Reliable Client Accounting for P2P-Infrastructure Hybrids

Paarijaat Aditya[†] Mingchen Zhao[‡] Yin Lin[⋆◇]

Andreas Haeberlen[‡] Peter Druschel[†] Bruce Maggs[⋆◇] Bill Wishon[◇]

[†]Max Planck Institute for Software Systems (MPI-SWS) [‡]University of Pennsylvania [⋆]Duke University [◇]Akamai Technologies

Abstract

Content distribution networks (CDNs) have started to adopt *hybrid* designs, which employ both dedicated edge servers and resources contributed by clients. Hybrid designs combine many of the advantages of infrastructure-based and peer-to-peer systems, but they also present new challenges. This paper identifies *reliable client accounting* as one such challenge. Operators of hybrid CDNs are accountable to their customers (i.e., content providers) for the CDN's performance. Therefore, they need to offer reliable quality of service and a detailed account of content served. Service quality and accurate accounting, however, depend in part on interactions among untrusted clients. Using the Akamai NetSession client network in a case study, we demonstrate that a small number of malicious clients used in a clever attack could cause significant accounting inaccuracies.

We present a method for providing reliable accounting of client interactions in hybrid CDNs. The proposed method leverages the unique characteristics of hybrid systems to limit the loss of accounting accuracy and service quality caused by faulty or compromised clients. We also describe RCA, a system that applies this method to a commercial hybrid content-distribution network. Using trace-driven simulations, we show that RCA can detect and mitigate a variety of attacks, at the expense of a moderate increase in logging overhead.

1 Introduction

An increasing number of commercial content-distribution networks (CDNs) are based on a hybrid architecture, which combines peer-to-peer and infrastructure-based elements. These include, for instance, Velocix [5], LiveSky [35], Pando Networks [4], Octoshape [3], and PPLive [33]. Even Akamai, the largest CDN, which was originally purely infrastructure-based, has recently added peer-assisted technologies like NetSession [8]. There are good reasons for this trend. Hybrid architectures combine many of the benefits of peer-to-peer systems (cost reduction through the use of client resources, use of local resources, e.g., within a corporate intranet) with those of infrastructure-based systems (dedicated backup resources, dependability, centralized management and control); moreover, by utilizing both peer and infrastructure resources, hybrids are able to provide better service than pure peer-to-peer or infrastructure-based systems. Studies have shown impressive advantages: the potential bandwidth savings for the service provider are considerable [21], and hybrid CDNs can significantly reduce the costs of all parties involved in the content distribution process, *including* the edge ISPs [22].

However, compared to purely infrastructure-based systems, hybrid architectures face an inherent challenge: By definition, peer-to-peer communication occurs between untrusted clients, and therefore cannot be observed directly by the trusted infrastructure. As a result, faulty or compromised clients can mishandle peer communication in ways that are not observable by the infrastructure, and they can under- or overreport peer interactions. In principle, a compromised client may be able to censor or modify content, and inject unauthorized content; it may refuse, delay, or abort transfers to deny or degrade service to other clients; and it may misreport peer transfers in an attempt to manipulate the accounting for commercial content or services.

In practice, hybrid systems can take measures to mitigate this risk. For instance, clients can obtain signed content hashes from the trusted infrastructure to verify content received from peers. The infrastructure can control which clients may peer, in order to make collusion of faulty clients more difficult. Client logs can be checked and suspicious or inconsistent records excluded at some cost in logging accuracy. Clients that have repeatedly been involved in disputed or aborted transactions can be blacklisted at some risk of blocking legitimate clients. Nevertheless, the potential remains for compromised clients to disrupt service quality and affect logging accuracy.

There is little evidence of widespread attacks of this type against hybrid systems today. However, as these systems become more popular, it is important to understand the risks. Therefore, we have (with permission) performed a 'red team' evaluation of one specific hybrid CDN, Akamai's NetSession system, which has a large deployment with currently over 24 million clients. We have identified an attack vector that enables a single malicious client to report more than 30 GB of fictitious download activity per hour, an amount that can be further inflated through a Sybil attack [17].

While this specific vulnerability has been removed, the underlying challenge remains: a hybrid system's accounting is based on information from untrusted peers that is difficult to verify, and a determined attacker could find other ways to exploit this vulnerability. The challenge also applies to other commercial hybrid content delivery systems, not just to NetSession, and may apply to other types of hybrid systems as well. For example, CDNs have developed methods for owner-operated network appliances to serve customer content and report on download activity [23]. Unlike edge servers, which are part of the CDN's infrastructure, these appliances are not under the administration of the CDN. Similarly, certain network games rely on direct communication and interaction between game consoles, with outcomes ultimately reported to a centralized infrastructure [7]. Hence, we are interested in a principled approach to reliable accounting of client interactions.

To address this challenge, we present a method for providing reliable client accounting in hybrid distributed systems. Our method uses the infrastructure nodes to establish reliable facts about the clients, such as an upper bound on their available resources (which is essential to limit the effect of Sybil attacks). Moreover, all clients record a tamper-evident log [20] of their actions and must periodically upload their log to an infrastructure node; this severely restricts the ability of malicious clients to lie without getting caught.

A key feature of our approach is the ability to *quarantine* suspicious clients. Quarantined clients cannot interact with other clients, and their requests are served directly by the infrastructure; thus, such clients are unable to misreport their actions or disrupt other clients. A key insight is that *in hybrid systems, quarantining is safe*: if an honest client is accidentally quarantined, its quality of service does not change, it merely causes a small amount of extra load on the infrastructure. Thus, the infrastructure can afford to err on the side of caution e.g., by using anomaly detection techniques with high false-positive rates, which are difficult for an adversary to escape.

To demonstrate that our approach is effective, we present RCA, a system that applies our approach to NetSession. We report results from a trace-driven evaluation, based on traces from Akamai's production deployment. Our results indicate that RCA increases the protocol overhead from 0.06% without RCA to 0.47% with RCA, relative to the amount of content served, and it requires clients to maintain approximately 550 bytes of extra information per MB of downloaded content that must be uploaded to, and checked by, the infrastructure. We also show that RCA is effective against a variety of attacks and misbehaviors by malicious clients. In summary, our contributions are as follows:

- A case study of a hybrid content delivery system based on Akamai NetSession (Section 3);
- A demonstration of an accounting attack on NetSession (Section 4);
- A method for providing reliable client accounting in hybrid distributed systems (Section 5);
- RCA, an application of our approach to NetSession (Section 6); and
- A comprehensive evaluation, based on traces from NetSession (Section 7).

2 Related work

Hybrid systems: Several studies, e.g., [21, 22], have predicted considerable benefits for peer-assisted CDN designs, and measurement studies of commercial peer-assisted CDNs, such as LiveSky [35] and PPlive [33], seem to confirm that these benefits are being achieved in practice. Client misbehavior in peer-to-peer CDNs has been observed empirically, e.g., collusion among users of the Maze system [24]. Most existing defenses assume rational clients, who misbehave to increase their own performance or minimize their cost. Misbehavior of this type can be prevented with robust incentives; for instance, Dandelion [31] rewards uploads from clients with virtual currency that can be used to purchase downloads from other clients or the infrastructure. However, such incentives are not effective against malicious attacks of the type we consider in this paper. Dandelion focuses on preventing freeloading, while RCA considers more general Byzantine behavior as well.

Some systems do consider certain types of malicious behavior. For example, Antfarm [28], which uses cryptographically signed tokens as payment for block downloads, can detect forged tokens; [28] also briefly sketches possible extensions that could mitigate additional attacks, such as double-spending. However, Antfarm does not use the tokens for reporting, but rather for allocating the infrastructure's bandwidth among the different swarms, and consequently has lower requirements for the accuracy of reporting. Also, Antfarm's weapon against malicious clients is to excise them from the swarm. Unlike quarantined clients in RCA, excised clients no longer receive service. Therefore, clients can be excised only in cases of cryptographically verifiable misbehavior, such as forged tokens or double-spending.

