

IETF SOC Working Group  
Internet-Draft  
Intended status: Standards Track  
Expires: September 3, 2012

C. Shen  
AT&T  
H. Schulzrinne  
Columbia U.  
A. Koike  
NTT  
March 2, 2012

A Session Initiation Protocol (SIP) Load Control Event Package  
draft-ietf-soc-load-control-event-package-03.txt

Abstract

We define a load control event package for the Session Initiation Protocol (SIP). It allows SIP servers to distribute load filters to other SIP servers in the network. The load filters contain rules to throttle calls based on their source or destination domain, telephone number prefix or for a specific user. The mechanism helps to prevent signaling overload and complements feedback-based SIP overload control efforts.

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 3, 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 . . . . .   | 4  |
| 2. Requirements Notation . . . . .  | 5  |
| 3. Design Requirements . . . . .  | 6  |
| 4. SIP Load Filtering Overview . . . . .  | 6  |
| 4.1. Filter Format . . . . .  | 6  |
| 4.2. Filter Computation . . . . .   | 6  |
| 4.3. Filter Distribution . . . . .  | 7  |
| 4.4. Applicability in Different Network Environments . . . . .                    | 10 |
| 5. Load Control Event Package . . . . .   | 11 |
| 5.1. Event Package Name . . . . .   | 11 |
| 5.2. Event Package Parameters . . . . .   | 11 |
| 5.3. SUBSCRIBE Bodies . . . . .   | 11 |
| 5.4. SUBSCRIBE Duration . . . . .   | 11 |
| 5.5. NOTIFY Bodies . . . . .  | 12 |
| 5.6. Notifier Processing of SUBSCRIBE Requests . . . . .                          | 12 |
| 5.7. Notifier Generation of NOTIFY Requests . . . . .                             | 12 |
| 5.8. Subscriber Processing of NOTIFY Requests . . . . .                           | 12 |
| 5.9. Handling of Forked Requests . . . . .  | 13 |
| 5.10. Rate of Notifications . . . . .   | 13 |
| 5.11. State Delta . . . . .   | 13 |
| 5.12. State Agents . . . . .  | 14 |
| 6. Load Control Document . . . . .  | 14 |
| 6.1. Format . . . . .   | 14 |
| 6.2. Namespace . . . . .  | 15 |
| 6.3. Conditions . . . . .   | 15 |
| 6.3.1. Call Identity . . . . .  | 15 |
| 6.3.2. Validity . . . . .   | 18 |
| 6.3.3. Method . . . . .   | 18 |
| 6.4. Actions . . . . .  | 18 |
| 6.5. Complete Examples . . . . .  | 19 |
| 7. XML Schema Definition for Load Control . . . . .                               | 21 |
| 8. Related Work . . . . .   | 23 |
| 8.1. Relationship with Load Filtering in PSTN . . . . .                           | 23 |
| 8.2. Relationship with Other IETF SIP Load Control Efforts . . . . .              | 24 |
| 9. Discussion of this specification meeting the requirements of RFC5390 . . . . . | 25 |
| 10. Security Considerations . . . . .   | 30 |
| 11. IANA Considerations . . . . .   | 31 |
| 11.1. Load Control Event Package Registration . . . . .                           | 31 |

|  |    |
|--|----|
| 11.2. application/load-control+xml MIME Registration . . . . . | 31 |
| 11.3. Load Control Schema Registration . . . . .               | 32 |
| 12. Acknowledgements . . . . .                                 | 32 |
| 13. References . . . . .                                       | 33 |
| 13.1. Normative References . . . . .                           | 33 |
| 13.2. Informative References . . . . .                         | 33 |
| Authors' Addresses . . . . .                                   | 34 |

## 1. Introduction

Proper functioning of Session Initiation Protocol (SIP) [RFC3265] signaling servers is critical in SIP-based communications networks. The performance of SIP servers can be severely degraded when the server is overloaded with excessive number of signaling requests. Both legitimate and malicious traffic can overload SIP servers, despite appropriate capacity planning.

There are three common examples of legitimate short-term increases in call volumes. Viewer-voting TV shows or ticket giveaways may generate millions of calls within a few minutes. Call volume may also spike during special holidays such as New Year's Day and Mother's Day. Finally, callers may want to reach friends and family in natural disaster areas such as those affected by earthquakes. When possible, only calls traversing overloaded servers should be throttled under those conditions.

SIP load control mechanisms are needed to prevent congestion collapse in these cases [RFC5390]. There are two types of load control approaches. In the first approach, feedback control, SIP servers provide load limits to upstream servers, to reduce the incoming rate of all SIP requests [I-D.ietf-soc-overload-control]. These upstream servers then drop or delay incoming SIP requests. Feedback control is reactive and affects signaling messages that have already been issued by user agent clients. They work well when SIP proxy servers in the core networks (core proxy servers) or destination-specific SIP proxy servers in the edge networks (edge proxy servers) are overloaded. By their nature, they need to distribute rate, drop or window information to all upstream SIP proxy servers and normally affect all calls equally, regardless of destination. However, feedback control is usually ineffective for overload of more general purpose SIP edge proxy servers. For example, in the ticket giveaway case, almost all calls to the hotline will fail at the core proxy servers; if the edge proxy servers leading to the core proxy servers are also overloaded, calls to other destinations will also be rejected or dropped.

Here, we propose an additional, complementary mechanism, called load filtering. Network operators create load filters that indicate that calls to specific destinations or from specific sources should be rate-limited or randomly dropped. These load filters are then distributed to SIP servers and possibly user agents likely to generate calls to the affected destinations or from the affected sources. Load filtering works best if it prevents calls as close to the user agent clients as possible.

Performing SIP load filtering requires three components: load filter

format, load filter computation method, and load filter distribution mechanism. This specification addresses two of these three components. The load filter format is defined in a SIP load control event package, while the load filter distribution mechanism is built upon the existing SIP event framework. The remaining component, load filter computation method, depends heavily on the actual network topology and service provider policies. Therefore it is out of scope of this specification.

It is helpful to clarify two aspects regarding some terminology used in this specification. Firstly, although the SIP load filtering mechanism is motivated by the overload control problem, which is why this specification refers extensively to other parallel SIP overload control related efforts, the applicability of filtering extends beyond the overload control purpose. For example, it can also be used to implement quality of service or other service level agreement commitments. Therefore, we use the term SIP "load control event package", instead of a narrower term "overload control event package". Secondly, since we are describing a specific control mechanism based on filtering, the term "load control" in this specification is used inter-changeably with the term "load filtering" unless when associated with other explicit context. This specification, however, does not preclude the load control document defined here (Section 6) to be extended in the future for other forms of control as appropriate.

The rest of this specification is structured as follows: we begin by listing the design requirements for this work in Section 3. We then give an overview of load filtering operation in Section 4. The load control event package for filter distribution is detailed in Section 5. The load filter format is defined in the two sections that follow, with Section 6 introducing the XML document for load control and Section 7 listing the associated schema. Section 8 relates this work to corresponding mechanisms in PSTN and other IETF efforts addressing SIP load control. Section 9 evaluates whether this specification meets the SIP overload control requirements set forth by RFC5390 [RFC5390]. Finally, Section 10 presents security considerations and Section 11 provides IANA considerations.

## 2. Requirements Notation

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 [RFC2119].

## 3. Design Requirements

The SIP load filtering mechanism needs to satisfy the following requirements:

- o To simplify the solution, we focus on a method for controlling SIP load, rather than a generic application-layer mechanism.
- o The load filter needs to be distributed efficiently to possibly a large subset of all SIP elements.
- o The solution should re-use existing SIP protocol mechanisms to reduce implementation and deployment complexity.
- o For predictable overload situations, such as holidays and call-in events, the load filter should specify during what time period it is to be applied, so that the information can be distributed ahead of time.
- o For destination-specific overload situations, the load filter needs to be able to describe the callee.
- o To address accidental and intentional high-volume call generators, the load filter should allow to specify the caller.
- o Caller and callee need to be specified as both SIP URIs and 'Tel' URIs[RFC3966].
- o For telephone numbers, it should be possible to specify prefixes which allow control over limited regionally-focused overloads.
- o The solution should draw upon experiences from related PSTN mechanisms where applicable.
- o The solution should be extensible to meet future needs.

#### 4. SIP Load Filtering Overview

##### 4.1. Filter Format

A load filter contains both conditions and actions. Filter conditions include the identities of the targets to be controlled. For example, there are two typical resource limits in a possible overload situation, i.e., human destination limits (N number of call takers) and proxy capacity limits. The control targets in these two cases can be the specific callee numbers or the destination domains corresponding to the overload. Filter conditions also indicate the period of time during which the control should be activated, and the specific message type to be controlled, e.g., the INVITE message of a SIP session. Filter actions describe the desired control functions such as limiting the request rate below a certain level. Detailed formats of filter conditions and actions are defined in Section 6.

##### 4.2. Filter Computation

Load filter computation needs to take into consideration information such as the overload time, scope and network topology, as well as service policies. It is also important to make sure that there is no

resource allocation loop, and that loads are allocated in a way which both prevents overload and minimizes the likelihood of network resource under-utilization. In some cases, in order to better utilize system resources, it may be preferable to employ a dynamic load computation algorithm which adapts to current network status, rather than using a purely static mechanism. The load filter computation algorithm is out of scope of this specification.

#### 4.3. Filter Distribution

For load filter distribution, this specification defines the SIP event package for load control, which is an "instantiation" of the generic SIP events framework [RFC3265]. The SIP events framework provides an existing method for SIP entities to subscribe to and receive notifications when certain events have occurred. Such a framework forms a scalable event distribution architecture that suits our needs. This specification also defines the XML schema of a load control document (Section 6), which is used to encode load filtering rules.

