Abstract

This document provides for the ability to send and receive notifications using HTTP over TCP/IP and administratively scoped unreliable multicast UDP. Provisions are made for the use of intermediary arbiters, called subscription arbiters, which handle routing notifications to their intended destination.

1 Introduction

This document provides for the sending of HTTP Notifications using administratively scoped unreliable Multicast UDP. Multicast UDP is useful in that it allows a single notification to be delivered to a potentially large group of recipients using only a single request.

However administrative scoped unreliable Multicast UDP suffers from a number of problems including: how to route it, how to handle firewalling and how to deal with multiple scopes. This document only addresses the use of Multicast UDP within a single administrative scope and only countenances its use for scenarios where there is no notification proxy.

In order to allow for notifications to be distributed beyond a single administrative multicast scope it is necessary to provide for relaying arbiters. These arbiters, called subscription arbiters, are able to form, through an unspecified mechanism, relationships with other subscription arbiters in order to distribute notifications.
This allows a client to send a single HTTP notification to its subscription arbiter and have that notification forwarded on to one or more recipients. It is the subscription arbiter, not the client, who is responsible for maintaining the list of potential recipients and distributing notifications to those recipients.

In order for subscription arbiters to know to whom to distribute notifications clients who wish to receive notifications, known as subscribers, must subscribe to the subscription arbiter.

2 Definitions

2.1 Event

Any occurrence that may potentially trigger a notification.

2.2 Subscription

An established relationship in which a resource has indicated interest in receiving certain notifications.

2.3 Subscriber

A resource that negotiates a subscription with a subscription arbiter.

2.4 Implied Subscriber

A resource that did not negotiate a subscription with a subscription arbiter but will still be notified of events on that arbiter.

2.5 Notifying Resource

A resource that issues notifications, for example, a subscription arbiter.

2.6 Subscription Arbiter

A resource that accepts subscriptions, receives notifications and forwards those notifications to its subscribers.

2.7 Notification

A message sent by a notifying resource to provide notification of an event.

2.8 Notification Type

A mechanism to classify notifications into categories. This allows subscribers to specify exactly what class of notifications they want to receive. It also allows notifying resources to specify what class of notification they are sending out.
Notification types do not necessarily identify a single event but rather identify a group of related notifications. The notification sub-type is used to specify a particular event.

2.9 Notification Sub-Type

Identification of a particular event within a notification type.

For example, the fictional notification of type home:doors may contain notification sub-types such as door:open, close:door, etc.

There is no requirement that the URI identifying the notification type and the URI identifying the notification sub-type have a syntactic relationship, only a semantic one.

2.10 Subscription ID

A unique identifier for a subscription. Subscription IDs MUST be URIs and MUST be unique across all subscriptions across all resources for all time.

2.11 Scope

Scopes are used in a subscription to indicate the notifying resource the subscriber is interested in.

3 Notification Model

The notification model for GENA is based on the following picture:

[Subscriber] <- [1+ Subscription Arbiters] <- [Notifying Resource]
               Notification Request        Notification Request
                   ->                         Subscription Request

Subscribers send subscription requests to their subscription arbiter. The arbiter will then forward to the subscriber any notifications it receives which match the subscriber’s subscription.

Notifying resources send notifications to their subscription arbiter to be passed on to subscribers.

Subscription arbiters communicate with each other in order to pass along notifications and subscriptions. Subscription arbiter to subscription arbiter communication is out of scope for this specification.

For the purposes of this protocol all communication is between subscribers/notifying resources and their subscription arbiter. This does not preclude direct communication between a subscriber and a
notifying resource. Rather it means that the notifying resource is
acting as a subscription arbiter.

This document also deals with a degenerate case where no
subscription arbiter is available but administratively scoped
unreliable multicast UDP facilities are. In that case provisions are
made to allow a notifying resource to send its notifications
directly to a previously agreed upon administratively scoped
multicast UDP address where interested resources can listen in to
the notification.

3.1 Sending HTTP Notifications through a Subscription Arbiter

A notifying resource finds its subscription arbiter through an
unspecified mechanism. The notifying resource will send all of its
notifications to the subscription arbiter who will then forward
those subscriptions on to subscribers.

