              Session Initiation Protocol (SIP) Overload Control

Abstract

   Overload occurs in Session Initiation Protocol (SIP) networks when
   SIP servers have insufficient resources to handle all the SIP
   messages they receive. Even though the SIP protocol provides a
   limited overload control mechanism through its 503 (Service
   Unavailable) response code, SIP servers are still vulnerable to
   overload. This document defines the behavior of SIP servers involved
   in overload control and also specifies a loss-based overload scheme
   for SIP.

Copyright Notice

Table of Contents

1. Introduction

   As with any network element, a Session Initiation Protocol (SIP)
   [RFC3261] server can suffer from overload when the number of SIP
   messages it receives exceeds the number of messages it can process.
   Overload can pose a serious problem for a network of SIP servers.
   During periods of overload, the throughput of a network of SIP
   servers can be significantly degraded. In fact, overload may lead to
   a situation where the retransmissions of dropped SIP messages may
   overwhelm the capacity of the network. This is often called
   "congestion collapse".

   Overload is said to occur if a SIP server does not have sufficient
   resources to process all incoming SIP messages. These resources may
   include CPU processing capacity, memory, input/output, or disk
   resources.

   For overload control, this document only addresses failure cases
   where SIP servers are unable to process all SIP requests due to
   resource constraints. There are other cases where a SIP server can
   successfully process incoming requests but has to reject them due to
   failure conditions unrelated to the SIP server being overloaded. For
   example, a Public Switched Telephone Network (PSTN) gateway that runs
   out of trunks but still has plenty of capacity to process SIP
   messages should reject incoming INVITEs using a 488 (Not Acceptable
   Here) response [RFC4412]. Similarly, a SIP registrar that has lost
   connectivity to its registration database but is still capable of
   processing SIP requests should reject REGISTER requests with a 500
   (Server Error) response [RFC3261]. Overload control does not apply
   to these cases, and SIP provides appropriate response codes for them.

   The SIP protocol provides a limited mechanism for overload control
   through its 503 (Service Unavailable) response code. However, this
   mechanism cannot prevent overload of a SIP server, and it cannot
   prevent congestion collapse. In fact, the use of the 503 (Service
   Unavailable) response code may cause traffic to oscillate and shift
   between SIP servers, thereby worsening an overload condition. A
   detailed discussion of the SIP overload problem, the problems with
   the 503 (Service Unavailable) response code, and the requirements for
   a SIP overload control mechanism can be found in [RFC5390].

   This document defines the protocol for communicating overload
   information between SIP servers and clients so that clients can
   reduce the volume of traffic sent to overloaded servers, avoiding
   congestion collapse and increasing useful throughput. Section 4
   describes the Via header parameters used for this communication. The

general behavior of SIP servers and clients involved in overload
control is described in Section 5. In addition, Section 7 specifies
a loss-based overload control scheme.

This document specifies the loss-based overload control scheme
(Section 7), which is mandatory to implement for this specification.
In addition, this document allows other overload control schemes to
be supported as well. To do so effectively, the expectations and
primitive protocol parameters common to all classes of overload
control schemes are specified in this document.

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 [RFC2119].

In this document, the terms "SIP client" and "SIP server" are used in
their generic forms. Thus, a "SIP client" could refer to the client
transaction state machine in a SIP proxy, or it could refer to a user
agent client (UAC). Similarly, a "SIP server" could be a user agent
server (UAS) or the server transaction state machine in a proxy.
Various permutations of this are also possible, for instance, SIP
clients and servers could also be part of back-to-back user agents
(B2BUAs).

However, irrespective of the context these terms are used in (i.e.,
proxy, B2BUA, UAS, UAC), "SIP client" applies to any SIP entity that
provides overload control to traffic destined downstream. Similarly,
"SIP server" applies to any SIP entity that is experiencing overload
and would like its upstream neighbor to throttle incoming traffic.

Unless otherwise specified, all SIP entities described in this
document are assumed to support this specification.

The normative statements in this specification as they apply to SIP
clients and SIP servers assume that both the SIP clients and SIP
servers support this specification. If, for instance, only a SIP
client supports this specification and not the SIP server, then the
normative statements in this specification pertinent to the behavior
of a SIP server do not apply to the server that does not support this
specification.

3. Overview of Operations

   This section provides an overview of how the overload control
   mechanism operates by introducing the overload control parameters.
   Section 4 provides more details and normative behavior on the
   parameters listed below.

   Because overload control is performed hop-by-hop, the Via header
   parameter is attractive since it allows two adjacent SIP entities to
   indicate support for, and exchange information associated with,
   overload control [RFC6357]. Additional advantages of this choice are
   discussed in Section 10.1.1. An alternative mechanism using SIP
   event packages was also considered, and the characteristics of that
   choice are further outlined in Section 10.1.2.

   This document defines four new parameters for the SIP Via header for
   overload control. These parameters provide a mechanism for conveying
   overload control information between adjacent SIP entities. The "oc"
   parameter is used by a SIP server to indicate a reduction in the
   number of requests arriving at the server. The "oc-algo" parameter
   contains a token or a list of tokens corresponding to the class of
   overload control algorithms supported by the client. The server
   chooses one algorithm from this list. The "oc-validity" parameter
   establishes a time limit for which overload control is in effect, and
   the "oc-seq" parameter aids in sequencing the responses at the
   client. These parameters are discussed in detail in the next
   section.

4. Via Header Parameters for Overload Control

   The four Via header parameters are introduced below. Further context
   about how to interpret these under various conditions is provided in
   Section 5.

4.1. The "oc" Parameter

   This parameter is inserted by the SIP client and updated by the SIP
   server.

   A SIP client MUST add an "oc" parameter to the topmost Via header it
   inserts into every SIP request. This provides an indication to
   downstream neighbors that the client supports overload control.
   There MUST NOT be a value associated with the parameter (the value
   will be added by the server).

   The downstream server MUST add a value to the "oc" parameter in the
   response going upstream to a client that included the "oc" parameter
   in the request. Inclusion of a value to the parameter represents two

things. First, upon the first contact (see Section 5.1), addition of
a value by the server to this parameter indicates (to the client)
that the downstream server supports overload control as defined in
this document. Second, if overload control is active, then it
indicates the level of control to be applied.

When a SIP client receives a response with the value in the "oc"
parameter filled in, it MUST reduce, as indicated by the "oc" and
"oc-algo" parameters, the number of requests going downstream to the
SIP server from which it received the response (see Section 5.10 for
pertinent discussion on traffic reduction).

4.2. The "oc-algo" Parameter

   This parameter is inserted by the SIP client and updated by the SIP
   server.

   A SIP client MUST add an "oc-algo" parameter to the topmost Via
   header it inserts into every SIP request, with a default value of
   "loss".

   This parameter contains names of one or more classes of overload
   control algorithms. A SIP client MUST support the loss-based
   overload control scheme and MUST insert at least the token "loss" as
   one of the "oc-algo" parameter values. In addition, the SIP client
   MAY insert other tokens, separated by a comma, in the "oc-algo"
   parameter if it supports other overload control schemes such as a
   rate-based scheme [RATE-CONTROL]. Each element in the comma-
   separated list corresponds to the class of overload control
   algorithms supported by the SIP client. When more than one class of
   overload control algorithms is present in the "oc-algo" parameter,
   the client may indicate algorithm preference by ordering the list in
   a decreasing order of preference. However, the client cannot assume
   that the server will pick the most preferred algorithm.

   When a downstream SIP server receives a request with multiple
   overload control algorithms specified in the "oc-algo" parameter
   (optionally sorted by decreasing order of preference), it chooses one
   algorithm from the list and MUST return the single selected algorithm
   to the client.