In order for load filters to be properly distributed, each SIP proxy server in the network is required to subscribe to the load control event package from all its outgoing signaling neighbors, known as notifiers (Section 5.6). Subscription is initiated and maintained during normal server operation. Signaling neighbors are defined by sending signaling messages. For instance, if A sends signaling requests to B, B is an outgoing signaling neighbor of A. A needs to subscribe to the load control event package of B in case B wants to curb requests from A. On the other hand, if B also sends signaling requests to A, then B also subscribes to A. Subscription of neighboring SIP entities needs to be persistent so that they are in place independently of any specific load filtering events. Key to this is the fact that notification following initial subscription includes an empty message body if no events are configured (Section 5.7), and that the subscription needs to be refreshed periodically to make it persistent, as described in Section 3.1.6 and Section 3.1.4.2 of [RFC3265]. The notifier will send a notification with its current control policy to its subscribers each time a new subscription or a subscription refreshing is accepted (Section 5.7). The same subscription dialog can also be used to convey policies for multiple different load filtering events in a set of rules (Section 6.1).

We use the example architecture shown in Figure 1 to illustrate load filter distribution based on the SIP load control event package. This scenario consists of two networks belonging to Service Provider A and Service Provider B, respectively. Each provider's network is made up of two SIP core proxy servers and four SIP edge proxy

servers. The core proxy servers and edge proxy servers of Service Provider A are denoted as CPa1 to CPa2 and EPa1 to EPa4; the core proxy servers and edge proxy servers of Service Provider B are denoted as CPb1 to CPb2 and EPb1 to EPb4.

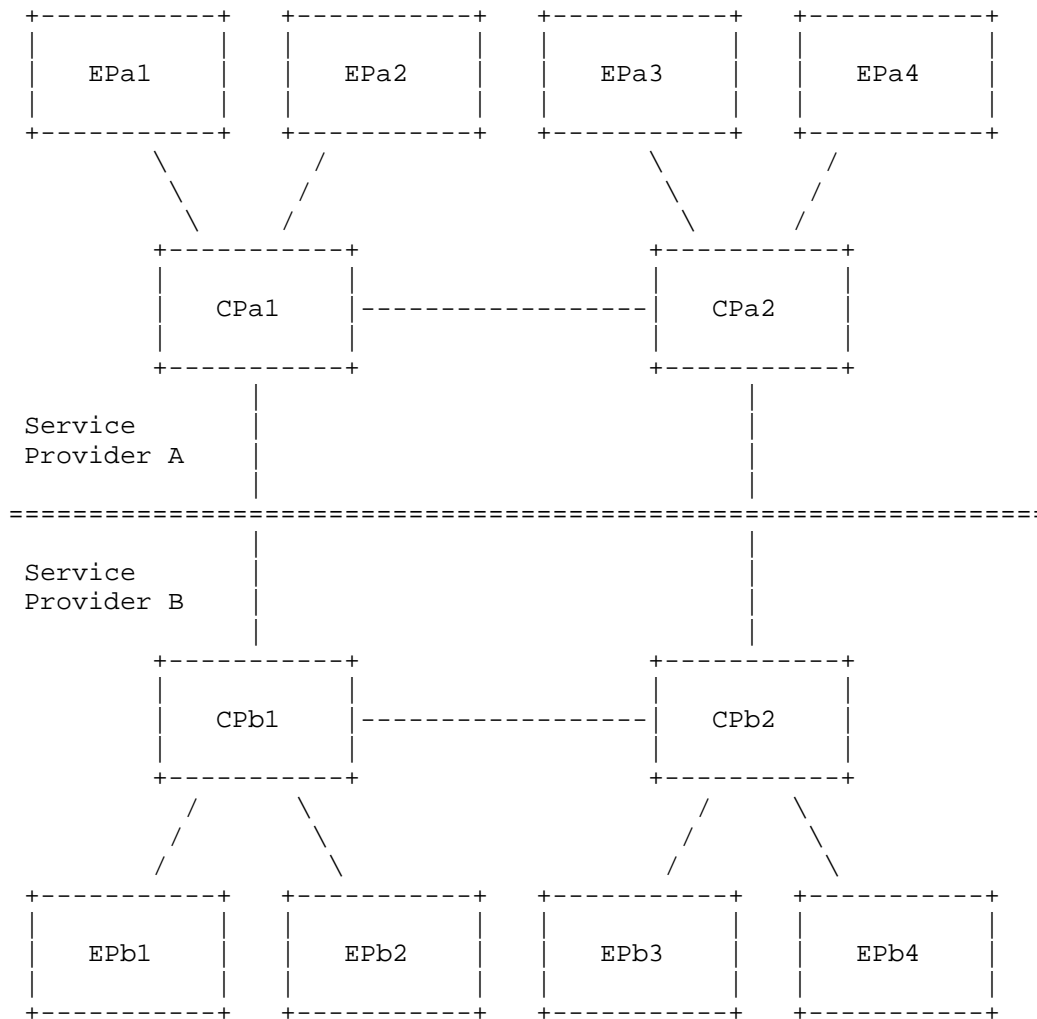


Figure 1: Example Network Scenario Using SIP Load Control Event Package Mechanism

At the initialization stage, the proxy servers first identify all



their outgoing signaling neighbors and subscribe to them. The neighbor identification process can be performed by service providers through direct provisioning, or by the proxy servers themselves via progressively learning from the signaling messages sent and received. Assuming all signaling relationship in Figure 1 is bi-directional, after this initialization stage, each proxy server will be subscribed to all its neighbors. That is, EPa1 subscribes to CPa1; CPa1 subscribes to EPa1, EPa2, CPa2 and CPb1, so on and so forth. The following cases then show two examples of how load filter distribution in this network works.

Case I: EPa1 serves a TV program hotline and decides to limit the total number of incoming calls to the hotline to prevent an overload. To do so, EPa1 sends a notification to CPa1 with the specific hotline number, time of activation and total acceptable call rate. Depending on the filter computation algorithm, CPa1 may allocate the received total acceptable rate among its neighbors, namely, EPa2, CPa2, and CPb1, and notify them about the resulting allocation along with the hotline number and the activation time. CPa2 and CPb1 may perform further allocation among their own neighbors and notify the corresponding proxy servers. This process continues until all edge proxy servers in the network have been informed about the event and have proper load filter configured.

Case II: an earthquake affects the region covered by CPb2, EPb3 and EPb4. All the three proxy servers are overloaded. The rescue team determines that outbound calls are more valuable than inbound calls in this specific situation. Therefore, EPb3 and EPb4 are configured with filters to accept more outbound calls than inbound calls. CPb2 may be configured the same way or receive dynamically computed filters from EPb3 and EPb4. Depending on the filter computation algorithm, CPb2 may also send out notifications to its outside neighbors, namely CPb1 and CPa2, specifying a limit on the acceptable rate of inbound calls to CPb2's responsible domain. CPb1 and CPa2 may subsequently notify their neighbors about limiting the calls to CPb2's area. The same process could continue until all edge proxy servers are notified and have filters configured.

In the above two cases, the network entity where load filtering policy is first introduced is the SIP server to be protected. In other cases, the network entry point of load filtering policy could also be an entity that the protected SIP server is connected to. For example, an operator may host an application server that performs 800 number translation services. The application server may itself be a SIP proxy or a SIP Back-to-Back User Agent (B2BUA). If one of the 800 numbers hosted at the application server creates the overload condition, the load filtering policies can be introduced from the application server and then propagated to other SIP proxy servers in

the network.

Note that this specification does not define the provisioning interface between the party who determines the load control policy and the network entry point where the policy is introduced. One of the options for the provisioning interface is the Extensible Markup Language (XML) Configuration Access Protocol (XCAP) [RFC4825].

#### 4.4. Applicability in Different Network Environments

SIP load filtering is more effective when the filters can be pushed to the proximity of signaling sources. But even if only part of the signaling path towards the signaling source could be covered, use of this mechanism can still be beneficial. In fact, due to possibly sophisticated call routing and security concerns, trying to apply automated load filter distribution in the entire inter-domain network path could get extremely complicated and be unrealistic.

The scenarios where this mechanism could be most useful are environments consisting of servers with secure and trust relationship and with relatively straightforward routing configuration known to the filter computation algorithm. These scenarios may include intra-domain environments such as those inside a service provider or enterprise domain; inter-domain environments such as where enterprise connecting to a few service providers or between service providers with manageable routing configurations.

Another important aspect that affects the applicability of SIP load filtering is that all possible signaling source neighbors need to participate and enforce the designated filter. Otherwise, a single non-conforming neighbor could make the whole control efforts useless by pumping in excessive traffic to overload the server. Therefore, the SIP server that initiates the filter needs to take counter-measures towards any non-conforming neighbors. A simple policy is to reject excessive requests with 500 responses as if they were obeying the rate. Considering the rejection costs, a more complicated but fairer policy would be to allocate at the overloaded server the same amount of processing to the combination of both normal processing and rejection as the overloaded server would devote to processing requests for a conforming upstream SIP server. These approaches work as long as the total rejection cost does not overwhelm the entire server resources. In addition, whatever the actual policy is, SIP servers SHOULD honor the Resource-Priority Header (RPH) [RFC4412] when processing messages. The RPH contents may indicate high priority requests that should be preserved as much as possible, or low priority requests that could be dropped during overload. SIP request rejection and message prioritization at an overloaded server are also discussed in Section 5.1 of [I-D.ietf-soc-overload-control]

and Section 12 of [RFC6357].

