IETF SOC Working Group
Internet-Draft
Intended status: Standards Track
Expires: July 21, 2011

C. Shen
H. Schulzrinne
Columbia U.
A. Koike
NTT

January 17, 2011

A Session Initiation Protocol (SIP) Load Control Event Package
draft-ietf-soc-load-control-event-package-00.txt

Abstract

This document defines a load control event package for the Session Initiation Protocol (SIP). It allows SIP servers to distribute user load control information to other SIP servers in the network. The load control can 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 July 21, 2011.

Copyright Notice

Table of Contents

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 if core or destination-specific SIP proxies are overloaded. By their nature, they need to distribute rate, drop or window information to all upstream SIP proxies and generally affect all calls equally, regardless of destination. However, feedback control is ineffective for edge-server overload. For example, for the ticket giveaway case, almost all such calls will fail in the core SIP server. If the edge server is 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 filters that indicate that calls to specific destinations or from specific sources should be rate-limited or randomly dropped. These 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 filters work best if they prevent calls as close to the user agent client as possible.

Performing SIP load filtering control requires three components: filter content format definition, filter content computation methods, and filter distribution mechanism. This document addresses two of the three components. The filter format is defined by the contents

of a SIP load control event package, while the filter distribution mechanism is built upon the existing SIP event framework. The remaining component, filter content computation, depends heavily on the actual network topology and service provider policies. Therefore it is out of scope of this document.

The rest of this document is structured as follows: we begin by listing the design requirements for this work in Section 3. We then give an overview of the load filtering control operation in Section 4. The load control event package is detailed in Section 5. The load filter content format definition is discussed in the two sections that follow, with Section 6 defining the load control XML document and Section 7 defining the corresponding XML schema. Section 8 relates this work to corresponding mechanisms in PSTN and other IETF efforts addressing SIP load control. Section 9 evaluates whether this document 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 control mechanism needs to satisfy the following requirements:

- To simplify the solution, we focus on SIP load control, rather than a generic application-layer mechanism.
- The load filter information needs to be distributed efficiently to possibly a large subset of all SIP elements.
- The solution should re-use existing SIP protocol mechanisms to reduce implementation and deployment complexity.
- For predictable overload situations, such as holidays and call-in events, the mechanism should specify during what time period it is to be applied, so that the information can be distributed ahead of time.
- For destination-specific overload situations, the load filter needs to be able to describe the callee.
- To address accidental and intentional high-volume call generators, the filter should allow to specify the caller.
- 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. Load Filtering Control 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 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

The load filter content computation method needs to take into consideration information such as the overload time, scope and the 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 that both prevents overload and minimizes the likelihood of network 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 filter content computation algorithm is out of scope of this document.

4.3. Filter Distribution

For load filter distribution, this document 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 document also defines the XML schema used to encode the load control document. The choice of XML allows us to reuse existing SIP-specific policy related XML schemas when applicable, and also

fits our goal of flexibility and extensibility.

