Internet Engineering Task Force
Internet-Draft
Intended status: Informational
Expires: October 21, 2011

L. Peterson
J. Hartman
Verivue Inc.
M. Pilarski
Orange Labs/WUT

April 19, 2011

A Simple Approach to CDN Interconnection
draft-peterson-cdni-strawman-00

Abstract

This document presents a simple strawman for CDN interconnection, and uses the strawman as a basis for articulating a set of design principles and exploring (parts of) the CDNI design space. Our intent is to spur discussion about what information needs to be exchanged between CDN peers, which is a prerequisite for crafting interfaces and protocols to communicate that information.

Status of this Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79. This document may not be modified, and derivative works of it may not be created, and it may not be published except as an Internet-Draft.

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 October 21, 2011.

Copyright Notice

Table of Contents

1. Introduction

This document presents a simple strawman for CDN interconnection, and uses the strawman as a basis for articulating a set of design principles and exploring (parts of) the CDNI design space. In terms of the four CDNI-related interfaces outlined in [I-D.jenkins-cdni-problem-statement], this document first describes several approaches to Request Routing, and then based on that foundation, discusses possible Logging, Control, and CDNI Metadata interfaces.

Our general strategy is to explore low barrier-to-entry CDN interconnection. This strategy has three implications. First, we take a "best effort" approach by including only essential functionality. We expect enhanced inter-CDN control features to be added incrementally if and when they prove necessary in practice. Second, we pursue an approach in which two cooperating CDNs directly interconnect with no third-party mediation or involvement. Third, we exploit "in-band" signaling that leverages existing protocols (e.g., DNS and HTTP), rather than define new "out-of-band" control interfaces. This is not to say that advanced control features, third-party mediation, or out-of-band interfaces will never be required, but rather, that the best way to avoid unnecessary complexity is to fully explore the limits of what can be done with simple mechanisms.

An implicit question asked throughout this document is: "What information needs to be shared between CDNI peers?" Until we can answer this question, details about the precise interfaces and protocols needed to exchange that information are premature.

This document is partially informed by ongoing CDN interconnection trials, an early instance of which is reported in [I-D.bertrand-cdni-experiments].

2. Use Cases

Several use cases for CDN interconnection drive our discussion. Examples include:

Delivery Termination: A global CDN might peer with one or more regional CDNs, with the latter terminating content delivery to locally connected end-users.

Pair-wise Peers: Two peer CDNs, each serving a distinct set of content providers and end-users, might each agree to serve the other's content to its local users.

International CDN: A set of semi-autonomous national affiliates belonging to a common multi-national operator might cooperate to form a single international CDN.

On-Net/Off-Net Delivery: An operator serving content to end-users directly connected to its network might also serve those same users when they are connected off-network

Managed/Unmanaged Networks: An operator that offers separate managed (IPTV services including Catch-up and VoD) and unmanaged (broadband) CDNs can serve content from one to the other.

Each of these usage scenarios involves an upstream CDN that serves content on behalf of a content provider (CP), and a downstream CDN that delivers content to a local end-user. Some of the scenarios are asymmetric (e.g., Delivery Termination) with content flowing in only one direction, and some are symmetric (e.g., Pair-wise Peers) with content flowing in both directions.

In general, there might also exist one or more transit CDNs that sit between the upstream and downstream CDNs. While we believe the interaction between any pair of CDNs (e.g., upstream/transit, transit/transit, transit/downstream) is exactly as in the simple upstream/downstream case, we do not specifically consider transit CDNs in this document.

3. Autonomy Requirement

Any approach to interconnecting CDNs must preserve administrative boundaries between autonomous organizations. To this end, information hiding is the key design principle, by which we mean minimizing the information autonomous CDNs must share with (advertise to) each other. Note that we are not necessarily concerned about one CDN being able to infer something about another (e.g., by interpreting URLs), but rather, we are focused on minimizing the information one CDN must explicitly advertise to another to facilitate interconnection.

Minimizing information sharing argues for each independent CDN not having to explicitly advertise their internal caching hierarchy and not having to reveal how they interpret the rest of the URL (after the host name). This second point implies peer CDNs access origin servers indirectly through the CDN that serves the CP; they do not

directly contact the origin servers themselves.

To fully understand the ramifications, recall the distinction between the upstream CDN (the CDN that has a relationship with the CP) and the downstream CDN (the CDN that delivers the content to an end-user). In this context, there are two issues: (1) processing the original request flow from upstream CDN to downstream CDN so as to select the best delivery node for the end-user, and (2) processing the request flow from downstream CDN to upstream CDN to fetch the content in response to a cache miss in the downstream CDN. Consider each, in turn.

First, in order to know the target downstream CDN to which to redirect a request (assuming each CDN has more than one peer), the upstream CDN must be aware of the set of end-users (e.g., IP address blocks) a given downstream CDN is able to serve. Care must be taken in configuring the two CDNs so that the downstream CDN does not inadvertently redirect a request back to the upstream CDN, creating an infinite loop.

Second, if we assume the rest of the URL contains information that only the upstream CDN can use to identify the origin server, then once a delivery node is selected for a given end-user by the downstream CDN, if that node does not have the requested content in its cache, it requests the content from the upstream CDN rather than contacting the origin server directly. In general, the delivery node might request the missing content from some place higher in its caching hierarchy, but eventually some cache in the downstream CDN will need to pull the data from some cache in the upstream CDN.

