R. Tewari
T. Niranajan
S. Ramamurthy
IBM

Internet-Draft
Document: draft-tewari-webi-wcdp-00.txt
Category: Experimental
Expires: August 2002

February 2002

WCDP 2.0: Web Content Distribution Protocol
draft-tewari-webi-wcdp-00.txt

Status of this Memo

This document is an Internet-Draft and is in full conformance with all provisions of Section 10 of RFC 2026 except that the right to produce derivative works is not granted.
Internet-Drafts are working documents of the Internet Engineering Task Force (IETF), its areas, and its working groups. Note that other groups MAY also distribute working documents as Internet-Drafts.
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'.
The list of current Internet-Drafts can be accessed at http://www.ietf.org/ietf/1id-abstracts.txt
The list of Internet-Draft Shadow Directories can be accessed at http://www.ietf.org/shadow.html.

Abstract

Cache consistency at web intermediaries is required for scalable content delivery on the web. In this document we describe the Web Content Distribution protocol (WCDP), which is an invalidation and update protocol to maintain cache consistency for a large number of frequently changing web objects. WCDP supports different levels of consistency: strong, delta, weak, and explicit consistency. It supports atomic invalidates and mutual consistency among objects and handles multiple deployment architectures. WCDP handles scalability by grouping objects and messages and by hierarchical intermediary organization. WCDP operates between the origin server, mirror sites, and the participating web intermediaries. It is not, however, targeted for inter-CDN operations but should be able to work with a peering protocol.

Table of Contents

WCDP 2.0: Web Content Distribution Protocol 1
Status of this Memo 1

Abstract 1
Table of Contents 1
1 Introduction 2
2 Terminology 3
3 Design Features 5
3.1 Scalability 5
3.2 Consistency levels 5
3.3 Security 7
3.4 Invalidates and Updates 7
3.5 Message delivery architecture 8
3.6 Deployment cases 8
4 WCDP Protocol Details 8
4.1 Object Invalidation Identity 8
4.2 Object Grouping 9
4.3 Content Groups and Subscription: 10
4.4 Consistency support: 11
4.5 Message Types and Formats 13
4.5.1 Invalidation request 13
4.5.2 Invalidation response 14
4.5.3 Register request 14
4.5.4 Register response 14
4.5.5 Join request 15
4.5.6 Join response 15
4.5.7 Commit request 15
4.5.8 Commit response 15
4.5.9 Heartbeat request 16
4.6 Distribution Hierarchies 16
4.7 Failure and Recovery 16
4.8 Transport protocol 17
4.9 Message Exchange Examples 18
4.10 End to End Flow 18
5 References 19
6 Acknowledgements 20
7 Author address 20

1 Introduction

A web cache invalidation (and update) protocol is required to reduce client-observed latency and server load for moderately changing web objects. In the traditional proxy cache mode, the intermediary (proxy cache) pulls and refreshes data on demand from the origin server. In this mode, consistency management is controlled by the intermediary. The content provider can provide hints based on HTTP cache control headers with expiration times or a max-age directive. However, these are coarse mechanisms and do not control the level and degrees of consistency that may be required by content providers for dynamic content. Most expiration times are not known a priori and content providers resort to setting very small values to control consistency. The frequent polling by the intermediaries using the HTTP if-modified-since (IMS) requests reduces the benefit of caching by adding to the latency observed by the users. The aim of the WCDP invalidation protocol is to enable server-driven

consistency where the content provider can dynamically control the propagation and visibility of an object update. In server-driven consistency, the origin server invalidates or updates the data when it changes without the intermediary resorting to frequent polling. The server-driven invalidation can extend over an infinite period of time or be guaranteed for a shorter time duration governed by a *lease* between the intermediary and the origin server.
For a web caching system we assume there is a designated origin server (the content provider's server), a group of replica/mirror origin servers, a group of intermediary caches, and the user agents. The intermediaries, the origin server and the user agents are not necessarily in the same administrative domain. The WCDP protocol operates between the origin server (or a delegated server) and the intermediaries. WCDP does not address the issue of maintaining consistency at the user agent.
WCDP supports multiple levels of consistency: (1) strong consistency for mirror sites, (2) delta consistency for participating intermediaries, and (3) explicit consistency as a default. It supports atomic invalidates for maintaining mutual consistency among objects. The intermediaries can explicitly subscribe to receive invalidation (or update) notifications or can rely on an implicit subscription, set up by an administrator for mirror sites or partnering intermediaries.
WCDP enhances scalability by grouping objects together as an addressable unit and grouping messages together. It supports distribution hierarchies for scalable message delivery. WCDP assumes a reliable underlying transport mechanism for message delivery.
Although not part of the protocol, WCDP supports authorization and authentication for different levels of security support.
The following sections describe the design features of WCDP and the protocol details.