   Once the SIP server has chosen a mutually agreeable class of overload
   control algorithms and communicated it to the client, the selection
   stays in effect until the algorithm is changed by the server.
   Furthermore, the client MUST continue to include all the supported
   algorithms in subsequent requests; the server MUST respond with the
   agreed-to algorithm until the algorithm is changed by the server.

The selection SHOULD stay the same for a non-trivial duration of time
to allow the overload control algorithm to stabilize its behavior
(see Section 5.8).

The "oc-algo" parameter does not define the exact algorithm to be
used for traffic reduction; rather, the intent is to use any
algorithm from a specific class of algorithms that affect traffic
reduction similarly. For example, the reference algorithm in
Section 7.2 can be used as a loss-based algorithm, or it can be
substituted by any other loss-based algorithm that results in
equivalent traffic reduction.

4.3. The "oc-validity" Parameter

This parameter MAY be inserted by the SIP server in a response; it
MUST NOT be inserted by the SIP client in a request.

This parameter contains a value that indicates an interval of time
(measured in milliseconds) that the load reduction specified in the
value of the "oc" parameter should be in effect. The default value
of the "oc-validity" parameter is 500 (milliseconds). If the client
receives a response with the "oc" and "oc-algo" parameters suitably
filled in, but no "oc-validity" parameter, the SIP client should
behave as if it had received "oc-validity=500".

A value of 0 in the "oc-validity" parameter is reserved to denote the
event that the server wishes to stop overload control or to indicate
that it supports overload control but is not currently requesting any
reduction in traffic (see Section 5.7).

A non-zero value for the "oc-validity" parameter MUST only be present
in conjunction with an "oc" parameter. A SIP client MUST discard a
non-zero value of the "oc-validity" parameter if the client receives
it in a response without the corresponding "oc" parameter being
present as well.

After the value specified in the "oc-validity" parameter expires and
until the SIP client receives an updated set of overload control
parameters from the SIP server, overload control is not in effect
between the client and the downstream SIP server.

4.4. The "oc-seq" Parameter

This parameter MUST be inserted by the SIP server in a response; it
MUST NOT be inserted by the SIP client in a request.

This parameter contains an unsigned integer value that indicates the sequence number associated with the "oc" parameter. This sequence number is used to differentiate two "oc" parameter values generated by an overload control algorithm at two different instants in time. "oc" parameter values generated by an overload control algorithm at time t and t+1 MUST have an increasing value in the "oc-seq" parameter. This allows the upstream SIP client to properly collate out-of-order responses.

   Note: A timestamp can be used as a value of the "oc-seq" parameter.

If the value contained in the "oc-seq" parameter overflows during the period in which the load reduction is in effect, then the "oc-seq" parameter MUST be reset to the current timestamp or an appropriate base value.

   Note: A client implementation can recognize that an overflow has occurred when it receives an "oc-seq" parameter whose value is significantly less than several previous values. (Note that an "oc-seq" parameter whose value does not deviate significantly from the last several previous values is symptomatic of a tardy packet. However, overflow will cause the "oc-seq" parameter value to be significantly less than the last several values.) If an overflow is detected, then the client should use the overload parameters in the new message, even though the sequence number is lower. The client should also reset any internal state to reflect the overflow so that future messages (following the overflow) will be accepted.

5. General Behavior

   When forwarding a SIP request, a SIP client uses the SIP procedures of [RFC3263] to determine the next-hop SIP server. The procedures of [RFC3263] take a SIP URI as input, extract the domain portion of that URI for use as a lookup key, query the Domain Name Service (DNS) to obtain an ordered set of one or more IP addresses with a port number and transport corresponding to each IP address in this set (the "Expected Output").

   After selecting a specific SIP server from the Expected Output, a SIP client determines whether overload controls are currently active with that server. If overload controls are currently active (and the "oc-validity" period has not yet expired), the client applies the relevant algorithm to determine whether or not to send the SIP request to the server. If overload controls are not currently active with this server (which will be the case if this is the initial contact with the server, the last response from this server had

"oc-validity=0", or the time period indicated by the "oc-validity"
parameter has expired), the SIP client sends the SIP message to the
server without invoking any overload control algorithm.

5.1. Determining Support for Overload Control

If a client determines that this is the first contact with a server,
the client MUST insert the "oc" parameter without any value and MUST
insert the "oc-algo" parameter with a list of algorithms it supports.
This list MUST include "loss" and MAY include other algorithm names
approved by IANA and described in corresponding documents. The
client transmits the request to the chosen server.

If a server receives a SIP request containing the "oc" and "oc-algo"
parameters, the server MUST determine if it has already selected the
overload control algorithm class with this client. If it has, the
server SHOULD use the previously selected algorithm class in its
response to the message. If the server determines that the message
is from a new client or a client the server has not heard from in a
long time, the server MUST choose one algorithm from the list of
algorithms in the "oc-algo" parameter. It MUST put the chosen
algorithm as the sole parameter value in the "oc-algo" parameter of
the response it sends to the client. In addition, if the server is
currently not in an overload condition, it MUST set the value of the
"oc" parameter to be 0 and MAY insert an "oc-validity=0" parameter in
the response to further qualify the value in the "oc" parameter. If
the server is currently overloaded, it MUST follow the procedures in
Section 5.2.

   Note: A client that supports the rate-based overload control
   scheme [RATE-CONTROL] will consider "oc=0" as an indication not to
   send any requests downstream at all. Thus, when the server
   inserts "oc-validity=0" as well, it is indicating that it does
   support overload control, but it is not under overload mode right
   now (see Section 5.7).

5.2. Creating and Updating the Overload Control Parameters

A SIP server provides overload control feedback to its upstream
clients by providing a value for the "oc" parameter to the topmost
Via header field of a SIP response, that is, the Via header added by
the client before it sent the request to the server.

Since the topmost Via header of a response will be removed by an
upstream client after processing it, overload control feedback
contained in the "oc" parameter will not travel beyond the upstream

SIP client. A Via header parameter therefore provides hop-by-hop
semantics for overload control feedback (see [RFC6357]) even if the
next-hop neighbor does not support this specification.

The "oc" parameter can be used in all response types, including
provisional, success, and failure responses (please see Section 5.11
for special consideration on transporting overload control parameters
in a 100 Trying response). A SIP server can update the "oc"
parameter in a response, asking the client to increase or decrease
the number of requests destined to the server or to stop performing
overload control altogether.

A SIP server that has updated the "oc" parameter SHOULD also add a
"oc-validity" parameter. The "oc-validity" parameter defines the
time in milliseconds during which the overload control feedback
specified in the "oc" parameter is valid. The default value of the
"oc-validity" parameter is 500 (milliseconds).

When a SIP server retransmits a response, it SHOULD use the "oc" and
"oc-validity" parameter values consistent with the overload state at
the time the retransmitted response was sent. This implies that the
values in the "oc" and "oc-validity" parameters may be different than
the ones used in previous retransmissions of the response. Due to
the fact that responses sent over UDP may be subject to delays in the
network and arrive out of order, the "oc-seq" parameter aids in
detecting a stale "oc" parameter value.

Implementations that are capable of updating the "oc" and "oc-
validity" parameter values during retransmissions MUST insert the
"oc-seq" parameter. The value of this parameter MUST be a set of
numbers drawn from an increasing sequence.

Implementations that are not capable of updating the "oc" and "oc-
validity" parameter values during retransmissions -- or
implementations that do not want to do so because they will have to
regenerate the message to be retransmitted -- MUST still insert a
"oc-seq" parameter in the first response associated with a
transaction; however, they do not have to update the value in
subsequent retransmissions.

The "oc-validity" and "oc-seq" Via header parameters are only defined
in SIP responses and MUST NOT be used in SIP requests. These
parameters are only useful to the upstream neighbor of a SIP server
(i.e., the entity that is sending requests to the SIP server) since
the client is the entity that can offload traffic by redirecting or
rejecting new requests. If requests are forwarded in both directions
between two SIP servers (i.e., the roles of upstream/downstream

neighbors change), there are also responses flowing in both
directions. Thus, both SIP servers can exchange overload
information.

