Network Working Group
Internet-Draft
Obsoletes: 3466 (if approved)
Intended status: Informational
Expires: November 6, 2014

L. Peterson
Akamai Technologies, Inc.
B. Davie
VMware, Inc.
R. van Brandenburg, Ed.
TNO

May 5, 2014

Framework for CDN Interconnection
draft-ietf-cdni-framework-11

Abstract

This document presents a framework for Content Distribution Network Interconnection (CDNI). The purpose of the framework is to provide an overall picture of the problem space of CDNI and to describe the relationships among the various components necessary to interconnect CDNs. CDN Interconnection requires the specification of interfaces and mechanisms to address issues such as request routing, distribution metadata exchange, and logging information exchange across CDNs. The intent of this document is to outline what each interface needs to accomplish, and to describe how these interfaces and mechanisms fit together, while leaving their detailed specification to other documents. This document, in combination with RFC 6707, obsoletes RFC 3466.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at http://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on November 6, 2014.

Table of Contents

1. Introduction

This document provides an overview of the various components necessary to interconnect CDNs, expanding on the problem statement and use cases introduced in [RFC6770] and [RFC6707]. It describes the necessary interfaces and mechanisms in general terms and outlines how they fit together to form a complete system for CDN Interconnection. Detailed specifications are left to other documents. This document makes extensive use of message flow examples to illustrate the operation of interconnected CDNs, but these examples should be considered illustrative rather than prescriptive.

[RFC3466] uses different terminology and models for "Content Internetworking (CDI)". It is also less prescriptive in terms of interfaces. To avoid confusion, this document obsoletes [RFC3466].

1.1. Terminology

This document uses the core terminology defined in [RFC6707]. It also introduces the following terms:

CDN-Domain: a host name (FQDN) at the beginning of a URL (excluding port and scheme), representing a set of content that is served by a given CDN. For example, in the URL http://cdn.csp.example/...rest of url..., the CDN domain is cdn.csp.example. A major role of CDN-Domain is to identify a region (subset) of the URI space relative to which various CDN Interconnection rules and policies are to apply. For example, a record of CDN Metadata might be defined for the set of resources corresponding to some CDN-Domain.

Distinguished CDN-Domain: a CDN-Domain that is allocated by a CDN for the purposes of communication with a peer CDN, but which is not found

in client requests. Such CDN-Domains may be used for inter-CDN acquisition, or as redirection targets, and enable a CDN to distinguish a request from a peer CDN from an end-user request.

Delivering CDN: the CDN that ultimately delivers a piece of content to the end-user. The last in a potential sequence of downstream CDNs.

Recursive CDNI Request Redirection: When an upstream CDN elects to redirect a request towards a downstream CDN, the upstream CDN can query the downstream CDN Request Routing system via the CDNI Request Routing Redirection Interface (or use information cached from earlier similar queries) to find out how the downstream CDN wants the request to be redirected, which allows the upstream CDN to factor in the downstream CDN response when redirecting the user agent. This approach is referred to as "Recursive" CDNI Request Redirection. Note that the downstream CDN may elect to have the request redirected directly to a Surrogate inside the downstream CDN, to the Request Routing System of the downstream CDN, to another CDN, or to whatever system is necessary to handle the redirected request appropriately.

Iterative CDNI Request Redirection: When an upstream CDN elects to redirect a request towards a downstream CDN, the upstream CDN can base its redirection purely on a local decision (and without attempting to take into account how the downstream CDN may in turn redirect the user agent). In that case, the upstream CDN redirects the request to the request routing system in the downstream CDN, which in turn will decide how to redirect that request: this approach is referred to as "Iterative" CDNI Request Redirection.

Synchronous CDNI operations: operations between CDNs that happen during the process of servicing a user request, i.e. between the time that the user agent begins its attempt to obtain content and the time at which that request is served.

Asynchronous CDNI operations: operations between CDNs that happen independently of any given user request, such as advertisement of footprint information or pre-positioning of content for later delivery.

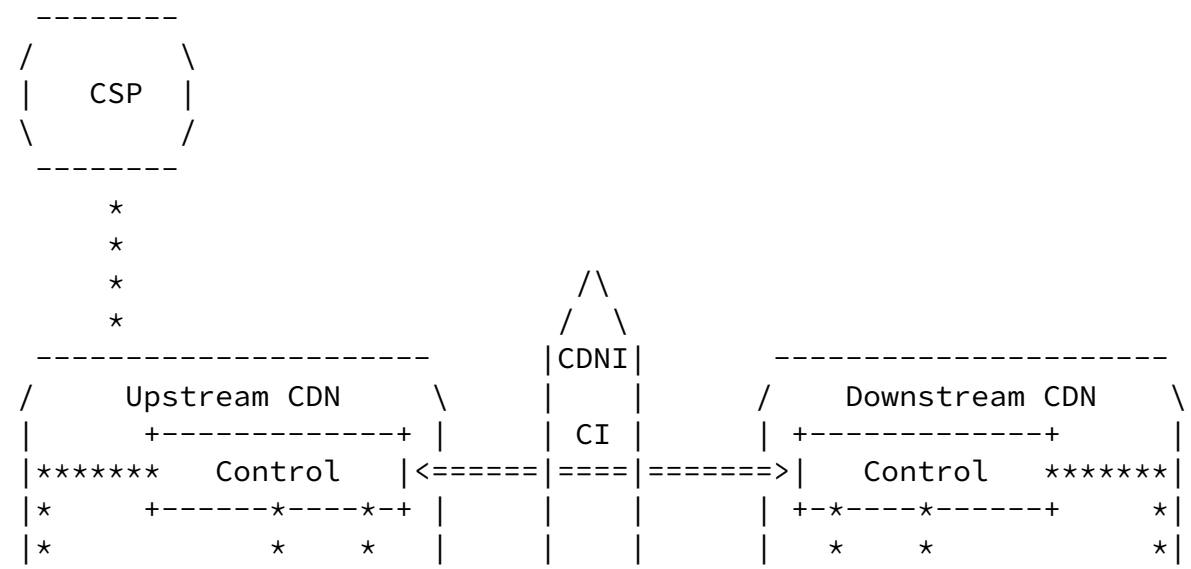
Trigger Interface: a subset of the CDNI Control interface that includes operations to pre-position, revalidate, and purge both metadata and content. These operations are typically called in response to some action (Trigger) by the CSP on the upstream CDN.

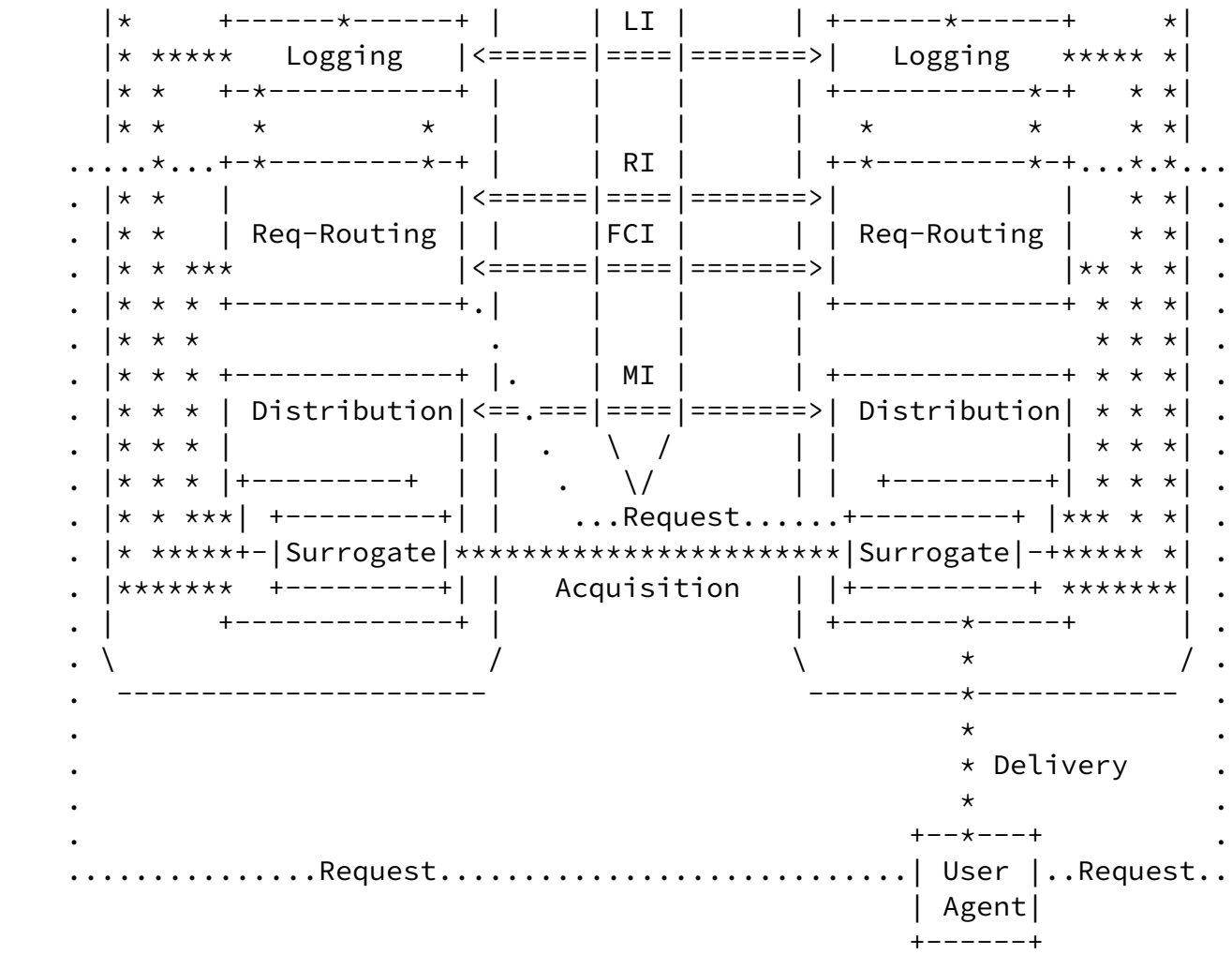
We also sometimes use uCDN and dCDN as shorthand for upstream CDN and downstream CDN, respectively.

At various points in this document, the concept of a CDN footprint is used. For a discussion on what constitutes a CDN footprint, the reader is referred to [I-D.ietf-cdni-footprint-capabilities-semantics].

1.2. Reference Model

This document uses the reference model in Figure 1, which expands the reference model originally defined in [RFC6707]. (The difference is that the expanded model splits the Request Routing Interface into its two distinct parts: the Request Routing Redirection interface and the Footprint and Capabilities Advertisement interface, as described below.)


CSP
Upstream CDN
Control
CDNI
CI
Downstream CDN
Control



<==> interfaces inside the scope of CDNI

**** and .... interfaces outside the scope of CDNI

Figure 1: CDNI Expanded Model and CDNI Interfaces

We note that while some interfaces in the reference model are "out of scope" for the CDNI WG (in the sense that there is no need to define new protocols for those interfaces) we still need to refer to them in this document to explain the overall operation of CDNI.

We also note that, while we generally show only one upstream CDN serving a given CSP, it is entirely possible that multiple uCDNs can serve a single CSP. In fact, this situation effectively exists today

in the sense that a single CSP can currently delegate its content delivery to more than one CDN.

The following briefly describes the five CDNI interfaces, paraphrasing the definitions given in [RFC6707]. We discuss these interfaces in more detail in Section 4.

- CDNI Control interface (CI): Operations to bootstrap and parameterize the other CDNI interfaces, as well as operations to pre-position, revalidate, and purge both metadata and content. The latter subset of operations is sometimes collectively called the "Trigger interface."

- CDNI Request Routing interface: Operations to determine what CDN (and optionally what surrogate within a CDN) is to serve end-user's requests. This interface is actually a logical bundling of two separate but related interfaces:

  * CDNI Footprint & Capabilities Advertisement interface (FCI): Asynchronous operations to exchange routing information (e.g., the network footprint and capabilities served by a given CDN) that enables CDN selection for subsequent user requests; and

  * CDNI Request Routing Redirection interface (RI): Synchronous operations to select a delivery CDN (surrogate) for a given user request.

- CDNI Metadata interface (MI): Operations to communicate metadata that governs how the content is delivered by interconnected CDNs. Examples of CDNI metadata include geo-blocking directives, availability windows, access control mechanisms, and purge directives. It may include a combination of:

  * Asynchronous operations to exchange metadata that govern subsequent user requests for content; and

  * Synchronous operations that govern behavior for a given user request for content.

- CDNI Logging interface (LI): Operations that allow interconnected

CDNs to exchange relevant activity logs. It may include a combination of:

* Real-time exchanges, suitable for runtime traffic monitoring; and

* Offline exchanges, suitable for analytics and billing.

The division between the sets of Trigger-based operations in the CDNI Control interface and the CDNI Metadata interface is somewhat arbitrary. For both cases, the information passed from the upstream CDN to the downstream CDN can broadly be viewed as metadata that describes how content is to be managed by the downstream CDN. For example, the information conveyed by CI to pre-position, revalidate or purge metadata is similar to the information conveyed by posting updated metadata via the MI. Even the CI operation to purge content could be viewed as a metadata update for that content: purge simply says that the availability window for the named content ends now. The two interfaces share much in common, so minimally, there will need to be a consistent data model that spans both.

The distinction we draw has to do with what the uCDN knows about the successful application of the metadata by the dCDN. In the case of the CI, the downstream CDN returning a successful status message guarantees that the operation has been successfully completed; e.g., the content has been purged or pre-positioned. This implies that the downstream CDN accepts responsibility for having successfully completed the requested operation. In contrast, metadata passed between CDNs via the MI carries no such completion guarantee. Returning success implies successful receipt of the metadata, but nothing can be inferred about precisely when the metadata will take effect in the downstream CDN, only that it will take effect eventually. This is because of the challenge in globally synchronizing updates to metadata with end-user requests that are currently in progress (or indistinguishable from currently being in progress). Clearly, a CDN will not be viewed as a trusted peer if "eventually" often becomes an indefinite period of time, but the acceptance of responsibility cannot be as crisply defined for the MI.

Finally, there is a practical issue that impacts all of the CNDI interfaces, and that is whether or not to optimize CDNI for HTTP Adaptive Streaming (HAS). We highlight specific issues related to delivering HAS content throughout this document, but for a more thorough treatment of the topic, see [RFC6983].

1.3. Structure Of This Document

The remainder of this document is organized as follows:

- Section 2 describes some essential building blocks for CDNI, notably the various options for redirecting user requests to a given CDN.
- Section 3 provides a number of illustrative examples of various CDNI operations.
- Section 4 describes the functionality of the main CDNI interfaces.
- Section 5 shows how various deployment models of CDNI may be achieved using the defined interfaces.
- Section 6 describes the trust model of CDNI and the issues of transitive trust in particular that CDNI raises.

2. Building Blocks

2.1. Request Redirection

At its core, CDN Interconnection requires the redirection of requests from one CDN to another. For any given request that is received by an upstream CDN, it will either respond to the request directly, or somehow redirect the request to a downstream CDN. Two main mechanisms are available for redirecting a request to a downstream CDN. The first leverages the DNS name resolution process and the second uses application-layer redirection mechanisms such as the HTTP 302 or RTSP 302 redirection responses. While there exists a large variety of application-layer protocols that include some form of redirection mechanism, this document will use HTTP (and HTTPS) in its examples. Similar mechanisms can be applied to other application-layer protocols. What follows is a short discussion of both DNS- and HTTP-based redirection, before presenting some examples of their use in Section 3.

2.1.1. DNS Redirection

DNS redirection is based on returning different IP addresses for the same DNS name, for example, to balance server load or to account for the client's location in the network. A DNS server, sometimes called the Local DNS (LDNS), resolves DNS names on behalf of an end-user. The LDNS server in turn queries other DNS servers until it reaches the authoritative DNS server for the CDN-Domain. The network

operator typically provides the LDNS server, although the user is free to choose other DNS servers (e.g., OpenDNS, Google Public DNS).

This latter possibility is important because the authoritative DNS server sees only the IP address of the DNS server that queries it, not the IP address of the original end-user.

The advantage of DNS redirection is that it is completely transparent to the end user; the user sends a DNS name to the LDNS server and gets back an IP address. On the other hand, DNS redirection is problematic because the DNS request comes from the LDNS server, not the end-user. This may affect the accuracy of server selection that is based on the user's location. The transparency of DNS redirection is also a problem in that there is no opportunity to take the attributes of the user agent or the URI path component into account. We consider two main forms of DNS redirection: simple and CNAME-based.