The WCDP protocol is based on implementation experiences with the Content Distribution Framework of the IBM Websphere Edge Server Version 2.0. However, the WCDP 2.0 protocol for invalidation, as described in this document, is not consistent with the current implementation of IBM WebSphere Edge Server. We are aware of patents filed in this area that relate to this protocol.

2 Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this
document are to be interpreted as described in RFC 2119 [4].

Definitions

WCDP Client: The intermediary between the user agent and the origin server. It is the client for receiving WCDP invalidation notifications.

Client: Client is used interchangeably with “WCDP client”.

User agent: The user agent is most commonly the user’s “browser” that makes a request to the origin server.

WCDP server: A designated server that acts as the authority for sending invalidation notifications. The origin server can function as the WCDP server or designate another server.

Origin server: The origin server stores and serves the authoritative copy of an object.

Sender: The initiator of a message. The sender can either be the WCDP client or the WCDP server

Receiver: The recipient of a message. It can either be the WCDP client or the WCDP sever

Message: A message is a unit of communication between the sender and receiver. The message can be sent by the WCDP server or the WCDP client and received by the other. A message contains a request or a response. Requests and responses are further classified by type according to their purpose. Types include invalidation, join, register, commit etc.

Message group: Multiple requests or responses batched together

Object group: Multiple objects that are addressed as a single unit for scalability.

Content group: Multiple objects that are treated as a unit for subscription. Content groups are typically larger than object groups. For example, objects can be organized by topics for subscription (similar to a channel concept).

Atomic invalidation: The invalidation of a set of objects that should be executed atomically using lock semantics.

Individual invalidation: The invalidation of an individual objects.

Consistency levels: The types of consistency that are supported. These include explicit consistency, strong consistency, delta consistency and mutual consistency.

Heartbeat: An "I-am-alive" message sent by the invalidation server to the client caches at regular intervals. Heartbeats are required to implement delta and strong consistency.

3 Design Features

3.1 Scalability

For the invalidation protocol to be scalable it should be able to scale with respect to the number of objects for which consistency has to be maintained, the number of messages sent and received, and the number of clients to which invalidations are sent. Scalability aims at reducing the state at the WCDP server and the number of messages such that their growth is not in proportion to the number of objects and the number of client caches. Similarly, the WCDP client also needs to scale to a large number of objects in the cache. Scalability can be achieved by aggregating objects into object-groups and aggregating multiple messages into a single message-group. The protocol can invalidate an individual object or a set of objects addressed as a unit in an object-group. An object-group consists of a set of objects that are related and addressed as one. An object can belong to *multiple* object-groups. Clients can be organized into a distribution hierarchy where the WCDP server only communicates with the WCDP clients at the top level of the hierarchy.

3.2 Consistency levels

WCDP supports multiple consistency levels to control how and when the object changes are notified to the WCP clients. In the simplest case a WCDP client relies on explicit

consistency based on the HTTP cache-control headers and Expires tag. In another case, WCDP supports best-effort invalidations providing weak consistency. Weak and explicit consistency are supported by default. More stringent forms of consistency such as delta and strong consistency are explicitly requested. The multiple consistency levels that are supported by WCDP include:
i) *strong consistency:* where a read at the WCDP client reflects the last committed update at the origin server, and the update at the origin server is not made "live" until the client state is known,
ii) *delta consistency*: where the read at the WCDP client cache can be up to "delta" time units stale with respect to the last committed update at the origin server,
iii) *weak consistency*: where the read at the WCDP client does not necessarily reflect the last update at the origin server but some correct previous value,
iv) *explicit consistency*: where an expiration time of an object is provided or a time-to-live (TTL) value is provided by the origin server given some a priori knowledge.
v) *mutual consistency*: where a group of objects are mutually consistent with respect to each other. In this case some objects in the group cannot be more current than the others.

Strong consistency is useful for mirror sites that need to reflect the current state at the origin. It is also required if WCDP is used to update multiple origin servers that are part of a big cluster. Strong consistency for invalidates is implemented by requiring the WCDP clients to acknowledge the receipt of an invalidation message. Only after the receipt of all acknowledgements (or a timeout value) is the new version of the object made live at the origin server. Combining strong consistency with updates is more complex. This is achieved in a two-phase manner, the details of which along with the failure scenarios are described in the following section on protocol details.