3.1. Limitations

One consequence of maintaining strong boundaries between CDNs is that there will necessarily be limits on how much visibility and control the upstream CDN has into the actual delivery of content from the downstream CDN to its end-users. Considers two such limits.

First, while it might be reasonable for the downstream CDN to inform the upstream CDN each time it receives a request for cached content that originated with the upstream CDN (Section 6 describes one such mechanism), it is problematic for the upstream CDN to learn (in real-time) the actual number of bytes transferred out of a downstream cache since not all requests result in a complete file download. The upstream CDN can learn this information off-line, as it processes traffic logs received from the downstream peer, but real-time delivery monitoring will be limited.

Second, while a given CDN may offer its own content providers

elaborate control over how their content is delivered to its directly connected end-users (e.g., fine-grain access control and service differentiation), attempting to compose such polices across CDN boundaries significantly raises the barrier to interconnection. Instead, we start with the simplifying assumption that peers will establish coarse-grained agreements in which the downstream CDN treats the upstream CDN as a single content source (without distinguishing among different upstream content providers) and the upstream CDN treats the downstream CDN as a single content sink (without distinguishing among different classes of end-users). This is not to say that content providers have no control over how their content is delivered across interconnected CDNs. Just as in a single-CDN scenario, the origin server is free to implement its own access control mechanisms, the assumption being that an end-user first acquires the necessary authorization directly from the content provider, and then downloads the content itself from a CDN.

Clearly, neither of these two examples is absolute. They are intended to illustrate that placing too many requirements on CDN interconnection has the potential to make the problem (and resulting mechanisms) prohibitively complex. Again, we take a "best effort" approach, adding requirements and mechanisms only after they prove essential in practice.

4. Available Mechanisms

This section reviews several mechanisms that can be used to interconnect CDNs. Our approach is to leverage existing protocols in a way that allows two CDNs to directly interconnect, without requiring third-party mediation.

4.1. Redirection

Request redirection is a building block for the request routing function of CDNI. There are two main mechanisms for redirecting a request. The first leverages the DNS name resolution process and the second uses in-protocol redirection mechanisms such as the HTTP 302 redirection response.

There is a third technique--transparent caching--in which the downstream CDN transparently intercepts content requests targeted at the upstream CDN. We do not discuss transparent caching any further in this report.

4.1.1. 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., Google Public DNS).

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 modify the path component of the URL being accessed by the client. 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 delivery nodes 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 timeliness 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.

4.1.2. HTTP Redirection

HTTP redirection makes use of the "302" redirection response of the HTTP protocol. 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 302 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; and (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.

The disadvantages of HTTP redirection are (1) it is visible to the application, so it requires application support and may affect the application behavior (e.g., web browsers will not send cookies if the URL changes to a different domain); (2) HTTP is a heavy-weight protocol layered on TCP so it has relatively high overhead; and (3) the results of HTTP redirection are not cached so that all redirections must go through to the server.

4.1.3. Assumptions

We make three assumptions regarding request routing. First, the language used in this document presumes a unified request routing service that handles both DNS and HTTP requests. In practice, DNS-based redirection and HTTP-based redirection might be handled by separate mechanisms in a given CDN. Some CDNs might support one but not necessarily both mechanisms, and our proposal takes this into account.

Second, we assume the request routing service is bootstrapped through some external mechanism, such as IP anycast. Thus, when we say "the Request Router responds to a DNS query for cdn.cp.com," we assume cp.com's DNS servers return an anycast address for a set of Request Routers. If IP anycast is not available, then we assume some other mechanism is used to pick a specific Request Router bound to that name.

Third, we assume the operator's LDNS is located within the same operator network as the end-user (i.e., both are contained in the same IP address block), and hence, the upstream CDN will be able to correctly identify the downstream CDN that serves the end-user based upon the client in DNS requests. Unfortunately, this is not necessarily true for end-users that use a global DNS service. We will revisit this situation in later sections.

4.2. Proxy Directives

In as much as a CDN can be viewed as a distributed proxy, 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 seems particularly relevant is the If-Modify-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.

Peer CDNs can use the If-Modify-Since directive to communicate two bits of information to each other. First, the downstream CDN can send a conditional GET to the upstream CDN to signal that cached content is being requested. This allows the upstream CDN to record that fact for real-time monitoring and reporting purposes. Second, the upstream CDN can respond with an HTTP error code that indicates the content is no longer available. This allows the upstream CDN to effectively purge content from the downstream CDN.

In addition, by including the X-Forwarded-For HTTP header along with the If-Modified-Since directive, the downstream CDN can report the end-user's IP address to the upstream CDN. This is useful for monitoring, and potentially, for access control.

4.3. Alternative Model - CDN Exchange

The approach outlined in this report involves direct DNS and HTTP interaction between a pair of CDNs. An intermediate "CDN Exchange" (or Broker) is not required. Direct interaction reduces the barrier to CDN interconnection, since DNS and HTTP are already well-established protocols. The problem is reduced to defining rules for how URLs are rewritten, as described in the next section.

Although not required, there remains a question of whether or not a CDN Exchange adds some value in certain circumstances. Consider two potential arguments.

One is that having all requests pass through an explicit exchange point provides an opportunity for a neutral third-party to record the transaction for billing and monitoring. We believe such a mechanism is not necessary, and instead, our approach requires the downstream CDN to periodically send the upstream CDN a traffic (billing) log off-line, coupled with the use of HTTP directives like If-Modify-Since to support monitoring. Note that even with the use of such HTTP directives, neither the upstream CDN nor a CDN exchange can know exactly how many bytes the downstream CDN delivers from its cache unless it puts itself on the data path, which is not practical. Since the upstream CDN must trust the traffic logs it receives from the downstream CDN anyway (augmented with any request signal it receives), a CDN exchange provides no value in terms of brokering requests.

