Network Working Group
Request for Comments: 5390
Category: Informational

J. Rosenberg
Cisco

December 2008

Requirements for Management of Overload in the Session Initiation Protocol

Status of This Memo

Copyright Notice

Abstract

Overload occurs in Session Initiation Protocol (SIP) networks when proxies and user agents have insufficient resources to complete the processing of a request. SIP provides limited support for overload handling through its 503 response code, which tells an upstream element that it is overloaded. However, numerous problems have been identified with this mechanism. This document summarizes the problems with the existing 503 mechanism, and provides some requirements for a solution.

Table of Contents

1. Introduction

Overload occurs in Session Initiation Protocol (SIP) [RFC3261] networks when proxies and user agents have insufficient resources to complete the processing of a request or a response. SIP provides limited support for overload handling through its 503 response code. This code allows a server to tell an upstream element that it is overloaded. However, numerous problems have been identified with this mechanism.

This document describes the general problem of SIP overload and reviews the current SIP mechanisms for dealing with overload. It then explains some of the problems with these mechanisms. Finally, the document provides a set of requirements for fixing these problems.

2. Causes of Overload

Overload occurs when an element, such as a SIP user agent or proxy, has insufficient resources to successfully process all of the traffic it is receiving. Resources include all of the capabilities of the element used to process a request, including CPU processing, memory, I/O, or disk resources. It can also include external resources such as a database or DNS server, in which case the CPU, processing, memory, I/O, and disk resources of those servers are effectively part of the logical element processing the request. Overload can occur for many reasons, including:

Poor Capacity Planning: SIP networks need to be designed with sufficient numbers of servers, hardware, disks, and so on, in order to meet the needs of the subscribers they are expected to serve. Capacity planning is the process of determining these needs. It is based on the number of expected subscribers and the types of flows they are expected to use. If this work is not done properly, the network may have insufficient capacity to handle predictable usages, including regular usages and predictably high ones (such as high voice calling volumes on Mother's Day).

Dependency Failures: A SIP element can become overloaded because a resource on which it is dependent has failed or become overloaded, greatly reducing the logical capacity of the element. In these cases, even minimal traffic might cause the server to go into overload. Examples of such dependency overloads include DNS servers, databases, disks, and network interfaces.

Component Failures: A SIP element can become overloaded when it is a member of a cluster of servers that each share the load of traffic, and one or more of the other members in the cluster fail. In this case, the remaining elements take over the work of the failed elements. Normally, capacity planning takes such failures into account, and servers are typically run with enough spare capacity to handle failure of another element. However, unusual failure conditions can cause many elements to fail at once. This is often the case with software failures, where a bad packet or bad database entry hits the same bug in a set of elements in a cluster.

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 REGISTER 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 [OUTBOUND]. When connectivity is restored, this is detected, and clients re-REGISTER, 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 [OUTBOUND] does provide some remedies to this case.

Flash Crowds: A flash crowd occurs when an extremely large number of users all attempt to simultaneously make a call. One example of how this can happen is a television commercial that advertises a number to call to receive a free gift. If the gift is compelling and many people see the ad, many calls can be simultaneously made to the same number. This can send the system into overload.

Denial of Service (DoS) Attacks: An attacker, wishing to disrupt service in the network, can cause a large amount of traffic to be launched at a target server. This can be done from a central source of traffic or through a distributed DoS attack. In all cases, the volume of traffic well exceeds the capacity of the server, sending the system into overload.

Unfortunately, the overload problem tends to compound itself. When a network goes into overload, this can frequently cause failures of the elements that are trying to process the traffic. This causes even more load on the remaining elements. Furthermore, during overload, the overall capacity of functional elements goes down, since much of their resources are spent just rejecting or treating load that they cannot actually process. In addition, overload tends to cause SIP messages to be delayed or lost, which causes retransmissions to be sent, further increasing the amount of work in the network. This compounding factor can produce substantial multipliers on the load in the system. Indeed, in the case of UDP, with as many as seven retransmits of an INVITE request prior to timeout, overload can multiply the already-heavy message volume by as much as seven!

3. 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].

