CalConnect TC PUSH

Calendaring and scheduling — Push Discovery and Notification Dispatch Protocol

Committee Draft Standard

Warning for drafts

This document is not a CalConnect Standard. It is distributed for review and comment, and is subject to change without notice and may not be referred to as a Standard. Recipients of this draft are invited to submit, with their comments, notification of any relevant patent rights of which they are aware and to provide supporting documentation.

Recipients of this draft are invited to submit, with their comments, notification of any relevant patent rights of which they are aware and to provide supporting documentation.

The Calendaring and Scheduling Consortium, Inc.

4390 Chaffin Lane
McKinleyville
California 95519
United States of America

copyright@calconnect.org
www.calconnect.org

Contents

Abstract

This specification defines a framework and protocols for a push notification system that allows clients, application servers and push notification servers to interact with each other in a standardized manner.

The Calendaring and Scheduling Consortium ("CalConnect") is a global non-profit organization with the aim to facilitate interoperability of collaborative technologies and tools through open standards.

CalConnect works closely with international and regional partners, of which the full list is available on our website (https://www.calconnect.org/about/liaisons-and-relationships).

The procedures used to develop this document and those intended for its further maintenance are described in the CalConnect Directives.

In particular the different approval criteria needed for the different types of CalConnect documents should be noted. This document was drafted in accordance with the editorial rules of the CalConnect Directives.

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights. CalConnect shall not be held responsible for identifying any or all such patent rights. Details of any patent rights identified during the development of the document will be provided in the Introduction.

Any trade name used in this document is information given for the convenience of users and does not constitute an endorsement.

This document was prepared by Technical Committee *PUSH*.

Introduction

In a client/server protocol clients can typically create update delete “resources” (data) on the server as well as retrieve data on the server.

In many cases data can appear on the server as the result of some other client or server-side process interacting with the server. Thus clients need a way to detect when the data on the server has changed.

Most protocols provide a data synchronization mechanism to support that but typically clients need to “poll” the server to find out when changes have occurred. Network based polling is inefficient and instead push notifications are preferred as a way of alerting clients to new data or changes to existing data on the server.

1. Scope

This document defines a framework and protocols for a push notification system, called the Push Discovery and Notification Dispatch Protocol, that allows clients application servers and push notification servers to interact with each other in a standardized manner.

This document further provides an implementation of the protocol on WebDAV as an extension.

2. Normative references

The following documents are referred to in the text in such a way that some or all of their content constitutes requirements of this document. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments) applies.

IETF RFC 2119, BRADNER, S. *Key words for use in RFCs to Indicate Requirement Levels*. 1997. RFC Publisher. https://www.rfc-editor.org/info/rfc2119.

IETF RFC 3339, KLYNE, G., C. NEWMAN and Internet Engineering Task Force. *Date and Time on the Internet: Timestamps*. 2002. RFC Publisher. https://www.rfc-editor.org/info/rfc3339.

IETF RFC 3688, MEALLING, M. *The IETF XML Registry*. 2004. RFC Publisher. https://www.rfc-editor.org/info/rfc3688.

IETF RFC 3986, BERNERS-LEE, T., R. FIELDING and L. MASINTER. *Uniform Resource Identifier (URI): Generic Syntax*. 2005. RFC Publisher. https://www.rfc-editor.org/info/rfc3986.

IETF RFC 7230, Internet Engineering Task Force (committee). *Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing*. 2014. RFC Publisher. https://www.rfc-editor.org/info/rfc7230.

Calendaring and scheduling — Push Discovery and Notification Dispatch Protocol

3. Conventions Used in This Document

The key words “MUST”, “MUST NOT”, “REQUIRED”, “SHALL”, “SHALL NOT”, “SHOULD”, “SHOULD NOT”, “RECOMMENDED”, “NOT RECOMMENDED”, “MAY”, and “OPTIONAL” in this document are to be interpreted as described in IETF RFC 2119.