This specification provides a good overload control mechanism that
can protect a SIP server from overload. However, if a SIP server
wants to limit advertisements of overload control capability for
privacy reasons, it might decide to perform overload control only for
requests that are received on a secure transport, such as Transport
Layer Security (TLS). Indicating support for overload control on a
request received on an untrusted link can leak privacy in the form of
capabilities supported by the server. To limit the knowledge that
the server supports overload control, a server can adopt a policy of
inserting overload control parameters in only those requests received
over trusted links such that these parameters are only visible to
trusted neighbors.

5.3. Determining the "oc" Parameter Value

The value of the "oc" parameter is determined by the overloaded
server using any pertinent information at its disposal. The only
constraint imposed by this document is that the server control
algorithm MUST produce a value for the "oc" parameter that it expects
the receiving SIP clients to apply to all downstream SIP requests
(dialogue forming as well as in-dialogue) to this SIP server. Beyond
this stipulation, the process by which an overloaded server
determines the value of the "oc" parameter is considered out of the
scope of this document.

   Note: This stipulation is required so that both the client and
   server have a common view of which messages the overload control
   applies to. With this stipulation in place, the client can
   prioritize messages as discussed in Section 5.10.1.

As an example, a value of "oc=10" when the loss-based algorithm is
used implies that 10% of the total number of SIP requests (dialogue
forming as well as in-dialogue) are subject to reduction at the
client. Analogously, a value of "oc=10" when the rate-based
algorithm [RATE-CONTROL] is used indicates that the client should
send SIP requests at a rate of 10 SIP requests or fewer per second.

5.4. Processing the Overload Control Parameters

A SIP client SHOULD remove the "oc", "oc-validity", and "oc-seq"
parameters from all Via headers of a response received, except for
the topmost Via header. This prevents overload control parameters
that were accidentally or maliciously inserted into Via headers by a
downstream SIP server from traveling upstream.

The scope of overload control applies to unique combinations of IP
and port values. A SIP client maintains the overload control values
received (along with the address and port number of the SIP servers
from which they were received) for the duration specified in the "oc-
validity" parameter or the default duration. Each time a SIP client
receives a response with an overload control parameter from a
downstream SIP server, it compares the "oc-seq" value extracted from
the Via header with the "oc-seq" value stored for this server. If
these values match, the response does not update the overload control
parameters related to this server, and the client continues to
provide overload control as previously negotiated. If the "oc-seq"
value extracted from the Via header is larger than the stored value,
the client updates the stored values by copying the new values of the
"oc", "oc-algo", and "oc-seq" parameters from the Via header to the
stored values. Upon such an update of the overload control
parameters, the client restarts the validity period of the new
overload control parameters. The overload control parameters now
remain in effect until the validity period expires or the parameters
are updated in a new response. Stored overload control parameters
MUST be reset to default values once the validity period has expired
(see Section 5.7 for the detailed steps on terminating overload
control).

5.5. Using the Overload Control Parameter Values

A SIP client MUST honor overload control values it receives from
downstream neighbors. The SIP client MUST NOT forward more requests
to a SIP server than allowed by the current "oc" and "oc-algo"
parameter values from that particular downstream server.

When forwarding a SIP request, a SIP client uses the SIP procedures
of [RFC3263] to determine the next-hop SIP server. The procedures of
[RFC3263] take a SIP URI as input, extract the domain portion of that
URI for use as a lookup key, query the DNS to obtain an ordered set
of one or more IP addresses with a port number and transport
corresponding to each IP address in this set (the Expected Output).

After selecting a specific SIP server from the Expected Output, the
SIP client determines if it already has overload control parameter
values for the server chosen from the Expected Output. If the SIP
client has a non-expired "oc" parameter value for the server chosen
from the Expected Output, then this chosen server is operating in
overload control mode. Thus, the SIP client determines if it can or
cannot forward the current request to the SIP server based on the
"oc" and "oc-algo" parameters and any relevant local policy.

The particular algorithm used to determine whether or not to forward
a particular SIP request is a matter of local policy and may take
into account a variety of prioritization factors. However, this
local policy SHOULD transmit the same number of SIP requests as the
sample algorithm defined by the overload control scheme being used.
(See Section 7.2 for the default loss-based overload control
algorithm.)

5.6. Forwarding the Overload Control Parameters

Overload control is defined in a hop-by-hop manner. Therefore,
forwarding the contents of the overload control parameters is
generally NOT RECOMMENDED and should only be performed if permitted
by the configuration of SIP servers. This means that a SIP proxy
SHOULD strip the overload control parameters inserted by the client
before proxying the request further downstream. Of course, when the
proxy acts as a client and proxies the request downstream, it is free
to add overload control parameters pertinent to itself in the Via
header it inserted in the request.

5.7. Terminating Overload Control

A SIP client removes overload control if one of the following events
occur:

1. The "oc-validity" period previously received by the client from
    this server (or the default value of 500 ms if the server did not
    previously specify an "oc-validity" parameter) expires.

2. The client is explicitly told by the server to stop performing
    overload control using the "oc-validity=0" parameter.

A SIP server can decide to terminate overload control by explicitly
signaling the client. To do so, the SIP server MUST set the value of
the "oc-validity" parameter to 0. The SIP server MUST increment the
value of "oc-seq" and SHOULD set the value of the "oc" parameter to
0.

   Note that the loss-based overload control scheme (Section 7) can
   effectively stop overload control by setting the value of the "oc"
   parameter to 0. However, the rate-based scheme [RATE-CONTROL]
   needs an additional piece of information in the form of "oc-
   validity=0".

When the client receives a response with a higher "oc-seq" number
than the one it most recently processed, it checks the "oc-validity"
parameter. If the value of the "oc-validity" parameter is 0, this
indicates to the client that overload control of messages destined to

the server is no longer necessary and the traffic can flow without
any reduction. Furthermore, when the value of the "oc-validity"
parameter is 0, the client SHOULD disregard the value in the "oc"
parameter.

5.8. Stabilizing Overload Algorithm Selection

   Realities of deployments of SIP necessitate that the overload control
   algorithm may be changed upon a system reboot or a software upgrade.
   However, frequent changes of the overload control algorithm must be
   avoided. Frequent changes of the overload control algorithm will not
   benefit the client or the server as such flapping does not allow the
   chosen algorithm to stabilize. An algorithm change, when desired, is
   simply accomplished by the SIP server choosing a new algorithm from
   the list in the client's "oc-algo" parameter and sending it back to
   the client in a response.

   The client associates a specific algorithm with each server it sends
   traffic to, and when the server changes the algorithm, the client
   must change its behavior accordingly.

   Once the server selects a specific overload control algorithm for a
   given client, the algorithm SHOULD NOT change the algorithm
   associated with that client for at least 3600 seconds (1 hour). This
   period may involve one or more cycles of overload control being in
   effect and then being stopped depending on the traffic and resources
   at the server.

      Note: One way to accomplish this involves the server saving the
      time of the last algorithm change in a lookup table, indexed by
      the client's network identifiers. The server only changes the
      "oc-algo" parameter when the time since the last change has
      surpassed 3600 seconds.

5.9. Self-Limiting

   In some cases, a SIP client may not receive a response from a server
   after sending a request. RFC 3261 [RFC3261] states:

      Note: When a timeout error is received from the transaction layer,
      it MUST be treated as if a 408 (Request Timeout) status code has
      been received. If a fatal transport error is reported by the
      transport layer ..., the condition MUST be treated as a 503
      (Service Unavailable) status code.