A second potential argument for a CDN exchange is that it reduces the amount of information peer CDNs much advertise to each other. Such information can be given to an intermediate party, but not advertised

to a peer. This may be particularly relevant to resolving billing logs. We return to this issue in a later section, after describing and evaluating candidate methods for direct (broker-less) CDN interconnection.

Of course, a CDN Exchange might also have non-technical value, for example, by providing a common peering agreement that lowers the barrier-to-entry from a business perspective. This document focuses on technical mechanisms (i.e., interfaces and protocols), and hence, considers CDN Exchanges through that lens.

5. Request Routing

This section presents three direct (in-band) methods to Request Routing, including a discussion of their relative merits. To simplify the exposition, we use the term "CDN-domain" to refer to the host name (a FQDN) at the beginning of each URL, and we assume Operator A provides an upstream CDN that serves content on behalf of a content provider with CDN-domain cdn.cp.com and Operator B provides a downstream CDN that delivers content to an end-user who makes a request for URL

http://cdn.cp.com/...rest of url...

Throughout the examples, we truncate the "/...rest of url..." from the URL to simplify the presentation. This simplification is consistent with the underlying design principle that the portion of the URL that follows the CDN-domain is opaque to peer CDNs. Only the upstream knows how to interpret the rest of the URL, and hence, is able to contact the origin server. The downstream CDN never contacts the origin server directly. (In practice, the full URL might include the actual origin server--e.g., http://cdn.cp.com/video.cp.com/video1.mp4--but not necessarily. The origin server might be implicit in the CDN-domain, or it might be identified in the URL by an opaque identifier that is later mapped into a URL for the origin server.) Also, while the examples use a customer-branded CDN-domain for each content provider (e.g., cdn.cp.com), this is not a requirement. It could also be the case that all content providers being served by a given operator share an operator-branded CDN-domain (e.g., cdn.operator.net).

To be clear about this, and the other example URLs presented in this section, we summarize our examples as follows:

- cdn.cp.com - An example CP-branded CDN-domain. We assume the upstream CDN provides the authoritative DNS name server for this CDN-domain. Note that this CDN-domain can be viewed as opaque for

all the methods. We present it as an example only to emphasize whether or not the URL seen by the end-user is within the scope of the original content provider (e.g., cp.com).

- peer.op-a.net, peer.op-b.net - Example distinguished operator-branded CDN-domains. We use the string "peer" to signify--by convention--that this CDN-domain is used as part of the CDN interconnection, distinguishing it from other CDN-domains the operator might use when publishing its own content via its CDN. We sometimes call such a distinguished CDN-domain an operator-domain.

- a.cdn.cp.com, b.cdn.cp.com - Examples of a modified CDN-domain that encodes a unique id for an operator. We do not mean the strings "a" and "b" literally; they could be integers. The only requirement is that a unique and well-known identifier is assigned to each participating CDN. In practice, using A and B's autonomous system (AS) number would be a good solution.

- dca.cdn.cp.com - An example of a modified CDN-domain that signifies a request is coming from a "Designated CDN Authority". We mean the string "dca" literally. There must be global agreement among operators that this particular designator is used.

5.1. First Method

The first method assumes Operators A and B use the distinguished CDN-domains peer.op-a.com and peer.op-b.com, respectively. We say these CDN-domains are distinguished because their use is limited to the interconnection mechanism; they are never embedded in URLs that end-users request. They are also unique to each CDN. Figure 1 depicts the exchange of DNS and HTTP requests. The following explains each exchange, keyed to the highlighted numbers.

