SCTE Cable-Tec Expo® 2010

Open Content Delivery Networks for Managed Video Delivery to Multiple Screens

Santosh Krishnan, Ph.D.
Vice President, Product Strategy
Verivue, Inc.
skrishnan@verivue.com
(978) 303-8107

Weidong Mao, Ph.D.
Senior Fellow
Office of the CTO
Comcast Cable
weidong_mao@cable.comcast.com
(215) 286-7306

ABSTRACT

The primary goal of a content delivery network (CDN) is to scale content libraries and bandwidth by utilizing distributed caching, as opposed to replicating entire libraries at each serving location in the network. Furthermore, in order to support multiple screens including existing and next-generation set-top-box-connected televisions, personal computers and other IP-enabled devices, the edge caches of a CDN must support multiple delivery mechanisms such as User Datagram Protocol (UDP) streaming to existing QAM-based set-top boxes and Hypertext Transfer Protocol (HTTP) streaming and progressive download to IP devices. We propose an open CDN architecture that interconnects such heterogeneous edge caches to origin servers. The architecture is based upon open interfaces between tiers of the CDN hierarchy and between various functional planes of the CDN. Interfaces are layered on top of standard HTTP with Representational State Transfer (REST), with certain extensions and message formats to account for multi-screen video-related functionality. We describe the motivations behind our choice and show how the proposed architecture may be used by operators to deploy a fully standards-based converged CDN.

1. INTRODUCTION

One of the primary goals of a content delivery network (CDN) is to scale content libraries by utilizing distributed caching, as opposed to replicating entire libraries at each serving location. Distributed caches offload capacity from core networks, simultaneously allowing to optimize network bandwidth, reduce latency, and to scale end-user bandwidth.

Video-on-demand (VOD) systems sometimes employ a form of distributed caching using the concept of *asset propagation*. In such networks, illustrated in Fig. 1(a), a central element monitors content popularity and pushes selected content to selected servers, with the goal of ensuring that popular content is pushed to edge VOD servers and unpopular content remains at centralized VOD servers. This represented a natural progression from the classical client-server model of previous-generation VOD networks. Such an approach might suffer from a central bottleneck, namely, the asset propagation system, which needs to monitor and process the access patterns in the entire network. In contrast, edge caching CDNs are based on a “pull” model, illustrated in Fig. 1(b), wherein all content is placed in origin servers, and caching decisions are made in a distributed fashion in each node. Content is requested by a downstream node to serve cache misses, and/or to fill its own cache, based upon the locally resident caching algorithm. This approach promises to yield highly scalable CDNs for managed video delivery.

In short, edge caching CDNs are based on the following broad principles:

- A content management system that prepares content and ingests it into *origin servers*. The origin servers provide the authoritative location for each asset served by the CDN.

- Distributed caching servers that pull content on a cache miss, directly from origin servers or from intermediate caches en route to the origin server. The distributed servers implement a caching algorithm based on local access patterns to determine which content belongs in the cache, with the primary goal of maximizing cache hit ratio.
- Portals designed with carefully crafted URI schemes (or resource representations for assets) that aid in routing requests to the correct servers in each level of the CDN hierarchy. The request routing is typically based on HTTP redirection [1] or DNS resolution [5].
- Policy-based content delivery at caching nodes, allowing for flexible CDN management. Policies typically allow caches to implement rules such as license windows, token authentication, geo-blocking, and various service level agreements.

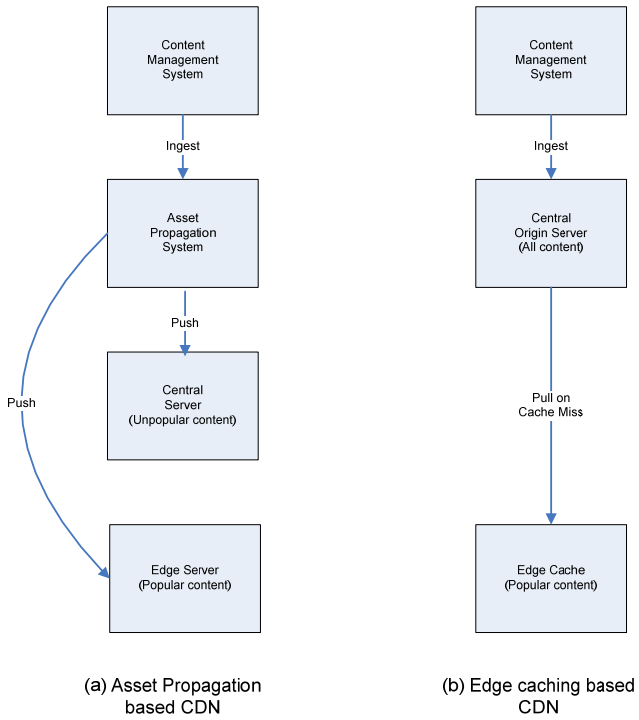

Figure 1: Content delivery models

We propose an HTTP-based (RFC 2616) [1] open CDN for managed multi-screen video delivery, based on best-of-breed IP principles that have been proven to scale, in support of expanding content libraries and increasing subscriber bandwidth. The CDN is designed to conform to principles of HTTP Representational State Transfer (REST) [2] in order to ensure cacheability across the CDN data and control planes. Most importantly, every server in the CDN is designed to be stateless, i.e., the retrieved resource (media asset or control plane artifact) does not depend on prior client requests. In order to support multiple screens, including existing set-top boxes with UDP streaming, and IP-enabled devices, including browser-enabled PCs and connected televisions with HTTP-based content delivery, the edge caches of the proposed CDN support multiple delivery mechanisms (as illustrated in Fig. 2).