   In the event of repeated timeouts or fatal transport errors, the SIP
   client MUST stop sending requests to this server. The SIP client
   SHOULD periodically probe if the downstream server is alive using any

mechanism at its disposal. Clients should be conservative in their
probing (e.g., using an exponential back-off) so that their liveness
probes do not exacerbate an overload situation. Once a SIP client
has successfully received a normal response for a request sent to the
downstream server, the SIP client can resume sending SIP requests.
It should, of course, honor any overload control parameters it may
receive in the initial, or later, responses.

5.10. Responding to an Overload Indication

   A SIP client can receive overload control feedback indicating that it
   needs to reduce the traffic it sends to its downstream server. The
   client can accomplish this task by sending some of the requests that
   would have gone to the overloaded element to a different destination.

   It needs to ensure, however, that this destination is not in overload
   and is capable of processing the extra load. A client can also
   buffer requests in the hope that the overload condition will resolve
   quickly and the requests can still be forwarded in time. In many
   cases, however, it will need to reject these requests with a "503
   (Service Unavailable)" response without the Retry-After header.

5.10.1. Message Prioritization at the Hop before the Overloaded Server

   During an overload condition, a SIP client needs to prioritize
   requests and select those requests that need to be rejected or
   redirected. This selection is largely a matter of local policy. It
   is expected that a SIP client will follow local policy as long as the
   result in reduction of traffic is consistent with the overload
   algorithm in effect at that node. Accordingly, the normative
   behavior in the next three paragraphs should be interpreted with the
   understanding that the SIP client will aim to preserve local policy
   to the fullest extent possible.

   A SIP client SHOULD honor the local policy for prioritizing SIP
   requests such as policies based on message type, e.g., INVITEs versus
   requests associated with existing sessions.

   A SIP client SHOULD honor the local policy for prioritizing SIP
   requests based on the content of the Resource-Priority header (RPH)
   [RFC4412]. Specific (namespace.value) RPH contents may indicate
   high-priority requests that should be preserved as much as possible
   during overload. The RPH contents can also indicate a low-priority
   request that is eligible to be dropped during times of overload.

   A SIP client SHOULD honor the local policy for prioritizing SIP
   requests relating to emergency calls as identified by the SOS URN
   [RFC5031] indicating an emergency request. This policy ensures that

when a server is overloaded and non-emergency calls outnumber
emergency calls in the traffic arriving at the client, the few
emergency calls will be given preference. If, on the other hand, the
server is overloaded and the majority of calls arriving at the client
are emergency in nature, then no amount of message prioritization
will ensure the delivery of all emergency calls if the client is to
reduce the amount of traffic as requested by the server.

A local policy can be expected to combine both the SIP request type
and the prioritization markings, and it SHOULD be honored when
overload conditions prevail.

5.10.2. Rejecting Requests at an Overloaded Server

If the upstream SIP client to the overloaded server does not support
overload control, it will continue to direct requests to the
overloaded server. Thus, for the non-participating client, the
overloaded server must bear the cost of rejecting some requests from
the client as well as the cost of processing the non-rejected
requests to completion. It would be fair to devote the same amount
of processing at the overloaded server to the combination of
rejection and processing from a non-participating client as the
overloaded server would devote to processing requests from a
participating client. This is to ensure that SIP clients that do not
support this specification don't receive an unfair advantage over
those that do.

A SIP server that is in overload and has started to throttle incoming
traffic MUST reject some requests from non-participating clients with
a 503 (Service Unavailable) response without the Retry-After header.

5.11. 100 Trying Provisional Response and Overload Control Parameters

The overload control information sent from a SIP server to a client
is transported in the responses. While implementations can insert
overload control information in any response, special attention
should be accorded to overload control information transported in a
100 Trying response.

Traditionally, the 100 Trying response has been used in SIP to quench
retransmissions. In some implementations, the 100 Trying message may
not be generated by the transaction user (TU) nor consumed by the TU.
In these implementations, the 100 Trying response is generated at the
transaction layer and sent to the upstream SIP client. At the
receiving SIP client, the 100 Trying is consumed at the transaction
layer by inhibiting the retransmission of the corresponding request.
Consequently, implementations that insert overload control
information in the 100 Trying cannot assume that the upstream SIP

client passed the overload control information in the 100 Trying to
their corresponding TU. For this reason, implementations that insert
overload control information in the 100 Trying MUST re-insert the
same (or updated) overload control information in the first non-100
Trying response being sent to the upstream SIP client.

6. Example

   Consider a SIP client, P1, which is sending requests to another
   downstream SIP server, P2. The following snippets of SIP messages
   demonstrate how the overload control parameters work.

```
        INVITE sips:user@example.com SIP/2.0
        Via: SIP/2.0/TLS p1.example.net;
          branch=z9hG4bK2d4790.1;oc;oc-algo="loss,A"
        ...

        SIP/2.0 100 Trying
        Via: SIP/2.0/TLS p1.example.net;
          branch=z9hG4bK2d4790.1;received=192.0.2.111;
          oc=0;oc-algo="loss";oc-validity=0
        ...
```

   In the messages above, the first line is sent by P1 to P2. This line
   is a SIP request; because P1 supports overload control, it inserts
   the "oc" parameter in the topmost Via header that it created. P1
   supports two overload control algorithms: "loss" and an algorithm
   called "A".

   The second line -- a SIP response -- shows the topmost Via header
   amended by P2 according to this specification and sent to P1.
   Because P2 also supports overload control and chooses the loss-based
   scheme, it sends "loss" back to P1 in the "oc-algo" parameter. It
   also sets the value of the "oc" and "oc-validity" parameters to 0
   because it is not currently requesting overload control activation.

   Had P2 not supported overload control, it would have left the "oc"
   and "oc-algo" parameters unchanged, thus allowing the client to know
   that it did not support overload control.

At some later time, P2 starts to experience overload. It sends the
following SIP message indicating that P1 should decrease the messages
arriving to P2 by 20% for 0.5 seconds.

```
      SIP/2.0 180 Ringing
      Via: SIP/2.0/TLS p1.example.net;
        branch=z9hG4bK2d4790.3;received=192.0.2.111;
        oc=20;oc-algo="loss";oc-validity=500;
        oc-seq=1282321615.782
      ...
```

After some time, the overload condition at P2 subsides. It then
changes the parameter values in the response it sends to P1 to allow
P1 to send all messages destined to P2.

```
      SIP/2.0 183 Queued
      Via: SIP/2.0/TLS p1.example.net;
        branch=z9hG4bK2d4790.4;received=192.0.2.111;
        oc=0;oc-algo="loss";oc-validity=0;oc-seq=1282321892.439
      ...
```

7. The Loss-Based Overload Control Scheme

Under a loss-based approach, a SIP server asks an upstream neighbor
to reduce the number of requests it would normally forward to this
server by a certain percentage. For example, a SIP server can ask an
upstream neighbor to reduce the number of requests this neighbor
would normally send by 10%. The upstream neighbor then redirects or
rejects 10% of the traffic originally destined for that server.

This section specifies the semantics of the overload control
parameters associated with the loss-based overload control scheme.
The general behavior of SIP clients and servers is specified in
Section 5 and is applicable to SIP clients and servers that implement
loss-based overload control.

7.1. Special Parameter Values for Loss-Based Overload Control

The loss-based overload control scheme is identified using the token
"loss". This token appears in the "oc-algo" parameter list sent by
the SIP client.

Upon entering the overload state, a SIP server that has selected the
loss-based algorithm will assign a value to the "oc" parameter. This
value MUST be in the range of [0, 100], inclusive. This value
indicates to the client the percentage by which the client is to
reduce the number of requests being forwarded to the overloaded
server. The SIP client may use any algorithm that reduces the
traffic it sends to the overloaded server by the amount indicated.