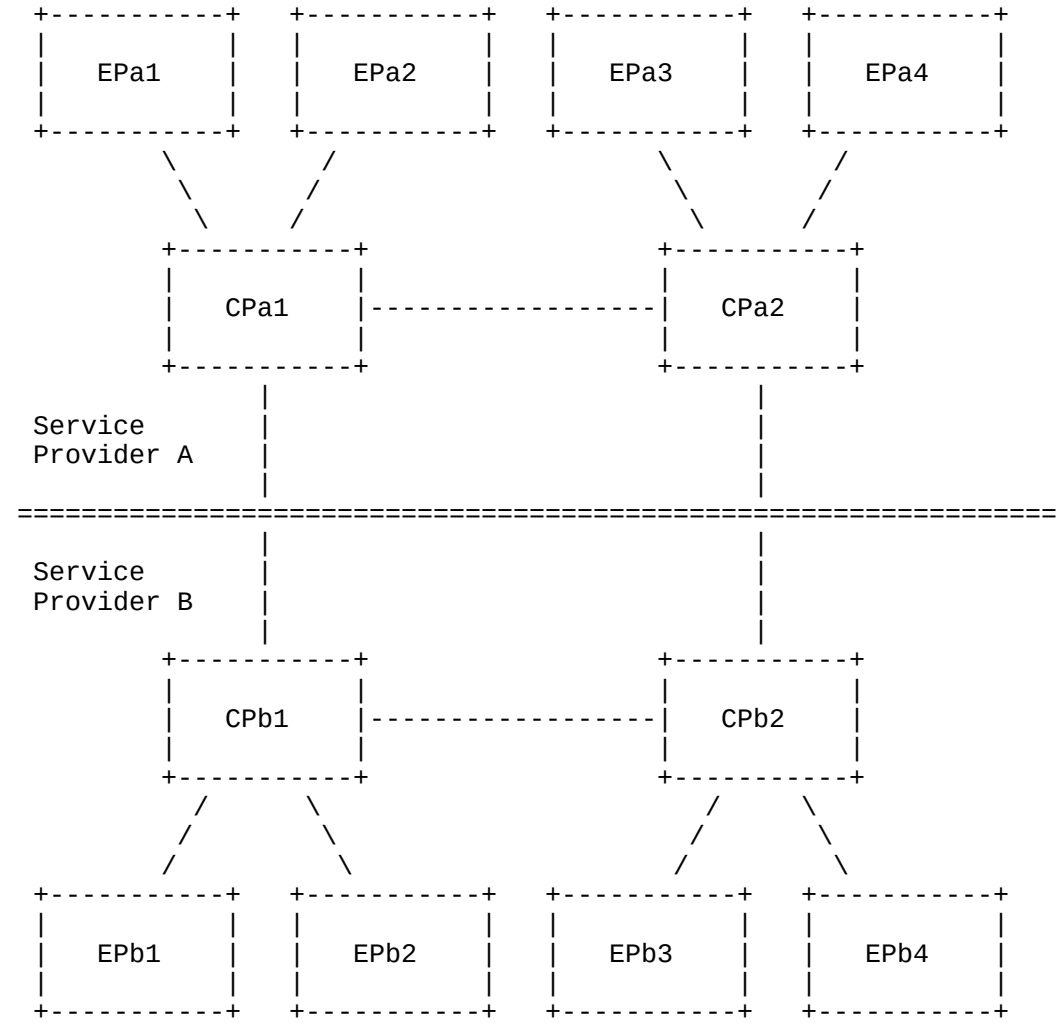

Figure 1: Example Network Scenario with SIP Load Control Event Notification

The load filter distribution based on the SIP load control event package is illustrated with an example architecture shown in Figure 1. 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 Proxies (CPs) and four SIP Edge Proxies (EPs). The CPs and EPs of Service Provider A are denoted as CPa1 to CPa2 and EPa1 to EPa4; the CPs and EPs of Service Provider B are denoted as CPb1 to CPb2 and EPb1 to EPb4.

In general, each SIP proxy server in the network is required to subscribe to the load control event package from all its outgoing signaling neighbors. 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. In the example topology of Figure 1, assuming all signaling relationship is bi-directional, each proxy will need to subscribe to all its neighbors. That is, EPa1 subscribes to CPa1; CPa1 subscribes to EPa1, EPa2, CPa2 and CPb1, so on and so forth. Notifications are always sent to all subscribing entities.

To begin load filter distribution on a network when the appropriate subscriptions among the SIP entities are ready, the initial filter contents are introduced to a SIP entity which acts as the network entry point for load filtering control. The filter is then propagated to other SIP entities throughout the network. The following shows two examples.

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 servers. This process continues until all edge proxies 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 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 dynamic 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.

The network entry point for load filtering control in the above two cases is the SIP server to be protected. In other cases, the filtering entry point 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 control can be introduced from the application server and then propogated to other SIP proxy servers in the network.

Note that this document does not define the provisioning interface between the load control policy maker and the policy entry point in the network. One of the possible solutions for the provisioning interface is the Extensible Markup Language (XML) Configuration Access Protocol (XCAP) [RFC4825].

4.4. Applicability in Different Network Environments

Load filtering control 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 decision maker. 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 the load filtering control 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. The 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 [I-D.ietf-soc-overload-design].

5. Load Control Event Package

This section defines the 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

A SUBSCRIBE request for load control policy MAY contain a body to filter the requested load control notification. For example, a subscriber may be interested in some specific types of load control information 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 message in this event package contains policy information regarding load control. 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 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

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

If the identity of the entity sending the SUBSCRIBE message is not allowed to receive overload control information, 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 the [RFC3265] specification, a notifier MUST send a NOTIFY with its current load control policy to the subscriber upon successfully accepting or refreshing a subscription. The NOTIFY request MAY include a body. If no applicable restriction is active when the subscription request is received, an empty document is attached to the NOTIFY request. 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.

This event package does not support notifications that contain deltas to previous information or partial information.

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 SIP entity 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 subscriber MUST remove all previously received load control information and process all calls without applying any restriction.

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 message itself overloading the system.

5.11. State Agents

The load control policy information 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 load control policy making.

