Thinning Akamai

Ao-Jan Su and Aleksandar Kuzmanovic
Department of Electrical Engineering & Computer Science
Northwestern University, Evanston, IL 60208, USA
ajsu@cs.northwestern.edu, akuzma@cs.northwestern.edu

ABSTRACT

Global-scale Content Distribution Networks (CDNs), such as Akamai, distribute thousands of servers worldwide providing a highly reliable service to their customers. Not only has reliability been one of the main design goals for such systems — they are engineered to operate under severe and constantly changing number of server failures occurring at all times. Consequently, in addition to being resilient to component or network outages, CDNs are inherently considered resilient to denial-of-service (DoS) attacks as well.

In this paper, we focus on Akamai's (audio and video) streaming service and demonstrate that the current system design is highly vulnerable to intentional service degradations. We show that (*i*) the discrepancy among streaming flows' lifetimes and DNS redirection timescales, (*ii*) the lack of isolation among customers and services, (*e.g.*, video on demand *vs.* live streaming), (*iii*) a highly transparent system design, (*iv*) a strong bias in the stream popularity, and (*v*) minimal clients' tolerance for low-quality viewing experiences, are all factors that make intentional service degradations highly feasible. We demonstrate that it is possible to impact arbitrary customers' streams in arbitrary network regions: not only by targeting appropriate points at the streaming network's edge, but by effectively provoking resource bottlenecks at a much higher level in Akamai's multicast hierarchy. We provide countermeasures to help avoid such vulnerabilities and discuss how lessons learned from this research could be applied to improve DoS-resiliency of large-scale distributed and networked systems in general.

Categories and Subject Descriptors

C.2.2 [**Computer-Communication Networks**]: Internet
C.4 [**Performance of Systems**]**:** Reliability, availability, and serviceability

IMC'08, October 20–22, 2008, Vouliagmeni, Greece.

General Terms

Security, Measurement, Experimentation

Keywords

Streaming, Akamai, CDN, Denial of service

1. INTRODUCTION

Streaming is thriving in the Internet. A recent study shows that as a result of streaming audio and video in web downloads, HTTP took back the leading position from peer-to-peer (p2p) applications for the first time in the last four years [19]. Streaming alone accounts for more than 20% of the total Internet traffic [19], and it is expected that new applications, such as Internet video and TV (*e.g.*, [5, 7, 10]), will further accelerate this trend over the coming years.

One of the key driving forces standing behind the success of streaming applications in the Internet is certainly their *quality*. Due to penetration of high-speed broadband access technologies and improved streaming dissemination techniques, the quality bars have been raised quite high. Indeed, a recent study conducted by `Akamai Technologies` explored fundamental elements related to the future success of online video: consumer preferences around video consumption and consumer reaction to low-quality viewing experiences. The most compelling results reveal that, having experienced poor video performance at an Internet site, more than half of online video users would seek content from a competing website, and a quarter would leave with a more negative brand perception and be less likely to return to the poorly performing site [13].

This apparently opens the doors for denial-of-service (DoS) attacks against streaming services. Indeed, generating even short server or network outages, or reducing the encoding rate can cause a stream's quality to degrade, producing "glitches," "slide-shows," and "freeze ups" as the user watches the stream. This, in turn, can dramatically impact clients' perception and can cause them to switch the channel or simply give up [13].

Incentives for conducting such misbehaviors are manifold. In addition to the rough competition among streaming content providers, other scenarios are possible as well. For example, many political and sport events are frequently streamed over the Internet nowadays, and opposing parties might be tempted to disrupt such broadcasts on the Internet, *e.g.*, interrupt the broadcast of a political speech on

CNN's web site to clients on the East Coast, or disrupt the broadcast of a basketball game on NBA's pay-per-view site on the West Coast. Unfortunately, such scenarios are not the matter of a distant future; we demonstrate that such sophisticated attacks could be launched *today*.

In this paper, we focus on Akamai's streaming architecture and its resilience to DoS attacks. While our experiments and evaluations are necessarily tied to Akamai's streaming infrastructure, the lessons learned from our study can be generalized not only to other DNS-driven and multicast streaming services, but can have important impact on the design and security of distributed and networked systems in general, as we discuss later in the paper. Akamai's streaming network is one of the largest streaming infrastructures in the world, capable of serving close to a million streams concurrently [3].[1] In addition to the built-in resilience to network and server failures [14], the architecture is characterized by several other desirable properties. Unlike p2p-based streaming architectures (*e.g.*, `ESM` [4]), which are vulnerable to misbehaving peers (*e.g.*, [28]), no clients are directly involved in distributing content in Akamai's case [22]. Unlike data-center-oriented systems (*e.g.*, `YouTube` [9]) that have a single point of failure (a data center itself), Akamai's streaming network aggressively distributes content all around the world; hence, no single point of failure exists.

Contrary to the common belief, we find that it is highly feasible to degrade service quality to arbitrary flows in arbitrary parts of the Akamai's streaming network. The key issue is that the redirection time-scales used for load balancing by DNS-driven systems are fundamentally inappropriate for live streaming. In particular, when a server (or the network access link) experiences increased load, the redirection might help the newly arriving clients to avoid the problem, but *not* the clients that are already fetching their streams from the given server.[2] Thus, contrary to the web case, where DNS redirections at time scales of tens of seconds can effectively help reduce the load from the troubled server, such an approach does not work for streaming, particularly when the system is under attack. This is because the flow lifetimes are fundamentally different for streaming and the web. Moreover, clients' tolerance for the former is dramatically thinner [13].

Unfortunately, other problems exist as well. In particular, streams belonging to different customers, channels, and services (*e.g.*, live audio, live video, or video on demand) all share the same infrastructure, and we show that no strong spatial nor temporal isolation among them exists, neither at the server nor at the network level. The combination of this problem and the above slow load balancing problem causes additional security implications. Because both popular and unpopular streams overlap at the same servers, it is possible to generate traffic surges by requesting unpopular streams, thus dramatically impacting the popular ones.

A related issue is that global-scale streaming services distribute streaming servers to edge regions which typically have limited, often moderate bandwith (*e.g.*, 100 Mbps or less), shared by the rest of the ISPs' traffic. As a result, only moderate attacker resources are needed to generate traffic surges and impact a streaming service at a particular edge region. Even though these attacks may not "melt down" the entire streaming service globally, they can dramatically endanger the reputation of a streaming service by targeting a desired customer's stream (*e.g.*, broadcast of a popular event) at a given network region.

In an attempt to verify the above hypotheses, while taking enormous care not to cause any problems to Akamai's clients, we perform Internet experiments. To excite Akamai's bottlenecks, we carefully and gradually increase the request rate for unique unpopular streams from appropriate Akamai servers. Whenever the bottleneck reaches its limit, we instantly abort the experiment. We verify the slow load balancing problem, and demonstrate that a stream's throughput can get thinned (*e.g.*, from 1.4 Mbps to 200 kbps) when the bottleneck reaches its capacity. Moreover, a highly transparent system design that feeds important *internal* information to the public (via URLs), opens the doors to additional unforeseen problems. We show that it is feasible to effectively exploit the transparent system design and excite resource bottlenecks not only at the streaming network's edge, but at a much higher level in the Akamai's multicast hierarchy: at reflectors or even at content providers' origin servers.

Providing a single comprehensive solution to the above problems is challenging for a number of reasons. Hence, we provide a set of countermeasures with the goal to significantly increase the bar for potential attackers rather than provide a "bullet-proof" solution. First, we argue that resource-based or graphic-puzzles based admission control schemes are either inappropriate or incapable of solving the problem. Second, we argue that *location-aware* admission control can dramatically raise the bar for the attackers: force them to use botnets, instead of a few high-bandwidth machines as we did. Third, we argue that a more careful edge cluster configuration, and most importantly, a less-transparent system design that is capable of hiding important internal information from the public, can dramatically raise the system's resilience to even botnet-equipped attackers. Finally, we discuss the tradeoffs between security and system transparency in networked and distributed systems in general.

This paper is structured as follows. In Section 2, we summarize a DNS-driven streaming architecture and illucidate the key vulnerabilities of this system. In Section 3 we perform a measurement study that reveals the security implications of this architecture, and we show how they can be exploited in Section 4. We provide design guidelines to avoid these exploits in Section 5. Finally, we conclude in Section 6.