This document does not provide a mechanism by which the notifying
resource can retrieve information about which resources have
subscribed to receive notifications from the notifying resource.

3.2 Receiving HTTP Notifications through a Subscription Arbiter

A subscribing resource finds its subscription arbiter through an
unspecified mechanism. It is the responsibility of the subscribing
resource to send subscription requests to the subscription arbiter
in order to inform the arbiter as to which notifications the
subscriber would like to receive.

A subscription request can be thought of as a persistent search
filter on the set of notifications that the subscription arbiter is
aware of. Whenever the subscription arbiter receives a notification
that matches the search filter it will forward the notification to
the subscriber.

This document defines a very basic search filter that allows a
subscribing resource to specify a particular resource and a type of
notification the subscribing resource is interested in. Whenever a
notification of the specified type is made by the specified resource
the subscription arbiter will forward the notification to the
subscriber.

4 Subscription Arbiters and Forwarded Notifications

When forwarding a notification the subscription arbiter will change
the Request-URI and the Host header value to match the subscriber
who is to be notified. Subscription arbiters MUST NOT make any other
changes to be made to the message unless the definition of the
header or body element specifically provides for such alteration
and/or for security reasons.
5 NOTIFY HTTP Method

The NOTIFY method is used to transmit a notification. The Request-URI of the notification method is the notifying resource’s subscription arbiter who will handle forwarding the message to interested subscribers.

The NOTIFY method may be sent using httpu or httpmu as specified in [HTTPUDP]. In the case of httpmu the multicast channel itself is treated as the subscription arbiter. NOTIFY methods sent using httpmu do not have responses.

The NOTIFY method MUST contain a NT header and MAY contain a body, a NTS header and SID. The NT header of a NOTIFY request MUST NOT contain more than one URI. Subscribers MAY ignore the body in a subscription request. Subscription arbiters MAY remove and/or alter the value of the SID header in order to set it to the value that their subscriber is expecting. Note that in general notifying resources will not put SID headers on their notifications. This is generally a value that subscription arbiters add.

Note that notifications to implied subscribers may not necessarily have SIDs. The client can tell the subscription arbiter to stop sending the notification by returning a 412 (Precondition Failed).

A subscription arbiter which sends a NOTIFY method to a subscriber and gets back a 404 (Not Found) or 410 (Gone) MAY end the subscription. The subscription arbiter is not required to remember all the values in the callback header using in the SUBSCRIPTION request and so is not required to fallback to one of the values listed there in the case the current one fails. However, nothing prevents a subscription arbiter from providing this service.

5.1 Response Codes

200 (OK) - This is the standard response to a NOTIFY received by a subscriber.

202 (Accepted) - This is the standard response to a NOTIFY received by a subscription arbiter.

412 (Precondition Failed) - The client doesn’t recognize the SID or the request doesn’t have a SID and the client doesn’t want to receive the notification.

5.2 Examples

5.2.1 TCP/IP

NOTIFY /foo/bar HTTP/1.1
Host: blah:923
NT: ixl:pop

Cohen et al.
NTS: clock:bark
Timeout: Second-10003
SID: uuid:kj9d4f5ae-7dec-11d0-a765-00a0c91e6bf6

HTTP/1.1 200 O.K.

A notification of type ixl:pop sub-type clock:bark has been sent out in response to the specified subscription. The request-URI could either identify a particular resource who is to be notified or a subscription arbiter who will then take responsibility for forwarding the notification to the appropriate subscribers.

5.2.2 Multicast UDP

NOTIFY * HTTP/1.1
Host: somemulticastIPaddress:923
Timeout: Second-159
NT: ixl:pop
NTS: clock:bark

As in the previous example this is a notification of type ixl:pop sub-type clock:bark but it has been sent out to the multicast channel as an unsolicited notification. Hence it does not have a SID header. Also, because it was sent out to a multicast UDP channel it also doesn't have a response.

6 SUBSCRIBE HTTP Method

The SUBSCRIBE method is used to provide a subscription arbiter with a search filter to be used in determining what notifications to forward to the subscriber.

The Request-URI of the SUBSCRIBE method specifies the subscription arbiter which will handle the subscription.

A SUBSCRIBE request MUST have a NT header unless it is a re-subscription request. The NT header specifies what sort of notification the subscriber wishes to be notified of.