Accounting mechanisms: Seuken and Parkes [30] examined the problem of reliable accounting in a decentralized setting with rational peers, and they have shown that, in this setting, no Sybil-proof accounting mechanism exists. This result does not apply to hybrid systems because the infrastructure can serve as a central, trusted monitoring component. Moreover, [30] assumes that peers are either cooperative or rational, whereas this paper also considers Byzantine peers.

Accountability: Accountability systems like PeerReview [20] can automatically detect a large subclass of Byzantine faults and tie them to the identities of specific faulty nodes. RCA's tamper-evident log is based on ideas from PeerReview, but differs from PeerReview in several ways; for example, RCA is designed to withstand Sybil attacks, and takes advantage of a central infrastructure in order to reduce overhead. Also, RCA's focus is on reliable accounting rather than fault detection.

Defenses against Sybil attacks: Douceur's original paper on the Sybil attack [17] suggests resource testing as a possible defense, and subsequent work has explored a variety of solutions for different types of resources, such as distinct physical locations [10], money [25], and social relationships with honest users [36, 37]. Our approach is perhaps closest to Tarzan [18], which uses IP addresses to identify instances of Sybil identities. Another approach relies on certification authorities, as in [6]. Certification is unsuitable for systems like NetSession, which seek to keep the registration process simple to encourage adoption.

Anomaly detection: RCA includes anomaly detection [13] and benefits in this regard from decades of research, e.g., on intrusion-detection systems [15]. However, when applied by itself, anomaly detection faces limitations in a security context [9], because an adversary may be able to avoid detection by shifting the system's workload gradually, e.g., in a frog-boiling attack [12]. To guard against such attacks, RCA complements anomaly detection with tamper-evident logs as well as consistency and invariant checks, which do not rely on assumptions about the system's workload.

3 Case study: Akamai NetSession

To provide some context for our discussion of reliable accounting in hybrid systems, we first describe the design of a concrete hybrid system: Akamai NetSession.

3.1 The NetSession system

The NetSession system is a peer-assisted content delivery network (CDN) operated by Akamai. Like Akamai's more widely known infrastructure-based service, NetSession's primary function is to accept *content*, such as software packages or videos, from a number of *content providers*, and to deliver that content to a potentially large number of *users*. While the infrastructure-based service delivers the content exclusively from a number of *edge servers* that are operated by Akamai, NetSession additionally leverages peer-to-peer transfers between the user nodes, or *clients*. Content can be downloaded from the edge servers, the peers, or a mixture of both.

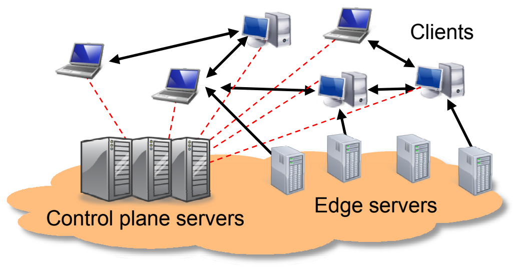

Figure 1: Overview of the NetSession system. Control connections are shown as dotted lines.

To use NetSession, users must install the NetSession client software on their machines. The NetSession software can maintain links to other clients and communicates with a number of *control plane* servers, which coordinate the connections between the clients (Figure 1). Once installed, the client can be reused for future downloads, and it offers an API to local applications. The client also provides a GUI for monitoring download progress, as well as a set of controls for enabling or disabling uploads to peers. Clients are identified by a GUID, which is chosen at random during installation and does not contain personally identifiable information. Since each client periodically connects to the control plane, we can use the number of such connections to estimate the number of clients in the system. During November 2011, the control plane registered connections from more than 24 million distinct GUIDs.

3.2 Why a hybrid architecture?

Historically, most CDNs have *either* been infrastructure-based, such as Akamai and Limelight [2], *or* peer-to-peer, such as PPLive [33], Joost [1], or BitTorrent [14]. Both architectures have their own advantages and disadvantages. The key advantage of peer-to-peer systems is their scalability and independence of infrastructure; infrastructure-based CDNs can only scale by provisioning additional infrastructure resources, which can be expensive. The key advantage of infrastructure-based systems is their predictable quality of service; the service provided by peer-to-peer systems, on the other hand, can be inconsistent [34], because end-user machines are often resource-constrained and unreliable.

Hybrid systems seem like an attractive design point because they can combine most of the advantages of both systems. On the one hand, they can scale and reduce cost by leveraging resources contributed by the clients; on the other hand, they can mask glitches and performance problems by falling back on the infrastructure, which ensures good quality of service. In addition, the infrastructure can be used to solve a variety of technical problems that tend to plague peer-to-peer systems. For example, it can serve as a rendezvous point for NAT traversal, it can perform global optimizations [28], it can provide extra capacity in the critical initial and final stages of the download, and it can validate content exchanged among clients. The infrastructure can also maintain a central directory, so that, when a client initiates a download, the infrastructure can immediately suggest a number of peers that are online, have a compatible NAT type, and are storing the requested file.

However, to secure hybrid systems, we need to understand whether their specific structure makes them vulnerable to new kinds of attacks, and what kinds of defenses may be appropriate for them. In this paper, we identify a new class of attacks that exploit the infrastructure's inability to observe client interactions directly, and we present a system that can detect and mitigate these (and other) attacks.

3.3 Operation

We now briefly describe how a peer-assisted download in NetSession works. When the user of a client c initiates a download, the NetSession software on c sends a request to the control plane, which returns a set of up to k peers (currently 40) $C := c_1, \ldots, c_k$ that are currently online and have blocks of the requested file. The list contains only peers that have compatible NATs and is biased towards peers that are close to c. Once c has received this list, it opens connections to some subset of C. The control plane also notifies the peers in C, so that both endpoints can open holes in their local NATs if necessary.

The actual download is done using a swarming protocol conceptually similar to the BitTorrent [14] protocol: the file is broken into fixed-size blocks, clients exchange bitmasks with their peers to indicate which blocks they have available, and clients can request ranges of blocks from each other. Swarming and downloads from edge servers can proceed concurrently. Each block is verified against a hash that is obtained from an edge server; thus, corrupted blocks can be discarded and downloaded again.

Clients only upload content file blocks to their peers when three conditions are met: 1) the local client has previously downloaded the file, 2) the content provider has enabled swarming for that file, and 3) swarming is enabled in the client's local NetSession installation (this can be changed by the user in a control panel setting). To avoid inconveniencing the user, NetSession limits the upload/download ratio, and it ensures that uploads do not cause resource contention on the local machine.

NetSession downloads are peer-*assisted* in the sense that downloading content from peers is helpful but not required for correctness: even if a client does not receive a single valid block from its peers (or there are currently no peers that have the requested content), it can still complete the download using only the edge servers. Nevertheless, it is important to identify faulty and misbehaving clients, since they can degrade the CDN's quality of service. For instance, faulty clients can request an inordinate number of blocks from their peers or send them corrupted data blocks, which subsequently fail the hash verification and must be downloaded again. This is not an issue in infrastructure-based CDNs, where all content is delivered by the edge servers.

3.4 Logging and reporting

NetSession generates a detailed log to document its performance. Each client regularly reports certain events to the control plane, including 1) the start and the completion of a download, 2) its performance during the download, and 3) the number of bytes that it has downloaded from peers and from the infrastructure. These reports are logged by the control plane.