2. DNS-BASED STREAMING SERVICES: BACKGROUND AND VULNERABILITIES

Here, we provide the necessary background on Akamai's DNS-driven streaming infrastructure. Then, we outline the main vulnerabilities characterstic not only for this particular infrastructure, but for large-scale DNS-driven and multicast streaming systems in general.

2.1 Background

In order to serve millions of audio, video, on-demand, and live streaming clients globally, Akamai designed and deployed an overlay streaming multicast network [14].

[1] Monthly peak 974,296 streams on May 17, 2007 [3].

[2] Even the newly arriving flows might end up redirected to a distant backup server, *e.g.*, to a different continent, with increased probability to experience poorer viewing quality.

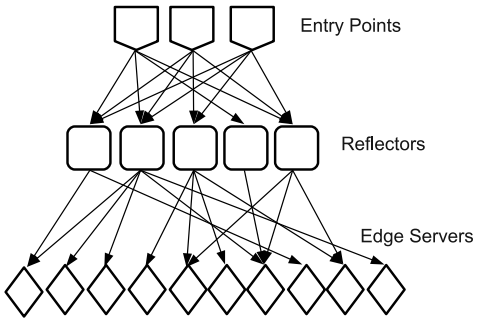


Figure 1: Akamai's overlay multicast streaming network

Figure 1 illustrates an abstract view of this streaming architecture. At the source, content providers (*e.g.*, a radio or a TV station) encode their streams and transfer them to the so-called *entry points* of the Akamai's streaming network. To distribute streams to a large number of regions in a scalable manner (*e.g.*, [26]), the streams are then replicated to multiple *set reflectors* [22]. Set reflectors in turn propagate the streams to *edge servers*. Finally, edge servers stream content to clients (not shown in the Figure).

Data in the multicast network is transferred via UDP. To tolerate network transfer errors, reflectors and edge servers *may* receive multiple copies of each packet and reassemble the data streams by pruning duplicate packets. Still, replicating each of the streams to each of the reflectors and edge servers is simply not feasible. This is because such an approach would overload set reflectors. Moreover, replicating all streams to all reflectors and edge servers is really not needed, because not all streams are equally popular [22].

Subscription system. To address this problem, the streaming CDN adopts a *reflector subscription system*. It allows set reflectors to propagate streams upon requests from downstream edge servers. Indeed, the subscription approach ensures that only watched streams get propagated to edge regions [22]. Still, even the development of a subscription system does not fully remove the potential to overload set reflectors. For example, if all popular edge regions subscribe to the same set reflector, they could overload that machine or a set of machines, even though the set reflector subsystem as a whole has plenty of spare capacity [22].

Portsets. To further address the problem, the approach is to group streams into buckets called *portsets*.[3] Then, it assigns portsets to different set reflectors to ensure the load is distributed appropriately. Indeed, a portset is no more than a collection of streams that is supposed to be transported through the same set reflectors [22]. Akamai groups popular and unpopular streams in the same portsets. Such an approach provides good load balancing capabilities. Likewise, when the load in a given portset starts growing, the system can adapt to the growth.

DNS-driven system. Following the successful design applied in the case of web,[4] all the mappings in the Akamai's streaming architecture, *i.e.*, those between clients and edge servers and between edge servers and reflectors are done via DNS. Below, we exemplify the first scenario — DNS-driven mapping between clients and edge servers.

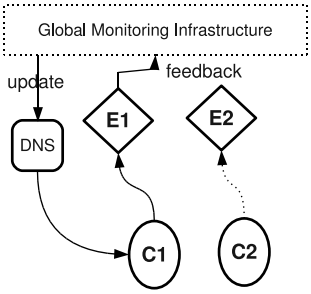


Figure 2: DNS-based load balancing

Figure 2 illustrates the DNS load balancing approach. When the monitoring infrastructure observes network or server overloading conditions of an edge server ($E1$), it updates appropriate DNS entry to redirect new clients to another edge server ($E2$). In this way, the system reduces the load placed on edge server $E1$ and helps newly arriving clients get better service from the server ($E2$). The critical issue, however, is the timescale at which the redirection takes place, which has significant security repercussions, as we first explain below.

2.2 Vulnerabilities

2.2.1 Slow Load Balancing

Here, we explain the slow load-balancing problem. DNS' Time-To-Live (TTL) value defines how frequently a client should query DNS system to get current IP addresses of a hostname. On the one hand, in DNS-driven systems, large TTL can lead to slow response to changes of network or server conditions [27]. On the other hand, small TTL may put unnecessarily high load on the DNS system. The question then becomes which TTL Akamai sets for their streaming service. While we answer this question in the next section, our key hypothesis is that if the redirections at time scales of several *tens of seconds* are applied, as used in the case of web [27], that might generate severe vulnerabilities in the case of live streaming.

Indeed, while DNS redirections at such time scales are capable of efficiently distributing the load in the case of web traffic, this is not the case for streaming for the following two reasons. First, once an edge server becomes overloaded in the web case, either due to a flash crowd or a DoS attack, even a slightly delayed DNS redirection can help effectively reduce the load from the troubled edge server. This is because web flows are relatively short-lived, and thus the overload quickly 'goes away' after the redirection. This is not the case with streaming. While a redirection can help new streaming clients to experience better service, the high overload on the original edge server will *not* disappear instantly (unless a subset of clients disconnect from the troubled server), precisely because streaming flows are longer lived.

Second, redirection time-scales at the order of tens of seconds leave a long window size to potential attackers, which can easily overload either network or server resources during

[3]In this paper, we interchangeably use terms 'portset' and 'channel.'

[4]Akamai routinely delivers between ten and twenty percent of all web traffic, at times reaching more than 650 Gigabits per second [3].

such relatively long intervals. Finally, clients are much more tolerable to minor delays in the web performance, which is not a 'live' medium. For example, a slightly delayed image appearance at a web page is not a catastrophic event. On the contrary, this does not hold for streaming where even a slight degradation can affect clients' perception and motivate users to seek content from elsewhere [13].

We explore the redirection timescales applied by the Akamai streaming network in Section 3.2.1, and we experimentally explore the slow load balancing problem in Section 4.2.1.

2.2.2 No Isolation

The idea of deploying a number of servers at the edge is to bring the clients to their closest servers in order to get better services. However, for a limited number of edge servers located at the same region, it is simply impossible to assign dedicated servers to different services. The situation aggravates in streaming services because each streaming connection can occupy significant server and network resource for a long period of time, and therefore impact the performance of other services co-located at the same region. For example, overlapping between media implies that traffic from one media can affect quality of others. Moreover, by artificially increasing traffic load at one media (*e.g.*, VoD), one can affect service to other medias (*e.g.*, live streaming).

This is a critical problem because it provides the way for an attacker to relatively easily collect a large amount of unique active VoD streams. Thus, before a live streaming event, the attackers can get prepared with enough amount of "bullets" to launch the attacks. We explore the level of isolation among different channels, customers, and medias in Section 3.3, and we experimentally explore the repercussions of the isolation problem in Section 4.2.2.

2.2.3 Suboptimal Migration

In a DNS-driven streaming service, when an edge region is overloaded, new clients might get redirected to distant regions, *e.g.*, to different continents. Contrary to the web case, where such redirections are not problematic, this might not be the case for streaming. Longer inter-continental paths might offer smaller bandwidth and poorer viewing experiences, hence turn away clients from accessing such services. We experimentally explore this problem in more depth in Section 4.2.3.

2.2.4 Amplification Attacks

DNS-driven streaming services adopt a multicast tree structure in order to optimize network resources (Figure 1). Data streams first travel to a regional reflector and then duplicate to edge servers. It is important to realize that each unique stream request from edge servers consumes resources from their shared reflector. Hence, if an attacker is capable of reverse-engineering the system to understand which edge servers map to given reflectors, then such an attacker might become capable of exciting a bottleneck at the *reflector level* by exploiting the appropriate edge servers in a given region as proxies.

This is a particularly dangerous vulnerability for the following reasons. First, because the edge servers are not directly attacked, it is hard to detect the attack at edges. Second, because such attacks can affect the service in an entire geographic region served by a given reflector (*i.e.*, all edge servers served by a given reflector), they are very dangerous. Below, in Section 3.1, we first reveal the method used for mapping between edge servers and reflectors. Next, in Section 3.4, we explore the characteristics of edge server clusters that are susceptible to become proxies in such attacks. Finally, in Section 4.3, we experimentally evaluate this issue in the Internet.

