                       Design Considerations for
            Session Initiation Protocol (SIP) Overload Control

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 discusses models and design considerations for a SIP
   overload control mechanism.

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 occurs if a SIP server does not have sufficient resources to
   process all incoming SIP messages. These resources may include CPU,
   memory, input/output, or disk resources.

   Overload can pose a serious problem for a network of SIP servers.
   During periods of overload, the throughput of SIP messages in a
   network of SIP servers can be significantly degraded. In fact,
   overload in a SIP server may lead to a situation in which the
   overload is amplified by retransmissions of SIP messages causing the
   throughput to drop down to a very small fraction of the original
   processing capacity. This is often called congestion collapse.

   An overload control mechanism enables a SIP server to process SIP
   messages close to its capacity limit during times of overload.
   Overload control is used by a SIP server if it is unable to process
   all SIP requests due to resource constraints. There are other
   failure cases in which a SIP server can successfully process incoming
   requests but has to reject them for other reasons. For example, a
   Public Switched Telephone Network (PSTN) gateway that runs out of
   trunk lines but still has plenty of capacity to process SIP messages
   should reject incoming INVITEs using a response such as 488 (Not
   Acceptable Here), as described in [RFC4412]. Similarly, a SIP
   registrar that has lost connectivity to its registration database but
   is still capable of processing SIP messages should reject REGISTER
   requests with a 500 (Server Error) response [RFC3261]. Overload
   control mechanisms do not apply in these cases and SIP provides
   appropriate response codes for them.

   There are cases in which a SIP server runs other services that do not
   involve the processing of SIP messages (e.g., processing of RTP
   packets, database queries, software updates, and event handling).
   These services may, or may not, be correlated with the SIP message
   volume. These services can use up a substantial share of resources
   available on the server (e.g., CPU cycles) and leave the server in a
   condition where it is unable to process all incoming SIP requests.
   In these cases, the SIP server applies SIP overload control
   mechanisms to avoid congestion collapse on the SIP signaling plane.
   However, controlling the number of SIP requests may not significantly
   reduce the load on the server if the resource shortage was created by
   another service. In these cases, it is to be expected that the
   server uses appropriate methods of controlling the resource usage of

other services. The specifics of controlling the resource usage of
other services and their coordination is out of scope for this
document.

The SIP protocol provides a limited mechanism for overload control
through its 503 (Service Unavailable) response code and the
Retry-After header. However, this mechanism cannot prevent overload
of a SIP server and it cannot prevent congestion collapse. In fact,
it 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 Retry-After header, and the
requirements for a SIP overload control mechanism can be found in
[RFC5390]. In addition, 503 is used for other situations, not just
SIP server overload. A SIP overload control process based on 503
would have to specify exactly which cause values trigger the overload
control.

This document discusses the models, assumptions, and design
considerations for a SIP overload control mechanism. The document
originated in the SIP overload control design team and has been
further developed by the SIP Overload Control (SOC) working group.

2. SIP Overload Problem

A key contributor to SIP congestion collapse [RFC5390] is the
regenerative behavior of overload in the SIP protocol. When SIP is
running over the UDP protocol, it will retransmit messages that were
dropped or excessively delayed by a SIP server due to overload and
thereby increase the offered load for the already overloaded server.
This increase in load worsens the severity of the overload condition
and, in turn, causes more messages to be dropped. A congestion
collapse can occur [Hilt] [Noel] [Shen] [Abdelal].

Regenerative behavior under overload should ideally be avoided by any
protocol as this would lead to unstable operation under overload.
However, this is often difficult to achieve in practice. For
example, changing the SIP retransmission timer mechanisms can reduce
the degree of regeneration during overload but will impact the
ability of SIP to recover from message losses. Without any
retransmission, each message that is dropped due to SIP server
overload will eventually lead to a failed transaction.

For a SIP INVITE transaction to be successful, a minimum of three
messages need to be forwarded by a SIP server. Often an INVITE
transaction consists of five or more SIP messages. If a SIP server
under overload randomly discards messages without evaluating them,
the chances that all messages belonging to a transaction are

successfully forwarded will decrease as the load increases. Thus,
the number of transactions that complete successfully will decrease
even if the message throughput of a server remains up and assuming
the overload behavior is fully non-regenerative. A SIP server might
(partially) parse incoming messages to determine if it is a new
request or a message belonging to an existing transaction.
Discarding a SIP message after spending the resources to parse it is
expensive. The number of successful transactions will therefore
decline with an increase in load as fewer resources can be spent on
forwarding messages and more resources are consumed by inspecting
messages that will eventually be dropped. The rate of the decline
depends on the amount of resources spent to inspect each message.