Certain type of applications can tolerate stale data as long as it is within some known time bound. For such applications delta consistency is recommended.
Delta consistency assumes that there is a bounded communication delay between the WCDP server and client. This does not require an acknowledgement from the clients before making the content live. An invalidation message is sent such that the object cached at the WCDP client is invalidated within "delta" time units after the change at the origin server.

Mutual consistency is useful when a certain set of objects at a WCDP client (e.g., the fragments within a sports score page, or within a financial page) need to be consistent with each other. In this case they are atomically invalidated such

that they all either reflect the new state or remain in the earlier stale state.

The protocol implementation MUST support weak and explicit consistency. Supporting more stringent consistency levels such as delta, mutual, or strong consistency is RECOMMENDED.

3.3 Security

The WCDP client needs to authenticate itself to the WCDP server and vice versa. The data can optionally be encrypted. Simple access control checks are supported to determine which clients are allowed to receive invalidations. Also an authorization check is supported to determine if the WCDP server has the authority to send an invalidation message. WCDP provides authentication and encryption by using HTTPS for communication. Server-side and Client-side SSL certificates are used. Plug-in points are provided for authorization, both at the WCDP server and at the WCDP client.
A more detailed description of the security support will be described in a later version of this draft.

3.4 Invalidates and Updates

If data is changing infrequently then, for small data sizes, sending the updated object instead of an invalidate message improves performance. Invalidation requires 3 messages (an invalidation message, a read request on a miss, and a new data transfer) and adds extra latency, while an update requires one message (new data transfer). Delta encoding techniques have been designed to reduce the size of the data to update by sending only the changes to the object[10]. (Delta encoding is not related to delta consistency). Updates, however, require better security guarantees and make strong consistency management more complex. Nevertheless, updates are useful for mirror sites where data needs to be "pushed" to the replicas when it changes. Updates are also useful for preloading caches with content that is expected to become popular in the near future

WCDP supports invalidation by default but also extends it to support updates via a refresh directive. In WCDP updates are used to "push" content from the origin server to mirror sites and is handled by combining an invalidation with an *immediate- refresh* directive that causes the WCDP client to send a read (or IMS) request to the origin server to get a new copy of the object. It is the responsibility of the client to load the new version of the object from the origin server. If all clients happen to load immediately, it may cause a load surge at the origin server. The origin server can further extend an invalidate with a *delayed-refresh-directive* and a TTL value that defines the duration the client must wait before sending a read request. A WCDP

client sends a read request to the origin server only after the TTL time interval. This limits the burst of requests at the origin server.

The protocol implementation SHOULD support invalidation messages to be combined with a delayed-refresh directive to support updates.

3.5 Message delivery architecture

There are 3 different delivery scenarios that the protocol supports: i) single point-to-point connections between the WCDP server and the WCDP clients, ii) an application layer multicast between the WCDP server and a hierarchy of WCDP clients, and iii) a gateway between multiple WCDP servers and multiple WCDP (and non-WCDP) clients.
The single point-to-point connections is the usual case. For scaling to a large number of clients, an application-layer multicast within a hierarchy of clients is desirable. A WCDP client can also act as a gateway, forwarding (and possibly translating) messages from multiple WCDP servers to other WCDP (or non-WCDP) clients and vice versa. The gateway will act as a proxy for the WCDP server(s). The gateway can also do protocol translation if the client is not a WCDP client and vice versa. The protocol does not determine how such gateways or hierarchies are assigned or located.

3.6 Deployment cases

There are 3 deployment cases for the invalidation protocol: i) where the WCDP server and the WCDP clients are within the same administrative control (e.g., an enterprise CDN), ii) where the WCDP server and the WCDP clients are in different (and possibly multiple) administrative domains (e.g., a bunch of proxy caches within different ISPs), and iii) where the WCDP server is in one domain and all the WCDP clients are in another administrative domain (e.g., clients within a CDN). The security considerations vary with the deployment scenario. Also the consistency levels could be based on the deployment scenarios.

4 WCDP Protocol Details

4.1 Object Invalidation Identity

In WCDP, an object is identified for invalidation by its i) obj_invalidation_id, and optionally, ii) a URL.