## 5. Load Control Event Package

The SIP load filtering mechanism uses the SIP event package for load control. This section defines details of the SIP event package for load control according to [RFC3265].

### 5.1. Event Package Name

The name of this event package is "load-control". This name is carried in the Event and Allow-Events header, as specified in [RFC3265].

### 5.2. Event Package Parameters

No package specific event header field parameters are defined for this event package.

### 5.3. SUBSCRIBE Bodies

The effectiveness of SIP load filtering relies on the scope of distribution and installation of the control policies in the network. Since wide distribution of control policies is desirable, subscribers SHOULD try to subscribe to all those notifiers with which they have regular signaling exchanges, although not all such notifiers may permit such a subscription.

A SUBSCRIBE request for the SIP load control event package MAY contain a body to filter the requested load control event notification. For example, a subscriber may be interested in some specific types of load control policy only. The details of the subscription filter specification are not yet defined.

A SUBSCRIBE request sent without a body implies the default subscription behavior as specified in Section 5.7.

### 5.4. SUBSCRIBE Duration

The default expiration time for a subscription to load control policy is one hour. Since the desired expiration time may vary significantly for subscriptions among SIP entities with different signaling relationships, the subscribers and notifiers are RECOMMENDED to explicitly negotiate appropriate subscription durations when knowledge about the mutual signaling relationship is available.

### 5.5. NOTIFY Bodies

The body of a NOTIFY request in this event package contains load control policy. As specified in [RFC3265], the format of the NOTIFY body MUST be in one of the formats defined in the Accept header field of the SUBSCRIBE request or be the default format. The default data format for the NOTIFY body of this event package is "application/load-control+xml" (defined in Section 6). This means that if no Accept header field is specified to a SUBSCRIBE request, the NOTIFY request will contain a body in the "application/load-control+xml" format. If the Accept header field is present, it MUST include "application/load-control+xml" and MAY include any other types.

### 5.6. Notifier Processing of SUBSCRIBE Requests

Notifier accepts a new subscription or updates an existing subscription upon receiving a valid SUBSCRIBE request.

If the identity of the subscriber sending the SUBSCRIBE request is not allowed to receive load control policy, the notifier MUST return a 403 "Forbidden" response.

If none of MIME types specified in the Accept header of the SUBSCRIBE is supported, the Notifier SHOULD return 406 "Not Acceptable" response.

### 5.7. Notifier Generation of NOTIFY Requests

Following [RFC3265] specification, a notifier MUST send a NOTIFY with its current load control policy to the subscriber upon successfully accepting or refreshing a subscription. If no applicable restriction is active when the subscription request is received, an empty message body is attached to the NOTIFY request. This is often the case when a subscription is initiated for the first time, e.g., when a SIP entity is just introduced, because there may be no planned events configured at that time. A notifier SHOULD generate NOTIFY requests each time the load control policy changes, with the maximum notification rate not exceeding values defined in Section 5.10.

### 5.8. Subscriber Processing of NOTIFY Requests

The way subscribers process NOTIFY requests depends on the contents of the notifications. Typically, a load control notification consists of rules that should be applied to requests matching certain identities. A subscriber receiving the notification first installs these rules and then filter incoming requests to enforce actions on appropriate requests, for example, limiting the sending rate of call requests destined for a specific SIP entity.

In the case when load control rules specify a future validity time, it is possible that when the validity time comes, the subscription to the specific notifier that conveyed the rules has expired. In this case, it is RECOMMENDED that the subscriber re-activate its subscription with the corresponding notifier. Regardless of whether this re-activation of subscription is successful or not, when the validity time is reached, the subscriber SHOULD enforce the corresponding rules.

Upon receipt of a NOTIFY request with a Subscription-State header field containing the value "terminated", the subscription status with the particular notifier will be terminated. However, subscribers SHOULD NOT change previously received load control policies from that notifier because of this change in subscription status, unless it has other specific reasons to do so. Modifications of existing load control policies at the subscriber is performed after directly receiving notifications containing updated load control policies.

The subscriber SHALL discard unknown bodies. If the NOTIFY request contains several bodies, none of them being supported, it SHOULD unsubscribe. A NOTIFY request that does not contain a body MUST be ignored.

#### 5.9. Handling of Forked Requests

Forking is not applicable when the load control event package is used within a single-hop distance between neighboring SIP entities. If the communication scope of the load control event package is among multiple hops, forking is not expected to happen either because the subscription request is addressed to a clearly defined SIP entity. However, in the unlikely case when forking does happen, the load control event package only allows the first potential dialog-establishing message to create a dialog, as specified in Section 4.4.9 of [RFC3265].

#### 5.10. Rate of Notifications

Rate of notifications is likely not a concern for this event package when it is used in a non-real-time mode for relatively static load control policies. Nevertheless, if situation does arise that a rather frequent load control policy update is needed, it is RECOMMENDED that the notifier does not generate notifications at a rate higher than once per-second in all cases, in order to avoid the NOTIFY request itself overloading the system.

#### 5.11. State Delta

It is likely that updates to specific load control events are made by

changing the control restriction parameter information only (e.g. rate, percent), but not other rule elements, such as call-identity. This will typically be because the utilisation of a resource subject to overload depends upon dynamic unknowns such as holding time and the relative distribution of offered loads over subscribing SIP entities. The updates could originate manually or be determined automatically by a dynamic filter computation algorithm (Section 4.2). Another factor usually not known precisely or is computed automatically is the validity duration of the load control event. Therefore it would also be common for the validity to change frequently.

This event package allows the use of state delta to accommodate frequent updates of partial rule parameters. As in [RFC3265], a version number that increases by exactly one is included in the NOTIFY body for each NOTIFY transaction in a subscription. When the subscriber receives a state delta, it associates the partial updates to the particular rules by matching the appropriate rule id (Section 6.5). If the subscriber receives a NOTIFY that has a version number that is increased by more than one, it knows that it has missed a state delta. The subscriber then keeps the version number, ignores the NOTIFY request containing the state delta, and re-sends a SUBSCRIBE to force a NOTIFY containing a complete state snapshot.

#### 5.12. State Agents

The load control policy can be directly generated by concerned SIP entities distributed in the network. Alternatively, qualified state agents external to the SIP entities MAY be defined to take charge of determining load control policies.

### 6. Load Control Document

#### 6.1. Format

A load control document is an XML document that inherits and enhances the common policy document defined in [RFC4745]. A common policy document contains a set of rules. Each rule consists of three parts: conditions, actions and transformations. The conditions part is a set of expressions containing attributes such as identity, domain, and validity time information. Each expression evaluates to TRUE or FALSE. Conditions are matched on "equality" or "greater than" style comparison. There is no regular expression matching. Conditions are evaluated on receipt of an initial SIP request for a dialog or standalone transaction. If a request matches all conditions in a rule set, the action part and the transformation part are consulted

to determine the "permission" on how to handle the request. Each action or transformation specifies a positive grant to the policy server to perform the resulting actions. Well-defined mechanisms are available for combining actions and transformations obtained from more than one source.

## 6.2. Namespace

The namespace URI for elements defined by this specification is a Uniform Resource Namespace (URN) ([RFC2141]), using the namespace identifier 'ietf' defined by [RFC2648] and extended by [RFC3688]. The URN is as follows:

```
urn:ietf:params:xml:ns:load-control
```

## 6.3. Conditions

[RFC4745] defines three condition elements: <identity>, <sphere> and <validity>. In this specification, we re-define an element for identity and reuse the <validity> element. The <sphere> element is not used.

### 6.3.1. Call Identity

Since the problem space of this specification is different from that of [RFC4745], the [RFC4745] <identity> element is not sufficient for use with load control. First, load control may be applied to different identities contained in a request, including identities of both the receiving entity and the sending entity. Second, the importance of authentication varies when different identities of a request are concerned. This specification defines new identity conditions that can accommodate the granularity of specific SIP identity header fields. The requirement for authentication depends on which field is to be matched.

The identity condition for load control is specified by the <call-identity> element and its sub-element <sip>. The <sip> element itself contains sub-elements representing SIP sending and receiving identity header fields: <from>, <to>, <request-uri> and <p-asserted-identity>, each is of the same type as the <identity> element in [RFC4745]. Therefore, they also inherit the sub-elements of the <identity> element, including <one>, <except>, and <many>.

The [RFC4745] <one> and <except> elements may contain an "id" attribute, which is the URI of a single entity to be included or excluded in the condition. When used in the <from>, <to>, <request-uri> and <p-asserted-identity> elements, this "id" value is the URI contained in the corresponding SIP header field, i.e., From, To,

Request-URI, and P-Asserted-Identity.

When the <call-identity> element contains multiple <sip> sub-elements, the result is combined using logical OR. When the <from>, <to>, <request-uri> and <p-asserted-identity> elements contain multiple <one>, <except>, or <many> sub-elements, the result is also combined using logical OR, similar to that of the <identity> element in [RFC4745]. However, when the <sip> element contains multiple of the <from>, <to>, <request-uri> and <p-asserted-identity> sub-elements, the result is combined using logical AND. This allows the call identity to be specified by multiple fields of a SIP request simultaneously, e.g., both the From and the To header fields.

The following shows an example of the <call-identity> element.

```
<call-identity>
  <sip>
    <to>
      <one id="sip:alice@hotline.example.com"/>
      <one id="tel:+1-212-555-1234"/>
    </to>
  </sip>
</call-identity>
```

This example matches call requests whose To header field contains the SIP URI "sip:alice@hotline.example.com", or the 'tel' URI "tel:+1-212-555-1234".

The [RFC4745] <many> and <except> elements may take a "domain" attribute. The "domain" attribute specifies a domain name to be matched by the domain part of the candidate identity. Thus, it allows matching a large and possibly unknown number of entities within a domain. The "domain" attribute works well for SIP URIs.