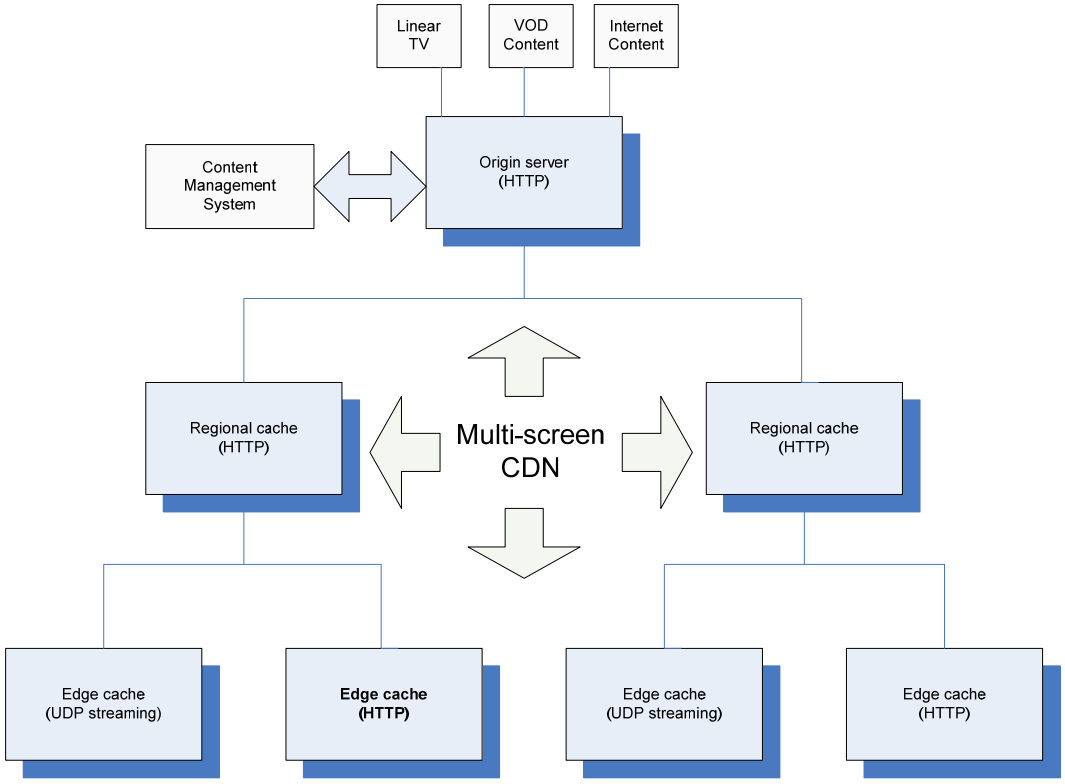

Figure 2: Target multi-screen managed content delivery network

The proposed multi-screen managed CDN may be broken down into three planes:

- Data plane: In the data plane, edge caches support UDP streaming as well as HTTP progressive download for video delivery to end users. The rest of the CDN is based on open HTTP-based content delivery, in support of both UDP streaming and HTTP delivery edge caches. Every data plane HTTP request is designed to be cacheable.
- Control plane—Request routing: The request routing stratum provides server location service in each tier of the CDN data plane hierarchy. We outline several open options based on HTTP and DNS, and their relationship with the chosen URI schemes. The results of request routing are designed to be cacheable, and amenable to persistent data plane connections, so that every data plane request does not trigger a routing request.
- Policy and management plane: The policy and management plane is responsible for enforcing caching and content delivery policies. Policy requests from the data plane to the policy plane are primarily based on HTTP and designed to be cacheable. In addition, this stratum is responsible for CDN element management and statistics collection.

Note that the proposed CDN should be viewed as an application of well-known IP principles and open standards towards building an infrastructure for managed video delivery. We focus on guiding principles for building such an infrastructure, and not on detailed interface specifications, which we leave for standards bodies in the industry.

Section 2 provides a brief background on HTTP and REST, which forms the basis of our proposed CDN. Section 3 introduces the CDN reference architecture, including the three planes enumerated above, along with the requirements associated with each. Use cases and applications of the CDN reference architecture from an operator's perspective are presented in Section 4. Sections 5-7 outline open proposals and recommendations for each of the CDN planes. Finally, Section 8 summarizes the proposed open CDN architecture.

2. BACKGROUND: HTTP AND REST

This proposal embraces the principles of REST for defining interactions between various components of a CDN. Here, we briefly overview those principles. In essence, REST [2] denotes the *architectural style* of the web. Given that all systems have tradeoffs, an architectural style represents a set of constraints applied to induce desired architectural properties while accepting certain tradeoffs. REST's constraints are intended to induce properties necessary for large-scale distributed hypermedia systems. Some of the key properties of so called *RESTful* systems are:

- Simplicity, performance and scalability
- Loose coupling between client and server
- Stateless self-describing (idempotent) requests
- Visibility allowing monitoring, mediation and caching
- Servers driving client state via hypermedia representations
- Negotiation of different types of content between client and server
- Ability of client and server to evolve independently

An important abstraction in REST is the *resource*, which essentially is anything that can be named, e.g., CDN files, control plane artifacts. Under REST, all resources are identified using uniform resource identifiers (URI), which must be created on the server side. To clients, URIs are essentially just opaque identifiers. Clients and servers exchange *representations of resource state* via a uniform interface. For example, when an HTTP client performs a GET request [1] for a particular resource, the server typically returns the HTML representation of the requested resource. The returned representation indicates the state of the resource at the time of the request. This approach is in contrast to prior distributed object systems or web-service remote procedure call (RPC) systems that provide a specialized interface for each different object type, and keep object state and data encapsulated within server-side entities.

A critically important constraint in REST is the notion of the server driving client state via hypermedia. RESTful services typically publish a single top-level URI that returns a hypermedia representation containing links to associated resources, which clients can traverse to decide which links it can use and the actions to be taken. To this end, RESTful resources tend to make use of standard hypermedia types such as (X)HTML [6] and Atom [7], which are well known and thus likely to be acceptable to independently developed clients, rather than inventing proprietary media types. Proprietary or non-standard media types limit the clients that can make use of a resource, and

media types that do not support hyperlinks force clients and servers to somehow exchange information out of band in order to execute, thereby increasing coupling.