The "*obj_invalidation_id*" is a unique opaque string assigned by the origin server for the purpose of explicit invalidation of objects by name. It is useful to not attach any semantic meaning to the "*obj_invalidation_id*", but rather to view it as an opaque unique identifier associated with an object. When the WCDP client cache makes a GET request to the origin

server, it receives an "*obj_invalidation_id"* corresponding to the URI requested. The "*obj_invalidation_id"* is embedded in the HTTP response as a private header and MUST be stripped from the response before forwarding to the user agent. In an application multicast or gateway scenario, the header can be directly passed on if the requestor is also a WCDP client. How the origin server selects the "obj_invalidation_id" is outside the scope of the protocol specification.

The motivation for "*obj_invalidation_id"* is that the entity informing the WCDP server of a change on any object may not know the external URL for that object. For example, if notifications are being created by publishing software, it will deal with filenames; similarly for a web server. An application server such as IBM Websphere Application Server could have its own abstract notion of object identity.

When a cache requests the origin server for an object, the origin server sends an "*obj_invalidation_id"* as a private header in the HTTP response. The WCDP client then maintains the mapping between an "*obj_invalidation_id"* and an internal "*obj_cache_id",* and the external URL. This mapping is essential, when we consider that an HTTP Server maps external URLs to local filenames, but there is no way to compute a reverse mapping from filenames to URLs. WCDP solves the problem by piggybacking this information in the response from the origin server, allowing for incremental construction of the reverse mappings at the requesting WCDP client.

The internal "*obj_cache_id"* is the identifier by which the WCDP client cache matches an incoming request with the local cached object. For static content, it is typically the same as the external URL. For dynamic content with fragments, the external URL along with other HTTP header tags and cookie values is combined to create the "*obj_cache_id"* for each fragment. Details about computing "*obj_cache_id"* are given in [1]. For a given "*obj_invalidation_id"*, there could be multiple "*obj_cache_id's"*. For example, if an object has multiple variants (for different languages, user agent types, etc.) it may use the same "obj_invalidation_id" but will have different "obj_cache_ids" for each variant. The form, specification and interpretation of the "obj_cache_id" is not within the scope of the WCDP protocol and is determined by each individual WCDP client cache implementation.

4.2 Object Grouping

In WCDP objects can be grouped into object groups and addressed as a unit. Object groups enhance scalability by limiting the size and number of messages, and the state at the WCDP server. For example, all objects in a sub-directory can belong to the same object group. Each WCDP client is informed of the object group a requested object belongs to by the origin server. The origin server sends the

"object_group_invalidation_id(s)" along with the "object_invalidation_id" in the HTTP response as a private header. Objects can belong to multiple object groups. Invalidation messages can be issued for all the objects in an object group by just naming the group itself. When a WCDP client receives a message for an "objectgroup_invalidation_id", it applies it to all the objects in the cache that have the same "objectgroup_invalidation_id". For caching efficiency, object groups are typically composed of a smaller number of objects. How the origin server groups objects is outside the scope of the protocol. For example all objects in a sub directory can belong to an object group.

A related concept is that of "*atomic invalidations*". Objects can be related to each other due to references (hyperlinks) between them or due to inclusion (multiple dynamically computed objects are assembled to form a personalized page). These objects must be invalidated atomically, i.e., there cannot be an object that is more current than the others. Therefore, the protocol allows for invalidation messages to specify that a certain invalidation or update message must be carried out atomically. All objects in that message will be invalidated or updated atomically. In an atomic invalidate/update, the objects are invalidated/updated using lock semantics. The objects are not accessible to the user agents (locked) until all the objects are invalidated/updated.

Note that understanding relationships between pages could be complex. The needed intelligence could be embedded at the WCDP client cache, or be at the WCDP server. The protocol addresses the latter situation. If the WCDP client cache itself has the necessary information, the WCDP server need not rely on the protocol support for atomic invalidations.

4.3 Content Groups and Subscription:

Subscription to notifications by the WCDP clients enhances the scalability of the system by reducing the number of messages transmitted. Subscriptions are at the granularity of content groups. A content group is a large aggregation of objects. Objects can belong to multiple "*content groups*". Each content group represents objects that are related by user interest. For instance, content groups can be topic-based, e.g., *sports, news, sports/baseball, etc.* Objects can be classified into content groups by the content creator at the origin server. Similar to the object group, a content group becomes metadata with which the object is associated, again returned as part of the HTTP response private header. To reduce administrative complexities, object groups can be the same as content groups.
The granularity of content groups is typically intended to be coarse, i.e., a content group can contain a large number of

objects and object groups. Content groups are used for subscription while object groups are used for lowering the overhead of invalidation.

