Aggregation-Based Certificate Transparency Gossip

Rasmus Dahlberg, Tobias Pulls, Jonathan Vestin, Toke Høiland-Jørgensen and Andreas Kassler

Karlstad University
Universitetsgatan 2, 651 88 Karlstad, Sweden
email: first.last@kau.se

***Abstract*—Certificate Transparency (CT) requires that every certificate which is issued by a certificate authority must be publicly logged. While a CT log can be untrusted in theory, it relies on the assumption that every client observes and cryptographically verifies the same log. As such, some form of gossip mechanism is needed in practice. Despite CT being adopted by several major browser vendors, no gossip mechanism is widely deployed. We suggest an aggregation-based gossip mechanism that passively observes cryptographic material that CT logs emit in plaintext, aggregating at packet processors (such as routers and switches) to periodically verify log consistency off-path. In other words, gossip is provided as-a-service by the network. Our proposal can be implemented for a variety of programmable packet processors at line-speed without aggregation distinguishers (throughput), and, based on 20 days of RIPE Atlas measurements that represent clients from 3500 autonomous systems, we show that significant protection against split-viewing CT logs can be achieved with a realistic threat model and an incremental deployment scenario.**

Keywords–Certificate Transparency; Gossip; P4; XDP.

I. Introduction

The HyperText Transfer Protocol Secure (HTTPS) ecosystem is going through a paradigm shift. As opposed to blindly trusting that Certificate Authorities (CAs) only issue certificates to the rightful domain owners—a model known for its weakest-link security [1]—transparency into the set of issued certificates is incrementally being required by major browser vendors [2][3]. This transparency is forced and takes the form of Certificate Transparency (CT) logs: the idea is to reject any Transport Layer Security (TLS) certificate that have yet to be publicly logged, such that domain owners can monitor the logs for client-accepted certificates to *detect* certificate mis-issuance *after the fact* [4]. While the requirement of certificate logging is a significant improvement to the HTTPS ecosystem, the underlying problem of trusting CAs cannot be solved by the status quo of trusted CT logs (described further in Section II-A). Therefore, it is paramount that nobody needs to trust these logs once incremental deployments are matured.

CT is formalized and cryptographically verifiable [5], supporting inclusion and consistency proofs. This means that a client can verify whether a log is operated correctly: said certificates are included in the log, and nothing is being removed or modified. Despite the ability to cryptographically verify these two properties, there are no assurances that everybody observes *the same log* [4][6]. For example, certificate mis-issuance would not be detected by a domain owner that monitors the logs if fraudulently issued certificates are shown to the clients selectively. A log that serves different versions of itself is said to present a *split view* [7]. Unless such log misbehaviour can be detected, we must trust it not to happen.

The solution to the split viewing problem is a gossip mechanism which ensures that everybody observes *the same* consistent log [4]. This assumption is simple in theory but remarkably hard in practice due to client privacy, varying threat models, and deployment challenges [7][8]. While Google started on a package that supports minimal gossip [9] and the mechanisms of Nordberg *et al.* [7], there is “next to no deployment in the wild” [10]. To this end, we propose a gossip mechanism that helps detecting split-view attacks retroactively based on the idea of packet processors, such as routers and middleboxes, that *aggregate* Signed Tree Heads (STHs)—succinct representations of the logs’ states—that are exposed to the network *in plaintext*. The aggregated STHs are then used to challenge the logs to prove consistency via an off-path, such that the logs cannot distinguish between challenges that come from different aggregators. Given this indistinguishability assumption, it is non-trivial to serve a consistent split-view to an unknown location [11]. Thus, all aggregators must be on the same view, and accordingly all clients that are covered by these aggregators must also be on the same view *despite not doing any explicit gossip themselves* because gossip is provided as-a-service by the network. An isolated client (i.e., untrusted network path to the aggregator) is notably beyond reach of any retroactive gossip [8].

The premise of having STHs in plaintext is controversial given current trends to encrypt transport protocols, which is otherwise an approach that combats inspection of network traffic and protocol ossification [12][13]. We argue that keeping gossip related material in plaintext to support aggregation-based gossip comes with few downsides though: it is easy to implement, there are no major negative privacy impacts, and it would offer significant protection for a large portion of the Internet with a realistic threat model *despite relatively small deployment efforts*. The three main limitations are no protection against isolated clients, reliance on clients that fetch STHs from the logs in plaintext, and possible concerns surrounding protocol ossification [13]. Our contributions are:

- Design and security considerations for a network-based gossip mechanism that passively aggregates STHs to verify log consistency off-path (Section III).
- Generic implementations of the aggregation step using Programming Protocol independent Packet Processors (P4) [14] and eXpress Data Path (XDP) [15] for plaintext STHs, supporting line-speed packet processing on systems that range from switches, routers, network interface cards, and Linux (Section IV).
- A simulation based on RIPE Atlas measurements that evaluate the impact of deploying aggregation-based

gossip at Autonomous Systems (ASes) and Internet Exchange Points (IXPs). Our evaluation shows that incremental roll-out at well-connected locations would protect a significant portion of all Internet clients from undetected split views (Section V).

Besides the sections referenced above, the paper introduces necessary background in Section II and provides discussion, conclusion, and future work in Sections VI–VIII. A full version with additional implementation details is available online [16].

II. Background

First, additional prerequisites are provided on CT and the status quo, then the techniques which allow us to program custom packet processors are introduced.