Other important REST constraints are:

- Statelessness: Clients, not servers, keep session state, and all requests are self-contained, thus inducing the properties of visibility (no need to look outside any given request to understand it), reliability (easier recovery from partial failures), and scalability (server need not manage individual client state for numerous clients).
- Caching: Responses are explicitly marked as to if and how they may be cached. Caching helps induce the properties of performance and scalability by reducing and eliminating certain interactions, thus improving latency and reducing network traffic.
- System layering: Components see only their interactions with their immediate neighbors, thus simplifying the overall system and allowing intermediaries such as load balancers, proxies, and caches to be introduced into a system without breaking interactions, which helps induce scalability and performance.

HTTP is the most widely known RESTful protocol. Its uniform interface consists of its verbs, mainly GET, PUT, POST, and DELETE. HTTP requests and responses are generally self-contained and self-descriptive, and resources use specific HTTP headers such as *Cache-Control, Expires, Etag*, and *Last-Modified* to control if and how their responses may be cached. The *Accept* header allows clients to negotiate representations. Intermediaries such as proxies, reverse proxies, caches, and load balancers prove the viability of REST's constraints and also contribute significantly to the utility and scale of the web. Developing RESTful HTTP services and resources generally requires paying careful attention to the hypermedia constraint, the stateless constraint, and how best to use URIs to identify resources and indicate relationships between them.

3. REQUIREMENTS AND REFERENCE ARCHITECTURE

In addition to a general preference for RESTful principles on all planes of a CDN—so as to yield a scalable distributed architecture that most leverages emerging IP technologies—we establish the following high-level requirements on multi-screen CDN design.

Data Plane

- Client-controlled transfer: Within the CDN infrastructure, a *client* is any caching node requesting content from a higher-tier node so as to serve a cache miss and/or to fill its own cache. In order to maintain a stateless server and have the client keep “session” state (for the cache miss/fill session), the transfer of content over the CDN must be coordinated by the client. The client must determine which content to request, when to make the request (e.g., proxy caching, background caching), how to make the request (e.g., assets in entirety or in segments), and whether or not to cache the requested content (caching algorithm).
- Clustering: The CDN infrastructure must allow for local clustering of multiple physical servers so as to realize a single logical node (edge, regional, origin). While this may be viewed as a form of local request routing, it should not be confused with the problem of “network” request routing that steers a request to a logical node. In fact, separating clustering

and network request routing is a manifestation of the "system layering" constraint of REST, though both might use similar techniques.

- Broad application support for multi-screen: The CDN must support multiple delivery technologies to end users. Specifically, it must support the multiple formats and multiple delivery protocols required to support the various devices.
- Multiple types of content sources: The CDN must accommodate multiple types of content sources, e.g., operator-provided origin servers for existing VOD and broadband content, and origins provided by third-party content aggregator sites. This requirement essentially mandates the ability to interface with standard origin servers, including standard-based restricted (i.e., content unaware) ones such as network-attached storage systems.
- User control functions—trick mode and random access: The CDN must support network-based VOD functions such as trick mode and time-based random access. The CDN itself must not restrict the richness of such functions, e.g., the number of trick speeds. This requirement may be relaxed in the future as video networks converge towards all-IP offerings, where such functions are subsumed by the client.

Control Plane—Request Routing

- Cacheability: In order to account for fragmented content delivery (e.g., in the case of HTTP adaptive streaming), request routing should not be invoked by the data plane nodes on every content request. This, in essence, restricts the options for request routing to those whose results can be "cached" by the data plane nodes. Here, caching does not strictly refer to HTTP caching—instead it refers to re-use of request routing state at the client.
- Persistent TCP connections: The request routing scheme should allow for persistent connections in the data plane. This constraint eliminates certain schemes, which rely on layer 7 techniques (e.g., certain kinds of simple HTTP redirection) that lead to frequent socket re-establishment.
- Multiple URI options in the content management system: The CDN must allow for multiple types of URI formats crafted by the content management systems. In some cases, content management systems may fully *qualify* the end-user content URI, making those URIs self-sufficient for request routing purposes. In other cases, content management systems may use different URIs (e.g., to hide origin locations from the end user) for different tiers in the network (e.g., end-user URIs and origin URIs). The request routing function must allow for URI conversions within the CDN.
- Multiple independent phases: CDN request routing must allow the option for multiple phases, each of which must be independent from the other. Here, we simultaneously impose the REST constraints of "system layering" and "independent evolution." Request routing must then be anchored at the data plane node (the client of the request routing stratum), which becomes the only stateful node in the architecture.

Policy and Management Plane

- Policy-based data plane: The CDN must provide for policy-based content delivery. This essentially implies that the data plane nodes must (directly, wherever possible) have the ability to invoke policies on each content request. Here, we do not restrict the types of policies and where they are executed (e.g., within the data plane node, via a policy interface to a policy application server)—instead, we must allow for various kinds of operator provisioned policies in the architecture.
- Cacheability: Responses to policy requests made by the data plane should be cacheable. This restricts the kinds of messages and URIs that may be used to carry out the policy decisions. For example, in fragmented content delivery, the policy decision should only be carried out on the first fragment request.

Towards these requirements, we propose the reference architecture illustrated in Fig. 3.