A SUBSCRIBE request MUST have a Callback header unless it is a re-subscription request. The Callback header specifies how the subscriber is to be contacted in order to deliver the notification.

A NTS header on a SUBSCRIBE method MUST be ignored. The base subscription search filter only supports filtering on the NT value of a notification. This limitation is meant to keep the subscription functionality at the minimum useful level. It is expected that future specifications will provide for more flexible subscription search filters.
A SUBSCRIBE method MUST have a scope header unless it is a re-subscription request. The scope header identifies the resource that the subscriber wishes to receive notifications about.

The Timeout request header, whose syntax is defined in section 9.8 of [RFC2518] MAY be used on a SUBSCRIBE request. The header is used to request that a subscription live for the specified period of time before having to be renewed. Subscription arbiters are free to ignore this header.

A subscription arbiter MUST ignore the body of a SUBSCRIBE request if it does not understand that body.

If a subscription is successful then the subscription arbiter is responsible for returning notifications of the type specified in the NT header on the resource listed in the scope header.

Notifications sent out as a result of a subscription MUST include a SID header set to the identifier of the subscription that caused the notification to be sent as well as a Timeout header identifying when the subscription will expire.

Subscription arbiters MUST support callback URLs of type http.

A successful response to the SUBSCRIBE method over http MUST include a Timeout response header and a SID header.

6.1 Re-Subscribing

When the period of time specified in the Timeout response header passes the subscription MAY expire. In order to keep the subscription alive the subscriber MUST issue a SUBSCRIBE method with a SID header set to the subscription to be re-subscribed. A re-subscribe request MUST NOT have a NT header but it MAY have a Timeout and/or a Callback header.

Note that the value in the Timeout response header will not take into account the time needed from when the value was generated until it was passed through the arbiter, put on the wire, sent to the subscriber, parsed by the subscriber's system and finally passed to the subscriber's program. Hence the value should be taken as an absolute upper bound. Subscribers are encouraged to re-subscribe a good period of time before the actual expiration of the subscription.

6.2 Response Codes

200 (OK) - The subscription request has been successfully processed and a subscription ID assigned.

400 Bad Request - A required header is missing.
412 Precondition Failed - Either the subscription arbiter doesn’t support any of the Callbacks, doesn’t support one or more of the NTs or doesn’t support one or more of the scopes.

6.3 Examples

6.3.1 Subscription

SUBSCRIBE dude HTTP/1.1
Host: iamthedude:203
NT: ixl:pop
Callback: <http://blah/bar:923>
Scope: http://icky/pop
Timeout: Infinite

HTTP/1.1 200 O.K.
Subscription-ID: uuid:kj9d4fae-7dec-11d0-a765-00a0c91e6bf6
Timeout: Second-604800

This subscription request asks the subscription arbiter http://iamthedude/dude:203 for a subscription on notifications of type ixl:pop from the resource http://icky/pop.

6.3.2 Re-Subscription

SUBSCRIBE dude HTTP/1.1
Host: iamthedude:203
Subscription-ID: uuid:kj9d4fae-7dec-11d0-a765-00a0c91e6bf6
Timeout: Infinite

HTTP/1.1 200 O.K.
Subscription-ID: uuid:kj9d4fae-7dec-11d0-a765-00a0c91e6bf6
Timeout: Second-604800

The subscription has been successfully renewed.

7 UNSUBSCRIBE HTTP Method

The UNSUBSCRIBE method is used to terminate a subscription. The UNSUBSCRIBE method MUST include a SID header with the value of the subscription to be un-subscribed.

If the SID identifies a subscription that the subscription arbiter does not recognize or knows is already expired then the arbiter MUST respond with a 200 (OK).

7.1 Example

UNSUBSCRIBE dude HTTP/1.1
Host: iamtheproxy:203
SID: uuid:kj9d4fae-7dec-11d0-a765-00a0c91e6bf6
HTTP/1.1 200 O.k.

8  New HTTP Headers

8.1 NT Header

The NT header is used to indicate the notification type.

NT = "NT" ":" absoluteURI ; See section 3 of [RFC2396]

8.2 NTS Response Header

The NTS response header is used to indicate the notification sub-type of a notification.

