SOC Working Group V. Gurbani, Ed.
Internet-Draft V. Hilt
Intended status: Standards Track Bell Laboratories, Alcatel-Lucent
Expires: September 13, 2012 H. Schulzrinne
Columbia University
March 12, 2012

Session Initiation Protocol (SIP) Overload Control
draft-ietf-soc-overload-control-08

Abstract

Overload occurs in Session Initiation Protocol (SIP) networks when SIP servers have insufficient resources to handle all 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 behaviour of SIP servers involved in overload control, and in addition, it specifies a loss-based overload scheme for SIP.

Status of this Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at http://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on September 13, 2012.

Copyright Notice

Copyright (c) 2012 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (http://trustee.ietf.org/license-info) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

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 in which the throughput drops down to a small fraction of the original processing capacity. 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, network bandwidth, input/output, or disk resources.

For overload control, we only consider 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 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 to shift between SIP servers and thereby worsen 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 general behaviour of SIP servers and clients involved in overload control in Section 5. In addition, Section 6 specifies a loss-based overload control scheme. SIP clients and servers conformant to this specification MUST implement the loss-based overload control scheme. They MAY implement other overload control schemes as well.

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. Similarly, a "SIP server" could be a user agent server 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 (i.e., proxy, B2BUA, UAS, UAC) these terms are used in, "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 neighbour 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 follows that 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

We now explain the 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 best performed hop-by-hop, the Via 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 9.1.1. An alternative mechanism using SIP event packages was also considered, and the characteristics of that choice are further outlined in Section 9.1.2.

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 amount 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 the 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. Inclusion of a value to the parameter represents two things: one, upon an initial handshake (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 SHOULD 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 the SIP request. This parameter contains one or more overload control algorithms. A SIP client MUST support the loss-based overload control scheme and MUST insert the token "loss" as the "oc-algo" parameter value. 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 ([I-D.noel-soc-overload-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 must not 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 MUST choose one algorithm from the list and MUST pare the list down to include the one chosen algorithm. The pared down list consisting of the chosen algorithm MUST be returned to the upstream SIP client in the response.

Once a SIP client and a SIP server have converged to a mutually agreeable class of overload control algorithm, the agreed upon class stays in effect for a non-trivial duration of time to allow the overload control algorithm to stabilize its behaviour (see Section 5.8). Furthermore, the client MUST continue to include all supported algorithms in subsequent requests; the server MUST respond with the agreed to algorithm until such time that the algorithm is changed by the server (see Section 5.8).

4.3. The oc-validity parameter

This parameter is inserted by the SIP server.

This parameter contains a value that indicates an interval of time (measured in milliseconds) that the load reduction specified value of the "oc" parameter should be in effect. The default value of the "oc-validity" parameter is 500 (millisecond).

A value of 0 in the "oc-validity" parameter is reserved to denote the event that the server wishes to stop overload control (see

Section 5.7 for more information).

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

When the period during which the load reduction is in effect expires, the SIP client MUST NOT accord any special meaning to the value of "oc", "oc-seq" and "oc-algo" parameters.

4.4. The oc-seq parameter

This parameter is inserted by the SIP server.

This parameter contains a value that indicates the sequence number associated with the "oc" parameter. Some implementations may be capable of updating the overload control information before the validity period specified by the "oc-validity" parameter expires. Such implementations MUST have an increasing value in the "oc-seq" parameter for each response sent to the upstream SIP client. This is to allow the upstream SIP client to properly collate out-of-order responses.

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

If the value contained in "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.

5. General behaviour

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 as input a SIP URI, extract the domain portion of that URI for use as a lookup key, and 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 MUST determine if it is operating under overload control mode with the server (see Section 5.5) or if this is the initial contact with the server.

If the client determines that this is the initial contact with the

server, it follows the steps outlined in the first paragraph of Section 5.1. Otherwise, the client has conversed with this server before and any overload control parameters established during the previous exchange remain in effect.

5.1. Handshake to determine support for overload control

If a client determines that this is the initial contact with the 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.

A server that supports overload control 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 of Section 5.2.

> A client that supports the rate-based overload control scheme [I-D.noel-soc-overload-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 MAY update the "oc" parameter in all responses it is sending. A SIP server MUST update the "oc" parameter to responses when the transmission of overload control feedback is required by the overload control algorithm to limit the traffic received by the server. I.e., a SIP server MUST update the "oc" parameter when the overload control algorithm sets the value of an "oc" parameter to a value different than the default value.