4. Current SIP Mechanisms

SIP provides very basic support for overload. It defines the 503 response code, which is sent by an element that is overloaded. RFC 3261 defines it thus:

> The server is temporarily unable to process the request due to a temporary overloading or maintenance of the server. The server MAY indicate when the client should retry the request in

> a Retry-After header field. If no Retry-After is given, the client MUST act as if it had received a 500 (Server Internal Error) response.
>
> A client (proxy or UAC) receiving a 503 (Service Unavailable) SHOULD attempt to forward the request to an alternate server. It SHOULD NOT forward any other requests to that server for the duration specified in the Retry-After header field, if present.
>
> Servers MAY refuse the connection or drop the request instead of responding with 503 (Service Unavailable).

The objective is to provide a mechanism to move the work of the overloaded server to another server so that the request can be processed. The Retry-After header field, when present, is meant to allow a server to tell an upstream element to back off for a period of time, so that the overloaded server can work through its backlog of work.

RFC 3261 also instructs proxies to not forward 503 responses upstream, at SHOULD NOT strength. This is to avoid the upstream server of mistakingly concluding that the proxy is overloaded when, in fact, the problem was an element further downstream.

5. Problems with the Mechanism

At the surface, the 503 mechanism seems workable. Unfortunately, this mechanism has had numerous problems in actual deployment. These problems are described here.

5.1. Load Amplification

The principal problem with the 503 mechanism is that it tends to substantially amplify the load in the network when the network is overloaded, causing further escalation of the problem and introducing the very real possibility of congestive collapse. Consider the topology in Figure 1.

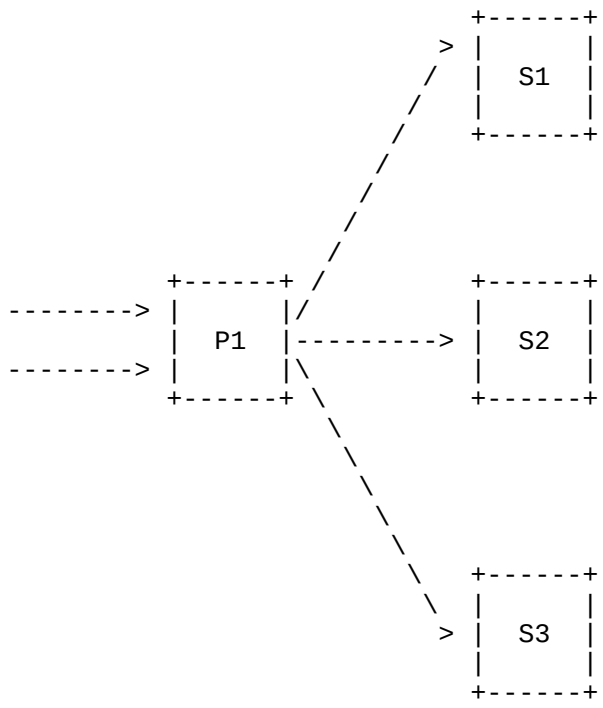

Figure 1

Proxy P1 receives SIP requests from many sources and acts solely as a load balancer, proxying the requests to servers S1, S2, and S3 for processing. The input load increases to the point where all three servers become overloaded. Server S1, when it receives its next request, generates a 503. However, because the server is loaded, it might take some time to generate the 503. If SIP is being run over UDP, this may result in request retransmissions, which further increase the work on S1. Even in the case of TCP, if the server is loaded and the kernel cannot send TCP acknowledgements fast enough, TCP retransmits may occur. When the 503 is received by P1, it retries the request on S2. S2 is also overloaded and eventually generates a 503, but in the interim may also be hit with retransmits. P1 once again tries another server, this time S3, which also eventually rejects it with a 503.

Thus, the processing of this request, which ultimately failed, involved four SIP transactions (client to P1, P1 to S1, P1 to S2, P1 to S3), each of which may have involved many retransmissions -- up to seven in the case of UDP. Thus, under unloaded conditions, a single request from a client would generate one request (to S1, S2, or S3) and two responses (from S1 to P1, then P1 to the client). When the

network is overloaded, a single request from the client, before timing out, could generate as many as 18 requests and as many responses when UDP is used! The situation is better with TCP (or any reliable transport in general), but even if there was never a TCP segment retransmitted, a single request from the client can generate three requests and four responses. Each server had to expend resources to process these messages. Thus, more messages and more work were sent into the network at the point at which the elements became overloaded. The 503 mechanism works well when a single element is overloaded. But when the problem is overall network load, the 503 mechanism actually generates more messages and more work for all servers, ultimately resulting in the rejection of the request anyway.