3. SCANNING AKAMAI

It is well known that before performing any attacks, one must first scan the system to understand its potentially vulnerable points. While our goal is *not* to attack the system, but rather to prevent anyone from ever becoming capable of conducting such misbehaviors (Section 5), we necessarily first attempt to collect information about the system. In particular, below we first analyze Akamai's streaming URLs which apparently feed a lot of internal system information. Then, we perform large-scale measurements to verify our vulnerability hypotheses explained above.

3.1 Understanding ARLs

Akamai encodes the information about each unique stream in specific URLs called Akamai Resource Locators (ARLs) [12]. In many cases, ARLs are embedded in the web pages by content providers (*e.g.*, `MTV`), and retrieved by clients. An example ARL for a (Microsoft Media Server) live stream is: `mms://a1897.l3072828839.c30728.g.lm.akamaistream.net/D/1897/30728/v0001/reflector:28839`. In other cases, ARLs are not coded into the web pages but embedded in media players (*e.g.*, Flash player) launched by web browsers. For this group, packet sniffing and analysis software (*e.g.*, URL Snooper [8]) can be used to discover ARLs.

From clients to edge servers. Once a client receives this ARL, it sends a request to its DNS server in an attempt to resolve the IP address of the hostname `a1897.l3072828839.c30728.g.lm.akamaistream.net`. The request further gets redirected to the Akamai's DNS system. Based on the client's location and the current network conditions and the load at Akamai's edge servers, the client is redirected to the edge server that should provide the optimal performance.

The example streaming ARL can be 'decoded' as follows: `a1897` is the portset (or channel) number; as explained above, multiple streams can share the same portset; `c30728` represents the customer number — National Basketball Association (NBA, `nba.com`) in this particular case; `lm` indicates live media service; `l3072828839` combines two names: `30728` is again the customer number and `28839` is the stream's unique identification number [1, 2].

From edge servers to reflectors. Once an edge server receives a request from the client for the given stream, it proceeds as follows. If the stream is already 'active' at the given edge *cluster*,[5] it is simply forwarded to the client. If this is not the case, the edge server must determine the appropriate reflector to fetch the stream from.[6] At this point, it enters the similar procedure as when a client queries the Akamai's DNS system to reach the appropriate edge server. In particular, the edge server in a given region queries the

[5] An edge cluster is a set of co-located edge servers.

[6] According to [11], this is not the case for the video-on-demand service, in which case the stream is fetched directly from origin servers. We discuss this issue later in the paper.

ID	Type	Chan.	Cust.ID	Str.ID
NASA	Live Video	a167	c18569	44670
CNN	Live Video	a466	c37606	51364
ABC	Live Video	a151	c10588	43249
NBA1	Live Video	a785	c30728	28857
NBA2	Live Video	a644	c30728	29417
NBA3	Live Video	a1020	c30728	28846
Blockbuster1	VoD	a1247	c26419	e33220
Blockbuster2	VoD	a1042	c26419	e33210
Blockbuster3	VoD	a1081	c26419	b10069
FM94.5	Live Audio	a1367	c20064	43805
FM92.1	Live Audio	a1819	c21650	45129
FM106.7	Live Audio	a774	c19810	44599

Table 1: Measured streams

DNS system *by looking up the name that contains a given portset and the region* [22]. This approach opens the doors to amplification attacks explained above. Indeed, this means that streams from a given edge region that use the same channel map to the same reflector. We discuss this issue in more depth later in Sections 3.4 and 4.3.

DNS is effectively used for load balancing and reducing excessive load from reflectors. If traffic on particular portsets is low, then the DNS names for those portsets can resolve to the same reflector. As traffic grows, then the DNS names can be changed to resolve to different set reflectors for different portsets. This allows DNS-driven streaming systems to scale up or down the reflector network based on customer demand, rather than having to size based on peak demand [22]. Indeed, the flexibility of DNS system has proven invaluable for Akamai's streaming network. Still, we show below that there are downsides as well. In particular, the time-scales both for redirecting clients to edge servers and for redirecting edge servers to reflectors might not be successful enough in reducing the load from overloaded servers, as this is the case with the web traffic.

3.2 Measurements and Implications

Here, we perform a large-scale measurement study that reveals the most vulnerable mechanisms and points in the Akamai's streaming architecture. In particular, we evaluate the vulnerability hypotheses from Section 2.2 above and explore (*i*) the redirection time-scales used for load balancing, (*ii*) the level of isolation among clients, channels, and services, and (*iii*) the size and location of streaming edge-server clusters. Throughout the section, we discuss important security implications of the revealed Akamai's mechanisms and policies.

Table 1 shows the streams that we use in the experiments in this section. In particular, we select a sample of streams from different media types, customers, and channels: Live streaming that belongs to different customers (NASA, CNN, and ABC), as well as to the same customer (NBA); Video on Demand (VoD) streams that belong to the same customer (Blockbuster) but operate on different channels (a1247, a1042, and 1081), featuring previews for different movies (e33220 - '3:10 to Yuma,' e33210 - 'The Big Bad Swim,' and b10069 - 'Pirates of the Caribbean: At World's End'); finally, we select three different audio streams (FM94.5, FM92.1, and FM106.7). While this is a relatively small number of streams, it is representative in the sense that it can help reveal internal Akamai's mechanisms and policies, as we show below.

To effectively probe the globally-deployed Akamai's streaming network, we use a set of 1,000 publicly accessible recursive DNS servers scattered all around the world. Necessarily, we do not fetch any streams, but rather query Akamai's DNS infrastructure from these 1,000 servers to resolve edge servers for given streams. For example, for the NASA's stream, we query a DNS server for the appropriate CNAME: `a167.l1856944670.c18569.g.lm.akamaistream.net`. The experiment lasts for four days. This enables us to reveal redirection timescales, overlap among different medias, customers, and channels, as well as the size and location of streaming edge server clusters.

3.2.1 Slow Load Balancing

Here, we measure redirection timescales in the Akamai's streaming network, *i.e.*, how frequently do clients get redirected to different streaming edge servers. While similar measurements have been done previously in the context of web (*e.g.*, [23, 27]), it is essential to understand which time scales hold for streaming. Then, we discuss security implications of our findings.

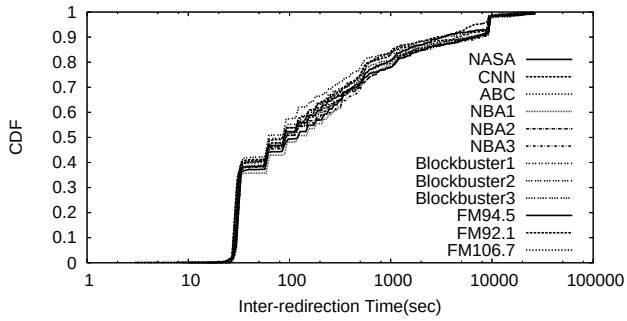
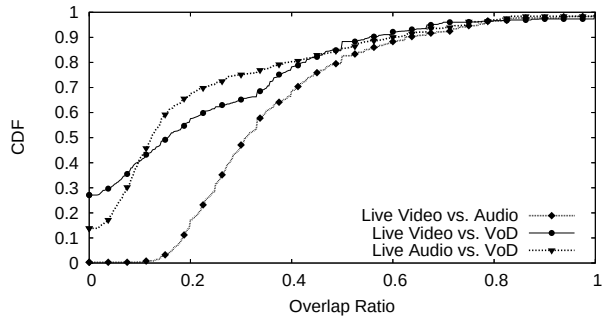


Figure 3: Inter-redirection time of streaming edge servers

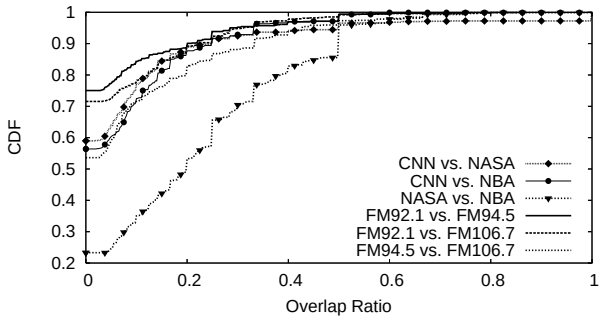
Figure 3 shows median inter-redirection times, obtained using the 1,000 vantage points for the streams shown in Table 1. The results are as follows. First, the observed time scales are in general similar to the ones reported for the web. This is not a surprise because Akamai obviously uses the same measurement system for the web and streaming. Second, our results indicate that 40% of vantage points show median inter-redirection times of 30 seconds, which corresponds to the time-scale at which we query each of the vantage points. Our additional experiments verify that the *minimum* time-scale at which the redirection happens is 20 seconds.