The NetSession logs are used internally by Akamai, e.g., for quality control and to improve system performance. However, they also serve another crucial function. Customers of CDNs expect to have access to the logs of all downloads of their content; they use these logs for analytics, i.e., to determine which content is popular, which clients are downloading the content, and what level of performance the CDN is providing to the clients. Since the logs are directly visible to the customer, it is essential (from the CDN operator's perspective) that the logged data is reliable. If the data in the logs were found to be inaccurate or subject to manipulation by users, this could undermine the CDN operator's credibility.

We hypothesize that logging and reporting are key features of hybrid systems in general, beyond the specific NetSession system, and perhaps even beyond hybrid CDNs. By definition, a hybrid system has an infrastructure component, which must be paid for by some party, and that party will want an accurate report of what they are paying for. Hence, attacks on reporting represent a serious threat to hybrid systems.

4 Attacks on hybrid systems

In this section, we describe attacks on hybrid CDNs, including a novel class of *inflation attacks* on the reporting system. To demonstrate that existing systems are vulnerable, we report results from a successful inflation attack on the NetSession system.

4.1 Threat model

In this paper, we assume that the nodes in the infrastructure (such as Akamai's edge servers) are correct and fully trusted by the operator. The clients, on the other hand, are untrusted and can be compromised or fail in various ways, so we conservatively assume that some subset of them is controlled by a malicious adversary.

Clients can communicate with infrastructure nodes and with their peers, but the infrastructure cannot observe direct peer-to-peer communication.

We also assume that the system does not use strong identities and that membership is open, i.e., any client is allowed to join the system. In particular, this means that the adversary can potentially mount a Sybil attack [17] on the system, e.g., by running multiple instances of the client software on the same machine.

4.2 Attack vectors

There are two aspects of this model that make reliable accounting fundamentally difficult. First, the clients are not fully trusted and could tell lies, suppress information, or mishandle the communication with their peers to deny them service. Second, a (potentially large) fraction of the events that are to be accounted for occur directly between the clients, where the infrastructure cannot observe them. Separately, each of these challenges would be easy to handle: if the clients were trusted, the infrastructure could rely on their correct handling of requests and the accuracy of their reports; if the infrastructure was directly involved in all communication (as in Akamai's infrastructure-based CDN), it could intercede when clients misbehave towards their peers, and perform accounting based on its own records.

In the following, we focus on a novel type of accounting attack that exploits these challenges. We will refer to this attack as an *inflation attack*. In an inflation attack, the attacker causes the system to overreport the amount of service that it has provided. Using the same technical approach, one could also mount *deflation attacks*, in which the attacker would cause the amount of service to be underreported instead.

4.3 Inflation attack on NetSession

To demonstrate the potential impact of inflation attacks, we carried out such an attack on the NetSession system. The attack could be carried out easily by modifying the NetSession client software. We decided against this approach, partly because a modified client might have accidentally disrupted other clients or the NetSession infrastructure, partly because we were able to carry out the attack even with an unmodified NetSession client.

To accomplish this, we wrote a script that uses the client's API to repeatedly download a certain file, as well as a small proxy that interposed between our local NetSession client and its peers. Whenever our client attempts to contact a peer, our proxy returns a spoofed response that indicates that the peer has all of the requested blocks, and whenever our client requests any blocks, the proxy returns the blocks at LAN speed. This man-in-the-middle attack was possible because, unlike the messages exchanged between clients and the infrastructure, communication between peers was not signed. The effect of this attack was that our client would trigger a large number of downloads, and these downloads would complete at LAN speed. The resulting large number of downloads were reported to the infrastructure, even though no content was actually transferred among the peers.

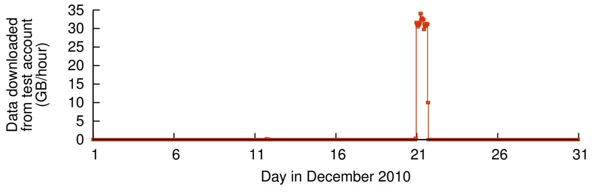

Figure 2: Effect of our attack on the NetSession logs.

4.4 Impact of the attack

We obtained permission from Akamai to test our attack on the deployed NetSession system. To avoid the risk of interfering with the production system, we targeted a set of special files that is normally used for testing; thus, there was no risk that legitimate clients would attempt to download files from our proxied client, or that our attack would affect the logs of legitimate content providers. We ran our attack for one full day; during that day, we requested files as quickly as possible. To assess the impact of the attack, Akamai gave us access to the control plane logs for our client's specific GUID.

Figure 2 shows the reported downloads in this log. Our single modified client was able to generate about 30 GB/hour of fictitious downloads, which show up as a sharp spike in the figure. We note that this is a proof-of-concept attack, and that its throughput was limited by the throughput of our proxy. Had we chosen to directly modify the client software, we could have reported downloads at arbitrary rates.

4.5 How serious is this attack?

Our specific proof-of-concept attack is not difficult to prevent; indeed, the attack as described no longer works. However, our specific attack is just one example from an entire class of attacks; for instance, we could have reverse-engineered the NetSession client software and directly modified the reports it sends to the NetSession infrastructure. So the root of the problem runs deeper.

Moreover, the vulnerability seems to exist in hybrid systems generally; it is inherent in the fact that hybrid systems must account for interactions between untrusted clients, who cannot be relied upon to report them accurately. To prevent similar attacks on NetSession and other hybrid systems, a more comprehensive solution is needed. Also, to show that a simple fix is not sufficient, we briefly discuss a few strawman solutions.

Sign all messages: Cryptography can be used to defend against man-in-the-middle attacks, but recall that this was just a trick we used to make our proof-of-concept attack easier to implement. Instead, an attacker could simply modify the client software to generate whatever messages he needs it to send.

Detect software modifications: There are techniques that aim to detect whether a client has modified software. Some of these techniques are purely software-based; for example, certain multiplayer games scan the clients' memory for known cheats [29]. However, an adversary can circumvent these techniques by disabling the scan or by reporting incorrect results. Other techniques require hardware support; for example, trusted platform modules can be used to certify that a client has loaded a certain software image [19]. However, the requisite hardware usually is not available on all clients, and even where it is available, it may be able to certify only that a certain binary was loaded, not that it is still running.

Limit clients to one download per file: This would thwart our specific attack; however, an adversary could still download many different files or create many Sybil identities that each download the file only once.

Anomaly detection: A massive load spike like the one in Figure 2 would probably raise the CDN operator's suspicion. However, there could be a legitimate reason for the spike, and there is no way to establish its provenance after the fact. Thus, the operator is caught in a dilemma: if the spike is genuine, it is probably important for the content provider to know about it, so it must be left in the log; if the spike is fake, its presence distorts the accounting, so it should be removed.

5 Reliable accounting

In this section, we describe our method for providing reliable accounting in hybrid systems. Although we developed this method with NetSession in mind, it should be applicable to other types of hybrid systems (such as P2P streaming or storage systems). We begin with a description of the general approach and then show (in Section 6) how it can be applied to NetSession.

5.1 System model

We consider a system that consists of a number of trusted infrastructure nodes and a (potentially much larger) number of untrusted clients. The system offers a service to the clients that can be provided either by the infrastructure nodes or by other clients. The infrastructure cannot directly observe interactions between the clients.

Our goal is to provide an *accounting mechanism* that reliably captures client activity. Specifically, we are interested in activity by faulty or compromised clients that could degrade the performance of the system or distort the record of services actually rendered.

F1	Fail to log exact set of messages sent or acknowledged	5.4
F2	Fail to log consistent sequence of messages	5.5
F3	Execute illegal, or fail to execute required, protocol action	5.6
F4	Faulty peers collude to report fictitious exchanges	5.7, 5.8
F5	Render poor service to peers	5.8
F6	Nefarious user requests	5.9
F7	Sybil attack	5.10

Table 1: Types of client misbehaviors. The last column shows the subsection that describes the countermeasure.