When an object changes, the WCDP server issues invalidation notifications to the WCDP clients that have subscribed to any of the content groups that the object belongs to. The WCDP server knows which WCDP clients to notify by one of the following two mechanisms:

Implicit Subscription: an administration step marks a WCDP client cache as an implicit subscriber. A subsequent request from that client cache to the origin server is assumed by the origin server to be an interest in receiving invalidation notifications for that object.

2. Explicit Subscription: the WCDP client sends an explicit *register* message to the WCDP server indicating the content groups that it is interested in. Once that message is acknowledged, the origin server will send invalidations to objects belonging to those content groups. The explicit *register* message can be invoked as a result of an administrator deciding that the WCDP client cache must now subscribe to those content groups. Alternatively, it can be invoked by an intelligent WCDP client cache that is monitoring its request activity, and once it crosses a threshold for the number and rate of requests for objects in a content group, will attempt to subscribe to that content group. The WCDP client cache will discover the content group name from the HTTP response header, and then send the *register* message.

To work with existing content, a purely administrative solution has been included as part of WCDP. Content groups and objects are associated by way of regular-expression-based mapping rules. An administrator configures the WCDP Server with the mapping rules. When an invalidation is to be sent, the WCDP Server would compute the content groups that the object belongs to, and forwards the notification to the subscribing WCDP clients. The location of the actual repository where the mappings are maintained is orthogonal to the protocol. For example, they could be maintained: (1) as metadata along with the content and served by the origin on a GET request, (2) by the WCDP Server, and (3) by an external name service such as an LDAP server that the WCDP clients, WCDP servers and the origin servers can query.

4.4 Consistency support:

WCDP by default supports explicit and weak consistency. The differences between strong and weak consistencies are not significant when pure invalidations are concerned.

With strong consistency and invalidations, the WCDP server waits (or times out) until it receives the invalidation responses from all the participating WCDP clients that were sent the invalidation requests. After the responses are received the object can be made "live" at the origin server. When strong consistency is desired with updates (that is, invalidates with the immediate refresh directive), the updates are propagated in a two-phase manner.
The goal of strong consistency in WCDP is to minimize the amount of skew between object versions at the WCDP clients. Strong consistency with updates can be configured at a per-notification level, or a per-node level.

1. Node-level strong consistency: Each participating WCDP client can be pre-configured to be strongly consistent. If so configured, invalidation with refresh notifications is issued in a two-phase manner. During the first phase, the WCDP server sends the invalidation notification with an "*immediate-refresh*" directive to all subscribing WCDP clients. Upon its receipt, each WCDP client pulls the content from the origin server and stores it a temporary location, and then sends an invalidation response to the WCDP server. When all WCDP clients have responded, the second phase commit message is sent by the WCDP server to all the WCDP clients, which causes them to make the new content "live"; the origin server also makes the content "live". The WCDP clients respond with a commit response. While the WCDP client caches are waiting for a commit request, any user agent request for the object is forwarded to the origin server and not cached. Failure scenarios are discussed in a later section. The underlying assumption made is that the messages are sent over a reliable transport such that the WCDP client and servers can determine if there is a failure of client, server or the network. Node level strong consistency is useful for mirror sites.
2. Notification-level strong consistency: The desired level of consistency can also be part of the notification message. This is useful in situations where the actual semantic of the update for that object requires strong consistency.

WCDP provides delta consistency if the message latency is bounded and is less than delta. With delta consistency, a WCDP client can be delta time units stale with respect to the origin server. The WCDP server sends a heartbeat messages with a period smaller than delta. The delta value can be defined per content group or explicitly per object or object group. However, different values of delta at fine-granularity adds a lot of overhead. It is recommended to have a common delta for a content group. In case a WCDP client does not receive a heartbeat for more than delta time units since the last heartbeat or request, it marks the corresponding content group as invalid. The result of this action is to revert to an explicit consistency mode for all objects at the WCDP client that belong to the content group.

4.5 Message Types

Messages in WCDP are either request or response messages which are sent and received by the WCDP clients and servers. Message can be grouped together into a message group to batch multiple messages together.