In simple DNS redirection, the authoritative DNS server for the name simply returns an IP address from a set of possible IP addresses. The answer is chosen from the set based on characteristics of the set (e.g., the relative loads on the servers) or characteristics of the client (e.g., the location of the client relative to the servers). Simple redirection is straightforward. The only caveats are (1) there is a limit to the number of alternate IP addresses a single DNS server can manage; and (2) DNS responses are cached by downstream servers so the TTL on the response must be set to an appropriate value so as to preserve the freshness of the redirection.

In CNAME-based DNS redirection, the authoritative server returns a CNAME response to the DNS request, telling the LDNS server to restart the name lookup using a new name. A CNAME is essentially a symbolic link in the DNS namespace, and like a symbolic link, redirection is transparent to the client; the LDNS server gets the CNAME response and re-executes the lookup. Only when the name has been resolved to an IP address does it return the result to the user. Note that DNAME would be preferable to CNAME if it becomes widely supported.

One of the advantages of DNS redirection compared to HTTP redirection is that it can be cached, reducing load on the redirecting CDN's DNS server. However, this advantage can also be a drawback, especially when a given DNS resolver doesn't strictly adhere to the TTL, which

is a known problem in some real world environments. In such cases, an end-user might end up at a dCDN without first having passed through the uCDN, which might be an undesirable scenario from a uCDN point of view.

2.1.2. HTTP Redirection

HTTP redirection makes use of the redirection response of the HTTP protocol (e.g.,"302" or "307"). This response contains a new URL that the application should fetch instead of the original URL. By changing the URL appropriately, the server can cause the user to redirect to a different server. The advantages of HTTP redirection are that (1) the server can change the URL fetched by the client to include, for example, both the DNS name of the particular server to use, as well as the original HTTP server that was being accessed; (2) the client sends the HTTP request to the server, so that its IP address is known and can be used in selecting the server; and (3) other attributes (e.g., content type, user agent type) are visible to the redirection mechanism.

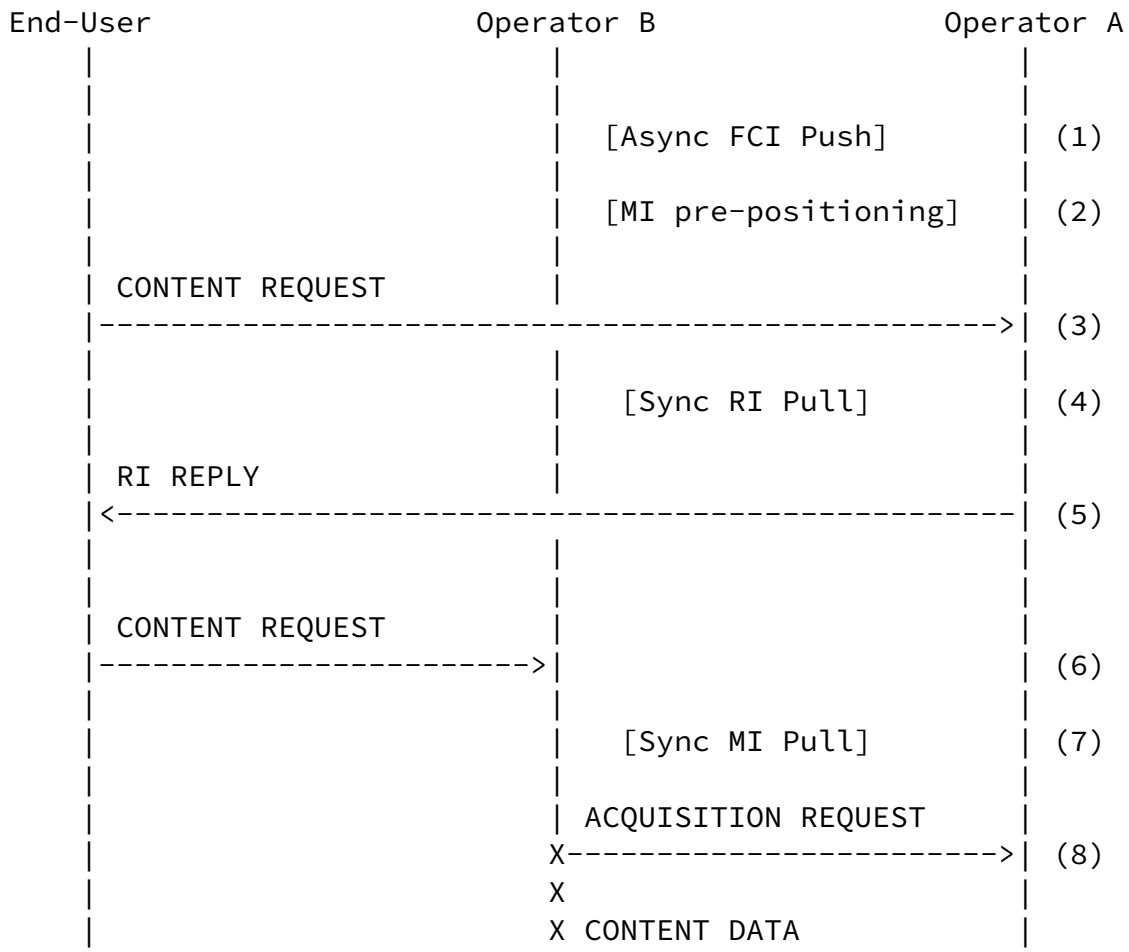
Just as is the case for DNS redirection, there are some potential disadvantages of using HTTP redirection. For example, it may affect application behavior, e.g. web browsers will not send cookies if the URL changes to a different domain. In addition, although this might also be an advantage, results of HTTP redirection are not cached so that all redirections must go through to the uCDN.

3. Overview of CDNI Operation

To provide a big picture overview of the various components of CDN Interconnection, we walk through a "day in the life" of a content item that is made available via a pair of interconnected CDNs. This will serve to illustrate many of the functions that need to be supported in a complete CDNI solution. We give examples using both DNS-based and HTTP-based redirection. We begin with very simple examples and then show how additional capabilities, such as recursive request redirection and content removal, might be added.

Before walking through the specific examples, we present a high-level view of the operations that may take place. This high-level overview is illustrated in Figure 2. Note that most operations will involve only a subset of all the messages shown below, and that the order and number of operations may vary considerably, as the more detailed examples illustrate.

The following shows Operator A as the upstream CDN (uCDN) and Operator B as the downstream CDN (dCDN), where the former has a relationship with a content provider and the latter being the CDN selected by Operator A to deliver content to the end-user. The interconnection relationship may be symmetric between these two CDN operators, but each direction can be considered as operating independently of the other so for simplicity we show the interaction in one direction only.



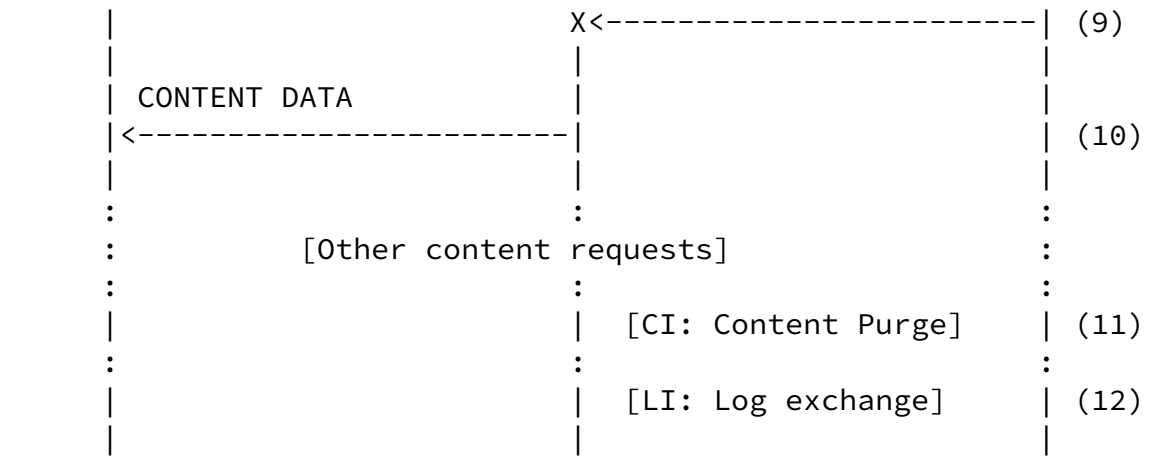


Figure 2: Overview of Operation

The operations shown in the Figure are as follows:

1. dCDN uses the FCI to advertise information relevant to its delivery footprint and capabilities prior to any content requests being redirected.

2. Prior to any content request, the uCDN uses the MI to pre-position CDNI metadata to the dCDN, thereby making that metadata available in readiness for later content requests.

3. A content request from a user agent arrives at uCDN.

4. uCDN may use the RI to synchronously request information from dCDN regarding its delivery capabilities to decide if dCDN is a suitable target for redirection of this request.

5. uCDN redirects the request to dCDN by sending some response (DNS, HTTP) to the user agent.

6. The user agent requests the content from dCDN.

7. dCDN may use the MI to synchronously request metadata related to this content from uCDN, e.g. to decide whether to serve it.

8. If the content is not already in a suitable cache in dCDN, dCDN may acquire it from uCDN.

9. The content is delivered to dCDN from uCDN.

10. The content is delivered to the user agent by dCDN.

11. Some time later, perhaps at the request of the CSP (not shown) uCDN may use the CI to instruct dCDN to purge the content, thereby ensuring it is not delivered again.

12. After one or more content delivery actions by dCDN, a log of delivery actions may be provided to uCDN using the LI.

The following sections show some more specific examples of how these operations may be combined to perform various delivery, control and logging operations across a pair of CDNs.

3.1. Preliminaries

Initially, we assume that there is at least one CSP that has contracted with an upstream CDN (uCDN) to deliver content on its behalf. We are not particularly concerned with the interface between the CSP and uCDN, other than to note that it is expected to be the same as in the "traditional" (non-interconnected) CDN case. Existing mechanisms such as DNS CNAMEs or HTTP redirects (Section 2) can be used to direct a user request for a piece of content from the CSP towards the CSP's chosen upstream CDN.

We assume Operator A provides an upstream CDN that serves content on behalf of a CSP with CDN-Domain cdn.csp.example. We assume that Operator B provides a downstream CDN. An end user at some point makes a request for URL

http://cdn.csp.example/...rest of url...

It may well be the case that cdn.csp.example is just a CNAME for some other CDN-Domain (such as csp.op-a.example). Nevertheless, the HTTP request in the examples that follow is assumed to be for the example URL above.

Our goal is to enable content identified by the above URL to be served by the CDN of operator B. In the following sections we will walk through some scenarios in which content is served, as well as other CDNI operations such as the removal of content from a downstream CDN.

3.2. Iterative HTTP Redirect Example

In this section we walk through a simple, illustrative example using HTTP redirection from uCDN to dCDN. The example also assumes the use of HTTP redirection inside uCDN and dCDN; however, this is independent of the choice of redirection approach across CDNs, so an alternative example could be constructed still showing HTTP redirection from uCDN to dCDN but using DNS for handling of request inside each CDN.

We assume for this example that Operators A and B have established an agreement to interconnect their CDNs, with A being upstream and B being downstream.

The operators agree that a CDN-Domain peer-a.op-b.example will be used as the target of redirections from uCDN to dCDN. We assume the name of this domain is communicated by some means to each CDN. (This could be established out-of-band or via a CDNI interface.) We refer to this domain as a "distinguished" CDN-Domain to convey the fact that its use is limited to the interconnection mechanism; such a domain is never used directly by a CSP.

We assume the operators also agree on some distinguished CDN-Domain that will be used for inter-CDN acquisition of CSP's content from uCDN by dCDN. In this example, we'll use op-b-acq.op-a.example.

We assume the operators also exchange information regarding which requests dCDN is prepared to serve. For example, dCDN may be prepared to serve requests from clients in a given geographical

region or a set of IP address prefixes. This information may again be provided out of band or via a defined CDNI interface.

We assume DNS is configured in the following way:

- The content provider is configured to make operator A the authoritative DNS server for cdn.csp.example (or to return a CNAME for cdn.csp.example for which operator A is the authoritative DNS server).

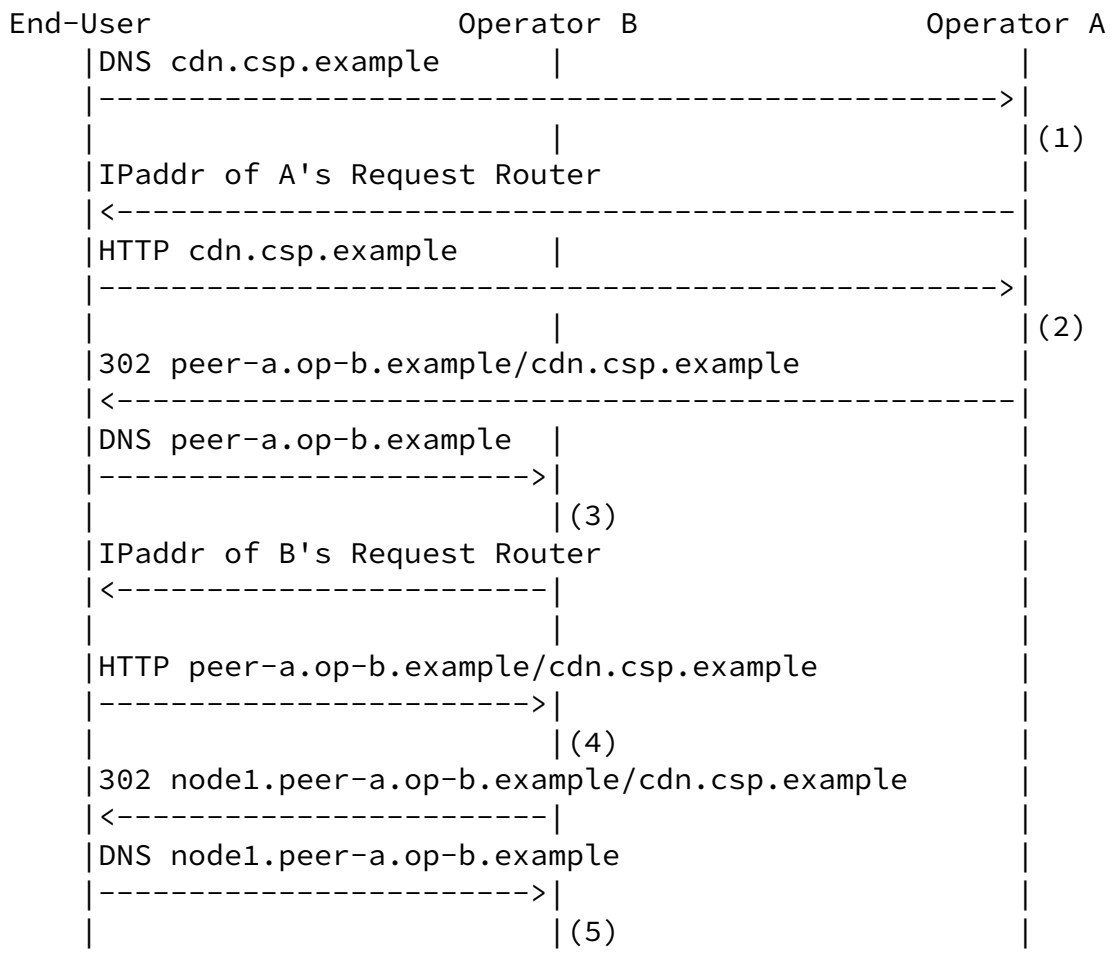
- Operator A is configured so that a DNS request for op-b-acq.op-a.example returns a request router in Operator A.

- Operator B is configured so that a DNS request for peer-a.op-b.example/cdn.csp.example returns a request router in Operator B.

Figure 3 illustrates how a client request for

http://cdn.csp.example/...rest of url...

is handled.