A. Certificate Transparency

The main motivation of CT is that the CA ecosystem is error-prone [17]: a CA can normally issue certificates for *any* domain name, and given that there are hundreds of trusted CAs an attacker only needs to target the weakest link [1]. While the requirement of CT logging all certificates cannot prevent mis-issuance proactively, it allows anyone to detect it retroactively by monitoring the logs [4]. After a log promises to include a certificate by issuing a Signed Certificate Timestamp (SCT), a new STH including the appended certificate must be issued within a Maximum Merge Delay (MMD). Typically, logs use 24 hour MMDs. Should non-included SCTs and/or inconsistent STHs be found, binding evidence of misbehaviour exists because these statements are digitally signed by the logs. Other than MMD a log's policy defines parameters such as STH frequency: the number of STHs that can be issued during an MMD, making it harder to track clients [7].

CT is being deployed across Apple's platform [2] and Google's Chrome [3]. The status quo is to trust a CA-signed certificate if it is accompanied by two or more SCTs, thereby relying on at least one log to append each certificate so that mis-issuance can be detected by monitors that inspect the logs. The next step of this incremental deployment is to *verify* that these certificates are logged by querying for inclusion [18], and that the log's append-only property is respected by challenging the log to prove STH consistency. Finally, to fully distrust CT logs we need mechanisms that detect split-views. One such mechanism which is based on programmable packet processors (introduced next) is presented in Section III, and it is compared to related work on CT gossip in Section VI.

B. Programmable Data Planes

Packet processors such as switches, routers, and network interface cards are typically integrated tightly using customized hardware and application-specific integrated circuits. This inflexible design limits the potential for innovation and leads to long product upgrade cycles, where it takes *years* to introduce new processing capabilities and support for different protocols and header fields (mostly following lengthy standardization cycles). The recent shift towards flexible *match+action* packet-processing pipelines—including Reconfigurable Match Tables (RMT) [19], Intel FlexPipe [20], Cavium XPliant packet Architecture (XPA) [21], and Barefoot Tofino [22]—now have the potential to change the way in which packet processing hardware is implemented: it enables programmability using high-level languages, such as P4, while at the same time maintaining performance comparable to fixed-function chips.

1) P4: The main goal of P4 is to simplify programming of protocol-independent packet processors by providing an abstract programming model for the network data plane [14]. In this setting, the functionality of a packet processing device is specified without assuming any hardwired protocols and headers. Consequently, a P4 program must parse headers and connect the values of those protocol fields to the actions that should be executed based on a pipeline of reconfigurable match+action tables. Based on the specified P4 code, a front-end compiler generates a high-level intermediate representation that a back-end compiler uses to create a target-dependent program representation. Compilers are available for several platforms, including the software-based simple switch architecture [23], SDNet for Xilinx's NetFPGA (Field-Programmable Gate Array) boards [24], and Netronome's smart network interfaces [25]. It is also possible to compile basic P4 programs into eBPF byte code [26].

2) XDP: The Berkeley Packet Filter (BPF) is a Linux-based packet filtering mechanism [27]. Verified bytecode is injected from user space, and executed for each received packet in kernel space by a just-in-time compiler. Extended BPF (eBPF) enhances the original BPF concept, enabling faster runtime and many new features. For example, an eBPF program can be attached to the Linux traffic control tool `tc`, and additional hooks were defined for XDP [15]. In contrast to the Intel Data Plane Development Kit (DPDK), which runs in user space and completely controls a given network interface that supports a DPDK driver, XDP cooperates with the Linux stack to achieve fast, programmable, and reconfigurable packet processing using C-like programs.

III. Design

An overview of aggregation-based gossip is shown in Figure 1. The setting consists of logs that send plaintext STHs to clients over a network, and, as part of the network, inline *packet processors* passively aggregate observed STHs to their own off-path *challengers* which challenge the logs to prove consistency. A log cannot present split views to different clients that share an aggregating vantage point because it would trivially be detected by that vantage point's challenger. A log also cannot present a persistent split view to different challengers because they are off-path in the sense that they are indistinguishable from one another. This means that every client that is covered by an aggregator must be on the same view because at least one challenger will otherwise detect an inconsistency and report it. A client that is not directly covered by an aggregator may receive indirect protection in the form of herd immunity. This is discussed in Section VII-D.

A. Threat Model and Security Notion

The overarching threat is undetectable domain impersonation (ex-post) by an attacker that is capable of compromising at least one CA and a sufficient number of CT logs to convince a client into accepting a forged certificate. We assume that any illegitimately issued certificate would be detected by the legitimate domain owner through self or delegated third-party monitoring. This means that an attacker must either provide a split view towards the victim or the monitoring entity. We also assume that clients query the logs for certificate inclusion based on STHs that they acquire from the logs via plaintext mechanisms that packet processors can observe, and that some

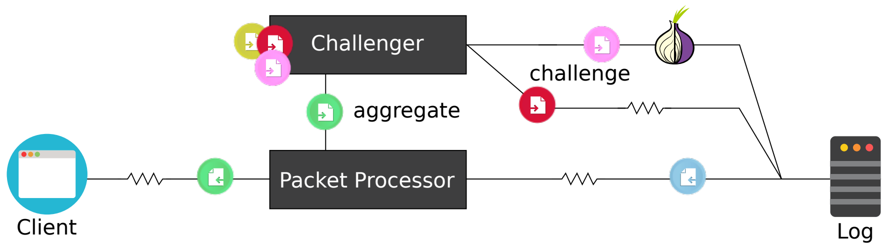


Figure 1. Packet processor that aggregates plaintext STHs for off-path verification.

other entities than challengers process STHs using the chosen off-paths (SectionVII-A). We do not account for the fact that CA compromises may be detected by other means, focusing solely on split-viewing CT logs.

1) Limitations: Our gossip mechanism is limited to STHs that packet processors can observe. As such, a client isolated by an attacker is not protected. We limit ourselves to attackers that act over a network some distance (in the sense of network path length) from a client in plaintext so that aggregation can take place. Our limitations and assumptions are further discussed in Section VII-A.

