structure to check whether an incoming UDP packet belongs to a VoIP call. The data structure needs to be space-efficient to adhere to the memory constraints of programmable data planes (***R1***), support queries in constant time to facilitate real-time detection, and support element decay to only keep source addresses that recently communicated to a VoIP server.

Double Buffering Bloom Filter. The DBBF belongs to the Bloom Filters (BF) family, which are space-efficient, constant-time, probabilistic data structures designed to answer set membership queries. Unlike regular BFs, DBBFs support *element decay* by operating in an active/ standby scheme to evict the stale elements in a first-in, first-out manner [24].

More specifically, the data structure consists of two equal and independent BFs, namely BF-A and BF-B, where one is in active mode while the other is in standby mode. The two BFs swap modes periodically after every *flush interval*. The active BF of each flush interval stores all signaling packets' IP addresses that arrived during this flush interval by encoding the IP addresses into BF indexes and setting the respective BF entries based on the VoIP service type. The standby BF stores only the packets that arrived in the second half of the current flush interval, namely the most recent ones. Consequently, as the swap occurs, the new active BF already contains the entries that have arrived in the last half flush interval.

We illustrate the operation of a representative DBBF in Fig. 3. DBBF works in four phases, with a flush action in every two phases. In `Phase 00`, two packets with IP addresses "1.1.1.1" and "2.2.2.2" arrive while BF-A is in the active mode. Thus, they both get inserted into the BF-A. Insertion into a BF works by hashing the address with multiple distinct hash functions – two hash functions in Fig. 3 – and setting the entries for the resulting indexes. The third packet with the source IP address "3.3.3.3" arrives during `Phase 01`, triggering an insertion into BF-A and BF-B.

During the first half of each flush interval, the controller flushes the standby BF. For instance, in Fig. 3 in `Phase 10`, the BFs swap modes. Thereby, the BF-B becomes the active BF, and the controller flushes the standby, i.e., BF-A. That means, unlike the first two packets' IP addresses, the third packet's IP address will remain in the DBBF after the swap-and-flush operation. Consequently, at any point in time, the DBBF contains the IP addresses corresponding to packets that arrived within the past two phases.

Keeping phase of DBBF in the data plane. The data plane uses packets' timestamps to determine the current phase. The phase dictates which BF(s) the switch should use for insertions and queries. The switch can use any two consecutive bits of the timestamp for phasing. Assume we use the two least significant bits of each packet timestamp. The first two packets in Fig. 3 arrive by a timestamp ending with 00, as the phase indicates. Upon arrival, this two-bit field matches on a phase table, defining the phase, and the packets get inserted into BF-A. The bits' position and the granularity of the timestamp determine the flush interval of the DBBF as the bits will flip based on the time value they represent. For instance, in the mentioned example, the flush interval would be every time the left-bit flips.

Storing the service type. In Fig. 3, each entry represents an eight-bit vector. As the switch detects a signaling packet, it encodes the packet's service type into eight-bit metadata. We use one-hot encoding, a vector representation of service types where each column in the array represents a distinct service. As the packet's source address is hashed into

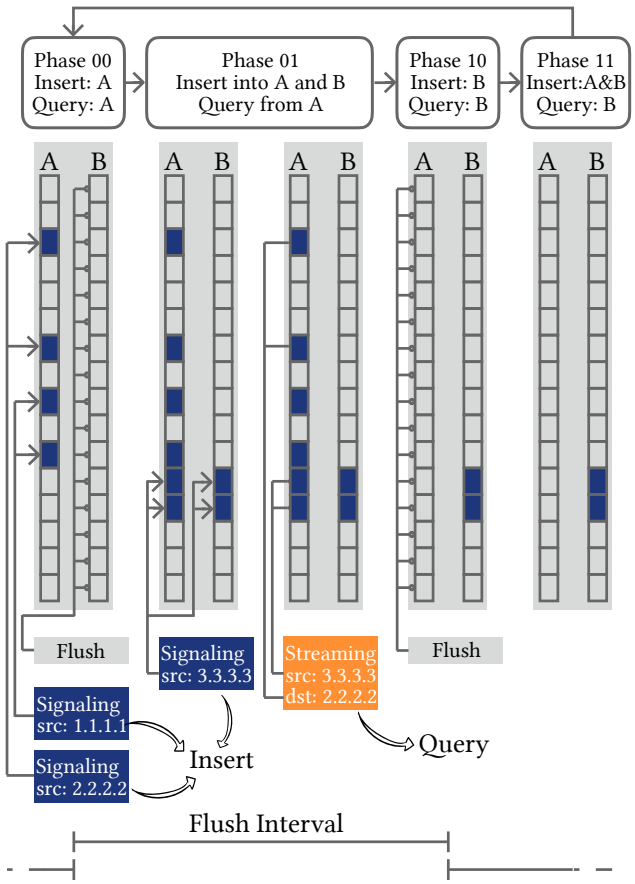


Figure 3: The DBBF operation in four phases. At any point in time, the DBBF preserves the insertions done within the last two phases.

the DBBF, this vector metadata is `XORed` into the respective indexes. Thereby, the DBBF preserves the packets' service types. In Fig. 4, we see two Skype packets getting hashed into four entries of the DBBF by flipping the first column bits of each entry. Meanwhile, the DBBF already contains Telegram, Line, and Viber insertions from previous insertions.

DBBF maintenance. The controller periodically flushes the standby BF. Flushes are asynchronous but must occur during the flush interval's first half. The switch inserts each packet's timestamp into a register for the controller to know which BF to flush. Since the register contains the most recent packet's timestamp, the controller reads this register at a high rate[4] and flushes the corresponding BF when the register's phase-bit flips.

[4]Since the flush interval is on the order of seconds in our use case, the control plane keeps up with the bit flips in our experiments.

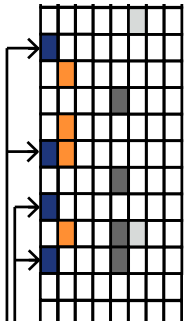
Mapping DBBFs to hardware. Implementing DBBF in the data plane is challenging since accessing a memory block multiple times is not possible (***R3***). The design comes with two shortcomings. First, the switch can not set or read multiple indexes (i.e., one for each hash output) corresponding to a BF (***R3***). Second, when a potential call packet arrives, the switch can not check whether both its source and destination addresses are in the BF in one pipeline pass (***R3***). We split each BF into k segments to address the first shortcoming and place each segment in a separate pipeline stage with an independent register block and a single hash function. To address the second shortcoming, we replicate the entire DBBF to have two identical copies to query one copy for the source IP address and the other for the destination IP address. In other words, we have four BF registers: two copies of the two sides (i.e., BF-A and BF-B) of the DBBF. As a result, the DBBF's capacity, i.e., the number of insertions it can support, is the bottleneck in our design due to other restrictions related to storage capacity (***R1***) and the number of stages (***R2***). We evaluate this bottleneck in §4.2.