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 mechanism are available for combining actions and transformations obtained from more than one sources.

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 document, 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 document 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 identity information 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 document defines new identity conditions that can accommodate the granularity of specific SIP identity header fields. 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 formats of 'tel' URIs: global format and local format. All phone numbers must be expressed in the global format when possible. The global format 'tel' URIs start with a "+". The rest of the phone numbers are expressed in a local format, 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 formats of the 'tel' URI make the resulting URI globally unique.

'Tel' URIs of global format 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-number format can be grouped by the value of the "phone-context" parameter.

To include the two formats 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 document, we decided on a simpler approach. There are basically two forms of grouping attribute values for both SIP URIs and 'tel' URIs: domain name or 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 forms 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 overlapping 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 format '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 'tel' URIs starting with those leading digits, and local 'tel' URIs having the same prefix in the "phone-context" parameter. These overlapping 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 overlapping is not inappropriate. Second, the use of the local format 'tel' URI in practice is expected to be rare. As a result, the chance of such overlapping 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 document 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] [I-D.ietf-soc-overload-design] for more details on rate-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 rule sets.

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">

    <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">

    <rule id="f3g44k2">
        <conditions>
            <lc:call-identity>
                <lc:sip>
                    <lc:to>
                        <many domain="pompeii.example.com"/>
                    </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] by defining new members of the <conditions> and <action> 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"/>

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

<!-- Action -->
```

```
   <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"/>
     <anyAtrribute namespace="##other" processContents="lax"/>
   </xs:element>

   <!-- 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>
   </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 document defines the SIP event 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 the time fields in the SIP load filter content definition in this document. However, the filter defined in this document is much more generic and flexible as opposed to its PSTN counterpart.

Firstly, PSTN 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 arbitrary grouped by SIP domains or any number of leading prefix of the telephone number.

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 overload 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 proxies and core proxies, 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 Reorder 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 filter content definition in this document is based on 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 filter content defined in this document is generic and is expected to be applicable not only to the load filtering control

mechanism but also to the feedback overload control mechanism in [I-D.ietf-soc-overload-control]. In particular, both of them could use specific or wildcard filter identities for load control and could share well-known load control actions. The time duration field in the filter content 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 predicable overload such as one caused by Mother's Day 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 document meeting the requirements of RFC5390

This section evaluates whether the load filtering control event package mechanism defined in this document satisfies the various SIP overload control requirements set forth by RFC5390 [RFC5390]. Not all the RFC5390 requirements are found applicable due to the scope limit 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 control 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 mechanism will throttle too much or too little.

> 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 document, while the distribution of the updated filters uses the event package mechanism of this document.

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 filter format [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 specific 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.

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 document 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 document. 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 aspect for notification control is that, in order to prevent the load control notification being used to launch denial of service attacks, 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.

Security considerations for filter enforcements vary depending on the filter contents. This document 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 document

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 document

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 SIPPING Working Group <sippping@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 SIPPING 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, Janet Gunn, Vijay Gurbani, Volker Hilt, Geoff Hunt, Timothy Moran, Eric Noel, Parthasarathi R, Keith Drage, Salvatore Loreto and other members of the SIPPING and SIP-OVERLOAD working group for helpful comments. Bruno Chatras proposed a number of text improvements, including adding the <method> condition element. Janet Gunn provided detailed text suggestions for Section 9.

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.

[RFC2648] Moats, R., "A URN Namespace for IETF Documents", RFC 2648, August 1999.

[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-00 (work in progress), November 2010.

[I-D.ietf-soc-overload-design] Hilt, V., Noel, E., Shen, C., and A. Abdelal, "Design Considerations for Session Initiation Protocol (SIP) Overload Control", draft-ietf-soc-overload-design-04 (work in progress), December 2010.

[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.

Authors' Addresses

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

Phone: +1 212 854 3109
Email: charles@cs.columbia.edu

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 &
NTT Washington DC Representative Office
1100 13th St., NW, Suite 900
Washington DC, 20005
USA

Phone: +1 202 312 1451
Email: koike.arata@lab.ntt.co.jp

SOC Working Group
Internet-Draft
Intended status: Standards Track
Expires: September 1, 2011

V. Gurbani, Ed.
Bell Laboratories, Alcatel-Lucent
V. Hilt
Bell Labs/Alcatel-Lucent
H. Schulzrinne
Columbia University
February 28, 2011

Session Initiation Protocol (SIP) Overload Control
draft-ietf-soc-overload-control-02

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 an overload control mechanism for SIP.