A SIP server that has updated the "oc" parameter to Via header SHOULD also add a "oc-validity" parameter to the same Via header. The "oc-validity" parameter defines the time in milliseconds during which the content (i.e., the overload control feedback) of the "oc" parameter is valid. The default value of the "oc-validity" parameter is 500 (millisecond). A SIP server SHOULD specify an oc-validity time that is responsive to changing client originated traffic rates, but not too short as to introduce instability. This is a complex subject and outside the scope of this specification. If the "oc-validity" parameter is not present, its default value is used. The "oc-validity" parameter MUST NOT be used in a Via header that did not originally contain an "oc" parameter when received.

When a SIP server retransmits a response, it SHOULD use the "oc" parameter value and "oc-validity" parameter value consistent with the overload state at the time the retransmitted response is sent. This implies that the values in the "oc" and "oc-validity" parameters may be different then 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 for 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 for 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

this is the entity that can offload traffic by redirecting/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.

Since overload control protects a SIP server from overload, it is RECOMMENDED that a SIP server uses the mechanisms described in this specification. However, if a SIP server wanted to limit its overload control capability for privacy reasons, it MAY decide to perform overload control only for requests that are received on a secure transport channel, such as TLS. This enables a SIP server to protect overload control information and ensure that it is only visible to trusted parties.

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 such that the receiving clients can apply it to all downstream requests (dialogue forming as well as in-dialogue). Beyond this stipulation, the process by which an overloaded server determines the value of the "oc" parameter is considered out of scope for this document.

> Note that this stipulation is required so that both the and server have an common view of which messages to include in the calculation of the feedback. 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 uses implies that 10% of all requests (dialog forming as well as in-dialogue) are subject to reduction at the client. Analogously, a value of "oc=10" when the rate-based algorithm [I-D.noel-soc-overload-rate-control] is used indicates that the client should send SIP requests at a rate no greater than or equal to 10 SIP requests per second.

5.4. Processing the Overload Control Parameters

A SIP client SHOULD remove "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 "oc" parameter 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 "oc" parameter from a downstream SIP server, it overwrites the "oc" value it has currently stored for this server with the new value received. The SIP client restarts the validity period of an "oc" parameter each time a response with an "oc" parameter is received from this server. A stored "oc" parameter value MUST be discarded once it has reached the end of its validity.

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" parameter value 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 as input a SIP URI, extract the domain portion of that URI for use as a lookup key, and 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, the SIP client MUST determine 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 MUST determine if it can or cannot forward the current request to the SIP server depending on the nature of the request and the prevailing overload conditions.

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 generate the same number of SIP requests as the default algorithm defined by the overload control scheme being used.

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.

5.7. Terminating overload control

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

1. The "oc-validity" period negotiated to put the server and client in overload state 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 6) can effectively stop overload control by setting the value of the "oc" parameter to 0. However, the rate-based scheme ([I-D.noel-soc-overload-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 currently is processing, it checks the "oc-validity" parameter. If the value of the "oc-validity" parameter is 0, the client MUST stop performing overload control of messages destined to the server and the traffic should 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 be renegotiated upon a system reboot or a software upgrade. However, frequent renegotiation of the overload control algorithm MUST be avoided. A rapid renegotiation of the overload control algorithm will not benefit the client or the server as such flapping does not allow the chosen algorithm to measure and fine tune its behavior over a period of time. Renegotiation, when desired, is simply accomplished by the SIP server choosing a new algorithm from the list in the "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 such that when the server changes the algorithm, the client must behave accordingly as well.

Once the client and server agree on an overload control algorithm, it MUST remain in effect for at least 3600 seconds (1 hour) before renegotiation occurs.

> One way to accomplish this involves the server saving the time of the last negotiation in a lookup table, indexed by the client's network identifiers. Renegotiation is only done when the time of the last negotiation 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. RFC3261 [RFC3261] defines that 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, it 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 for this probe at its disposal. Once a SIP client has successfully transmitted a request to the downstream server, the SIP client can resume normal traffic rates. It should, of course, honor any "oc" parameters it may receive subsequent to resuming normal traffic rates.

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 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 still can be forwarded in time. In many cases, however, it will need to reject these requests.

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. While this selection is largely a matter of local policy, certain heuristics can be suggested. One, during overload control, the SIP client should preserve existing dialogs as much as possible. This suggests that mid-dialog requests MAY be given preferential treatment. Similarly, requests that result in releasing

resources (such as a BYE) MAY also be given preferential treatment.

A SIP client SHOULD honor the local policy for prioritizing SIP requests such as policies based on the content of the Resource-Priority header (RPH, RFC4412 [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. Other indicators, such as the SOS URN [RFC5031] indicating an emergency request, may also be used for prioritization.

Local policy could also include giving precedence to mid-dialog SIP requests (re-INVITEs, UPDATEs, BYEs etc.) in times of overload. A local policy can be expected to combine both the SIP request type and the prioritization markings, and SHOULD be honored when overload conditions prevail.