When XML element types in the namespaces “DAV:” and “urn:ietf:params:xml:ns:dav-push” are referenced in this document outside of the context of an XML fragment, the string “DAV:” and “DAV-PUSH:” will be prefixed to the element type names respectively.

4. Terminology

Application Server	Provides resources a client application might want to monitor for changes. Typical applications are email calendars and address books.
Push Gateway	A service to provide a common standardized interface to Push Delivery Services. A Push Gateway provides or relays one or multiple delivery channels the so called Transports.
Push Delivery Service	A Service which provides the actual push transport mechanism to the client application.

Transport	A Transport is a logical channel to a specific Push Delivery Service provided by a Push Gateway. It is identified by the transport-uri.
Client Application	An application that uses the services of the Application Server and wants to get notified instantaneously about certain changes on the server. A client application typically runs on a mobile or desktop device.
Push Notification	A message sent from the Application server to the Client Application to notify the client of an update. The basic information carried by the notification is “there was a change” for a specific Topic.
Topic	A Topic is a name for a notification feed or channel. Each watchable resource has a Topic that clients can subscribe to. Each subscriber to a particular topic will receive a notification when a substantial change was made to any of the resources with that Topic.

5. Architecture

This document introduces an entity called “Push Gateway” which acts as a proxy between an application server and a push delivery service. A Push Gateway provides at least one Transport. Each Transport is identified by a URI and connects to exactly one Push Delivery Service.

Push Gateways MAY support relaying so a push gateway might forward all or some notifications to another push gateway.

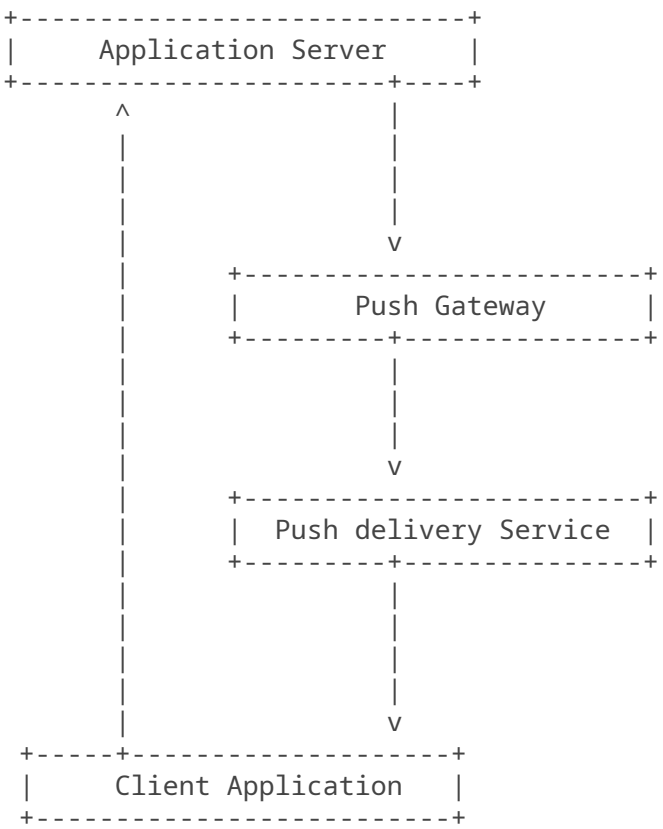

Figure 1

5.1. Application Server

The server is responsible for generating push topics and sending update notifications to the Push Gateway. A push topic is a unique token that identifies the update notification feed of a resource or

a group of resources. The topic is forwarded to the Push Gateway whenever a relevant change in one of these resources occurs.

This document doesn't specify how topics are generated. However for privacy reasons the topic MUST NOT contain user names user data (like folder/collection names) or URLs in plain text. If a server doesn't maintain opaque anonymous identifiers it SHOULD use a hash algorithm like SHA256 to generate an opaque identifier from resource properties.

Push topics MAY be generated on a per-user base for shared resources.

A server MAY change push topics at any time to improve privacy. If doing so the server MUST continue to send out push notifications for the old topic until all subscriptions to that topic have expired.