5.2 Threat analysis

Next, we characterize the kinds of threats that a faulty or malicious client poses to the rest of the system. To do this in a protocol-independent way, we model the client as an abstract state machine that accepts requests from the local user (e.g., names of files to download) and eventually produces responses (e.g., the contents of the file). The state machine can send and receive messages, and it must periodically upload a log of its actions to the infrastructure. Each state machine is expected to follow a specific protocol, and this protocol is not necessarily fully deterministic (e.g., clients might be allowed to choose the peers with the highest throughput).

Table 1 summarizes the types of threats we consider here. Faulty or malicious clients can *fail to log* the exact set of messages they have sent or acknowledged (F1), or fail to log them in a sequence that is causally consistent (F2). They can *violate the protocol*, either by making a bad state transition or by failing to make a required state transition (F3). They can collude with faulty peers in order to report fictitious transactions amongst each other (F4). They can *deliver poor performance*, e.g., by being slow to respond to messages from peers, by aborting or delaying peer transfers, or by sending corrupted content (F5). A user can issue *nefarious content requests* in order to create artificial demand for a provider's content or to degrade the service quality enjoyed by other clients (F6). Finally, a user can mount a *Sybil attack* by joining the system under more than one identity, in order to amplify other attacks (F7).

5.3 Approach

In a fully decentralized system, it is difficult or even impossible [30] to provide reliable accounting. In a hybrid system, however, we can do better by leveraging some of the unique characteristics of these systems, namely:

1. **Trusted infrastructure:** The nodes in the infrastructure are directly controlled by the operator;

2. **Central control:** The operator can prescribe a single protocol that all the nodes must follow;

3. **Global view:** The operator is able to observe the status of all clients eventually; and
4. **Dedicated resources:** The infrastructure has the capacity to take over for under-performing or suspicious clients.

In the following, we describe a sequence of techniques for constructing a reliable accounting mechanism that takes advantage of these characteristics. Each technique adds some constraints on the kinds of behaviors a faulty node can manifest without getting caught.

5.4 Require message commitment

First, we require clients to cryptographically sign all messages and acknowledgments they send. Moreover, clients record the signatures of received messages and acknowledgments in a log, which they periodically forward to the infrastructure. As a result, a faulty client can no longer deny having sent or received a message it has previously sent or acknowledged. Likewise, a faulty client cannot falsely claim that a correct client has sent or acknowledged a message, because the faulty client would not be able to produce the corresponding signature.

5.5 Check logs for consistency

Even with message commitment in place, a faulty client could give a false account of the order in which certain events happened. For instance, a client could receive an object from the infrastructure, acknowledge it, and then later decline a peer's request for the same object. In its logged record, it could claim that the peer's request had arrived before it received the object from the infrastructure, in order to justify its failure to serve the object.

This form of misrepresentation can be avoided by requiring each client to maintain a *tamper-evident log* [20] of its actions. For this purpose, the log entries form a hash chain, and the hash of the most recent log entry is included in the signature of any message that the node signs. Thus, the node commits to its entire event history each time it sends a message or acknowledgment. Because the logs and all of a node's signatures are eventually sent to and checked by the infrastructure, a client would be caught if it ever omitted, fabricated, or manipulated events in its logs, or gave inconsistent accounts of the sequence of events.

Each client is forced to log a single linear account of its actions that includes all acknowledged messages sent to, or received from, correct peers. If messages are forged, omitted, reordered, or tampered with, the client effectively makes a signed admission of guilt.

5.6 Check logs for plausibility

Even when a client's log is consistent with the logs of all other clients it has communicated with, the log can still be implausible; for example, a client A might download a file from a peer B and then, when a third peer C requests that file from A, serve a modified version of the file or claim that it no longer stores the file. To prevent this, our second step is to verify that a log is *plausible*, i.e., it is consistent with a valid execution of the software the client is expected to run.

We can decide whether a log is plausible by checking that it satisfies a set of *invariants*, which must hold in any correct execution of the client software. Typical example invariants state that a client must only serve content it has previously received, may only contact or accept requests from peers suggested by the infrastructure, etc.

5.7 Control client pairings

If clients are free to choose which clients to request services from, malicious clients can collude to request services from each other (and thus make consistent and plausible logs without actually doing any work), or they can 'gang up' on some correct clients to deny them services. To make collusion more difficult, the infrastructure can impose *restrictions* on the clients, e.g., by limiting each client i to only request services from peers in some set S_i.

Restrictions should be neither too tight nor too loose. In the above example, very small sets S_i will force the clients to contact the infrastructure frequently (e.g., to ask for additional clients if the ones in S_i fail), which increases overhead and decreases performance, whereas very large sets S_i increase the chances that a malicious client i will find an accomplice in its set S_i.

The infrastructure's choice of S_i may depend on which peers have the requested content, which ISP a peer is connected to, and whether it has a compatible type of NAT. Some of these factors can be influenced by a peer; for instance, a peer could download rarely requested content in order to increase the chance that it will be paired with a colluding peer that requests that same content. Therefore, controlling client pairing can only mitigate but not eliminate client collusion.

5.8 Quarantine anomalous clients

A faulty client can degrade the service received by its peers, e.g., by providing the requested services inconsistently or too slowly. Moreover, as discussed above, colluding peers could inflate their upload activity. Our next step is to apply statistical *anomaly detection* to identify potentially problematic clients, and to quarantine those clients.

Ordinarily, anomaly detection systems face a difficult tradeoff between effectivity and the number of false positives. Our key insight is that, in the specific case of hybrid systems, *this tradeoff can be avoided by redirecting any suspicious clients to the infrastructure*. In other words, when a client c manifests anomalous behavior, the infrastructure can restrict c to contacting only trusted infrastructure nodes, and it can tell other clients not to contact c.

If c is malicious, quarantining c will ensure accurate logging because a) c's interactions with the infrastructure will be logged by the trusted nodes, and b) c cannot plausibly log any interactions with other nodes. On the other hand, if c is correct and its detection was a false positive, c's requests will still be handled by the infrastructure, so c's user will still receive good service. Since quarantining clients is 'safe' from a QoS perspective, we can perform anomaly detection aggressively and accept a nontrivial false-positive rate, as long as the infrastructure has sufficient resources to handle the extra load.

5.9 Flag/throttle suspicious user behavior

In addition to the types of client misbehaviors covered in the previous section, there is a class of attacks that is caused solely by user activity, while the client software behaves as expected. For instance, a user could nefariously download content from a specific content provider, in order to drive up demand for that provider's content. (This type of attack would typically be combined with a Sybil attack and possibly a botnet. Also, note that this attack is not specific to hybrid CDNs.)

Based on the tamper-evident logs collected by the system, we can perform statistical anomaly detection to identify clients whose download activity stands out in terms of volume and content selection. A flagged client can be immediately subjected to a download rate-limit by the infrastructure, pending resolution by a human operator. At the same time, a human operator is notified of the anomaly for further inspection and resolution. For instance, an operator can contact a content provider to check if a sudden increase in demand (possibly from a specific set of IP addresses) is expected.

5.10 Enforce resource limits

Some of the attacks described in the previous sections can be amplified by a *Sybil attack*, where an attacker registers more than one instance of the client software for each physical node he controls. For instance, the impact of a colluding-peers attack or a nefarious download attack increases with the number of client instances. Without strong user identification, we cannot effectively prevent Sybil attacks, but we can at least constrain the aggregate amount of service that Sybils can log. For example, we can check that, in aggregate, the activities recorded in these logs cannot exceed the physical capacity of the adversary's nodes.

Since a hybrid system contains trusted infrastructure nodes, we can achieve this goal through resource testing. For example, a client could be required to demonstrate its upstream or downstream bandwidth in a short data exchange with the infrastructure. The infrastructure could refuse to accept multiple clients with the same IP address, and/or ask a group of clients with a common IP prefix to demonstrate that they run on separate machines by asking each of them to simultaneously solve a different crypto puzzle. The results could then be used to flag implausible client activity, such as clients who claim to have exchanged data with peers in different networks at a rate that exceeds their measured access link capacity, or clients who claim to have exchanged data with a number of clients on the same network that exceeds the number of separate machines they have demonstrated.