2) Attackers: Exceptionally powerful attackers can isolate clients, *but clients are not necessarily easy to isolate* for a significant number of relevant attackers. Isolation may require physical control over a device [28], clients may be using anonymity networks like Tor where path selection is inherently unpredictable [29], or sufficiently large parts of the network cannot be controlled to ensure that no aggregation takes place. This may be the case if we consider a nation state actor attacking another nation state actor, the prevalence of edge security middleboxes, and that home routers or network interfaces nearby the clients could aggregate. Any attacker that cannot account for these considerations is within our threat model.

3) Security Notion: To bypass our approach towards gossip, an adaptive attacker may attempt to actively probe the network for aggregating packet processors. This leads us to the key security notion: *aggregation indistinguishability*. An attacker should not be able to determine if a packet processor is aggregating STHs. The importance of aggregation indistinguishability motivates the design of our gossip mechanism into two distinct components: aggregation that takes place inline at packet processors, and periodic off-path log challenging that checks whether the observed STHs are consistent.

B. Packet Processor Aggregation

An aggregating packet-processor determines for each packet if it is STH-related. If so, the packet is cloned and sent to a challenging component for off-path verification. The exact definition of *STH-related* depends on the plaintext source, but it is ultimately the process of inspecting multiple packet headers such as transport protocol and port number. It should be noted that the original packet must not be dropped or modified. For example, an aggregator would have a trivial aggregation distinguisher if it dropped any malformed STH.

For each aggregating packet processor, we have to take IP fragmentation into consideration. Recall that IP fragmentation usually occurs when a packet is larger than the Maximum Transmission Unit (MTU), splitting it into multiple smaller IP packets that are reassembled at the destination host. Normally, an STH should not be fragmented because it is much smaller than the de-facto minimum MTU of (at least) 576 bytes [30][31], but an attacker could use fragmentation to *intentionally* spread expected headers across multiple packets. Assuming stateless packet processing, an aggregator cannot identify such fragmented packets as STH-related because some header would be absent (cf. stateless firewalls). All tiny fragments should therefore be aggregated to account for intentional IP fragmentation, which appears to have little or no impact on normal traffic because tiny fragments are anomalies [32]. The threat of multi-path fragmentation is discussed in Section VII-A.

Large traffic loads must also be taken into account. If an aggregating packet processor degrades in performance as the portion of STH-related traffic increases, a distant attacker may probe for such behaviour to determine if a path contains an aggregator. Each *implementation* must therefore be evaluated individually for such behaviour, and, if trivial aggregation distinguishers exist, this needs to be solved. For example, STH-related traffic could be aggregated probabilistically to reduce the amount of work. Another option is to load-balance the traffic before aggregation, i.e., avoid worst-case loads that cannot be handled.

C. Off-Path Log Challenging

A challenger is setup to listen for aggregated traffic, reassembling IP fragments and storing the aggregated STHs for periodic off-path verification. Periodic off-path verification means that the challenger challenges the log based on its own (off-path fetched) STHs and the observed (aggregated) STHs to verify log consistency periodically, e.g., every day. The definition of *off-path* is that the challenger must not be linkable to its aggregating packet processor(s) or any other challenger (including itself). Without an off-path, there is no gossip step amongst aggregator-challenger instances that are operated by different actors, and our approach towards gossip would only assert that clients behind the same vantage point observe the same logs. If a log cannot distinguish between different challengers due to the use of off-paths, however, it is non-trivial to maintain a targeted split-view towards an unknown location. Therefore, we get a form of *implicit gossip* [11] because at least one challenger would detect an inconsistency unless everybody observes the same log. If every challenger observes the same log, so does every client that is covered by an aggregating packet processor. Notably, the challenger component *does not run inline* to avoid timing distinguishers. Note that there are other important considerations when implementing a challenger, as discussed in Section VII-A.

IV. Distinguishability Experiments

There are many different ways to implement the aggregation step. We decided to use P4 and XDP because a large variety of programmable packet processors support these languages (Section II-B). The aggregated plaintext source is assumed to be CT-over-DNS [33], which means that a client obtains STHs by fetching IN TXT resource records. Since languages for programmable packet processors are somewhat restricted, we facilitated packet processing by requiring that at most one STH is sent per UDP packet. This is reasonable because logs should only have one *most recent* STH. A DNS STH is roughly 170 bytes without any packet headers and should normally not be fragmented, but to ensure that we do not miss any intentionally fragmented STHs we aggregate every tiny fragment. We did not implement the challenging component because it is relatively easy given an existing off-path. Should any scalability issue arise for the challenger there is nothing that prevents a distributed front-end that processes the aggregated material before storage. Storage is not an issue because there are only a limited amount of unique STHs per day and log (one new STH per hour is a common policy, and browsers recognize $\approx$ 40 logs). Further implementation details can be found online [16][34].

A. Setup

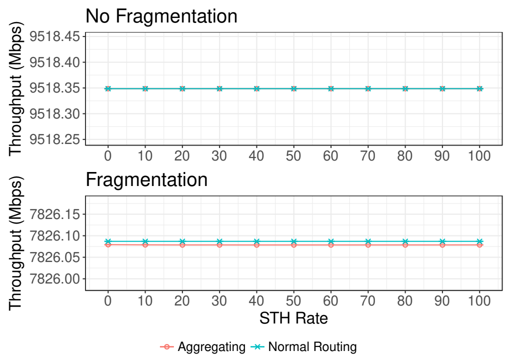
We used a test-bed consisting of a traffic generator, a traffic receiver, and an aggregating target in between. The first target is a P4-enabled NetFPGA SUME board that runs an adapted version of our P4 reference implementation. The second target is a net-next kernel v4.17.0-rc6 Linux machine that runs XDP on one core with a 10 Gb SFP+ X520 82599ES Intel card, a 3.6 GHz Intel Core i7-4790 CPU, and 16 GB of RAM at 1600 MHz (Hynix/Hyundai). We would like to determine whether there are any aggregation distinguishers as the fraction of STHs (experiment 1) and tiny fragments (experiment 2) in the traffic is increased from 0–100%, i.e., does performance degrade as a function of STH-related rate? Non-fragmented STH packets are 411 bytes (we used excessively large DNS headers to maximize the packet parsing overhead), and tiny fragments are 64 bytes. All background traffic has the same packet sizes but is not deemed STH-related.