NTS = "NTS" ":" absoluteURI

8.3 Callback Header

The Callback header specifies, in order of preference, the means the subscriber would like the subscription arbiter to use to deliver notifications.

Callback = "Callback" ":" *Coded-URI; See section 9.4 of [RFC2518]

8.4 Timeout Response Header

The Timeout response header has the same syntax as the Timeout request header defined in section 9.8 of [RFC2518]. The subscription arbiter informs the subscriber how long the subscription arbiter will keep their subscription active without a re-subscribe using the Timeout response header.

8.5 SID Header

The SID header contains a subscription ID.

SID = "SID" ":" absoluteURI

8.6 Scope Request Header

The scope request header indicates the resources the subscriber wishes to receive notifications about.

SCOPE = "Scope" ":" absoluteURI

9  Future Work

This specification defines a minimally useful set of notification functionality. It does not, however, address three critical issues that are needed by some notification environments. It is expected
that all of these features can be provided in extension specifications to this base specification.

The first issue is polling. In some environments, especially those with intermittent connectivity, it would be desirable for subscription arbiters to be able to pool up notifications and then to deliver them when the subscriber asks for them.

The second issue is subscription arbiter to subscription arbiter communication. It is likely that subscription arbiters will want to communicate directly with each other in order to efficiently distribute notifications and subscriptions. This requires provision for notification routing and loop prevention.

The third issue is support for depth functionality. In some systems one wishes to receive a notification about a resource and any of its children as the term is defined in [RFC2518].

10 Security Considerations

TBD.

[Notes:
The really horrible security concerns don’t start until you build the subscription arbiter to arbiter protocol. Otherwise the arbiter is very close to a proxy in that it takes confidential information from a subscriber and/or notifying resource and is expected to do the right thing (TM) with it. Authentication and such prevents bogus notifications and subscriptions.]

11 IANA Considerations

None.

12 Copyright

The following copyright notice is copied from RFC 2026 [Bradner, 1996], Section 10.4, and describes the applicable copyright for this document.

Copyright (C) The Internet Society February 10, 1998. All Rights Reserved.

This document and translations of it may be copied and furnished to others, and derivative works that comment on or otherwise explain it or assist in its implementation may be prepared, copied, published and distributed, in whole or in part, without restriction of any kind, provided that the above copyright notice and this paragraph are included on all such copies and derivative works. However, this document itself may not be modified in any way, such as by removing the copyright notice or references to the Internet Society or other Internet organizations, except as needed for the purpose of
developing Internet standards in which case the procedures for copyrights defined in the Internet Standards process must be followed, or as required to translate it into languages other than English.

The limited permissions granted above are perpetual and will not be revoked by the Internet Society or its successors or assignees.

This document and the information contained herein is provided on an "AS IS" basis and THE INTERNET SOCIETY AND THE INTERNET ENGINEERING TASK FORCE DISCLAIMS ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY WARRANTY THAT THE USE OF THE INFORMATION HEREIN WILL NOT INFRINGE ANY RIGHTS OR ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

13 Intellectual Property

The following notice is copied from RFC 2026 [Bradner, 1996], Section 10.4, and describes the position of the IETF concerning intellectual property claims made against this document.

The IETF takes no position regarding the validity or scope of any intellectual property or other rights that might be claimed to pertain to the implementation or use other technology described in this document or the extent to which any license under such rights might or might not be available; neither does it represent that it has made any effort to identify any such rights. Information on the IETF's procedures with respect to rights in standards-track and standards-related documentation can be found in BCP-11. Copies of claims of rights made available for publication and any assurances of licenses to be made available, or the result of an attempt made to obtain a general license or permission for the use of such proprietary rights by implementers or users of this specification can be obtained from the IETF Secretariat.

The IETF invites any interested party to bring to its attention any copyrights, patents or patent applications, or other proprietary rights which may cover technology that may be required to practice this standard. Please address the information to the IETF Executive Director.

14 References


15 Authors’ Addresses

Josh Cohen, Sonu Aggarwal, Yaron Y. Goland
Microsoft Corporation
One Microsoft Way
Redmond, WA 98052-6399

Email: {joshco,sonuag,yarong}@microsoft.com