Internet Engineering Task Force (IETF)
Request for Comments: 7415
Category: Standards Track
ISSN: 2070-1721

E. Noel
AT&T Labs
P. Williams
BT Innovate & Design

February 2015

Session Initiation Protocol (SIP) Rate Control

Abstract

The prevalent use of the Session Initiation Protocol (SIP) in Next Generation Networks necessitates that SIP networks provide adequate control mechanisms to maintain transaction throughput by preventing congestion collapse during traffic overloads. A loss-based solution to remedy known vulnerabilities of the SIP 503 (Service Unavailable) overload control mechanism has already been proposed. Using the same signaling, this document proposes a rate-based control scheme to complement the loss-based control scheme.

Status of This Memo

This is an Internet Standards Track document.

This document is a product of the Internet Engineering Task Force (IETF). It represents the consensus of the IETF community. It has received public review and has been approved for publication by the Internet Engineering Steering Group (IESG). Further information on Internet Standards is available in Section 2 of RFC 5741.

Information about the current status of this document, any errata, and how to provide feedback on it may be obtained at http://www.rfc-editor.org/info/rfc7415.

Table of Contents

1. Introduction

The use of SIP [RFC3261] in large-scale Next Generation Networks requires that SIP-based networks provide adequate control mechanisms for handling traffic growth. In particular, SIP networks must be able to handle traffic overloads gracefully, maintaining transaction throughput by preventing congestion collapse.

A promising SIP-based overload control solution has been proposed in [RFC7339]. That solution provides a communication scheme for overload control algorithms. It also includes a default loss-based overload control algorithm that makes it possible for a set of clients to limit offered load towards an overloaded server. However, such a loss control algorithm is sensitive to variations in load so that any increase in load would be directly reflected by the clients in the offered load presented to the overloaded servers. More importantly, a loss-based control scheme cannot guarantee an upper bound on the load from the clients towards an overloaded server and requires frequent updates that may have implications for stability.

In accordance with the framework defined in [RFC7339], this document proposes an alternate overload control scheme: the rate-based overload control scheme. The rate-based control algorithm guarantees an upper bound on the rate, constant between server updates, of

requests sent by clients towards an overloaded server. The trade-off is in terms of algorithmic complexity, since the overloaded server is more likely to use a different target (maximum rate) for each client than the loss-based approach.

The proposed rate-based overload control algorithm mitigates congestion in SIP networks while adhering to the overload signaling scheme in [RFC7339] and presenting a rate-based control scheme as an optional alternative to the default loss-based control scheme in [RFC7339].

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].

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

3. Rate-Based Algorithm Scheme

3.1. Overview

The server is the one protected by the overload control algorithm defined here, and the client is the one that throttles traffic towards the server.

Following the procedures defined in [RFC7339], the server and clients signal one another support for rate-based overload control.

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

When in overload, the server uses the "oc" parameter in the Via header field [RFC7339] of SIP responses in order to inform clients of its overload state and of the target maximum SIP request rate for that client.

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

3.2. Via Header Field Parameters for Overload Control

Four Via header parameters are defined in [RFC7339] and are summarized below:

- oc: Used by clients in SIP requests to indicate support for overload control per [RFC7339] and by servers to indicate the load reduction amount in the loss-based algorithm and the maximum rate, in messages per second, for the rate-based algorithm described here.

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

- oc-validity: Used by servers in SIP responses to indicate an interval of time (in milliseconds) that the load reduction should be in effect. A value of 0 is reserved for the server to stop overload control. A non-zero value is required in all other cases.

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

Consult Section 4 for an illustration of the usage of the "oc" parameter in the Via header field.

3.3. Client and Server Rate-Based Control Algorithm Selection

Per [RFC7339], new clients indicate supported overload control algorithms to servers by inserting "oc" and "oc-algo", with the names of the supported algorithms, in the Via header field of SIP requests destined to servers. The inclusion by the client of the token "rate" indicates that the client supports a rate-based algorithm. Conversely, servers notify clients of the selected overload control algorithm through the "oc-algo" parameter in the Via header field of SIP responses to clients. The inclusion by the server of the token "rate" in the "oc-algo" parameter indicates that the rate-based algorithm has been selected by the server.

Support of rate-based control MUST be indicated by clients including the token "rate" in the "oc-algo" list. Selection of rate-based control MUST be indicated by servers by setting "oc-algo" to the token "rate".

3.4. Server Operation

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

However, the server MUST periodically evaluate its overload state and estimate a target SIP request rate beyond which it would become overloaded. The server must determine how it will allocate the target SIP request rate among its client. The server may set the same rate for every client or may set different rates for different clients.

The maximum rate determined by the server for a client applies to the entire stream of SIP requests, even though throttling may only affect a particular subset of the requests, since as per [RFC7339] and REQ 13 of [RFC5390], request prioritization is a client’s responsibility.

When setting the maximum rate for a particular client, the server may need to take into account the workload (e.g., CPU load per request) of the distribution of message types from that client. Furthermore, because the client may prioritize the specific types of messages it sends while under overload restriction, this distribution of message types may be different from the message distribution for that client under non-overload conditions (e.g., it could have either higher or lower CPU load).