3.6 Identifying peer-to-peer VoIP traffic

VoIP calls use UDP as their transport layer protocol [27, 66]. Therefore, we filter UDP packets as potential VoIP traffic and check for each UDP packet whether it belongs to a connection between potential VoIP call participants whose IP addresses are stored in the same columns, i.e., the service type of the BF (see Fig. 4). In particular, we independently compute the same k hash values (i.e., BF indexes) from the packet's source and destination addresses and check if all the bits of those indexes of the active BF are simultaneously set at any column. We `AND` the metadata fields extracted from each k segment of the DBBF to detect a column that is set at every segment. As an illustration, in Fig. 3, when a potential streaming packet arrives (with source IP "3.3.3.3" and destination IP "2.2.2.2"), we check whether both its source and destination addresses correspond to IP addresses that have been previously inserted into the active BF. If both IP addresses are in the BF (the service type columns are omitted in Fig. 3), as in the figure, we know that the two hosts had sent signaling packets before, and we report the IP pair as an identified VoIP call with its service type.

4 EVALUATION

We test DELTA's hardware implementation on a 6.5 Tbps throughput Intel Tofino switch [42]. Our experiments focus on three aspects where the switches' performance directly impacts DELTA's effectiveness. More specifically, we study the following questions: *(i)* how many `Client-Hellos` can the switch inspect in one second; *(ii)* what the processing delay for discovering and disseminating the signaling packets is; and *(iii)* how many signaling addresses can a switch store with its limited memory resources.



Column	Service
1.	Skype
2.	Telegram
3.	Signal
4.	Whatsapp
5.	Line
6.	Viber
7.	n/a
8.	n/a

Figure 4: The DBBF arrays. Each BF entry consists of eight bits, where each bit represents a target service.

We first explain how we collect, synthesize, and generate our data set (§4.1). Then, we present experimental results and findings in light of our stress tests in hardware (§4.2).

4.1 Experimental setup

We build our experiments in three steps. First, we extract the publicly available VoIP information required to perform the attack, which we call reconnaissance. Then, we synthesize our data from real-life calls collected among volunteering participants. Finally, we generate enough traffic from this data set to stress the hardware in our lab testbed.

Reconnaissance. DELTA requires two publicly available information for each VoIP application. First, the possible set of IP addresses potentially hosting the service; second, the domain name we search for in the `Client-Hello` SNI field.

We consider Signal, Skype, Telegram, and WhatsApp for our experiments. We select all Amazon Web Services [4], and Microsoft Azure [12] IP addresses for Signal, as the application runs on both cloud providers. For Skype, we use Skype IP addresses announced explicitly by Microsoft [88]. In Telegram's case, we consider all the IP addresses advertised from the `AS62041`, Telegram's Autonomous System [67]. Finally, for WhatsApp, we use WhatsApp IP addresses provided by Meta [34]. Observe that this information is subject to change, but the core idea remains the same: the information is publicly available. In practice, attackers can also parse these services' IP address-domain name mappings from their authoritative DNS servers. For service names, we leverage the directions for network administrators to enable allow-lists for the respective services. For example, Microsoft asks operators to allow "*.broadcast.skype.com", "*.secure.skypeassets.com", "*.mstea.ms", among others, for Skype operations [88]. We exclusively search for these strings on the `Client-Hello` packets destined to the traced IP addresses.

Traffic synthesis. Our experiments require two types of traffic. First, we need hundreds of thousands of VoIP triangles (i.e., the three connections of a VoIP call) to measure the formation of hash collisions in the DBBF. This test reveals how many address insertions it takes to start obtaining false positives for arbitrary – non-VoIP – UDP flows. Second, we need millions of `Client-Hello` packets randomly distributed in size and service name to test how many packet inspections it takes to exhaust the recirculation port. We use the CAIDA traces [20, 21] to obtain this `Client-Hello` data set. Lastly, we base performance-related tests on these data.

For the realistic VoIP triangles, we follow three steps: First, we collect a small set of VoIP traffic by running Signal, Skype, Telegram, and WhatsApp calls between lab members, including overseas holidays and conferences for higher variation. Technically, these calls include WiFi to WiFi, mobile to mobile, and WiFi to mobile user connections, along with different cases behind firewalls or NATs in the case of WiFi connections. In these experimental calls, we observed mobile users represented by their public IP addresses, whereas WiFi users represented by their gateways' public IP addresses. In deployments where NAT traversal is not possible (such as cases in which networks use symmetric/restricted NATs), and consequently the application relays the VoIP call [54], we do not observe the triangular connection pattern. We therefore exclude relayed calls from the traces used in our evaluation, as DELTA does not apply to relayed calls. Second, we expand the original data set by generating new triangles with random user IP address tuples. Eventually, the process results in each recorded and synthesized call appearing as an independent VoIP call. We do not change the traffic shape as DELTA is oblivious to traffic characteristics. Still, we uniformly distribute the initial call formation (i.e., the time it takes for the call flow to appear) in the range of delays we observed. Finally, we combine the VoIP traffic with background traffic to measure the hash collisions in the DBBF. The background traffic consists of UDP packets with uniformly random source and destination addresses and blends in with the call flows coming from VoIP traffic.

Traffic generation. We use TRex [1] to replay the synthesized traffic, i.e., the VoIP calls and the background traffic. We control the traffic rate by adapting the number of generated flows to a single DBBF flush interval. TRex generates the traffic uniformly since we do not have an apparent reason to assume nonuniformity for short refresh intervals. If we describe the process with an example, suppose the VoIP calls rate is 100 000 calls (i.e., 100 000 triangles), and the background UDP flows are 1 000 000, both per flush interval. Then, by the time half of the flush interval is over, the switch observes 50 000 calls and 500 000 background UDP flows.

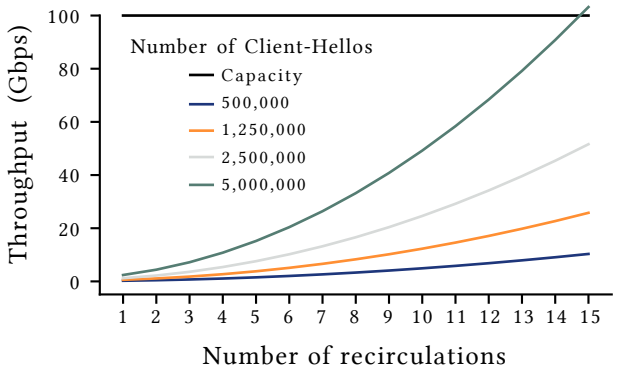


Figure 5: Recirculation port stress measurement. In the worst case, the switch can inspect 5 million packets for the 100 Gbps recirculation port capacity.

4.2 Experiment results

DELTA's effectiveness depends on hardware performance over three aspects. First, discovering signaling packets imposes a bandwidth overhead as the switch recirculates the cloned signaling packets multiple times. Since the dedicated recirculation port has a maximum capacity of 100 Gbps, it sets an upper bound on the number of possible packet inspections. Second, signaling discovery and dissemination both induce process time and overhead. That is because inspecting the signaling packets take multiple recirculations, and dissemination requires attaching (for the sender) and parsing (for the receiver) the messenger packets' headers. The delay packets experience in this process may impede the attack's reaction time. Third, storing signaling IP addresses require limited data plane memory. As memory dictates the allocated space for the DBBF, the switch memory directly impacts the attack's capacity.

Millions of packet inspections per second. Switches perform signaling inspections over all the `Client-Hello` packets destined to the IP addresses of a VoIP application. The number of total connections can be much higher than VoIP calls due to virtual hosting with other applications or in-app multiplexing (e.g., Skype calls, messages, and file transfers all connect to Skype servers). For this reason, we instead measure how many inspections a single recirculation port can support under different packet sizes.