Another interesting result from Figure 3 is that around 10% of vantage points (the upper right corner in the figure) experience almost *no* redirections at all (inter-redirection larger than 10,000 sec). Moreover, 90% of the nodes from this group redirect to the default edge server clusters in the Boston area.[7] This means that Akamai effectively applies the data center approach for approximately 10% of its clients. As a result, clients from the US west coast are routinely directed to this data center on the east coast. We address the downsides of such an approach later in the text.

[7]Subnetworks 72.246.103.0/24 or 72.247.145.0/24.



(a) Overlap among Services



(b) Overlap among Customers

Figure 4: Server overlaps

Implications. The slow load balancing problem is described in Section 2.2.1 above. Our measurement results indicate that the minimum DNS redirection time applied by the streaming CDN is 20 seconds, which is fundamentally inappropriate for live streaming service and opens the doors to DoS attacks. We experimentally evaluate the slow load balancing problem in Section 4.2.1 below.

3.3 No Isolation

Another implication of the results shown in Figure 3 is that they can help to locate busy streaming edge servers. In general, such servers are accessed by the bottom 40% of vantage points from Figure 3 that experience frequent redirections. Degrading service to a busy server is in general easier, because it already operates close to its capacity limit. Hence, even small resources are needed to push them into an overloaded state.

Here, we explore the level of isolation among different services, customers, and channels. A potentially large overlap among these entities raises serious security implications. First, different entities can necessarily impact each other because there is no isolation at either the server or the network level (as we verify later in the paper). Second, the large overlap enables third parties to effectively collect a sufficient number of unique streams belonging to overlapping entities, and then artificially increase traffic volume in these entities. In this way, it is possible to impact service of an arbitrary entity (media, customer, or channel) at overlapping servers, as we demonstrate later in the paper.

To quantify the level of overlap among different entities, we define the overlap metric as follows. Consider two entities, *e.g.*, customers A and B. Next, consider a vantage point and denote by S_A and S_B the subset of streaming edge servers that this vantage point gets redirected to over longer time scales. Likewise, denote by S_{AB} the overlap between the two sets. Finally, we define the overlap ratio as $S_{AB}/min(S_A, S_B)$.

3.3.1 Overlap among Services

In our experiments, we find a total of 1,318 distinct streaming edge servers. Out of this number, 703 servers support video-on-demand, 576 live video, and 688 live audio streaming. These numbers clearly indicate that there is overlap among different medias, as we exemplify below.

Figure 4(a) shows the CDF of the overlap ratio, taken over all 1,000 vantage points, for the following pairs: live audio vs. live video, live audio vs. VoD, and live video vs. VoD. The more to the right of the figure a curve is, the larger the overlap between given pairs. The figure clearly shows that live video and audio overlap more than it is the case for other pairs. We hypothesize that this happens for the following reasons. First, multiplexing large (video) and small (audio) bandwidth-demanding flows is meaningful from the traffic engineering point of view. Second, live audio and video streaming are similar protocolwise with varying playing times, whereas VoD streams are sometimes played within the HTTP protocol and have fixed file sizes. Moreover, VoD content is not streamed using reflectors, as we explain in detail below.

Implications. Figure 4(a) shows that there is significant overlap among different services, even for a relatively small sample of streams that we explored. For example, approximately 20% of vantage points (and consequently clients) have the overlap ratio larger than 0.5 for all three combinations. Implications are obvious: traffic from one media can affect quality of others. Moreover, by artificially increasing traffic load at one media (*e.g.*, VoD), one can affect service to other medias (*e.g.*, live streaming), as we show in Section 4.2.2.

3.3.2 Overlap among Customers

Here, we explore the overlap ratio among different Akamai's customer pairs, as shown in Figure 4(b). While the figure depicts the results for a very small customer sample, it is still insightful. First, the overlap ratio is necessarily smaller than it is the case with medias, simply because the number of customers is larger. Second, the figure shows relatively good isolation among customers, *e.g.*, for majority of the pairs, more than 50% of vantage points see no overlap among given customers. This result is slightly misleading because we show a very small sample. Our additional experiments (not shown) verify that the overlap necessarily increases when a larger number of customers is considered.

Figure 4(b) shows a somewhat larger overlap between NASA's and NBA's streams. We explore this in more depth, and find that the following is the case. NBA's hostname is `a785.l3072828857.c30728.g.lm.akamaistream.net`, while the corresponding Canonical Name (CNAME), used for redirections by Akamai's DNS infrastructure, is `a785.lmg5.akastream.net`. Similarly, NASA's hostname is `a167.l1856944670.c18569.g.lm.akamaistream.net`, and the corresponding CNAME is `a167.lmg5.akastream.net`. While

the two streams operate on different channels (a785 and a167), the common string in the two CNAMEs is the '`lmg5.akastream.net`' part of the domain. Indeed, Akamai uses this approach to group channels based on geographic regions or other properties.[8] Thus, hostnames that have the same 'g' in their CNAMEs have a larger degree of overlap.

Implications. Given the overlap among different customers, the implication is that traffic from one customer can impact other customers. Also, by exploiting the slow load balancing problem, it is possible to intentionally degrade service to arbitrary customers at given streaming servers, by artificially increasing traffic for other customers. Moreover, once the attacker selects a targeted customer, it is easy to locate the overlapping customers by simply querying DNS and finding those that share the same 'g' names.

3.3.3 Overlap among Channels

Finally, we explore the overlap among channels. As explained above, a channel is no more than a collection of streams that is supposed to be transported through the same set reflectors. For each vantage point, we compute the overlap ratio among the following channels: a667, a785, and a1020. The overlap ratio is computed as the overlap among the three channels normalized by the size (the number of edge servers seen by a given vantage point) of each channel. Figure 5 plots the CDF for the overlap ratio over all vantage points. The key point from the figure is that channels are apparently evenly distributed over the edge servers. This is expected as it represents a meaningful traffic engineering decision.

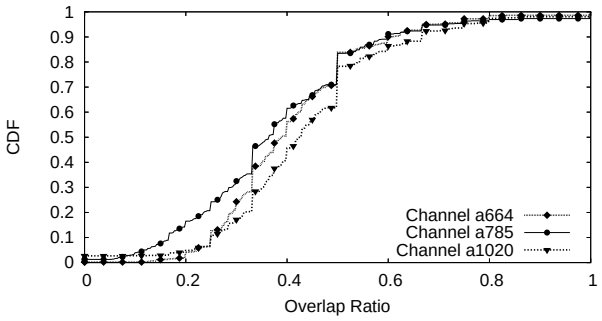


Figure 5: Overlap among channels

Implications. Necessarily, there exists overlap among different channels, *i.e.*, a same streaming server can host a number of channels. The implication is again clear: one can 'jam' a given channel by increasing activity in other overlapping channels at given servers. This further means that it is possible to 'jam' even so-called pay-per-view channels (*e.g.*, NBA), which require cookies to connect to and are hence considered more reliable and safe than other channels. Still, by artificially increasing traffic in co-located 'regular' channels, it is possible to affect the pay-per-view ones.

3.4 Migration and Amplification Attacks

Akamai groups streaming edge servers in clusters and co-locates them at different locations all around the world, thus bringing the content closer to end users. Here, we explore the size and location of such clusters, and then discuss important security implications for each of the issues.

For each channel, we define an edge cluster in a simple way. It is a set of edge servers in the same class C subnet that hosts that channel. For example, assume that edge servers E1 - E5 share the same subnet, such that E1 and E2 host channel A1, and the rest of streaming edge servers, E3-E5, host channel A2. In this case, these five edge servers are divided into the two clusters.

