Network Working Group   B. Carpenter
Internet-Draft   Univ. of Auckland
Intended status: Informational   S. Jiang
Expires: January 1, 2019   B. Liu
                                             Huawei Technologies Co., Ltd
                                                          June 30, 2018

Transferring Bulk Data over the GeneRic Autonomic Signaling Protocol (GRASP)
draft-carpenter-anima-grasp-bulk-02

Abstract

   This document describes how bulk data may be transferred between
   Autonomic Service Agents via the GeneRic Autonomic Signaling Protocol
   (GRASP).

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

The document [I-D.liu-anima-grasp-distribution] discusses how
information may be distributed within the secure Autonomic Networking
Infrastructure (ANI) [I-D.ietf-anima-reference-model]. Specifically,
it describes using the Synchronization and Flood Synchronization
mechanisms of the GeneRic Autonomic Signaling Protocol (GRASP)
[I-D.ietf-anima-grasp] for this purpose. However, those mechanisms
are limited to distributing GRASP Objective Options contained in
messages that cannot exceed the GRASP maximum message size of 2048
bytes.

There are scenarios in autonomic networks where this restriction is a
problem. One example is the distribution of network policy in
lengthy formats such as YANG or JSON. Another case might be an
Autonomic Service Agent (ASA) uploading a log file to the Network
Operations Center (NOC). A third case might be a supervisory system
downloading a software upgrade to an autonomic node. A related case
might be installing the code of a new or updated ASA to a target node
(see the discussion of ASA life cycles in
[I-D.carpenter-anima-asa-guidelines]).

Naturally, an existing solution such as a secure file transfer
protocol or secure HTTP might be used for this. Other management
protocols such as syslog [RFC5424] or NETCONF [RFC6241] might also be
used for related purposes, or might be mapped directly over GRASP.
The present document, however, applies to any scenario where it is

preferable to re-use the autonomic networking infrastructure itself
to transfer a significant amount of data, rather than install and
configure an additional mechanism. The basic model is to use the
GRASP Negotiation process to transfer and acknowledge multiple blocks
of data in successive negotiation steps.

The emphasis is placed on simplicity rather than efficiency, high
throughput, or advanced functionality. For example, if a transfer
gets out of step or data packets are lost, the strategy is to abort
the transfer and try again. In an enterprise network with low bit
error rates, and with GRASP running over TCP, this is not considered
a serious issue. Clearly, a more sophisticated approach could be
designed but if the application requires that, existing protocols
could be used, as indicated in the preceding paragraph.

NOTE: This is an early draft of a solution. As the specification
becomes more mature, the authors expect it to become precise enough
to be placed on the standards track.

2. General Method for Bulk Transfer

As for any GRASP operation, the two participants are considered to be
Autonomic Service Agents (ASAs) and they communicate using a specific
GRASP Objective Option, containing its own name, some flag bits, a
loop count, and a value. In bulk transfer, we can model the ASA
acting as the source of the transfer as a download server, and the
destination as a download client. No changes or extensions are
required to GRASP itself, but compared to a normal GRASP negotiation,
the communication pattern is slightly asymmetric:

1. The client first discovers the server by the GRASP discovery
    mechanism (M_DISCOVERY and M_RESPONSE messages).

2. The client then sends a GRASP negotiation request (M_REQ_NEG
    message). The value of the objective expresses the requested
    item (e.g., a file name - see the next section for a detailed
    example).

3. The server replies with a negotiation step (M_NEGOTIATE message).
    The value of the objective is the first section of the requested
    item (e.g., the first block of the requested file as a raw byte
    string).

4. The client replies with a negotiation step (M_NEGOTIATE message).
    The value of the objective is a simple acknowledgement (e.g., the
    text string 'ACK').

The last two steps repeat until the transfer is complete. The server signals the end by transferring an empty byte string as the final value. In this case the client responds with a normal end to the negotiation (M_END message with an O_ACCEPT option).

Errors of any kind are handled with the normal GRASP mechanisms, in particular by an M_END message with an O_DECLINE option in either direction.

The block size must be chosen such that each step does not exceed the GRASP message size limit of 2048 bits.

This approach is safe since each block must be positively acknowledged, and data transfer errors will be detected by TCP. If a future variant of GRASP runs over UDP, the mandatory UDP checksum for IPv6 will detect such errors. The method does not specify retransmission for failed blocks, so a failed transfer will need to be restarted.

An observant reader will notice that the GRASP loop count mechanism, intended to terminate endless negotiations, will cause a problem for large transfers. For this reason, both the client and server must artificially increment the loop count by 1 before each negotiation step.

If network load is a concern, the data rate can be limited by inserting a delay before each negotiation step, with the GRASP timeout set accordingly. Either the server or the client, or both, could insert such a delay. Also, either side could use the GRASP Confirm Waiting (M_WAIT) message to slow the other side down.