DELTA consumes the whole clone packet before concluding the packet is indeed *not destined* for a VoIP application. We use CAIDA traces [20, 21] to reason about the size of `Client-Hello` packets. We find that the `Client-Hello` packet size varies between 250 and 600 bytes. Specifically, in the worst-case scenario, a packet requires 15 rounds of recirculations. The black line in Fig. 5 shows that the recirculation

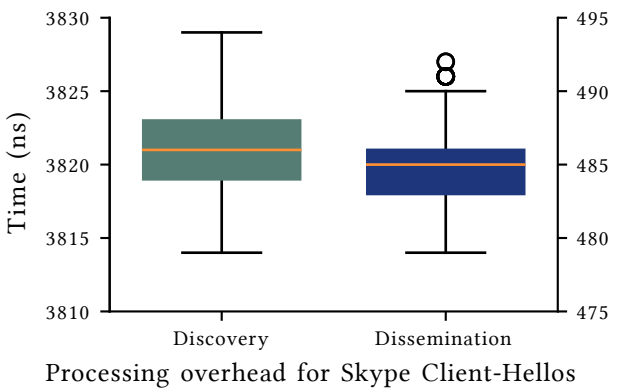


Figure 6: Processing overhead for Skype Client-Hello packets. The discovery process takes on average 3820.7 ns, while the dissemination process introduces a latency of 484.5 ns per hop.

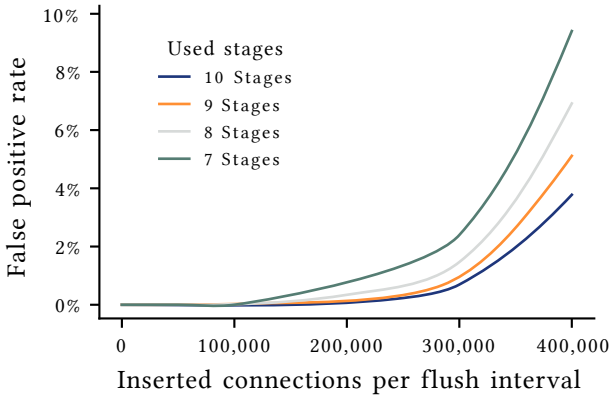


Figure 7: Call density, memory usage, and FPR tradeoff. The DBBF can store 100 000 calls for each flush interval per application without any hash collision.

port's capacity is 100 Gbps. In the worst-case scenario (i.e., 15 recirculations for each packet), a single switch can inspect up to 5 million `Client-Hello` packets per second, as illustrated with the green curve. Note that this inspection process is scalable as parallel switches can perform load-balancing.

Insignificant processing overhead. DELTA does not introduce a processing overhead to the transmission delay of original packets since it only processes on cloned `Client-Hello` packets and messenger packets. The cloned packets undertake two processes: the signaling packet discovery and the dissemination where the messenger packets flood the network. The DFA tables in our implementation occupy 8 (out of 12) pipeline stages. Fig. 6 demonstrates the processing time for 10 000 Skype `Client-Hello` packets. In Skype's case, the packet size is 364 bytes, requiring 4 recirculations until the switch detects the domain name in the SNI field. The mean latency for this process is 3820.7 *ns*. The dissemination step only requires the switch to parse the messenger packet and insert the discovered IP address in the DBBF. The mean latency for this process is 484.8 *ns* for each switch on the flood path. Observe that we do not measure the propagation time between the switches, as this highly depends on the actual topology.

Mechanics behind DBBF false positives. The DBBF false positives occur when a UDP packet's IP address tuple coincides with occupied entries in the DBBF due to hash collisions. The chances of a hash collision increase as the DBBF stores more addresses since each address insertion activates more entries. Thus, the false-positive rate (FPR) depends on the DBBF's occupancy. Eventually, the likelihood of any queried UDP flow falsely hitting the DBBF increases as the number of stored connections in the DBBF grows.

Sensitivity analysis. The attacker can tune the system based on *(i)* the number of pipeline stages to allocate; and *(ii)* the DBBF flush interval. Thereby, we conduct a sensitivity analysis of these two parameters for the false-positive rate.

First, each pipeline stage includes a limited amount of memory and hash function units. Since the DBBF's capacity increases with greater storage and more hash functions to query storage, more pipeline stages imply a bigger DBBF. In other words, for a given number of calls per time unit (i.e., the call density), more pipeline stages result in lower FPR due to fewer hash collisions. Second, the flush interval marks the duration between each reset of the standby BF (§3.5). Two notions restrain the flush interval time: the call formation time and the DBBF occupancy. The switch can only detect the triangular pattern if the call flow appears within half a flush interval of the signaling connections. Otherwise, the controller flushes the signaling event before the call packets hit the DBBF. On the other hand, a longer flush interval results in a higher DBBF occupancy, assuming uniform arrival of application connections. The higher occupancy implies a higher FPR due to more hash collisions. In our experiments, we fix the flush interval and vary the call density, i.e., the number of calls per flush interval, since the flush interval depends on many factors such as geography, VoIP application's ring time, and signaling protocol's keep-alive frequency.

Low DBBF false-positive rates. Fig. 7 illustrates how the FPR changes based on a varying call density. As the call density increases, the FPR increases for a given number of pipeline stages. The distinct curves on the plot represent the number of pipeline stages allocated. The curves show

that the DBBF supports up to 100 000 concurrent signaling connections for a flush interval in all cases. As the number of stages increases, the supported number of simultaneous connections can increase to 400 000 with a reasonable false-positive rate of less than 4%.

No false negatives. By design, the DBBF data structure does not allow false negatives (§3.5). However, hardware failures could still occur in our experiments. Indeed, we have not observed such gray failures, and all real and synthetic VoIP call flows hit the DBBF. Observe that false negatives can occur if the call establishment time exceeds half the flush interval due to faulty tuning of the attack.

5 DISCUSSION

Although DELTA shares similarities with generic traffic classification, the use case requires further discussion, which may concern intricacies that are not necessarily relevant for best-effort operational use cases such as QoS. This section first delves into DELTA's precision and recall and how to improve the attack's potency (§5.1). Then, we discuss potential limitations and explain our position regarding them (§5.2).

5.1 Precision and recall

Mass surveillance and policing is a "needle in the haystack" problem that aims to identify a small number of noteworthy events, such as dissident calls, in a vast pool. In this scenario, a low FPR and a low false-negative rate (FNR) alone are insufficient. Instead, it is also essential to have high precision (i.e., positive predictive value (PPV)) and high recall (i.e., true positive rate (TPR)). High precision defines the percentage of the reported events that are actual VoIP calls:

$$PPV = \frac{\text{Number of reported connections which are calls}}{\text{Number of reported connections}}$$

Whereas high recall marks the percentage of how many actual VoIP calls get predicted out of all the actual VoIP flows within the network:

$$TPR = \frac{\text{Number of reported connections which are calls}}{\text{Number of calls}}$$

A system design may have a very low FPR and FNR, but it may still frequently detect flows that are not VoIP calls if the PPV is low and may not detect many VoIP calls in the network if the TPR is low. We compute a rough estimate of the number of UDP flows per second based on CAIDA traces [3]. We observe the analyzed trace collected from a backbone router's single port contains around 10 000 UDP flows per second in 4.49 Gbps of traffic [22]. Assume a backbone router in our scenario has 64 ports with similar characteristics, adding up to 640 000 UDP flows per second. In the case of a traffic analysis-based classification method with an FPR of 1% – an optimistic assumption – the design would produce more than 5000 false-positive detections per second. That means the system may realistically have more false positives than true positives depending on how many VoIP calls occur within a second. The same reasoning also leads to a system overlooking VoIP calls even in case of a low FNR. Since traffic analysis methods function on probabilistic learning methods, this outcome is part of the design. However, small misses are unacceptable in the context of surveillance and policing, although we can safely tolerate them in other network applications such as traffic prioritization.