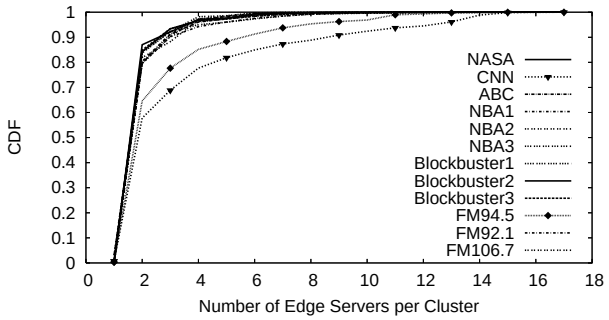


Figure 6: Cluster size

Figure 6 shows the CDF of cluster sizes for explored streams. The figure indicates that the majority of the clusters are small, and typically consist of two edge servers. A second server is typically used as the first choice for backup in case the first one gets overloaded. Still, larger-size clusters exist as well. For example, Figure 6 shows that approximately 10% of CNN's edge clusters have the size of ten and above.

Implications. There are two important security implications for cluster sizes. First small clusters are *potentially* vulnerable to migration misbehaviors. If a given small-size cluster is overloaded, not only that the existing clients (currently fetching streams from the given cluster) will suffer, but newly arriving clients might suffer as well. This is because new clients might get redirected to distant clusters, *e.g.*, to different continents. Contrary to the web case, where such redirections are not problematic, this might not be the case for streaming. Longer inter-continental paths might offer smaller bandwidth and poorer viewing experiences. We explore this in more depth in Section 4.2.3.

At the same time, bigger clusters open the doors to even more serious problems — new *reflector* level vulnerabilities. As indicated above (Section 3.1), an edge server in a given region queries the DNS system by looking up the name that contains the given portset (channel) and the region [22]. Thus, the edge servers that host the same channel and which are co-located in the same region will necessarily fetch new streams from the same reflector. Hence, it is possible to excite a bottleneck at that reflector (and affect service in an entire region) by exploiting the appropriate edge servers in a given cluster as proxies. We explore this issue in more depth in Section 4.3.

4. EVALUATION

Here, we perform Internet experiments to verify the identified vulnerabilities of the Akamai's streaming infrastructure. The key challenge that we face is how to validate our research hypothesis, yet without causing any trouble to Akamai or its clients. Indeed, the key purpose of our effort here

[8]For example, we found that 'g2' mainly serves channels from China.

is to prevent irresponsible or malicious parties to ever become capable of conducting misbehaviors against streaming services at a large scale, as we explain in more detail in Section 5. Thus, here we perform very cautiously engineered Internet experiments in which we carfully and gradually evaluate bottlenecks in the Akamai's network. Whenever we observe bottleneck conditions, we instantly terminate our experiments.

4.1 Experimental Methodology

In order to excite bottlenecks in the Akamai's streaming infrastructure, we collect a set of *unique* streams operating on a single channel at the time of our experiments. Because there is a strong bias in stream popularity, popular and unpopular streams are intentionally multiplexed on the same channels [22]. Hence, by requesting unpopular streams at a given channel, it is possible to generate traffic surges and provoke resource bottlenecks. To collect a set of active streams, we attempt to connect to Akamai's edge streaming servers at a specific channel ID and a specific stream ID to exam the status of the streams. Whenever we successfully connect, we discover a new active stream. In this way, we manage to gather 1,400 unique streams, including live video and audio, for a given channel.

In the experiments, we use seven machines to connect to Akamai's streaming infrastructure.[9] Each of the machines has access bandwith of about 100 Mbs and is assigned 200 unique streams, which it gradually requests as we explain in detail below. One important issue here is that we are able to connect to *any* Akamai's streaming edge server in the world from our experimental machines. In other words, it is possible to override DNS redirections. This dramatically simplifies our experiments here, but at the same time reveals another security 'hole' in Akamai's design.

While it may appear that fixing this single 'hole' would prevent the attacks, this is unfortunately not the case. In particular, these days attackers can rent botnets that can have millions of machines. In such a scenario, even if overriding DNS redirections would not be possible, the attacker can collect sufficient number of machines in a given region, that map to the same edge server. Even if each of the machines would have moderate access bandwith, this is not an issue for the attacker: even if a single attacking machine requests only a single unique stream from the streaming network, that would be sufficient to launch successful attacks. We discuss this and other related issues in more detail in the next section.

Throughout the experiments, we monitor DNS redirections to understand how quickly (or not) does Akamai adjust to induced bottleneck conditions. Also, to understand the impact of the experiments on the environment, and more importantly, to prevent any negative effects for Akamai or its clients, we install several monitoring points which fetch streams from given edge servers during experiments. Again, whenever we observe bottleneck conditions, we instantly abort the experiments. Since Akamai provides streaming service for different protocols (*e.g.*, Realplayer, Quicktime) with a uniform architecture, without loss of generality we use the Microsoft Media Server (MMS) protocol. In particular, we connect to Akamai's streaming servers via MMS by modifying the MiMMS program [6]. This helps us to observe and record a monitored flow's throughput in each second.

[9]To preserve the physical locations of some of the most vulnerable points in the Akamai's streaming network, we refrain from providing detailed network maps in this section.

4.2 Edge-level Experiments

4.2.1 Slow Load Balancing Experiment

Here, we evaluate the slow load balancing problem elaborated in Section 3.2.1 above. We demonstrate that DNS-driven redirections are fundamentally incapable of preserving high quality experiences during system overloads. We setup our experiment as follows: we assign seven machines as probers and one as an observer, which has two roles here. First, it monitors Akamai's DNS redirections; and second, it monitors the throughput of a video stream it is fetching from an edge server.

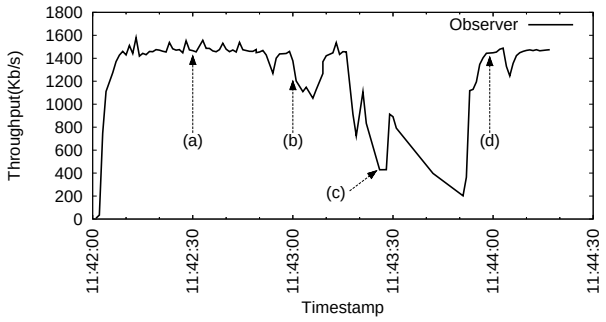


Figure 7: Slow load balancing experiment

Figure 7 shows the throughput of the observed video stream in time. At 11:42:00 the observer connects to an edge server appointed by Akamai's DNS and start streaming a live video. Each second, the observer also monitors DNS redirections by sending queries to Akamai's DNS to obtain the up to date IP addresses associated with the hostname in the stream's URL. After the observering stream reaches the projected throughput (*i.e.*, 1.4 Mb/s), at 11:42:30 (a), the seven probing machines start requesting streams from the same edge server that the observer is connected to. As explained above, they are overriding DNS redirections and request one additional unique stream per second.

At 11:43:00 (b), the edge server (or the access network link) starts becoming overloaded and throughput of the observed stream starts flapping and gradually degrades. At 11:43:28 (c), we observe that the edge server has been removed from DNS entries. At the same moment, the seven probing machines abort their connections to the edge server. At that point, the observing stream's throughput has already been degraded to approximately 300Kb/s. In addition, the throughput does not recover immediately, but reestablishes approximately 30 seconds later (at 11:44:00 (d)).

Thus, we confirm that the DNS-based system is incapable of reacting quickly to overloaded conditions. By the time the DNS entry gets updated and refreshed, the server is already overloaded. In our experiment, the congestion clears after 30 seconds, but this happens because our seven probing machines disconnected. In a real scenario, *e.g.*, either due to a flash crowd or a DoS attack, regular clients would have to disconnect. And such disruptions can motivate clients to search content from competing sites [13]. In section 5, we discuss how to mitigate this problem.

4.2.2 No Isolation Experiment

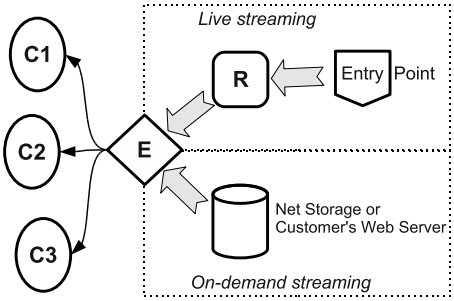


Figure 8: Akamai's streaming models

Here, we evaluate the problem of the lack of isolation among different media in the Akamai's architecture elaborated in Section 3.3 above. In particular, we focus on live video and video-on-demand services.