Table of Contents

1. Introduction

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

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

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

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

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. 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. Additional advantages of this choice are discussed in Section 15.1.1. An alternative mechanism using SIP event packages was also considered, and the characteristics of that choice are further outlined in Section 15.1.2.

This document defines four new parameters for the SIP Via header for overload control. These parameters provide a SIP mechanism for conveying overload control information between adjacent SIP entities.) These parameters are:

1. oc: This parameter serves a dual purpose; when inserted by a SIP client in a request going downstream, the parameter indicates that the SIP client supports overload control. When the downstream SIP server sends a response, the downstream SIP server will add a value to the parameter that indicates a loss rate (in percentage) by which the requests arriving at the downstream SIP server should be reduced.
2. oc-algo: This parameter serves a dual purpose: when inserted by a SIP client in a request going downstream, the parameter contains a comma-separated list of the class of overload control algorithm supported by the SIP client. When the downstream SIP server sends a response, the downstream SIP server will pick one overload control algorithm from the list and will pare the list down to include the one chosen algorithm. In this manner, the upstream SIP client and the downstream SIP server can negotiate the specific class of algorithm that is utilized for overload control.

3. oc-validity: Inserted by the SIP server sending a response upstream. This parameter contains a value that indicates the time (in millisecond resolution) that the load reduction specified by the "oc" parameter should be in effect.
4. oc-seq: Inserted by the SIP server sending a response upstream. This parameter contains a value that indicates the sequence number associated with the "oc" parameter defined above.