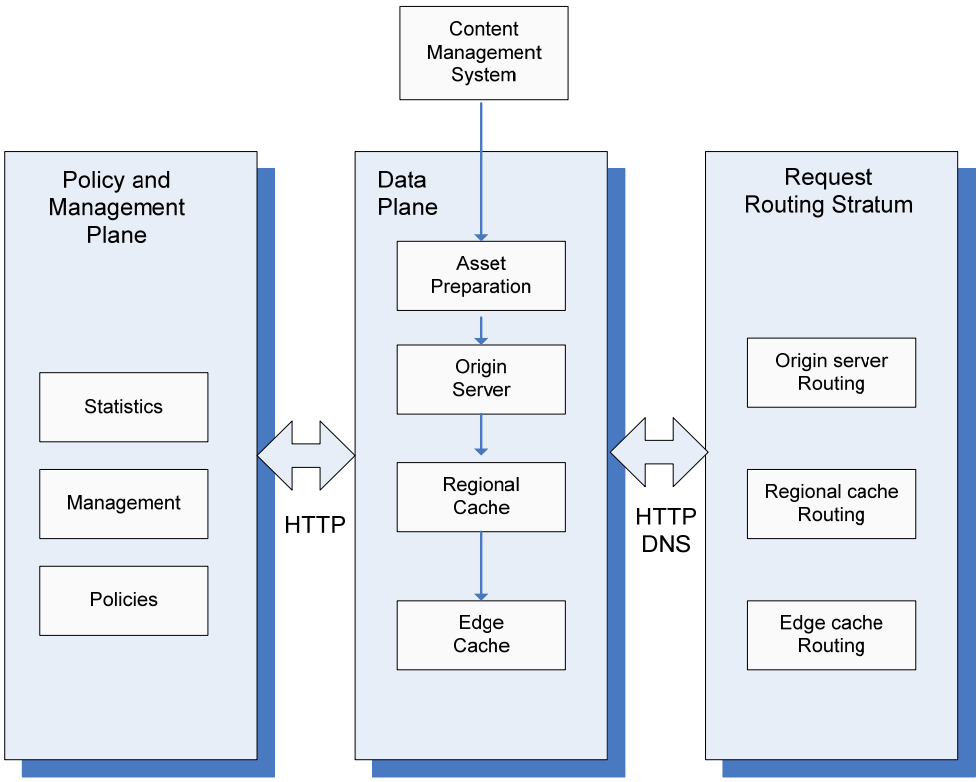

Figure 3: Reference architecture for an open Content Delivery Network

The above architecture is essentially an HTTP-based CDN for multi-screen content delivery, derived from best-of-breed IP concepts, with a liberal application of HTTP and REST principles. The screen-specific variations are relegated to the two ends of the CDN, namely, at the content management system and at the end-user client. In support of multiple types of end-user clients, the edge cache implements the necessary transport protocols such as UDP and HTTP. The rest of the CDN is all based on standard HTTP for content transfer.

The content management system (CMS) is responsible for making assets available, transforming them into the necessary formats to account for multiple types of clients, and preparing URIs and making them available. Strictly speaking, the CMS is not part of the CDN. However, it may sometimes provide information to the request routing stratum (via an interface not shown in the figure) depending upon the chosen URI schemes. The CMS may be augmented by an *asset preparation system* (unless the function is subsumed within the CMS itself), which is responsible for generating CDN specific metadata, e.g., index files to serve time-based content requests to the edge cache, and to assist in trick mode within the CDN.

The request routing stratum shows several request routing functional blocks, one for each tier of the data plane. For example, the *origin server routing* block provides origin server location to a requesting regional cache. It monitors the health of the available origin servers using a standard status interface (not explicitly shown in the figure). Depending upon the chosen URI implementation, some of those blocks may not be necessary. Also, in practice, several blocks may be consolidated into a single physical system if necessary.

4. CDN: EXAMPLE APPLICATIONS

There are two major applications of the proposed CDN reference architecture for managed video delivery: large scale VOD libraries [3] for existing set-top boxes using traditional UDP-based edge streaming, and IP video delivery [4] of managed content to multiple IP devices using HTTP adaptive streaming. Both applications can leverage the same core CDN infrastructure, with specific adaptations—of content formats and content delivery schemes—for various CPE devices.

In the first application, namely, VOD CDN, content is typically encoded using MPEG-2 or MPEG-4 AVC compression, encapsulated into MPEG-2 single program transport streams. They may be ingested into one or more origin servers by the CMS, through asset preparation. The CMS manages the content life cycle including licensing windows. The set of origin servers is also called the VOD library, or persistent store, in this application. Common VOD index files, for supporting pre-generated trick files or dynamic on-the-fly trick mode, may be generated by an asset preparation system during ingest to the origin servers.

- **Data Plane**: The data plane of the VOD CDN application uses regional caches in the regional data center and VOD streaming servers as edge caches, performing UDP streaming, in the local head-end. Upon request from the digital set-top boxes, the selected content files may be pulled using HTTP, by the edge cache from a regional cache, and by the regional cache from the respective origin server, if there are cache misses at the various stages of the CDN. The net effect is that only the most popular VOD content is cached at edge caches and/or regional caches, while long tail content remains at origin servers. This allows the VOD application to provide an almost infinite amount of popular and long-tail content, such as movies, TV shows, and other on-demand content.
- **Control Plane:** The control plane of the VOD CDN application uses HTTP- or DNS-based request routing from edge caches to regional caches, and from regional caches to origin servers. The request routing associated with the selection of an edge cache (or VOD streaming server) for UDP streaming of an MPEG-2 single program transport stream is performed by the VOD session and resource management. In this CDN application, the VOD

session manager is responsible for establishing a session upon receiving a request from a set-top box, following which it interfaces with various *resource managers* for resource allocation. One example of a resource manager is the *edge resource manager* responsible for access network bandwidth allocation.

- **Policy and Management Plane:** The policy and management plane of the VOD CDN application includes functions such as content status and verification, data warehouse reporting, content cache policy control, and network management.

In the second application, namely, managed IP video CDN, content is typically encoded using multiple bit rate profiles with MPEG-4 AVC compression, and encapsulated into fragmented MP4 containers. Such fragmentation allows the client to switch bit rates, on a fragment by fragment basis, in order to adapt to server and network conditions. Content may be encrypted during ingest into the origin server. They may be ingested into one or more origin servers by the CMS through asset preparation. The CMS manages the content life cycle including licensing windows. In addition, it is responsible for transcoding content into multiple formats, suitable for the various targeted IP devices.

- **Data Plane:** The data plane of the managed IP video CDN application uses regional caches in the regional data center, and edge caches in the regional data center or local head-end. Unlike the first VOD CDN application, the data plane of the managed IP video CDN application can leverage HTTP caching and adaptive streaming all the way to the end-user IP devices. Upon request from the IP devices, the selected content files (fragments) may be pulled using HTTP by the edge cache from a regional cache, and by the regional cache from an origin server, if there are cache misses at the various tiers of the CDN. HTTP adaptive streaming allows the various cache nodes to be unaware of content formats, with all content awareness subsumed at the origin. At the same time, it may be possible to have further optimizations at the cache nodes to improve caching efficiency and latency, based upon the various HTTP adaptive streaming formats.
- **Control Plane:** The control plane of the managed IP video CDN application uses HTTP- or DNS-based request routing from the client to an edge cache, from the edge cache to regional cache, and from the regional cache to origin server. Unlike the first VOD CDN application, the control plane for the managed IP video CDN application can leverage similar request routing techniques at every stage of the content delivery network.
- **Policy and Management Plane:** The policy and management plane of the managed IP video CDN application is somewhat similar to that of the VOD CDN application. It may include functions such as content status and verification, data warehouse reporting, content cache policy control, and network management.