The problem becomes amplified further if one considers proxies upstream from P1, as shown in Figure 2.

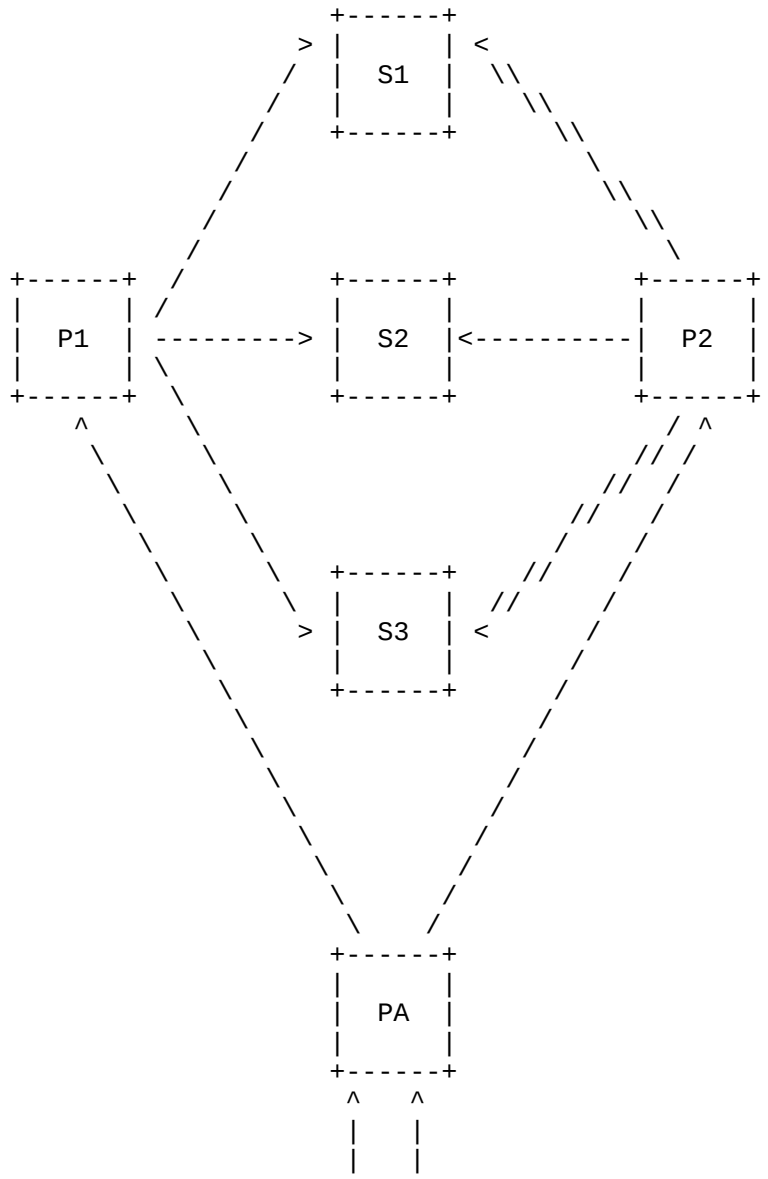

Figure 2

Here, proxy PA receives requests and sends these to proxies P1 or P2. P1 and P2 both load balance across S1 through S3. Assuming again S1 through S3 are all overloaded, a request arrives at PA, which tries P1 first. P1 tries S1, S2, and then S3, and each transaction results in many request retransmits if UDP is used. Since P1 is unable to

eventually process the request, it rejects it. However, since all of its downstream dependencies are busy, it decides to send a 503. This propagates to PA, which tries P2, which tries S1 through S3 again, resulting in a 503 once more. Thus, in this case, we have doubled the number of SIP transactions and overall work in the network compared to the previous case. The problem here is that the fact that S1 through S3 were overloaded was known to P1, but this information was not passed back to PA and through to P2, so that P2 retries S1 through S3 again.