Such an algorithm should honor the message prioritization discussion
in Section 5.10.1. While a particular algorithm is not subject to
standardization, for completeness, a default algorithm for loss-based
overload control is provided in Section 7.2.

7.2. Default Algorithm for Loss-Based Overload Control

This section describes a default algorithm that a SIP client can use
to throttle SIP traffic going downstream by the percentage loss value
specified in the "oc" parameter.

The client maintains two categories of requests. The first category
will include requests that are candidates for reduction, and the
second category will include requests that are not subject to
reduction except when all messages in the first category have been
rejected and further reduction is still needed. Section 5.10.1
contains directives on identifying messages for inclusion in the
second category. The remaining messages are allocated to the first
category.

Under overload condition, the client converts the value of the "oc"
parameter to a value that it applies to requests in the first
category. As a simple example, if "oc=10" and 40% of the requests
should be included in the first category, then:

    10 / 40 * 100 = 25

Or, 25% of the requests in the first category can be reduced to get
an overall reduction of 10%. The client uses random discard to
achieve the 25% reduction of messages in the first category.
Messages in the second category proceed downstream unscathed. To
affect the 25% reduction rate from the first category, the client
draws a random number between 1 and 100 for the request picked from
the first category. If the random number is less than or equal to
the converted value of the "oc" parameter, the request is not
forwarded; otherwise, the request is forwarded.

   A reference algorithm is shown below.

```
cat1 := 80.0          // Category 1 -- Subject to reduction
cat2 := 100.0 - cat1 // Category 2 -- Under normal operations,
// only subject to reduction after category 1 is exhausted.
// Note that the above ratio is simply a reasonable default.
// The actual values will change through periodic sampling
// as the traffic mix changes over time.

while (true) {
  // We're modeling message processing as a single work
  // queue that contains both incoming and outgoing messages.
  sip_msg := get_next_message_from_work_queue()

  update_mix(cat1, cat2)  // See Note below

  switch (sip_msg.type) {

    case outbound request:
      destination := get_next_hop(sip_msg)
      oc_context := get_oc_context(destination)

      if (oc_context == null)  {
          send_to_network(sip_msg) // Process it normally by
          // sending the request to the next hop since this
          // particular destination is not subject to overload.
      }
      else  {
         // Determine if server wants to enter in overload or is in
         // overload.
         in_oc := extract_in_oc(oc_context)
         oc_value := extract_oc(oc_context)
         oc_validity := extract_oc_validity(oc_context)

         if (in_oc == false or oc_validity is not in effect)  {
            send_to_network(sip_msg) // Process it normally by sending
            // the request to the next hop since this particular
            // destination is not subject to overload.  Optionally,
            // clear the oc context for this server (not shown).
         }
         else  {  // Begin performing overload control.
            r := random()
            drop_msg := false

            category := assign_msg_to_category(sip_msg)

            pct_to_reduce_cat1 = oc_value / cat1 * 100
```

```
        if (oc_value <= cat1)  {  // Reduce all msgs from category 1
            if (r <= pct_to_reduce_cat1 && category == cat1)  {
               drop_msg := true
            }
        }
        else  { // oc_value > category 1.  Reduce 100% of msgs from
                // category 1 and remaining from category 2.
           pct_to_reduce_cat2 = (oc_value - cat1) / cat2 * 100
           if (category == cat1)  {
              drop_msg := true
           }
           else  {
              if (r <= pct_to_reduce_cat2)  {
                  drop_msg := true;
              }
           }
        }

        if (drop_msg == false) {
            send_to_network(sip_msg) // Process it normally by
            // sending the request to the next hop.
        }
        else  {
           // Do not send request downstream; handle it locally by
           // generating response (if a proxy) or treating it as
           // an error (if a user agent).
        }

    }  // End perform overload control.
  }

end case // outbound request

case outbound response:
  if (we are in overload) {
    add_overload_parameters(sip_msg)
  }
  send_to_network(sip_msg)

end case // outbound response

case inbound response:

   if (sip_msg has oc parameter values)  {
       create_or_update_oc_context()  // For the specific server
       // that sent the response, create or update the oc context,
       // i.e., extract the values of the oc-related parameters
       // and store them for later use.
```

```
        }
        process_msg(sip_msg)

     end case // inbound response
     case inbound request:

       if (we are not in overload)  {
          process_msg(sip_msg)
       }
       else {  // We are in overload.
          if (sip_msg has oc parameters)  {  // Upstream client supports
             process_msg(sip_msg)  // oc; only sends important requests.
          }
          else {  // Upstream client does not support oc
             if (local_policy(sip_msg) says process message)  {
                process_msg(sip_msg)
             }
             else  {
                send_response(sip_msg, 503)
             }
          }
       }
     end case // inbound request
   }
}
```

   Note: A simple way to sample the traffic mix for category 1 and
   category 2 is to associate a counter with each category of message.
   Periodically (every 5-10 seconds), get the value of the counters, and
   calculate the ratio of category 1 messages to category 2 messages
   since the last calculation.

   Example: In the last 5 seconds, a total of 500 requests arrived at
   the queue. 450 out of the 500 were messages subject to reduction,
   and 50 out of 500 were classified as requests not subject to
   reduction. Based on this ratio, cat1 := 90 and cat2 := 10, so a
   90/10 mix will be used in overload calculations.

8. Relationship with Other IETF SIP Load Control Efforts

   The overload control mechanism described in this document is reactive
   in nature, and apart from the message prioritization directives
   listed in Section 5.10.1, the mechanisms described in this document
   will not discriminate requests based on user identity, filtering
   action, and arrival time. SIP networks that require pro-active
   overload control mechanisms can upload user-level load control
   filters as described in [RFC7200]. Local policy will also dictate
   the precedence of different overload control mechanisms applied to

the traffic. Specifically, in a scenario where load control filters
are installed by signaling neighbors [RFC7200] and the same traffic
can also be throttled using the overload control mechanism, local
policy will dictate which of these schemes shall be given precedence.
Interactions between the two schemes are out of the scope of this
document.

9. Syntax

   This specification extends the existing definition of the Via header
   field parameters of [RFC3261]. The ABNF [RFC5234] syntax is as
   follows:

```
    via-params  =/ oc / oc-validity / oc-seq / oc-algo
    oc          = "oc" [EQUAL oc-num]
    oc-num      = 1*DIGIT
    oc-validity = "oc-validity" [EQUAL delta-ms]
    oc-seq      = "oc-seq" EQUAL 1*12DIGIT "." 1*5DIGIT
    oc-algo     = "oc-algo" EQUAL DQUOTE algo-list *(COMMA algo-list)
                   DQUOTE
    algo-list   = "loss" / *(other-algo)
    other-algo  = %x41-5A / %x61-7A / %x30-39
    delta-ms    = 1*DIGIT
```

10. Design Considerations

   This section discusses specific design considerations for the
   mechanism described in this document. General design considerations
   for SIP overload control can be found in [RFC6357].

10.1. SIP Mechanism

   A SIP mechanism is needed to convey overload feedback from the
   receiving to the sending SIP entity. A number of different
   alternatives exist to implement such a mechanism.

10.1.1. SIP Response Header

   Overload control information can be transmitted using a new Via
   header field parameter for overload control. A SIP server can add
   this header parameter to the responses it is sending upstream to
   provide overload control feedback to its upstream neighbors. This
   approach has the following characteristics:

   o A Via header parameter is light-weight and creates very little
      overhead. It does not require the transmission of additional
      messages for overload control and does not increase traffic or
      processing burdens in an overload situation.

   o Overload control status can frequently be reported to upstream
      neighbors since it is a part of a SIP response. This enables the
      use of this mechanism in scenarios where the overload status needs
      to be adjusted frequently. It also enables the use of overload
      control mechanisms that use regular feedback, such as window-based
      overload control.

   o With a Via header parameter, overload control status is inherent
      in SIP signaling and is automatically conveyed to all relevant
      upstream neighbors, i.e., neighbors that are currently
      contributing traffic. There is no need for a SIP server to
      specifically track and manage the set of current upstream or
      downstream neighbors with which it should exchange overload
      feedback.

   o Overload status is not conveyed to inactive senders. This avoids
      the transmission of overload feedback to inactive senders, which
      do not contribute traffic. If an inactive sender starts to
      transmit while the receiver is in overload, it will receive
      overload feedback in the first response and can adjust the amount
      of traffic forwarded accordingly.

   o A SIP server can limit the distribution of overload control
      information by only inserting it into responses to known upstream
      neighbors. A SIP server can use transport-level authentication
      (e.g., via TLS) with its upstream neighbors.