Figure 8 depicts models of two Akamai's streaming services [11]. As explained in Section 2.1, for live streaming, customers' streams enter Akamai's network via entry points, they are replicated to reflectors, transferred to subscribed edge servers, and finally transmitted to connected clients. For video on demand streaming, streams are transferred to edge servers directly from customers' web servers or Akamai's net storage (cache server).

To verify our hypothesis described above, we perform an Internet experiment, similar to the one from Section 4.2.1, but at a different edge server. In this experiment, the observer downloads a video on demand stream via the MMS protocol, while the seven probing machines again request unique live video streams from the same edge server. We select a video on demand stream that is transferred from an Akamai's cache in an attempt to avoid any potential bottlenecks that are more likely to happen at a customer's web site.

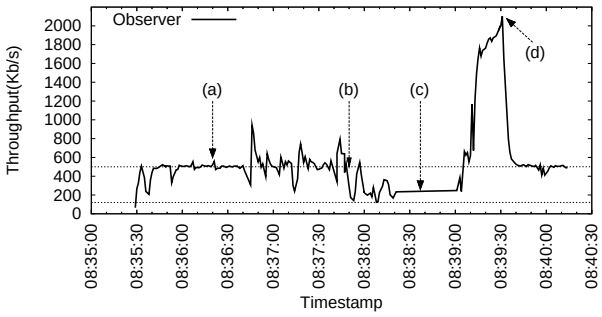


Figure 9: No-isolation experiment

Figure 9 depicts the observed video on demand throughput. After the VoD flow establishes, at time 8:36:20 (a), the seven probing nodes start sending requests to the edge server. At 8:37:50 (b), the throughput starts flapping again and it degrades from 500Kb/s to 123Kb/s when the edge server (or the network) becomes overloaded. We abort the experiment at 08:38:40 (c), immediately after we observe the congestion. Contrary to the video streaming experiment (Figure 7), after the probing machines are terminated, there is a traffic burst as high as 2Mb/s (d). Apparently, no congestion exists on the path between Akamai's cache and the edge server. Therefore, the edge server is able to buffer the stream when the out-bound connection (from the edge server to our monitoring node) is congested. Consequently, after congestion clears, the edge server attempts to 'refill' client's media player buffer by delivering a burst of data to the client.

4.2.3 Migration Experiment

Here, we explore the migration problem elaborated in Section 3.4 above. The question is what happens to newly arriving flows that get redirected once a default edge server becomes overloaded. Given that majority of edge clusters are relatively small in size (Figure 6), once they become overloaded, clients might be redirected to distant edge clusters, or the default IDC at Boston.

It is well known that such long redirections work well for the web for two reasons. First, web objects are relatively small in size, while browsing sessions might be long; hence, a client does not necessarily remain with the backup server for the entire browsing session. Second, downloading web objects is not time critical, *i.e.*, slow images loading do not severely deteriorate client's browsing experience. On the contrary, since live streaming is very sensitive to network latency as well as bandwidth [24], this approach might not be as effective. We explore this issue below.

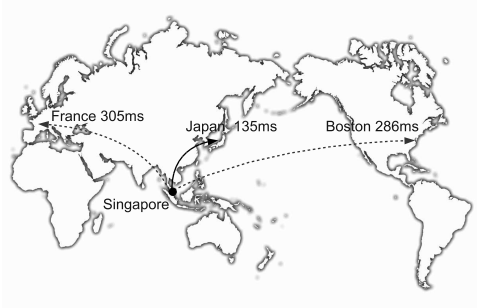


Figure 10: Migration model

Figure 10 shows the redirections we observe from one network node located in Singapore. We measure DNS redirections from that node to one of the hostnames in our collected set of Akamai's streaming ARLs. We find that, for 80% of time, it is redirected to an edge server located in Japan for this specific hostname. In addition, the measured network RTT between the two machines is 135 ms on average. Meanwhile, we also observe that this node gets redirected to edge servers located in Boston and France during congestion epochs. The corresponding RTTs are 286 ms and 305 ms, respectively.

Next, we measure streaming throughputs between the Singapore's node and the three Akamai's edge servers (Japan, Boston, and France), for a particular live stream. For the edge server located in Japan, we enjoy a throughput of 1.4Mb/s, which corresponds to the encoding bit rate for the stream. Whereas, for the edge servers located in Boston and France, we are only able to obtain 800Kb/s and 600Kb/s, respectively. We recorded the three traces and replayed them to colleagues in our institution. All involved in this evaluation confirm that the video quality for the paths to Boston

and France is perceptibly degraded relative to the Japan case.

4.3 Reflector-level Experiments

Here, we explore the potential to excite bottlenecks at a higher level in the Akamai's multicast network - at reflectors, as explained in Section 3.4 above. Because Akamai uses the same DNS mechanisms to balance the load by redirecting edge servers to reflectors, the slow load balancing problem holds for this scenario as well. Moreover, nodes at a higher level in the multicast hierarchy (reflectors) are necessarily carring higher traffic load than the leafs (edge servers). Finally, given that streams from the same region and the same channel share the same reflectors, *i.e.*, [22], implies that it is possible to provoke congestion at the reflector level, as we explain below.

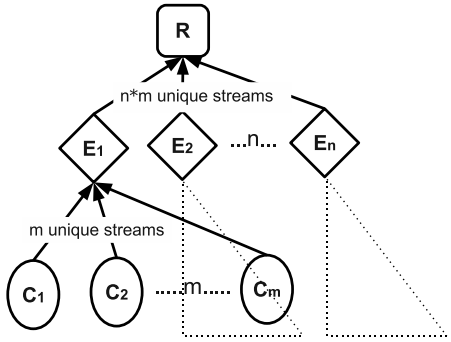


Figure 11: Amplification scenario

Figure 11 depicts this scenario. Assume that reflector `R` is associated with `n` streaming edge servers. Next, assume that each edge server gets requests for `m` unique streams. Inevitably, the number of clients' requests is amplified to `n*m` at the reflector level. Clearly, the larger the number of edges, `n`, in an edge server cluster, the more vulnerable the reflector becomes. In Section 3.4, we have demonstrated that relatively large (*e.g.*, about 10 servers) clusters do exist. Hence, we select one such cluster to verify this hypothesis. In particular, we select 7 Akamai's edge servers in the same class C subnet as our experimental objects. For each of the edge servers, we assign one observer and one probing machine.

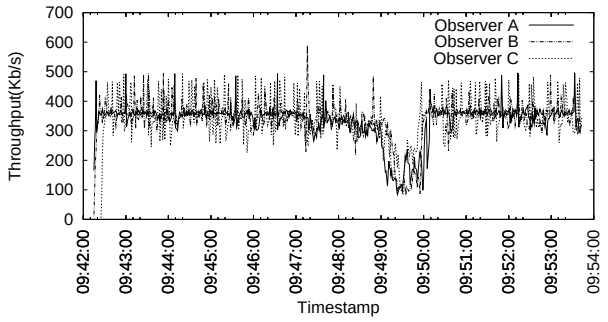


Figure 12: Amplification experiment

Figure 12 depicts recorded throughput from 3 of the 7 observers. In general, we repeat the same procedure as above. The difference is that our seven nodes probe *different* edge servers in the class C network. At 09:48:45, the 3 observers perceive throughput degradation at a very similar pace. In addition, the lowest throughput points (100Kb/s) recorded by the observers are almost identical to each other. Further, after the probing processes are aborted at 09:49:23, the measured throughput recover almost simultaneously.

The results indicate that thinning did not happen at the edge servers for two reasons. First, we perform additional experiments and verify that no local bottlenecks exist between our machines and the edge servers for the given request rates. In particular, we execute our experimental setup to 7 edge servers individually at different times to make sure the observing stream does *not* get thinned. Second, thinning shown in Figure 12 did not happen at the edge because other 4 monitors (not shown in the figure), which fetch their streams from the same edge cluster, did not experience any degradation.

We hypothesize that the following happened. The seven edge servers, while sharing the same channel, are 'backed-up' by two different reflectors. When one of them, which served the three flows shown in the figure, was overloaded, the thinning happened. Another possibility is that the three flows experienced a bottleneck at the network level. Whatever happened, two things are evident. First, it was the probing machines which created thinning, because it cleared when they stopped. Second, this experiment demonstrates that it is possible to excite bottlenecks at a higher level in the multicast tree, by using edge servers as proxies.