B. Results

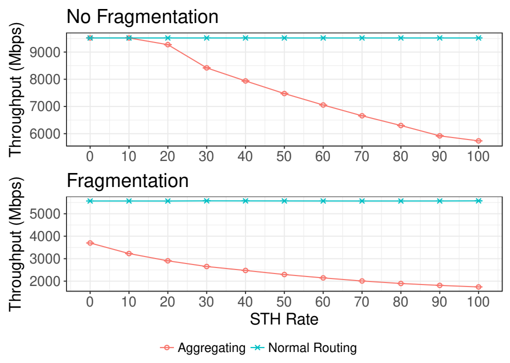
Figure 2a shows throughput as a function of STH-related rate for the P4-enabled NetFPGA. While we were unable to observe any distinguisher between normal routing and the edge case of 100% aggregation for non-fragmented STH packets, there is a small constant throughput difference for tiny fragments (7.5 Kbps). This is a non-negligible *program distinguisher* if a packet processor is physically isolated as in our benchmark, i.e., something other than a routing program is running but it is not necessarily an aggregator because performance does not degrade as a function of increased STH-related rate. However, we found such degradation behaviour for the single-core XDP case (Figure 2b). If line-speed is higher than 2 Gbps, STHs could be aggregated probabilistically or traffic could be load-balanced to *overcome* this issue.

C. Lessons learned

P4-NetFPGA provides aggregation indistinguishability regardless of STH load. For XDP, it depends on the scenario:



(a) P4 NetFPGA



(b) XDP on a single core

Figure 2. Throughput as a function of STH-related traffic that is aggregated.

what is the line-rate criteria and how many cores are available. For example, five cores support 10 Gbps aggregation indistinguishability without probabilistic filtering or load balancing.

V. Estimated Impact of Deployment

We conducted 20 daily traceroute measurements during the spring of 2018 on the RIPE Atlas platform to evaluate the effectiveness of aggregation-based gossip. The basic idea is to look at client coverage as central ASes and IXPs aggregate STHs. If any significant client coverage can be achieved, the likelihood of pulling off an undetected split-view will be small.

A. Setup

We scheduled RIPE Atlas measurements from roughly 3500 unique ASes that represent 40% of the IPv4 space, trace-routing Google's authoritative CT-over-DNS server and NORDUnet's CT log to simulate clients that fetch DNS STHs in plaintext. Each traceroute result is a list of traversed IPs, and it can be translated into the corresponding ASes and IXPs using public data sets [35][36]. In other words, traversed ASes and IXPs can be determined for each probe. Since we are interested in client coverage as ASes and IXPs aggregate, each probe is weighted by the IPv4 space of its AS. While an IP address is an imperfect representation of a client, e.g., an IP may be unused or reused, it gives a decent idea of how significant it is to cover a given probe.

B. Results

Figure 3 shows AS/IXP path length and stability from the probes to the targets. If the AS path length is one, a single

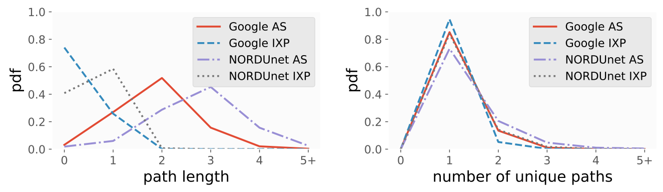


Figure 3. Path length and stability towards Google and NORDUnet.

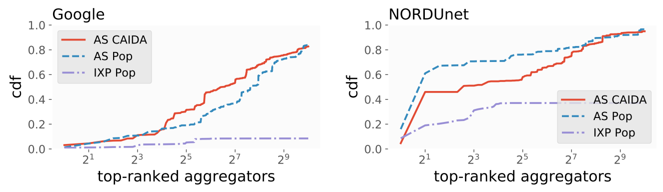


Figure 4. Coverage as a function of aggregation opt-in.

AS is traversed *before reaching the target*. It is evident that an AS path tends to be one hop longer towards NORDUnet than Google because there is a rough off-by-one offset on the x-axis. A similar trend of greater path length towards NORDUnet can be observed for IXPs. For example, 74.0% of all paths traversed no IXP towards Google, but 58.5% of all paths traversed a single IXP towards NORDUnet. These results can be explained by infrastructural differences of our targets: since Google is a worldwide actor an average path should be shorter than compared to a region-restricted actor like NORDUnet. We also observed that AS and IXP paths tend to be quite stable over 20 days (the duration of our measurements). In other words, if AS a and b are traversed it is unlikely to suddenly be routed via AS c.

Figure 4 shows coverage of the RIPE Atlas network as $1 \ldots n$ actors aggregate STHs. For example, 100% and 50% coverage means that at least 40% and 20% of the full IPv4 space is covered. The aggregating ASes and IXPs were selected based on the most commonly traversed vantage points in our measurements (Pop), as well as CAIDA's largest AS ranking [37]. We found that more coverage is achieved when targeting NORDUnet than Google. This is expected given that the paths tend to be longer. Further, if CAIDA's top-32 enabled aggregation the coverage would be significant towards Google (31.6%) and NORDUnet (58.1%).