The application server maintains a mapping of subscribed push topics to a list of push gateways. It updates this mapping whenever:

— A new subscription request is received
— A response from the push gateway indicates that there are no active subscribers for a particular topic.

The application server doesn't maintain references to push clients because this information is opaque to the application server.

5.2. Push Gateway

The Push Gateway maintains a mapping of push-topics to a list of subscribed clients and expiration times. It updates the list whenever:

— it receives a new subscription
— a subscription expires or
— the Push Delivery Service returns that a specific client is no longer available.

If a push message for a specific topic is received the push gateway will notify all clients with an active (not expired) subscription for that topic.

A push gateway may relay messages for other gateways. A gateway that supports relaying MUST maintain a map of topics to gateways just like an application server.

5.3. Push Delivery Service

TBD: Minimum requirements for PDS to support this protocol. Describe what state information the PDS needs to maintain.

5.4. Client

TBD: what information does the client need to maintain

6. Protocol Workflows

6.1. App Server <→ Push Gateway bootstrap workflow

This protocol allows an application server to initialize the supported push transports by querying a set of configured push gateways. This requires that the application server knows the root URL of each configured gateway. In order to retrieve the list of supported transports it posts a JSON object with an empty list of push-transports to each gateway.

The following request shows the bootstrap request of an application server that was configured with the Push Gateway URL https://push.example.com/gateway

```
POST /gateway
Host: push.example.com
Content-Type: application/json
Content-Length: xxx

{ "push-transports": []}
```

Figure 2

The push gateway responds with a JSON object that contains an array of push transports.

```
HTTP/1.1 200 OK
Content-Type: application/json
Content-Length: xxx

{ "push-transports": [
  {
    "transport": {
      "transport-uri":
        "https://push.example.com/transport",
      "refresh-interval": 172800,
      "transport-data" : { ... }
    }
  }{
    "transport": {
      "transport-uri":
        "urn:uuid:01234567-0123-0123-0123-0123456789ab",
      "refresh-interval": 172800,
      "transport-data" : { ... }
    }
  }]
}
```

Figure 3

6.2. Client <→ App Server workflow

The communication between Client and Application Server is defined in the respective application protocol. The application protocol needs to be extended in order to support push.

This document describes the general idea behind the required extensions and gives a concrete definition for a WebDAV extension.

6.2.1. Client discovery and subscription workflow — Generic

TBD:

6.2.2. Unsubscribe

This document doesn't specify an explicit unsubscribe method. A client that doesn't wish to receive any further push notifications for a specific topic MAY send a subscription with an expiration date in the past.

An application server which receives such a subscription MUST handle it like any other subscription. In particular the Application Server MUST:

— verify the Push Topic and

— forward the susbcription to the Push Gateway.

A Push Gateway which receives a subscription with a passed expiration date MUST:

— remove the client from the list of subscribers to this topic and
— not send out any further push messages to this client.

6.2.3. Client discovery and subscription workflow — WebDAV

6.2.3.1. Push discovery

The following example shows a PROPFIND request on a user's calendar home to discover push support.

```
PROPFIND http://calendar.example.com/calendars/
Content-Type: application/xml
Depth: 0
Content-Length: xxx

<D:propfind xmlns:D="DAV:" xmlns:P="urn:ietf:params:xml:ns:dav-push">
  <D:prop>
    <P:subscribe-URL> />
    <P:supported-transport-set />
    <P:topic />
    <P:version />
  </D:prop>
</D:propfind>
```

