Network Working Group
Request for Comments: 5573
Category: Experimental

M. Thomson
Andrew

June 2009

Asynchronous Channels for the Blocks Extensible Exchange Protocol (BEEP)

Status of This Memo

Copyright Notice

Abstract

The Blocks Extensible Exchange Protocol (BEEP) provides a protocol framework for the development of application protocols. This document describes a BEEP feature that enables asynchrony for individual channels.

Table of Contents

1. Introduction

The Blocks Extensible Exchange Protocol (BEEP) provides a protocol framework that manages many of the aspects necessary in developing an application protocol: framing, encoding, privacy, authentication, and asynchrony. However, the asynchrony provided by BEEP is limited to asynchrony between channels; replies to messages sent on any channel are strictly ordered.

Serial processing behavior is desirable for a range of applications. However, serial processing is less suitable for applications that rely more heavily on asynchrony. In particular, if a response takes a significant amount of time to create, the channel is effectively blocked until the request has been processed and the response sent. Pipelining only ensures that network latency does not add to this time; subsequent requests cannot be processed until a response is made to the first request.

Asynchronous applications require a protocol that is able to support a large number of concurrent outstanding requests. The analogy of a channel as a thread does not scale to the large number of threads used in modern systems. Modern applications regularly have large numbers of concurrent processing threads. Thus, a better way of multiplexing large numbers of concurrent requests is required.

This document describes a BEEP feature, an extension to BEEP, that enables the creation of an asynchronous channel. An asynchronous channel is a channel where response ordering is not fixed to the order of the requests sent by the client peer. An asynchronous channel is identical to other channels, using unmodified framing; except that requests may be processed in parallel and responses may be sent in any order.

An asynchronous channel enables the efficient use of a single channel for multiple concurrent requests. There is no impact on requests arising from the timing of responses to other requests. The requesting peer can process responses to the requests it sends as they come available; similarly, the serving peer can take advantage of parallel processing without artificial constraints on the order of responses.

Asynchronous channels allow for greater throughput where the serving peer requires any time to process requests. This is particularly relevant where the serving peer needs to perform lengthy computations or make network-based requests as a part of servicing the request.

BEEP feature negotiation is used to ensure that both peers are mutually willing to create asynchronous channels. A means for establishing an asynchronous channel is described.

This document is published as an Experimental RFC in order to find out whether the extension is going to be deployed for use in a variety of use cases and applications.

2. Asynchronous BEEP Channels

This document defines a BEEP feature that enables the use of asynchronous channels. An asynchronous channel is a BEEP channel that is not subject to the restrictions of Section 2.6.1 of [RFC3080] regarding ordering of responses; requests can be processed and responded to in any order by the serving peer.

Asynchronous channels use the "msgno" element of the BEEP frame header to correlate request and response. Regular BEEP channels do not use "msgno" for request/response correlation, contrary to what might be inferred by the presence of the parameter. In a regular BEEP channel, the "msgno" only serves as a means of checking for protocol errors.

Asynchronous channels are not suitable for applications where state established by requests is relied upon in subsequent requests or the ordering of messages is significant.

2.1. Asynchronous Feature

The "feature" attribute in the BEEP greeting contains a whitespace-separated list of features supported by each peer. If both lists contain the same feature, that feature may be used by either peer.

This document registers the feature "async". If either peer does not include this feature in the greeting message, neither peer is able to create an asynchronous channel.

Figure 1 shows an example exchange where both peers declare willingness to use this feature.

```
L: <wait for incoming connection>
I: <open connection>
L: RPY 0 0 . 0 133
L: Content-Type: application/beep+xml
L:
L: <greeting features="async x-foo">
L:    <profile uri="http://example.com/beep/APP" />
L: </greeting>
L: END
I: RPY 0 0 . 0 69
I: Content-Type: application/beep+xml
I:
I: <greeting features="async" />
I: END
```

Figure 1: BEEP Greetings with Asynchronous Feature

The registration template for BEEP features is included in Section 5.

2.2. Starting an Asynchronous Channel

To create an asynchronous channel, an "async" parameter set to "true" is included in the "start" request. If omitted, or set to "false", the channel is not asynchronous.

Figure 2 shows how the "async" attribute can be used to start an asynchronous channel.

```
C: MSG 0 1 . 52 130
C: Content-Type: application/beep+xml
C:
C: <start number="1" async="true">
C:    <profile uri="http://example.org/protocol"/>
C: </start>
C: END
S: RPY 0 1 . 221 102
S: Content-Type: application/beep+xml
S:
S: <profile uri="http://example.org/protocol"/>
S: END
```

Figure 2: Asynchronous Channel Start

If the serving peer is unable to create an asynchronous channel for any reason, the channel start is rejected. This could occur if the selected profile is not suitable for an asynchronous channel. The response can include the "553" response code (parameter invalid) and an appropriate message, as shown in Figure 3.

```
C: MSG 0 1 . 52 128
C: Content-Type: application/beep+xml
C:
C: <start number="1" async="true">
C:    <profile uri="http://example.org/serial"/>
C: </start>
C: END
S: ERR 0 1 . 221 152
S: Content-Type: application/beep+xml
S:
S: <error code="553">Profile &lt;http://example.org/serial&gt;
S: cannot be used for asynchronous channels.</error>
S: END
```

Figure 3: Asynchronous Channel Start Error

2.3. Asynchronous Channel Behavior

Asynchronous channels differ from normal BEEP channels in one way only: an asynchronous channel is not subject to the restrictions in Section 2.6.1 of [RFC3080] regarding the processing and response ordering. A peer in the serving role may process and respond to requests in any order it chooses.