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

```
INVITE sips:user@example.com SIP/2.0
Via: SIP/2.0/TLS p1.example.net;
  branch=z9hG4bK2d4790.1;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=20;oc-algo="loss";oc-validity=500;
  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.

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 sends back further overload control parameters towards P1 requesting that P1 reduce the incoming traffic by 20% for 500ms. P2 updates the "oc" parameter to add a value and inserts the remaining two parameters, "oc-validity" and "oc-seq".

4. Via Header Parameters for Overload Control

4.1. The oc paramater

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 this server supports overload control.

When inserted into a request by a SIP client to indicate support for overload control, there MUST NOT be a value associated with the parameter.

The downstream server MUST add a value to the "oc" parameter in the response going upstream; this value indicates a loss rate (in percentage) by which the requests arriving at the downstream server should be reduced.

When adding a value to the "oc" parameter, the downstream server MUST restrain that value to a number between 0 and 100. This value describes the percentage by which the traffic (SIP requests) destined to the SIP server should be reduced. The default value for this parameter is 0.

When a SIP client receives a response with the value in the "oc" parameter filled in, it SHOULD reduce, in terms of a percentage, the number of requests going downstream to the SIP server from which it received the response (see Section 11 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 MAY add an "oc-algo" parameter to the topmost Via header it inserts into the SIP request. This parameter contains a comma-separated list of a class of overload control algorithms. Currently, three classes of overload control algorithms are known: loss-based, rate-based, and window-based. This document supports overload control through a loss-based mechanism, therefore the single mandatory to implement class of overload control algorithm is loss-based. All implementations that support overload control MUST implement a loss-based overload control mechanism.

If a SIP client only supports the loss-based overload control mechanism, then the "oc-algo" parameter can be omitted. When a SIP server receives a request without an "oc-algo" parameter, it MUST NOT add the parameter in the response going upstream as the absence of the parameter in the request implied that the upstream SIP client only supported a loss-based overload control mechanism.

If a SIP client supports multiple class of overload control algorithms, then it will insert a comma-separated list in the "oc-algo" parameter value. Each element in the comma-separated list corresponds to the class of overload control algorithms supported by the SIP client. Currently, three classes of overload control

algorithms are known: loss-based, rate-based, and window-based. When a downstream SIP server receives a request with a choice of overload control algorithms specified in the "oc-algo" parameter value, 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.

It is RECOMMENDED that once an upstream SIP client and a downstream 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. That is, the adjacent peers MUST NOT renegotiate the overload control algorithm class per transaction, or per request-response message exchange. 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.

Exigent realities of deployments of SIP clients and servers necessitate that the overload control algorithm be renegotiated upon a system reboot or a software upgrade, however, frequent renegotiations of the overload control algorithm MUST be avoided. Renegotiation, when desired, is simply accomplished by the SIP client sending a fresh "oc-algo" parameter in a request going downstream. The downstream server, as before, 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 and stays in effect until the next renegotiation.

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).

The "oc-validity" parameter can only be present in a Via header in conjunction with an "oc" parameter.

4.4. The oc-seq parameter

This parameter is inserted by the SIP server.

This parameter contains a value that indicates the sequence number associated with the "oc" parameter. Some implementations may be

capable of updating the overload control information before the validity period specified by the "oc-validity" parameter expires. Such implementations MUST have an increasing value in the "oc-seq" parameter for each response sent to the upstream SIP client. This is to allow the upstream SIP client to properly collate out-of-order responses.

5. Creating and updating the overload control parameters

A SIP server can provide overload control feedback to its upstream neighbors by providing a value for the "oc" parameter to the topmost Via header field of a SIP response. The topmost Via header is determined after the SIP server has removed its own Via header; i.e., it is the Via header that was generated by the upstream neighbor.

Since the topmost Via header of a response will be removed by an upstream neighbor 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 [I-D.ietf-soc-overload-design]) 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 12 for special consideration on transporting overload control parameters in a 100-Trying response). A SIP server MAY update the "oc" parameter in all responses it is sending. A SIP server MUST update the "oc" parameter to responses when the transmission of overload control feedback is required by the overload control algorithm to limit the traffic received by the server. I.e., a SIP server MUST update the "oc" parameter when the overload control algorithm sets the value of an "oc" parameter to a value different than the default value.

A SIP server that has updated the "oc" parameter to Via header SHOULD also add a "oc-validity" parameter to the same Via header. The "oc-validity" parameter defines the time in milliseconds during which the content (i.e., the overload control feedback) of the "oc" parameter is valid. The default value of the "oc-validity" parameter is 500 (millisecond). A SIP server SHOULD use a shorter "oc-validity" time if its overload status varies quickly and MAY use a longer "oc-validity" time if this status is more stable. 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. Furthermore, when a SIP server receives a request with the topmost Via header containing only an "oc-validity" parameter without the accompanying

"oc" parameter. it MUST ignore the "oc-validity" parameter.

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 use 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.

6. Determining the ’oc’ Parameter Value

The value of the "oc" parameter is determined by an overload control algorithm (see [I-D.ietf-soc-overload-design]). This specification

does not mandate the use of a specific overload control algorithm. However, the output of an overload control algorithm MUST be compliant to the semantics of this Via header parameter.

The "oc" parameter value specifies the percentage by which the load forwarded to this SIP server should be reduced. Possible values range from 0 (the traffic forwarded is reduced by 0%, i.e., all traffic is forwarded) to 100 (the traffic forwarded is reduced by 100%, i.e., no traffic forwarded). The default value of this parameter is 0.

7. Processing the Overload Control Parameters

A SIP client compliant to this specification 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.

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.

8. Using the Overload Control Parameter Values

A SIP client compliant to this specification 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 a 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, and 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 and rate of SIP requests as the default algorithm (to be determined), which treats all requests as equal.

In the absence of a different local policy, the SIP client SHOULD use the following default algorithm to determine if it can forward the request downstream (TODO: Need to devise an algorithm. The original simple algorithm based on random number generation does not suffice for all cases.)

9. Forwarding the overload control parameters

A SIP client MAY forward the content of an "oc" parameter it has received from a downstream neighbor on to its upstream neighbor. However, forwarding the content of the "oc" parameter is generally NOT RECOMMENDED and should only be performed if permitted by the configuration of SIP servers. For example, a SIP server that only relays messages between exactly two SIP servers may forward an "oc" parameter. The "oc" parameter is forwarded by copying it from the Via in which it was received into the next Via header (i.e., the Via header that will be on top after processing the response). If an "oc-validity" parameter is present, MUST be copied along with the "oc" parameter.

10. Self-Limiting

In some cases, a SIP client may not receive a response from a downstream 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 occasionally forward a single request to probe if the downstream server is alive. 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.

OPEN ISSUE: If a downstream neighbor does not respond to a request at all, the upstream SIP client will stop sending requests to the downstream neighbor. The upstream SIP client will periodically forward a single request to probe the health of its downstream neighbor. It has been suggested --- see http://www.ietf.org/mail-archive/web/sip-overload/current/msg00229.html --- that we have a notification mechanism in place for the downstream neighbor to signal to the upstream SIP client that it is ready to receive requests. This notification scheme has advantages, but comes with obvious disadvantages as well. Need some more discussion around this.

11. 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.

11.1. 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.

11.2. 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 a fraction of requests from upstream neighbors that do not support overload control.

12. 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.

13. 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 11.1 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].

14. 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 0-100]
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
```

15. 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

[I-D.ietf-soc-overload-design].

15.1. 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.

15.1.1. 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.