Figure 4

The server responds with the respective properties:

```
HTTP/1.1 207 Multistatus
Content-Type: application/xml; charset=UTF-8
Content-Length: xxx

<D:multistatus xmlns:D="DAV:">
  <D:response>
    <D:href>/calendars/</D:href>
    <D:propstat>
      <D:prop>
        <P:subscribe-URL>
          <D:href>https://calendar.example.com/subscribe</D:href>
        </P:subscribe-URL>
        <P:supported-transport-set>
          <P:transport />
          <P:transport>
            <P:transport-uri>urn:uuid:01234567-0123-0123-0123-0123456789ab</P:
transport-uri>
            <P:transport-data>...</P:transport-data>
            <P:refresh-interval>172800</P:refresh-interval>
          </P:transport>
          <P:transport>
            <P:transport-uri>https://push.example.com/transport</P:transport-uri>
            <P:transport-data>...</P:transport-data>
            <P:refresh-interval>172800</P:refresh-interval>
          </P:transport>
        </P:supported-transport-set>
        <P:topic>123</P:topic>
        <P:version>1</P:version>
```



```
      </D:prop>
      <D:status>HTTP/1.1 200 OK</D:status>
    </D:propstat>
  </D:response>
</D:multistatus>
```

Figure 5

6.2.3.2. Push subscribe

Calendar server → Client — CS server advertises its supported push mechanisms

Clients request POST to P:subscribe-URL — does the actual subscription to the calendar server:

```
POST /subscribe HTTP/1.1
Host: calendar.example.com
Content-Type: application/xml; charset=UTF-8

<P:subscribe xmlns:P="urn:ietf:params:xml:ns:dav-push">
  <P:topic>123<P:topic>
  <P:topic>abc<P:topic>
  <P:selected-transport>
    <P:transport-uri>https://push.example.com/transport</D:transport-uri>
    <P:client-data>XYZ</D:client-data>
  </P:selected-transport>
  <P:expires>2017-10-07T12:00:00Z</P:expires>
</P:subscribe>
```

Figure 6

If one or more topics are invalid the entire request MUST fail without any subscriptions being recorded. In this case the server MUST return an error response containing a list of topics that failed. If a topic is valid but the authenticated user doesn't have access to any of the resources that the topic belongs to the server SHOULD treat this topic as being invalid and the request SHOULD fail.

6.3. App Server → Push Gateway subscribe workflow

When a client sends a request to subscribe to specific topics the application server MUST forward the subscription to the chosen gateway or to the gateway that announced itself as a proxy for the chosen gateway.

If a gateway acts as a proxy for another gateway it MUST forward the request to the proxied gateway.

The following example shows a request to subscribe to two topics.

```
POST / HTTP/1.1
Host: push.example.com
Content-Type: application/json

{
  "push-subscribe": {
    "topics": [ "123", "abc" ],
    "transport": {
      "transport-uri": "https://push.example.com/transport",
      "client-data": "XYZ"
    },
    "expires": "2017-10-07T12:00:00Z"
  }
```



```
}
```

Figure 7

To acknowledge the subscription the gateway SHOULD send an initial PUSH notification to the client.

A successful response contains the URL to send update messages to. The URL may be different than the transport URL. An Application Server MUST use this URL when sending push notifications to transports provided by clients.

```
HTTP/1.1 200 OK
Content-Type: application/json

{ "push-url": "https://push.example.com/" }
```

Figure 8

6.4. App Server → Push Gateway push workflow

Whenever a substantial change occurs in any of the resources the application server sends a Push Message to the gateway containing the Topics of the resources that have changed.

The following example sends a push notification for the Topics “123” and “abc”. The message for Topic “123” also contains a “client-id” to omit any notification to the sole client that modified the resource and caused this push message. The second message has a low priority and no “client-id”. Such a message could be generated by multiple clients acknowledging an alarm on a shared calendar.

```
POST / HTTP/1.1
Host: push.example.com
Content-Type: application/json

{
  "push": {
    "messages" : [{
      "topic": "123",
      "priority": 100,
      "timestamp": "2017-10-01T14:00:52Z",
      "client-id": "xyz"
    }, {
      "topic": "abc",
      "priority": 0,
      "timestamp": "2017-10-01T14:00:53Z"
    }]
  }
}
```

Figure 9

Response: HTTP status for success or HTTP status for failure with a XML/JSON error response body. It’s not an error if a topic is unknown or there are no active subscribers for this topic. Instead the response will contain a list of all topics without subscribers. The application server SHOULD update its topic-to-gateway mapping accordingly. The application server MUST assume that topics which were in the request and not in the “no-subscribers” list have been pushed to the client.