4.4 Source-Level Exploits

Here, we explore a similar, yet probably even more severe vulnerability. In the video-on-demand service (Figure 8), no reflectors are used, and streams are fetched directly from customers' web servers or Akamai's net storage (cache server). In both cases, web server or network storage hostnames are embedded in ARLs, and thus publicly available. Such a design enables edge servers to be stateless, because all information needed to redirect traffic is already present in ARLs. Likewise, this approach enables efficient network storage load balancing using the unified DNS redirection mechanism adopted throughout Akamai's network. Still, this approach opens the doors to misbehaviors in a similar way as in the above reflector scenario (Figure 11), yet even more straightforwardly. We discover that the source hosts embedded in the ARLs can be modified by clients, thus disregarding Akamai's original assignments. Consequently, misbehaving clients can overload the source hosts by proxying their requests via edge servers. We discuss methods to resolve such vulnerabilities below.

5. COUNTERMEASURES

Here, we propose a set of countermeasures to help address the above problems. First, we discuss existing solutions to related problems that appear applicable to our problem; yet we argue that such solutions would not solve the problem in a comprehensive way. Second, we present a set of countermeasures that can dramatically raise the bar for the attackers and hence make the system more secure. Finally, we discuss how our findings and countermeasures could be applied in a general way to improve the resilience of large-scale distributed and networked systems to DoS attacks.

5.1 Existing Approaches are Not Comprehensive

5.1.1 Stream Replication

The key problem in the Akamai's streaming architecture stems from the slow redirection timescales, which opens the doors to DoS attacks, *i.e.*, overloading edge servers or reflectors. One way to address this problem is to stream the same packets from multiple sources to a single destination. In particular, (*i*) from several reflectors to a single edge server; and (*ii*) from several edge servers to a single client. Thus, even if a reflector or an edge server becomes overloaded, the receiver could effectively recover the stream because it is receiving packets from multiple sources. According to [22], in certain scenarios, Akamai may feed a single stream to an edge server from multiple reflectors. However, this happens only in areas prone to packet losses.

In general, it is infeasible to replicate each of the streams to each of the reflectors or edge servers due to resource limitations. Moreover, we showed that it is possible to create artificial streaming flash crowds at reflectors or origin servers. Consequently, by the time multiple reflectors get invoked to help the overloaded reflector, the damage has already been done. On the other side, sending multiple copies of a packet from multiple edge servers to a client can utilize precious resources. As we explained above, global-scale streaming services distribute streaming servers to edge regions that typically have limited, often moderate bandwidth, which is shared by the rest of the ISPs' traffic.

5.1.2 Resource-Based Admission Control

Resource-based admission control at the edge servers could help address certain aspects of the problem. The approach is to reject all additional incoming requests whenever a server reaches a resource limit (*e.g.*, a predefined number of streams). The state of the art streaming servers apply this approach to preserve the quality perceived by admitted clients. While it may appear counter productive to reject clients when there exists sufficient spare capacity (at other edge servers), this is not the case. Our research implies that due to slow DNS redirections, rejecting clients is required to protect the quality of the admitted ones.

Unfortunately, resource-based admission control would still not solve many aspects of the problem. In particular, (*i*) when the resource bottleneck does *not* reside at the server side, but rather in the network, server-level admission control is necessarily not effective. Indeed, given that Akamai's streaming edge servers typically do not have any guaranteed bandwith at edge networks, but simply share the same pipes with the rest of the traffic, network-level bottlenecks are quite possible to excite, as we demonstrated above. Likewise, (*ii*) resource-based admission control cannot effectively protect against 'migration attacks' (Section 4.2.3). Despite the fact that the admitted flows might get protected during an attack, the newly arriving clients might get redirected to distant edge servers. Hence, the probability to experience lower streaming quality increases. Finally, (*iii*) the resource-based admission control at edge servers does not solve the potential to excite resource bottlenecks at reflectors, as we have shown in Section 4.3.

5.1.3 Solving Puzzles

Another approach, successful in a web server scenario, is to apply admission control mechanisms capable of accurately distinguishing among DDoS attacks and flash crowds (*e.g.*, [21]). In particular, the approach is as follows. In moments of increased load, the server brings up a graphical puzzle that only humans can solve. In this way, large-scale automatically orchestrated DDoS attacks can be prevented.

There are several reasons why this approach might not be the 'best fit' for DNS-driven streaming services. First, such systems provides a *transparent* service to its customers (*e.g.*, `CNN`). Indeed, one of the benefits of the content hosting approach is that the middle provider, Akamai in this case, stays fully invisible to end users. Thus, bringing up graphical puzzles from the middle servers might be annoying for clients. Second, this approach would still not solve the problem when an attack is targeted towards a reflector, and edge servers are only used as proxies. Because the edge servers are not under attack, and hence may experience only moderate load, no graphical puzzles will be enforced. Finally, imposing graphical puzzles at reflectors is probably even more inappropriate than it is the case with edge servers.

5.2 Raising the Bar for Attackers

5.2.1 Location-aware Admission Control

In an attempt to improve clients' experiences, Akamai's measurement infrastructure uses DNS to redirect clients to approximately the closest edge server in the network sense [3]. At the same time, we demonstrated that it is possible to override these DNS 'recommendations,' and connect to an arbitrary Akamai's streaming edge server from anywhere in the Internet and fetch a stream.

This presents a serious security problem because it enables potential attackers to use machines from all around the world and target a given point in the Akamai's network. Hence, the countermeasure is that edge servers simply reject access to clients that override DNS recommendations. Enforcing such *location-aware admission control* at even a relatively coarse-grained scale would be highly useful. While this approach would still not fully solve the problem (because one can recruit a botnet and exploit a sufficient number of machines in a given network area) this approach alone would dramatically raise the bar for potential attackers.

5.2.2 Reducing System Transparency

The ability to effectively reverse-engineer internal system parameters and mechanisms significantly increases system vulnerability. Hence, reducing system transparency is an important countermeasure that we propose below.

Shielding vincible IP addresses.

One of the reasons that makes DNS-based load balancing approach vulnerable to misbehaviors is exposing edge servers' real IP addresses to the public (Figure 2). This enables third parties to measure the characteristics of the system and unveil the underlying vulnerabilities, as we demonstrated in Section 3. This problem can be mitigated by installing load balancers at edge clusters, *e.g.*, [15, 17]. As widely adopted in IDC-based architectures (*e.g.*, [9]), real servers are associated with a load balancer which has a virtual IP address (Figure 13). The virtual IP address is the only publicly available knowledge to the clients. The clients requesting service are given the virtual IP, they connect to

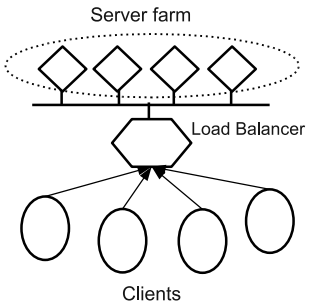


Figure 13: IDC-based load balancing

the load balancer, and they are then re-routed to the real server by internal configuration [16]. Moreover, assume an edge cluster of size n (servers); since the real servers are not directly accessible from the Internet, degrading the system is n times harder than when targeting a single server.

The disadvantage of the hardware-level load balancer approach (Figure 13) relative to the DNS-based approach is the lack of application-level load balancing capabilities, *e.g.*, distinguishing among different customers. This can be solved by assigning multiple virtual IP addresses, *e.g.*, one per customer or channel, to the load balancer and then configure server assignments internally. In this way, the secured information can prevent third parties from targeting a single server and interrupting existing long-live streaming clients connected to the server.

Shielding administrative information. An underlying problem standing behind the ability to target reflectors or customers' origin servers deep in the Akamai's multicast tree is the transparency of the Akamai's streaming infrastructure. Indeed, portset names and even origin server names (in the case of VoD) are embedded in ARLs, and hence available to clients. We hypothesize that this design decision is introduced in order to simplify the system management. By embedding portset or origin server names in ARLs, such information need not be kept at edge servers, which makes them stateless. Consequently, by simply stripping the portset or an origin server name from an ARL, it is easy to submit an appropriate DNS request and get redirected to the right reflector or origin server.