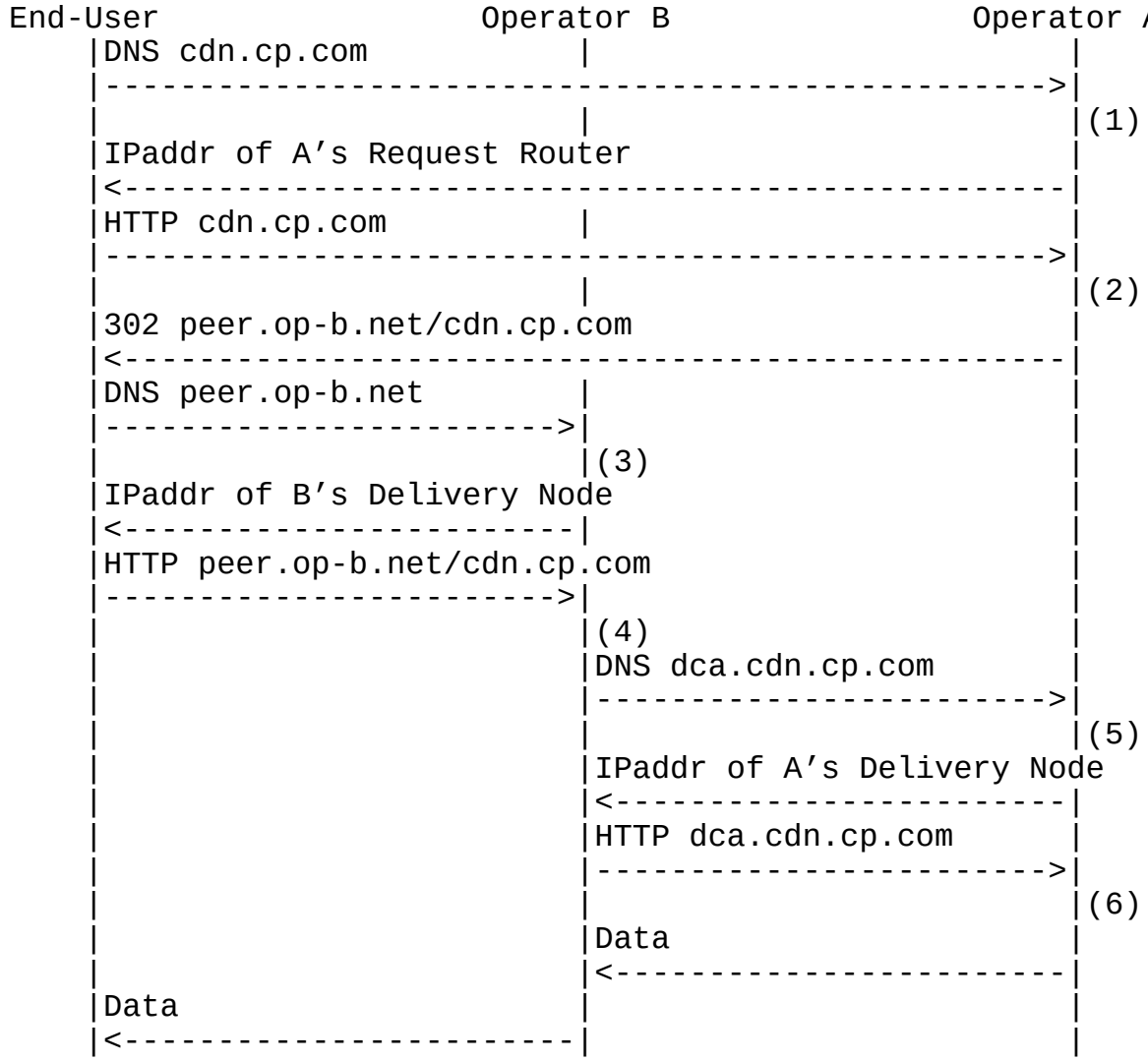

Figure 1: Request Trace for Method One

1. A Request Router for Operator A processes the DNS request for its customer based on CDN-domain cdn.cp.com and recognizes that the end-user is best served by another operator's CDN, but in order to redirect the end-user to that CDN, it must first get the end-user to issue an HTTP request. Thus, it returns the IP address of a Request Router in Operator A.

2. A Request Router for Operator A processes the HTTP request and again 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 (e.g., peer.op-b.net) on the front of the original URL.

3. The end-user does a DNS lookup using Operator B's distinguished CDN-domain (e.g., peer.op-b.net). B's Request Router returns a suitable delivery node.

4. The end-user requests the content from B's delivery node, potentially resulting in a cache miss. B sees its distinguished CDN-domain and so "pops it off" the URL, revealing the original CDN-domain cdn.cp.com. Operator 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 constructed by augmenting the original CDN-domain with a distinguished token (e.g., dca.cdn.cp.com).

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.

6. Operator A serves content for the requested CDN-domain. 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 main advantage of this design is that it is simple: each CDN need only know a single 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, 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). This makes our first example most appropriate for a heterogeneous set of CDN peers (i.e., CDNs utilizing different vendor technology).

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. It is important that any redirected URL be in the same domain (e.g., cp.com) if the browser is expected to send any cookies associated with that domain.

5.1.1. Configuration Summary

Operators must exchange the following information to peer with each other:

- The operator's distinguished CDN-domain (operator-domain); and
- Set of IP prefixes for which the operator is prepared to deliver to end-users.

Operators must perform the following URL conversions:

- When a Request Router in an upstream sees an end-user IP address best served by a downstream peer, it converts "cdn-domain" to "operator-domain/cdn-domain" (for the selected peer's operator-domain) and returns an HTTP 302 redirect for the new URL.
- When a delivery node in a downstream sees a URL of the form "operator-domain/cdn-domain" (for its operator-domain), it verifies that "cdn-domain" is served by a known CDN peer, and if so, converts it to "dca.cdn-domain" and issues an HTTP request for that new URL.

DNS must be configured in the following way:

- The content provider must be configured to make the operator that serves "cdn-domain" the authoritative DNS server for that name.
- An operator that serves "cdn-domain" must be configured so that a request for "dca.cdn-domain" returns a delivery node.
- An operator with CDN-domain "operator-domain" must be configured so that a request for "operator-domain/cdn-domain" returns a delivery node.

5.2. Second Method

The second method addresses the cookie issue by assigning a unique identifier to each CDN--we use "a" and "b" in our example--and including this identifier as a token in a CDN-domain belonging to the original content provider (e.g., a.cdn.cp.com). Note that "a" and "b" can be integers; the only requirement is that a unique and well-known identifier is assigned to each participating CDN. In practice, using A and B's autonomous system (AS) number would be a good solution.