The description above concerns bulk download from a server (responding ASA) to a client (requesting ASA). The data transfer could also be in the opposite (upload) direction with minor modifications to the procedure: the client would send the file name and the data blocks, and the server would send acknowledgements.

3. Example for File Transfer

This example describes a client ASA requesting a file download from a server ASA.

Firstly we define a GRASP objective informally:

["411:mvFile", 3, 6, value]

The formal CDDL definition [I-D.ietf-cbor-cddl] is:

    mvfile-objective = ["411:mvFile", objective-flags, loop-count, value]

    objective-flags = ; as in the GRASP specification
    loop-count = ; as in the GRASP specification
    value = any

    The objective-flags field is set to indicate negotiation.

    Dry run mode must not be used.

    The loop-count is set to a suitable value to limit the scope of
    discovery. A suggested default value is 6.

    The value takes the following forms:

    o In the initial request from the client, a UTF-8 string containing
       the requested file name (with file path if appropriate).

    o In negotiation steps from the server, a byte string containing at
       most 1024 bytes. However:

       * If the file does not exist, the first negotiation step will
          return an M_END, O_DECLINE response.

       * After sending the last block, the next and final negotiation
          step will send an empty byte string as the value.

    o In negotiation steps from the client, the value is the UTF-8
       string 'ACK'.

    Note that the block size of 1024 is chosen to guarantee not only that
    each GRASP message is below the size limit, but also that only one
    TCP data packet will be needed, even on an IPv6 network with a
    minimum link MTU.

    We now present outline pseudocode for the client and the server ASA.
    The API documented in [I-D.ietf-anima-grasp-api] is used in a
    simplified way, and error handling is not shown in detail.

Pseudo code for client ASA (request and receive a file):

```
requested_obj = objective('411:mvFile')
locator = discover(requested_obj)
requested_obj.value = 'etc/test.pdf'
received_obj = request_negotiate(requested_obj, locator)
if error_code == declined:
    #no such file
    exit

file = open(requested_obj.value)
file.write(received_obj.value) #write to file
eof = False
while not eof:
    received_obj.value = 'ACK'
    received_obj.loop_count = received_obj.loop_count + 1
    received_obj = negotiate_step(received_obj)
    if received_obj.value == null:
        end_negotiate(True)
        file.close()
        eof = True
    else:
        file.write(received_obj.value) #write to file

#file received
exit
```

Pseudo code for server ASA (await request and send a file):

```
supported_obj = objective('411:mvFile')
requested_obj = listen_negotiate(supported_obj)
file = open(requested_obj.value) #open the source file
if no such file:
    end_negotiate(False) #decline negotiation
    exit

eof = False
while not eof:
    chunk = file.read(1024) #next block of file
    requested_obj.value = chunk
    requested_obj.loop_count = requested_obj.loop_count + 1
    requested_obj = negotiate_step(requested_obj)
    if chunk == null:
        file.close()
        eof = True
        end_negotiate(True)
        exit
    if requested_obj.value != 'ACK':
        #unexpected reply...
```

4. Loss Detection

The above description and example assume that GRASP is implemented
over a reliable transport layer such as TCP, such that lost or
corrupted messages are not likely. Rarely, an error might be
detected via a missing ACK, in which case the transfer would be
aborted and restarted. In the event that GRASP is implemented over
an unreliable transport layer such as UDP, it would be possible to
add a block number to both the data block and acknowledgement
objectives, so that missing blocks can be retransmitted, or duplicate
blocks can be ignored. For example, the objective in Section 3 would
become:

```
mvfile-objective = ["411:mvFile", objective-flags, loop-count, value]

objective-flags = ; as in the GRASP specification
loop-count = ; as in the GRASP specification
value = [block-number, any]
block-number = uint
```

It would also be necessary for the transport layer to detect data
errors, for example by enabling UDP checksums.

5. Maximum Transmission Unit

   In an IPv6 environment, a minimal MTU of 1280 bytes can be assumed,
   and assuming that high throughput is not a requirement, bulk
   transfers can be designed to match that MTU. However, there are
   environments where the underlying physical MTU is much smaller. For
   example, on an IEEE 802.15.4 network it may be less than 100 bytes
   [RFC4944]. In such a case, a bulk transfer solution has several
   choices:

   1. Accept the overhead of an adaptation layer, and therefore assume
       a network-layer MTU of 1280 bytes. Indeed, the presence of such
       an adaptation layer may be impossible to detect.

   2. Attempt to determine the actual MTU available without lower-layer
       fragmentation. This however will be impossible without using
       low-level functions of the socket interface.

   3. Attempt to determine a message size that provides optimum
       performance, by some sort of trial-and-error solution.

   These complexities suggest that using a GRASP-based mechanism is
   unlikely to be optimal in environments with a very small physical
   MTU.