5. DATA PLANE

Content Origination

There are two models for ingest into the origin server, as shown in Fig. 4. In the *direct write* model (Fig. 4(a)), media content and metadata are directly written into the origin HTTP server, using HTTP POST or FTP PUT methods, to the URI(s) crafted by the CMS. Typically, an asset preparation server is notified by the CMS about new content availability, using an ingest provisioning message

(e.g., a VOD asset ingest interface). The asset preparation server then writes the media, possibly modified, and the generated metadata into the origin. For example, in the VOD CDN application, the asset preparation server may fetch content from a staging folder (content source) using FTP and generate VOD index files, specifically to enable network-based trick mode in the CDN. In the managed IP video CDN application, the asset preparation server may perform transcoding and fragmenting, and write the resulting fragments and metadata into the origin. In practice, the asset preparation function may be performed by a pipeline of multiple physical servers.

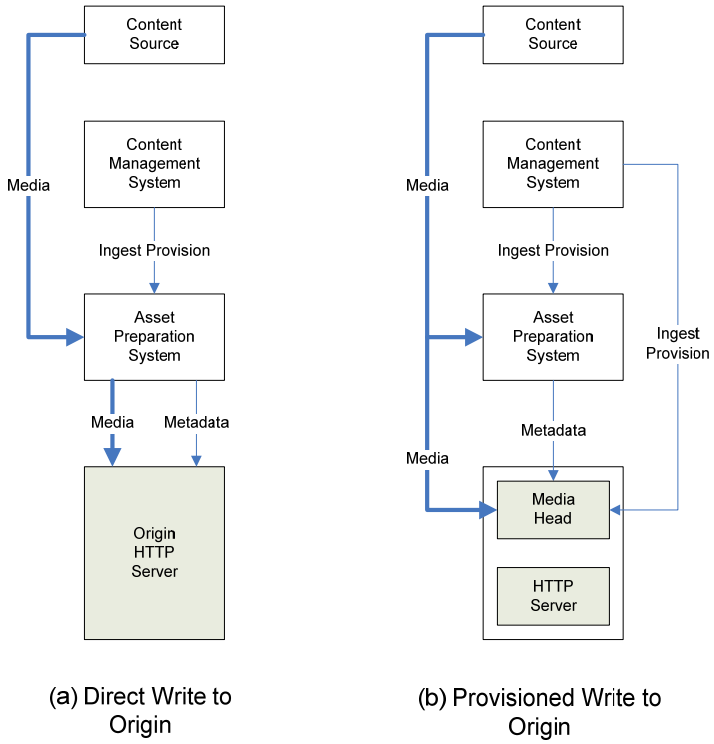

Figure 4: Models for ingest into the origin server

In the *provisioned write* model (Fig. 4(b)), there is no element in the content management workflow (including any asset preparation servers) that writes directly into the origin server. Instead, the CMS notifies a *media head* function about new content and metadata availability. The media head, in turn, fetches the content and metadata from respective sources (e.g., staging folder and the asset preparation server, for media and metadata, respectively) using HTTP GET or FTP GET methods, and writes it into the origin HTTP server. It should be apparent that in the direct write model, the asset preparation system essentially subsumes the media head function.

Typically, there is no need to ingest content directly into other CDN nodes. As an optimization in some cases, caches in other CDN nodes may be "warmed" under the direction of the CMS. Such cache warming may be achieved by sending an ingest provision message to downstream caches, by forcing a cache-fill by sending a dummy request to the cache, or by pushing content and metadata via multicast techniques.

CDN Data Plane Nodes

Edge Caches

The edge caches of the CDN implement multiple delivery protocols in support of multi-screen video delivery, e.g., UDP streaming in the VOD CDN application and HTTP delivery in the managed IP video CDN application. In addition, it implements an HTTP-based open *content delivery interface* to fetch content on a cache miss and/or to fill its own cache. In essence, it functions as a streaming gateway in the VOD CDN application and as an *opaque* (i.e., media un-aware) proxy cache for IP Video. The edge cache may use diverse *caching modes* including any combination of

- Segment-based transfer (using ranges) or file-based transfer;
- Background cache fill, i.e., not while serving a cache miss, or piggy-backed cache fill; and
- Best-effort versus rate-based transfer (e.g., at some nominal rate)

It uses judiciously chosen *caching algorithms* (e.g., direct-replacement cache-fill with least-recently-used (LRU) eviction, threshold-based cache-fill) in order to maximize the cache-hit ratio. While the content delivery interface must be standardized in order to ensure a multi-vendor CDN, we submit that the caching mode and algorithm should be client-controlled in accordance with principles of REST. The content delivery interface must allow the client (in this case, the edge cache) to execute any of the above modes as desired.

In practice, for the VOD CDN application, existing streaming servers in the head-ends may be repurposed into edge caches by adding support for the open content delivery interface—to fetch content in real-time as opposed to being based on offline VOD ingest provisioning.

Origin Servers

We envision two types of origin servers in an open CDN, both based on HTTP. A *basic* origin server is an HTTP v1.1 compliant web server, essentially providing file-based access to downstream nodes. A basic origin server may suffice if a downstream node (e.g., an edge cache or the end-user client itself) implements media-aware requests. For example, in the VOD CDN application, an edge cache may use HTTP to serve its cache misses, assisted by VOD index files to support trick mode and time-based random access operations. As another example, in some HTTP streaming applications (e.g., [8]), the end-user client may employ playlist files and use HTTP to request segments from a basic origin server.