5.2. Underutilization

Interestingly, there are also examples of deployments where the network capacity was greatly reduced as a consequence of the overload mechanism. Consider again Figure 1. Unfortunately, RFC 3261 is unclear on the scope of a 503. When it is received by P1, does the proxy cease sending requests to that IP address? To the hostname? To the URI? Some implementations have chosen the hostname as the scope. When the hostname for a URI points to an SRV record in the DNS, which, in turn, maps to a cluster of downstream servers (S1, S2, and S3 in the example), a 503 response from a single one of them will make the proxy believe that the entire cluster is overloaded. Consequently, proxy P1 will cease sending any traffic to any element in the cluster, even though there are elements in the cluster that are underutilized.

5.3. The Off/On Retry-After Problem

The Retry-After mechanism allows a server to tell an upstream element to stop sending traffic for a period of time. The work that would have otherwise been sent to that server is instead sent to another server. The mechanism is an all-or-nothing technique. A server can turn off all traffic towards it, or none. There is nothing in between. This tends to cause highly oscillatory behavior under even mild overload. Consider a proxy P1 that is balancing requests between two servers S1 and S2. The input load just reaches the point where both S1 and S2 are at 100% capacity. A request arrives at P1 and is sent to S1. S1 rejects this request with a 503, and decides to use Retry-After to clear its backlog. P1 stops sending all traffic to S1. Now, S2 gets traffic, but it is seriously overloaded -- at 200% capacity! It decides to reject a request with a 503 and a Retry-After, which now forces P1 to reject all traffic until S1's Retry-After timer expires. At that point, all load is shunted back to S1, which reaches overload, and the cycle repeats.

It's important to observe that this problem is only observed for servers where there are a small number of upstream elements sending it traffic, as is the case in these examples. If a proxy is accessed

by a large number of clients, each of which sends a small amount of traffic, the 503 mechanism with Retry-After is quite effective when utilized with a subset of the clients. This is because spreading the 503 out amongst the clients has the effect of providing the proxy more fine-grained controls on the amount of work it receives.

5.4. Ambiguous Usages

Unfortunately, the specific instances under which a server is to send a 503 are ambiguous. The result is that implementations generate 503 for many reasons, only some of which are related to actual overload. For example, RFC 3398 [RFC3398], which specifies interworking from SIP to ISDN User Part (ISUP), defines the usage of 503 when the gateway receives certain ISUP cause codes from downstream switches. In these cases, the gateway has ample capacity; it's just that this specific request could not be processed because of a downstream problem. All subsequent requests might succeed if they take a different route in the Public Switched Telephone Network (PSTN).

This causes two problems. First, during periods of overload, it exacerbates the problems above because it causes additional 503 to be fed into the system, causing further work to be generated in conditions of overload. Second, it becomes hard for an upstream element to know whether to retry when a 503 is received. There are classes of failures where trying on another server won't help, since the reason for the failure was that a common downstream resource is unavailable. For example, if servers S1 and S2 share a database and the database fails, a request sent to S1 will result in a 503, but retrying on S2 won't help since the same database is unavailable.

6. Solution Requirements

In this section, we propose requirements for an overload control mechanism for SIP that addresses these problems.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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

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.

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.

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

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

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.

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.

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.

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.

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.

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.

7. Security Considerations

Like all protocol mechanisms, a solution for overload handling must prevent against malicious inside and outside attacks. This document includes requirements for such security functions.

Any mechanism that improves the behavior of SIP elements under load will result in more predictable performance in the face of application-layer denial-of-service attacks.

8. Acknowledgements

The author would like to thank Steve Mayer, Mouli Chandramouli, Robert Whent, Mark Perkins, Joe Stone, Vijay Gurbani, Steve Norreys, Volker Hilt, Spencer Dawkins, Matt Mathis, Juergen Schoenwaelder, and Dale Worley for their contributions to this document.

9. References

9.1. Normative Reference

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

9.2. Informative References

[OUTBOUND] Jennings, C. and R. Mahy, "Managing Client Initiated Connections in the Session Initiation Protocol (SIP)", Work in Progress, October 2008.

[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.

[RFC3398] Camarillo, G., Roach, A., Peterson, J., and L. Ong, "Integrated Services Digital Network (ISDN) User Part (ISUP) to Session Initiation Protocol (SIP) Mapping", RFC 3398, December 2002.

[RFC4412] Schulzrinne, H. and J. Polk, "Communications Resource Priority for the Session Initiation Protocol (SIP)", RFC 4412, February 2006.

Author's Address

Jonathan Rosenberg
Cisco
Edison, NJ
US

EMail: jdrosen@cisco.com
URI: http://www.jdrosen.net