6. Pipelining

   The above description and example descibe a simple handshake model
   where each block is acknowledged before the next block is sent. For
   the scenarios discussed in Section 1, this should be acceptable.
   Therefore we do not suggest adding a pipelining or windowing
   mechanism. If high throughput is required, a conventional file
   transfer protocol should be used.

7. Other Considerations

   If multiple transfers are requested simultaneously, each one will
   proceed as a separate GRASP negotiation session. The ASA acting as
   the server must be coded accordingly, like any ASA that needs to
   handle simultaneous sessions [I-D.carpenter-anima-asa-guidelines].

   Bulk transfer might become a utility function for use by various
   ASAs, such as those supporting YANG or JSON distribution, log file
   uploads, or code downloads. In this case some form of user space API
   for bulk transfer will be required. This could be in the form of an
   inter-process communication call between the ASA in question and the
   ASA implementing the bulk transfer mechanism. The details are out of
   scope for this document.

8. Security Considerations

All GRASP transactions are secured by the mandatory security
substrate required by [I-D.ietf-anima-grasp]. No additional security
issues are created by the application of GRASP described in this
document.

9. IANA Considerations

This document makes no request of the IANA.

10. Acknowledgements

Thanks to Joel Halpern and other members of the ANIMA WG.

11. References

11.1. Normative References

[I-D.ietf-anima-grasp]
          Bormann, C., Carpenter, B., and B. Liu, "A Generic
          Autonomic Signaling Protocol (GRASP)", draft-ietf-anima-
          grasp-15 (work in progress), July 2017.

[I-D.ietf-cbor-cddl]
          Birkholz, H., Vigano, C., and C. Bormann, "Concise data
          definition language (CDDL): a notational convention to
          express CBOR data structures", draft-ietf-cbor-cddl-02
          (work in progress), February 2018.

11.2. Informative References

[I-D.carpenter-anima-asa-guidelines]
          Carpenter, B., Ciavaglia, L., Jiang, S., and P. Pierre,
          "Guidelines for Autonomic Service Agents", draft-
          carpenter-anima-asa-guidelines-04 (work in progress),
          March 2018.

[I-D.ietf-anima-grasp-api]
          Carpenter, B., Liu, B., Wang, W., and X. Gong, "Generic
          Autonomic Signaling Protocol Application Program Interface
          (GRASP API)", draft-ietf-anima-grasp-api-01 (work in
          progress), March 2018.

   [I-D.ietf-anima-reference-model]
             Behringer, M., Carpenter, B., Eckert, T., Ciavaglia, L.,
             and J. Nobre, "A Reference Model for Autonomic
             Networking", draft-ietf-anima-reference-model-06 (work in
             progress), February 2018.

   [I-D.liu-anima-grasp-distribution]
             Liu, B., Jiang, S., Xiao, X., Hecker, A., and Z.
             Despotovic, "Information Distribution in Autonomic
             Networking", draft-liu-anima-grasp-distribution-05 (work
             in progress), February 2018.

   [RFC4944] Montenegro, G., Kushalnagar, N., Hui, J., and D. Culler,
             "Transmission of IPv6 Packets over IEEE 802.15.4
             Networks", RFC 4944, DOI 10.17487/RFC4944, September 2007,
             <https://www.rfc-editor.org/info/rfc4944>.

   [RFC5424] Gerhards, R., "The Syslog Protocol", RFC 5424,
             DOI 10.17487/RFC5424, March 2009,
             <https://www.rfc-editor.org/info/rfc5424>.

   [RFC6241] Enns, R., Ed., Bjorklund, M., Ed., Schoenwaelder, J., Ed.,
             and A. Bierman, Ed., "Network Configuration Protocol
             (NETCONF)", RFC 6241, DOI 10.17487/RFC6241, June 2011,
             <https://www.rfc-editor.org/info/rfc6241>.

Appendix A. Change log [RFC Editor: Please remove]

   draft-carpenter-anima-grasp-bulk-02, 2018-06-30:

   Update reference, fix TBDs.

   draft-carpenter-anima-grasp-bulk-01, 2018-03-03:

   Updates after IETF100 discussion.

   draft-carpenter-anima-grasp-bulk-00, 2017-09-12:

   Initial version.

Authors' Addresses

Brian Carpenter
Department of Computer Science
University of Auckland
PB 92019
Auckland   1142
New Zealand

Email: brian.e.carpenter@gmail.com

Sheng Jiang
Huawei Technologies Co., Ltd
Q14, Huawei Campus, No.156 Beiqing Road
Hai-Dian District, Beijing, 100095
P.R. China

Email: jiangsheng@huawei.com

Bing Liu
Huawei Technologies Co., Ltd
Q14, Huawei Campus
No.156 Beiqing Road
Hai-Dian District, Beijing 100095
P.R. China

Email: leo.liubing@huawei.com