If there is a subscriber for each topic in the request the no-subscribers list MUST be omitted.

```
HTTP/1.1 200 OK
Content-Type: application/json
```



```
{ "push-response": {} }
```

Figure 10

If there are topics without active subscribers:

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "push-response": {
    "no-subscribers": [
      { "topic": "123"}
    ]
  }
}
```

Figure 11

7. Syntax Elements/Properties

7.1. Push gateway protocol

7.1.1. Bootstrapping

```
; root element
root {
        push-transports
}

; a list of push transports supported by a gateway
; in the request sent by the application server this is empty
push-transports "push-transports" [
        * transport
]

transport "transport" {
        ; The uri of the transport.
        "transport-uri" : uri,
        transport-data?
}

; optional data the client needs to know in order to subscribe
; to allow easy conversion to other formats,
; this object MUST NOT contain structured data.
transport-data "transport-data" {
  ^"": any
}
```

Figure 12

7.1.2. Subscription

```
; root element
root {
        push-subscribe
}

; the object describing the subscription
push-subscribe "push-subscribe" {
```



```
        topic-list,
        selected-transport,
        expires
}

; The list of topics to subscribe to. Unless a previous
; subscription is updated by a request, existing
; subscriptions won't be affected by new subscriptions.
topic-list "topics" [
        * topic
}

; The chosen transport type
selected-transport "selected-transport" {
        ; The transport-uri of the chosen transport
        "transport-uri" : uri,
        ; The client-data string as sent by the client
        "client-data" : string
}

; The time of when the subscription expires
; must be a UTC timestamp following
; https://tools.ietf.org/html/rfc3339
expires "expires" : RFC 3339 timestamp
```

Figure 13

7.1.3. Update notification

```
; The root object
root {
        "push" [ 1* message ]
}

; A message object, describing the update
message {
        topic,
        ? priority,
        timestamp,
        ? client-id
}

; The topic of the resource that has been updated
topic "topic" : string

; The priority of the change, with 0 being the lowest and 100
; being the highest priority
; If omitted, implementations SHOULD default to 50.
priority "priority" : integer 0..100

; The time of when the change occurred. The value MUST be a
; timestamp in UTC following https://tools.ietf.org/html/rfc3339
; If the server aggregated multiple updates before sending the push
; message, this MUST be the timestamp of the most recent update.
timestamp "timestamp" : RFC 3339 timestamp

; An optional id that identifies the client that triggered the update
; notification. Push gateways can use this information to suppress
; push messages to this particular client, in order to avoid
; unnecessary sync operations.
```



```
; If the server aggregated multiple updates from different clients
; into one message, it MUST omit the client-id to ensure all clients
; receive the push message.
client-id "client-id": string

; root element of the push subscribe response
root {
        ? no-subscribers-list
}

; A list of topics without active subscribers.
; Applications servers SHOULD not send further push messages for the
; enlisted topics to this transport unless a new client subscribes on
; this transport.
no-subscribers-list "no-subscribers" [
    1* topic }
]
```

Figure 14

8. XML Element definitions

8.1. WebDAV Properties

8.1.1. DAV-PUSH:push-subscribe-URL

Name push-subscribe-URL

Namespace urn:ietf:params:xml:ns:dav-push

Purpose Specifies the address to send the subscription requests to.

Description The push-subscribe-URL element contains exactly one DAV:href element with a URL that points to the subscription service endpoint.

Definition

```
<!ELEMENT push-subscribe-URL (DAV:href)>
```

Figure 15

8.1.2. DAV-PUSH:supported-transport-set

Name supported-transport-set

Namespace urn:ietf:params:xml:ns:dav-push

Purpose Specifies a list of transports supported by the application server.

Description This element contains the set of push transports supported by the server. The transport-uri element of each transport must be unique within the set of transports.

The set MAY contain one transport element without any child elements to indicate that the client may provide its own transport.

Definition