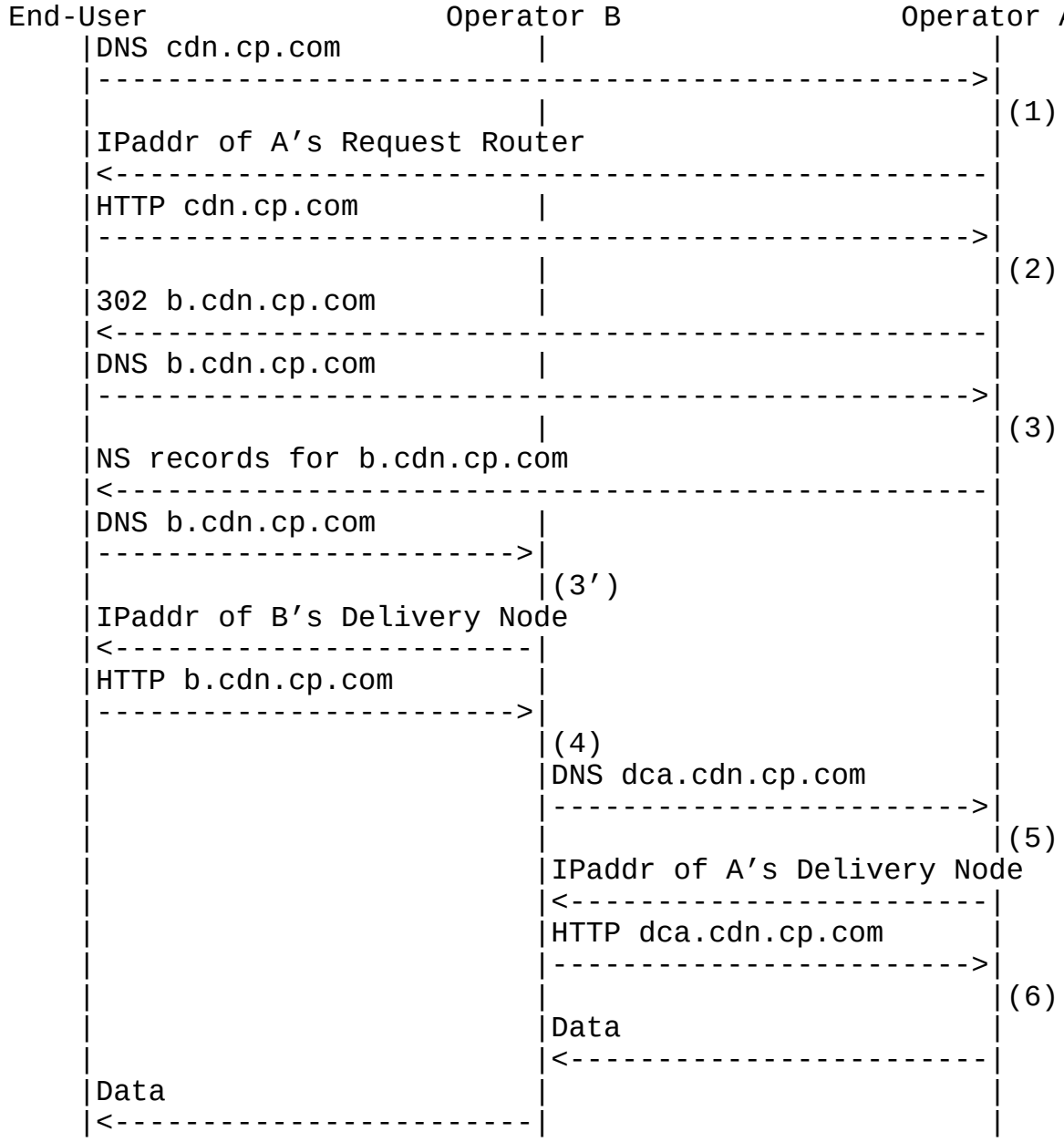

Figure 2: Request Trace for Method Two

Figure 2 depicts the exchange of DNS and HTTP requests. The main differences from Figure 1 are the alternative strategy for constructing CDN-domains and the replacement of Step 3 with two sub-steps, denoted 3 and 3'. We summarize as follows.

1. A Request Router for Operator A processes the DNS request for its customer based on CDN-domain cdn.cp.com and recognizes that the end-user is best served by another operator's CDN, but in order to redirect the end-user to that CDN, it must first get the end-

user to issue an HTTP request. Thus, it returns the IP address of a Request Router in Operator A.

2. A Request Router for Operator A processes the HTTP request and again 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" the distinguished token for Operator B onto the original CDN-domain. For example, the redirected URL might be b.cdn.cp.com.

3. The end-user does a DNS lookup using the modified CDN-domain (e.g., b.cdn.cp.com). This name is first resolved by A's Request Router (which is responsible for resolving cdn.cp.com), which returns an NS record for B's Request Router to the end-user. Sub-step 3' then involves B's Request Router selecting a suitable delivery node. Alternatively, Step 3 could be implemented by having A's Request Router recursively call B's Request Router,returning B's answer to the end-user.

4. The end-user requests the content from B's delivery node, potentially resulting in a cache miss. B sees its distinguished token on the CDN-domain and so "pops it off," recovering cdn.cp.com. Operator 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 constructed by augmenting the original CDN-domain with a distinguished token (e.g., dca.cdn.cp.com).

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.

6. Operator A serves content for the requested CDN-domain. 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.

Note that while the second method always uses a CDN-domain that is contained in domain name cp.com, it requires a tighter interaction between A and B. Specifically, Operator A must know the NS records for Operator B's DNS-based redirection service. This also forces the downstream CDN to participate in DNS-based redirection, which potentially infringes on B's ability to use an alternative redirection strategy. This makes the second method potentially less suitable for a heterogeneous CDN interconnection scenario.