DELTA's attacker model assumes full observability over the attacked network, which is realistically the case for our scenario. DELTA theoretically produces no false negatives and positives in this setting, assuming enough capacity for the DBBF. However, in practice, the DBBF is limited by hardware constraints and requires optimization not to waste resources. One way to utilize the DBBF better is to filter its query space with domain-specific knowledge. Since the PPV depends on the "population prevalence", that is, the fraction of VoIP calls within the queried connections, we can filter the UDP space into a subpopulation and only query that to increase the query space's prevalence.

Selecting which connections to query. DELTA eliminates a significant fraction of the UDP traffic by drawing on known distinguishing features of large classes of non-VoIP traffic. Many applications use UDP on known ports, such as DNS (53), NTP (123), and QUIC (80/443). DELTA trivially marks such traffic as non-VoIP within the data plane and does not query these packets in the DBBF. Statistics from CAIDA traces [22] indicate that eliminating DNS and NTP alone reduces the query space by more than 80%. Observe that in practice, VoIP calls run on unknown random ports for operational reasons.

5.2 Limitations

We discuss three limitations of DELTA for completeness purposes. Due to lack of evidence and unavailability of required data, we can not make strong assumptions about the edge cases we mention.

P2P false positives. DELTA draws application-level inference based on network-level connections. One weakness of this approach is when users time multiplex multiple applications. For example, users A and B may simultaneously connect to Skype servers to call users C and D. Meanwhile, users A and B play an online game where they communicate over a P2P UDP stream. This scenario results in an incorrect detection of a Skype call between users A and B.

Unnecessary DBBF insertions. As mentioned in §3.3, our signaling packet filtering mechanism searches for specific domain names to identify signaling packets. However, VoIP platforms usually facilitate multiple purposes such as conference calls, messaging, and file sharing. The same signaling mechanism occurs in all these cases, and DELTA captures all. Specifically, DELTA keeps track of the IP addresses of all the service users instead of VoIP callers, which overloads the DBBF capacity. We can not quantitatively analyze this inefficiency due to the lack of user-base data. Observe that this does not cause incorrect call detections.

Service type collisions. As illustrated in Fig. 4, when a VoIP call is detected, the VoIP service is deduced from the BF arrays. Since the DBBF has no false negatives by design, the correct service name is present in this array. However, this design requires other columns not to have a false positive. If this requirement is not satisfied, the switch returns multiple service types, including the ground truth. Since the BF size is equal for all the columns, but the mentioned false-positive rates depend on the number of insertions for each service type, we can not quantitatively analyze this phenomenon.

6 MITIGATION METHODS

DELTA demonstrates that by leveraging programmable data planes, oppressive entities can practically come up with network attacks. As we intend this work as a wake-up call and by no means condone these activities, we discuss best practices that users can deploy today to evade DELTA.

Hiding IP addresses behind VPNs. DELTA tracks callers' IP addresses by tracing their direct connection in the network. The users can protect themselves from DELTA by hiding their IP addresses via a VPN to relay both their signaling connection with the VoIP servers and their VoIP call flows. In this scenario, the attacker will not be able to capture the P2P link as the call will flow in a VPN tunnel. We tested this methodology manually, and it evades DELTA as expected. The downside of this countermeasure is that most countries subject to mass surveillance and censorship also have difficulty accessing VPN services as these services are legally banned [79]. Thereby, although technically satisfying, this solution may not prove to be practically as effective.

VoIP over Tor. Researchers have recently discovered that VoIP calls over the Tor network are relatively feasible with today's Tor infrastructure performance [78]. They claim that the recent advances in computing and networking now enable a sufficient QoS for VoIP calls which researchers have argued to be not possible multiple times in the past [43, 49, 76]. Community projects such as Orbot [41], Torfone [37], and Donar [17] facilitate VoIP calls over Tor. Again, the downside of these approaches is that the regions that need such services the most are also the regions that can not legally access the Tor network [58].

Opt-in to relayed calls. We tested Signal, Skype, Telegram, and WhatsApp to see if they support relayed mode (see §2.2) for calls on user request. Among all the services, only Signal supports selective relaying of calls [53]. Although relayed calls come with a performance cost [23], privacy-conscious users can take advantage of this preference setting against DELTA. Observe that Signal, known for its privacy-conscious culture, is explicitly banned in various countries [96], hinting at the possibility of existing surveillance attacks.

Application-side mitigations. VoIP service providers could counter DELTA's current implementation by changing the signaling and streaming traffic features that the attack exploits. First, the application can encrypt the SNI field such that the switch cannot identify the VoIP service in the TLS `Client-Hello` packet.[5] Second, the application can increase the delay between call signaling and streaming as the increased time interval would require the attacker to store the signaling state for longer and thus require more memory. However, although such mitigations make DELTA less efficient, the underlying side-channel attack remains.

7 RELATED WORK

A vast amount of VoIP-related security and privacy work exists in the literature. We divide this section into two main categories. First, we discuss VoIP protecting systems where the goal is to protect the VoIP metadata from leaking and anonymizing the users against observing entities. Second, we discuss traffic analysis attacks against VoIP applications.

7.1 VoIP defense

As mentioned in §6, VoIP over Tor is the only solution with existing VoIP software and an underlying anonymization network readily available. However, Tor is vulnerable to traffic analysis attacks due to its asynchronous circuit switching and message passing [48]. Unlike VoIP over Tor, Herd [49] is a metadata-private system specific to VoIP that can also defend against traffic analysis. However, Herd assumes the user connects the system through an honest server.

Loopix [65] hides metadata by relaying messages through multiple mix servers and randomly delaying messages at each hop. However, the high variance latency of Loopix is unsuitable for VoIP calls. Karaoke [47] is a messaging system that guarantees differential privacy for metadata by routing messages through a mixnet and adding noise. Karaoke's minimum end-to-end latency is six seconds, making it unfeasible for VoIP calls. Systems such as Pung [9] and Dissent [93] rely

[5] An IETF Internet-Draft suggests a mechanism for adding this functionality to TLS 1.3, namely TLS Encrypted Client-Hello. [69].

on CPU-intensive cryptographic primitives to protect the metadata and therefore suffer from high latency and poor scalability. Yodel [48] is the first system to hide voice call metadata from an adversary that controls the network and can compromise the servers. As with the other systems mentioned, Yodel is yet to be deployed and must reach a sufficient scale to provide enough bandwidth for many users.