In an asynchronous channel, the "msgno" element of the frame header is used to correlate request and response. A BEEP peer receiving responses in a different order than the requests that triggered them must not regard this as a protocol error.

"MSG" messages sent on an asynchronous channel may be processed in parallel by the serving peer. Responses ("RPY", "ANS", "NUL", or "ERR" messages) can be sent in any order. Different "ANS" messages that are sent in a one-to-many exchange may be interleaved with responses to other "MSG" messages.

An asynchronous channel must still observe the rules in [RFC3080] regarding segmented messages. Each message must be completed before any other message can be sent on that same channel.

Note: An exception to this rule is made in [RFC3080] for interleaved "ANS" segments sent in response to the same "MSG". It is recommended that BEEP peers do not generate interleaved ANS segments.

The BEEP management channel (channel 0) is never asynchronous.

2.4. Error Handling

BEEP does not provide any mechanism for managing a peer that does not respond to a request. Synchronous channels cannot be used or even closed if a peer does not provide a response to a request. The only remedy available is closing the underlying transport. While an asynchronous channel cannot be closed, it can still be used for further requests. However, any outstanding request still consumes state resources. Client peers may dispose of such state after a configured interval, but must be prepared to discard unrecognized responses if they do so.

3. Alternatives

The option presented in this document provides for asynchronous communication. Asynchronous channels might not be applicable in every circumstance, particularly where ordering of requests is significant. Depending on application protocol requirements, the alternatives discussed in this section could be more applicable.

3.1. Increasing Throughput

In some cases, asynchronous channels can be used to remove limitations on message processing throughput. Alternatively, pipelining of requests can increase throughput significantly where network latency is the limiting factor. Spreading requests over several channels increases overall throughput, if throughput is the only consideration.

Note: Be wary of false optimizations that rely on the pipelining of requests. If later requests in a series of pipelined requests rely on state established by earlier requests, errors in earlier requests could invalidate later requests.

The flow control window used in the TCP mapping [RFC3081] can introduce a limiting factor in throughput for individual channels. Choice of TCP window size similarly limits throughput, see [RFC1323]. To avoid limitations introduced by flow control, receiving peers can increase the window size used; sending peers can open additional channels with the same profile. Correctly managing flow control also applies to asynchronous channels.

3.2. Asynchrony in the Application Protocol

With changes to the application protocol, serial channels can be used for asynchronous exchanges. Asynchrony can be provided at a protocol layer above BEEP by separating request and response. This requires the addition of proprietary MIME headers or modifications to the application protocol.

The serving peer provides an immediate "RPY" (or "NUL") response to requests. This frees the channel for further requests. The actual response is sent as a separate "MSG" using a special identifier included in the original request to correlate the response with the request. This second "MSG" can be sent on the same channel (since these are full duplex) or on a channel specifically created for this purpose.

This method is not favored since it requires that the application protocol solve the problem of correlating request with response. BEEP aims to provide a general framework for the creation of an application protocol, and for it to not provide request/response correlation would limit its usefulness. Using a MIME header is also possible, but using "msgno" is the most elegant solution.

4. Security Considerations

Enabling asynchronous messaging for a channel potentially requires the maintenance of additional state information. A peer in the server role that does not reply to messages can cause the accumulation of state at the client peer. If this state information were not limited, this mode could be used to perform denial of service. This problem, while already present in BEEP, is potentially more significant due to the nature of the processing on the serving peer that might occur for requests received on an asynchronous channel. The extent to which denial is possible is limited by what a serving peer accepts; the number of outstanding requests can be restricted to protect against excessive accumulation of state.

A client peer maintains state for each request that it sends. A client peer should enforce a configurable limit on the number of requests that it will allow to be outstanding at any time. This limit could be enforced at channel, connection, or application scope. Once this limit is reached, the client peer might prevent or block further requests from been generated.

Peers that serve requests on asynchronous channels also accumulate state when a request is accepted for processing. Peers in the serving role may similarly limit to the number of requests that are processed concurrently. Once this limit is reached the receiving

peer can either stop reading new requests, or might start rejecting such requests by generating error responses. Alternatively, the flow control [RFC3081] can be used; "SEQ" frames can be suppressed, allowing the flow control window to close and preventing the receipt of further requests.

5. IANA Considerations

This section registers the BEEP "async" feature in the BEEP parameters registry, following the template from Section 5.2 of [RFC3080].

Feature Identification: async

Feature Semantics: This feature enables the creation of asynchronous channels, see Section 2 of RFC 5573.

Contact Information: Martin Thomson <martin.thomson@andrew.com>

6. References

6.1. Normative References

[RFC3080] Rose, M., "The Blocks Extensible Exchange Protocol Core", RFC 3080, March 2001.

6.2. Informative References

[RFC3081] Rose, M., "Mapping the BEEP Core onto TCP", RFC 3081, March 2001.

[RFC1323] Jacobson, V., Braden, B., and D. Borman, "TCP Extensions for High Performance", RFC 1323, May 1992.

Author's Address

Martin Thomson
Andrew
PO Box U40
Wollongong University Campus, NSW 2500
AU

Phone: +61 2 4221 2915
EMail: martin.thomson@andrew.com
URI: http://www.andrew.com/