A URI identifying a SIP user, however, can also be a 'tel' URI. We therefore need a similar way to match a group of 'tel' URIs. According to [RFC3966], there are two forms of 'tel' URIs for global numbers and local numbers, respectively. All phone numbers must be expressed in global form when possible. The global number 'tel' URIs start with a "+". The rest of the numbers are expressed as local numbers, which must be qualified by a "phone-context" parameter. The "phone-context" parameter may be labelled as a global number or any number of its leading digits, or a domain name. Both forms of the 'tel' URI make the resulting URI globally unique.

'Tel' URIs of global numbers can be grouped by prefixes consisting of



any number of common leading digits. For example, a prefix formed by a country code or both the country and area code identifies telephone numbers within a country or an area. Since the length of the country and area code for different regions are different, the length of the number prefix is also variable. This allows further flexibility such as grouping the numbers into sub-areas within the same area code. 'Tel' URIs of local numbers can be grouped by the value of the "phone-context" parameter.

To include the two forms of 'tel' URI grouping in the <many> and <except> elements, one approach is to add a new attribute similar to the "domain" attribute. In this specification, we decided on a simpler approach. There are basically two types of grouping attribute values for both SIP URIs and 'tel' URIs: domain name and number prefix starting with "+". Both of them can be expressed as strings. Therefore, we re-interpret the existing "domain" attribute of the <many> and <except> elements to allow it to contain both types of grouping attribute values. In particular, when the "domain" attribute value starts with "+", it denotes a number prefix, otherwise, the value denotes a domain name. Note that the tradeoff of this simpler approach is the overlap in matching different types of URIs. Specifically, a domain name in the "domain" attribute could be matched by both a SIP URI with that domain name and a local number 'tel' URI containing the same domain name in the "phone-context". On the other hand, a number prefix in the "domain" attribute could be matched by both global number 'tel' URIs starting with those leading digits, and local number 'tel' URIs having the same prefix in the "phone-context" parameter. These overlap situations would not be a big problem because of two reasons. First, when the "phone-context" coincides with the SIP domain name or the global number prefix, it is usually the case that the related phone numbers indeed belong to the same domain or the same area, which means the overlap is not inappropriate. Second, use of the local number 'tel' URI in practice is expected to be rare. As a result, the chance of such overlap happening is very small.

The following example shows the use of the re-interpreted "domain" attribute.

```
<call-identity>
  <sip>
    <from>
      <many>
        <except domain="+1-212"/>
        <except domain="manhattan.example.com"/>
      </many>
    </from>
  <to>
```

```
        <one id="tel:+1-202-999-1234"/>
      </to>
    </sip>
  </call-identity>
```

This example matches those requests calling to the number "+1-202-999-1234" but are not calling from a "+1-212" prefix or a SIP From URI domain of "manhattan.example.com".

#### 6.3.2. Validity

A rule is usually associated with a validity period condition. This specification reuses the <validity> element of [RFC4745], which specifies a period of validity time by pairs of <from> and <until> sub-elements. When multiple time periods are defined, the validity condition is evaluated to TRUE if the current time falls into any of the specified time periods. i.e., it represents a logical OR operation across all validity time periods.

The following example shows a <validity> element specifying a valid period from 12:00 to 15:00 US Eastern Standard Time on 2008-05-31.

```
<validity>
  <from>2008-05-31T12:00:00-05:00</from>
  <until>2008-05-31T15:00:00-05:00</until>
</validity>
```

#### 6.3.3. Method

The load created on a SIP server depends on the type of an initial SIP request for a dialog or standalone transaction. The <method> element specifies the SIP method to which a particular action applies. When this element is not included, the rule actions are applicable to all initial methods.

The following example shows the use of the <method> element.

```
<method>INVITE</method>
```

#### 6.4. Actions

As [RFC4745] specified, conditions form the 'if'-part of rules, while actions and transformations form the 'then'-part. Transformations are not used in the load control document. The actions for load control are defined by the <accept> element, which takes any one of the three sub-elements <rate>, <percent>, and <win>. The <rate> element denotes an absolute value of the maximum acceptable request rate in requests per second; the <percent> element specifies the

relative percentage of incoming requests that should be accepted; the <win> element describes the acceptable window size supplied by the receiver, which is applicable in window-based load control. In static load filter configuration scenarios, using the <rate> sub-element is RECOMMENDED because it is hard to enforce the percentage rate or window-based control when the incoming load from upstream or the reactions from downstream are uncertain. (See [I-D.ietf-soc-overload-control] [RFC6357] for more details on rate-based, loss-based and window-based load control)

In addition, the <accept> element takes an optional "alt-action" attribute which can be used to explicitly specify the desired action in case a request cannot be accepted. The possible "alt-action" values are "drop" for simple drop, "reject" for explicit rejection (e.g., sending a "500 Server Internal Error" response message to an INVITE request), and "forward". The default value is "reject" in order to avoid possible SIP retransmissions when an unreliable transport is used. If the "alt-action" value is "forward", an "alt-target" attribute MUST be defined. The "alt-target" specifies a URI where the request should be forwarded (e.g., an answering machine with explanation of why the request cannot be accepted).

In the following <actions> element example, the server accepts maximum of 100 call requests per second. The remaining calls are forwarded to an answering machine.

```
<actions>
  <accept alt-action="forward" alt-target=
    "sip:answer-machine@example.com">
    <rate>100</rate>
  </accept>
</actions>
```

## 6.5. Complete Examples

This section presents two complete examples of load control documents vliad with respect to the XML schema defined in Section 7.

The first example assumes that a set of hotlines are set up at "sip:alice@hotline.example.com" and "tel:+1-212-555-1234". The hotlines are activated from 12:00 to 15:00 US Eastern Standard Time on 2008-05-31. The goal is to limit the incoming calls to the hotlines to 100 requests per second. Calls that exceed the rate limit are explicitly rejected.

```
<?xml version="1.0" encoding="UTF-8"?>
<ruleset xmlns="urn:ietf:params:xml:ns:common-policy"
  xmlns:lc="urn:ietf:params:xml:ns:load-control"
```

```
        version="0" state="full">

    <rule id="f3g44k1">
        <conditions>
            <lc:call-identity>
                <lc:sip>
                    <lc:to>
                        <one id="sip:alice@hotline.example.com"/>
                        <one id="tel:+1-212-555-1234"/>
                    </lc:to>
                </lc:sip>
            </lc:call-identity>
            <validity>
                <from>2008-05-31T12:00:00-05:00</from>
                <until>2008-05-31T15:00:00-05:00</until>
            </validity>
        </conditions>
        <actions>
            <lc:accept alt-action="reject">
                <lc:rate>100</lc:rate>
            </lc:accept>
        </actions>

    </rule>
</ruleset>
```

The second example considers optimizing server resource usage of a three-day period during the aftermath of an earthquake. Incoming calls to the earthquake domain "pompeii.example.com" will be limited to a rate of 100 requests per second, except for those calls originating from a particular rescue team domain "rescue.example.com". Outgoing calls from the earthquake domain or calls within the local domain are never limited. All calls that are throttled due to the rate limit will be forwarded to an answering machine with updated earthquake rescue information.

```
<?xml version="1.0" encoding="UTF-8"?>
<ruleset xmlns="urn:ietf:params:xml:ns:common-policy"
  xmlns:lc="urn:ietf:params:xml:ns:load-control"
  version="1" state="full">

    <rule id="f3g44k2">
        <conditions>
            <lc:call-identity>
                <lc:sip>
                    <lc:to>
                        <many domain="pompeii.example.com"/>
                    </lc:to>
                </lc:sip>
            </lc:call-identity>
        </conditions>
        <actions>
            <lc:accept alt-action="reject">
                <lc:rate>100</lc:rate>
            </lc:accept>
        </actions>

    </rule>
</ruleset>
```

```

        </lc:to>
        <lc:from>
            <many>
                <except domain="pompeii.example.com"/>
                <except domain="rescue.example.com"/>
            </many>
        </lc:from>
    </lc:sip>
</lc:call-identity>
<validity>
    <from>79-08-24T09:00:00+01:00</from>
    <until>79-08-27T09:00:00+01:00</until>
</validity>
</conditions>
<actions>
    <lc:accept alt-action="forward" alt-target=
        "sip:earthquake@update.example.com">
        <lc:rate>100</lc:rate>
    </lc:accept>
</actions>

</rule>
</ruleset>

```

## 7. XML Schema Definition for Load Control

This section defines the XML schema for the load control document. It extends the Common Policy schema in [RFC4745] in two ways. Firstly, it defines two mandatory attributes for the ruleset element: version and state. The version attribute allows the recipient of the notification to properly order them. Versions start at 0, and increase by one for each new document sent to a subscriber within the same subscription. Versions MUST be representable using a non-negative 32 bit integer. The state attribute indicates whether the document contains a full control policy update, or whether it contains only state delta as partial update. Secondly, it defines new members of the <conditions> and <actions> elements.

```

<?xml version="1.0" encoding="UTF-8"?>
<xs:schema targetNamespace="urn:ietf:params:xml:ns:load-control"
    xmlns:lc="urn:ietf:params:xml:ns:load-control"
    xmlns:cp="urn:ietf:params:xml:ns:common-policy"
    xmlns:xs="http://www.w3.org/2001/XMLSchema"
    elementFormDefault="qualified"
    attributedFormDefault="unqualified">