Note that the "oc" parameter for the rate-based algorithm is an upper bound (in messages per second) on the traffic sent by the client to the server. The client may send traffic at a rate significantly lower than the upper bound for a variety of reasons.

In other words, when multiple clients are being controlled by an overloaded server, at any given time, some clients may receive requests at a rate below their target (maximum) SIP request rate while others above that target rate. But the resulting request rate presented to the overloaded server will converge towards the target SIP request rate.

Upon detection of overload and the determination to invoke overload controls, the server MUST follow the specifications in [RFC7339] to notify its clients of the allocated target SIP request rate and to notify them that rate-based control is in effect.

The server MUST use the "oc" parameter defined in [RFC7339] to send a target SIP request rate to each of its clients.

When a client supports the default loss-based algorithm and not the rate-based algorithm, the client would be handled in the same way as in Section 5.10.2 of [RFC7339].

3.5. Client Operation

3.5.1. Default Algorithm

In determining whether or not to transmit a specific message, the client may use any algorithm that limits the message rate to the "oc" parameter in units of messages per second. For ease of discussion, we define T = 1/["oc" parameter] as the target inter-SIP request interval. The algorithm may be strictly deterministic, or it may be probabilistic. It may, or may not, have a tolerance factor to allow for short bursts, as long as the long-term rate remains below 1/T.

The algorithm may have provisions for prioritizing traffic in accordance with REQ 13 of [RFC5390].

If the algorithm requires other parameters (in addition to "T", which is 1/["oc" parameter]), they may be set autonomously by the client, or they may be negotiated between client and server independently of the SIP-based overload control solution.

In either case, the coordination is out of the scope of this document. The default algorithms presented here (one with and one without provisions for prioritizing traffic) are only examples.

To throttle new SIP requests at the rate specified by the "oc" parameter sent by the server to its clients, the client MAY use the proposed default algorithm for rate-based control or any other equivalent algorithm that forward messages in conformance with the upper bound of 1/T messages per second.

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

Conceptually, the leaky bucket algorithm can be viewed as a finite capacity bucket whose real-valued content drains out at a continuous rate of 1 unit of content per time unit and whose content increases by the increment T for each forwarded SIP request. T is computed as the inverse of the rate specified by the "oc" parameter, namely T = 1 / ["oc" parameter].

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

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

Note that the capacity of the bucket (the upper bound of the counter) is (T + TAU).

The tolerance parameter TAU determines how close the long-term admitted rate is to an ideal control that would admit all SIP requests for arrival rates less than 1/T and then admit SIP requests precisely at the rate of 1/T for arrival rates above 1/T. In particular, at mean arrival rates close to 1/T, it determines the tolerance to deviation of the inter-arrival time from T (the larger TAU, the more tolerance to deviations from the inter-departure interval T).

This deviation from the inter-departure interval influences the admitted rate burstiness or the number of consecutive SIP requests forwarded to the server (burst size proportional to TAU over the difference between 1/T and the arrival rate).

In situations where clients are configured with some knowledge about the server (e.g., operator pre-provisioning), it can be beneficial to choose a value of TAU based on how many clients will be sending requests to the server.

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

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

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

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

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

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

TAU can assume any positive real number value and is not necessarily bounded by T.

TAU=4*T is a reasonable compromise between burst size and throttled rate adaptation at low offered rates.

Note that specification of a value for TAU and any communication or coordination between servers are beyond the scope of this document.

A reference algorithm is shown below.

No priority case:

```
// T: inter-transmission interval, set to 1 / ["oc" parameter]
// TAU: tolerance parameter
// ta: arrival time of the most recent arrival received by the
//     client
// LCT: arrival time of last SIP request that was sent to the server
//      (initialized to the first arrival time)
// X: current value of the leaky bucket counter (initialized to
//    TAU0)

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

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

3.5.2. Priority Treatment

As with the loss-based algorithm in [RFC7339], a client implementing the rate-based algorithm also prioritizes messages into two or more categories of requests, for example, requests that are candidates for reduction and requests that are not subject to reduction (except under extenuating circumstances when there aren't any messages in the first category that can be reduced).

Accordingly, the proposed leaky bucket implementation is modified to support priority using two thresholds for SIP requests that are candidates for reduction. With two priorities, the proposed leaky bucket requires two thresholds: TAU1 and TAU2 (where TAU1 < TAU2):

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

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

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

TAU1 and TAU2 can assume any positive real number value and are not necessarily bounded by T.

Reasonable values for TAU0, TAU1, and TAU2 are:

- TAU0 = 0,
- TAU1 = 1/2 * TAU2, and
- TAU2 = 10 * T.

Note that specification of a value for TAU1 and TAU2 and any communication or coordination between servers are beyond the scope of this document.

A reference algorithm is shown below.

Priority case:

```
// T: inter-transmission interval, set to 1 / ["oc" parameter]
// TAU1: tolerance parameter of no-priority SIP requests
// TAU2: tolerance parameter of priority SIP requests
// ta: arrival time of the most recent arrival received by the
//     client
// LCT: arrival time of last SIP request that was sent to the server
//      (initialized to the first arrival time)
// X: current value of the leaky bucket counter (initialized to
//    TAU0)

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

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

3.5.3. Optional Enhancement: Avoidance of Resonance

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

If the network topology is such that resonance can occur, then a simple way to avoid resonance is to randomize the bucket occupancy at two appropriate points -- at the activation of control and whenever the bucket empties -- as described below.