There are 9 types of messages in WCDP
1. Invalidation request
2. Invalidation response
3. Register request
4. Register response
5. Join request
6. Join response
7. Commit request
8. Commit response
9. Heartbeat request

4.5.1 Invalidation request

The invalidation request is sent by the WCDP server to the (subscribing) WCDP clients. The invalidation request consists of 1) list of <identifiers>, 2) the invalidation action, 3) the invalidation type, 4) the consistency level, 5) an optional list of WCDP servers to pull data from.
Each request is also tagged with a unique, monotonically increasing, request sequence number.

The identifier consists of "object_invalidation_id(s)" and/or "obj_group_invalidation_id(s)" along with an optional external URL. An invalidation request may contain multiple identifiers in order to perform multiple invalidations together using a single message or for requiring an atomic invalidate of the identifiers in the request. Multiple request messages can be batched and sent together in a message group.

The invalidation action consists of either: 1) immediate invalidate, 2) delayed invalidate at a specific time or interval, 3) immediate update (invalidate with an immediate-refresh directive), 4) delayed update (invalidate with delayed-refresh after a specified interval) and possible combinations. The refresh directive is to implement content "updates" by requiring the WCDP client cache to pull the content from the origin server. However, a WCDP client may not comply with the refresh directive and ignore pulling the content from the origin server. The delayed refresh is useful to stagger the requests at the origin server to avoid a surge of requests. It can also be used to schedule an update at a given time. The WCDP client can pull the content from the origin server only after the specified time has elapsed.

The invalidation action also contains a "force" option. The meaning of the force option depends on the requested action. For an "immediate invalidate" or "delayed invalidate", the force option requires the WCDP client to delete the content from its local repository. This is useful to remove all stale copies when an object is deleted and cannot be refreshed. For an "immediate update" or "delayed update", the force option requires the cache to pull the content; the absence of the force option would allow the WCDP client to decide, based on local metrics, whether to pull the content or not.

The invalidation type specifies if the object(s) need to be invalidated "*atomically*" or *individually*. In an individual invalidate each object or object in an object group is treated individually.

The consistency level determines the type of consistency required. Weak and explicit consistency are supported by default. Details about supporting delta and strong consistency are presented in a later section.

4.5.2 Invalidation response

The invalidation response is sent by the WCDP client to the WCDP server after receiving and processing the corresponding invalidation request. The invalidation response consists of a status code for each invalidation request. This consists of the 1) request sequence number, 2) status code for each object_invalidation_id.

Since multiple requests can be grouped in a message group the request sequence number is useful to match the requests and responses.

Examples of status codes are: "SUCCESS_OK", "OBJECT_NOT_FOUND", "WAITING_FOR_COMMIT", "NOT_AUTHORIZED".

4.5.3 Register request

The register request is sent by the WCDP client to the WCDP server to subscribe to notifications for objects in a content group. The register request consists of: i) a list of <contentgroup_id(s), last invalidate request sequence number if known> and ii) the consistency level supported by the client, iii) request sequence number. Depending on authorization requirements, it could contain credentials and other authorization information.

4.5.4 Register response

The register response is sent by the WCDP server to the WCDP client in response to a corresponding register message. The register response consists of: 1) status code and 2) lease

duration (optional), iii) request sequence number, 3) desired consistency level for each content group.
The WCDP server initiates a catch-up sequence to get the WCDP client up-to-date with respect to the WCPD server. It computes the last set of invalidate requests for the objects in the content group (that are later than the last invalidation sequence number that the WCDP client has seen) and sends them with the register response.

4.5.5 Join request

The join request is sent by the WCDP client to the WCDP server after recovering from a failure. The purpose of a join request is to initiate a catchup sequence where the WCDP client can get up-to-date with the WCDP server. The join request consists of 1) list of <content group id, last invalidate request sequence number>, 2) request sequence number

4.5.6 Join response

The join response is sent by the WCDP server to the WCDP client cache. On receiving a join request, the WCDP server will compute the list of invalidations that need to be sent to the joining client. It will then package the invalidations and send them in the join response. The WCDP server will resume sending heartbeats (if had stopped them due to the failure).
The join response consists of 1) status code 2) request sequence number.
This is followed with a batch of invalidation requests.

Note that instead of a join request the WCDP client could have just sent another register request. We have tried to distinguish the rejoin after failure and the initial register.

4.5.7 Commit request

When strong consistency is desired, the notifications are sent in a two-phase manner. The commit request is sent by the WCDP server to the WCDP client cache after it receives all acknowledgment responses (WAITING_FOR_COMMIT) back from the client caches. The commit request consists of 1) invalidation sequence number, 2) identifier (used in the original invalidation request).