```

```
<xs:import namespace="urn:ietf:params:xml:ns:common-policy"/>

<!-- RULESET -->

<xs:element name="ruleset">
  <xs:complexType>
    <xs:complexContent>
      <xs:restriction base="xs:anyType">
        <xs:sequence>
          <xs:element name="rule" type="cp:ruleType"
            minOccurs="0" maxOccurs="unbounded"/>
        </xs:sequence>
      </xs:restriction>
    </xs:complexContent>
    <xs:attribute name="version" type="xs:integer" use="required"/>
    <xs:attribute name="state" use="required"/>
    <xs:simpleType>
      <xs:restriction base="xs:string">
        <xs:enumeration value="full"/>
        <xs:enumeration value="partial"/>
      </xs:restriction>
    </xs:simpleType>
  </xs:complexType>
</xs:element>

<!-- CONDITIONS -->

<!-- CALL IDENTITY -->
<xs:element name="call-identity" type="lc:call-type"/>
<xs:element name="method" type="lc:method-type"/>

<!-- CALL TYPE -->
<xs:complexType name="call-type">
  <xs:choice>
    <xs:element name="sip" type="lc:sip-id-type"/>
    <any namespace="##other" processContents="lax" minOccurs="0"
      maxOccurs="unbounded"/>
  </xs:choice>
  <anyAttribute namespace="##other" processContents="lax"/>
</xs:complexType>

<!-- SIP ID TYPE -->
<xs:complexType name="sip-id-type">
  <xs:sequence>
    <element name="from" type="cp:identityType" minOccurs="0"/>
    <element name="to" type="cp:identityType" minOccurs="0"/>
    <element name="request-uri" type="cp:identityType" minOccurs="0"/>
  </xs:sequence>
</xs:complexType>
```

```
<element name="p-asserted-identity" type="cp:identityType"
  minOccurs="0"/>
<any namespace="##other" processContents="lax" minOccurs="0"
  maxOccurs="unbounded"/>
</xs:sequence>
<anyAttribute namespace="##other" processContents="lax"/>
</xs:complexType>

<!-- METHOD TYPE -->
<xs:simpleType name="method-type">
  <xs:restriction base="xs:string">
    <xs:enumeration value="INVITE"/>
    <xs:enumeration value="MESSAGE"/>
    <xs:enumeration value="REGISTER"/>
    <xs:enumeration value="SUBSCRIBE"/>
    <xs:enumeration value="OPTIONS"/>
    <xs:enumeration value="PUBLISH"/>
  </xs:restriction>
</xs:simpleType>

<!-- ACTIONS -->

<xs:element name="accept">
  <xs:choice>
    <element name="rate" type="xs:decimal" minOccurs="0"/>
    <element name="win" type="xs:integer" minOccurs="0"/>
    <element name="percent" type="xs:decimal" minOccurs="0"/>
    <any namespace="##other" processContents="lax" minOccurs="0"
      maxOccurs="unbounded"/>
  </xs:choice>
  <xs:attribute name="alt-action" type="xs:string" default="reject"/>
  <xs:attribute name="alt-target" type="xs:anyURI"/>
  <anyAttribute namespace="##other" processContents="lax"/>
</xs:element>

</xs:schema>
```

## 8. Related Work

### 8.1. Relationship with Load Filtering in PSTN

It is known that the existing PSTN network also uses a load filtering mechanism to prevent overload and the filter configuration is done manually. This specification defines a SIP events framework based distribution mechanism which allows automated filter distribution in suitable environments.

There are control messages associated with PSTN overload control which would specify an outgoing control list, call gap duration and control duration [AINGR]. These items could be roughly correlated to the identity, action and time fields of the SIP load filter defined in this specification. However, the filter defined in this specification is much more generic and flexible as opposed to its PSTN counterpart.

Firstly, PSTN load filtering only applies to telephone numbers, and the number of prefix to be matched for a group of telephone numbers is usually a fixed set. The SIP filter identity allows both SIP URI and telephone numbers (through Tel URI) to be specified. The identities can be arbitrarily grouped by SIP domains or any number of leading prefix of the telephone numbers.

Secondly, the PSTN filtering action is usually limited to call gapping with a fixed set of allowed gapping intervals. The action field in the SIP load filter allows more flexible rate throttle and other possibilities.

Thirdly, the duration field in PSTN filtering specifies a value in seconds for the control duration only, and the allowed values are mapped into a value set. The time field in the SIP load filter may specify not only a duration, but also a future activation time which could be especially useful for automating load control for predictable overloads.

PSTN filtering can be performed in both edge switches and transit switches; SIP filtering can also be applied in both edge proxy servers and core proxy servers, and even in capable user agents.

PSTN overload control also has special accommodation for High Probability of Completion (HPC) calls, which would be similar to the calls designated by the SIP Resource Priority Headers [RFC4412]. SIP filtering mechanism can also prioritize the treatment of these calls by specifying favorable actions for these calls.

PSTN filtering also provides administrative option for routing failed call attempts to either Recorder Tone or a special announcement. Similar capability can be provided in the SIP filtering mechanism by specifying the appropriate "alt-action" attribute in the SIP filtering action field.

## 8.2. Relationship with Other IETF SIP Load Control Efforts

The load filtering rules in this specification consists of identity, action and time. The identity can range from a single specific user to an arbitrary user aggregate, domains or areas. The user can be



identified by either the source or the destination. When the user is identified by the source and a favorable action is specified, the result is to some extent similar to identifying a priority user based on authorized Resource Priority Headers [RFC4412] in the requests. Specifying a source user identity with an unfavorable action would cause an effect to some extent similar to an inverse SIP resource priority mechanism.

The load filter defined in this specification is generic and expected to be applicable not only to the load filtering mechanism but also to the feedback overload control mechanism in [I-D.ietf-soc-overload-control]. In particular, both mechanisms could use specific or wildcard filter identities for load control and could share well-known load control actions. The time duration field in the load filter could also be used in both mechanisms. As mentioned in Section 1, the load filter distribution mechanism and the feedback overload control mechanism address complementary areas in the load control problem space. Load filtering is more proactive and focuses on distributing the filter towards the source of the traffic; the hop-by-hop feedback based approach is reactive and targets more at traffic already accepted in the network. Therefore, they could also make different use of the generic filter components. For example, the load filtering mechanism may use the time field in the filter to specify not only a control duration but also a future activation time to accommodate a predictable overload such as the one caused by Mother's Day greetings or a viewer-voting program; the feedback-based control might not need to use the time field or might use the time field to specify an immediate control duration.

#### 9. Discussion of this specification meeting the requirements of RFC5390

This section evaluates whether the load control event package defined in this specification satisfies the various SIP overload control requirements set forth by RFC5390 [RFC5390]. Not all RFC5390 requirements are found applicable due to the scope of this document. Therefore, we categorize the assessment results into Yes (meet the requirement), P/A (partially applicable), No (must be used in conjunction with another mechanism to meet the requirement), and N/A (not applicable).

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.

P/A. The goal of the load filtering is to prevent overload or maintain overall goodput during the time of overload, but it is dependent on the advance predictions of the load. If the predictions are incorrect, in either direction, the effectiveness of the mechanism will be affected.

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.

N/A if filter values are installed in advance and do not change during the potential overload period. P/A if filter values are dynamically adjusted due to the specific filter computation algorithm. The dynamic filter computation algorithm is outside the scope of this specification, while the distribution of the updated filters uses the event package mechanism of this specification.

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.

No. This mechanism is entirely dependent on advance configuration, based on advance knowledge. In order to satisfy Req 3, it should be used in conjunction with other mechanisms which are not based on advance configuration.

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.

No. This mechanism is entirely dependent on the participation of all possible neighbors. In order to satisfy Req 4, it should be used in conjunction with other mechanisms, some of which are described in Section 4.4.

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.

No. This mechanism is entirely dependent on the non-malicious participation of all possible neighbors. In order to satisfy Req 5, it should be used in conjunction with other mechanisms, some of which are described in Section 4.4.

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.

N/A. This mechanism signals anticipated overload, not actual overload. However the signals in this mechanism are not used for any other purpose.

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.

Yes. This event package allows rate/loss/windows-based overload control options as discussed in Section 6.4.

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.

N/A to the load control event package itself.

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.

Yes. For example, the load filter [Section 4.1] allows the inclusion of alternative forwarding destinations for rejected requests.

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.

No. Because this mechanism requires advance configuration of specifically identified neighbors, it does not support environments where the number and identity of the upstream neighbors are not known in advance. In order to satisfy Req 10, it should be used in conjunction with other mechanisms.

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.

Yes. See also answer to REQ 10.

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

Yes. The load control event package is not limited by domain boundaries. However, it is likely more applicable in intra-domain scenarios than in inter-domain scenarios due to security and other concerns (See also Section 4.4).

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.

P/A. This mechanism does not specifically address the prioritizing of work during times of overload. But it does not preclude any particular local policy.

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.

N/A to the load control event package itself.

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.

P/A. Because the filters are provisioned in advance, they are not affected by the overload or failure of other nodes. But, on the other hand, they may not, in those cases, be able to protect the overloaded node (see Req 1).

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

Yes. The standardized SIP event package mechanism RFC3265 [RFC3265] is used.

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

P/A. This mechanism does provide a potential avenue for malicious attacks. Therefore the security mechanisms for SIP event packages in general [RFC3265] and of section 10 of this specification should be used.

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.

Yes. The identity of load indication is covered in the filter format definition in Section 4.1.

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.

N/A to the load control event package itself.

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.

No. This mechanism is entirely dependent on the participation of all possible neighbors. In order to satisfy Req 20, it should be used in conjunction with other mechanisms, some of which are described in

## Section 4.4.

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.

N/A to the load control event package itself.

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.

N/A to the load control event package itself.

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.