A SIP client SHOULD honor user-level load control filters installed by signaling neighbors [I-D.ietf-soc-load-control-event-package] by sending the SIP messages that matched the filter downstream.

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, the overloaded server must bear the cost of rejecting some session requests as well as the cost of processing other 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 as the overloaded server would devote to processing requests from an upstream SIP client that supported overload control. This is to ensure that SIP servers that do not support this specification don't receive an unfair advantage over those that do.

A SIP server that is under overload and has started to throttle incoming traffic MUST reject this request with a "503 (Service Unavailable)" response without Retry-After header to reject some requests from upstream neighbors that do not support overload control.

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 response being sent to the upstream SIP client.

6. The loss-based overload control scheme

A loss percentage enables a SIP server to ask an upstream neighbor to reduce the number of requests it would normally forward to this server by X%. 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 that is destined for this server.

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

6.1. Special parameter values for loss-based overload control

The loss-based overload control scheme is identified using the token "loss". This token MUST appear in the "oc-algo" parameter.

A SIP server, upon entering the overload state, will assign a value to the "oc" parameter. This value MUST be restricted in the range of [0, 100], inclusive. This value MUST be interpreted as a percentage, and the SIP client MUST reduce the number of requests being forwarded to the overloaded server by that amount. The SIP client may use any algorithm that reduces the traffic arriving at the overloaded server by the amount indicated. Such an algorithm SHOULD honor the message prioritization discussion of 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 6.3.

When a SIP server receives a request from a client with an "oc" parameter but without a value, and the SIP server is not experiencing overload, it MUST assign a value of 0 to the "oc" parameter in the response. Assigning such a value lets the client know that the server supports overload control and is not currently experiencing overload.

When the "oc-validity" parameter is used to signify overload control termination (Section 5.7), the server MUST insert a value of 0 in the "oc-validity" parameter. The server MUST insert a value of 0 in the "oc" parameter as well. When a client receives a response whose "oc-validity" parameter contains a 0, it MUST treat any non-zero value in the "oc" parameter as if it had received a value of 0 in that parameter.

6.2. 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";
...
```

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 some 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, it chooses the "loss" based scheme and sends that back to P1 in the "oc-algo" parameter. It also sets the value of "oc" parameter to 0.

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 1s.

```
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=1000;
  oc-seq=1282321615.782
...
```

After 500ms, the overload condition at P2 subsides. It then sends out the message below 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=1282321887.783
...
```

6.3. Default algorithm for loss-based overload control

This section describes a default algorithm that a SIP client can 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 under extenuating circumstances when there aren't any messages in the first category that can be reduced). Section Section 5.10.1 contains normative directives on how to prioritize messages for inclusion in the second category. The remaining messages can be allocated to the first category.

The client determines the mix of requests falling into the first category and those falling into the second category. For example, 40% of the requests may be eligible for reduction and 60% not eligible (and therefore, must be sent downstream).

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 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 --- Not subject to
                       // reduction.  80/20 mix.

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()

  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  {
            category := assign_msg_to_category(sip_msg)
            drop_msg := false
            pct_to_reduce := min(100, oc_value / cat1 * 100)

            r := random()
            if (r <= pct_to_reduce)  {
               drop_msg := true
            }

            if (category == cat2 && drop_msg == true)  {
               if (local_policy(sip_msg, oc_value) says
                                                 process message) {
                   drop_msg := 0  // See Note below
               }
            }

            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 locally by
               // generating response (if a proxy) or treating as
               // an error (if a user agent).
            }
         }
      }

    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: local_policy() will have to decide whether to allow a category 2 request downstream if that request has been marked for discard. Some discussion on how to make this decision is captured in Section 5.10.1.

There will be four cases to consider in figuring out how local_policy() should behave. These are enunciated below, and in these cases, t is the inter-invocation time of local_policy() and oc is the value of the "oc" parameter.

```
Case 1: t is small (default: <= 10 times/sec) and oc is small
        (default: < 20%)
Case 2: t is large (default: >= 200 times/sec) and oc is large
        (default: > 70%)
Case 3: t is small and oc is large
Case 4: t is large and oc is small
```

The decision in cases 1 and 3 seems simple. In case 1, local_policy() is not invoked as often and the oc value is small. On the few times that local_policy() is invoked, it could allow the request to to be sent to the server.

In case 3, local_policy() is not invoked as often but the oc value

is large. This implies that there are enough category 1 messages that are being dropped. On the few times that local_policy() is invoked, it could allow the request to be sent to the server.

It is cases 2 and 4 that local_policy() should do something more intelligent.