In contrast, an *augmented* origin server is an HTTP v1.1 compliant application server, which maps the requested URI, using server-based index or manifest files, or other logic, into a resource to be delivered. In addition, an augmented origin server may provide support for additional features such as bit-rate profiles for content delivery. In fact, in several HTTP adaptive streaming technologies for the managed IP video CDN application, an augmented origin is designed to be the only media-aware network node, with opaque caching all the way from the client. As another example, in the VOD CDN application, an augmented origin server may be used, providing time-based access to normal-mode and trick-mode MPEG-2 transport streams.

Regional Caches

In most cases, as shown in Fig. 5(a), the regional caches in our proposed CDN are expected to be *opaque* HTTP v1.1 caches—irrespective of whether the origin servers are basic or augmented. In

order to ensure such cacheability, augmented origin servers must support RESTful URIs. The latter is indeed the case with all HTTP adaptive streaming technologies.

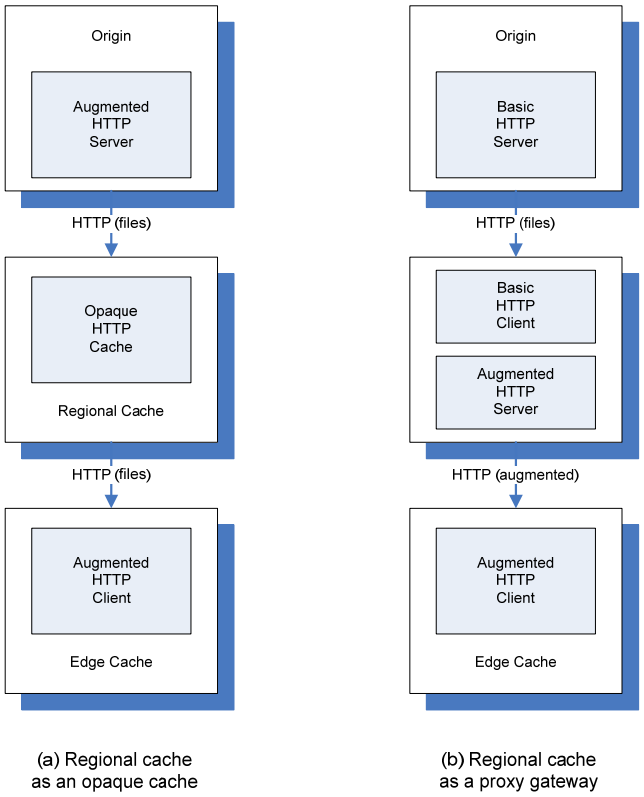

Figure 5: Regional cache options

In some cases, as shown in Fig. 5(b), the regional cache may function as a gateway between a basic origin (e.g., a NAS server) and an edge cache (or downstream client) that requires augmented services, either in the form of special streaming media URIs or additional delivery features. For example, in the VOD CDN application, an edge cache may require additional features, such as network-based trick mode, or bit-rate provisions, while relying on a basic origin. In this case, the regional cache is implemented as a *back-to-back* client-server, functioning as a proxy gateway.

Content Delivery Interface

While the caching modes and algorithms are best left to cache implementation, the industry must standardize the content delivery interface between two adjacent tiers of the open CDN. We propose a standard interface based on the HTTP v1.1 GET method. This includes support for HTTP byte-ranges, for random access and segmented transfer, and standard HTTP redirect to support local clustering and network request routing. We submit that this suffices for basic origin services.

For the managed IP video CDN application, the emerging HTTP adaptive streaming techniques provide for augmented origin services by employing special URI formats (for content fragments), *manifest* files that allow to interpret those formats, and index data-structures to translate from the

time domain to file byte-ranges. We propose a similar augmented origin service for the VOD CDN application, layered on top of basic HTTP. Specifically, we propose the adoption of:

- Special URI formats for VOD, to specify both normal-mode and trick-mode playback
- Support for HTTP time-range requests: for random-access and segmented transfer
- Bit-rate profiles via extended HTTP headers or independent HTTP (e.g., POST) messages.

This augmented service provides for quality-of-service, which might be important in the absence of bit-rate adaptation, and removes the need for clients to fetch and process VOD index files, hence increasing client scalability. We note that companion manifest files may be needed to maintain alignment of HTTP time-range requests so as to aid in segment cacheability.

In accordance with standard HTTP, we eschew special commands for aborting an open content transfer. Content transfer is terminated either when the entire requested range (byte or time) or fragment is served, or when the client closes the TCP connection, whichever is earlier. Since the CDN is stateless across transfers, there is no likelihood of orphaned state in any server. For fragment-by-fragment transfer, we recommend that the client (including the CDN caches, where applicable) use persistent TCP connections for successive fragment requests.

6. CONTROL PLANE: REQUEST ROUTING

The request routing function straddles the CDN data and control planes as shown in Fig. 6. In the figure, we illustrate the concept of “layered” request routing, wherein a server location decision in each tier of the CDN undergoes multiple phases, e.g., a *global request routing* decision that directs a request to a specific data center, and a *local request routing* decision that selects a specific server within that data center. This multi-layered decision is anchored at the requesting client, thus allowing any of the phases to be optional (e.g., a global routing decision that subsumes the local server selection, or a local routing decision subsumed by an HTTP proxy server), and allowing each phase to evolve independently, thus achieving one of the desirable properties of REST. For example, an operator may start with DNS techniques for global routing requesting, and, as the server footprint in a data center increases over time, add HTTP redirection for local routing.

Referring to Fig. 6, we separate out the logical functions associated with request routing in each tier of the CDN, again, allowing for independent evolution of each tier. For example, an *edge cache routing* function monitors the status of the available edge caches and directs end-user requests to the most appropriate edge cache, while a *regional cache routing* function monitors the status of the available regional caches and directs edge-cache requests (cache misses) to the most appropriate regional cache. As an example of independent evolution, the edge cache routing may be DNS-based and content unaware, the regional cache routing may be based on simple URI rewrite rules (or configuration), while origin routing may be based on HTTP redirection with content awareness.

We propose an open HTTP-based interface for server monitoring in each tier. In support of monitoring, each server in the hierarchy (including edge caches, regional caches, and origin servers) publishes a status URI, which the request routing engine of that tier may access via a standard GET message. In addition to server monitoring, a request routing engine may utilize various kinds of “location services,” e.g., to determine the location of a requesting client using geo-databases of IP

addresses, or to determine the location of a physical server that contains the requested content using information from the content management system.