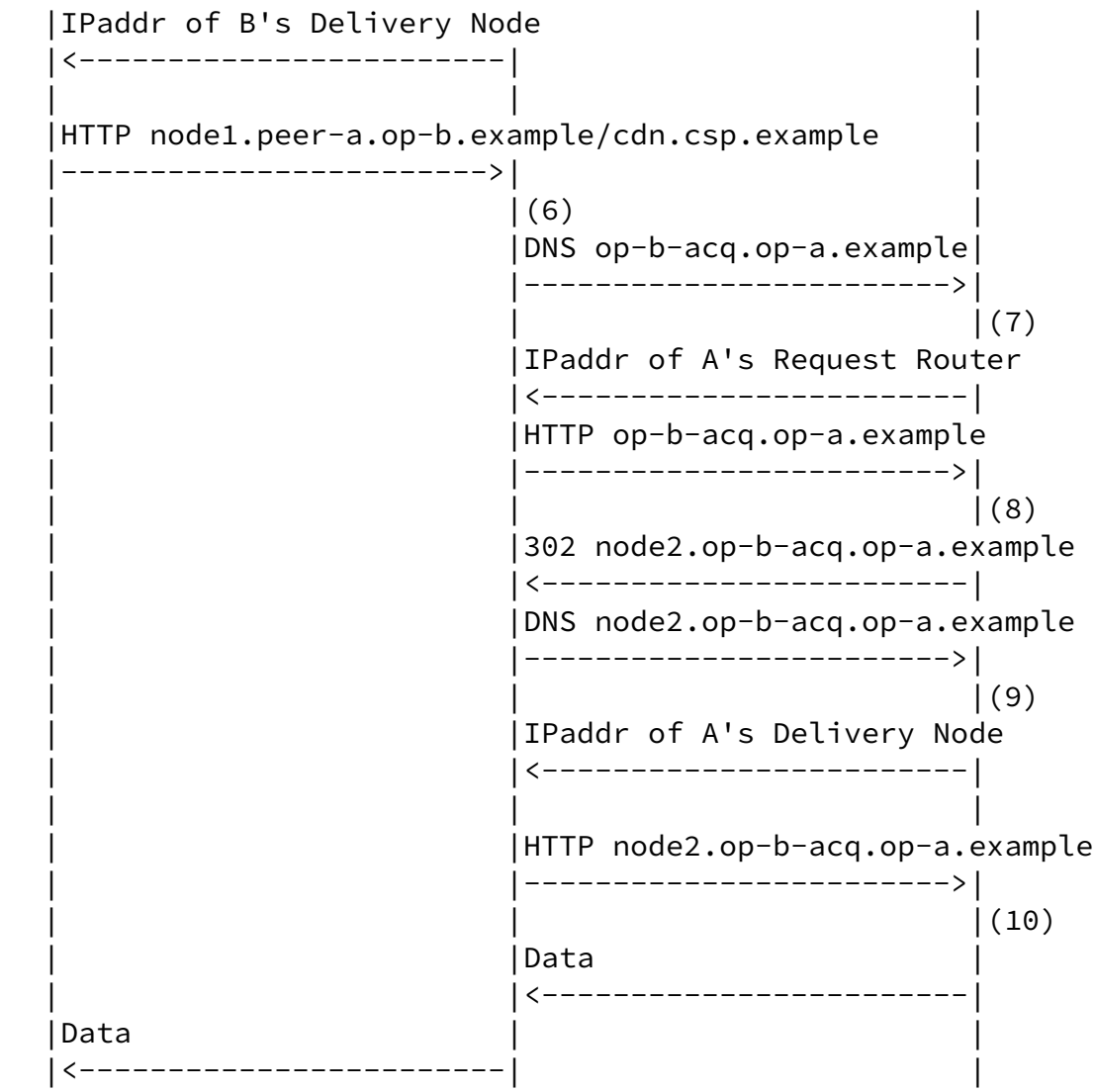


Figure 3: Message Flow for Iterative HTTP Redirection

The steps illustrated in the figure are as follows:

1. A DNS resolver for Operator A processes the DNS request for its customer based on CDN-Domain cdn.csp.example. It returns the IP address of a request router in Operator A.

2. A Request Router for Operator A processes the HTTP request and recognizes that the end-user is best served by another CDN, specifically one provided by Operator B, and so it returns a 302 redirect message for a new URL constructed by "stacking" Operator B's distinguished CDN-Domain (peer-a.op-b.example) on the front of the original URL. (Note that more complex URL manipulations are possible, such as replacing the initial CDN-Domain by some opaque handle.)

3. The end-user does a DNS lookup using Operator B's distinguished CDN-Domain (peer-a.op-b.example). B's DNS resolver returns the IP address of a request router for Operator B. Note that if request routing within dCDN was performed using DNS instead of HTTP redirection, B's DNS resolver would also behave as the request router and directly return the IP address of a delivery node.

4. The request router for Operator B processes the HTTP request and selects a suitable delivery node to serve the end-user request, and returns a 302 redirect message for a new URL constructed by replacing the hostname with a subdomain of the Operator B's distinguished CDN-Domain that points to the selected delivery node.

5. The end-user does a DNS lookup using Operator B's delivery node subdomain (node1.peer-a.op-b.example). B's DNS resolver returns the IP address of the delivery node.

6. The end-user requests the content from B's delivery node. In the case of a cache hit, steps 6, 7, 8, 9 and 10 below do not happen, and the content data is directly returned by the delivery node to the end-user. In the case of a cache miss, the content needs to be acquired by dCDN from uCDN (not the CSP). The distinguished CDN-Domain peer-a.op-b.example indicates to dCDN that this content is to be acquired from uCDN; stripping the CDN-Domain reveals the original CDN-Domain cdn.csp.example and dCDN may verify that this CDN-Domain belongs to a known peer (so as to avoid being tricked into serving as an open proxy). It then does a DNS request for an inter-CDN acquisition CDN-Domain as agreed above (in this case, op-b-acq.op-a.example).

7. Operator A's DNS resolver processes the DNS request and returns the IP address of a request router in operator A.

8. The request router for Operator A processes the HTTP request from Operator B delivery node. Operator A request router recognizes that the request is from a peer CDN rather than an end-user because of the dedicated inter-CDN acquisition domain (op-b-acq.op-a.example). (Note that without this specially

defined inter-CDN acquisition domain, operator A would be at risk of redirecting the request back to operator B, resulting in an infinite loop). The request router for Operator A selects a suitable delivery node in uCDN to serve the inter-CDN acquisition request and returns a 302 redirect message for a new URL constructed by replacing the hostname with a subdomain of the Operator A's distinguished inter-CDN acquisition domain that points to the selected delivery node.

9. Operator A DNS resolver processes the DNS request and returns the IP address of the delivery node in operator A.

10. Operator B requests (acquires) the content from Operator A. Although not shown, Operator A processes the rest of the URL: it extracts information identifying the origin server, validates that this server has been registered, and determines the content provider that owns the origin server. It may also perform its own content acquisition steps if needed before returning the content to dCDN.

The main advantage of this design is that it is simple: each CDN need only know the distinguished CDN-Domain for each peer, with the upstream CDN "pushing" the downstream CDN-Domain onto the URL as part of its redirect (step 2) and the downstream CDN "popping" its CDN-Domain off the URL to expose a CDN-Domain that the upstream CDN can correctly process. Neither CDN needs to be aware of the internal structure of the other's URLs. Moreover, the inter-CDN redirection is entirely supported by a single HTTP redirect; neither CDN needs to be aware of the other's internal redirection mechanism (i.e., whether it is DNS or HTTP based).

One disadvantage is that the end-user's browser is redirected to a new URL that is not in the same domain of the original URL. This has implications on a number of security or validation mechanisms sometimes used on endpoints. For example, it is important that any redirected URL be in the same domain (e.g., csp.example) if the browser is expected to send any cookies associated with that domain. As another example, some video players enforce validation of a cross domain policy that needs to accommodate the domains involved in the CDN redirection. These problems are generally solvable, but the solutions complicate the example, so we do not discuss them further in this document.

We note that this example begins to illustrate some of the interfaces that may be required for CDNI, but does not require all of them. For example, obtaining information from dCDN regarding the set of client IP addresses or geographic regions it might be able to serve is an aspect of request routing (specifically of the CDNI Footprint & Capabilities Advertisement interface). Important configuration information such as the distinguished names used for redirection and inter-CDN acquisition could also be conveyed via a CDNI interface (e.g., perhaps the CDNI Control interface). The example also shows how existing HTTP-based methods suffice for the acquisition interface. Arguably, the absolute minimum metadata required for CDNI is the information required to acquire the content, and this information was provided "in-band" in this example by means of the URI handed to the client in the HTTP 302 response. The example also

assumes that the CSP does not require any distribution policy (e.g. time window, geo-blocking) or delivery processing to be applied by the interconnected CDNs. Hence, there is no explicit CDNI Metadata interface invoked in this example. There is also no explicit CDNI Logging interface discussed in this example.

We also note that the step of deciding when a request should be redirected to dCDN rather than served by uCDN has been somewhat glossed over. It may be as simple as checking the client IP address against a list of prefixes, or it may be considerably more complex, involving a wide range of factors, such as the geographic location of the client (perhaps determined from a third party service), CDN load, or specific business rules.

This example uses the "iterative" CDNI request redirection approach. That is, uCDN performs part of the request redirection function by redirecting the client to a request router in the dCDN, which then performs the rest of the redirection function by redirecting to a suitable surrogate. If request routing is performed in the dCDN using HTTP redirection, this translates in the end-user experiencing two successive HTTP redirections. By contrast, the alternative approach of "recursive" CDNI request redirection effectively coalesces these two successive HTTP redirections into a single one, sending the end-user directly to the right delivery node in the dCDN. This "recursive" CDNI request routing approach is discussed in the next section.

While the example above uses HTTP, the iterative HTTP redirection mechanism would work over HTTPS in a similar fashion. In order to make sure an end-user's HTTPS request is not downgraded to HTTP along the redirection path, it is necessary for every request router along the path from the initial uCDN Request Router to the final surrogate in the dCDN to respond to an incoming HTTPS request with an HTTP Redirect containing an HTTPS URL. It should be noted that using HTTPS will have the effect of increasing the total redirection process time and increasing the load on the request routers, especially when the redirection path includes many redirects and thus many TLS/SSL sessions. In such cases, a recursive HTTP redirection mechanism, as described in an example in the next section, might help to reduce some of these issues.

3.3. Recursive HTTP Redirection Example

The following example builds on the previous one to illustrate the use of the request routing interface (specifically the CDNI Request Routing Redirection interface) to enable "recursive" CDNI request routing. We build on the HTTP-based redirection approach because it illustrates the principles and benefits clearly, but it is equally

possible to perform recursive redirection when DNS-based redirection is employed.

In contrast to the prior example, the operators need not agree in advance on a CDN-Domain to serve as the target of redirections from uCDN to dCDN. We assume that the operators agree on some distinguished CDN-Domain that will be used for inter-CDN acquisition of CSP's content by dCDN. In this example, we'll use op-b-acq.op-a.example.

We assume the operators also exchange information regarding which requests dCDN is prepared to serve. For example, dCDN may be prepared to serve requests from clients in a given geographical region or a set of IP address prefixes. This information may again be provided out of band or via a defined protocol.

We assume DNS is configured in the following way:

- The content provider is configured to make operator A the

authoritative DNS server for cdn.csp.example (or to return a CNAME for cdn.csp.example for which operator A is the authoritative DNS server).

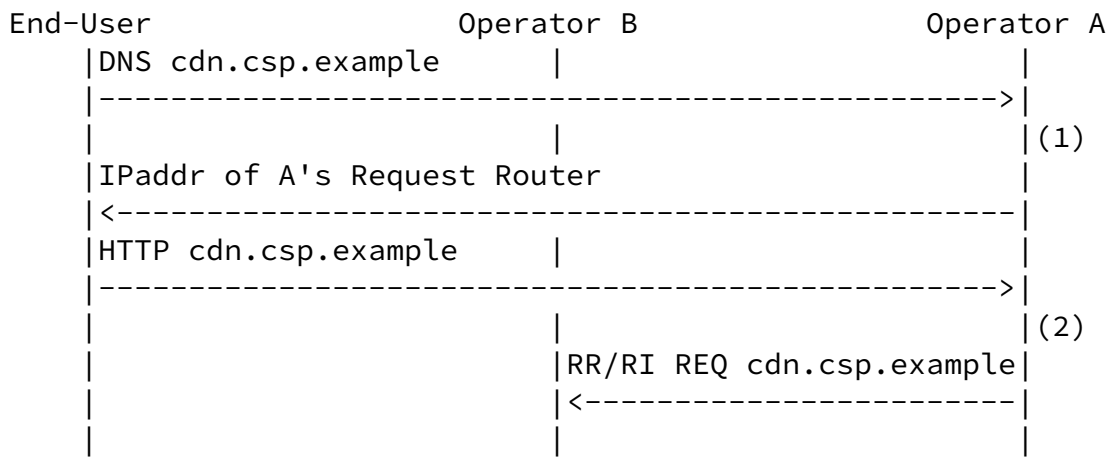
- Operator A is configured so that a DNS request for op-b-acq.op-a.example returns a request router in Operator A.

- Operator B is configured so that a request for node1.op-b.example/cdn.csp.example returns the IP address of a delivery node. Note that there might be a number of such delivery nodes.

Figure 3 illustrates how a client request for

http://cdn.csp.example/...rest of url...

is handled.



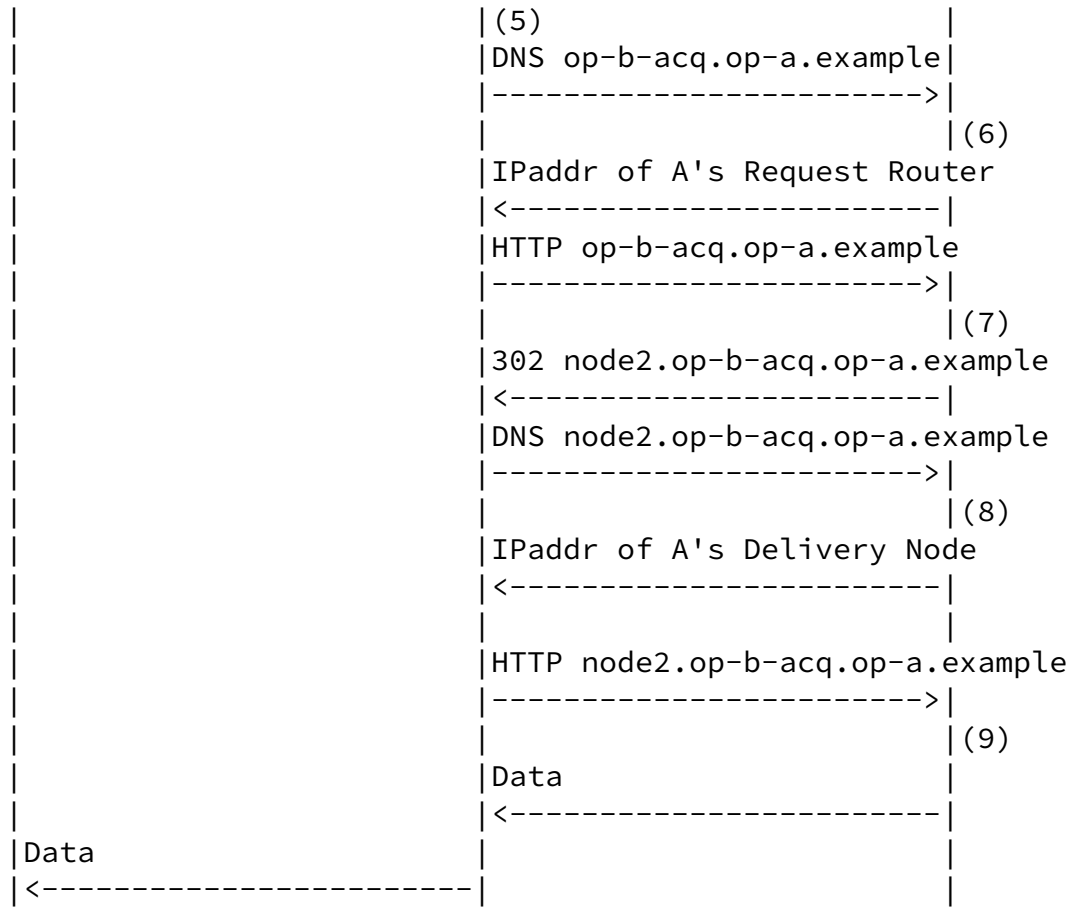


Figure 4: Message Flow for Recursive HTTP Redirection

The steps illustrated in the figure are as follows:

1. A DNS resolver for Operator A processes the DNS request for its customer based on CDN-Domain cdn.csp.example. It returns the IP address of a Request Router in Operator A.

2. A Request Router for Operator A processes the HTTP request and recognizes that the end-user is best served by another CDN--specifically one provided by Operator B--and so it queries the

CDNI Request Routing Redirection interface of Operator B, providing a set of information about the request including the URL requested. Operator B replies with the DNS name of a delivery node.

3. Operator A returns a 302 redirect message for a new URL obtained from the RI.

4. The end-user does a DNS lookup using the host name of the URL just provided (node1.op-b.example). B's DNS resolver returns the IP address of the corresponding delivery node. Note that, since the name of the delivery node was already obtained from B using the RI, there should not be any further redirection here (in contrast to the iterative method described above.)

5. The end-user requests the content from B's delivery node, potentially resulting in a cache miss. In the case of a cache miss, the content needs to be acquired from uCDN (not the CSP.) The distinguished CDN-Domain op-b.example indicates to dCDN that this content is to be acquired from another CDN; stripping the CDN-Domain reveals the original CDN-Domain cdn.csp.example, dCDN may verify that this CDN-Domain belongs to a known peer (so as to avoid being tricked into serving as an open proxy). It then does a DNS request for the inter-CDN Acquisition "distinguished" CDN-Domain as agreed above (in this case, op-b-acq.op-a.example).