C. Lessons learned

A vast majority of all clients traverse *at least* one AS that could aggregate. It is relatively rare to traverse IXPs towards Google but not NORDUnet. We also learned that paths tends to be stable, which means that the time until split view detection would be at least 20 days *if* it is possible to find an unprotected client. This increases the importance of aggregation indistinguishability. Finally, we identified vantage points that are commonly traversed using Pop, and these vantage points are represented well by CAIDA's independent AS ranking. Little opt-in from ASes and IXPs provides significant coverage against an attacker that is relatively close to a client (cf. world-wide infrastructure of Google). Although we got better coverage for NORDUnet, any weak attacker would approach Google's coverage by renting infrastructure nearby. Any weak attacker could also circumvent IXP aggregation by detecting the IXP itself [38]. As such, top-ranked AS aggregation should give the best split-view protection.

VI. Related Work

Earlier approaches towards CT gossip are categorized as *proactive* or *retroactive* in Figure 5. We consider an approach proactive if gossip takes place *before* SCTs and/or STHs reach the broader audience of clients. Syta *et al.* proposed proactive witness cosigning, in which an STH is collectively signed by a *large* number of witnesses and at most a fraction of those can be faulty to ensure that a benevolent witness observed an STH [8]. STH cross-logging [9][39][40] is similar in that an STH must be proactively disclosed in another transparency log to be trusted, avoiding any additional cosigning infrastructure at the cost of reducing the size and diversity of the witnessing group. Tomescu and Devadas [41] suggested a similar cross-logging scheme, but split-view detection is instead reduced to the difficulty of forking the Bitcoin blockchain (big-O cost of downloading all block headers as a TLS client). The final proactive approach is STH pushing, where a trusted third-party pushes the same verified STH history to a base of clients [18].

We consider a gossip mechanism retroactive if gossip takes place *after* SCTs and/or STHs reach the broader audience of clients. Chuat *et al.* proposed that TLS clients and TLS servers be modified to pool exchanged STHs and relevant consistency proofs [6]. Nordberg *et al.* continued this line of work, suggesting privacy-preserving client-server pollination of fresh STHs [7]. Nordberg *et al.* also proposed that clients feedback SCTs and certificate chains on every server revisit, and that trusted auditor relationships could be engaged if privacy need not be protected. The latter is somewhat similar to the formalized client-monitor gossip of Chase and Meiklejohn [43], as well as the CT honey bee project where a client process fetches and submits STHs to a pre-compiled list of auditors [42]. Laurie suggested that a client can resolve privacy-sensitive SCTs to privacy-insensitive STHs via DNS (which are easier to gossip) [33]. Private information retrievals could likely achieve something similar [44]. Assuming that TLS clients are indistinguishable from one another, split-view detection could also be implicit as proposed by Gunn *et al.* for the verifiable key-value store CONIKS [11][45].

Given that aggregation-based gossip takes place after an STH is issued, it is a retroactive approach. As such, we cannot protect an isolated client from split-views [8]. Similar to STH pooling and STH pollination, we rely on client-driven communication and an existing infrastructure of packet processors to aggregate. Our off-path verification is based on the same multi-path probing and indistinguishability assumptions as Gunn *et al.* [11][46][47]. Further, given that aggregation is application neutral and deployable on hosts, it could provide gossip *for* the CT honey bee project (assuming plaintext STHs) and any other transparency application like Trillian [48]. Another benefit when compared to browsing-centric and vendor-specific approaches is that a plethora of HTTPS clients are covered, ranging from niche web browsers to command line tools and embedded libraries that are vital to protect but yet lack the resources of major browser vendors [49][50].

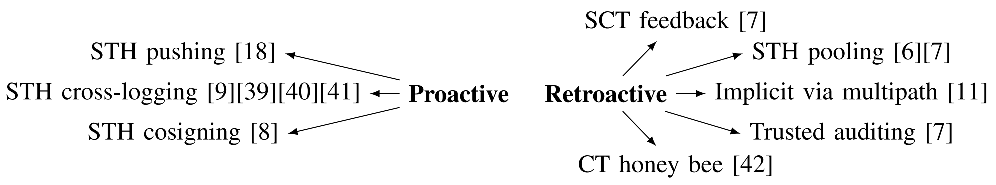


Figure 5. A categorization of approaches towards CT gossip.

Our approach coexists well with witness cosigning and cross-logging due to different threat models, but not necessarily STH pushing if the secure channel is encrypted (no need to fetch what a trusted party provides).

VII. Discussion

Next, we discuss assumptions, limitations and deployment, showing that our approach towards retroactive gossip can be deployed to detect split-views by many relevant attackers with relatively little effort. The main drawback is reliance on clients fetching STHs in plaintext, e.g., using CT-over-DNS [33].

A. Assumptions and Limitations

Aggregation-based gossip is limited to network traffic that packet processors can observe. The strongest type of attacker in this setting—who can completely isolate a client—trivially defeats our gossip mechanism and other retroactive approaches in the literature (see Section VI). A weaker attacker cannot isolate a client, but is located nearby in a network path length sense. This limits the opportunity for packet processor aggregation, but an attacker cannot rule it out given aggregation indistinguishability. Section IV showed based on performance that it is non-trivial to distinguish between (non-)aggregating packet processors on two different targets using P4 and XDP. Off-path challengers must also be indistinguishable from one another to achieve *implicit gossip*. While we suggested the use of anonymity networks like Tor, a prerequisite is that this is in and of itself not an aggregation distinguisher. Therefore, we assume that other entities also use off-paths to fetch and verify STHs. The fact that a unique STH *is not audited* from an off-path could also be an aggregation distinguisher. To avoid this we could rely on a verifiable STH history [51] and wait until the next MMD to audit or simply monitor the full log so that consistency proofs are unnecessary.