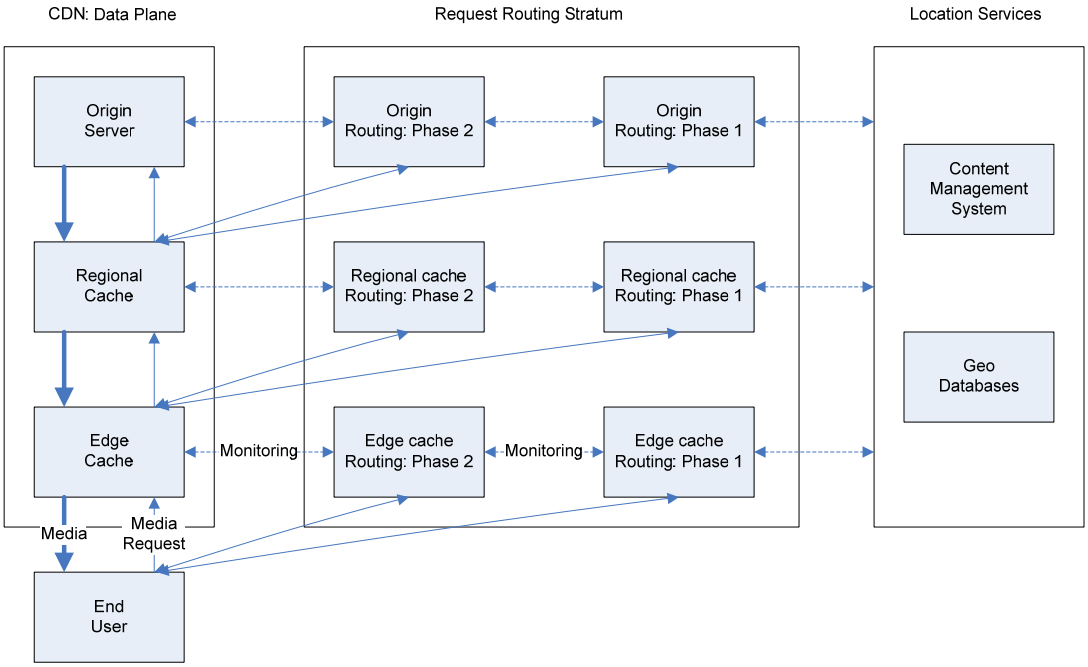

Figure 6: Multi-layered request routing architecture

Request routing relies heavily on the structure of the URIs crafted by the content management system. In some cases, the end-user URI contains sufficient information to assist in request routing all the way to the origin. Alternatively, request routing engines may need a location service from the content management system to map one URI to another, e.g., an end-user URI to an origin URI. Where needed, we propose an open HTTP-based interface for location service. In support of such a service, the content management system may publish a location service URI, which a request routing function may access via a standard GET message. The correct content management system to query may be deduced by the request router via rules on the incoming URI.

We enumerate four request routing techniques for an open CDN, any of which may be used in any of the phases/tiers of the multi-layered requesting routing stratum.

Independent URI Resolution

In this technique, the requesting client makes an *independent* HTTP request for an asset location, and is provided a *post-routed* URI (sometimes a post-routed URI prefix) that points directly to the correct server in the next tier. This is in contrast to *transparent* request routing, which happens as part of making the HTTP request for the asset itself. As the request routing is not transparent, it requires a special HTTP client that can first make the independent routing request, and can subsequently use the returned URI (or URI prefix) to make future requests for (different portions of) the same asset. Since the request routing uses an independent HTTP request, this scheme is amenable to persistent TCP connections in the data plane. Similarly, since a special HTTP client

cannot be avoided, the same client can implement logic to "cache" the URI prefix and use it for subsequent requests, thus ensuring cacheability.

As an example, in the VOD CDN application, the edge cache may query a request router using the asset identifier provided by the VOD session manager, and is provided a URI prefix that points to the correct regional cache or origin server. In the managed IP video CDN application, an edge cache may query a request router using the "content prefix" portion of the URI supplied by the end-user client, and is provided a different URI prefix that points to the correct regional cache or origin server. The special HTTP client in the edge cache contains the logic to use the same URI prefix for subsequent fragment requests.

URI Rewrite Rules

This is a lightweight technique that relies on a sufficiently expressive URI provided by the content management system to the end-user, and caches that are configured with rules to convert the incoming URI to a post-routed URI. No external request routing phases are necessary, making it easily amenable to persistent TCP connections in the data plane, while retaining the benefits of caching—as the same set of rules are applied on each incoming URI. The only drawbacks of this scheme are that it might be difficult to create such expressive URIs (and associated rules) in complex multi-tier CDNs, and it typically mandates additional means for local request routing and server monitoring. Specifically, URI rewrite rules may need to be coupled with local server selection, and the client may need to monitor the status of the server cluster itself.

As an example, the content management system may provide the end-user with a URI prefix *http://ss2.edge-server10.domain/hbo.com/superman/*. On a cache miss, *edge-server10* may replace the hostname portion with its configured regional cache, and originate a request for *http://ss2.regional-server5.domain/hbo.com/superman/*. The regional cache, in turn may convert the URI prefix to *http://ss2.hbo.origin.domain/superman/*, using a URI rewrite rule, in order to direct the request to the second streaming server in the origin cluster configured for *hbo*.

Enhanced DNS

DNS is one of the most popular request routing techniques in traditional CDNs. This technique employs an *authoritative* DNS server, responsible for resolving virtual host names (of the URIs served by the CDN) to IP addresses of the selected CDN nodes. The DNS server typically uses the location of the requesting node to determine the nearest server. This scheme is transparent, i.e., it needs no special logic in the client, amenable to persistent TCP connections in the data plane, and cacheable using standard DNS caching.