4.5.8 Commit response

The commit response is sent by the WCDP client cache to the WCDP server and consists of 1) the invalidation sequence number, 2) status code

4.5.9 Heartbeat request

If delta consistency or strong consistency is required, the WCDP server sends a periodic heartbeat message to the client. The heartbeat period is determined by the value of delta and should be smaller than delta. Also the heartbeat interval should be smaller than the timeout value used by the strong consistency implementation. Typically delta consistency is defined per content group and the heartbeats are then associated with a content group. If there are multiple content groups that a WCDP client subscribes to, each with a different value of delta, the server selects the lowest value to determine the heartbeat interval.

4.6 Distribution Hierarchies

The distribution of invalidation notifications can be made scalable by constructing a distribution hierarchy. The hierarchy has the WCDP server at the root, WCDP clients (or clients acting as gateways) at the intermediate levels and WCDP clients at the leaf level. A WCDP client could belong to multiple distribution hierarchies, and a distribution hierarchy could propagate invalidations for multiple WCDP servers. A WCDP client at the non-leaf level acts as a proxy for the WCDP server, receiving notifications from the higher level and forwarding them to the WCDP clients in the lower level in its sub-tree. With a tree-like organization the load on the WCDP server can be reduced, providing scalability. Notifications are distributed in successive *waves*, where, each wave refers to distribution to one level of the hierarchy. Each wave is initiated at the WCDP Server after the previous wave has completed.

WCDP delinks notification distribution from content distribution. However, scalability needs to be built into the content transport as well. With updates using a refresh directive how content is distributed is important in guaranteeing proper consistency. Therefore, WCDP clients can be configured such that they form a *content hierarchy* as well. At the root of the hierarchy is the origin server where content was first created. When the wave of notifications is distributed from one level of the hierarchy to the next it carries with it the list of WCDP clients and servers that have successfully processed the notification and from whom the data can be pulled. The WCDP clients can be optionally told which WCDP server(s) or clients they can pull the content from.

4.7 Failure and Recovery

WCDP recovers gracefully from failures. When the WCDP client fails and comes back up, it sends a *join* message to the WCDP server, which contains information about the last notification it acted upon. The WCDP server plays back all

subsequent notifications that the cache had missed, while being down. To perform the above task, the invalidation server must maintain a persistent record of the notifications that it issued . Only the latest one is stored for each object, since invalidation and update actions are idempotent. During the *join* phase, the cache reverts to being explicit consistency, i.e., it obeys the *Cache Control* headers. Once the cache has caught up, the invalidation server will resume sending heartbeats.

When a network partition occurs between a WCDP client and the WCDP server, the cache reverts to obeying *Cache Control* headers. It will then attempt to subscribe with an alternate WCDP server. This can be an administrative task, or automatic. In any case, once it re-establishes a path to a WCDP server, it will send it a *register* message if it is a new server, or a *join* message if it was the old server. The WCDP server on a timeout on an invalidation response (for strong consistency) will remove the client from the registered set.

When a WCDP server fails, it stops sending heartbeats to its WCDP clients; this causes the clients to start following the *Cache Control* headers. Just as in the case of a network partition, clients will attempt to find another WCDP server. If there are none, they will periodically send *join* messages to the WCDP server. When it comes back up, it will respond to the *join* by sending any missed invalidations and then heartbeats.

If a WCDP client has failed, the WCDP server detects it by way of a timeout on an invalidation response, and stops sending it any heartbeats and removes it from the set of registered clients.

If the data pull from a WCDP client fails because of a problem with the origin server that it is pulling from on a refresh directive, the cache will fail over to another WCDP server from the set of servers that it knows about from the notification. If there are no other servers, the cache will send a failure in the invalidation notification response.

When a transient network outage or other circumstances causes the failure of a notification, the WCDP server will stop sending heartbeats (for strong and delta consistency) to the cache to force it to rejoin.

4.8 Transport protocol

WCDP messages are in XML and sent using an HTTP POST method, possibly over a persistent TCP connection. Possible extensions include a SOAP-based model, as well as configuring the invalidation server as a Web Service.