```
<!ELEMENT supported-transport-set (transport*)>
```

Figure 16

8.1.3. DAV-PUSH:transport

Name transport

Namespace urn:ietf:params:xml:ns:dav-push

Purpose Describes a specific transport.

Description A transport element represents a specific push transport path to clients on a specific service. In general it contains a transport-uri element that uniquely identifies the transport.

Definition

```
<!ELEMENT transport (transport-uri,
                     transport-data, refresh-interval)?>
```

Figure 17

8.1.4. DAV-PUSH:transport-uri

Name transport-uri

Namespace urn:ietf:params:xml:ns:dav-push

Purpose Specifies the URI that identifes the transport.

Description Clients compare the provided transport-uris to the transport-uris they support.

Definition

```
<!ELEMENT transport-uri (#PCDATA)>
PCDATA value: The URI identifying the transport.
```

Figure 18

8.1.5. DAV-PUSH:transport-data

Name transport-data

Namespace urn:ietf:params:xml:ns:dav-push

Purpose Contains a list of additional attributes that client needs to know in order to subscribe on this transport.

Description
Definition

```
<!ELEMENT transport-data ANY>
```

Figure 19

8.1.6. DAV-PUSH:refresh-interval

Name refresh-interval

Namespace urn:ietf:params:xml:ns:dav-push

Purpose Specifies the maximum refresh interval.

Description Specifies the duration in seconds after which the client is expected to re-subscribe. If the client didn't res-subscribe within this period of time the gateway MUST remove all subscriptions and no further push notifications will be delivered to the client until it subscribes again.

A Push Gateway MUST not accept subscription requests with an expiration time that would exceed the refresh interval.

Definition

```
<!ELEMENT refresh-interval (#PCDATA)>
PCDATA value: the maximum refresh interval in seconds
```

Figure 20

8.1.7. DAV-PUSH:topic

Name topic

Namespace urn:ietf:params:xml:ns:dav-push

Purpose Specifies the push topic of a resource.

Description The topic identifies the name of the update channel for a resource. Clients send the topic in a subscription request to inform application server and gateway that it wants to receive update notifications for the resource.

This document doesn't specify a specific format for topics nor a specifc algorithm to generate them.

Server developers MUST ensure that topics on different installations won't collide.

Resources within the same domain MAY share topics.

Definition

```
<!ELEMENT topic (#PCDATA)>
PCDATA value: the push topic
```

Figure 21

8.1.8. DAV-PUSH:version

Name push-version

Namespace urn:ietf:params:xml:ns:dav-push

Purpose Specifies the highest version number of the push protocol supported by this server.

Description
Definition

```
<!ELEMENT push-version (#PCDATA)>
PCDATA value: the highest push protocol version number
              supported by the application server
```

Figure 22

8.2. Subscription request

8.2.1. DAV-PUSH:subscribe

Name subscribe

Namespace urn:ietf:params:xml:ns:dav-push

Purpose Represents a subscription request document.

Description The subscribe request contains all information to subscribe to specific topics selecting a specific transport to deliver push notifications.

A subscription must have an expiration date after which the subscriptions will become void.

Definition

```
<!ELEMENT subscribe (topic+, selected-transport,
                     expires)>
```

Figure 23

8.2.2. DAV-PUSH:selected-transport

Name selected-transport

Namespace urn:ietf:params:xml:ns:dav-push

Purpose Specifies the transport the client has chosen.

Description The selected-transport element contains the transport-uri of the transport that the client has chosen for push delivery. It also contains a client-data element to be forwarded to the push gateway.

Definition

```
<!ELEMENT selected-transport (transport-uri,
                     client-data)>
```

Figure 24

8.2.3. DAV-PUSH:client-data

Name client-data

Namespace urn:ietf:params:xml:ns:dav-push

Purpose Contains a string the client needs to provide to the push-gateway for the chosen transport.

Description This element provides a mechanism for the client to communicate to the gateway. The format of the data string is not defined in this document. The application server MUST forward the client-data string as provided by the client.

Gateways SHOULD use this to authenticate clients.

Definition