10.1.2. SIP Event Package

   Overload control information can also be conveyed from a receiver to
   a sender using a new event package. Such an event package enables a
   sending entity to subscribe to the overload status of its downstream
   neighbors and receive notifications of overload control status
   changes in NOTIFY requests. This approach has the following
   characteristics:

   o Overload control information is conveyed decoupled from SIP
      signaling. It enables an overload control manager, which is a
      separate entity, to monitor the load on other servers and provide
      overload control feedback to all SIP servers that have set up
      subscriptions with the controller.

   o With an event package, a receiver can send updates to senders that
      are currently inactive. Inactive senders will receive a
      notification about the overload and can refrain from sending
      traffic to this neighbor until the overload condition is resolved.

The receiver can also notify all potential senders once they are
permitted to send traffic again. However, these notifications do
generate additional traffic, which adds to the overall load.

o A SIP entity needs to set up and maintain overload control
   subscriptions with all upstream and downstream neighbors. A new
   subscription needs to be set up before/while a request is
   transmitted to a new downstream neighbor. Servers can be
   configured to subscribe at boot time. However, this would require
   additional protection to avoid the avalanche restart problem for
   overload control. Subscriptions need to be terminated when they
   are not needed any more, which can be done, for example, using a
   timeout mechanism.

o A receiver needs to send NOTIFY messages to all subscribed
   upstream neighbors in a timely manner when the control algorithm
   requires a change in the control variable (e.g., when a SIP server
   is in an overload condition). This includes active as well as
   inactive neighbors. These NOTIFYs add to the amount of traffic
   that needs to be processed. To ensure that these requests will
   not be dropped due to overload, a priority mechanism needs to be
   implemented in all servers these requests will pass through.

o As overload feedback is sent to all senders in separate messages,
   this mechanism is not suitable when frequent overload control
   feedback is needed.

o A SIP server can limit the set of senders that can receive
   overload control information by authenticating subscriptions to
   this event package.

o This approach requires each proxy to implement user agent
   functionality (UAS and UAC) to manage the subscriptions.

10.2. Backwards Compatibility

   A new overload control mechanism needs to be backwards compatible so
   that it can be gradually introduced into a network and function
   properly if only a fraction of the servers support it.

   Hop-by-hop overload control (see [RFC6357]) has the advantage that it
   does not require that all SIP entities in a network support it. It
   can be used effectively between two adjacent SIP servers if both
   servers support overload control and does not depend on the support
   from any other server or user agent. The more SIP servers in a
   network support hop-by-hop overload control, the better protected the
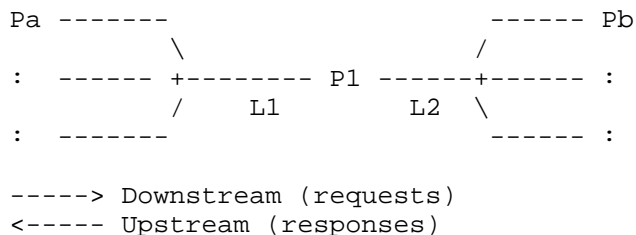   network is against occurrences of overload.

A SIP server may have multiple upstream neighbors from which only
some may support overload control. If a server would simply use this
overload control mechanism, only those that support it would reduce
traffic. Others would keep sending at the full rate and benefit from
the throttling by the servers that support overload control. In
other words, upstream neighbors that do not support overload control
would be better off than those that do.

A SIP server should therefore follow the behavior outlined in
Section 5.10.2 to handle clients that do not support overload
control.

11. Security Considerations

Overload control mechanisms can be used by an attacker to conduct a
denial-of-service attack on a SIP entity if the attacker can pretend
that the SIP entity is overloaded. When such a forged overload
indication is received by an upstream SIP client, it will stop
sending traffic to the victim. Thus, the victim is subject to a
denial-of-service attack.

To better understand the threat model, consider the following
diagram:

```
    Pa -------                        ------ Pb
          \                          /
    : ------ +-------- P1 ------+------ :
            /     L1        L2  \
    : -------                        ------ :

    -----> Downstream (requests)
    <----- Upstream (responses)
```

Here, requests travel downstream from the left-hand side, through
Proxy P1, towards the right-hand side; responses travel upstream from
the right-hand side, through P1, towards the left-hand side. Proxies
Pa, Pb, and P1 support overload control. L1 and L2 are labels for
the links connecting P1 to the upstream clients and downstream
servers.

If an attacker is able to modify traffic between Pa and P1 on link
L1, it can cause a denial-of-service attack on P1 by having Pa not
send any traffic to P1. Such an attack can proceed by the attacker
modifying the response from P1 to Pa such that Pa's Via header is
changed to indicate that all requests destined towards P1 should be
dropped. Conversely, the attacker can simply remove any "oc", "oc-
validity", and "oc-seq" markings added by P1 in a response to Pa. In

such a case, the attacker will force P1 into overload by denying
request quenching at Pa even though Pa is capable of performing
overload control.

Similarly, if an attacker is able to modify traffic between P1 and Pb
on link L2, it can change the Via header associated with P1 in a
response from Pb to P1 such that all subsequent requests destined
towards Pb from P1 are dropped. In essence, the attacker mounts a
denial-of-service attack on Pb by indicating false overload control.
Note that it is immaterial whether Pb supports overload control or
not; the attack will succeed as long as the attacker is able to
control L2. Conversely, an attacker can suppress a genuine overload
condition at Pb by simply removing any "oc", "oc-validity", and "oc-
seq" markings added by Pb in a response to P1. In such a case, the
attacker will force P1 into sending requests to Pb even under
overload conditions because P1 would not be aware that Pb supports
overload control.

Attacks that indicate false overload control are best mitigated by
using TLS in conjunction with applying BCP 38 [RFC2827]. Attacks
that are mounted to suppress genuine overload conditions can be
similarly avoided by using TLS on the connection. Generally, TCP or
WebSockets [RFC6455] in conjunction with BCP 38 makes it more
difficult for an attacker to insert or modify messages but may still
prove inadequate against an adversary that controls links L1 and L2.
TLS provides the best protection from an attacker with access to the
network links.

Another way to conduct an attack is to send a message containing a
high overload feedback value through a proxy that does not support
this extension. If this feedback is added to the second Via header
(or all Via headers), it will reach the next upstream proxy. If the
attacker can make the recipient believe that the overload status was
created by its direct downstream neighbor (and not by the attacker
further downstream), the recipient stops sending traffic to the
victim. A precondition for this attack is that the victim proxy does
not support this extension since it would not pass through overload
control feedback otherwise.

A malicious SIP entity could gain an advantage by pretending to
support this specification but never reducing the amount of traffic
it forwards to the downstream neighbor. If its downstream neighbor
receives traffic from multiple sources that correctly implement
overload control, the malicious SIP entity would benefit since all
other sources to its downstream neighbor would reduce load.

   Note: The solution to this problem depends on the overload control
   method. With rate-based, window-based, and other similar overload
   control algorithms that promise to produce no more than a
   specified number of requests per unit time, the overloaded server
   can regulate the traffic arriving to it. However, when using
   loss-based overload control, such policing is not always obvious
   since the load forwarded depends on the load received by the
   client.