In case 2, local_policy() is getting invoked very often and the oc is also large. This implies that category 1 requests are being dropped as much as possible and it will help to drop a good number of category 2 requests as well. Thus, it seems reasonable to drop all but the SOS URN [RFC5031] requests and high priority RPH content requests.

In case 4, local_policy() is getting invoked very often, but the oc value is small. This implies that the bulk of traffic to be dropped consists of category 2 requests. So here, it seems reasonable to drop all but the SOS URN [RFC5031] requests.

7. Relationship with other IETF SIP load control efforts

The overload control mechanism described in this document is reactive in nature and apart from message prioritization directives listed in Section 5.10.1 the mechanisms described in this draft 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 [I-D.ietf-soc-load-control-event-package].

8. Syntax

This specification extends the existing definition of the Via header field parameters of [RFC3261] as follows:

```
via-params  =  via-ttl / via-maddr
               / via-received / via-branch
               / oc / oc-validity
               / oc-seq / oc-algo / via-extension


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
```

9. 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].

9.1. 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.

9.1.1. 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:

- 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.
- 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.

9.1.2. 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 request 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.

9.2. Backwards Compatibility

An new overload control mechanism needs to be backwards compatible so that it can be gradually introduced into a network and functions 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 behaviour outlined in Section 5.10.2 to handle clients that do not support overload control.

10. 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.

An attacker can create forged overload feedback by inserting itself into the communication between the victim and its upstream neighbors. The attacker would need to add overload feedback indicating a high load to the responses passed from the victim to its upstream neighbor. Proxies can prevent this attack by communicating via TLS. Since overload feedback has no meaning beyond the next hop, there is no need to secure the communication over multiple hops.

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 headers (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 which correctly implement overload control, the malicious SIP entity would benefit since all other sources to its downstream neighbor would reduce load.

> The solution to this problem depends on the overload control method. For rate-based and window-based overload control, it is very easy for a downstream entity to monitor if the upstream neighbor throttles traffic forwarded as directed. For percentage throttling this is not always obvious since the load forwarded depends on the load received by the upstream neighbor.

11. 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	RFCXXXX
Via	oc-validity	Yes	RFCXXXX
Via	oc-seq	Yes	RFCXXXX
Via	oc-algo	Yes	RFCXXXX

RFC XXXX [NOTE TO RFC-EDITOR: Please replace with final RFC number of this specification.]

NOTE: Do we need to do anything special to register "loss" as a value for "oc-algo" parameter?

12. References

12.1. 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.

12.2. Informative References

[I-D.ietf-soc-load-control-event-package]
Shen, C., Schulzrinne, H., and A. Koike, "A Session Initiation Protocol (SIP) Load Control Event Package", draft-ietf-soc-load-control-event-package-03 (work in progress), March 2012.

[I-D.noel-soc-overload-rate-control]

Noel, E. and P. Williams, "Session Initiation Protocol (SIP) Rate Control", draft-noel-soc-overload-rate-control-02 (work in progress), December 2011.

[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.

Appendix A. Acknowledgements

Many thanks to Bruno Chatras, Keith Drage, Janet Gunn, Rich Terpstra, Daryl Malas, R. Parthasarathi, Antoine Roly, Jonathan Rosenberg, Charles Shen, Rahul Srivastava, Padma Valluri, Shaun Bharrat, Paul Kyzivat and Jeroen Van Bemmel for their contributions to this specification.

Adam Roach and Eric McMurry helped flesh out the different cases for handling SIP messages described in the algorithm of Section 6.3.

Appendix B. RFC5390 requirements

Table 1 provides a summary 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	Partially
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

Summary of meeting requirements in RFC5390

Table 1

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.

Meeting 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.

Meeting 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.

Meeting REQ 3: Partially. On the server side, the overload condition is determined monitoring S (c.f., 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.

Meeting REQ 4: Partially. 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.

Meeting 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 10 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.

Meeting 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.

Meeting REQ 7: Yes, please see Section 5.5 and Section 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.

Meeting 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.

Meeting 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.

Meeting 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.

Meeting REQ 11: Yes, there are no constraints on the number of upstream clients.

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

Meeting REQ 12: Yes, there are no inherent limitations on using overload control between 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.

Meeting REQ 13: Yes, please see Section 5.10.

REQ 14: 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 avalanche restart.

Meeting 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.

Meeting 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.

Meeting 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.

Meeting 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 10 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.

Meeting 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.

Meeting 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.

Meeting 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.

Meeting REQ 21: Yes, the overload control mechanism described in this draft 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.

Meeting 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.

Meeting 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) in Section 6).

Authors' Addresses

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

Email: vkg@bell-labs.com

Volker Hilt
Bell Laboratories, Alcatel-Lucent
791 Holmdel-Keyport Rd
Holmdel, NJ 07733
USA

Email: volkerh@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