```
<!ELEMENT client-data (#PCDATA)>
```



```
PCDATA value: client data as required by the push gateway
```

Figure 25

Name invalid-topics (precondition)

Purpose The request could not succeed, because it contained invalid push topics. This element contains one topic element for each rejected push topic. The client may repeat the request without those topics.

Definition

```
<!ELEMENT invalid-topics (topic+)>
```

Figure 26

9. HTTP Headers for DAV-Push

9.1. Push-Client-Id Header

```
Push-Client-Id = "Push-Client-Id" ":" token
```

Figure 27

The client sends this header to identify itself to the application server as the modifying instance. If the application server didn't coalesce multiple updates from different clients into a single push message, it SHOULD include the value in the update notification message. The provided token ([RFC7230]) MUST be percent-encoded as per [RFC3986]. Gateways can use this information to suppress push messages to this particular client.

The actual value of token is part of the contract between client and gateway. The token MUST NOT contain any sensitive data like user name or device identifiers. It SHOULD be either a random or an obfuscated token (using a cryptographic hash function).

10. Guidelines

10.1. Application Servers

Servers may want to implement some form of "keep-alive" within the push protocol to ensures clients know they are still connected in cases where actual data changes happen at long intervals (e.g., a calendar user who only makes changes once a day).

Priorities:

Range 0 — 100 — 0 is lowest and 100 is highest

e.g.:

— low priority — updates due to other attendees changing their partstat
— high priority — updates to events occurring in the next 24 hours

Priority is used by a client to indicate what level of push they want at a specific time. It can also be used by the push gateway or push delivery system to throttle push notifications to the client based on load.

Servers MAY delay the delivery of push notifications for several seconds in order to coalesce notifications. This is useful to give the server a certain amount of control over the client's behavior during times of high load.

Servers MUST NOT coalesce push notifications based on priority.

Application servers MAY allow clients to provide their own transports. If the transport-uri is not among the transport-uris as advertised by the application server, the transport-uri MUST be an HTTPS URL. If a client sends such a transport-uri, the application server SHOULD perform a transport discovery on the provided URL to discover all transports supported on this gateway.

10.2. Clients

Clients MUST be prepared that they might receive an initial push notification that acknowledges the subscription before the response to the push-subscribe request has been received.

Clients SHOULD NOT rely solely on push notifications. The framework described in this document does not make any guarantees about the delivery of a push notification. Clients should be prepared to trigger a synchronization themselves if no push message has been received within some time period.

Clients can expect that sometimes they will get a push but then not detect any actual changes when they sync (i.e., “no-op” push from server as a “keep-alive” mechanism).

10.3. Push Gateway

A Push Gateway SHOULD require some kind of authentication to be encoded in the client-data string. This document doesn’t specify any authentication methods. However, among others, encrypting the client-data string with a shared secret and digitally signing the data are two possible options to achieve this.

Client data MAY contain additional per-client preferences, like minimum priority to deliver or maximum delay of notifications when doing coalescing. This is part of the contract between client and transport an not subject of this specification.

Gateways MAY coalesce push notifications based on priority.

11. Security Considerations

To prevent abuse of the service, Push Gateways SHOULD require either servers or clients or both to authenticate. Servers SHOULD authenticate every request of Protocol #2 via HTTP.

Push Gateways use the `<gateway-data>` information to authenticate subscription requests from a Server by relating them to Client authorization requests. Clients will typically be authenticating to Servers to access protected data on the server and thus SHOULD authenticate when using Protocol #1.

12. IANA Considerations

This document uses a URN to describe a new XML namespace conforming to the registry mechanism described in RFC 3688.

12.1. Namespace Registration

Registration request for the push namespace:

URI	urn:ietf:params:xml:ns:dav-push
Registrant Contact	The IESG <iesg@ietf.org>

XML	None — not applicable for namespace registrations.

Bibliography