There are two ways to address this problem. The first approach is to keep the state about channels, customers, *etc* at edge servers. Given the number of channels (*e.g.*, around 2,000) and origin servers, such an approach is feasible. The cost of this approach is that whenever a new stream becomes active (or inactive), *e.g.*, the broadcast of a new show, information about its channel membership or about the origin server names must be disseminated to all edge servers. Luckily, efficient solutions to such problems do exist (*e.g.*, [20]).

The second approach is to preserve the integrity of ARLs, so that they do not feed internal information to the public. This can be done by using a credible hash function and/or encoding algorithm at edge servers that can effectively shield sensitive information embedded in ARLs: encode information given to clients and decode in the reverse direction. This approach can be accomplished at a price of several CPU cycles per request, and it is immediately deployable. Given that the request rates are much smaller in the case of streaming than it is the case with web, this approach appears viable.

5.3 Broader Context

Here, we discuss the implications of our findings and countermeasures in a slightly broader context.

Bandwidth-targeted DoS attacks are not dead. DoS attacks are becoming more and more sophisticated, and it is becoming evident that attackers are moving away from bandwith-targeted DoS attacks to more sophisticated, *e.g.*, application-level attacks. In this paper, we demonstrated that bandwith-targeted DoS attacks against streaming services are highly feasible and easy to conduct. Moreover, we argue that the increased level of streaming in the Internet has the potential to reverse the above trend. On the one hand, streaming flows consume a lot of bandwith. On the other hand, streaming clients are easily discouraged from accessing such services when the performance is poor. Consequently, attempts to address such problems at a global Internet scale (*e.g.*, [18]) are certainly valuable.

Tensions between transparency and security. One of the ways to increase the resiliency of DNS-based systems to DoS attacks is to reduce system transparency (Section 5.2.2). While this approach is well-suited for proprietary systems such as Akamai, generalizing such an approach to the Internet as a whole might bring novel problems. For example, many ISPs reduce their transparency for security reasons, *e.g.*, routers in edge networks already disable ICMP TTL Time Exceeded replies using firewalls, which are needed for traceroute to discover topologies. Moreover, in light of new attacks against the Internet infrastructure, proposals to disallow ICMP timestamp replies are becoming more prominent [29]. At the same time, the Internet transparency (*e.g.*, [25]) is critically needed for the performance of large-scale distributed systems. Unfortunately, it is becoming obvious that the two approaches are at fundamental odds.

6. CONCLUSIONS

In this paper, we explored the resilience of Akamai's streaming architecture to intentional service degradation attempts. We demonstrated that the current system design is incapable of preserving high-quality experiences to streaming clients in such scenarios. In particular, we showed that (*i*) the DNS-based redirection subsystem, which has proven very successful in the case of web, is fundamentally inappropriate for live streaming. Moreover, (*ii*) no isolation at either the network or the server level among customers, channels, and services, (*iii*) a strong bias in the stream popularity, and (*iv*) a highly transparent and security-oblivious system design make Akamai's streaming network extremely vulnerable. We demonstrated that it is feasible to impact arbitrary customers' streams in arbitrary network regions. This is possible to achieve not only by exciting bottlenecks at edge servers, but by effectively exploiting edge servers as proxies to provoke bottlenecks at reflectors and even origin servers.

We provided a set of countermeasures to help avoid such vulnerabilities. We showed that a less transparent system design capable of hiding important internal information (*e.g.*, about edge servers, reflectors, origin servers, channel assignment, *etc.*) from the public can dramatically raise the system's resilience to misbehaviors. Still, it should be clear that minimal users' tolerance for low quality experiences ac-

companied by the lack of isolation and QoS mechanisms at servers and at the global Internet makes high-quality streaming inherently vulnerable to jamming misbehaviors.

7. REFERENCES

[1] http://www.pqm.net/webcast/marcon/cgiagm01022005.html.

[2] http://eventcompressiongroup.com/_fav/encoder_links.htm.

[3] Akamai. http://www.akamai.com.

[4] End System Multicast. http://esm.cs.cmu.edu/.

[5] Joost. http://www.joost.com/.

[6] MiMMS. http://savannah.nongnu.org/projects/mimms.

[7] Rinera Networks. http://www.rinera.com/.

[8] URL Snooper. http://www.donationcoder.com/Software/Mouser/urlsnooper/index.html.

[9] YouTube. http://www.youtube.com/.

[10] Zattoo. http://zattoo.com/.

[11] Akamai Technologies. Akamai Media Delivery. http://www.akamai.com/html/solutions/media_delivery.html.

[12] Akamai Technologies. How "Akamaization" Works, 2000. http://www.akamai.com/html/about/press/releases/2000/press_061300.html.

[13] Akamai Technologies. Akamai study uncovers critical link between video quality and audience retention, revenue opportunities, 2007. http://www.akamai.com/html/about/press/releases/2007/press_080707.html.

[14] K. Andreev, B. Maggs, A. Meyerson, and R. Sitaraman. Designing overlay multicast networks for streaming. In *Proceedings of ACM SPAA '03*, San Diego, CA, June 2003.

[15] B. Augustin, T. Friedman, and R. Teixeira. Measuring load-balanced paths in the internet. In *Proceedings of ACM IMC '07*, San Diego, CA, Oct. 2007.

[16] Cisco Systems, Inc. Configuring server load balancing. http://www.cisco.com/univercd/cc/td/doc/product/software/ios122/122cgcr/fipr_c/ipcprt1/1cfsflb.htm.

[17] Cisco Systems, Inc. How does load balancing work? http://www.cisco.com/warp/public/105/46.html.

[18] C. Dixon, T. Anderson, and A. Krishnamurthy. Phalanx: Withstanding multimillion-node botnets. In *Proceedings of ACM NSDI'08*, San Francisco, CA, Apr. 2008.

[19] Ellacoya Networks. Web traffic overtakes peer-to-peer (p2p) as largest percentage of bandwidth on the network, June 2007. http://www.ellacoya.com/news/pdf/2007/NXTcommEllacoyaMediaAlert.pdf.

[20] C. Gkantsidis, T. Karagiannis, and M. Vojnovic. Planet scale software updates. In *Proceedings of ACM SIGCOMM '06*, Pisa, Italy, Sept. 2006.

[21] S. Kandula, D. Katabi, M. Jacob, and A. Burger. Botz-4-Sale: Surviving DDoS attacks that mimic flash crowds. In *Proceedings of ACM NSDI '05*, Boston, MA, May 2005.

[22] L. Kontothanassis, R. Sitaraman, J. Wein, D. Hong, R. Kleinberg, B. Mancuso, D. Shaw, and D. Stodolsky. A Transport Layer for Live Streaming in a Content Delivery Network. *Proceedings of the IEEE*, 92(9):1408–1419, 2004.

[23] B. Krishnamurthy, C. Wills, and Y. Zhang. On the use and performance of content distribution networks. In *Proceedings of ACM SIGCOMM IMW '01*, San Francisco, CA, Nov. 2001.

[24] J. Liu, S. G. Rao, B. Li, and H. Zhang. Opportunities and Challenges of Peer-to-Peer Internet Video Broadcast. In *Proceedings of the IEEE Special Issue on Recent Advances in Distributed Multimedia Communications*, 2007.

[25] H. Madhyastha, T. Isdal, M. Piatek, C. Dixon, T. Anderson, A. Krishnamurthy, and A. Venkataramani. iPlane: an information plane for distributed services. In *Proceedings of USENIX OSDI'06*, Seattle, WA, Nov. 2006.

[26] K. Sripanidkulchai, A. Ganjam, B. Maggs, and H. Zhang. The feasibility of supporting large-scale live streaming applications with dynamic application end-points. In *Proceedings of ACM SIGCOMM '04*, Portland, Oregon, 2004.

[27] A.-J. Su, D. Choffnes, A. Kuzmanovic, and F. Bustamante. Drafting behind akamai (travelocity-based detouring). In *Proceedings of ACM SIGCOMM '06*, Pisa, Italy, Sept. 2006.

[28] A. Walters, D. Zage, and C. Nita-Rotaru. Mitigating attacks against measurement-based adaptation mechanisms in unstructured multicast overlay networks. In *Proceedings of IEEE ICNP '06*, Santa Barbara, CA, Nov. 2006.

[29] Y. Zhang, Z. Mao, and J. Wang. Low-rate tcp-targeted dos attack disrupts internet routing. In *Proceedings of ISOC NDSS '07*, San Diego, CA, Feb. 2007.