7.2 VoIP attack

Numerous VoIP attacks proposed in the literature uncover caller and callee IP addresses similar to DELTA. Lu et al. [52] proposed to observe the sequence of timestamps of the VoIP packets and used Fourier transforms to correlate the relayed flows with each other. Coskun et al. [28] proposed a locality-sensitive hash algorithm to correlate relayed streaming flows. They use packet sizes and the packet arrival time as inputs to the algorithm. Wang et al. [25, 90] proposed an active correlation attack where the attacker embeds timing watermarks into VoIP flows by slightly delaying packets selected randomly. Zhang et al. [99] proposed a timing attack that works by observing the flows' unique starting and ending times to correlate the flows that belong to the same connection. This line of work requires an off-line analysis of call flows to correlate or process the flow characteristics. Besides, the implied attacks work on relayed calls, which was the industry standard before 2011 (i.e., Microsoft's acquisition of Skype). We believe these works are still crucial against relayed calls and deem DELTA orthogonal to these attacks.

Another line of work is in traffic analysis attacks, where the statistical learning methods apply to the general classification of traffic. Although these papers do not necessarily target VoIP traffic, we assume they can accurately achieve VoIP classification. Moore et al. [56] proposed a Bayesian method to identify application protocols. Zhang et al. [100] devised a robust traffic classification scheme by combining supervised and unsupervised learning. Williams et al. [92] compared five supervised models in their classification accuracy and computation performance. Nguyen et al. [59] trained ML models with various techniques in a way that can classify bi-directional flows. Ye et al. [98] developed a hybrid method with heuristic rules to classify P2P traffic. Sun et al. [82] proposed a traffic classification technique based on transfer learning. Di Mauro et al. [31, 32] analyzed the statistical features of Skype and WebRTC flows by using a decision tree and random forest algorithm and report their results over a set with several hundreds of flows.

In the last three years, a line of work on applying traffic classification based on flow analysis in programmable switch data planes has been evolving. Busse-Grawitz et al. [19] and Lee et al. [50] implemented random forests in the data plane for in-network traffic classification. Xiong et al. [94] implemented various supervised machine learning approaches into a P4-programmable FPGA for in-network traffic classification. Barradas et al. [14] implemented a hybrid-offloading traffic classification approach to detect botnet chatter and fingerprint websites. These works show no evidence of detecting the VoIP service from the call flows or achieving high precision and recall suitable to surveillance or censorship needs. We consider DELTA orthogonal to traffic analysis approaches but with a better focus on our specific use case.

8 CONCLUSION

This work is a wake-up call against threats potentially exacerbated by commercial off-the-shelf programmable switches. We demonstrate the risk by introducing DELTA, a novel network-level side-channel attack that can efficiently identify VoIP calls. DELTA keeps track of the VoIP signaling mechanism used for peer discovery and call setup to reveal the addresses of a call's participants and identifies the VoIP service provider even if the traffic is encrypted.

In summary, we show that programmable data planes allow identifying VoIP calls in-path, in real-time, and at scale by introducing three main components: *(i)* a parsing mechanism that extracts domain names from TLS `Client-Hello` packets; *(ii)* a dissemination technique that broadcasts detected packets within the network; and *(iii)* a space-efficient query data structure that continuously keeps track of the inserted entries' liveness.

Our experiments on real and synthesized VoIP traces show that a single switch can simultaneously hold up to 100 000 unique VoIP calls for each VoIP application and inspect at least 5 million `Client-Hello` packets per second.

ACKNOWLEDGMENTS

The authors would like to thank the reviewers and our shepherd, Amit Sheoran, for their insightful comments. The authors also thank Edgar Costa Molero and Vladimir Gurevich for their valuable feedback.

REFERENCES

[1] 2022. Cisco-System-Traffic-Generator/Trex-Core. Cisco Systems Traffic Generators. https://github.com/cisco-system-traffic-generator/trex-core

[2] BBC . 2016. Turkey Coup Accused 'Traced via Messaging App'. *BBC News* (Aug. 2016). https://www.bbc.com/news/technology-36976693

[3] CAIDA . 2018. The CAIDA Anonymized Internet Traces Dataset (April 2008 - January 2019). https://www.caida.org/catalog/datasets/passive_dataset/

[4] Documentation . 2022. AWS IP Address Ranges - AWS General Reference. https://docs.aws.amazon.com/general/latest/gr/aws-ip-ranges.html

[5] AccessNow. 2022. U.S.-Canadian Firm Sandvine Fosters Russian Censorship Infrastructure. https://www.accessnow.org/sandvine-russian-censorship/

[6] AlJazeera. 2018. Iran Releases Messaging App to Replace Telegram. https://www.aljazeera.com/news/2018/4/26/iran-releases-messaging-app-soroush-to-replace-telegram

[7] AMS-IX. 2022. AMS-IX Amsterdam. https://www.ams-ix.net/ams

[8] AMS-IX. 2022. Total Stats | AMS-IX Amsterdam. https://www.ams-ix.net/ams/documentation/total-stats

[9] Sebastian Angel and Srinath Setty. 2016. Unobservable Communication over Fully Untrusted Infrastructure. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*. USENIX Association, Savannah, GA, 551–569. https://www.usenix.org/conference/osdi16/technical-sessions/presentation/angel

[10] Maria Apostolaki, Gian Marti, Jan Müller, and Laurent Vanbever. 2019. SABRE: Protecting Bitcoin against Routing Attacks. In *26th Annual Network and Distributed System Security Symposium, NDSS 2019, San Diego, California, USA, February 24-27, 2019*. The Internet Society. https://www.ndss-symposium.org/ndss-paper/sabre-protecting-bitcoin-against-routing-attacks/

[11] Netberg Aurora. 2022. Netberg Aurora 750 100G BM Switch Preloaded with ONIE. https://bm-switch.com/index.php/netberg-aurora750-100g-bms.html

[12] Microsoft Azure. 2021. Scaling Secure Enclave Environments with Signal and Azure Confidential Computing. https://customers.microsoft.com/en-us/story/1374464612401582154-signal-nonprofit-azure-security

[13] Alireza Bahramali, Amir Houmansadr, Ramin Soltani, Dennis Goeckel, and Don Towsley. 2020. Practical Traffic Analysis Attacks on Secure Messaging Applications. In *Proceedings 2020 Network and Distributed System Security Symposium*. Internet Society, San Diego, CA. https://doi.org/10.14722/ndss.2020.24347

[14] Diogo Barradas, Nuno Santos, Luís Rodrigues, Salvatore Signorello, Fernando MV Ramos, and André Madeira. 2021. FlowLens: Enabling Efficient Flow Classification for ML-based Network Security Applications.. In *NDSS*.

[15] Pat Bosshart, Dan Daly, Glen Gibb, Martin Izzard, Nick McKeown, Jennifer Rexford, Cole Schlesinger, Dan Talayco, Amin Vahdat, George Varghese, and David Walker. 2014. P4: Programming Protocol-Independent Packet Processors. *SIGCOMM Comput. Commun. Rev.* 44, 3 (July 2014), 87–95. https://doi.org/10.1145/2656877.2656890

[16] Pat Bosshart, Glen Gibb, Hun-Seok Kim, George Varghese, Nick McKeown, Martin Izzard, Fernando Mujica, and Mark Horowitz. 2013. Forwarding Metamorphosis: Fast Programmable Match-Action Processing in Hardware for SDN. In *Proceedings of the ACM SIGCOMM 2013 Conference on SIGCOMM (SIGCOMM '13)*. Association for Computing Machinery, New York, NY, USA, 99–110. https://doi.org/10.1145/2486001.2486011

[17] Yérom-David Bromberg, Quentin Dufour, Davide Frey, and Étienne Rivière. 2022. Donar: Anonymous VoIP over Tor. In *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*. USENIX Association, Renton, WA, 249–265. https://www.usenix.org/conference/nsdi22/presentation/bromberg