Yes. The load control event package does not preclude its use in a scenario with server farms.

## 10. Security Considerations

Two aspects of security considerations arise from this specification. One is the SIP event framework based filter distribution mechanism, the other is the filter enforcement mechanism.

Security considerations for SIP event framework based mechanisms are covered in Section 5 of [RFC3265]. A particularly relevant security concern for this event package is that if the notifiers can be spoofed, attackers can send fake notifications asking subscribers to throttle all traffic, leading to Denial-of-Service attacks. Therefore, all load control notification MUST be authenticated and authorized before being accepted. Standard authentication and authorization mechanisms recommended in [RFC3261] such as TLS [RFC5246] and IPSec [RFC4301] may serve this purpose. On the other hand, if a legitimate notifier is itself compromised, additional mechanisms will be needed to detect the attack.

Security considerations for filter enforcements vary depending on the filter itself. This specification defines possible filter match of the following SIP header fields: <from>, <to>, <request-uri> and

<p-asserted-identity>. The exact requirement to authenticate and authorize these fields is up to the service provider. In general, if the identity field represents the source of the request, it SHOULD be authenticated and authorized; if the identity field represents the destination of the request, the authentication and authorization is optional.

## 11. IANA Considerations

This specification registers a SIP event package, a new MIME type, a new XML namespace, and a new XML schema.

### 11.1. Load Control Event Package Registration

This section registers an event package based on the registration procedures defined in [RFC3265].

Package name: load-control

Type: package

Published specification: This specification

Person to contact: Charles Shen, charles@cs.columbia.edu

### 11.2. application/load-control+xml MIME Registration

This section registers a new MIME type based on the procedures defined in [RFC4288] and guidelines in [RFC3023].

MIME media type name: application

MIME subtype name: load-control+xml

Mandatory parameters: none

Optional parameters: Same as charset parameter application/xml in [RFC3023]

Encoding considerations: Same as encoding considerations of application/xml in [RFC3023]

Security considerations: See Section 10 of [RFC3023] and Section 10 of this specification

Interpretability considerations: None

Published Specification: This specification

Applications which use this media type: load control of SIP entities

Additional information:

Magic number: None

File extension: .xml

Macintosh file type code: 'TEXT'

Personal and email address for further information:

Charles Shen, charles@cs.columbia.edu

Intended usage: COMMON

Author/Change Controller: IETF SOC Working Group

<sip-overload@ietf.org>, as designated by the IESG <iesg@ietf.org>

### 11.3. Load Control Schema Registration

URI: urn:ietf:params:xml:schema:load-control

Registrant Contact: IETF SOC working group, Charles Shen  
(charles@cs.columbia.edu).

XML: the XML schema to be registered is contained in Section 7.

Its first line is

```
<?xml version="1.0" encoding="UTF-8"?>
```

and its last line is

```
</xs:schema>
```

## 12. Acknowledgements

The authors would like to thank Bruno Chatras, Martin Dolly, Keith Drage, Ashutosh Dutta, Janet Gunn, Vijay Gurbani, Volker Hilt, Geoff Hunt, Hadriel Kaplan, Paul Kyzivat, Salvatore Loreto, Timothy Moran, Eric Noel, Parthasarathi R, Shida Schubert, Robert Sparks, Phil Williams and other members of the SOC and SIPPING working group for many helpful comments. In addition, Bruno Chatras proposed the <method> condition element. Janet Gunn provided detailed text



suggestions for Section 9. Shida made many suggestions about terminology usage. Phil Williams suggested adding support for delta updates. Ashutosh Dutta gave pointers to PSTN references.

### 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.
- [RFC2141] Moats, R., "URN Syntax", RFC 2141, May 1997.
- [RFC3023] Murata, M., St. Laurent, S., and D. Kohn, "XML Media Types", RFC 3023, January 2001.
- [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.
- [RFC3265] Roach, A., "Session Initiation Protocol (SIP)-Specific Event Notification", RFC 3265, June 2002.
- [RFC3688] Mealling, M., "The IETF XML Registry", BCP 81, RFC 3688, January 2004.
- [RFC3966] Schulzrinne, H., "The tel URI for Telephone Numbers", RFC 3966, December 2004.
- [RFC4288] Freed, N. and J. Klensin, "Media Type Specifications and Registration Procedures", BCP 13, RFC 4288, December 2005.
- [RFC4745] Schulzrinne, H., Tschofenig, H., Morris, J., Cuellar, J., Polk, J., and J. Rosenberg, "Common Policy: A Document Format for Expressing Privacy Preferences", RFC 4745, February 2007.

#### 13.2. Informative References

- [AINGR] Bell Communications Research, "AINGR: Service Control Point (SCP) Network Traffic Management", GR-2938-CORE , December 1996.
- [I-D.ietf-soc-overload-control] Gurbani, V., Hilt, V., and H. Schulzrinne, "Session Initiation Protocol (SIP) Overload Control",

draft-ietf-soc-overload-control-07 (work in progress),  
January 2012.

- [RFC2648] Moats, R., "A URN Namespace for IETF Documents", RFC 2648, August 1999.
- [RFC4301] Kent, S. and K. Seo, "Security Architecture for the Internet Protocol", RFC 4301, December 2005.
- [RFC4412] Schulzrinne, H. and J. Polk, "Communications Resource Priority for the Session Initiation Protocol (SIP)", RFC 4412, February 2006.
- [RFC4825] Rosenberg, J., "The Extensible Markup Language (XML) Configuration Access Protocol (XCAP)", RFC 4825, May 2007.
- [RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, August 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.

#### Authors' Addresses

Charles Shen  
AT&T Security Research Center  
33 Thomas Street  
New York, NY 10007  
USA

Phone: +1 212 513 2081  
Email: shen@att.com

Henning Schulzrinne  
Columbia University  
Department of Computer Science  
1214 Amsterdam Avenue, MC 0401  
New York, NY 10027  
USA

Phone: +1 212 939 7004  
Email: schulzrinne@cs.columbia.edu

Arata Koike  
NTT Service Integration Labs &  
3-9-11 Midori-cho Musashino-shi  
Tokyo, 184-0013  
Japan

Phone: +81 422 59 6099  
Email: koike.arata@lab.ntt.co.jp



SOC Working Group

Internet-Draft

Intended status: Standards Track

Expires: January 7, 2013

V. Gurbani, Ed.

V. Hilt

Bell Laboratories, Alcatel-Lucent

H. Schulzrinne

Columbia University

July 6, 2012

Session Initiation Protocol (SIP) Overload Control  
draft-ietf-soc-overload-control-09

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 January 7, 2013.

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 . . . . .   | 4  |
| 2. Terminology . . . . .  | 5  |
| 3. Overview of operations . . . . .   | 5  |
| 4. Via header parameters for overload control . . . . .                             | 6  |
| 4.1. The oc parameter . . . . .   | 6  |
| 4.2. The oc-algo parameter . . . . .  | 7  |
| 4.3. The oc-validity parameter . . . . .  | 7  |
| 4.4. The oc-seq parameter . . . . .   | 8  |
| 5. General behaviour . . . . .  | 8  |
| 5.1. Handshake to determine support for overload control . . . . .                  | 9  |
| 5.2. Creating and updating the overload control parameters . . . . .                | 9  |
| 5.3. Determining the 'oc' Parameter Value . . . . .                                 | 11 |
| 5.4. Processing the Overload Control Parameters . . . . .                           | 11 |
| 5.5. Using the Overload Control Parameter Values . . . . .                          | 12 |
| 5.6. Forwarding the overload control parameters . . . . .                           | 12 |
| 5.7. Terminating overload control . . . . .   | 13 |
| 5.8. Stabilizing overload algorithm selection . . . . .                             | 13 |
| 5.9. Self-Limiting . . . . .  | 14 |
| 5.10. Responding to an Overload Indication . . . . .                                | 14 |
| 5.10.1. Message prioritization at the hop before the<br>overloaded server . . . . . | 14 |
| 5.10.2. Rejecting requests at an overloaded server . . . . .                        | 15 |
| 5.11. 100-Trying provisional response and overload control<br>parameters . . . . .  | 15 |
| 6. The loss-based overload control scheme . . . . .                                 | 16 |
| 6.1. Special parameter values for loss-based overload<br>control . . . . .          | 16 |
| 6.2. Example . . . . .  | 17 |
| 6.3. Default algorithm for loss-based overload control . . . . .                    | 18 |
| 7. Relationship with other IETF SIP load control efforts . . . . .                  | 22 |
| 8. Syntax . . . . .   | 22 |
| 9. Design Considerations . . . . .  | 23 |
| 9.1. SIP Mechanism . . . . .  | 23 |
| 9.1.1. SIP Response Header . . . . .  | 23 |
| 9.1.2. SIP Event Package . . . . .  | 24 |
| 9.2. Backwards Compatibility . . . . .  | 25 |
| 10. Security Considerations . . . . .   | 25 |
| 11. IANA Considerations . . . . .   | 26 |
| 12. References . . . . .  | 27 |
| 12.1. Normative References . . . . .  | 27 |
| 12.2. Informative References . . . . .  | 27 |
| Appendix A. Acknowledgements . . . . .  | 28 |
| Appendix B. RFC5390 requirements . . . . .  | 28 |
| Authors' Addresses . . . . .  | 34 |

## 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 with a default value of "loss".

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.ietf-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 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.

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. 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.

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.ietf-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 the overload control feedback specified in 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 prevents the "oc" value from timing out before the next response is sent and allows clients to discard stale "oc" values if they have not communicated with a server for some time. 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 used 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.ietf-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.ietf-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 pl.example.net;
    branch=z9hG4bK2d4790.1;oc;oc-algo="loss,A"
...