6 Application to NetSession

In this section, we describe the RCA system, which applies the method from Section 5 to NetSession.

6.1 Overview

Our design instantiates each of the building blocks we have presented in Section 5. We use resource certificates (Section 6.3) to limit the aggregate bandwidth of Sybils, a novel implementation of tamper-evident logs that has been optimized for hybrid systems (Sections 6.4 and 6.5) to check for consistency, a set of NetSession-specific invariants (Section 6.6) to check for plausibility, and a set of statistical tests (Section 6.7) for quarantining anomalous clients. The rest of this section describes each of these building blocks in more detail.

The basic workflow in RCA is as follows. When a client i first joins RCA, it contacts one of the control plane servers and uploads a short file to demonstrate its link capacity; the control plane then issues the client a private key σ_i (for signing messages), a public key π_i, and a certificate Γ_i that encodes the measured capacity. The client can then download or upload content, just as in the original NetSession system, but it additionally maintains a tamper-evident log, which it periodically uploads to the control plane. The control plane forwards the logs to a set of backend servers, which process them and produce the accounting information. The control plane also applies statistical tests to detect and quarantine anomalous clients.

6.2 Assumptions

The design of RCA relies on the following assumptions:

1. Infrastructure nodes are trusted by the operator and can only fail by crashing.
2. All nodes have access to a cryptographic hash function H.
3. Faulty clients cannot forge the signature of correct peers or of the infrastructure.

Assumption 1 seems reasonable in a centrally managed CDN like NetSession; assumptions 2 and 3 are commonly assumed to hold for hash functions like SHA-256 and algorithms such as RSA, provided that the keys are sufficiently strong.

6.3 Resource certificates

To prevent malicious clients from reporting more activity than they physically have the capacity to perform, RCA uses a few simple resource tests, as discussed in Section 5.10.

When a client i first joins the system, it contacts one of the control plane servers and requests a key pair, which will constitute the client's identity for the purposes of reporting. The control plane then exchanges some amount of data with the client, and it measures the maximum throughput C_i that i achieves during the exchange.[1] Finally, the control plane then generates a fresh key pair σ_i/π_i and returns it to the client, along with a certificate $\sigma_P(\pi_i, G_i, C_i, A_i, T_i)$ that is signed with the control plane server's private key σ_P and binds the client's public key π_i to the client's GUID G_i, its measured capacity C_i, its IP address A_i, and an expiration time T_i (on the order of a few hours). When a client's current certificate expires or its IP address changes, the client repeats this process to obtain a fresh certificate.

To prevent Sybil attackers from obtaining multiple certificates for the same IP address, the control plane internally maintains a table with all unexpired certificates. Suppose a client c requests a new certificate from an address A_k while there are still unexpired certificates for A_k. Then the control plane revokes[2] any certificates for A_k whose clients are not currently logged in, and it asks the remaining clients to upload *at the same time* as c. It then measures the aggregate bandwidth C of all the uploads and issues c a certificate for the difference between C and the sum of the capacities from the existing certificates. To defend this mechanism against malicious clients that attempt to overload the control plane with requests for new certificates, clients can be required to solve a puzzle [27] before submitting a request; the difficulty of the puzzle can be a function of the current load on the control plane.

In summary, the infrastructure ensures that there can be only one valid certificate per IP address at a time, and that an adversary with an aggregate capacity C cannot obtain certificates whose aggregate capacity exceeds C. Additional resource tests could be implemented and the results included in the resource certificate.

[1]Note that this requires two-way communication and thus prevents malicious clients from obtaining certificates for spoofed addresses.

[2]In our setting, revocation is comparatively easy because each client has to show its current certificate to the control plane when logging in.

6.4 Tamper-evident log

RCA requires clients to maintain a tamper-evident log of all the messages they send and receive. Unlike previous implementations designed for decentralized systems [20], a hybrid system requires different tradeoffs. On the one hand, logs are audited exclusively by the infrastructure, which simplifies the implementation. On the other hand, we need to aggressively minimize the overhead for the infrastructure—particularly the number of cryptographic signatures it has to verify—since we expect the number of clients to be orders of magnitude higher than the number of infrastructure nodes.

Each client maintains a log of entries $e_k := (h_k, s_k, t_k, c_k)$, where h_k is a hash value, s_k a sequence number, t_k an entry type (SEND or RECV), and c_k some type-specific content. The hash values form a hash chain of the form $h_k := H(h_{k-1}||s_k||t_k||c_k)$. Whenever a node i sends a message, it must attach an *authenticator* $(s_k, h_k, \sigma_i(s_k \,||\, h_k))$, which is signed with i's private key σ_i and represents a commitment to the current state of i's hash chain. Each message must be acknowledged, and clients may have at most $\mathrm{n}_{\mathrm{max}}$ unacknowledged messages in flight at any given point in time. Finally, each message or acknowledgment contains enough information to verify that its transmission has been recorded in the log. If i forges, omits, or tampers with log entries after the fact, the infrastructure can detect this by comparing the log i has uploaded to the authenticators i has sent to its peers.

RCA's tamper-evident log maintains sub-chains for each pair of communicating peers. As a result, RCA's authenticators are cumulative, i.e., the authenticator in a message or acknowledgment from i can be used to verify all previous messages or acknowledgments from i, respectively. Hence, each client need only keep one pair of authenticators for each other peer it has communicated with–rather than one for each message it has sent or received–which dramatically reduces the number of authenticators that must be uploaded to, and verified by, the infrastructure.

In summary, the tamper-evident log ensures that inconsistencies between logs can be attributed to a specific misbehaving client.

6.5 Consistency checking

When a client i is ready to upload its log λ_i, it signs λ_i with its private key σ_i, and it attaches its certificate Γ_i as well as the set A_i of authenticators it has collected from other nodes. The infrastructure must then check the log for consistency and plausibility.

At first glance, the consistency check seems to require cross-checking logs from different clients. However, RCA's tamper-evident log is structured such that, for each message transmission, *both* endpoints have sufficient evidence (specifically, an authenticator from the message or its acknowledgment) to show that the entry in the local endpoint's log is consistent with the entry in the remote endpoint's log. Thus, logs can be checked individually, which makes the log checking both efficient and trivially scalable.

Although i uploads the above information to a specific infrastructure node, other infrastructure nodes may also require information about i, namely authenticators or a certificate revocation. Before checking can begin, all information about i must be collected at a single node $H(i)$, which can be chosen, e.g., via consistent hashing. This requires a 'shuffle' step (similar to the one in MapReduce) in which each infrastructure node sends copies of its received authenticators to the nodes that are 'responsible' for them.

Next, the infrastructure inspects λ_i and checks whether a) the log is well-formed and signed with σ_i; b) the certificate Γ_i is valid and matches σ_i; c) Γ_i was not expired or revoked when the log was signed, d) at no point in the log were there more than n_{max} unacknowledged messages, e) each of the sub-hashchains is intact, f) the end of each sub-hashchain corresponds to one of the authenticators uploaded by i, and g) λ_i is consistent with all authenticators signed with σ_i. The last check can be done incrementally if more authenticators are uploaded. If any of the above checks fails, i is clearly faulty. The GUID of a faulty client is immediately disabled so that it cannot participate in peer-to-peer transactions or infrastructure downloads; also, the system operator is notified.