The existence of multiple network paths are fundamental to the structure and functioning of the Internet. A weak attacker may use IP fragmentation such that each individual STH fragment is injected from a different location to make aggregation harder, approaching the capabilities of a stronger attacker that is located closer to the client. This is further exacerbated by the deployment of multi-path transport protocols like MPTCP (which can also be fragmented). Looking back at our RIPE Atlas measurements in Section V, the results towards Google's world-wide infrastructure better represent an active attacker that takes *some* measures to circumvent aggregation by approaching a client nearby the edge. Given that the likelihood of aggregation is high if *any* IXP is present (Figure 4), aggregation at well-connected IXPs are most likely to be circumvented.

B. Deployment

Besides aggregating at strategic locations in the Internet's backbone, Internet Service Providers (ISPs) and enterprise networks have the opportunity to protect all of their clients with relatively little effort. Deployment of special-purpose middleboxes are already prevalent in these environments, and then the inconvenience of fragmentation tends to go away due to features such as packet reassembly. Further, an attacker cannot trivially circumvent the edge of a network topology—especially not if aggregation takes place on an end-system: all fragments are needed to reassemble a packet, which means that multi-path fragmentation is no longer a threat. If aggregation-based gossip is deployed on an end-system, STHs could be hooked using other approaches than P4/XDP. For example, shim-layers that intercept TLS certificates higher up in the networking stack were already proposed by Bates *et al.* [52] and O'Neill *et al.* [53]. In this setting, an end-system is viewed as the aggregating packet processor, and it reports back to an off-path challenger that may be a local process running on the same system or a remote entity, e.g., a TelCo could host challengers that collect aggregated STHs from smartphones.

While we looked at programming physical packet processors like routers, STH aggregation could be approached in hypervisors and software switches [54] to protect many virtual hosts. If CT-over-DNS is used to fetch STHs, it would be promising to output DNS server caches to implement the aggregation step. Similar to DNS servers, so called Tor exist relays also operate DNS caches. In other words, P4 and XDP are only examples of how to *instantiate* the aggregation step. Depending on the used plaintext source, packet processor, and network topology other approaches may be more suitable, e.g., C for vendor-specific middleboxes.

C. Retroactive Gossip Benefits From Plaintext

As opposed to an Internet core that only forwards IP packets, extra functionality is often embedded which causes complex processing dependencies and protocol ossification [13]. Many security and protocol issues were found for middleboxes that provides extra functionality [12][55], resulting in the mindset that *everything* should be encrypted [55]. Our work is controversial because it goes against this mindset and advocates that STHs should be communicated in plaintext. We argue that this makes sense in the context of STHs due to the absence of privacy concerns and because the entire point of gossip is to make STHs *available* (rather than end-to-end). The idea of intentionally exposing information to the network is not new, e.g., MPQUIC is designed to support traffic shaping [56].

While we used CT-over-DNS as a plaintext source, there is a push towards DNS-over-TLS [57] and DNS-over-HTTPS [58]. Wide use of these approaches could undermine our gossip mechanism, but ironically the security of TLS could be jeopardized unless gossip is deployed. In other words, long term gossip is an essential component of CT and other transparency logs to avoid becoming yet another class of trusted third-parties. If proactive approaches such as witness cosigning are rejected in favour of retroactive mechanisms, then ensuring that STHs are widely spread and easily accessible is vital. An STH needs no secrecy if the appropriate measures are taken to make it privacy-insensitive [7]. While secure channels also provide integrity and replay protection, an STH is already signed by logs and freshness is covered by MMDs, as well as issue frequency to protect privacy. A valid argument against exposing any plaintext to the network is protocol ossification. We emphasize that our design motivates why packet processors should fail open: otherwise there is no aggregation indistinguishability. Note that there are other plaintext sources than CT-over-DNS that could be aggregated. However, if these sources require stream-reassembly it is generally hard to process in languages such as P4 and XDP [59].

D. Indistinguishability and Herd Immunity

An attacker that gains control over a CT log is bound to be more risk averse than an attacker that compromises a CA. There is an order of magnitude fewer logs than CAs, and client vendors are likely going to be exceptionally picky when it comes to accepted and rejected logs. We have already seen examples of this, including Google Chrome disqualifying logs that made mistakes: Izenpe used the same key for production and testing [60], and Venafi suffered from an unfortunate power outage [61]. Risk averse attackers combined with packet processors that are aggregation indistinguishable may lead to *herd immunity*: despite a significant fraction of clients that lack aggregators, indirect protection may be provided because the risk of eventual detection is unacceptable to many attackers. Hof and Carle [40] and Nordberg *et al.* [7] discussed herd immunity briefly before us. While herd immunity is promising, it should be noted that aggregation distinguishable packet processors at *the edge of a network topology* may be acceptable for some. In other words, if an aggregator cannot be circumvented but it is detectable split-views would still be deterred against covered clients if the challenger is off-path.

VIII. Conclusion and Future Work

Wide spread modifications of TLS clients are inevitable to support CT gossip. We propose that these modifications include challenging the logs to prove certificate inclusion based on STHs *fetched in plaintext*, thereby enabling the traversed packet processors to assist in split view detection retroactively by aggregating STHs for periodic off-path verification. Our results show that the aggregation-step can be implemented without throughput-based distinguishers for a distant attacker, and that our approach offers rapid incremental deployment with high impact on a significant fraction of Internet users. Beyond being an application neutral approach that is complementary to proactive gossip, a compelling aspect is that core packet processors are used (rather than clients) as a key building block: should a consistency issue arise, it is already in the hands of an actor that is better equipped to investigate the cause manually. Further, considering that far from all TLS clients are backed by big browser vendors (not to mention other use-cases of transparency logs in general) it is likely a long-term win to avoid pushing complex retroactive gossip logic into all the different types of clients when there are orders of magnitudes fewer packet processors that could aggregate to their own off-path challengers. Future work includes different instantiations of the aggregation step and evaluating whether aggregation indistinguishability is provided based on throughput and/or latency. The setting may also change in some scenarios, e.g., if DNS caches are aggregated the transport need not be plaintext.