SIP/2.0 100 Trying
Via: SIP/2.0/TLS pl.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 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 pl.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 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 pl.example.net;
    branch=z9hG4bK2d4790.4;received=192.0.2.111;
    oc=0;oc-algo="loss";oc-validity=0;oc-seq=1282321892.439
...
```

### 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 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 --- 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 perform overload control
    r := random()
    drop_msg := false

    if (cat1 >= cat2) {
        category := assign_msg_to_category(sip_msg)
        pct_to_reduce_cat2 := 0
        pct_to_reduce_cat1 := oc_value / cat1 * 100
        if (pct_to_reduce_cat1 > 100) {
            // Get remaining messages from category 2
            pct_to_reduce_cat2 := 100 - pct_to_reduce_cat1
            pct_to_reduce_cat1 := 100
        }

        if (category == cat1) {
            if (r <= pct_to_reduce_cat1) {
                drop_msg := true
            }
        }
        else { // Message from category 2
            if (r <= pct_to_reduce_cat2) {
                drop_msg := true
            }
        }
    }
    else { // More category 2 messages than category 1;
        // indicative of an emergency situation. Since
        // there are more category 2 messages, don't bother
        // distinguishing between category 1 or 2 --- treat
        // them equal (for simplicity).
        if (r <= oc_value)
            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 locally by
        // generating response (if a proxy) or treating 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 5-10s) 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. Assume that 450 out of 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, or a 90/10 mix will be used in overload calculations.

## 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:

- 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.

### 9.1.1.2. 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.ietf-soc-overload-rate-control]  
Noel, E. and P. Williams, "Session Initiation Protocol (SIP) Rate Control", draft-ietf-soc-overload-rate-control-02 (work in progress), June 2012.
- [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 Bommel for their contributions to this specification.

Adam Roach and Eric McMurphy 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](mailto:hgs@cs.columbia.edu)  
URI: <http://www.cs.columbia.edu>





SOC Working Group  
Internet-Draft  
Intended status: Standards Track  
Expires: November 1, 2012

Eric Noel  
AT&T Labs  
Philip M Williams  
BT Innovate & Design

June 1, 2012

Session Initiation Protocol (SIP) Rate Control  
draft-ietf-soc-overload-rate-control-02.txt

Abstract

The prevalent use of Session Initiation Protocol (SIP) [RFC3261] in Next Generation Networks necessitates that SIP networks provide adequate control mechanisms to maintain transaction throughput by preventing congestion collapse during traffic overloads. Already [draft-ietf-soc-overload-control-08] proposes a loss-based solution to remedy known vulnerabilities of the [RFC3261] SIP 503 (service unavailable) overload control mechanism. This document proposes a rate-based control solution to complement the loss-based control defined in [draft-ietf-soc-overload-control-08].

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 November 1, 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.....  | 2  |
| 2. Terminology.....   | 3  |
| 3. Rate-based algorithm scheme.....                             | 4  |
| 3.1. Overview.....  | 4  |
| 3.2. Summary of via headers parameters for overload control.... | 4  |
| 3.3. Client and server rate-control algorithm selection.....    | 5  |
| 3.4. Server operation.....                                      | 5  |
| 3.5. Client operation.....                                      | 6  |
| 3.5.1. Default algorithm.....                                   | 6  |
| 3.5.2. Optional enhancement: avoidance of resonance.....        | 10 |
| 4. Example.....   | 11 |
| 5. Syntax.....  | 12 |
| 6. Security Considerations.....                                 | 13 |
| 7. IANA Considerations.....                                     | 14 |
| 8. References.....  | 14 |
| 8.1. Normative References.....                                  | 14 |
| 8.2. Informative References.....                                | 14 |
| Appendix A. Contributors.....                                   | 15 |
| Appendix B. Acknowledgments.....                                | 15 |

## 1. Introduction

The use of SIP in large scale Next Generation Networks requires that SIP based networks provide adequate control mechanisms for handling traffic growth. In particular, SIP networks must be able to handle traffic overloads gracefully, maintaining transaction throughput by preventing congestion collapse.

A promising SIP based overload control solution has been proposed in [draft-ietf-soc-overload-control-08]. That solution provides a communication scheme for overload control algorithms. It also includes a default loss-based overload control algorithm that makes

it possible for a set of clients to limit offered load towards an overloaded server.

However, such loss control algorithm is sensitive to variations in load so that any increase in load would be directly reflected by the clients in the offered load presented to the overloaded servers. More importantly, a loss-based control cannot guarantee clients to produce a bounded offered load from the clients towards an overloaded server and requires frequent updates which may have implications for stability.

This document proposes extensions to [draft-ietf-soc-overload-control-08] to support a rate-based control that guarantees an upper bound on the rate, constant between server updates, of requests sent by clients towards an overloaded server.. The tradeoff is in terms of algorithmic complexity, since the overloaded server must estimate a separate target for each client, rather than an overall loss percentage, equally applicable to all clients.

The proposed rate-based overload control algorithm mitigates congestion in SIP networks while adhering to the overload signaling scheme in [draft-ietf-soc-overload-control-08] and presenting a rate based control as an optional alternative to the default loss-based control in [draft-ietf-soc-overload-control-08].

## 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].

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. Rate-based algorithm scheme

#### 3.1. Overview

The server is what the overload control algorithm defined here protects and the client is what throttles traffic towards the server.

Following the procedures defined in [draft-ietf-soc-overload-control-08], the server and clients signal one another support for rate-based overload control.

Then periodically, the server relies on internal measurements (e.g. CPU utilization, queueing delay...) to evaluate its overload state and estimate a target SIP request rate (as opposed to target percent loss in the case of loss-based control).

When in overload, the server uses [draft-ietf-soc-overload-control-08] via header oc parameters of SIP responses to inform the clients of its overload state and of the target SIP request rate.

Upon receiving the oc parameters with a target SIP request rate, each client throttles new SIP requests towards the overloaded server.

#### 3.2. Summary of via headers parameters for overload control

oc: Used by SIP clients to indicate draft-ietf-soc-overload-control-08 support and by SIP servers to indicate the load reduction amount.

oc parameters defined in draft-ietf-soc-overload-control-08 are summarized below:

oc-algo: Used by SIP clients to advertise supported overload control algorithms and by SIP servers to notify clients of algorithm in effect. Support values: loss (default), rate (optional).

oc-validity: Used by SIP servers to indicate an interval of time (msec) that the load reduction should be in effect. A value of 0 is reserved for server to stop overload control. A non-zero value is required in conjunction with an "oc" parameter.

oc-seq: A sequence number associated with the "oc" parameter.

The use of the via header oc parameter(s) inform of the desired rate, but they don't explicitly ''inform clients of the overload state''.

### 3.3. Client and server rate-control algorithm selection

Per [draft-ietf-soc-overload-control-08], new clients indicate supported overload control algorithms to servers by inserting oc and oc-algo, with the names of the supported algorithms, in Via header of SIP requests destined to servers. The inclusion by the client of the string ''rate'' indicates that the client supports a rate based algorithm. Conversely, servers notify clients of selected overload control algorithm through the oc-algo parameter in the Via header of SIP responses to clients. The inclusion by the server of the string ''rate'' indicates that the rate based algorithm has been selected by the server.

Support of rate-based control MUST be indicated by clients setting oc-algo to "rate". Selection of rate-based control MUST be indicated by servers by setting oc-algo to ''rate''.

### 3.4. Server operation

The actual algorithm used by the server to determine its overload state and estimate a target SIP request rate is beyond the scope of this document.

However, the server MUST be able to evaluate periodically its overload state and estimate a target SIP request rate beyond which it would become overloaded. The server must allocate a portion of the target SIP request rate to each of its client. The server may set the same rate for every client, or may set a different rates for different clients.

The max rate determined by the server for a client applies to the entire stream of SIP requests, even though throttling may only affect a particular subset of the requests, since as per [draft-ietf-soc-overload-control-08] and REQ 13 of RFC 5390, request prioritization is the client responsibility. But when deriving this rate the server may need to take into account characteristics of the requests, and the effect of the client prioritization on the load it

receives, e.g. CPU utilization will depend upon the characteristics of the requests which would presumably allow the server to take in account prioritization.

Note that the target SIP request rate is a max rate that may not be attained by the arrival rate at the client, and the server cannot assume that it will.

Upon detection of overload, the server **MUST** follow the specifications in [draft-ietf-soc-overload-control-08] to notify its clients of the allocated target SIP request rate.

The server **MUST** use [draft-ietf-soc-overload-control-08] oc parameter to send a target SIP request rate to each of its clients.

### 3.5. Client operation

#### 3.5.1. Default algorithm

To throttle new SIP requests at the rate specified in the oc value sent by the server to its clients, the client **MAY** use the proposed default algorithm for rate-based control or any other equivalent algorithm.

The default Leaky Bucket algorithm presented here is based on [ITU-T Rec. I.371] Appendix A.2. The algorithm makes it possible for clients to deliver SIP requests at a rate specified in the oc value with tolerance parameter TAU (preferably configurable).

Conceptually, the Leaky Bucket algorithm can be viewed as a finite capacity bucket whose real-valued content drains out at a continuous rate of 1 unit of content per time unit and whose content increases by the increment T for each forwarded SIP request. T is computed as the inverse of the rate specified in the oc value, namely  $T = 1 / \text{oc-value}$ .

Note that when the oc-value is 0 with a non-zero oc-validity, then the client should reject 100% of SIP requests destined to the overload server. However, when both oc-value and oc-validity are 0, the client should immediately stop throttling.

If at a new SIP request arrival the content of the bucket is less than or equal to the limit value TAU, then the SIP request is forwarded to the server; otherwise, the SIP request is rejected.

Note that the capacity of the bucket (the upper bound of the counter) is  $(T + \text{TAU})$ .

The tolerance parameter TAU determines how close the long-term admitted rate is to an ideal control that would admit all SIP requests for arrival rates less than  $1/T$  and then admit SIP requests precisely rate at  $1/T$  for arrival rates above  $1/T$ . In particular at mean arrival rates close to  $1/T$ , it determines the tolerance to deviation of the inter-arrival time from  $T$  (the larger TAU the more tolerance to deviations from the inter-departure interval  $T$ ). This deviation from the inter-departure interval influences the admitted rate burstiness, or the number consecutive SIP requests forwarded to the SIP server (burst size proportional to TAU over the difference between  $1/T$  and the arrival rate).

SIP servers with a very large number of clients, each with a relatively small arrival rate, will generally benefit from a smaller value for TAU in order to limit queuing (and hence response times) at the server when subjected to a sudden surge of traffic from all clients. Conversely, a SIP server with a relatively small number of clients, each with proportionally larger arrival rate, will benefit from a larger value of TAU.

At the arrival time of the  $k$ -th new SIP request  $ta(k)$  after control has been activated, the content of the bucket is provisionally updated to the value

$$X' = X - ([ta(k) - LCT])$$

where  $X$  is the content of the bucket after arrival of the last forwarded SIP request, and  $LCT$  is the time at which the last SIP request was forwarded.

If  $X'$  is less than or equal to the limit value TAU, then the new SIP request is forwarded and the bucket content  $X$  is set to  $X'$  (or to 0 if  $X'$  is negative) plus the increment  $T$ , and  $LCT$  is set to the current time  $ta(k)$ . If  $X'$  is greater than the limit value TAU, then

the new SIP request is rejected and the values of X and LCT are unchanged.

When the first response from the server has been received indicating control activation ( $oc\_validity > 0$ ), LCT is set to the time of activation, and the occupancy of the bucket is initialized to the parameter TAU0 (preferably configurable) which is 0 or larger but less than or equal to TAU.

Following [draft-ietf-soc-overload-control-0808], the client is responsible for message priority and for maintaining two categories of requests: Requests candidate for reduction, requests not subject to reduction (except under extenuating circumstances when there aren't any messages in the first category that can be reduced).

Accordingly, the proposed Leaky bucket implementation is modified to support priority using two thresholds for SIP requests in the set of requests candidate for reduction. With two priorities, the proposed Leaky bucket requires two thresholds  $TAU1 < TAU2$ :

- . All new requests would be admitted when the bucket fill is at or below TAU1,
- . Only higher priority requests would be admitted when the bucket fill is between TAU1 and TAU2,
- . All requests would be rejected when the bucket fill is above TAU2.

This can be generalized to n priorities using n thresholds for  $n > 2$  in the obvious way.

With a priority scheme that relies on two tolerance parameters (TAU2 influences the priority traffic, TAU1 influences the non-priority traffic), always set  $TAU1 \leq TAU2$  (TAU is replaced by TAU1 and TAU2). Setting both tolerance parameters to the same value is equivalent to having no priority. TAU1 influences the admitted rate the same way as TAU does when no priority are set. And the larger the difference between TAU1 and TAU2, the closer to the control is to strict priority.

TAU (or TAU1 and TAU2) can assume any positive real number value and is not necessarily bounded by T.



Note that specification of a value for TAU is beyond the scope of this document.

A reference algorithm is shown below.

No priority case:

```
// T: emission interval, set to 1 / TargetRate
// TAU: tolerance parameter
// ta: arrival time of last arrival
// LCT: arrival time of last conforming SIP request (initialized to
the first arrival time)
// X: current value of leaky bucket counter (initialized to 0)

// After first arrival, calculate auxiliary variable Xp
Xp = X - (ta - LCT);

if (Xp <= TAU) {
    // Accept SIP request
    // Update X and LCT
    X = max(0,Xp) + T;
    LCT = ta;
} else {
    // Reject SIP request
    // Do not update X and LCT
}
```

Priority case:

```
// T: emission interval, set to 1 / TargetRate
// TAU1: tolerance parameter of no priority SIP requests
// TAU2: tolerance parameter of priority SIP requests
// ta: arrival time of last arrival
// LCT: arrival time of last conforming SIP request (initialized to
the first arrival time)
// X: current value of leaky bucket counter (initialized to 0)

// After first arrival, calculate auxiliary variable Xp
Xp = X - (ta - LCT);
```

```
if (AnyRequestReceived && Xp <= TAU1 || PriorityRequestReceived &&
Xp <= TAU2 && Xp > TAU1) {
    // Accept SIP request
    // Update X and LCT
    X = max(0,Xp) + T;
    LCT = ta;
} else {
    // Reject SIP request
    // Do not update X and LCT
}
```

### 3.5.2. Optional enhancement: avoidance of resonance

As the number of client sources of traffic increases and the throughput of the server decreases, the maximum rate admitted by each client needs to decrease, and therefore the value of  $T$  becomes larger. Under some circumstances, e.g. if the traffic arises very quickly simultaneously at many sources, the occupancies of each bucket can become synchronized, resulting in the admissions from each source being close in time and batched or very 'peaky' arrivals at the server, which not only gives rise to control instability, but also very poor delays and even lost messages. An appropriate term for this is 'resonance' [Erramilli].

If the network topology is such that this can occur, then a simple way to avoid this is to randomize the bucket occupancy at two appropriate points: At the activation of control, and whenever the bucket empties, as follows.

After updating the bucket occupancy to  $X'$ , generate a value  $u$  as follows:

```
if  $X' > 0$ , then  $u=0$ 
```

```
else if  $X' \leq 0$  then uniformly distributed between  $-1/2$  and  $+1/2$ 
```

Then (only) if the arrival is admitted, increase the bucket by an amount  $T + uT$ , which will therefore be just  $T$  if the bucket hadn't emptied, or lie between  $T/2$  and  $3T/2$  if it had.

This randomization should also be done when control is activated, i.e. instead of simply initializing the bucket fill to  $TAU0$ , initialize it to  $TAU0 + uT$ , where  $u$  is uniformly distributed as above. Since activation would have been a result of response to a request sent by the client, the second term in this expression can

be interpreted as being the bucket increment following that admission.

This method has the following characteristics:

- . If  $\text{TAU}_0$  is chosen to be equal to  $\text{TAU}$  and all sources were to activate control at the same time due to an extremely high request rate, then the time until the first request admitted by each client would be uniformly distributed over  $[0, T]$ ;
- . The maximum occupancy is  $\text{TAU} + (3/2)T$ , rather than  $\text{TAU} + T$  without randomization;
- . For the special case of 'classic gapping' where  $\text{TAU}=0$ , then the minimum time between admissions is uniformly distributed over  $[T/2, 3T/2]$ , and the mean time between admissions is the same, i.e.  $T+1/R$  where  $R$  is the request arrival rate;
- . At high load randomization rarely occurs, so there is no loss of precision of the admitted rate, even though the randomized 'phasing' of the buckets remains.

#### 4. Example

Adapting [draft-ietf-soc-overload-control-08] example in section 6.2 where SIP client P1 sends requests to a downstream server P2:

```
INVITE sips:user@example.com SIP/2.0

Via: SIP/2.0/TLS p1.example.net;
branch=z9hG4bK2d4790.1;received=192.0.2.111;
oc;oc-algo="loss,rate"
...

SIP/2.0 100 Trying

Via: SIP/2.0/TLS p1.example.net;
branch=z9hG4bK2d4790.1;received=192.0.2.111;
```

```
oc-algo="rate";oc-validity=0;

oc-seq=1282321615.781
```

...

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 rate.

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 "rate" based scheme and sends that back to P1 in the "oc-algo" parameter. It uses oc-validity=0 to indicate no overload.

At some later time, P2 starts to experience overload. It sends the following SIP message indicating P1 should send SIP requests at a rate no greater than or equal to 150 SIP requests per seconds.

```
SIP/2.0 180 Ringing

Via: SIP/2.0/TLS pl.example.net;

branch=z9hG4bK2d4790.1;received=192.0.2.111;

oc=150;oc-algo="rate";oc-validity=1000;

oc-seq=1282321615.782
```

...

## 5. Syntax

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

```
oc           = "oc" EQUAL oc-value

oc-value     = "NaN" / oc-num

oc-num       = 1*DIGIT
```

## 6. Security Considerations

For completeness, draft-ietf-soc-overload-control-08 Security Considerations section is repeated here.

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.

## 7. IANA Considerations

None.

## 8. References

### 8.1. Normative References

[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.

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

### 8.2. Informative References

[draft-ietf-soc-overload-control-08]  
Gurbani, V., Hilt, V., Schulzrinne, H., "Session Initiation Protocol (SIP) Overload Control", draft-ietf-soc-overload-control-08.

[ITU-T Rec. I.371]  
"Traffic control and congestion control in B-ISDN", ITU-T Recommendation I.371.

[Erramilli]  
A. Erramilli and L. J. Forys, "Traffic Synchronization Effects In Teletraffic Systems", ITC-13, 1991.

#### Appendix A. Contributors

Significant contributions to this document were made by Janet Gunn.

#### Appendix B. Acknowledgments

Many thanks for comments and feedback on this document to: Volker Hilt.

This document was prepared using 2-Word-v2.0.template.dot.

#### Authors' Addresses

Eric Noel  
AT&T Labs  
200s Laurel Avenue  
Middletown, NJ, 07747  
USA

Philip M Williams  
BT Innovate & Design  
UK