   To prevent such attacks, servers should monitor client behavior to
   determine whether they are complying with overload control policies.
   If a client is not conforming to such policies, then the server
   should treat it as a non-supporting client (see Section 5.10.2).

   Finally, a distributed denial-of-service (DDoS) attack could cause an
   honest server to start signaling an overload condition. Such a DDoS
   attack could be mounted without controlling the communications links
   since the attack simply depends on the attacker injecting a large
   volume of packets on the communication links. If the honest server
   attacked by a DDoS attack has a long "oc-validity" interval and the
   attacker can guess this interval, the attacker can keep the server
   overloaded by synchronizing the DDoS traffic with the validity
   period. While such an attack may be relatively easy to spot,
   mechanisms for combating it are outside the scope of this document
   and, of course, since attackers can invent new variations, the
   appropriate mechanisms are likely to change over time.

12. IANA Considerations

   This specification defines four new Via header parameters as detailed
   below in the "Header Field Parameter and Parameter Values" sub-
   registry as per the registry created by [RFC3968]. The required
   information is:

Header Field	Parameter Name	Predefined Values	Reference
Via	oc	Yes	[RFC7339]
Via	oc-validity	Yes	[RFC7339]
Via	oc-seq	Yes	[RFC7339]
Via	oc-algo	Yes	[RFC7339]

13. References

13.1. Normative References

   [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate
              Requirement Levels", BCP 14, RFC 2119, March 1997.

   [RFC3261] Rosenberg, J., Schulzrinne, H., Camarillo, G., Johnston,
              A., Peterson, J., Sparks, R., Handley, M., and E.
              Schooler, "SIP: Session Initiation Protocol", RFC 3261,
              June 2002.

   [RFC3263] Rosenberg, J. and H. Schulzrinne, "Session Initiation
              Protocol (SIP): Locating SIP Servers", RFC 3263, June
              2002.

   [RFC3968] Camarillo, G., "The Internet Assigned Number Authority
              (IANA) Header Field Parameter Registry for the Session
              Initiation Protocol (SIP)", BCP 98, RFC 3968, December
              2004.

   [RFC4412] Schulzrinne, H. and J. Polk, "Communications Resource
              Priority for the Session Initiation Protocol (SIP)", RFC
              4412, February 2006.

   [RFC5234] Crocker, D. and P. Overell, "Augmented BNF for Syntax
              Specifications: ABNF", STD 68, RFC 5234, January 2008.

13.2. Informative References

   [RATE-CONTROL]
              Noel, E. and P. Williams, "Session Initiation Protocol
              (SIP) Rate Control", Work in Progress, July 2014.

   [RFC2827] Ferguson, P. and D. Senie, "Network Ingress Filtering:
              Defeating Denial of Service Attacks which employ IP Source
              Address Spoofing", BCP 38, RFC 2827, May 2000.

   [RFC5031] Schulzrinne, H., "A Uniform Resource Name (URN) for
              Emergency and Other Well-Known Services", RFC 5031,
              January 2008.

   [RFC5390] Rosenberg, J., "Requirements for Management of Overload in
              the Session Initiation Protocol", RFC 5390, December 2008.

   [RFC6357] Hilt, V., Noel, E., Shen, C., and A. Abdelal, "Design
              Considerations for Session Initiation Protocol (SIP)
              Overload Control", RFC 6357, August 2011.

   [RFC6455] Fette, I. and A. Melnikov, "The WebSocket Protocol", RFC
              6455, December 2011.

   [RFC7200] Shen, C., Schulzrinne, H., and A. Koike, "A Session
              Initiation Protocol (SIP) Load-Control Event Package", RFC
              7200, April 2014.

Appendix A. Acknowledgements

   The authors acknowledge the contributions of Bruno Chatras, Keith
   Drage, Janet Gunn, Rich Terpstra, Daryl Malas, Eric Noel, R.
   Parthasarathi, Antoine Roly, Jonathan Rosenberg, Charles Shen, Rahul
   Srivastava, Padma Valluri, Shaun Bharrat, Paul Kyzivat, and Jeroen
   Van Bemmel to this document.

   Adam Roach and Eric McMurry helped flesh out the different cases for
   handling SIP messages described in the algorithm in Section 7.2.
   Janet Gunn reviewed the algorithm and suggested changes that led to
   simpler processing for the case where "oc_value > cat1".

   Richard Barnes provided invaluable comments as a part of the Area
   Director review of the document.

Appendix B. RFC 5390 Requirements

   Table 1 provides a summary of how this specification fulfills the
   requirements of [RFC5390]. A more detailed view on how each
   requirements is fulfilled is provided after the table.

```
+-------------+-------------------+
| Requirement | Meets requirement |
+-------------+-------------------+
| REQ 1       | Yes               |
| REQ 2       | Yes               |
| REQ 3       | Partially         |
| REQ 4       | Yes               |
| REQ 5       | Partially         |
| REQ 6       | Not applicable    |
| REQ 7       | Yes               |
| REQ 8       | Partially         |
| REQ 9       | Yes               |
| REQ 10      | Yes               |
| REQ 11      | Yes               |
| REQ 12      | Yes               |
| REQ 13      | Yes               |
| REQ 14      | Yes               |
| REQ 15      | Yes               |
| REQ 16      | Yes               |
| REQ 17      | Partially         |
| REQ 18      | Yes               |
| REQ 19      | Yes               |
| REQ 20      | Yes               |
| REQ 21      | Yes               |
| REQ 22      | Yes               |
| REQ 23      | Yes               |
+-------------+-------------------+
```

Table 1: Summary of Meeting Requirements in RFC 5390

REQ 1: The overload mechanism shall strive to maintain the overall useful throughput (taking into consideration the quality-of-service needs of the using applications) of a SIP server at reasonable levels, even when the incoming load on the network is far in excess of its capacity. The overall throughput under load is the ultimate measure of the value of an overload control mechanism.

   Meets REQ 1: Yes. The overload control mechanism allows an
   overloaded SIP server to maintain a reasonable level of throughput
   as it enters into congestion mode by requesting the upstream
   clients to reduce traffic destined downstream.

REQ 2: When a single network element fails, goes into overload, or suffers from reduced processing capacity, the mechanism should strive to limit the impact of this on other elements in the network. This helps to prevent a small-scale failure from becoming a widespread outage.

Meets REQ 2: Yes. When a SIP server enters overload mode, it will
request the upstream clients to throttle the traffic destined to
it. As a consequence of this, the overloaded SIP server will
itself generate proportionally less downstream traffic, thereby
limiting the impact on other elements in the network.

REQ 3: The mechanism should seek to minimize the amount of
configuration required in order to work. For example, it is better
to avoid needing to configure a server with its SIP message
throughput, as these kinds of quantities are hard to determine.

Meets REQ 3: Partially. On the server side, the overload
condition is determined monitoring "S" (cf., Section 4 of
[RFC6357]) and reporting a load feedback "F" as a value to the
"oc" parameter. On the client side, a throttle "T" is applied to
requests going downstream based on "F". This specification does
not prescribe any value for "S" nor a particular value for "F".
The "oc-algo" parameter allows for automatic convergence to a
particular class of overload control algorithm. There are
suggested default values for the "oc-validity" parameter.

REQ 4: The mechanism must be capable of dealing with elements that do
not support it so that a network can consist of a mix of elements
that do and don't support it. In other words, the mechanism should
not work only in environments where all elements support it. It is
reasonable to assume that it works better in such environments, of
course. Ideally, there should be incremental improvements in overall
network throughput as increasing numbers of elements in the network
support the mechanism.