Acknowledgment

We would like to thank Stefan Alfredsson and Philipp Winter for their RIPE Atlas credits, as well as Jonas Karlsson and Ricardo Santos for helping with the NetFPGA setup. We also received funding from the HITS research profile which is funded by the Swedish Knowledge Foundation.

References

[1] Z. Durumeric, J. Kasten, M. Bailey, and J. A. Halderman, "Analysis of the HTTPS certificate ecosystem," in ICM, 2013, pp. 291–304.

[2] "Apple's certificate transparency policy," 2018, URL: https://web.archive.org/web/20190401135231/https://support.apple.com/en-us/HT205280 [accessed 2019-09-04].

[3] D. O'Brien, "Certificate transparency enforcement in Google Chrome," 2018, URL: https://groups.google.com/a/chromium.org/forum/\#!msg/ct-policy/wHILiYf31DE/iMFmpMEkAQAJ [accessed 2019-09-04].

[4] B. Laurie, A. Langley, and E. Kasper, "Certificate transparency," RFC 6962, 2013.

[5] B. Dowling, F. Günther, U. Herath, and D. Stebila, "Secure logging schemes and certificate transparency," in ESORICS, 2016, pp. 140–158.

[6] L. Chuat, P. Szalachowski, A. Perrig, B. Laurie, and E. Messeri, "Efficient gossip protocols for verifying the consistency of certificate logs," in CNS, 2015, pp. 415–423.

[7] L. Nordberg, D. K. Gillmor, and T. Ritter, "Gossiping in CT," Internet-draft draft-ietf-trans-gossip-05, 2018.

[8] E. Syta *et al.*, "Keeping authorities "honest or bust" with decentralized witness cosigning," in IEEE SP, 2016, pp. 526–545.

[9] D. Drysdale, "Minimal gossip," 2018, URL: https://github.com/google/certificate-transparency-go/blob/master/gossip/minimal [accessed 2019-09-04].

[10] O. Gasser, B. Hof, M. Helm, M. Korczynski, R. Holz, and G. Carle, "In log we trust: Revealing poor security practices with certificate transparency logs and Internet measurements," in PAM, 2018, pp. 173–185.

[11] L. J. Gunn, A. Allison, and D. Abbott, "Safety in numbers: Anonymization makes keyservers trustworthy," in HotPETs, 2017, pp. 1–2.

[12] Z. Durumeric *et al.*, "The security impact of HTTPS interception," in NDSS, 2017.

[13] M. Honda, Y. Nishida, C. Raiciu, A. Greenhalgh, M. Handley, and H. Tokuda, "Is it still possible to extend TCP?" in ICM, 2011, pp. 181–194.

[14] P. Bosshart *et al.*, "P4: Programming protocol-independent packet processors," CCR, vol. 44, no. 3, 2014, pp. 87–95.

[15] T. Høiland-Jørgensen *et al.*, "The express data path: Fast programmable packet processing in the operating system kernel," in CoNEXT, 2018, pp. 54–66.

[16] R. Dahlberg, T. Pulls, J. Vestin, T. Høiland-Jørgensen, and A. Kassler, "Aggregation-based gossip for certificate transparency," CoRR, vol. abs/1806.08817, 2019, pp. 1–20.

[17] B. Laurie, "Certificate transparency," ACM Queue, vol. 12, no. 8, 2014, pp. 10–19.

[18] R. Sleevi and E. Messeri, "Certificate transparency in Chrome: Monitoring CT logs consistency," 2017, URL: https://docs.google.com/document/d/1FP5J5Sfsg0OR9P4YT0q1dM02iavhi8ix1mZlZe_z-ls/edit?pref=2&pli=1 [accessed 2019-09-04].

[19] P. Bosshart *et al.*, "Forwarding metamorphosis: Fast programmable match-action processing in hardware for SDN," in ACM SIGCOMM, 2013, pp. 99–110.

[20] "Intel ethernet switch FM600 series: 10/40 GbE low latency switching silicon," URL: https://www.intel.com/content/dam/www/public/us/en/documents/product-briefs/ethernet-switch-fm6000-series-brief.pdf [accessed 2019-09-04].

[21] "Cavium and XPliant introduce a fully programmable switch silicon family scaling to 3.2 terabits per second," URL: https://cavium.com/newsevents-cavium-and-xpliant-introduce-a-fully-programmable-switch-silicon-family.html [accessed 2019-09-04].

[22] "Tofino: World's fastest P4-programmable ethernet switch ASICs," URL: https://barefootnetworks.com/products/brief-tofino/ [accessed 2019-09-04].

[23] "Behavioral model repository," URL: https://github.com/p4lang/behavioral-model [accessed 2019-09-04].

[24] G. Brebner, "P4 for an FPGA target," in P4 Workshop, 2015, URL: \fullversion{https://web.archive.org/web/20190418085926/https://p4workshop2015.sched.com/event/3ZQA/p4-for-an-fpga-target}{https://p4workshop2015.sched.com/event/3ZQA/p4-for-an-fpga-targe} [accessed 2019-09-04].

[25] "Programming NFP with P4 and C," URL: https://www.netronome.com/media/redactor_files/WP_Programming_with_P4_and_C.pdf [accessed 2019-09-04].

[26] M. Budiu, "Compiling P4 to eBPF," URL: https://github.com/iovisor/bcc/tree/master/src/cc/frontends/p4 [accessed 2019-09-04].