5.2.1. Configuration Summary

Operators must exchange the following information to peer with each other:

- The operator's unique id (operator-id) that can be used to construct a distinguish CDN-domain;
- The set of IP prefixes for which the operator is prepared to deliver to the end-user; and
- NS records for the operator's set of externally visible redirection servers.

Operators must perform the following URL conversions:

- When a Request Router in an upstream CDN sees an end-user IP address best served by a downstream peer, it converts "cdn-domain to "operator-id.cdn-domain" (for the appropriate peer's operator-id) and returns an HTTP 302 redirect for the new URL.
- When a delivery node in a downstream sees a URL of the form "operator-id.cdn-domain" (for its operator-id), it verifies that "cdn-domain" is served by a known CDN peer, and if so, converts it to "dca.cdn-domain", and issues an HTTP request for that new URL.

DNS must be configured in the following way:

- The content provider must be configured to make the operator that serves "cdn-domain" the authoritative DNS server for that name.
- The operator that serves "cdn-domain" must be configured so that a request for "dca.cdn-domain" returns a delivery node, and a request for "operator-id.cdn-domain" is redirected to the peer denoted by operator-id.
- An operator with unique id operator-id must be configured so that a request for "operator-id.cdn-domain" returns a delivery node.

5.3. Third Method

The third method relies on indirection within the DNS protocol to avoid the additional HTTP redirections of the first two methods. This method also has the advantage that it is transparent to the end-user. No user-visible HTTP redirection is necessary, so the participating CDNs need not use CDN-domains that are contained in cp.com. Figure 3 depicts the exchange of DNS and HTTP requests. The main differences from Figures 1 and 2 are the lack of HTTP

redirection and transparency to the end-user.

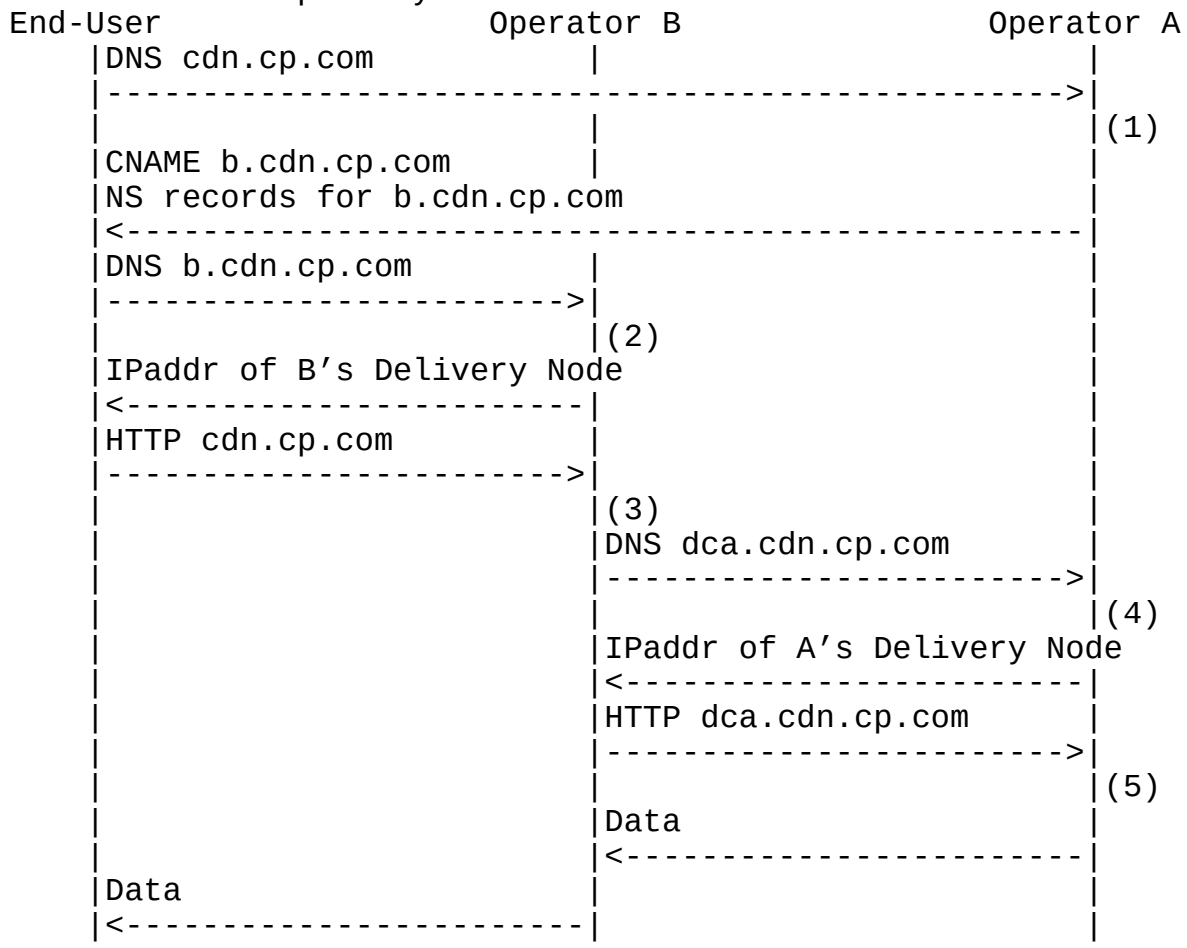

Figure 3: Request Trace for Method Three

1. Request Router for Operator A processes the DNS request for its customer based on CDN-domain cdn.cp.com and recognizes that the end-user is best served by another CDN. The client IP used in this determination is obtained either through the DNS client subnet extension or by embedding the client IP in the DNS name [I-D.vandergaast-edns-client-subnet]. The Request Router returns a DNS CNAME response by "stacking" a distinguished token for Operator B onto the original CDN-domain (e.g., b.cdn.cp.com), plus an NS record that maps b.cdn.cp.com to B's Request Router.