After updating the value of the leaky bucket to X', generate a value u as follows:

if X' > 0, then u = 0

else if X' <= 0, then let u be set to a random value uniformly distributed between -1/2 and +1/2

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

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

This method has the following characteristics:

- If TAU0 is chosen to be equal to TAU and all sources activate control at the same time due to an extremely high request rate, then the time until the first request admitted by each client would be uniformly distributed over [0,T].

- The maximum occupancy is TAU + (3/2)T, rather than TAU + T without randomization.

- For the special case of 'classic gapping' where TAU=0, then the minimum time between admissions is uniformly distributed over [T/2, 3T/2], and the mean time between admissions is the same, i.e., T+1/R where R is the request arrival rate.

- As high load randomization rarely occurs, there is no loss of precision of the admitted rate, even though the randomized 'phasing' of the buckets remains.

4. Example

The example in this section adapts the example in Section 6 of [RFC7339], where client P1 sends requests to a downstream server P2:

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

Via: SIP/2.0/TLS p1.example.net;

 branch=z9hG4bK2d4790.1;received=192.0.2.111;

 oc;oc-algo="loss,rate"

...

SIP/2.0 100 Trying

Via: SIP/2.0/TLS p1.example.net;

 branch=z9hG4bK2d4790.1;received=192.0.2.111;

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

 oc-seq=1282321615.781

...
```

The first message above is sent by P1 to P2. This message is a SIP request; because P1 supports overload control, it inserts the "oc" parameter in the topmost Via header field that it created. P1 supports two overload control algorithms: loss and rate.

The second message, a SIP response, shows the topmost Via header field amended by P2 according to this specification and sent to P1. Because P2 also supports overload control, it chooses the rate-based scheme and sends that back to P1 in the "oc-algo" parameter. It uses oc-validity=0 to indicate no overload control. In this example, "oc=0", but "oc" could be any value as "oc" is ignored when "oc-validity=0".

At some later time, P2 starts to experience overload. It sends the following SIP message indicating P1 should send SIP requests at a rate no greater than or equal to 150 SIP requests per second and for a duration of 1,000 milliseconds.

```
SIP/2.0 180 Ringing

Via: SIP/2.0/TLS p1.example.net;

 branch=z9hG4bK2d4790.1;received=192.0.2.111;

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

 oc-seq=1282321615.782

 ...
```

5. Syntax

This specification extends the existing definition of the Via header field parameters of [RFC7339] as follows:

```
algo-list =/ "rate"
```

6. Security Considerations

Aside from the resonance concerns discussed in Section 3.5.3, this mechanism does not introduce any security concerns beyond the general overload control security issues discussed in [RFC7339]. Methods to mitigate the risk of resonance are discussed in Section 3.5.3.

7. IANA Considerations

IANA has registered the "oc-algo" parameter of the Via header field in the "Header Field Parameters and Parameter Values" subregistry of the "Session Initiation Protocol (SIP) Parameters" registry. The entry appears as follows:

Header Field	Parameter Name	Predefined Values	References
Via	oc-algo	Yes	[RFC7339] [RFC7415]

8. References

8.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997, <http://www.rfc-editor.org/info/rfc2119>.

[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, <http://www.rfc-editor.org/info/rfc3261>.

[RFC5390] Rosenberg, J., "Requirements for Management of Overload in the Session Initiation Protocol", RFC 5390, December 2008, <http://www.rfc-editor.org/info/rfc5390>.

[RFC7339] Gurbani, V., Ed., Hilt, V., and H. Schulzrinne, "Session Initiation Protocol (SIP) Overload Control", RFC 7339, September 2014, <http://www.rfc-editor.org/info/rfc7339>.

8.2. Informative References

[ITU-T-I.371] ITU-T, "Traffic control and congestion control in B-ISDN", ITU-T Recommendation I.371, March 2004.

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

Acknowledgments

Many thanks to the following individuals for comments and feedback on this document: Richard Barnes, Keith Drage, Vijay Gurbany, Volker Hilt, Christer Holmberg, Winston Hong, Peter Yee, and James Yu.

Contributors

Significant contributions to this document were made by Janet Gunn.

Authors' Addresses

Eric Noel
AT&T Labs
200 S Laurel Avenue
Middletown, NJ 07747
United States

EMail: eric.noel@att.com

Philip M. Williams
BT Innovate & Design
Ipswich, IP5 3RE
United Kingdom

EMail: phil.m.williams@bt.com