15.1.2. 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:

- 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.
- 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.
- 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.
- 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.
- As overload feedback is sent to all senders in separate messages, this mechanism is not suitable when frequent overload control feedback is needed.
- A SIP server can limit the set of senders that can receive overload control information by authenticating subscriptions to this event package.
- This approach requires each proxy to implement user agent functionality (UAS and UAC) to manage the subscriptions.

15.2. 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 [I-D.ietf-soc-overload-design]) 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 use 5xx responses towards upstream neighbors that do not support overload control. The server should reject the same amount of requests with 5xx responses that would be otherwise be rejected/redirected by the upstream neighbor if it would support overload control. If the load condition on the server does not permit the creation of 5xx responses, the server should drop all requests from servers that do not support overload control.

16. 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.

17. 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.]

18. References

18.1. 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.

18.2. 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-00 (work in progress), January 2011.

[I-D.ietf-soc-overload-design] Hilt, V., Noel, E., Shen, C., and A. Abdelal, "Design Considerations for Session Initiation Protocol (SIP) Overload Control", draft-ietf-soc-overload-design-04 (work in progress), December 2010.

[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.

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, and Paul Kyzivat for their contributions to this specification.

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 [I-D.ietf-soc-overload-design]) 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 conformant to this specification will attempt to reduce traffic destined towards the non-responsive server as outlined in Section 10.

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 16 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 8 and Section 11.

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 11.

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 10 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 10 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 16 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 8.

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 11.

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 [I-D.ietf-soc-overload-design] 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 Labs/Alcatel-Lucent
791 Holmdel-Keyport Rd
Holmdel, NJ 07733
USA

Email: volkerh@bell-labs.com

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

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

IETF SOC Working Group
Internet-Draft
Intended status: Standards Track
Expires: June 4, 2011

C. Shen
H. Schulzrinne
Columbia U.
A. Koike
NTT

December 1, 2010

A Mechanism for Session Initiation Protocol (SIP) Avalanche Restart Overload Control
draft-shen-soc-avalanche-restart-overload-01

Abstract

When a large number of clients register with a SIP registrar server at approximately the same time, the server may become overloaded. Near-simultaneous floods of SIP SUBSCRIBE and PUBLISH requests may have similar effects. Such request avalanches can occur, for example, after a power failure and recovery in a metropolitan area. This document describes how to avoid such overload situations. Under this mechanism, a server estimates an avalanche restart backoff interval during its normal operation and conveys this interval to its clients through a new Restart-Timer header in normal response messages. Once an avalanche restart actually occurs, the clients perform backoff based on the previously received Restart-Timer header value before sending out the first request attempt. Thus, the mechanism spreads all the initial client requests and prevents them from overloading the server.

Table of Contents

1. Introduction

A Session Initiation Protocol (SIP) [RFC3261] server can be overloaded for a number of different reasons. One of them is avalanche restart, which is described in [RFC5390] as follows:

> Avalanche Restart: One of the most troubling sources of overload is avalanche restart. This happens when a large number of clients all simultaneously attempt to connect to the network with a SIP registration. Avalanche restart can be caused by several events. One is the "Manhattan Reboots" scenario, where there is a power failure in a large metropolitan area, such as Manhattan. When power is restored, all of the SIP phones, whether in PCs or standalone devices, simultaneously power on and begin booting. They will all then connect to the network and register, causing a flood of SIP registration messages. Another cause of avalanche restart is failure of a large network connection, for example, the access router for an enterprise. When it fails, SIP clients will detect the failure rapidly using the mechanisms in [RFC5626]. When connectivity is restored, this is detected, and clients re-registration, all within a short time period. Another source of avalanche restart is failure of a proxy server. If clients had all connected to the server with TCP, its failure will be detected, followed by re-connection and re-registration to another server. Note that [RFC5626] does provide some remedies to this case.

The SIP server avalanche restart overload problem is caused by the synchronized, simultaneous initial registration attempts after a failure recovery. If the first round of registration attempts from all clients cause server overload, most of those registrations will fail. Those clients will then by default all retry after the same amount of time, causing repeated server avalanche restart overload. [RFC5626] describes how to alleviate this situation: if the initial registration attempt after the boot fails, the clients wait for a randomized backoff time before retrying. This mechanism reduces the possibility of repeated avalanche restart. However, since all clients still send registration immediately after boot, it does not prevent the initial avalanche restart overload.

A key method to prevent avalanche restart server overload is to have clients backoff before their first registration attempt. The backoff intervals of each client must be carefully selected so that their registration attempts are spaced sufficiently far apart not to overload the server, and they are also not too conservative which may cause unnecessary client registration delays. An individual client, without knowing the state information of all other peer clients and the registrar server, is inherently incapable of choosing such an

appropriate backoff interval.