6. Operator A DNS resolver processes the DNS request and returns the IP address of a request router in operator A.

7. The request router for Operator A processes the HTTP request from Operator B delivery node. Operator A request router recognizes that the request is from a peer CDN rather than an end-user because of the dedicated inter-CDN acquisition domain (op-b-acq.op-a.example). (Note that without this specially defined inter-CDN acquisition domain, operator A would be at risk of redirecting the request back to operator B, resulting in an infinite loop). The request router for Operator A selects a suitable delivery node in uCDN to serve the inter-CDN acquisition request and returns a 302 redirect message for a new URL constructed by replacing the hostname with a subdomain of the Operator A's distinguished inter-CDN acquisition domain that points to the selected delivery node.

8. Operator A recognizes that the DNS request is from a peer CDN rather than an end-user (due to the internal CDN-Domain) and so returns the address of a delivery node. (Note that without this specially defined internal domain, Operator A would be at risk of

redirecting the request back to Operator B, resulting in an infinite loop.)

9. Operator B requests (acquires) the content from Operator A. Operator A serves content for the requested CDN-Domain to dCDN. Although not shown, it is at this point that Operator A processes the rest of the URL: it extracts information identifying the origin server, validates that this server has been registered, and determines the content provider that owns the origin server. It may also perform its own content acquisition steps if needed before returning the content to dCDN.

Recursive redirection has the advantage over iterative of being more transparent from the end-user's perspective, but the disadvantage of each CDN exposing more of its internal structure (in particular, the addresses of edge caches) to peer CDNs. By contrast, iterative redirection does not require dCDN to expose the addresses of its edge caches to uCDN.

This example happens to use HTTP-based redirection in both CDN A and CDN B, but a similar example could be constructed using DNS-based redirection in either CDN. Hence, the key point to take away here is simply that the end user only sees a single redirection of some type, as opposed to the pair of redirections in the prior (iterative) example.

The use of the RI requires that the request routing mechanism be appropriately configured and bootstrapped, which is not shown here. More discussion on the bootstrapping of interfaces is provided in Section 4

3.4. Iterative DNS-based Redirection Example

In this section we walk through a simple example using DNS-based redirection for request redirection from uCDN to dCDN (as well as for request routing inside dCDN and uCDN). As noted in Section 2.1, DNS-based redirection has certain advantages over HTTP-based redirection (notably, it is transparent to the end-user) as well as some drawbacks (notably the client IP address is not visible to the request router).

As before, Operator A has to learn the set of requests that dCDN is willing or able to serve (e.g. which client IP address prefixes or geographic regions are part of the dCDN footprint). We assume Operator has and makes known to operator A some unique identifier that can be used for the construction of a distinguished CDN-Domain, as shown in more detail below. (This identifier strictly needs only to be unique within the scope of Operator A, but a globally unique

identifier, such as an AS number assigned to B, is one easy way to achieve that.) Also, Operator A obtains the NS records for Operator B's externally visible redirection servers. Also, as before, a distinguished CDN-Domain, such as op-b-acq.op-a.example, must be assigned for inter-CDN acquisition.

We assume DNS is configured in the following way:

- The CSP is configured to make Operator A the authoritative DNS server for cdn.csp.example (or to return a CNAME for cdn.csp.example for which operator A is the authoritative DNS server).

- When uCDN sees a request best served by dCDN, it returns CNAME and NS records for "b.cdn.csp.example", where "b" is the unique identifier assigned to Operator B. (It may, for example, be an AS number assigned to Operator B.)

- dCDN is configured so that a request for "b.cdn.csp.example" returns a delivery node in dCDN.

- uCDN is configured so that a request for "op-b-acq.op-a.example" returns a delivery node in uCDN.

Figure 5 depicts the exchange of DNS and HTTP requests. The main differences from Figure 3 are the lack of HTTP redirection and transparency to the end-user.

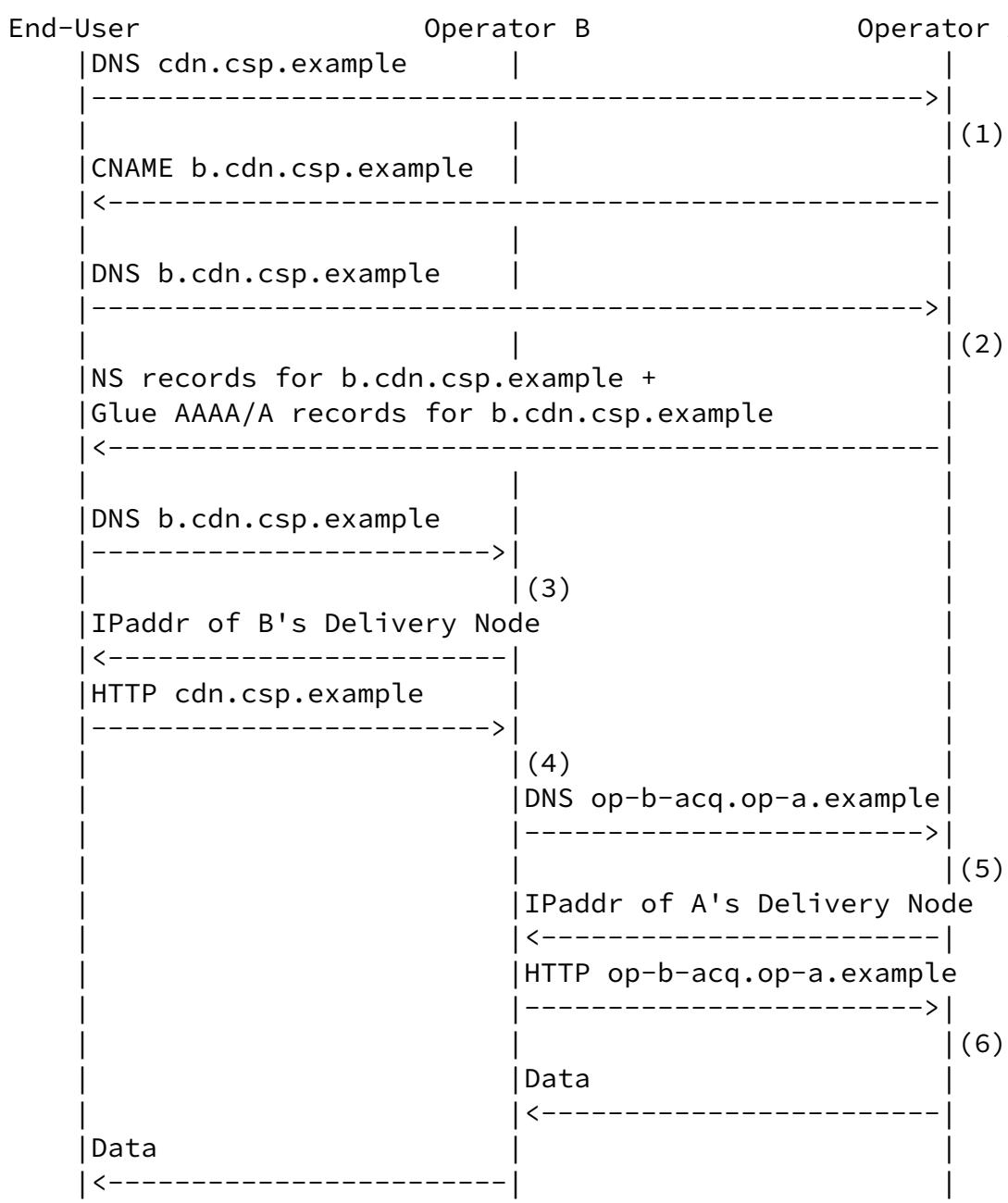

End-User
Operator B
Operator A
DNS cdn.csp.example
(1)
CNAME b.cdn.csp.example
DNS b.cdn.csp.example
(2)
NS records for b.cdn.csp.example +
Glue AAAA/A records for b.cdn.csp.example
DNS b.cdn.csp.example
(3)
IPaddr of B's Delivery Node
HTTP cdn.csp.example
(4)
DNS op-b-acq.op-a.example
(5)
IPaddr of A's Delivery Node
HTTP op-b-acq.op-a.example
(6)
Data
Data

Figure 5: Message Flow for DNS-based Redirection

The steps illustrated in the figure are as follows:

1. Request Router for Operator A processes the DNS request for CDN-Domain cdn.csp.example and recognizes that the end-user is best served by another CDN. (This may depend on the IP address of the user's local DNS resolver, or other information discussed below.) The Request Router returns a DNS CNAME response by "stacking" the distinguished identifier for Operator B onto the original CDN-Domain (e.g., b.cdn.csp.example).

2. The end-user sends a DNS query for the modified CDN-Domain (i.e. b.cdn.csp.example) to Operator A's DNS server. The Request Router for Operator A processes the DNS request and return a delegation to b.cdn.csp.example by sending an NS record plus glue AAAA/A records pointing to Operator B's DNS server. (This extra step is necessary since typical DNS implementation won't follow an NS record when it is sent together with a CNAME record, thereby necessitating a two-step approach).

3. The end-user sends a DNS query for the modified CDN-Domain (i.e., b.cdn.csp.example) to Operator B's DNS server, using the NS and AAAA/A records received in step 2. This causes B's Request Router to respond with a suitable delivery node.

4. The end-user requests the content from B's delivery node. The requested URL contains the name cdn.csp.example. (Note that the returned CNAME does not affect the URL.) At this point the delivery node has the correct IP address of the end-user and can do an HTTP 302 redirect if the redirections in steps 2 and 3 were incorrect. Otherwise B verifies that this CDN-Domain belongs to a known peer (so as to avoid being tricked into serving as an open proxy). It then does a DNS request for an "internal" CDN-Domain as agreed above (op-b-acq.op-a.example).

5. Operator A recognizes that the DNS request is from a peer CDN rather than an end-user (due to the internal CDN-Domain) and so returns the address of a delivery node in uCDN.

6. Operator A serves content to dCDN. Although not shown, it is at this point that Operator A processes the rest of the URL: it extracts information identifying the origin server, validates that this server has been registered, and determines the content provider that owns the origin server.

The advantages of this approach are that it is more transparent to the end-user and requires fewer round trips than HTTP-based redirection (in its worst case, i.e., when none of the needed DNS information is cached). A potential problem is that the upstream CDN depends on being able to learn the correct downstream CDN that serves the end-user from the client address in the DNS request. In standard DNS operation, uCDN will only obtain the address of the client's local DNS resolver (LDNS), which is not guaranteed to be in the same network (or geographic region) as the client. If not--e.g., the end-user uses a global DNS service--then the upstream CDN cannot determine the appropriate downstream CDN to serve the end-user. In this case, and assuming the uCDN is capable of detecting that situation, one option is for the upstream CDN to treat the end-user as it would any user not connected to a peer CDN. Another option is

for the upstream CDN to "fall back" to a pure HTTP-based redirection strategy in this case (i.e., use the first method). Note that this problem affects existing CDNs that rely on DNS to determine where to redirect client requests, but the consequences are arguably less serious for CDNI since the LDNS is likely in the same network as the dCDN serves.

As with the prior example, this example partially illustrates the various interfaces involved in CDNI. Operator A could learn dynamically from Operator B the set of prefixes or regions that B is willing and able to serve via the CDNI Footprint & Capabilities Advertisement interface. The distinguished name used for acquisition and the identifier for Operator B that is prepended to the CDN-Domain on redirection are examples of information elements that might also be conveyed by CDNI interfaces (or, alternatively, statically configured). As before, minimal metadata sufficient to obtain the content is carried "in-band" as part of the redirection process, and standard HTTP is used for inter-CDN acquisition. There is no explicit CDNI Logging interface discussed in this example.

3.4.1. Notes on using DNSSEC

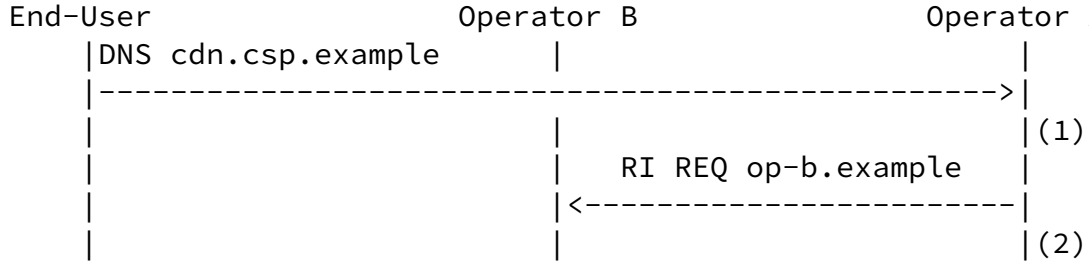
Although it is possible to use DNSSEC in combination with the Iterative DNS-based Redirection mechanism explained above, it is important to note that the uCDN might have to sign records on the fly, since the CNAME returned, and thus the signature provided, can potentially be different for each incoming query. Although there is nothing preventing a uCDN from performing such on-the-fly signing, this might be computationally expensive. In the case where the number of dCDNs, and thus the number of different CNAMEs to return, is relatively stable, an alternative solution would be for the uCDN to pre-generate signatures for all possible CNAMEs. For each incoming query the uCDN would then determine the appropriate CNAME and return it together with the associated pre-generated signature. Note: In the latter case maintaining the serial and signature of SOA might be an issue since technically it should change every time a different CNAME is used. However, since in practice direct SOA queries are relatively rare, a uCDN could defer incrementing the serial and resigning the SOA until it is queried and then do it on-the-fly.

Note also that the NS record and the glue AAAA/A records used in step 2 in the previous section should generally be identical to those of their authoritative zone managed by Operator B. Even if they differ, this will not make the DNS resolution process fail, but the client DNS server will prefer the authoritative data in its cache and use it for subsequent queries. Such inconsistency is a general operational issue of DNS, but it may be more important for this architecture

because the uCDN (operator A) would rely on the consistency to make the resulting redirection work as intended. In general, it is the administrator's responsibility to make them consistent.

3.5. Dynamic Footprint Discovery Example

There could be situations where being able to dynamically discover the set of requests that a given dCDN is willing and able to serve is beneficial. For example, a CDN might at one time be able to serve a certain set of client IP prefixes, but that set might change over time due to changes in the topology and routing policies of the IP network. The following example illustrates this capability. We have chosen the example of DNS-based redirection, but HTTP-based redirection could equally well use this approach.


End-User
Operator B
Operator A
DNS cdn.csp.example
(1)
RI REQ op-b.example
(2)

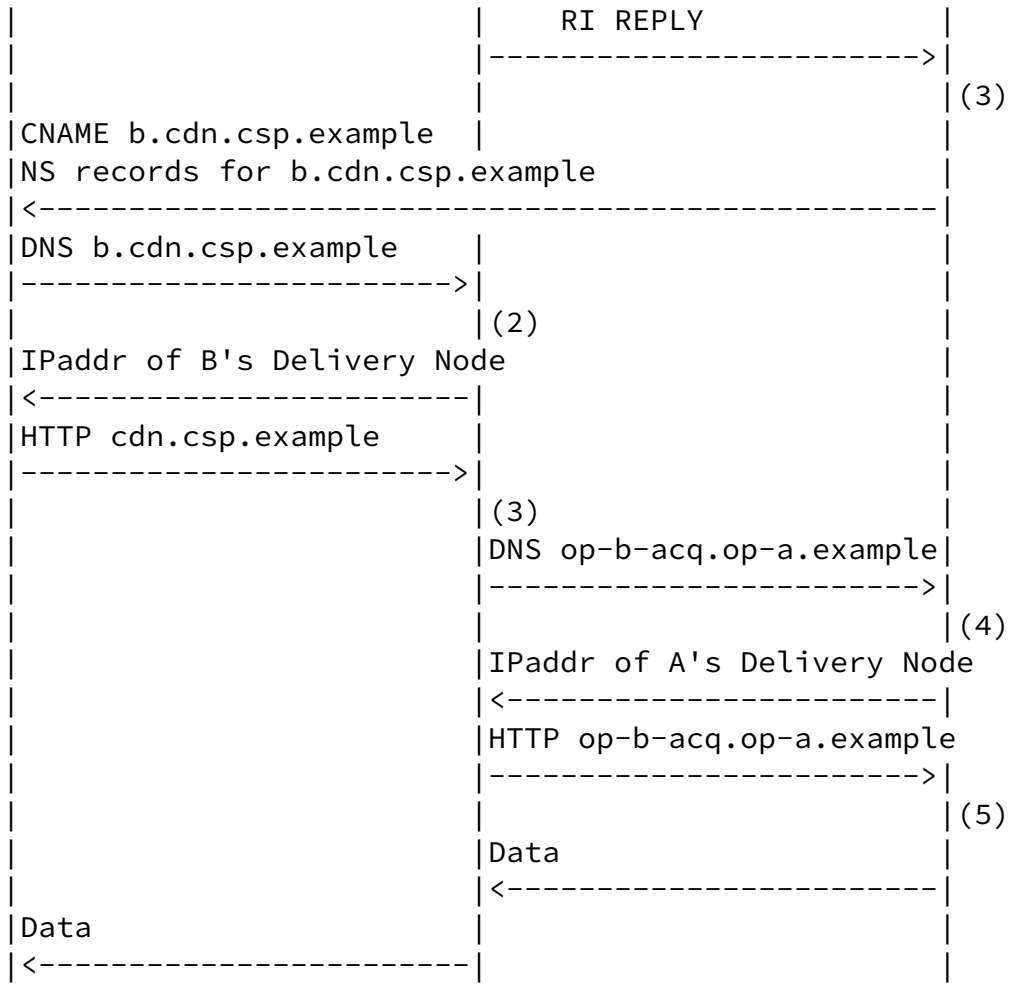


Figure 6: Message Flow for Dynamic Footprint Discovery

This example differs from the one in Figure 5 only in the addition of a FCI request (step 2) and corresponding response (step 3). The RI REQ could be a message such as "Can you serve clients from this IP Prefix?" or it could be "Provide the list of client IP prefixes you can currently serve". In either case the response might be cached by operator A to avoid repeatedly asking the same question. Alternatively, or in addition, Operator B may spontaneously advertise to Operator A information (or changes) on the set of requests it is willing and able to serve on behalf of operator A; in that case, Operator B may spontaneously issue RR/RI REPLY messages that are not in direct response to a corresponding RR/RI REQ message. (Note that

the issues of determining the client's subnet from DNS requests, as described above, are exactly the same here as in Section 3.4.)