[18] Sean Buckley. 2017. Google Cloud, Barefoot Networks Create P4 Runtime Open Source Project. https://www.fiercetelecom.com/telecom/google-cloud-barefoot-networks-create-p4-runtime-open-source-project

[19] Coralie Busse-Grawitz, Roland Meier, Alexander Dietmüller, Tobias Bühler, and Laurent Vanbever. 2019. pForest: In-Network Inference with Random Forests. https://doi.org/10.48550/arXiv.1909.05680 arXiv:1909.05680 [cs]

[20] CAIDA. 2018. Passive Monitor: Equinix-Chicago. https://www.caida.org/catalog/datasets/monitors/passive-equinix-chicago/

[21] CAIDA. 2018. Passive Monitor: Equinix-Nyc. https://www.caida.org/catalog/datasets/monitors/passive-equinix-nyc/

[22] CAIDA. 2018. Trace Statistics for CAIDA Passive OC48 and OC192 Traces. https://www.caida.org/catalog/datasets/trace_stats/

[23] Amarnath Chakraborty. 2021. How to Always Relay Calls to Hide IP Address in Signal. https://techviral.net/signal-always-relay-calls/

[24] F. Chang, Wu-chang Feng, and Kang Li. 2004. Approximate Caches for Packet Classification. In *IEEE INFOCOM 2004*, Vol. 4. 2196–2207 vol.4. https://doi.org/10.1109/INFCOM.2004.1354643

[25] Shiping Chen, Xinyuan Wang, and S. Jajodia. 2006. On the Anonymity and Traceability of Peer-to-Peer VoIP Calls. *IEEE Network* 20, 5 (2006), 32–37. https://doi.org/10.1109/MNET.2006.1705881

[26] Selina Cheng. 2021. Hongkongers Quit WhatsApp and Flock to Signal as Privacy Watchdog Seeks Extra Time for Users to Decide on New Terms. https://hongkongfp.com/2021/01/12/hongkongers-quit-whatsapp-and-flock-to-signal-as-privacy-watchdog-seeks-extra-time-for-users-to-decide-on-new-terms/

[27] Cisco. 2001. Quality of Service for Voice over IP. https://www.cisco.com/c/en/us/td/docs/ios/solutions_docs/qos_solutions/QoSVoIP/QoSVoIP.html

[28] Baris Coskun and Nasir Memon. 2010. Tracking Encrypted VoIP Calls via Robust Hashing of Network Flows. In *2010 IEEE International Conference on Acoustics, Speech and Signal Processing*. 1818–1821. https://doi.org/10.1109/ICASSP.2010.5495398

[29] Trisha Datta, Nick Feamster, Jennifer Rexford, and Liang Wang. 2019. SPINE: Surveillance Protection in the Network Elements. In *9th USENIX Workshop on Free and Open Communications on the Internet, FOCI 2019, Santa Clara, CA, USA, August 13, 2019*, Susan E. McGregor and Michael Carl Tschantz (Eds.). USENIX Association. https://www.usenix.org/conference/foci19/presentation/datta

[30] Ryan Devereaux and Glenn Greenwald. 2014. The NSA Is Recording Every Cell Phone Call in the Bahamas. https://theintercept.com/2014/05/19/data-pirates-caribbean-nsa-recording-every-cell-phone-call-bahamas/

[31] Mario Di Mauro and Maurizio Longo. 2014. Skype Traffic Detection: A Decision Theory Based Tool. In *2014 International Carnahan Conference on Security Technology (ICCST)*. 1–6. https://doi.org/10.1109/CCST.2014.6986975

[32] Mario Di Mauro and Maurizio Longo. 2015. Revealing Encrypted WebRTC Traffic via Machine Learning Tools. In *2015 12th International Joint Conference on E-Business and Telecommunications (ICETE)*, Vol. 04. 259–266.

[33] Donald E. Eastlake 3rd. 2011. *Transport Layer Security (TLS) Extensions: Extension Definitions*. Request for Comments RFC 6066. Internet Engineering Task Force. https://doi.org/10.17487/RFC6066

[34] Facebook. 2022. Set Up and Debug Your Network - WhatsApp Business On-Premises API - Documentation. https://developers.facebook.com/docs/whatsapp/guides/network-requirements/

[35] Ryan Gallagher. 2020. Sandvine's Technology Used for Web Censoring in More Than a Dozen Nations. https://financialpost.com/pmn/business-pmn/sandvines-technology-used-for-web-censoring-in-more-than-a-dozen-nations

[36] Ryan Gallagher. 2020. U.S. Company Faces Backlash After Belarus Uses Its Tech to Block Internet. *Bloomberg.com* (Sept. 2020). https://www.bloomberg.com/news/articles/2020-09-11/sandvine-use-to-block-belarus-internet-rankles-staff-lawmakers

[37] Van Gegel. 2022. TORFone: Secure VoIP Tool. http://torfone.org/

[38] Albert Gran Alcoz, Martin Strohmeier, Vincent Lenders, and Laurent Vanbever. 2022. Aggregate-Based Congestion Control for Pulse-Wave DDoS Defense. In *ACM SIGCOMM*. Amsterdam, The Netherlands.

[39] Glenn Greenwald. 2013. NSA Collecting Phone Records of Millions of Verizon Customers Daily. https://www.theguardian.com/world/2013/jun/06/nsa-phone-records-verizon-court-order

[40] Glenn Greenwald and Ewen MacAskill. 2013. NSA Prism Program Taps in to User Data of Apple, Google and Others. https://www.theguardian.com/world/2013/jun/06/us-tech-giants-nsa-data

[41] Project Guardian. 2022. Orbot: Proxy with Tor. https://guardianproject.info/apps/org.torproject.android/

[42] Vladimir Gurevich and Andy Fingerhut. 2021. P416 Programming for Intel® Tofino™ Using Intel P4 Studio™.

[43] Stephan Heuser, Bradley Reaves, Praveen Kumar Pendyala, Henry Carter, Alexandra Dmitrienko, William Enck, Negar Kiyavash, Ahmad-Reza Sadeghi, and Patrick Traynor. 2017. Phonion: Practical Protection of Metadata in Telephony Networks. *Proc. Priv. Enhancing Technol.* 2017, 1 (2017), 170–187.