6.6 Plausibility checking

When a client's log passes the consistency check, we know that its recorded sequence of messages is consistent with the log of other clients. However, the messages in the log do not necessarily correspond to a valid execution of the client software. To detect misbehaving nodes, RCA checks each log to see if it satisfies the following invariants, which capture the essence of RCA's swarming protocol. Specifically, clients

1. may only exchange data with peers or edge servers that were suggested to them by the infrastructure;
2. may only serve data they have already downloaded;
3. must not modify blocks before serving them;
4. must serve blocks they have available (i.e., blocks they store and for which peering is enabled); and
5. may only request blocks they do not already store.

If the log does not satisfy all of the invariants, RCA disables the GUID of the client and notifies the system operator. Otherwise, RCA identifies, for each uploaded data block, the content provider that owns the corresponding file, and it tallies, for each content provider, the number of bytes that were uploaded on its behalf, minus any bytes that would exceed the bandwidth in the client's resource certificate.

6.7 Statistical tests and quarantine

RCA's control plane continually maintains statistics about the download and upload activities of each client, such as its IP address, its geolocation, or the number of bytes downloaded and uploaded during the last k days. The control plane uses this data and a set of statistical tests to identify anomalous clients.

When a client i is flagged as anomalous, the infrastructure quarantines i and redirects any future download requests from i to the infrastructure nodes. The client will continue to receive service, however; RCA merely ensures that any interactions with i involve at least one trusted endpoint, so i's actions can be accounted accurately. (Logs produced by i before the quarantine will still be accepted, provided that they pass all the other tests.) In other cases, the infrastructure merely notifies a human operator for resolution.

There are many kinds of statistical tests that could be useful. In Section 7.7, we describe and validate a small set of statistical tests for the NetSession system, and in Section 8, we discuss other tests that could be applied.

6.8 Limitations

RCA's tamper-evident log is only guaranteed to detect inconsistencies in message exchanges when at least one of the two endpoints is an honest node; if the infrastructure (unknowingly) pairs up two colluding clients, one of them can claim to have downloaded a large part of the file from the other without actually having done so. Controlling client pairing, applying statistical tests, and quarantining help to mitigate this limitation.

A second limitation is related to the use of anomaly detection. Since the operator usually does not know which clients are malicious, he can only base the statistical test on the observed behavior of all clients, which is only safe as long as the fraction f of clients controlled by a single adversary is small. If f is large, the adversary can slowly change the behavior of his clients over the course of several weeks or months, analogous to a frog-boiling attack [12]; this might prompt the operator to adjust the statistical tests, which would progressively relax the constraints on the adversary. However, we expect that in practice, few adversaries would have both the required number of clients and the necessary patience.

7 Evaluation

To evaluate RCA, we implemented a clone of the NetSession client and infrastructure software, called

NetSession-Base, which includes all functionality required for our experiments. NetSession-Base is complete enough to run on the Internet. However, we perform most of our experiments in a network emulation environment, which can run hundreds of NetSession-Base clients on a single machine. The network emulator models bandwidth (upstream and downstream) and propagation delay, but not packet loss. Emulations are driven by a trace that defines the node characteristics (geolocation, link capacities, IP and GUIDs) as well as the workload, i.e., the downloads and their precise timing.

We then added RCA's defenses, including the tamper-evident log, the consistency and plausibility checks, the statistical checks, and the client quarantine. We use RSA with 1024-bit keys for the cryptographic signatures. We will refer to the system with defenses enabled as NetSession-RCA.

7.1 Validation

The goal of our first experiment is to verify that our clone matches the behavior of the Akamai NetSession system closely enough so that we can use the NetSession-Base system as a baseline in subsequent experiments. For this purpose, we used Akamai's NetSession client to download a 760 MB file in the live system, and used Wireshark to capture the network traffic from and to the client.

From the captured network traffic, we then compiled a trace that replicates this download in the emulator, such that the same proportion of data are downloaded from the same number of peers and the infrastructure. We then ran NetSession-Base using this trace, and we measured the client's control and data traffic exchanged with each peer and the infrastructure. The results were all within 1% of those obtained with Akamai's NetSession.

7.2 Experimental setup

For the following experiments, we used a 30-day trace from Akamai's live NetSession system recorded in December, 2010. The trace includes an identifier and size for each object requested, the time when the download was initiated and completed, and the number of bytes downloaded from other peers. Our network emulator cannot scale to the entire workload recorded in the trace, so we used a sample that includes all downloads initiated by a randomly chosen subset of 500 clients.

We assigned client link capacities by randomly sampling from a measured distribution of download and upload speeds in residential broadband networks [16].

The NetSession traces do not record how many and which peers were actually used in a download, or how many bytes were obtained from each. In any case, because our emulation only includes a small sample of the actual peers, it would not include many of the peers who

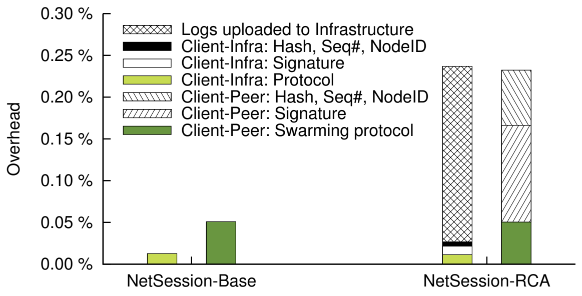

Figure 3: Network traffic overhead, normalized to the size of the downloaded content. For each system, the figure shows the communication with the infrastructure (left bar) and with peers (right bar).

actually uploaded to one of the peers in the sample. Instead, our emulation assumes that all peers in our sample can serve all files to other peers. The infrastructure suggests a random set of emulated peers for each download, and the swarming protocol dynamically requests content from this set of peers based on the observed bandwidth. Since our evaluation is not concerned with the dynamics of the swarming protocol, this approximation does not affect the results. To be conservative, we fixed the overall ratio of bytes downloaded from peers versus bytes downloaded from the infrastructure to 80%; based on our observations from the trace, this will overestimate the overhead of our system.

7.3 Cost: Traffic

To quantify the additional bandwidth requirements of NetSession-RCA, we measured a) the total number of bytes downloaded as actual payload by all clients, and b) total number of bytes transmitted in the system. The difference between the two numbers is an estimate of the bandwidth overhead of the system relative to the actual payload bytes; it was 0.06% for NetSession-Base and 0.47% for NetSession-RCA. This amounts to a 7.8-fold increase in bandwidth overhead for NetSession-RCA.

While the relative increase in overhead is substantial, it is important to note that the absolute bandwidth requirement is still modest. The average per-peer bandwidth requirement is only 192 KB/day for NetSession-RCA and 26 KB/day for NetSession-Base. In return, NetSession-RCA provides much more fine-grained and reliable information about peer behavior than NetSession-Base.

Figure 3 shows a more detailed breakdown of the results. Both NetSession-RCA and NetSession-Base exchange some control messages with peers and with the infrastructure; the corresponding amount of traffic is small and identical in both systems. RCA adds an authenticator and an acknowledgment for each message. The overhead is higher for the infrastructure traffic because the number of messages is higher: in addition to the data blocks, this traffic also contains a number of small control messages. Finally, clients must upload their logs to the infrastructure.

7.4 Cost: CPU

NetSession-RCA requires more client CPU than NetSession-Base, because it must generate and verify the signatures in authenticators. Because NetSession is intended to run in the background without inconveniencing the user, this additional computation must not consume more than a small fraction of the CPU.

To estimate the cost, we measured the number of signature generations and verifications performed by clients as part of the download activity. We then benchmarked RSA-1024 signature generation and verification on a single core of an Intel Xeon X5650 CPU, and we used these benchmarks to estimate the additional CPU load that would be caused by these operations. The maximum additional CPU load over all clients was never more than 0.5%.

7.5 Cost: Log storage and log upload

NetSession requires each client to maintain a log, and to periodically upload this log to the control plane. However, NetSession-RCA's log is considerably more detailed because it keeps track of individual messages, whereas NetSession-Base's log merely records occasional download progress reports. In both cases, the exact size depends on the client's activity.