Once Operator A obtains the RI response, it is now able to determine that Operator B's CDN is an appropriate dCDN for this request and therefore a valid candidate dCDN to consider in its Redirection decision. If that dCDN is selected, the redirection and serving of the request proceeds as before (i.e. in the absence of dynamic footprint discovery).

3.6. Content Removal Example

The following example illustrates how the CDNI Control interface may be used to achieve pre-positioning of an item of content in the dCDN. In this example, user requests for a particular content, and corresponding redirection of such requests from Operator A to Operator B CDN, may (or may not) have taken place earlier. Then, at some point in time, the uCDN (for example, in response to a corresponding Trigger from the Content Provider) uses the CI to request that content identified by a particular URL be removed from dCDN. The following diagram illustrates the operation.

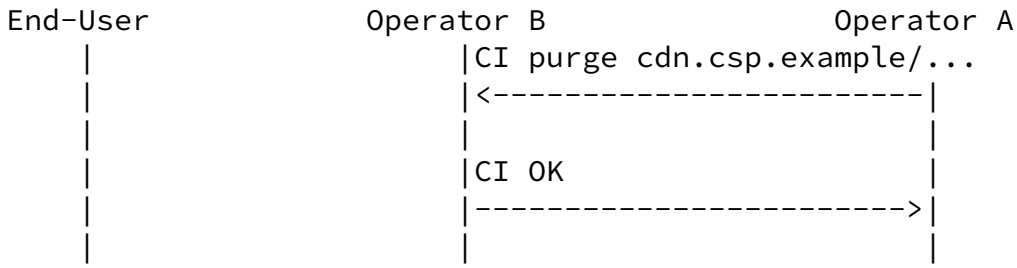


Figure 7: Message Flow for Content Removal

The CI is used to convey the request from uCDN to dCDN that some previously acquired content should be deleted. The URL in the request specifies which content to remove. This example corresponds to a DNS-based redirection scenario such as Section 3.4. If HTTP-based redirection had been used, the URL for removal would be of the form peer-a.op-b.example/cdn.csp.example/...

The dCDN is expected to confirm to the uCDN, as illustrated by the CI OK message, the completion of the removal of the targeted content from all the caches in dCDN.

3.7. Pre-Positioned Content Acquisition Example

The following example illustrates how the CI may be used to pre-position an item of content in the dCDN. In this example, Operator A uses the CDNI Metadata interface to request that content identified by a particular URL be pre-positioned into Operator B CDN.

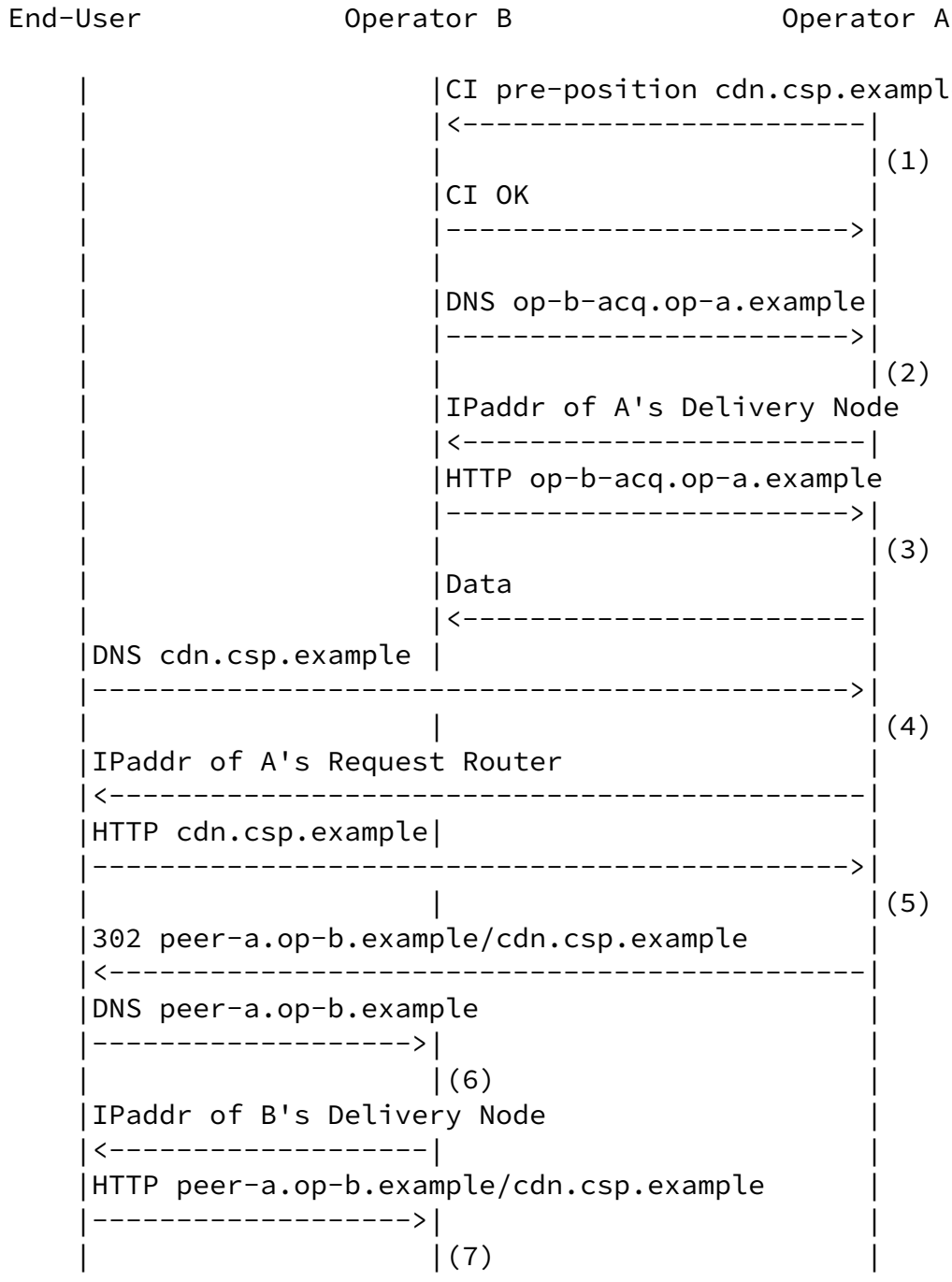


Figure 8: Message Flow for Content Pre-Positioning

The steps illustrated in the figure are as follows:

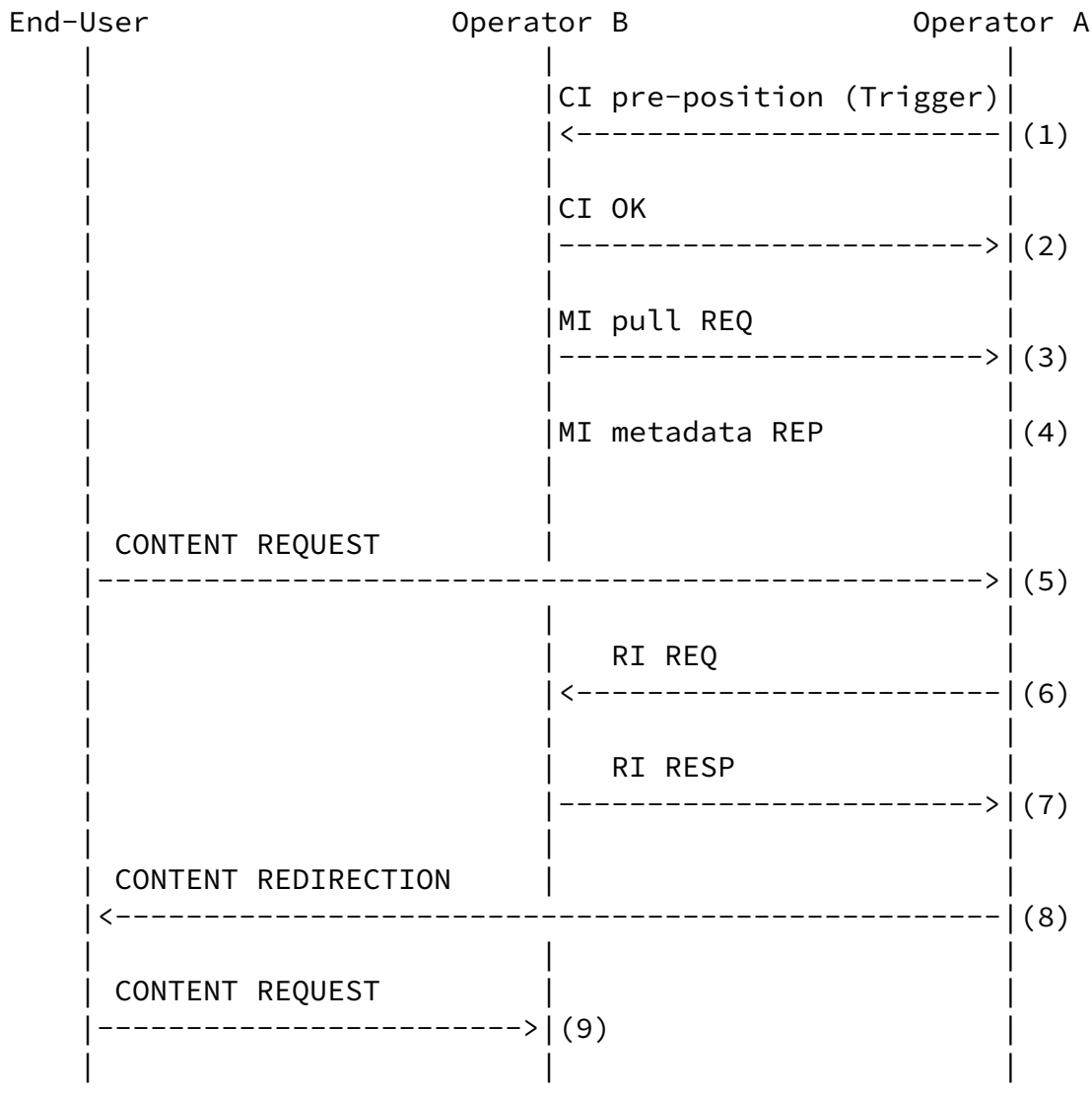
1. Operator A uses the CI to request that Operator B pre-positions a particular content item identified by its URL. Operator B responds by confirming that it is willing to perform this operation.

Steps 2 and 3 are exactly the same as steps 5 and 6 of Figure 3, only this time those steps happen as the result of the Pre-positioning request instead of as the result of a cache miss.

Steps 4, 5, 6, 7 are exactly the same as steps 1, 2, 3, 4 of Figure 3, only this time Operator B CDN can serve the end-user request without triggering dynamic content acquisition, since the content has been pre-positioned in dCDN. Note that, depending on dCDN operations and policies, the content pre-positioned in the dCDN may be pre-positioned to all, or a subset of, dCDN caches. In the latter case, intra-CDN dynamic content acquisition may take place inside the dCDN serving requests from caches on which the content has not been pre-positioning; however, such intra-CDN dynamic acquisition would not involve the uCDN.

3.8. Asynchronous CDNI Metadata Example

In this section we walk through a simple example illustrating a scenario of asynchronously exchanging CDNI metadata, where the downstream CDN obtains CDNI metadata for content ahead of a corresponding content request. The example that follows assumes that HTTP-based inter-CDN redirection and recursive CDNI request-routing are used, as in Section 3.3. However, Asynchronous exchange of CDNI Metadata is similarly applicable to DNS-based inter-CDN redirection and iterative request routing (in which cases the CDNI metadata may be used at slightly different processing stages of the message flows).


End-User
Operator B
Operator A
CI pre-position (Trigger)
(1)
CI OK
(2)
MI pull REQ
(3)
MI metadata REP
(4)
CONTENT REQUEST
(5)
RI REQ
(6)
RI RESP
(7)
CONTENT REDIRECTION
(8)
CONTENT REQUEST
(9)

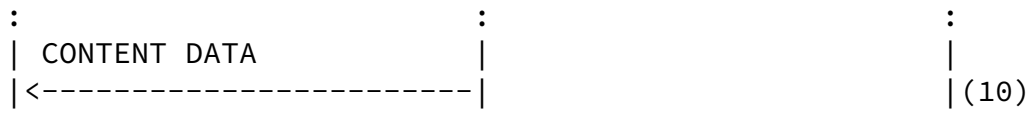


Figure 9: Message Flow for Asynchronous CDNI Metadata

The steps illustrated in the figure are as follows:

1. Operator A uses the CI to Trigger to signal the availability of CDNI metadata to Operator B.

2. Operator B acknowledges the receipt of this Trigger.

3. Operator B requests the latest metadata from Operator A using the MI.

4. Operator A replies with the requested metadata. This document does not constrain how the CDNI metadata information is actually

represented. For the purposes of this example, we assume that Operator A provides CDNI metadata to Operator B indicating that:

* this CDNI Metadata is applicable to any content referenced by some CDN-Domain.

* this CDNI metadata consists of a distribution policy requiring enforcement by the delivery node of a specific per-request authorization mechanism (e.g. URI signature or token validation).

5. A Content Request occurs as usual.

6. A CDNI Request Routing Redirection request (RI REQ) is issued by operator A CDN, as discussed in Section 3.3. Operator B's request router can access the CDNI Metadata that are relevant to the requested content and that have been pre-positioned as per Steps 1-4, which may or may not affect the response.

7. Operator B's request router issues a CDNI Request Routing Redirection response (RI RESP) as in Section 3.3.

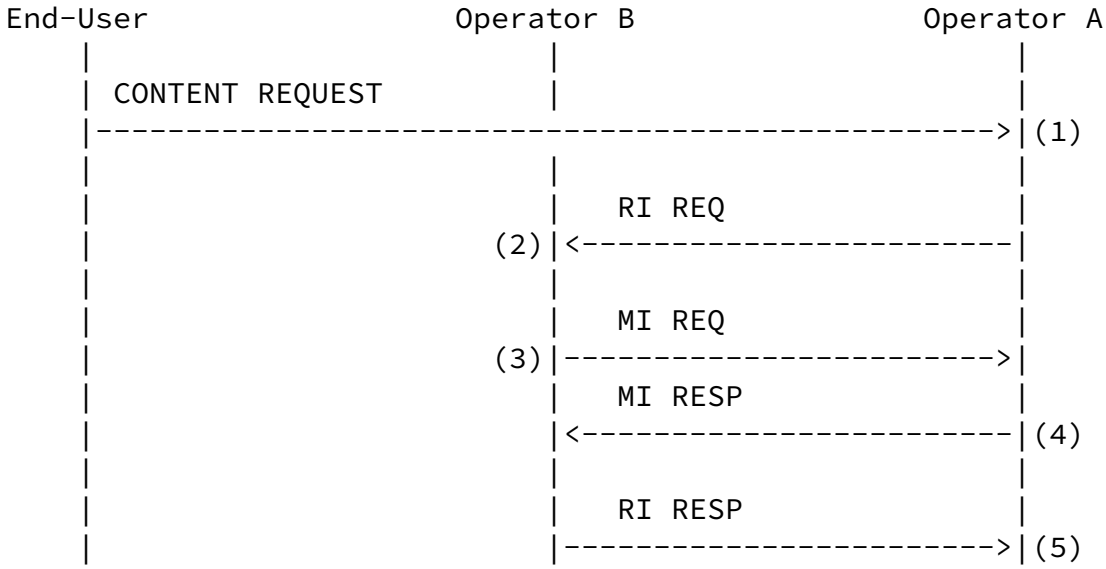
8. Operator B performs content redirection as discussed in Section 3.3.