2. The end-user does a DNS lookup using the modified CDN-domain (e.g., b.cdn.cp.com). This causes B's Request Router to respond with a suitable delivery node. Note that as sub-steps, CP's authoritative DNS server will redirect the client to Operator A's Request Router, which will in turn redirect the client to Operator B's Request Router, in both cases by returning the appropriate NS records.

3. The end-user requests the content from B's delivery node. The requested URL contains the name cdn.cp.com. (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 constructed by augmenting the original CDN-domain with a distinguished token (e.g., dca.cdn.cp.com).

4. 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.

5. Operator A serves content for the requested CDN-domain. At this point the delivery node can issue an HTTP 302 redirect if the wrong delivery node was selected in step 4. 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.

A potential problem with this method is that the upstream CDN depends on being able to learn the network (CDN) that serves the end-user from the client address in the DNS request. If either the ClientIP extension is used or if the end-user uses the operator's LDNS, then this is the case. If not--i.e., 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, 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).

5.3.1. Configuration Summary

Operators must exchange the following information to peer with each other:

- The operator's unique id (operator-id) that can be used to construct a distinguish CDN-domain;
- The set of IP prefixes for which the operator is prepared to deliver to the end-user; and
- NS records for the operator's set of externally visible redirection servers.

Operators must perform the following URL conversions:

- When a delivery node in a downstream sees a URL for a non-local "cdn-domain", it verifies that "cdn-domain" is served by a known CDN peer, and if so, converts it to "dca.cdn-domain" and issues an HTTP request for that new URL.

DNS must be configured in the following way:

- The content provider must be configured to make the operator that serves "cdn-domain" the authoritative DNS server for that name.

- When the operator that serves "cdn-domain" sees an end-user IP address best served by a downstream peer, it returns CNAME and NS records for "operator-id.cdn-domain" (for the selected peer's operator-id).

- An operator with unique id operator-id must be configured so that a request for "operator-id.cdn-domain" returns a delivery node.

- The operator that serves "cdn-domain" must be configured so that a request for "dca.cdn-domain" returns a delivery node.

5.4. Discussion

We conclude this section by tying up three loose ends. A fourth loose end, verifying that a CDN-domain belongs to a peer, is postponed to the section on Security Considerations.

5.4.1. Method Selection

One take away form this discussion is that no single request-forwarding method is suitable for all situations. Instead, we expect a pair of operators will agree to use the best available method, depending on circumstances. The method selection protocol might be as follows:

- If the correct downstream CDN can be determined (i.e., a global LDNS is not used) and both CDNs support the third method, then use the third method.

- Else-if both CDNs support the second method, then use the second method.

- Else use the first method.

5.4.2. Overload Conditions

In the event the downstream CDN is overloaded, it can redirect the end-user back to the upstream CDN by sending an HTTP 302 redirect. It will need to use a URL that informs the upstream CDN that it should not re-redirect the end-user back to the downstream CDN. The distinctive CDN-domain "dca.cdn-domain" could serve this purpose, but another distinctive token (e.g., "overload.cdn-domain") could be used instead to disambiguate the two scenarios for which the upstream CDN is to serve the content rather than redirect the user.

Note that the upstream CDN has an opportunity to learn about the capacity of the downstream CDN by monitoring how often such overload redirects happen. It is not necessary for the two CDNs to exchange dynamic capacity information out-of-band, although it would be reasonable for operators to exchange course-grained capacity expectations as part of a peering agreement.

It is also possible to piggyback load information on other HTTP messages exchanged between operators. (They can also implicitly determine live-ness via DNS queries.) However, we view such information as a hint--as would also be the case with any out-of-band interface--since it's always possible that no capacity is available at the moment an actual user request is processed. In other words, any approach to interconnection will need to accommodate overload redirects; we simply propose to make this the primary means for communicating load information between CDNs.

5.4.3. Advertising Peering Information

Each of the methods requires CDN peers to exchange information with each other. Depending on the method(s) supported, this includes

- The operator's unique id (operator-id) or distinguished CDN-domain (operator-domain);
- The set of IP prefixes for which the operator is prepared to deliver to the end-user; and
- NS records for the operator's set of externally visible redirection servers.

Of these, the two operator identifiers are fixed, and can be exchanged off-line as part of a peering agreement. The IP address blocks served are relatively static, and perhaps even negotiated as part of a peering agreement. It's not obvious that a dynamic protocol is required to exchange this information. The third potentially changes with some frequency, but an existing protocol--

DNS--can be used to dynamically track this information. That is, a peer can do a DNS lookup on operator-domain to retrieve the set of NS records corresponding to the peer's redirection service.

6. Additional Interfaces

The discussion to this point has focused on request routing. This section extends the scope to include the other elements of a complete CDN interconnection scheme.

For example, it is necessary for the upstream CDN to have visibility into the delivery of content it originates to end-users connected to the 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 sometimes called the Logging interface, although we also consider the related (but distinguishable) Monitoring interface.