4.9 Message Exchange Examples

An example invalidation request message

```
POST   /invalidate_request  HTTP/1.1
Content Length:  512

<?xml version=”1.0”>
<!DOCTYPE Invalidate Request  “wcdp.dtd”>
<invalidation sequence_number = 100>
<identifier>
<object_invalidation_id  =  12345 />
<object_invalidation_group_id =  9876/>
</identifier>
<action>
<invalidate =immediate/>
<refresh= yes/>
<delay = 60 sec />
<force = no />
</action>
<type atomic= no />
<consistency  require_commit = no />
</invalidation>
```

The corresponding invalidate response message is

```
POST   /invalidate_response  HTTP/1.1
Content Length:  256

<?xml version=”1.0”>
<!DOCTYPE Invalidate Response  “wcdp.dtd”>
<invalidation sequence_number = 100>
<identifier>
<object_invalidation_id  =  12345 />
<status = OK />
</identifier>
<identifier>
<object_invalidation_group_id =  9876/>
<status = OK/>
</identifier>
</invalidation>
```

4.10 End to End Flow

The WCDP client sends an HTTP request to the origin server. In the HTTP response it obtains the invalidation identifiers for the object and any object group ids and the content group ids.
(a) For explicit registration the WCDP client sends a register message to the WCDP server (this is the origin

server unless the origin has redirected the message to another server) for the content groups it is interested in. It could determine them from a known repository or obtain it from the HTTP response message. (b) for implicit registration with admin support no register message is required.
If a register message is received the WCDP server responds with the register response followed by the last set of invalidation requests.
When an object changes the WCDP server sends an invalidation request with the desired action and invalidation type.
The WCDP client on receiving an invalidation request sends an invalidation response with the status code. In case of an update requested by the refresh directive it pulls the data. In case of strong consistency it waits to commit the update until a commit message is received.
The commit request message is sent by the WCDP server after it receives all the invalidation responses form the clients. A timeout value is set larger than the heartbeat interval. The client is removed from the registered set if no response is received. The server stops sending the client a heartbeat.
When requiring strong or weak consistency the server sends a regular heartbeat message. When the client does not receive a (or few say 3) heartbeat message(s) it sends a join request to the WCDP server.
When the WCDP client fails it sends a join request to the WCDP server on recovery.
The server responds with a join response and allows the client to do a catch-up.

5 References

1. Jim Challenger and George Copeland, CacheId Specification, IBM Corp. November 2001, http://agent86.watson.ibm.com/fragmenttag/index.html

2. Dan Li, P. Cao and M. Dahlin, WCIP: Web Cache Invalidation Protocol, March 2001, http://search.ietf.org/internet-drafts/draft-danli-wrec-wcip-01.txt

3. James Gwertzman and Margo Seltzer, " World-Wide Web cache consistency", In Proceedings of 1996 USENIX Technical Conference, pages 141-151, San Diego, CA, January 1996.

4. S. Bradner, “Key words for use in RFCs to indicate requirement levels,” BCP 14, RFC 2119, March 1997.

5. P. Cao and C. Liu, "Maintaining strong cache consistency in the World Wide Web" 17th International Conference on Distributed Computing Systems. 27-30 May 1997. IEEE Transactions on Computers (April 1998) vol.47, no.4 p. 445-57

6. V. Duvvuri, P. Shenoy and R. Tewari, “Adaptive Leases: A Strong cache consistency mechanism for the WWW”, INFOCOM, March1999.

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

8. D. Li and D. R. Cheriton. "Scalable Web Caching of Frequently Objects using Reliable Multicast", 2nd USENIX Symposium on Internet Technologies and Systems (USITS'99). October 1999. ftp://ftp.dsg.stanford.edu/pub/papers/mmo.htm

9. J. Yin, L. Alvisi, M. Dahlin, C. Lin, "Using leases to support server-driven consistency in large-scale systems" Proceedings of 18th International Conference on Distributed Systems. 26-29 May 1998.

10. J. C. Mogul, F. Douglis, A. Feldman, B. Krishnamurthy, "Potential benefits of delta encoding and data compression for HTTP" Proceedings of SIGCOMM, September 1997.

6 Acknowledgements

The authors would like to thank Jim Challenger, David E. Martin and Ron Doyle for valuable comments and corrections in the early stages of this document. They would also like to thank Lee Rafalow and Lisa Amini for the frequent discussions that have influenced this document. Finally, they are indebted to IBM’s Websphere Edge Server team and in particular Rajesh Agarwalla for contributions to the original invalidation protocol design.

7 Author address

Renu Tewari
IBM T.J. Watson Research Center
tewarir@us.ibm.com

Thirumale Niranjan
IBM Software Group
niranjan@us.ibm.com

Srikanth Ramamurthy
IBM Software Group
ramamur@us.ibm.com