9. On receipt of the Content Request by the end user, the delivery node detects that previously acquired CDNI metadata is applicable to the requested content. In accordance with the specific CDNI metadata of this example, the delivery node will invoke the appropriate per-request authorization mechanism, before serving the content. (Details of this authorization are not shown.)

10. Assuming successful per-request authorization, serving of Content Data (possibly preceded by inter-CDN acquisition) proceeds as in Section 3.3.

3.9. Synchronous CDNI Metadata Acquisition Example

In this section we walk through a simple example illustrating a scenario of Synchronous CDNI metadata acquisition, in which the downstream CDN obtains CDNI metadata for content at the time of handling a first request for the corresponding content. As in the preceding section, this example assumes that HTTP-based inter-CDN redirection and recursive CDNI request-routing are used (as in Section 3.3), but dynamic CDNI metadata acquisition is applicable to other variations of request routing.



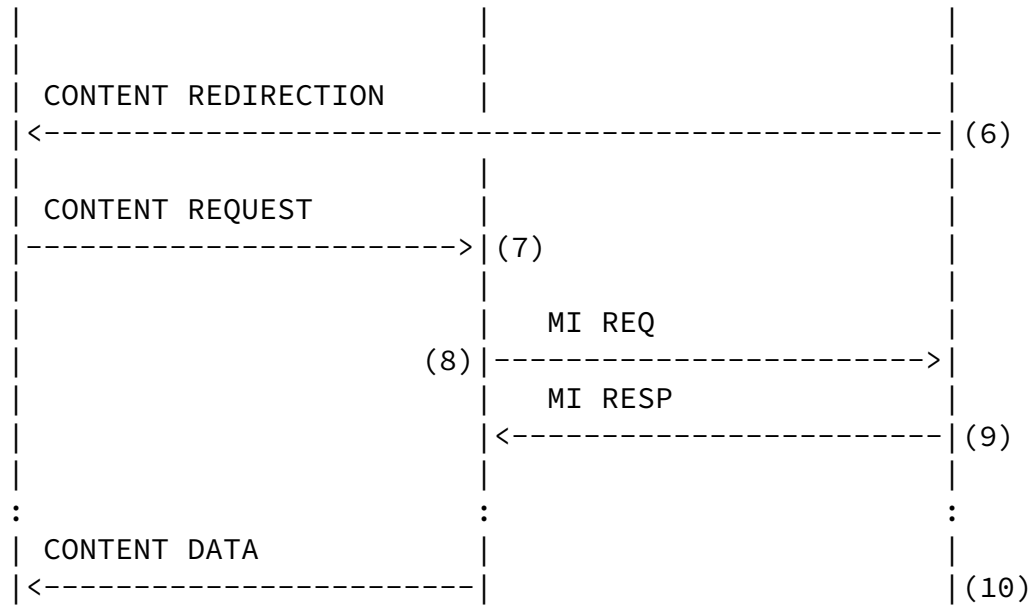


Figure 10: Message Flow for Synchronous CDNI Metadata Acquisition

The steps illustrated in the figure are as follows:

1. A Content Request arrives as normal.

2. An RI request occurs as in the prior example.

3. On receipt of the CDNI Request Routing Request, Operator B's CDN initiates Synchronous acquisition of CDNI Metadata that are needed for routing of the end-user request. We assume the URI for the a Metadata server is known ahead of time through some out-of-band means.

4. On receipt of a CDNI Metadata Request, Operator A's CDN responds, making the corresponding CDNI metadata information available to Operator B's CDN. This metadata is considered by operator B's CDN before responding to the Request Routing request. (In a simple case, the metadata could simply be an allow or deny response for this particular request.)

5. Response to the RI request as normal.

6. Redirection message is sent to the end user.

7. A delivery node of Operator B receives the end user request.

8. The delivery node Triggers dynamic acquisition of additional CDNI metadata that are needed to process the end-user content request. Note that there may exist cases where this step need not happen, for example because the metadata were already acquired previously.

9. Operator A's CDN responds to the CDNI Metadata Request and makes the corresponding CDNI metadata available to Operator B. This metadata influence how Operator B's CDN processes the end-user request.

10. Content is served (possibly preceded by inter-CDN acquisition) as in Section 3.3.

3.10. Content and Metadata Acquisition with Multiple Upstream CDNs

A single dCDN may receive end-user requests from multiple uCDNs. When a dCDN receives an end-user request, it must determine the identity of the uCDN from which it should acquire the requested content.

Ideally, the acquisition path of an end-user request will follow the redirection path of the request. The dCDN should acquire the content from the same uCDN which redirected the request.

Determining the acquisition path requires the dCDN to reconstruct the redirection path based on information in the end-user request. The method for reconstructing the redirection path differs based on the redirection approach: HTTP or DNS.

With HTTP-redirection, the rewritten URI should include sufficient information for the dCDN to directly or indirectly determine the uCDN when the end-user request is received. The HTTP-redirection approach can be further broken-down based on the how the URL is rewritten during redirection: HTTP-redirection with or without Site

Aggregation. HTTP-redirection with Site Aggregation hides the

identity of the original CSP. HTTP-redirection without Site Aggregation does not attempt to hide the identity of the original CSP. With both approaches, the rewritten URI includes enough information to identify the immediate neighbor uCDN.

With DNS-redirection, the dCDN receives the published URI (instead of a rewritten URI) and does not have sufficient information for the dCDN to identify the appropriate uCDN. The dCDN may narrow the set of viable uCDNs by examining the CDNI metadata from each to determine which uCDNs are hosting metadata for the requested content. If there is a single uCDN hosting metadata for the requested content, the dCDN can assume that the request redirection is coming from this uCDN and can acquire content from that uCDN. If there are multiple uCDNs hosting metadata for the requested content, the dCDN may be ready to trust any of these uCDNs to acquire the content (provided the uCDN is in a position to serve it). If the dCDN is not ready to trust any of these uCDNs, it needs to ensure via out of band arrangements that, for a given content, only a single uCDN will ever redirect requests to the dCDN.

Content acquisition may be preceded by content metadata acquisition. If possible, the acquisition path for metadata should also follow the redirection path. Additionally, we assume metadata is indexed based on rewritten URIs in the case of HTTP-redirection and is indexed based on published URIs in the case of DNS-redirection. Thus, the RI and the MI are tightly coupled in that the result of request routing (a rewritten URI pointing to the dCDN) serves as an input to metadata lookup. If the content metadata includes information for acquiring the content, then the MI is also tightly coupled with the acquisition interface in that the result of the metadata lookup (an acquisition URL likely hosted by the uCDN) should serve as input to the content acquisition.

4. Main Interfaces

Figure 1 illustrates the main interfaces that are in scope for the CDNI WG, along with several others. The detailed specifications of these interfaces are left to other documents, but see [RFC6707] and [I-D.ietf-cdni-requirements] for some discussion of the interfaces.

One interface that is not shown in Figure 1 is the interface between the user and the CSP. While for the purposes of CDNI that interface is out of scope, it is worth noting that it does exist and can provide useful functions, such as end-to-end performance monitoring and some forms of authentication and authorization.

There is also an important interface between the user and the Request Routing function of both uCDN and dCDN (shown as the "Request" Interface in Figure 1). As we saw in some of the preceding examples, that interface can be used as a way of passing metadata, such as the minimum information that is required for dCDN to obtain the content from uCDN.

In this section we will provide an overview of the functions performed by each of the CDNI interfaces and discuss how they fit into the overall solution. We also examine some of the design tradeoffs, and explore several cross-interface concerns. We begin with an examination of one such tradeoff that affects all the interfaces - the use of in-band or out-of-band communication.

4.1. In-Band versus Out-of-Band Interfaces

Before getting to the individual interfaces, we observe that there is a high-level design choice for each, involving the use of existing in-band communication channels versus defining new out-of-band interfaces.

It is possible that the information needed to carry out various interconnection functions can be communicated between peer CDNs using existing in-band protocols. The use of HTTP 302 redirect is an example of how certain aspects of request routing can be implemented in-band (embedded in URIs). Note that using existing in-band protocols does not imply that the CDNI interfaces are null; it is still necessary to establish the rules (conventions) by which such protocols are used to implement the various interface functions.

There are other opportunities for in-band communication beyond HTTP redirects. For example, many of the HTTP directives used by proxy servers can also be used by peer CDNs to inform each other of caching activity. Of these, one that is particularly relevant is the If-Modified-Since directive, which is used with the GET method to make it conditional: if the requested object has not been modified since the time specified in this field, a copy of the object will not be returned, and instead, a 304 (not modified) response will be returned.

4.2. Cross Interface Concerns

Although the CDNI interfaces are largely independent, there are a set of conventions practiced consistently across all interfaces. Most important among these is how resources are named, for exampmle, how the CDNI Metadata and Control interfaces identify the set of

resources to which a given directive applies, or the CDNI Logging

interface identifies the set of resources for which a summary record applies.

While in the limit the CDNI interfaces could explicitly identify every individual resource, in practice, they name resource aggregates (sets of URIs) that are to be treated in a similar way. For example, URI aggregates can be identified by a CDN-Domain (i.e., the FQDN at the beginning of a URI) or by a URI-Filter (i.e., a regular expression that matches a subset of URIs contained in some CDN-Doman). In other words, CDN-Domains and URI-Filters provide a uniform means to aggregate sets (and subsets) of URIs for the purpose of defining the scope for some operation in one of the CDNI interfaces.

4.3. Request Routing Interfaces

The Request Routing interface comprises two parts: the Asynchronous interface used by a dCDN to advertize footprint and capabilities (denoted FCI) to a uCDN, allowing the uCDN to decide whether to redirect particular user requests to that dCDN; and the Synchronous interface used by the uCDN to redirect a user request to the dCDN (denoted RI). (These are somewhat analogous to the operations of routing and forwarding in IP.)

As illustrated in Section 3, the RI part of request routing may be implemented in part by DNS and HTTP. Naming conventions may be established by which CDN peers communicate whether a request should be routed or content served.

We also note that RI plays a key role in enabling recursive redirection, as illustrated in Section 3.3. It enables the user to be redirected to the correct delivery node in dCDN with only a single redirection step (as seen by the user). This may be particularly valuable as the chain of interconnected CDNs increases beyond two CDNs. For further discussion on the RI, see [I-D.ietf-cdni-redirection].

In support of these redirection requests, it is necessary for CDN peers to exchange additional information with each other, and this is

the role of the FCI part of request routing. Depending on the method(s) supported, this might include:

- The operator's unique id (operator-id) or distinguished CDN-Domain (operator-domain);
- NS records for the operator's set of externally visible request routers;

- The set of requests the dCDN operator is prepared to serve (e.g. a set of client IP prefixes or geographic regions that may be served by dCDN).
- Additional capabilities of the dCDN, such as its ability to support different CDNI Metadata requests.

Note that the set of requests that dCDN is willing to serve could in some cases be relatively static (e.g., a set of IP prefixes) which could be exchanged off-line, or might even be negotiated as part of a peering agreement. However, it may also be more dynamic, in which case the exchange supported by FCI would be be helpful. A further discussion of the Footprint & Capability Advertisement interface can be found in [I-D.ietf-cdni-footprint-capabilities-semantics].

4.4. CDNI Logging Interface

It is necessary for the upstream CDN to have visibility into the delivery of content that it redirected to a downstream CDN. This allows the upstream CDN to properly bill its customers for multiple deliveries of content cached by the downstream CDN, as well as to report accurate traffic statistics to those content providers. This is one role of the LI.

Other operational data that may be relevant to CDNI can also be exchanged by the LI. For example, dCDN may report the amount of content it has acquired from uCDN, and how much cache storage has been consumed by content cached on behalf of uCDN.

Traffic logs are easily exchanged off-line. For example, the following traffic log is a small deviation from the Apache log file format, where entries include the following fields:

- Domain - the full domain name of the origin server
- IP address - the IP address of the client making the request
- End time - the ending time of the transfer
- Time zone - any time zone modifier for the end time
- Method - the transfer command itself (e.g., GET, POST, HEAD)
- URL - the requested URL
- Version - the protocol version, such as HTTP/1.0
- Response - a numeric response code indicating transfer result

- Bytes Sent - the number of bytes in the body sent to the client
- Request ID - a unique identifier for this transfer
- User agent - the user agent, if supplied
- Duration - the duration of the transfer in milliseconds
- Cached Bytes - the number of body bytes served from the cache
- Referer - the referrer string from the client, if supplied

Of these, only the Domain field is indirect in the downstream CDN--it is set to the CDN-Domain used by the upstream CDN rather than the actual origin server. This field could then used to filter traffic log entries so only those entries matching the upstream CDN are reported to the corresponding operator. Further discussion of the LI can be found in [I-D.ietf-cdni-logging].

One open question is who does the filtering. One option is that the downstream CDN filters its own logs, and passes the relevant records directly to each upstream peer. This requires that the downstream CDN knows the set of CDN-Domains that belong to each upstream peer. If this information is already exchanged between peers as part of another interface, then direct peer-to-peer reporting is

straightforward. If it is not available, and operators do not wish to advertise the set of CDN-Domains they serve to their peers, then the second option is for each CDN to send both its non-local traffic records and the set of CDN-Domains it serves to an independent third-party (i.e., a CDN Exchange), which subsequently filters, merges, and distributes traffic records on behalf of each participating CDN operator.

A second open question is how timely traffic information should be. For example, in addition to offline traffic logs, accurate real-time traffic monitoring might also be useful, but such information requires that the downstream CDN inform the upstream CDN each time it serves upstream content from its cache. The downstream CDN can do this, for example, by sending a conditional HTTP GET request (If-Modified-Since) to the upstream CDN each time it receives an HTTP GET request from one of its end-users. This allows the upstream CDN to record that a request has been issued for the purpose of real-time traffic monitoring. The upstream CDN can also use this information to validate the traffic logs received later from the downstream CDN.

There is obviously a tradeoff between accuracy of such monitoring and the overhead of the downstream CDN having to go back to the upstream CDN for every request.

Another design tradeoff in the LI is the degree of aggregation or summarization of data. One situation that lends itself to summarization is the delivery of HTTP adaptive streaming (HAS), since the large number of individual chunk requests potentially results in large volumes of logging information. This case is discussed below, but other forms of aggregation may also be useful. For example, there may be situations where bulk metrics such as bytes delivered per hour may suffice rather than the detailed per-request logs outlined above. It seems likely that a range of granularities of logging will be needed along with ways to specify the type and degree of aggregation required.

4.5. CDNI Control Interface

The CDNI Control interface is initially used to bootstrap the other interfaces. As a simple example, it could be used to provide the address of the logging server in dCDN to uCDN in order to bootstrap the CDNI Logging interface. It may also be used, for example, to

establish security associations for the other interfaces.

The other role the CI plays is to allow the uCDN to pre-position, revalidate, or purge metadata and content on a dCDN. These operations, sometimes collectively called the Trigger interface, are discussed further in [I-D.ietf-cdni-control-triggers].

4.6. CDNI Metadata Interface

The role of the CDNI Metadata interface is to enable CDNI distribution metadata to be conveyed to the downstream CDN by the upstream CDN. Such metadata includes geo-blocking restrictions, availability windows, access control policies, and so on. It may also include information to facilitate acquisition of content by dCDN (e.g., alternate sources for the content, authorization information needed to acquire the content from the source). For a full discussion of the CDNI Metadata Interface, see [I-D.ietf-cdni-metadata]

Some distribution metadata may be partially emulated using in-band mechanisms. For example, in case of any geo-blocking restrictions or availability windows, the upstream CDN can elect to redirect a request to the downstream CDN only if that CDN's advertised delivery footprint is acceptable for the requested URL. Similarly, the request could be forwarded only if the current time is within the availability window. However, such approaches typically come with shortcomings such as inability to prevent from replay outside the time window or inability to make use of a downstream CDN that covers a broader footprint than the geo-blocking restrictions.

Similarly, some forms of access control may also be performed on a per-request basis using HTTP directives. For example, being able to respond to a conditional GET request gives the upstream CDN an opportunity to influence how the downstream CDN delivers its content. Minimally, the upstream CDN can invalidate (purge) content previously cached by the downstream CDN.