[44] Theo Jepsen, Daniel Alvarez, Nate Foster, Changhoon Kim, Jeongkeun Lee, Masoud Moshref, and Robert Soulé. 2019. Fast String Searching on PISA. In *Proceedings of the 2019 ACM Symposium on SDN Research (SOSR '19)*. Association for Computing Machinery, New York, NY, USA, 21–28. https://doi.org/10.1145/3314148.3314356

[45] Qiao Kang, Lei Xue, Adam Morrison, Yuxin Tang, Ang Chen, and Xiapu Luo. 2020. Programmable In-Network Security for Context-Aware BYOD Policies. In *29th USENIX Security Symposium (USENIX Security 20)*. USENIX Association, 595–612. https://www.usenix.org/conference/usenixsecurity20/presentation/kang

[46] Anthony Wing Kosner. 2012. Will Microsoft's Changes To The Architecture Of Skype Make It Easier To Snoop? https://www.forbes.com/sites/anthonykosner/2012/07/18/did-microsoft-change-the-architecture-of-skype-to-make-it-easier-to-snoop/

[47] David Lazar, Yossi Gilad, and Nickolai Zeldovich. 2018. Karaoke: Distributed Private Messaging Immune to Passive Traffic Analysis. In *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*. USENIX Association, Carlsbad, CA, 711–725. https://www.usenix.org/conference/osdi18/presentation/lazar

[48] David Lazar, Yossi Gilad, and Nickolai Zeldovich. 2019. Yodel: Strong Metadata Security for Voice Calls. In *Proceedings of the 27th ACM Symposium on Operating Systems Principles (SOSP '19)*. Association for Computing Machinery, New York, NY, USA, 211–224. https://doi.org/10.1145/3341301.3359648

[49] Stevens Le Blond, David Choffnes, William Caldwell, Peter Druschel, and Nicholas Merritt. 2015. Herd: A Scalable, Traffic Analysis Resistant Anonymity Network for VoIP Systems. In *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication (SIGCOMM '15)*. Association for Computing Machinery, New York, NY, USA, 639–652. https://doi.org/10.1145/2785956.2787491

[50] Jong-Hyouk Lee and Kamal Singh. 2020. SwitchTree: In-Network Computing and Traffic Analyses with Random Forests. *Neural Computing and Applications* (Nov. 2020). https://doi.org/10.1007/s00521-020-05440-2

[51] Dahlia Lithwick and Steve Vladeck. 2013. Taking the "Meh" out of Metadata. https://slate.com/news-and-politics/2013/11/nsa-and-metadata-how-the-government-can-spy-on-your-health-political-beliefs-and-religious-practices.html

[52] Yuanchao Lu and Ye Zhu. 2010. Correlation-Based Traffic Analysis on Encrypted VoIP Traffic. In *2010 Second International Conference on Networks Security, Wireless Communications and Trusted Computing*, Vol. 2. 45–48. https://doi.org/10.1109/NSWCTC.2010.265

[53] Moxie Marlinspike. 2017. Video Calls for Signal out of Beta. https://signal.org/blog/signal-video-calls/

[54] Philip Matthews, Jonathan Rosenberg, and Rohan Mahy. 2010. *Traversal Using Relays around NAT (TURN): Relay Extensions to Session Traversal Utilities for NAT (STUN)*. Request for Comments RFC 5766. Internet Engineering Task Force. https://doi.org/10.17487/RFC5766

[55] Roland Meier, Vincent Lenders, and Laurent Vanbever. 2022. Ditto: WAN Traffic Obfuscation at Line Rate. In *NDSS Symposium 2022*.

[56] Andrew W. Moore and Denis Zuev. 2005. Internet Traffic Classification Using Bayesian Analysis Techniques. In *Proceedings of the 2005 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS '05)*. Association for Computing Machinery, New York, NY, USA, 50–60. https://doi.org/10.1145/1064212.1064220

[57] Mark Moore. 2020. China Reportedly Using Mobile Networks to Monitor US Users Abroad. https://nypost.com/2020/12/15/china-using-mobile-networks-to-monitor-us-users-abroad-report/

[58] Tejas Nair. 2022. Is It Illegal to Use Tor Browser in 2022? https://cyberwaters.com/is-tor-illegal/

[59] Thuy T. T. Nguyen, Grenville Armitage, Philip Branch, and Sebastian Zander. 2012. Timely and Continuous Machine-Learning-Based Classification for Interactive IP Traffic. *IEEE/ACM Transactions on Networking* 20, 6 (2012), 1880–1894. https://doi.org/10.1109/TNET.2012.2187305

[60] Amelia Nierenberg. 2020. Signal Downloads Are Way Up Since the Protests Began. *The New York Times* (June 2020). https://www.nytimes.com/2020/06/11/style/signal-messaging-app-encryption-protests.html

[61] Ed O'Keefe. 2013. Transcript: Dianne Feinstein, Saxby Chambliss Explain, Defend NSA Phone Records Program. https://www.washingtonpost.com/news/post-politics/wp/2013/06/06/transcript-dianne-feinstein-saxby-chambliss-explain-defend-nsa-phone-records-program/

[62] Eva Papadogiannaki and Sotiris Ioannidis. 2021. A Survey on Encrypted Network Traffic Analysis Applications, Techniques, and Countermeasures. *Acm Computing Surveys* 54, 6, Article 123 (July 2021). https://doi.org/10.1145/3457904

[63] Sarah Perez. 2016. WhatsApp Hits 100 Million Calls per Day. https://social.techcrunch.com/2016/06/24/whatsapp-hits-100-million-calls-per-day/

[64] Billy Perrigo. 2020. The Inside Story of How Signal Became the Private Messaging App for an Age of Fear and Distrust. https://time.com/5893114/signal-app-privacy/

[65] Ania M. Piotrowska, Jamie Hayes, Tariq Elahi, Sebastian Meiser, and George Danezis. 2017. The Loopix Anonymity System. In *Proceedings of the 26th USENIX Conference on Security Symposium (SEC'17)*. USENIX Association, USA, 1199–1216.

[66] Gokul Prabhakar, Rajeev Rastogi, and Marina Thottan. 2005. Oss Architecture and Requirements for VoIP Networks. *Bell Labs Technical Journal* 10, 1 (2005), 31–45. https://doi.org/10.1002/bltj.20077

[67] AS Rank. 2022. AS Rank: AS62041 (Telegram Messenger Inc). https://asrank.caida.org/asns/62041

[68] Eric Rescorla. 2018. *The Transport Layer Security (TLS) Protocol Version 1.3.* Request for Comments RFC 8446. Internet Engineering Task Force. https://doi.org/10.17487/RFC8446

[69] Eric Rescorla, Kazuho Oku, Nick Sullivan, and Christopher A. Wood. 2022. *TLS Encrypted Client Hello.* Internet Draft draft-ietf-tls-esni-14. Internet Engineering Task Force. https://datatracker.ietf.org/doc/draft-ietf-tls-esni

[70] Reuters. 2020. NSA Surveillance Exposed by Snowden Was Illegal, Court Rules Seven Years On. *The Guardian* (Sept. 2020). https://www.theguardian.com/us-news/2020/sep/03/edward-snowden-nsa-surveillance-guardian-court-rules

[71] Adi Robertson. 2013. Phone Spying and PRISM Internet Surveillance: What's the Difference? https://www.theverge.com/2013/6/7/4407782/phone-spying-and-prism-internet-surveillance-whats-the-difference

[72] Ola Salman, Imad H. Elhajj, Ayman Kayssi, and Ali Chehab. 2020. A Review on Machine Learning–Based Approaches for Internet Traffic Classification. *Annals of Telecommunications* 75, 11 (Dec. 2020), 673–710. https://doi.org/10.1007/s12243-020-00770-7

[73] Sandvine. 2022. Advanced Traffic Classification. https://www.sandvine.com/hubfs/Sandvine_Redesign_2019/Downloads/2020 / Whitepapers / Sandvine_WP_Advanced % 20Traffic % 20Classification.pdf

[74] Sandvine. 2022. Deploying Virtualized DPI at Multi-Tbps Scale. https://www.sandvine.com/hubfs/downloads/technology/virtualization/sandvine-wp-deploying-multi-tbps-virtualization.pdf

[75] Sandvine. 2022. Intelligent Traffic Management | Sandvine. https://www.sandvine.com/enterprise/intelligent-traffic-management