To quantify how much log data is generated, we ran NetSession-Base and NetSession-RCA with log uploading disabled; thus, each of 500 clients kept its *entire* 30-day log on its local disk. We then examined the log sizes at the end of the experiment. The average log size was 2.5 MB; the 5th and 95th percentiles were at 1.4 MB and 3.4 MB, respectively. The largest log was 86 MB. If we (conservatively) estimate the average download activity per client at 1 GB per month, a larger deployment with 100 million GUIDs would generate about 1.8 TB of logs per day. This corresponds to only 18 kB of logs per client per day.

Figure 4 provides a detailed breakdown of the log contents collected from the clients. A comparison with Figure 3 shows that the log is considerably smaller than a complete message trace; this is because a) the log contains only a hash of each data block rather than the actual bytes (which are known to the infrastructure anyway), and b) the log does not contain every single authenticator, but only the most recent one for each client.

Each client periodically uploads its log to the control plane and then deletes the entries once they have been acknowledged. Thus, the amount of storage that is needed locally on each client depends on the upload interval.

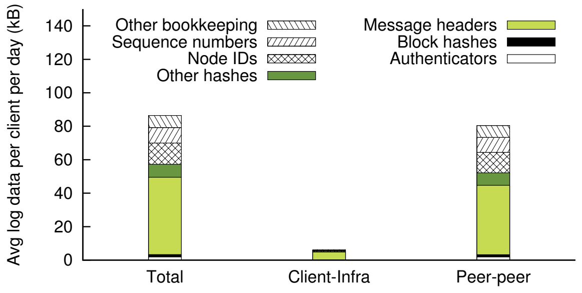

Figure 4: Average log size per client

With daily uploads, less than a MB of storage is required. Since every logged byte must eventually be uploaded, the amount of network traffic generated by the uploads is largely independent of the upload interval.

7.6 Cost: Log processing

Once the logs have been uploaded, the infrastructure must perform the consistency and plausibility checks described in Section 6. The required processing time depends on what actions are recorded in each log, but we expect it to be correlated with the overall log size.

To estimate the overhead, we performed the consistency and invariant checks on each of the logs produced by the clients. We measured the total processing time, as well as the fraction of time spent on consistency and invariant checks. Since we expect cryptographic operations to be a major factor, we separately measured the time spent verifying signatures (recall that the infrastructure does not generate signatures).

On average, about 0.78 MB's worth of log data was processed per second on a single CPU. 9% of the time was spent on consistency checking and 91% on invariant checking; overall, signature verifications accounted for about 3% of the processing time. Log processing can easily be parallelized, e.g., using MapReduce.

Based on these results, we estimate that a deployment with 100 million GUIDs would require about 28 extra machines to process the logs. For comparison, NetSession's current log processing system requires around 10 machines. Both estimates assume that the machines are fully utilized; in practice, log processing is one of several jobs that runs on a larger cluster.

7.7 Examples of statistical tests

Developing a full set of statistical tests for NetSession would require a detailed characterization of its workload, which is beyond the scope of the present paper. However, we use the set of simple tests in Table 2 to illustrate the general principle. These tests are fully automated; they are designed to constrain clients who collude to over-report uploads or deliver bad service to peers. We expect that more sophisticated tests will be based on de-

#	Which peers are quarantined?	Parameters	+Load
T1	IP has downloaded $> N_1$ bytes during the last k_1 days	$k_1 = 20$ $N_1 = 15.2G$	1.00%
T2	IP has been used by $> g_2$ GUIDs during the last k_2 days	$k_2 = 1$ $g_2 = 2$	0.57%
T3	GUID has downloaded $> N_3$ bytes during the last k_3 days	$k_3 = 20$ $N_3 = 10.4G$	0.99%
T4	GUID has downloaded $> N_4$ files during the last k_4 days	$k_4 = 20$ $N_4 = 140$	0.92%
T5	GUID failed to validate $> N_5$ bytes during the last k_5 days	$k_5 = 1$ $N_5 = 200K$	1.00%
	Tests T1–T5 combined		2.81%

Table 2: Statistical tests used with NetSession-RCA, along with the additional load (#bytes served) they place on the infrastructure due to quarantined clients.

tailed workload characteristics, e.g., channel switching patterns [11] or the clients' response to the quarantine.

More aggressive tests reduce the amount of misbehavior an adversary can get away with, but they also increase the load on the infrastructure due to false positives. To give a rough impression of how aggressive our simple tests could be, we used the 12/2010 trace to determine, for each test, the set of parameters that a) causes at most 1% additional load on the infrastructure, and among those, the one that b) constrains the adversary the most. These parameters, and the resulting load increases, are also shown in Table 2. Note that a given client can trigger more than one test; hence, the load increase from a set of tests is lower than the sum of the increases from the individual tests.

7.8 Effectivity

As a sanity check for the NetSession-RCA implementation, we injected a set of sample attacks. Specifically, we injected five inflation attacks and one corruption attack: In *blatant liars*, one client uploaded a fabricated log claiming to have downloaded 1 TB. In *collusion*, two clients continuously requested files and then reported that they had downloaded them from each other, regardless of which clients the infrastructure suggested. In *flash mob*, five clients joined the system simultaneously and rapidly requested rare files, 'faking' downloads whenever the infrastructure paired up two of them. In *leechers*, five clients joined the system simultaneously and downloaded random files as quickly as possible. In *Sybil attack*, one client joined the system with five GUIDs; the first GUID downloaded a rare file, and the others then tried to download the same file from each other. Finally, in *confused clients*, one client uploaded malformed log entries.

In all cases, the system behaved as expected. Logs from faulty clients were discarded, clients with abnormal behavior were quarantined, and the uploads affected by flash mob and Sybil attackers were effectively capped. In particular, *blatant liars* and *confused clients* were caught by our consistency checks (section 6.5), *collusion* was detected by the plausibility checks (section 6.6), *flash mob* was identified by resource certificates (section 6.3), and finally *leechers* and *Sybil attack* were flagged by statistical checks (section 6.7).

8 Statistical tests

As discussed in Section 6.7, RCA uses a set of statistical tests to decide which clients should be quarantined. We suggest the following general approach to choosing a suitable set of tests:

1. Identify the metrics that an attack is likely to affect;
2. Closely characterize the system's normal workload in those metrics; and
3. Choose a threshold for each metric such that, under the normal workload, no more than a small fraction of the clients are above the threshold (and would thus be quarantined).

A detailed characterization of NetSession's workload is beyond the scope of this paper, and is part of our ongoing work. However, for completeness we briefly summarize some of results from our initial investigation below.

One key set of metrics for NetSession is, obviously, the number of downloads and their distribution across the different files. Overall, file popularity in NetSession follows the usual power law, and the workload has the usual diurnal pattern. However, there is a lot more fine structure. For instance, some files are more popular at certain times of the day, and this pattern can vary between content providers; also, load can shift between files in a predictable pattern. To avoid being quarantined, an attacker would have to 'blend in' and closely imitate the current request pattern. This would be difficult, however, because of the information asymmetry that is inherent in hybrid systems like NetSession: the pattern is trivial to observe for the infrastructure but can only be approximated by an attacker.

Another set of metrics is related to the location of the clients, which can be obtained from a geolocation service such as Akamai's EdgeScape. The popularity of content can vary between regions, and it would be anomalous if content that is usually popular in the Middle East were to suddenly become popular in South America. Location-based metrics are particularly interesting because attackers cannot easily choose the location of compromised clients. Even botnets, a potential source of compromised clients, are often biased towards particular regions [32].