There are two significant drawbacks to enhanced DNS:

- The authoritative DNS server has to deduce the exact location of the requesting client, since the DNS request it receives originates from the *recursive* DNS server of the client, and not from the client itself. In some cases, this may not provide the necessary granularity to make the correct server selection.
- The DNS server has access only to the host name portion of the requested URI. This makes it difficult to perform content-aware request routing, where necessary. This drawback is sometime circumvented by including a host name prefix that contains information about the requested asset, e.g., a hash function of the asset, or the type of the asset. Nevertheless, it

might be difficult to incorporate the precise location of each asset, e.g., as populated by a content management system into multiple origin servers, into the host name prefix.

HTTP Redirection

HTTP redirection, typically using the standard temporary redirect (307) feature of HTTP, is another popular transparent request routing technique. No special clients are necessary, and unlike DNS-based schemes, the request router has access to the exact location of the requesting client. In this scheme, as part of standard HTTP redirection, a new URI is provided to the client in the location header of the response. Typically, this is an *absolute URI*, i.e., the entire original URI is modified.

HTTP redirection with absolute URIs is not particularly amenable to persistent TCP connections. Since each HTTP request must be redirected (since they have different absolute URIs), those requests cannot be multiplexed into a single TCP connection in the data plane. Due to the same reasons, each HTTP request must be individually redirected, i.e., it is not possible to re-use request routing state at the client across URI requests.

The above limitation may be addressed by including a *relative URI* in the location header of the HTTP response. The client may now use the same relative URI across requests that share the same prefix. This also makes it suitable for persistent TCP connections. However, the request router now has access only to the relative portion (prefix) of the requested URI, making it difficult to perform content-aware routing, where necessary. Again, this drawback may be circumvented by including another prefix to the host name that contains information about the requested asset.

7. POLICY AND MANAGEMENT

The CDN policy and management plane is the most customizable portion of the architecture, and hence the most variable. In order to allow for maximum flexibility, the CDN data plane nodes must be implemented as *policy-based* content delivery elements. In general, our proposed architecture allows for two types of policies:

- **Inline policies:** These refer to rules executed by a CDN data plane node, as part of delivering the requested content. For example, the URI rewrite rule for origin determination (described in the previous section) is a form of inline policy. Inline policies are typically implemented using a URI rules engine or scripts executed as part of HTTP delivery.
- **External policies:** These refer to policies executed by external policy servers. Such policies are realized by a CDN data plane node by making requests to the configured policy servers as part of HTTP delivery. The responses to such requests are either binary in nature, i.e., whether or not to serve the requested content, or are (X)HTML documents, which may be executed by scripts in the CDN node.

In the proposed open CDN, for each external policy, the respective policy server publishes a URI, which a CDN data plane node may access via a standard HTTP message. The responses to such policy requests may be cached by the CDN data plane node. Policy requests may be content-based, e.g., how long may an asset reside in cache, or subscriber-based, e.g., is a particular subscriber authorized to view content, or a combination of both.

A complete enumeration of various possible content delivery policies is beyond the scope of this paper. However, we provide the following examples for illustrative purposes:

- Session authentication: A token-based session authentication can be enforced at various tiers of the CDN, including caching nodes and origin server. In this case, the requesting CPE device uses a pre-assigned or pre-determined session token when requesting the content from an edge cache. The edge cache in turn queries an external policy server using the token and other subscriber information to carry out session authentication. The result of this authentication is cacheable until a time-to-live (TTL) period for any subsequent request.
- Cache policy control: This content-based policy ensures that the assets that are removed by the content management system from the origin servers are purged from downstream caches. In a typical implementation, the CMS notifies the policy server about content removal. The CDN data plane nodes (edge caches, regional caches) query the policy server before an asset is served. To aid in cacheability, the policy server responds with time-to-live (TTL) counters for each binary response. A short TTL results in quick purging, while a long TTL is better for cacheability.

8. SUMMARY

We have proposed guiding principles for the design of an open multi-screen content delivery network, based on best-of-breed IP technology principles. The proposed architecture is primarily based on the HTTP protocol with Representational State Transfer (REST) concepts. The CDN uses these concepts across its various planes, including the data plane, request routing stratum and the policy and management plane. We submit that such an approach will provide operators with architectural options that will result in the same scalability and distribution properties of the World Wide Web, and address the growing needs of video delivery to multiple types of end-user devices.

ACKNOWLEDGEMENTS

We thank the SCTE for the comments and support provided during the preparation of this paper. Special thanks is due to Steve Vinoski for several educational conversations on the topic of REST.

ABBREVIATIONS AND ACRONYMS

CDN	Content Delivery Network
CMS	Content Management System
CPE	Customer Premises Equipment
DNS	Domain Name Service
FTP	File Transfer Protocol
HTML	HyperText Markup Language
HTTP	HyperText Transfer Protocol

IP	Internet Protocol
NAS	Network Attached Storage
QAM	Quadrature Amplitude Modulation
REST	Representational State Transfer
RPC	Remote Procedure Call
TCP	Transmission Control Protocol
TTL	Time to Live
UDP	User Datagram Protocol
URI	Universal Resource Identifier
VOD	Video on Demand

REFERENCES

[1] R. Fielding et al., "HyperText Transfer Protocol—HTTP/1.1," Request for Comments (RFC) 2616, Internet Engineering Task Force, 1999

[2] R. Fielding, "Architectural Styles and the Design of Network-based Software Architectures," Doctoral Dissertation, Chap. 5, UC Irvine, 2000.

[3] W. Mao, "Building Large VOD Libraries with Next Generation On Demand Architecture", NCTA Technical Papers, 2008

[4] W. Mao, "Key Architecture and Interface Options for IP Video Over Cable", SCTE Conference on Emerging Technologies, 2009

[5] P. Mockapetris, "Domain Names – Implementation and Specification," Request for Comments (RFC) 1035, Internet Engineering Task Force, 1989

[6] T. Bray, ed., "Extensible Markup Language (XML) 1.0, Fifth Edition," W3C recommendation, *http://www.w3c.org/TR/REC-xml,* Nov 2008

[7] M. Nottingham et al., "The Atom Syndication Format," Request for Comments (RFC) 4287, Internet Engineering Task Force, Dec 2005

[8] R. Pantos, ed., "HTTP Live Streaming," *draft-pantos-http-live-streaming-01*, Internet Draft (work in progress), Internet Engineering Task Force, Jun 2009