This document specifies a solution to the avalanche restart overload problem by allowing the registrar server to instruct the clients how long they should wait before the initial registration upon a restart event. Under this mechanism, the server estimates an avalanche restart backoff interval during its normal operation. This interval is the minimum period of time that the server needs to serve all the expected registration requests after an avalanche restart, assuming all the registration requests are properly spaced. In order for the server to convey this interval to its clients, this document defines a new SIP Restart-Timer header. The registrar server places the avalanche restart backoff interval into the Restart-Timer header and inserts it into regular responses to client registration requests. When an avalanche restart actually happens, each client waits a randomly-chosen period between 0 and the avalanche restart backoff interval.

This document also defines an algorithm to determine the avalanche restart backoff interval based on the server's processing capability and the number of clients it is serving. The effectiveness of this algorithm depends on the assumption that both the server processing capability and the number of clients the server serves remain similar before and after the avalanche restart. This assumption holds true in most cases when the registrar server before and after the avalanche restart is the same one, e.g., in the "Manhattan Reboots" scenario, and the loss and recovery of large network connection scenario.

The method defined in this document is intended to be used for real avalanche restart situations, rather than just any local reboot or connection recovery. Therefore, the device employing this mechanism SHOULD try to estimate the nature of the restart incidents whenever possible. Some devices, especially mobile terminals, may also have lower layer (e.g., Physical or Data Link layer) backoff or blocking mechanisms during avalanche restart or network congestion. In those cases, operators may also disable this application layer avalanche restart protection method. Such cross-layer optimizations, however, are out of scope of this document.

Throughout this document our description assumes the typical scenario where the clients send out REGISTER messages as the first message type after a restart and cause avalanche restart overload on the registrar server. It should be noted that similar procedures are applicable to scenarios where the first message after reboot is of a different type and causes overload. For example, when SIP based configuration mechanism is followed, the clients may first send out SUBSCRIBE messages to a SIP configuration server to get the registrar

address after a reboot. In that case, the server that determines the restart backoff interval needs to be the corresponding SIP configuration server, and the backoff mechanism could then be similarly applied to the sending of the initial SUBSCRIBE messages.

This document complements other SIP server overload control specifications which address different aspects of the SIP server overload space, such as [I-D.hilt-sipping-overload], [I-D.shen-sipping-load-control-event-package], and [I-D.ietf-sipping-overload-design].

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 [RFC2119].

3. Restart-Timer Header for Registration Responses

This document defines the SIP Restart-Timer header for registration responses. The value of the Restart-Timer header, in seconds, denotes the avalanche restart backoff interval, which is the minimum time the server needs to successfully service all likely client registration requests under an avalanche restart situation, assuming all requests are spaced evenly in time.

3.1. Generating the Restart-Timer Header

A SIP registrar server inserts a Restart-Timer header containing its most up-to-date avalanche restart backoff interval value in the responses to registration requests.

Example:

```
SIP/2.0 200 OK
Restart-Timer: 300
```

3.2. Determining the Restart-Timer Header Value

A registrar server computes and updates the Restart-Timer header values and conveys them to the clients during its normal operations. Once an avalanche restart actually happens, the most recent Restart-Timer header value that the clients have received from the registrar server are used. A registrar server MAY use the following algorithm for determining the appropriate Restart-Timer header value.

During the normal operation period, the SIP registrar server maintains the current count of all its registrants, e.g., assuming the number of registered clients is R. The SIP registrar server also estimates its processing capacity, e.g., assuming it is C requests per second. The Restart-Timer value can be set to (R/C)*(1+k), where k is a small coefficient that provides a capacity redundancy. A recommended value of k is 0.1.

It should be noted that change of either R or C adjusts the server computed Restart-Timer value. The value C is usually stable on the same server and with the same registration request pattern. The value R may change over time. The server SHOULD recompute the Restart-Timer value whenever there is a change in either R or C, unless it is considered too expensive to do so, which is normally not the case. Since the updated Restart-Timer value is only pushed to the clients when the client sends in a registration, there might be a short period where the server side updated Restart-Timer value and some client side stored Restart-Timer values are not synchronized. However, considering that the changing pace of R is slow, and the time scale between the possible happenings of avalanche restarts (e.g., months) is usually much larger than the interval between typical registration renewals (e.g., hours), these short periods of discrepancies are not a concern. Therefore, in general this approach provides a sufficiently accurate characterization of the system status. More importantly, the values of R and C are expected to remain constant for the same server before and after typical avalanche restart events, e.g., a power failure and recovery.