Meets REQ 4: Yes. The mechanism is designed to reduce congestion
when a pair of communicating entities support it. If a downstream
overloaded SIP server does not respond to a request in time, a SIP
client will attempt to reduce traffic destined towards the non-
responsive server as outlined in Section 5.9.

REQ 5: The mechanism should not assume that it will only be deployed
in environments with completely trusted elements. It should seek to
operate as effectively as possible in environments where other
elements are malicious; this includes preventing malicious elements
from obtaining more than a fair share of service.

Meets REQ 5: Partially. Since overload control information is
shared between a pair of communicating entities, a confidential
and authenticated channel can be used for this communication.
However, if such a channel is not available, then the security
ramifications outlined in Section 11 apply.

REQ 6: When overload is signaled by means of a specific message, the
message must clearly indicate that it is being sent because of
overload, as opposed to other, non-overload-based failure conditions.
This requirement is meant to avoid some of the problems that have
arisen from the reuse of the 503 response code for multiple purposes.
Of course, overload is also signaled by lack of response to requests.
This requirement applies only to explicit overload signals.

   Meets REQ 6: Not applicable. Overload control information is
   signaled as part of the Via header and not in a new header.

REQ 7: The mechanism shall provide a way for an element to throttle
the amount of traffic it receives from an upstream element. This
throttling shall be graded so that it is not "all or nothing" as with
the current 503 mechanism. This recognizes the fact that overload is
not a binary state and that there are degrees of overload.

   Meets REQ 7: Yes. Please see Sections 5.5 and 5.10.

REQ 8: The mechanism shall ensure that, when a request was not
processed successfully due to overload (or failure) of a downstream
element, the request will not be retried on another element that is
also overloaded or whose status is unknown. This requirement derives
from REQ 1.

   Meets REQ 8: Partially. A SIP client that has overload
   information from multiple downstream servers will not retry the
   request on another element. However, if a SIP client does not
   know the overload status of a downstream server, it may send the
   request to that server.

REQ 9: That a request has been rejected from an overloaded element
shall not unduly restrict the ability of that request to be submitted
to and processed by an element that is not overloaded. This
requirement derives from REQ 1.

   Meets REQ 9: Yes. A SIP client conformant to this specification
   will send the request to a different element.

REQ 10: The mechanism should support servers that receive requests
from a large number of different upstream elements, where the set of
upstream elements is not enumerable.

   Meets REQ 10: Yes. There are no constraints on the number of
   upstream clients.

REQ 11: The mechanism should support servers that receive requests
from a finite set of upstream elements, where the set of upstream
elements is enumerable.

   Meets REQ 11: Yes.   There are no constraints on the number of
   upstream clients.

REQ 12: The mechanism should work between servers in different
domains.

   Meets REQ 12: Yes.   There are no inherent limitations on using
   overload control between domains. However, interconnections
   points that engage in overload control between domains will have
   to populate and maintain the overload control parameters as
   requests cross domains.

REQ 13: The mechanism must not dictate a specific algorithm for
prioritizing the processing of work within a proxy during times of
overload. It must permit a proxy to prioritize requests based on any
local policy so that certain ones (such as a call for emergency
services or a call with a specific value of the Resource-Priority
header field [RFC4412]) are given preferential treatment, such as not
being dropped, being given additional retransmission, or being
processed ahead of others.

   Meets REQ 13: Yes.   Please see Section 5.10.

REQ 14: The mechanism should provide unambiguous directions to
clients on when they should retry a request and when they should not.
This especially applies to TCP connection establishment and SIP
registrations in order to mitigate against an avalanche restart.

   Meets REQ 14: Yes.   Section 5.9 provides normative behavior on
   when to retry a request after repeated timeouts and fatal
   transport errors resulting from communications with a non-
   responsive downstream SIP server.

REQ 15: In cases where a network element fails, is so overloaded that
it cannot process messages, or cannot communicate due to a network
failure or network partition, it will not be able to provide explicit
indications of the nature of the failure or its levels of congestion.
The mechanism must properly function in these cases.

   Meets REQ 15: Yes.   Section 5.9 provides normative behavior on
   when to retry a request after repeated timeouts and fatal
   transport errors resulting from communications with a non-
   responsive downstream SIP server.

REQ 16: The mechanism should attempt to minimize the overhead of the
overload control messaging.

   Meets REQ 16: Yes. Overload control messages are sent in the
   topmost Via header, which is always processed by the SIP elements.

REQ 17: The overload mechanism must not provide an avenue for
malicious attack, including DoS and DDoS attacks.

   Meets REQ 17: Partially. Since overload control information is
   shared between a pair of communicating entities, a confidential
   and authenticated channel can be used for this communication.
   However, if such a channel is not available, then the security
   ramifications outlined in Section 11 apply.

REQ 18: The overload mechanism should be unambiguous about whether a
load indication applies to a specific IP address, host, or URI so
that an upstream element can determine the load of the entity to
which a request is to be sent.

   Meets REQ 18: Yes. Please see discussion in Section 5.5.

REQ 19: The specification for the overload mechanism should give
guidance on which message types might be desirable to process over
others during times of overload, based on SIP-specific
considerations. For example, it may be more beneficial to process a
SUBSCRIBE refresh with Expires of zero than a SUBSCRIBE refresh with
a non-zero expiration (since the former reduces the overall amount of
load on the element) or to process re-INVITEs over new INVITEs.

   Meets REQ 19: Yes. Please see Section 5.10.

REQ 20: In a mixed environment of elements that do and do not
implement the overload mechanism, no disproportionate benefit shall
accrue to the users or operators of the elements that do not
implement the mechanism.

   Meets REQ 20: Yes. An element that does not implement overload
   control does not receive any measure of extra benefit.

REQ 21: The overload mechanism should ensure that the system remains
stable. When the offered load drops from above the overall capacity
of the network to below the overall capacity, the throughput should
stabilize and become equal to the offered load.

   Meets REQ 21: Yes. The overload control mechanism described in
   this document ensures the stability of the system.

REQ 22: It must be possible to disable the reporting of load
information towards upstream targets based on the identity of those
targets. This allows a domain administrator who considers the load
of their elements to be sensitive information to restrict access to
that information. Of course, in such cases, there is no expectation
that the overload mechanism itself will help prevent overload from
that upstream target.

   Meets REQ 22: Yes. An operator of a SIP server can configure the
   SIP server to only report overload control information for
   requests received over a confidential channel, for example.
   However, note that this requirement is in conflict with REQ 3 as
   it introduces a modicum of extra configuration.

REQ 23: It must be possible for the overload mechanism to work in
cases where there is a load balancer in front of a farm of proxies.

   Meets REQ 23: Yes. Depending on the type of load balancer, this
   requirement is met. A load balancer fronting a farm of SIP
   proxies could be a SIP-aware load balancer or one that is not SIP-
   aware. If the load balancer is SIP-aware, it can make conscious
   decisions on throttling outgoing traffic towards the individual
   server in the farm based on the overload control parameters
   returned by the server. On the other hand, if the load balancer
   is not SIP-aware, then there are other strategies to perform
   overload control. Section 6 of [RFC6357] documents some of these
   strategies in more detail (see discussion related to Figure 3(a)
   of that document).

Authors' Addresses

   Vijay K. Gurbani (editor)
   Bell Labs, Alcatel-Lucent
   1960 Lucent Lane, Rm 9C-533
   Naperville, IL 60563
   USA

   EMail: vkg@bell-labs.com

   Volker Hilt
   Bell Labs, Alcatel-Lucent
   Lorenzstrasse 10
   70435 Stuttgart
   Germany

   EMail: volker.hilt@bell-labs.com

   Henning Schulzrinne
   Columbia University/Department of Computer Science
   450 Computer Science Building
   New York, NY 10027
   USA

   Phone: +1 212 939 7004
   EMail: hgs@cs.columbia.edu
   URI: http://www.cs.columbia.edu