A third set of metrics is related to the download topology. Consider a graph with a vertex for each client and an edge for each pair of clients that have downloaded from one another. In this graph, which can easily be constructed by the control plane, a set of colluding peers would show up as a tightly connected subgraph. Efficient heuristics for detecting such subgraphs have already been developed for botnet detection, e.g., in BotGrep [26]. The control plane could use such techniques to detect suspicious subgraphs of clients, and then quarantine these clients or redirect their download requests to other, unrelated peers.

We note that the above list is not meant to be exhaustive or universal. A rich literature of other anomaly detection techniques exists, which could be applied to NetSession. On the other hand, the specific methods mentioned above may not be appropriate for all hybrid systems.

9 Conclusion

In this paper, we have examined a fundamental challenge in P2P-infrastructure hybrids: how to reliably account for the actions of untrusted clients. In current hybrid systems, malicious peers can report fictitious content downloads and degrade the system's quality of service. We described and evaluated RCA, a system that leverages the unique characteristics of P2P-infrastructure hybrids to limit the loss of accounting accuracy and service quality resulting from faulty or malicious clients. RCA reliably discovers all misreporting and protocol violations by individual clients, and it can automatically quarantine potentially colluding clients, at a moderate cost in terms of bandwidth and load on the infrastructure.

Acknowledgments

We thank our shepherd John Byers and the anonymous reviewers for their comments and suggestions. We also thank Miroslav Ponec and James Chalfant from Akamai Technologies for helpful comments on earlier drafts of this paper. This work was supported in part by NSF grants CNS-1040672 and CNS-1054229 and by the Max Planck Society.

References

[1] Joost. `http://www.joost.com/`.

[2] Limelight networks. `http://www.limelight.com/`.

[3] Octoshape. `http://www.octoshape.com/`.

[4] Pando networks. `http://www.pandonetworks.com/`.

[5] Velocix P2P assisted delivery. `http://www.velocix.com/network_delivery.php`.

[6] A. Adya, W. J. Bolosky, M. Castro, G. Cermak, R. Chaiken, J. R. Douceur, J. Howell, J. R. Lorch, M. Theimer, and R. P. Wattenhofer. FARSITE: Federated, available, and reliable storage for an incompletely trusted environment. In *Proc. OSDI*, Dec. 2002.

[7] S. Agarwal and J. R. Lorch. Matchmaking for online games and other latency-sensitive P2P systems. In *Proc. SIGCOMM*, 2009.

[8] Akamai acquires Red Swoosh. `http://www.akamai.com/html/about/press/releases/2007/press_041207.html`, Apr. 2007.

[9] M. Barreno, B. Nelson, R. Sears, A. D. Joseph, and J. D. Tygar. Can machine learning be secure? In *Proc. AsiaCCS*, 2006.

[10] R. A. Bazzi and G. Konjevod. On the establishment of distinct identities in overlay networks. In *PODC*, pages 312–320, 2005.

[11] M. Cha, P. Rodriguez, J. Crowcroft, S. Moon, and X. Amatriain. Watching television over an IP network. In *Proc. IMC*, 2008.

[12] E. Chan-Tin, D. Feldman, Y. Kim, and N. Hopper. The frog-boiling attack: Limitations of anomaly detection for secure network coordinates. In *Proc. SecureComm*, 2009.

[13] V. Chandola, A. Banerjee, and V. Kumar. Anomaly detection: A survey. *ACM Comput. Surv.*, 41:15:1–15:58, July 2009.

[14] B. Cohen. Incentives build robustness in BitTorrent. In *Proc. P2PEcon*, June 2003.

[15] D. E. Denning. An intrusion-detection model. *IEEE Trans. on Software Engineering*, 13(2):222–232, 1987.

[16] M. Dischinger, A. Haeberlen, K. P. Gummadi, and S. Saroiu. Characterizing residential broadband networks. In *Proc. IMC*, Oct 2007.

[17] J. R. Douceur. The Sybil attack. In *Proc. IPTPS*, Mar 2002.

[18] M. J. Freedman and R. Morris. Tarzan: a peer-to-peer anonymizing network layer. In *Proc. ACM CCS*, 2002.

[19] T. Garfinkel, B. Pfaff, J. Chow, M. Rosenblum, and D. Boneh. Terra: A virtual machine-based platform for trusted computing. In *Proc. SOSP*, Oct. 2003.

[20] A. Haeberlen, P. Kuznetsov, and P. Druschel. PeerReview: Practical accountability for distributed systems. In *SOSP*, Oct. 2007.

[21] C. Huang, A. Wang, J. Li, and K. W. Ross. Understanding hybrid CDN-P2P: why Limelight needs its own Red Swoosh. In *Proc. NOSSDAV*, 2008.

[22] T. Karagiannis, P. Rodriguez, and K. Papagiannaki. Should Internet service providers fear peer-assisted content distribution? In *Proc. IMC*, 2005.

[23] D. M. Lewin, B. Maggs, and J. J. Kloninger. Internet Content Delivery Service with Third Party Cache Interface Support. U.S. Patent Number 7,010,578, Mar. 2006.

[24] Q. Lian, Z. Zhang, M. Yang, B. Y. Zhao, Y. Dai, and X. Li. An empirical study of collusion behavior in the Maze P2P file-sharing system. In *Proc. ICDCS*, 2007.

[25] N. B. Margolin and B. N. Levine. Financial cryptography and data security; Chapter "Quantifying resistance to the Sybil attack". Springer-Verlag, 2008.

[26] S. Nagaraja, P. Mittal, C.-Y. Hong, M. Caesar, and N. Borisov. Botgrep: finding P2P bots with structured graph analysis. In *Proceedings of the 19th USENIX conference on Security*, Aug. 2010.

[27] B. Parno, D. Wendlandt, E. Shi, A. Perrig, B. M. Maggs, and Y.-C. Hu. Portcullis: Protecting connection setup from denial-of-capability attacks. In *Proc. SIGCOMM*, 2007.

[28] R. S. Peterson and E. G. Sirer. Antfarm: efficient content distribution with managed swarms. In *Proc. NSDI*, 2009.

[29] Major features in PunkBuster. `http://www.evenbalance.com/index.php?page=info.php`.

[30] S. Seuken and D. C. Parkes. On the Sybil-proofness of accounting mechanisms. In *Proc. NetEcon*, June 2011.

[31] M. Sirivianos, J. H. Park, X. Yang, and S. Jarecki. Dandelion: Cooperative content distribution with robust incentives. In *Proc. USENIX ATC*, June 2007.

[32] B. Stone-Gross, M. Cova, L. Cavallaro, B. Gilbert, M. Szydlowski, R. Kemmerer, C. Kruegel, and G. Vigna. Your botnet is my botnet: analysis of a botnet takeover. In *Proc. CCS*, 2009.

[33] L. Vu, I. Gupta, K. Nahrstedt, and J. Liang. Understanding overlay characteristics of a large-scale peer-to-peer IPTV system. *ACM Trans. Multim. Comp. Comm. Appl.*, 6:31:1–31:24, 2010.

[34] C. Wu, B. Li, and S. Zhao. Diagnosing network-wide P2P live streaming inefficiencies. In *Proc. IEEE INFOCOM*, Apr. 2009.

[35] H. Yin, X. Liu, T. Zhan, V. Sekar, F. Qiu, C. Lin, H. Zhang, and B. Li. Design and deployment of a hybrid CDN-P2P system for live video streaming: experiences with LiveSky. In *Proc. ACM MM*, 2009.

[36] H. Yu, P. B. Gibbons, M. Kaminsky, and F. Xiao. SybilLimit: A near-optimal social network defense against Sybil attacks. In *Proc. IEEE S&P*, 2008.

[37] H. Yu, M. Kaminsky, P. B. Gibbons, and A. Flaxman. Sybilguard: defending against Sybil attacks via social networks. In *Proc. SIGCOMM '06*, Aug. 2006.