[27] S. McCanne and V. Jacobson, "The BSD packet filter: A new architecture for user-level packet capture," in Usenix Winter Technical Conference, 1993, pp. 259–270.

[28] "Apple challenges FBI: All writs act order (CA)," URL: https://www.eff.org/cases/apple-challenges-fbi-all-writs-act-order [accessed 2019-09-04].

[29] R. Dingledine, N. Mathewson, and P. F. Syverson, "Tor: The second-generation onion router," in USENIX Security, 2004, pp. 303–320.

[30] R. Braden, "Requirements for Internet hosts—communication layers," RFC 1122.

[31] S. Deering and R. Hinden, "Internet protocol version 6 (IPv6) specification," RFC 8200.

[32] C. Shannon, D. Moore, and K. C. Claffy, "Beyond folklore: Observations on fragmented traffic," IEEE/ACM Trans. Netw., vol. 10, no. 6, 2002, pp. 709–720.

[33] B. Laurie, "Certificate transparency over DNS," 2016, URL: https://github.com/google/certificate-transparency-rfcs/blob/master/dns [accessed 2019-09-04].

[34] "Aggregation-based gossip for certificate transparency logs," 2018, URL: https://github.com/rgdd/ctga [acessed 2019-09-04].

[35] "The CAIDA UCSD IXPs dataset," February 2018, URL: https://www.caida.org/data/ixps/ [accessed 2019-09-04].

[36] "The Routeviews MRT format RIBs and UPDATEs dataset," March 2018, URL: http://archive.routeviews.org/bgpdata/2018.03/RIBS/ [accessed 2019-09-04].

[37] "ARank," URL: http://as-rank.caida.org/ [accessed 2019-09-04].

[38] G. Nomikos and X. A. Dimitropoulos, "traIXroute: Detecting IXPs in traceroute paths," in PAM, 2016, pp. 346–358.

[39] B. Hof, "STH cross logging," Internet-draft draft-hof-trans-cross-00, 2017.

[40] B. Hof and G. Carle, "Software distribution transparency and auditability," CoRR, vol. abs/1711.07278, 2017, pp. 1–14.

[41] A. Tomescu and S. Devadas, "Catena: Efficient non-equivocation via Bitcoin," in IEEE SP, 2017, pp. 393–409.

[42] A. Ayer, "Lightweight program that pollinates STHs between certificte transparency logs and auditors," 2018, URL: https://github.com/SSLMate/ct-honeybee [accessed 2019-09-04].

[43] M. Chase and S. Meiklejohn, "Transparency overlays and applications," in CCS, 2016, pp. 168–179.

[44] W. Lueks and I. Goldberg, "Sublinear scaling for multi-client private information retrieval," in FC, 2015, pp. 168–186.

[45] M. S. Melara, A. Blankstein, J. Bonneau, E. W. Felten, and M. J. Freedman, "CONIKS: Bringing key transparency to end users," in USENIX Security, 2015, pp. 383–398.

[46] M. Alicherry and A. D. Keromytis, "DoubleCheck: Multi-path verification against man-in-the-middle attacks," in ISCC, 2009, pp. 557–563.

[47] D. Wendlandt, D. G. Andersen, and A. Perrig, "Perspectives: Improving SSH-style host authentication with multi-path probing," in USENIX ATC, 2008, pp. 321–334.

[48] A. Eijdenberg, B. Laurie, and A. Cutter, "Verifiable data structures," 2015, URL: https://github.com/google/trillian/blob/master/docs/papers [accessed 2019-09-04].

[49] M. Backes, S. Bugiel, and E. Derr, "Reliable third-party library detection in Android and its security applications," in CCS, 2016, pp. 356–367.

[50] E. Derr, S. Bugiel, S. Fahl, Y. Acar, and M. Backes, "Keep me updated: An empirical study of third-party library updatability on Android," in CCS, 2017, pp. 2187–2200.

[51] L. Nordberg, "Re: [Trans] providing the history of STHs a log has issued (in 6962-bis)," URL: https://mailarchive.ietf.org/arch/msg/trans/JbFiwO90PjcYzXrEgh-Y7bFG5Fw [accessed 2019-09-04].

[52] A. M. Bates *et al.*, "Securing SSL certificate verification through dynamic linking," in CCS, 2014, pp. 394–405.

[53] M. O'Neill *et al.*, "TrustBase: An architecture to repair and strengthen certificate-based authentication," in USENIX Security, 2017, pp. 609–624.

[54] M. Shahbaz *et al.*, "PISCES: a programmable, protocol-independent software switch," in ACM SIGCOMM, 2016, pp. 525–538.

[55] A. Langley *et al.*, "The QUIC transport protocol: Design and Internet-scale deployment," in SIGCOMM, 2017, pp. 183–196.

[56] Q. D. Coninck and O. Bonaventure, "Multipath QUIC: Design and evaluation," in CoNEXT, 2017, pp. 160–166.

[57] S. Dickinson, D. Gillmor, and T. Reddy, "Usage profiles for DNS over TLS and DNS over DTLS," RFC 8310, 2016.

[58] P. Hoffman and P. McManus, "DNS queries over HTTPS (DoH)," RFC 8484, 2018.

[59] R. Dahlberg, "Aggregating certificate transparency gossip using programmable packet processors," Master Thesis, Karlstad University, 2018.

[60] R. Sleevi, "Upcoming CT log removal: Izenpe," 2018, URL: https://groups.google.com/a/chromium.org/forum/#!topic/ct-policy/qOorKuhL1vA [accessed 2019-09-04].

[61] ——, "Upcoming log removal: Venafi ct log server," 2017, URL: https://groups.google.com/a/chromium.org/forum/#!topic/ct-policy/KMAcNT3asTQ [accessed 2019-09-04].