All of these in-band techniques serve to illustrate that uCDNs have the option of enforcing some of their access control policies themselves (at the expense of increased inter-CDN signaling load), rather than delegating enforcement to dCDNs using the MI. As a

consequence, the MI could provide a means for the uCDN to express its desire to retain enforcement for itself. For example, this might be done by including a "check with me" flag in the metadata associated with certain content. The realization of such in-band techniques over the various inter-CDN acquisition protocols (e.g., HTTP) requires further investigation and may require small extensions or semantic changes to the acquisition protocol.

4.7. HTTP Adaptive Streaming Concerns

We consider HTTP Adaptive Streaming (HAS) and the impact it has on the CDNI interfaces because large objects (e.g., videos) are broken into a sequence of small, independent chunks. For each of the following, a more thorough discussion, including an overview of the tradeoffs involved in alternative designs, can be found in RFC 6983.

First, with respect to Content Acquisition and File Management, which are out-of-scope for the CDNI interfaces but nontheless relevant to the overall operation, we assume no additional measures are required to deal with large numbers of chunks. This means that the dCDN is not explicitly made aware of any relationship between different chunks and the dCDN handles each chunk as if it were an individual and independent content item. The result is that content acquisition between uCDN and dCDN also happens on a per-chunk basis. This approach is in line with the recommendations made in RFC 6983, which also identifies potential improvements in this area that might be considered in the future.

Second, with respect to Request Routing, we note that HAS manifest files have the potential to interfere with request routing since manifest files contain URLs pointing to the location of content chunks. To make sure that a manifest file does not hinder CDNI request routing and does not place excessive load on CDNI resources, the use of manifest files could either be limited to those containing relative URLs or the uCDN could modify the URLs in the manifest. Our approach for dealing with these issues is twofold. As a mandatory requirement, CDNs should be able to handle unmodified manifest files

containing either relative or absolute URLs. To limit the number of redirects, and thus the load placed on the CDNI interfaces, as an optional feature uCDNs can use the information obtained through the CNDI Request Routing Redirection interface to modify the URLs in the

manifest file. Since the modification of the manifest file is an optional uCDN-internal process, this does not require any standardization effort beyond being able to communicate chunk locations in the CDNI Request Routing Redirection interface.

Third, with respect to the CDNI Logging interface, there are several potential issues, including the large number of individual chunk requests potentially resulting in large volumes of logging information, and the desire to correlate logging information for chunk requests that correspond to the same HAS session. For the initial CDNI specification, our approach is to expect participating CDNs to support per-chunk logging (e.g. logging each chunk request as if it were an independent content request) over the CDNI Logging interface. Optionally, the LI may include a Content Collection IDentifier (CCID) and/or a Session IDentifier (SID) as part of the logging fields, thereby facilitating correlation of per-chunk logs into per-session logs for applications benefiting from such session level information (e.g. session-based analytics). This approach is in line with the recommendations made in RFC 6983, which also identifies potential improvements in this area that might be considered in the future.

Fourth, with respect to the CDNI Control interface, and in particular purging HAS chunks from a given CDN, our approach is to expect each CDN supports per-chunk content purge (e.g. purging of chunks as if they were individual content items). Optionally, a CDN may support content purge on the basis of a "Purge IDentifier (Purge-ID)" allowing the removal of all chunks related to a given Content Collection with a single reference. It is possible that this Purge-ID could be merged with the CCID discussed above for HAS Logging, or alternatively, they may remain distinct.

4.8. URI Rewriting

When using HTTP redirection, content URIs may be rewritten when redirection takes place within an uCDN, from an uCDN to a dCDN, and within the dCDN. In the case of cascaded CDNs, content URIs may be rewritten at every CDN hop (e.g., between the uCDN and the dCDN acting as the transit CDN, and between the transit CDN and the dCDN serving the request. The content URI used between any uCDN/dCDN pair becomes a common handle that can be referred to without ambiguity by both CDNs in all their inter-CDN communications. This handle allows the uCDN and dCDN to correlate information exchanged using other CDNI

interfaces in both the downstream direction (e.g., when using the MI) and the upstream direction (e.g., when using the LI).

Consider the simple case of a single uCDN/dCDN pair using HTTP redirection. We introduce the following terminology for content URIs to simplify the discussion:

"u-URI" represents a content URI in a request presented to the uCDN;

"ud-URI" is a content URI acting as the common handle across uCDN and dCDN for requests redirected by the uCDN to a specific dCDN;

"d-URI" represents a content URI in a request made within the delegate dCDN.

In our simple pair-wise example, the "ud-URI" effectively becomes the handle that the uCDN/dCDN pair use to correlate all CDNI information. In particular, for a given pair of CDNs executing the HTTP redirection, the uCDN needs to map the u-URI to the ud-URI handle for all MI message exchanges, while the dCDN needs to map the d-URI to the ud-URI handle for all LI message exchanges.

In the case of cascaded CDNs, the transit CDN will rewrite the content URI when redirecting to the dCDN, thereby establishing a new handle between the transit CDN and the dCDN, that is different from the handle between the uCDN and transit CDN. It is the responsibility of the transit CDN to manage its mapping across handles so the right handle for all pairs of CDNs is always used in its CDNI communication.

In summary, all CDNI interfaces between a given pair of CDNs need to always use the "ud-URI" handle for that specific CDN pair as their content URI reference.

5. Deployment Models

In this section we describe a number of possible deployment models that may be achieved using the CDNI interfaces described above. We note that these models are by no means exhaustive, and that many other models may be possible.

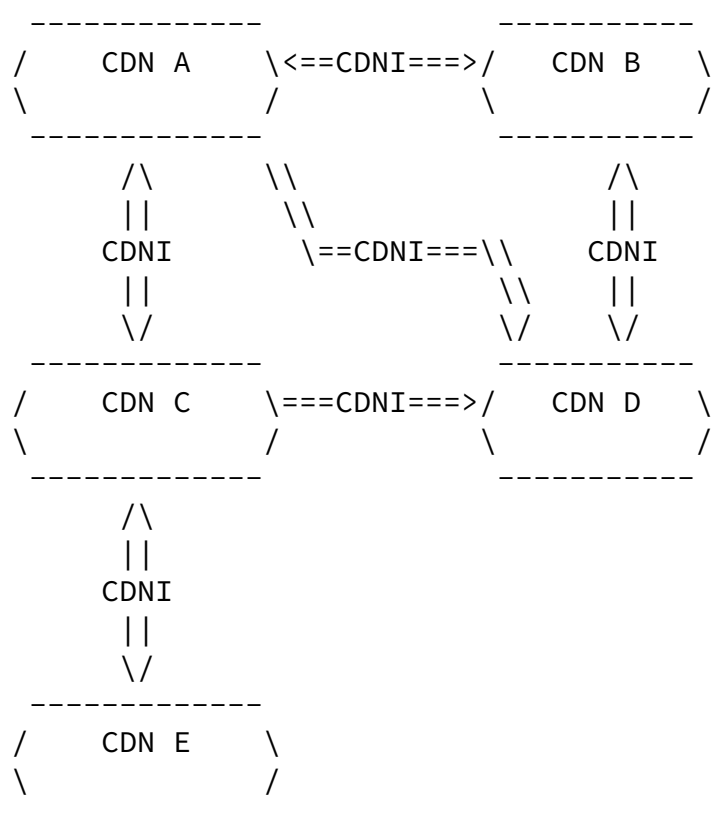
Although the reference model of Figure 1 shows all CDN functions on each side of the CDNI interface, deployments can rely on entities that are involved in any subset of these functions, and therefore only support the relevant subset of CDNI interfaces. As already noted in Section 3, effective CDNI deployments can be built without

necessarily implementing all the interfaces. Some examples of such deployments are shown below.

Note that, while we refer to upstream and downstream CDNs, this distinction applies to specific content items and transactions. That is, a given CDN may be upstream for some transactions and downstream for others, depending on many factors such as location of the requesting client and the particular piece of content requested.

5.1. Meshed CDNs

Although the reference model illustrated in Figure 1 shows a unidirectional CDN interconnection with a single uCDN and a single dCDN, any arbitrary CDNI meshing can be built from this, such as the example meshing illustrated in Figure 11. (Support for arbitrary meshing may or may not be in the initial scope for the working group, but the model allows for it.)



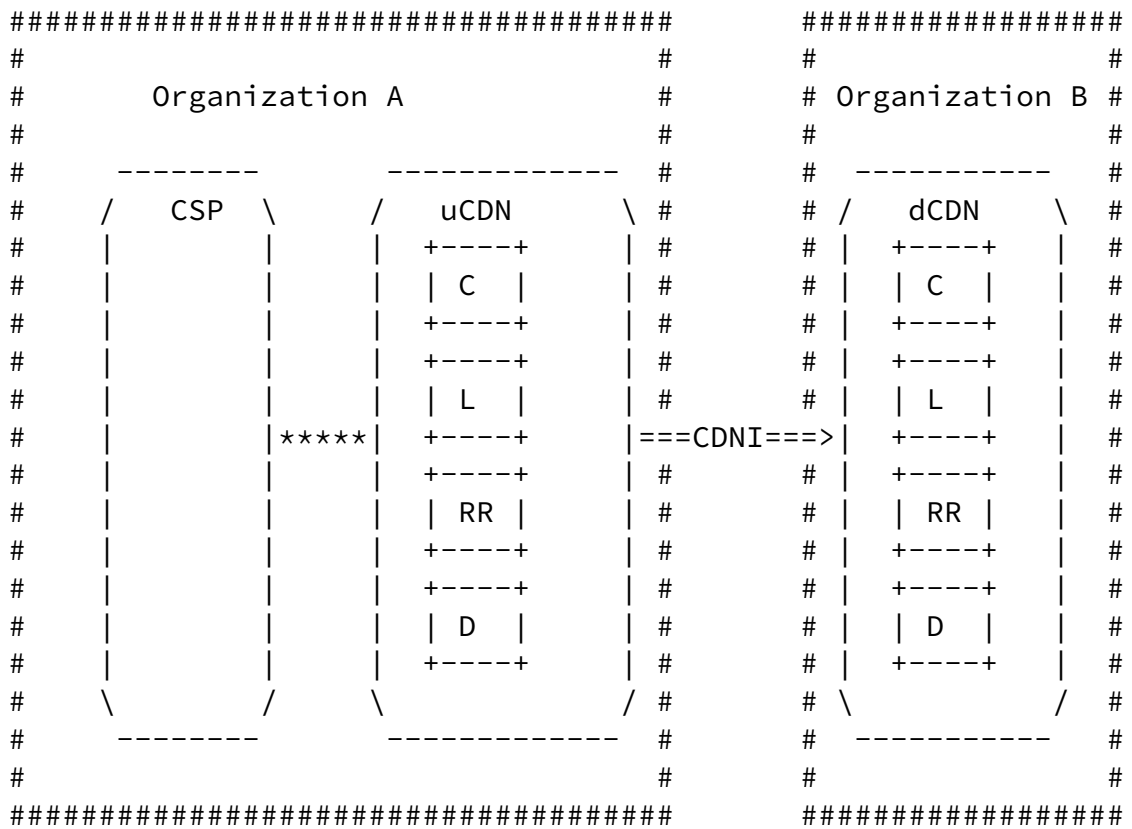
===> CDNI interfaces, with right-hand side CDN acting as dCDN to left-hand side CDN

<==> CDNI interfaces, with right-hand side CDN acting as dCDN to left-hand side CDN and with left-hand side CDN acting as dCDN to right-hand side CDN

Figure 11: CDNI Deployment Model: CDN Meshing Example

5.2. CSP combined with CDN

Note that our terminology refers to functional roles and not economic or business roles. That is, a given organization may be operating as both a CSP and a fully fledged uCDN when we consider the functions performed, as illustrated in Figure 12.



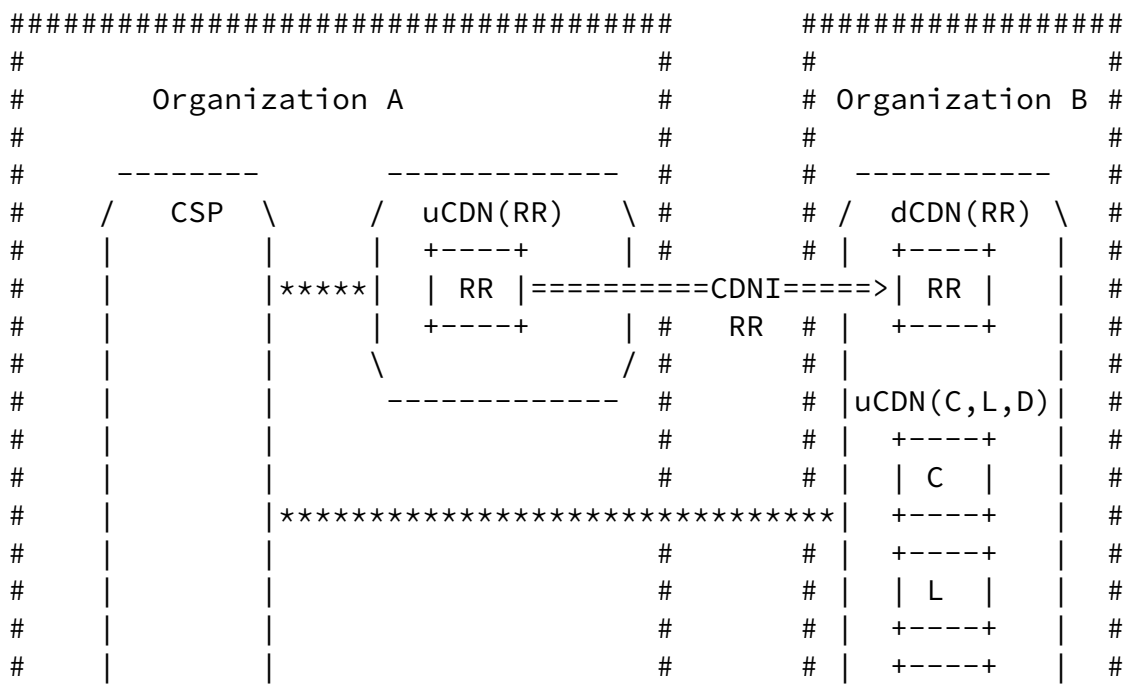
C Control component of the CDN
L Logging component of the CDN
RR Request Routing component of the CDN
D Distribution component of the CDN

Figure 12: CDNI Deployment Model: Organization combining CSP & uCDN

5.3. CSP using CDNI Request Routing Interface

As another example, a content provider organization may choose to run its own request routing function as a way to select among multiple candidate CDN providers; In this case the content provider may be modeled as the combination of a CSP and of a special, restricted case of a CDN. In that case, as illustrated in Figure 13, the CDNI Request Routing interfaces can be used between the restricted CDN

operated by the content provider Organization and the CDN operated by the full CDN organization acting as a dCDN in the request routing control plane. Interfaces outside the scope of the CDNI work can be used between the CSP functional entities of the content provider organization and the CDN operated by the full CDN organization acting as a uCDN) in the CDNI control planes other than the request routing plane (i.e. Control, Distribution, Logging).



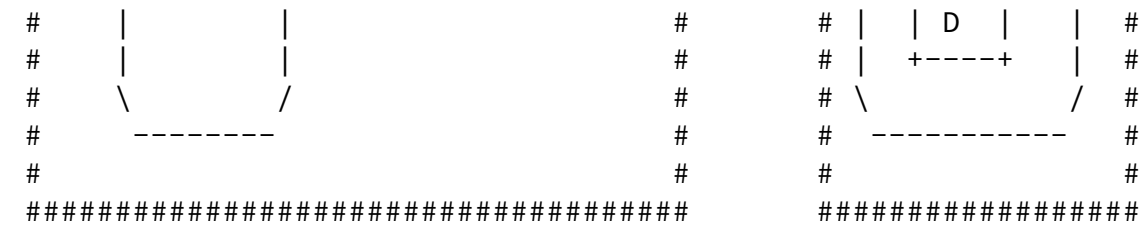


Figure 13: CDNI Deployment Model: Organization combining CSP and partial CDN

5.4. CDN Federations and CDN Exchanges

There are two additional concepts related to, but distinct from CDN Interconnection. The first is CDN Federation. Our view is that CDNI is the more general concept, involving two or more CDNs serving content to each other's users, while federation implies a multi-lateral interconnection arrangement, but other CDN interconnection agreements are also possible (e.g., symmetric bilateral, asymmetric bilateral). An important conclusion is that CDNI technology should

not presume (or bake in) a particular interconnection agreement, but should instead be general enough to permit alternative interconnection arrangements to evolve.