[76] David Schatz, Michael Rossberg, and Guenter Schaefer. 2017. Reducing Call Blocking Rates for Anonymous Voice over IP Communications. In *2017 9th International Congress on Ultra Modern Telecommunications and Control Systems and Workshops (ICUMT).* 382–390. https://doi.org/10.1109/ICUMT.2017.8255174

[77] Zane Schwartz. 2020. Sandvine's Technology Used to Censor the Internet in over a Dozen Countries. https://thelogic.co/briefing/sandvines-technology-used-to-censor-the-internet-in-over-a-dozen-countries/

[78] Piyush Kumar Sharma, Shashwat Chaudhary, Nikhil Hassija, Mukulika Maity, and Sambuddho Chakravarty. 2020. The Road Not Taken: Re-thinking the Feasibility of Voice Calling Over Tor. *Proceedings on Privacy Enhancing Technologies* 2020, 4 (Oct. 2020), 69–88. https://doi.org/10.2478/popets-2020-0063

[79] Timothy Shim. 2021. Are VPNs Legal? 10 Countries That Ban the Usage of VPN. https://www.webhostingsecretrevealed.net/blog/security/are-vpns-legal/

[80] Guardian Staff. 2014. Russia's Eavesdropping on Phone Calls Examined by Strasbourg Court. https://www.theguardian.com/world/2014/sep/24/strasbourg-court-human-rights-russia-eavesdropping-texts-emails-fsb-

[81] Steve Stecklow. 2012. Special Report: Chinese Firm Helps Iran Spy on Citizens. https://www.reuters.com/article/us-iran-telecoms-idUSBRE82L0B820120322

[82] Guanglu Sun, Lili Liang, Teng Chen, Feng Xiao, and Fei Lang. 2018. Network Traffic Classification Based on Transfer Learning. *Computers & Electrical Engineering* 69 (July 2018), 920–927. https://doi.org/10.1016/j.compeleceng.2018.03.005

[83] Signal Support. 2022. How Do I Know My Communication Is Private? https://support.signal.org/hc/en-us/articles/360007318911-How-do-I-know-my-communication-is-private-

[84] Telegram Support. 2022. End-to-End Encrypted Voice Calls. https://core.telegram.org/api/end-to-end/voice-calls

[85] WhatsApp Support. 2022. WhatsApp Help Center - About End-to-End Encryption. https://faq.whatsapp.com/general/security-and-privacy/end-to-end-encryption/?lang=en

[86] Telegram Team. 2017. Voice Calls: Secure, Crystal-Clear, AI-Powered. https://telegram.org/blog/calls

[87] Rahul Tripathi. 2016. Dangerous Signal: This Encrypted App Is Helping ISIS Members in India to Communicate - ET Telecom. http://telecom.economictimes.indiatimes.com/news/dangerous-signal-this-encrypted-app-is-helping-isis-members-in-india-to-communicate/51774226

[88] Kelley Vice, Priya Rackshith, and Cern McAtee. 2022. Office 365 URLs and IP Address Ranges - Microsoft 365 Enterprise. https://docs.microsoft.com/en-us/microsoft-365/enterprise/urls-and-ip-address-ranges

[89] Sarah Wagner. 2021. APS Networks® Launches Three TIP OpenBNG Programmable Switches to Boost the Disaggregated Telco Broadband Market – APS Networks. https://www.aps-networks.com/press/aps-networks-launches-three-tip-openbng-programmable-switches/

[90] Xinyuan Wang, Shiping Chen, and Sushil Jajodia. 2005. Tracking Anonymous Peer-to-Peer VoIP Calls on the Internet. In *Proceedings of the 12th ACM Conference on Computer and Communications Security (CCS '05).* Association for Computing Machinery, New York, NY, USA, 81–91. https://doi.org/10.1145/1102120.1102133

[91] Big Brother Watch Team. 2021. UK Mass Surveillance Found Unlawful by Europe's Highest Human Rights Court — Big Brother Watch. https://bigbrotherwatch.org.uk/2021/05/uk-mass-surveillance-found-unlawful-by-europes-highest-human-rights-court/

[92] Nigel Williams, Sebastian Zander, and Grenville Armitage. 2006. A Preliminary Performance Comparison of Five Machine Learning Algorithms for Practical IP Traffic Flow Classification. *SIGCOMM Comput. Commun. Rev.* 36, 5 (Oct. 2006), 5–16. https://doi.org/10.1145/1163593.1163596

[93] David Isaac Wolinsky, Henry Corrigan-Gibbs, Bryan Ford, and Aaron Johnson. 2012. Dissent in Numbers: Making Strong Anonymity Scale. In *10th USENIX Symposium on Operating Systems Design and Implementation (OSDI 12).* USENIX Association, Hollywood, CA, 179–182. https://www.usenix.org/conference/osdi12/technical-sessions/presentation/wolinsky

[94] Zhaoqi Xiong and Noa Zilberman. 2019. Do Switches Dream of Machine Learning? Toward in-Network Classification. In *Proceedings of the 18th ACM Workshop on Hot Topics in Networks (HotNets '19).* Association for Computing Machinery, New York, NY, USA, 25–33. https://doi.org/10.1145/3365609.3365864

[95] Diwen Xue, Reethika Ramesh, Valdik S S, Leonid Evdokimov, Andrey Viktorov, Arham Jain, Eric Wustrow, Simone Basso, and Roya Ensafi. 2021. Throttling Twitter: An Emerging Censorship Technique in Russia. In *Proceedings of the 21st ACM Internet Measurement Conference (IMC '21).* Association for Computing Machinery, New York, NY, USA, 435–443. https://doi.org/10.1145/3487552.3487858

[96] Maria Xynou and Arturo Filastò. 2021. How Countries Attempt to Block Signal Private Messenger App around the World. https://ooni.org/post/2021-how-signal-private-messenger-blocked-around-the-world/

[97] Jing Yang. 2020. WeChat Becomes a Powerful Surveillance Tool Everywhere in China. https://www.wsj.com/articles/wechat-becomes-a-powerful-surveillance-tool-everywhere-in-china-11608633003

[98] Wujian Ye and Kyungsan Cho. 2014. Hybrid P2P Traffic Classification with Heuristic Rules and Machine Learning. *Soft Computing* 18, 9 (Sept. 2014), 1815–1827. https://doi.org/10.1007/s00500-014-1253-5

[99] Ge Zhang and Stefan Berthold. 2010. Hidden VoIP Calling Records from Networking Intermediaries. In *Principles, Systems and Applications of IP Telecommunications (IPTComm '10)*. Association for Computing Machinery, New York, NY, USA, 12–21. https://doi.org/10.1145/1941530.1941533

[100] Jun Zhang, Xiao Chen, Yang Xiang, Wanlei Zhou, and Jie Wu. 2015. Robust Network Traffic Classification. *IEEE/ACM Transactions on Networking* 23, 4 (2015), 1257–1270. https://doi.org/10.1109/TNET.2014.2320577

[101] Menghao Zhang, Guanyu Li, Shicheng Wang, Chang Liu, Ang Chen, Hongxin Hu, Guofei Gu, Qianqian Li, Mingwei Xu, and Jianping Wu. 2020. Poseidon: Mitigating Volumetric Ddos Attacks with Programmable Switches. In *The 27th Network and Distributed System Security Symposium (NDSS 2020)*.