IETF SOC Working Group
Internet-Draft
Intended status: Standards Track
Expires: August 15, 2014

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

February 11, 2014

# A Mechanism for Session Initiation Protocol (SIP) Avalanche Restart Overload Control
## draft-shen-soc-avalanche-restart-overload-07

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.ietf-soc-overload-control], [I-D.ietf-soc-load-control-event-package], and [RFC6357].

## 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 SOC 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.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-13 (work in progress), December 2013.

[I-D.ietf-soc-overload-control] Gurbani, V., Hilt, V., and H. Schulzrinne, "Session Initiation Protocol (SIP) Overload Control", draft-ietf-soc-overload-control-14 (work in progress), December 2013.

[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.

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

Authors' Addresses

Charles Shen
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 Network Technology Labs &
3-9-11 Midori-cho Musashino-shi
Tokyo  180-8585
Japan

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