3.3. Processing the Restart-Timer Header

Before receiving the very first registration response from a new registrar server, the client restart backoff value for that registrar server is zero, i.e., the restart backoff mechanism is disabled.

Upon receiving a response to the registration request containing the Restart-Timer header, a SIP client that supports this specification MUST check if there is an existing Restart-Timer header value for this registrar stored in the system. If not, it stores the newly received Restart-Timer header value. Otherwise, it compares the new value with the existing one and updates it if they differ. The value of Restart-Timer header SHOULD be stored together with the corresponding identity of the server, e.g., the DNS name of the registrar server. There is no separate validity period parameter for Restart-Timer. The validity duration of the Restart-Timer header is the same as that of the corresponding registration operation.

3.4. Using the Restart-Timer Header

At the client side, avalanche restart backoff is disabled by default, unless the client that supports this specification has received a positive Restart-Timer header value from the corresponding registrar server.

A SIP client always keeps the most updated Restart-Timer header value. When this value is positive and if the client detects that it is about to perform the first registration with the same registrar server after a power-off reboot or a connection-loss recovery, the client SHOULD generate a uniformly distributed random interval between 0 and the current Restart-Timer value, and wait until the end of that interval to send the registration request. However, the client side backoff MAY be manually disabled by a human operator when necessary, e.g., when the operator is expecting an urgent call, or when the power-off or connection-loss event is known as a local incident rather than a global event.

It should be noted that the power-off reboot case requires that the state information about the Restart-Timer value and the registrar server identity be stored in a memory space that could survive power restart.

4. Syntax

The new Restart-Timer header adds the following lines to the existing SIP header definition.

```
message-header = Restart-Timer

Restart-Timer = "Restart-Timer" HCOLON delta-seconds
```

5. Backward Compatibility

If a registrar server supports this specification, but not all of its clients are upgraded, then those non-compliant clients will ignore the Restart-Timer header and not perform backoff. Although it appears that this might give the non-compliant clients an unfair advantage over those clients that do perform backoff, since the non-compliant clients will send synchronized registration attempts to the registrar server and can cause server overload, they will be penalized by registration failures. Depending on the number of the non-compliant clients vs. compliant clients, if the registrar server can still process requests when it does not receive registration storms from all the non-compliant clients, the requests from the

compliant clients which spread apart, are more likely to succeed.

6. Security Considerations

The Restart-Timer header can be used by an attacker to launch a possible denial-of-service attack on SIP clients if the attacker can insert an infinitely large Restart-Timer value in the response sent to the clients. In those situations, the client may generate a very large backoff time before it attempts to send a registration request, and therefore the client is subject to denial-of-service attack. However, this kind of attack is only applicable after a power-cycle reboot or failure and recovery of a large network connection, which is rare. Furthermore, if the attacker can modify the registration request or response, that attacker can very easily prevent registration in any number of ways, so the Restart-Timer header does not introduce new types of attacks. One method to prevent the registration request and response from being altered by attackers is to use TLS between the client and the registrar server.

7. IANA Considerations

[TBD]

8. Acknowledgements

The authors would like to thank Janet P Gunn, Parthasarathi R and other members of the SIPPING and SIP-OVERLOAD working group for useful comments.

9. References

9.1. 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.

9.2. Informative References

[I-D.hilt-sipping-overload]

Hilt, V. and H. Schulzrinne, "Session Initiation Protocol (SIP) Overload Control", draft-hilt-sipping-overload-08 (work in progress), April 2010.

[I-D.ietf-sipping-overload-design]
Hilt, V., Noel, E., Shen, C., and A. Abdelal, "Design Considerations for Session Initiation Protocol (SIP) Overload Control", draft-ietf-sipping-overload-design-02 (work in progress), July 2009.

[I-D.shen-sipping-load-control-event-package]
Shen, C., Schulzrinne, H., and A. Koike, "A Session Initiation Protocol (SIP) Load Control Event Package", draft-shen-sipping-load-control-event-package-04 (work in progress), April 2010.

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

[RFC5626] Jennings, C., Mahy, R., and F. Audet, "Managing Client-Initiated Connections in the Session Initiation Protocol (SIP)", RFC 5626, October 2009.

Authors' Addresses

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

Phone: +1 212 854 3109
Email: charles@cs.columbia.edu

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

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

Arata Koike
NTT Service Integration Labs &
NTT Washington DC Representative Office
1100 13th St., NW, Suite 900
Washington DC, 20005
USA

Phone: +1 202 312 1451
Email: koike.arata@lab.ntt.co.jp