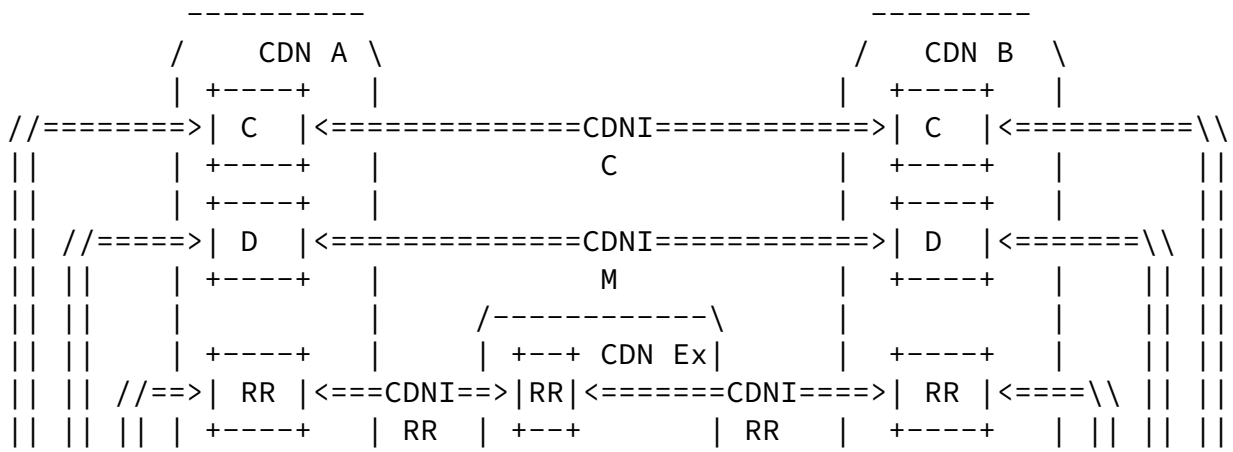
The second concept often used in the context of CDN Federation is CDN Exchange--a third party broker or exchange that is used to facilitate a CDN federation. Our view is that a CDN exchange offers valuable machinery to scale the number of CDN operators involved in a multi-lateral (federated) agreement, but that this machinery is built on top of the core CDNI interconnection mechanisms. For example, as illustrated in Figure 14, the exchange might aggregate and redistribute information about each CDN footprint and capacity, as well as collect, filter, and redistribute traffic logs that each participant needs for interconnection settlement, but inter-CDN request routing, inter-CDN content distribution (including inter-CDN acquisition) and inter-CDN control which fundamentally involve a direct interaction between an upstream CDN and a downstream CDN--operate exactly as in a pair-wise peering arrangement. Turning to Figure 14, we observe that in this example:

- each CDN supports a direct CDNI Control interface to every other CDN

- each CDN supports a direct CDNI Metadata interface to every other CDN

- each CDN supports a CDNI Logging interface with the CDN Exchange

- each CDN supports both a CDNI Request Routing interface with the CDN Exchange (for aggregation and redistribution of dynamic CDN footprint discovery information) and a direct RI to every other CDN (for actual request redirection).



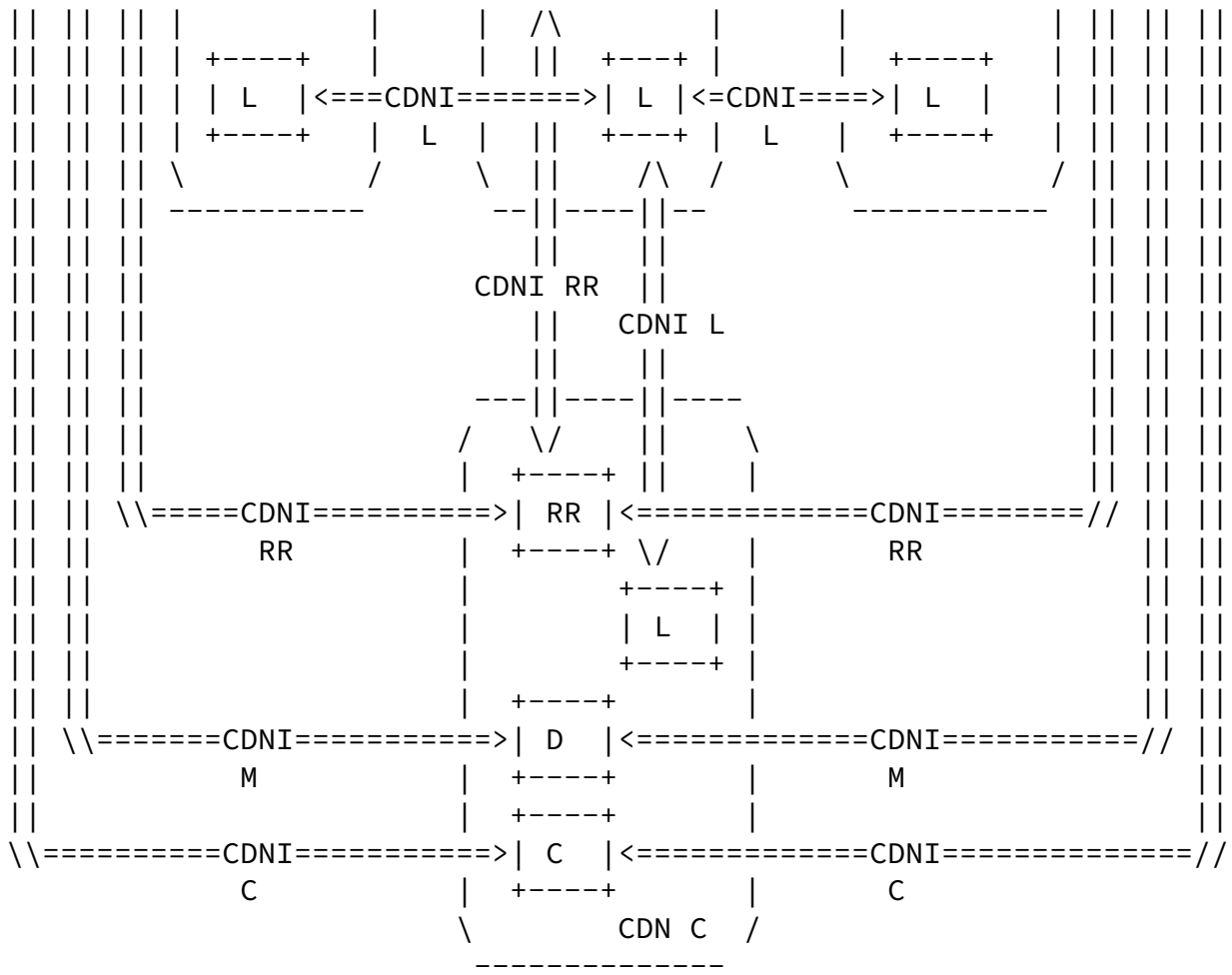


Figure 14: CDNI Deployment Model: CDN Exchange

Note that a CDN exchange may alternatively support a different set of functionality (e.g. Logging only, or Logging and full request

routing, or all the functionality of a CDN including content distribution). All these options are expected to be allowed by the IETF CDNI specifications.

6. Trust Model

There are a number of trust issues that need to be addressed by a CDNI solution. Many of them are in fact similar or identical to those in a simple CDN without interconnection. In a standard CDN environment (without CDNI), the CSP places a degree of trust in a single CDN operator to perform many functions. The CDN is trusted to deliver content with appropriate quality of experience for the end user. The CSP trusts the CDN operator not to corrupt or modify the content. The CSP often relies on the CDN operator to provide reliable accounting information regarding the volume of delivered content. The CSP may also trust the CDN operator to perform actions such as timely invalidation of content and restriction of access to content based on certain criteria such as location of the user and time of day, and to enforce per-request authorization performed by the CSP using techniques such as URI signing.

A CSP also places trust in the CDN not to distribute any information that is confidential to the CSP (e.g., how popular a given piece of content is) or confidential to the end user (e.g., which content has been watched by which user).

A CSP does not necessarily have to place complete trust in a CDN. A CSP will in some cases take steps to protect its content from improper distribution by a CDN, e.g. by encrypting it and distributing keys in some out of band way. A CSP also depends on monitoring (possibly by third parties) and reporting to verify that the CDN has performed adequately. A CSP may use techniques such as client-based metering to verify that accounting information provided by the CDN is reliable. HTTP conditional requests may be used to provide the CSP with some checks on CDN operation. In other words, while a CSP may trust a CDN to perform some functions in the short term, the CSP is able in most cases to verify whether these actions have been performed correctly and to take action (such as moving the content to a different CDN) if the CDN does not live up to expectations.

The main trust issue raised by CDNI is that it introduces transitive trust. A CDN that has a direct relationship with a CSP can now "outsource" the delivery of content to another (downstream) CDN. That CDN may in term outsource delivery to yet another downstream CDN, and so on.

The top level CDN in such a chain of delegation is responsible for ensuring that the requirements of the CSP are met. Failure to do so is presumably just as serious as in the traditional single CDN case. Hence, an upstream CDN is essentially trusting a downstream CDN to perform functions on its behalf in just the same way as a CSP trusts a single CDN. Monitoring and reporting can similarly be used to verify that the downstream CDN has performed appropriately. However, the introduction of multiple CDNs in the path between CSP and end user complicates the picture. For example, third party monitoring of CDN performance (or other aspects of operation, such as timely invalidation) might be able to identify the fact that a problem occurred somewhere in the chain but not point to the particular CDN at fault.

In summary, we assume that an upstream CDN will invest a certain amount of trust in a downstream CDN, but that it will verify that the downstream CDN is performing correctly, and take corrective action (including potentially breaking off its relationship with that CDN) if behavior is not correct. We do not expect that the trust relationship between a CSP and its "top level" CDN will differ significantly from that found today in single CDN situations. However, it does appear that more sophisticated tools and techniques for monitoring CDN performance and behavior will be required to enable the identification of the CDN at fault in a particular delivery chain.

We expect that the detailed designs for the specific interfaces for CDNI will need to take the transitive trust issues into account. For example, explicit confirmation that some action (such as content removal) has taken place in a downstream CDN may help to mitigate some issues of transitive trust.

7. IANA Considerations

This memo includes no request to IANA.

8. Privacy Considerations

In general, a CDN has the opportunity to collect detailed information about the behavior of end-users e.g. by logging which files are being downloaded. While the concept of interconnected CDNs as described in this document doesn't necessarily allow any given CDN to gather more information on any specific user, it potentially facilitates sharing of this data by a CDN with more parties. As an example, the purpose of the CDNI Logging Interface is to allow a dCDN to share some of its log records with a uCDN, both for billing purposes as well as for sharing traffic statistics with the Content Provider on which behalf the content was delivered. The fact that the CDNI Interfaces provide

mechanisms for sharing such potentially sensitive user data, shows that it is necessary to include in these interface appropriate privacy and confidentiality mechanisms. The definition of such mechanisms is dealt with in the respective CDN interface documents.

9. Security Considerations

While there are a variety of security issues introduced by a single CDN, we are concerned here specifically with the additional issues that arise when CDNs are interconnected. For example, when a single CDN has the ability to distribute content on behalf of a CSP, there may be concerns that such content could be distributed to parties who are not authorized to receive it, and there are mechanisms to deal with such concerns. Our focus in this section is on how CDN interconnection introduces new security issues not found in the single CDN case. For a more detailed analysis of the security requirements of CDNI, see section 9 of [I-D.ietf-cdni-requirements].

Many of the security issues that arise in CDNI are related to the transitivity of trust (or lack thereof) described in Section 6. As noted above, the design of the various interfaces for CDNI must take account of the additional risks posed by the fact that a CDN with whom a CSP has no direct relationship is now potentially distributing content for that CSP. The mechanisms used to mitigate these risks may be similar to those used in the single CDN case, but their suitability in this more complex environment must be validated.

CDNs today offer a variety of means to control access to content, such as time-of-day restrictions, geo-blocking, and URI signing. These mechanisms must continue to function in CDNI environments, and this consideration is likely to affect the design of certain CDNI interfaces (e.g. metadata, request routing). For more information on URI signing in CDNI, see [I-D.leung-cdni-uri-signing].

Just as with a single CDN, each peer CDN must ensure that it is not used as an "open proxy" to deliver content on behalf of a malicious CSP. Whereas a single CDN typically addresses this problem by having CSPs explicitly register content (or origin servers) that are to be served, simply propagating this information to peer downstream CDNs may be problematic because it reveals more information than the upstream CDN is willing to specify. (To this end, the content acquisition step in the earlier examples force the dCDN to retrieve

content from the uCDN rather than go directly to the origin server.)

There are several approaches to this problem. One is for the uCDN to encode a signed token generated from a shared secret in each URL routed to a dCDN, and for the dCDN to validate the request based on this token. Another one is to have each upstream CDN advertise the

set of CDN-Domains they serve, where the downstream CDN checks each request against this set before caching and delivering the associated object. Although straightforward, this approach requires operators to reveal additional information, which may or may not be an issue.

9.1. Security of CDNI Interfaces

It is noted in [I-D.ietf-cdni-requirements] that all CDNI interfaces must be able to operate securely over insecure IP networks. Since it is expected that the CDNI interfaces will be implemented using existing application protocols such as HTTP or XMPP, we also expect that the security mechanisms available to those protocols may be used by the CDNI interfaces. Details of how these interfaces are secured will be specified in the relevant interface documents.

9.2. Digital Rights Management

Issues of digital rights management (DRM, also sometimes called digital restrictions management) is often employed for content distributed via CDNs. In general, DRM relies on the CDN to distribute encrypted content, with decryption keys distributed to users by some other means (e.g. directly from the CSP to the end user.) For this reason, DRM is considered out of scope [RFC6707] and does not introduce additional security issues for CDNI.

10. Contributors

The following individuals contributed to this document:

- Matt Caulfield
- Francois le Faucheur
- Aaron Falk

- David Ferguson
- John Hartman
- Ben Niven-Jenkins
- Kent Leung

11. Acknowledgements

The authors would like to thank Huw Jones and Jinmei Tatuya for their helpful input to this document. In addition, the authors would like to thank Stephen Farrell, Ted Lemon and Alissa Cooper for their reviews, which have helped to improve this document.

12. Informative References

[I-D.ietf-cdni-control-triggers]
Murray, R. and B. Niven-Jenkins, "CDNI Control Interface / Triggers", draft-ietf-cdni-control-triggers-02 (work in progress), December 2013.

[I-D.ietf-cdni-footprint-capabilities-semantics]
Seedorf, J., Peterson, J., Previdi, S., Brandenburg, R., and K. Ma, "CDNI Request Routing: Footprint and Capabilities Semantics", draft-ietf-cdni-footprint-capabilities-semantics-02 (work in progress), February 2014.

[I-D.ietf-cdni-logging]
Faucheur, F., Bertrand, G., Oprescu, I., and R. Peterkofsky, "CDNI Logging Interface", draft-ietf-cdni-logging-11 (work in progress), March 2014.

[I-D.ietf-cdni-metadata]
Niven-Jenkins, B., Murray, R., Watson, G., Caulfield, M., Leung, K., and K. Ma, "CDN Interconnect Metadata", draft-ietf-cdni-metadata-06 (work in progress), February 2014.

[I-D.ietf-cdni-redirection]

Niven-Jenkins, B. and R. Brandenburg, "Request Routing Redirection Interface for CDN Interconnection", draft-ietf-cdni-redirection-02 (work in progress), April 2014.

[I-D.ietf-cdni-requirements]
Leung, K. and Y. Lee, "Content Distribution Network Interconnection (CDNI) Requirements", draft-ietf-cdni-requirements-17 (work in progress), January 2014.

[I-D.leung-cdni-uri-signing]
Leung, K., Faucheur, F., Downey, B., Brandenburg, R., and S. Leibrand, "URI Signing for CDN Interconnection (CDNI)", draft-leung-cdni-uri-signing-05 (work in progress), March 2014.

[RFC3466] Day, M., Cain, B., Tomlinson, G., and P. Rzewski, "A Model for Content Internetworking (CDI)", RFC 3466, February 2003.

[RFC6707] Niven-Jenkins, B., Le Faucheur, F., and N. Bitar, "Content Distribution Network Interconnection (CDNI) Problem Statement", RFC 6707, September 2012.

[RFC6770] Bertrand, G., Stephan, E., Burbridge, T., Eardley, P., Ma, K., and G. Watson, "Use Cases for Content Delivery Network Interconnection", RFC 6770, November 2012.

[RFC6983] van Brandenburg, R., van Deventer, O., Le Faucheur, F., and K. Leung, "Models for HTTP-Adaptive-Streaming-Aware Content Distribution Network Interconnection (CDNI)", RFC 6983, July 2013.

Authors' Addresses

Larry Peterson
Akamai Technologies, Inc.
8 Cambridge Center
Cambridge, MA 02142
USA

Email: lapeters@akamai.com

Bruce Davie
VMware, Inc.
3401 Hillview Ave.
Palo Alto, CA 94304
USA

Email: bdavie@vmware.com

Ray van Brandenburg (editor)
TNO
Brassersplein 2
Delft 2612CT
the Netherlands

Phone: +31-88-866-7000
Email: ray.vanbrandenburg@tno.nl