Another challenge for SIP overload control is controlling the rate of
the true traffic source. Overload is often caused by a large number
of user agents (UAs), each of which creates only a single message.
However, the sum of their traffic can overload a SIP server. The
overload mechanisms suitable for controlling a SIP server (e.g., rate
control) may not be effective for individual UAs. In some cases,
there are other non-SIP mechanisms for limiting the load from the
UAs. These may operate independently from, or in conjunction with,
the SIP overload mechanisms described here. In either case, they are
out of scope for this document.

3. Explicit vs. Implicit Overload Control

   The main difference between explicit and implicit overload control is
   the way overload is signaled from a SIP server that is reaching
   overload condition to its upstream neighbors.

   In an explicit overload control mechanism, a SIP server uses an
   explicit overload signal to indicate that it is reaching its capacity
   limit. Upstream neighbors receiving this signal can adjust their
   transmission rate according to the overload signal to a level that is
   acceptable to the downstream server. The overload signal enables a
   SIP server to steer the load it is receiving to a rate at which it
   can perform at maximum capacity.

   Implicit overload control uses the absence of responses and packet
   loss as an indication of overload. A SIP server that is sensing such
   a condition reduces the load it is forwarding to a downstream
   neighbor. Since there is no explicit overload signal, this mechanism
   is robust, as it does not depend on actions taken by the SIP server
   running into overload.

   The ideas of explicit and implicit overload control are in fact
   complementary. By considering implicit overload indications, a
   server can avoid overloading an unresponsive downstream neighbor. An

explicit overload signal enables a SIP server to actively steer the
incoming load to a desired level.

4. System Model

The model shown in Figure 1 identifies fundamental components of an
explicit SIP overload control mechanism:

SIP Processor: The SIP Processor processes SIP messages and is the
   component that is protected by overload control.

Monitor: The Monitor measures the current load of the SIP Processor
   on the receiving entity. It implements the mechanisms needed to
   determine the current usage of resources relevant for the SIP
   Processor and reports load samples (S) to the Control Function.

Control Function: The Control Function implements the overload
   control algorithm. The Control Function uses the load samples (S)
   and determines if overload has occurred and a throttle (T) needs
   to be set to adjust the load sent to the SIP Processor on the
   receiving entity. The Control Function on the receiving entity
   sends load feedback (F) to the sending entity.

Actuator: The Actuator implements the algorithms needed to act on
   the throttles (T) and ensures that the amount of traffic forwarded
   to the receiving entity meets the criteria of the throttle. For
   example, a throttle may instruct the Actuator to not forward more
   than 100 INVITE messages per second. The Actuator implements the
   algorithms to achieve this objective, e.g., using message gapping.
   It also implements algorithms to select the messages that will be
   affected and determine whether they are rejected or redirected.

The type of feedback (F) conveyed from the receiving to the sending
entity depends on the overload control method used (i.e., loss-based,
rate-based, window-based, or signal-based overload control; see
Section 9), the overload control algorithm (see Section 11), as well
as other design parameters. The feedback (F) enables the sending
entity to adjust the amount of traffic forwarded to the receiving
entity to a level that is acceptable to the receiving entity without
causing overload.

Figure 1 depicts a general system model for overload control. In
this diagram, one instance of the control function is on the sending
entity (i.e., associated with the actuator) and one is on the
receiving entity (i.e., associated with the Monitor). However, a
specific mechanism may not require both elements. In this case, one
of two control function elements can be empty and simply passes along
feedback. For example, if (F) is defined as a loss-rate (e.g.,

reduce traffic by 10%), there is no need for a control function on
the sending entity as the content of (F) can be copied directly into
(T).

The model in Figure 1 shows a scenario with one sending and one
receiving entity. In a more realistic scenario, a receiving entity
will receive traffic from multiple sending entities and vice versa
(see Section 6). The feedback generated by a Monitor will therefore
often be distributed across multiple Actuators. A Monitor needs to
be able to split the load it can process across multiple sending
entities and generate feedback that correctly adjusts the load each
sending entity is allowed to send. Similarly, an Actuator needs to
be prepared to receive different levels of feedback from different
receiving entities and throttle traffic to these entities
accordingly.