Similarly, the upstream CDN may also require control into how the downstream CDN delivers its content, for example, allowing it to purge content from the downstream CDN's caches or control what end-users are permitted to download its content. This is sometimes called the Control interface.

Finally, the upstream CDN may need to inform the downstream CDN about the content it is expected to deliver, for example, to what regions (e.g., countries) the content may be delivered and at what times the content may be delivered. This is sometimes called the CDNI Metadata interface.

6.1. Logging

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
- Referrer - 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 is then used to filter traffic log entries so only those entries matching the upstream CDN are reported to the corresponding operator.

The only 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 (e.g., to validate the upstream CDN), 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.

6.2. Monitoring

In addition to off-line traffic logs, accurate real-time traffic monitoring requires that the downstream CDN inform the upstream CDN each time it serves upstream content from its cache. The downstream CDN can do this 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.

6.3. Control

Being able to respond to a conditional GET request also 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.

Fine-grain control over how the downstream CDN delivers content on behalf of the upstream CDN is also possible. For example, by including the X-Forwarded-For HTTP header with the conditional GET request, the downstream CDN can report the end-user's IP address to the upstream CDN, giving it an opportunity to control whether the downstream CDN should serve the content to this particular end-user. The upstream CDN would communicate its control directive through its response to the conditional GET. The downstream CDN can cache information for a period of time specified by the upstream CDN, thereby reducing control overhead.

Thinking beyond what control operations can be done in-line, it is reasonable to argue that all CDNs already export a "content purge" operation to their customers, and so it is straightforward to also export this interface to an upstream peer. Of course, agreement as to the syntax and semantics of this call will be required.

6.4. CDNI Metadata

We save the CDNI Metadata for last because its utility is less clear. The intent is to give the upstream CDN an opportunity to inform downstream peers about the rules governing the content it might be asked to deliver. However, the mechanisms already presented may mitigate the need for an explicit CDNI Metadata interface.

Specifically, instead of the upstream CDN using an out-of-band Metadata interface to inform the downstream CDN of any geo-blocking restrictions or availability windows, the upstream has two options. The first is to redirect a given request to the downstream CDN only if that CDN's advertised delivery footprint is acceptable for the requested URL. Similarly, the request should be forwarded only if the current time is within the availability window. The second is to perform access control on a per-request basis, as outlined in the previous section. That is, the CDNI Metadata interface is effectively handled in-band.

Both strategies keep the locus of control over access decisions with the upstream CDN, which has a direct relationship with the content provider, and hence, authoritative knowledge about all relevant metadata. Of course, the downstream CDN is free to cache this information according to any upstream CDN caching directives.

7. IANA Considerations

This memo includes no request to IANA.

8. Security Considerations

Each of the above request routing methods includes a step that requires the downstream CDN to validate that a peer CDN serves the requested CDN-domain. This is a critical step to ensure that a malicious content provider or client cannot trick a downstream CDN into serving as an open proxy. Although other approaches are possible--for example, a signed token generated from a shared secret could be encoded in each URL--we summarize two straightforward ways to validate the upstream CDN.

The first approach 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. An operator also has to report the CDN-domains it serves in order to facilitate billing (see Section 6), but this can be done through an independent third-party (a so-called CDN Exchange) rather than by directly advertising CDN-domains to each peer CDN.

A second, less intrusive approach is for the upstream CDN to advertise its set of externally accessible DNS-based Request Routers. This is essentially a set of NS records, which is already required to be advertised by methods two and three. The downstream CDN can validate that a server in this set is used to resolve the distinguished CDN-domain "dca.cdn-domain". Note that advertising this information is a new requirement for method one, but it can be avoided by encoding the upstream operator’s distinguished CDN-domain in the URL returned in Step 2 and the URL requested in Step 4. For example, the returned URL returned in Step 2 would be peer.op-b.net/peer.op-a.net/cdn.cp.com, where the downstream CDN issues a request for peer.op-a.net/cdn.cp.com in step 4, ensuring that only the upstream peer with distinguished CDN-domain peer.op-a.net provides the data.

9. References

9.1. Normative References

[I-D.bertrand-cdni-experiments]
Bertrand, G., Le Faucheur, F., and L. Peterson, "Content Distribution Network Interconnection (CDNI) Experiments", February 2011.

[I-D.jenkins-cdni-problem-statement]
Niven-Jenkins, B., Le Faucheur, F., and N. Bitar, "Content Distribution Network Interconnection (CDNI) Problem Statement", March 2011.

[I-D.vandergaast-edns-client-subnet]
Contavalli, C., van der Gaast, W., Leach, S., and D. Rodden, "Client subnet in DNS requests", January 2011.

9.2. Informative References

[RFC2616] Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., and T. Berners-Lee, "Hypertext Transfer Protocol -- HTTP/1.1", RFC 2616, June 1999.

Authors' Addresses

Larry Peterson
Verivue Inc.
2 Research Way
Princeton,
NJ

Phone: +1 978 303 8032
Email: lpeterson@verivue.com

John Hartman
Verivue Inc.
2 Research Way
Princeton,
NJ

Phone: +1 978 303 8038
Email: jhartman@verivue.com

Marcin Pilarski
Orange Labs/Warsaw University of Technology
pl. Politechniki 1
Warsaw, Mazowieckie 00-661
Poland

Phone: +48 22 699 56 01
Email: marcin.pilarski@telekomunikacja.pl