In a realistic deployment, SIP messages will flow in both directions,
from server B to server A as well as server A to server B. The
overload control mechanisms in each direction can be considered
independently. For messages flowing from server A to server B, the
sending entity is server A and the receiving entity is server B, and
vice versa. The control loops in both directions operate
independently.

```
              Sending              Receiving
               Entity                Entity
        +---------------+      +---------------+
        |   Server A    |      |   Server B    |
        | +---------+   |      | +---------+   |    -+
        | | Control |   |  F   | | Control |   |     |
        | | Function|<-+------+--| Function|   |     |
        | +---------+   |      | +---------+   |     |
        |    T |        |      |      ^        |     | Overload
        |      v        |      |      | S      |     | Control
        | +---------+   |      | +---------+   |     |
        | | Actuator|   |      | | Monitor |   |     |
        | +---------+   |      | +---------+   |     |
        |     |         |      |      ^        |    -+
        |     v         |      |      |        |    -+
        | +---------+   |      | +---------+   |     |
   <-+--|  SIP    |   |      | |  SIP    |   |     | SIP
   --+->|Processor |--+------+->|Processor |--+->   | System
        | +---------+   |      | +---------+   |     |
        +---------------+      +---------------+    -+
```
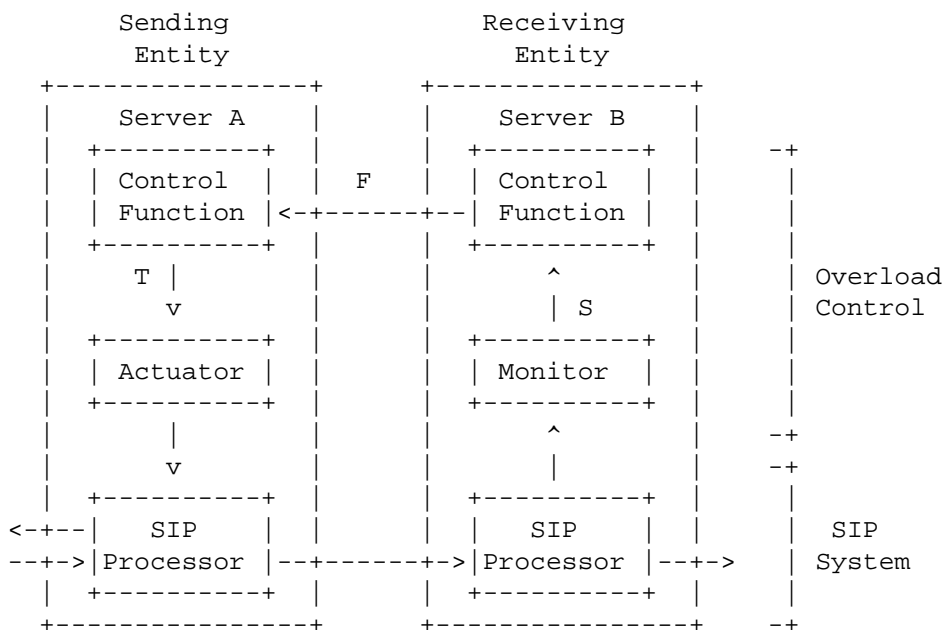
              Figure 1: System Model for Explicit Overload Control

5. Degree of Cooperation

   A SIP request is usually processed by more than one SIP server on its
   path to the destination. Thus, a design choice for an explicit
   overload control mechanism is where to place the components of
   overload control along the path of a request and, in particular,
   where to place the Monitor and Actuator. This design choice
   determines the degree of cooperation between the SIP servers on the
   path. Overload control can be implemented hop-by-hop with the
   Monitor on one server and the Actuator on its direct upstream
   neighbor. Overload control can be implemented end-to-end with
   Monitors on all SIP servers along the path of a request and an
   Actuator on the sender. In this case, the Control Functions
   associated with each Monitor have to cooperate to jointly determine
   the overall feedback for this path. Finally, overload control can be
   implemented locally on a SIP server if the Monitor and Actuator
   reside on the same server. In this case, the sending entity and
   receiving entity are the same SIP server, and the Actuator and
   Monitor operate on the same SIP Processor (although, the Actuator
   typically operates on a pre-processing stage in local overload
   control). Local overload control is an internal overload control
   mechanism, as the control loop is implemented internally on one
   server. Hop-by-hop and end-to-end are external overload control
   mechanisms. All three configurations are shown in Figure 2.
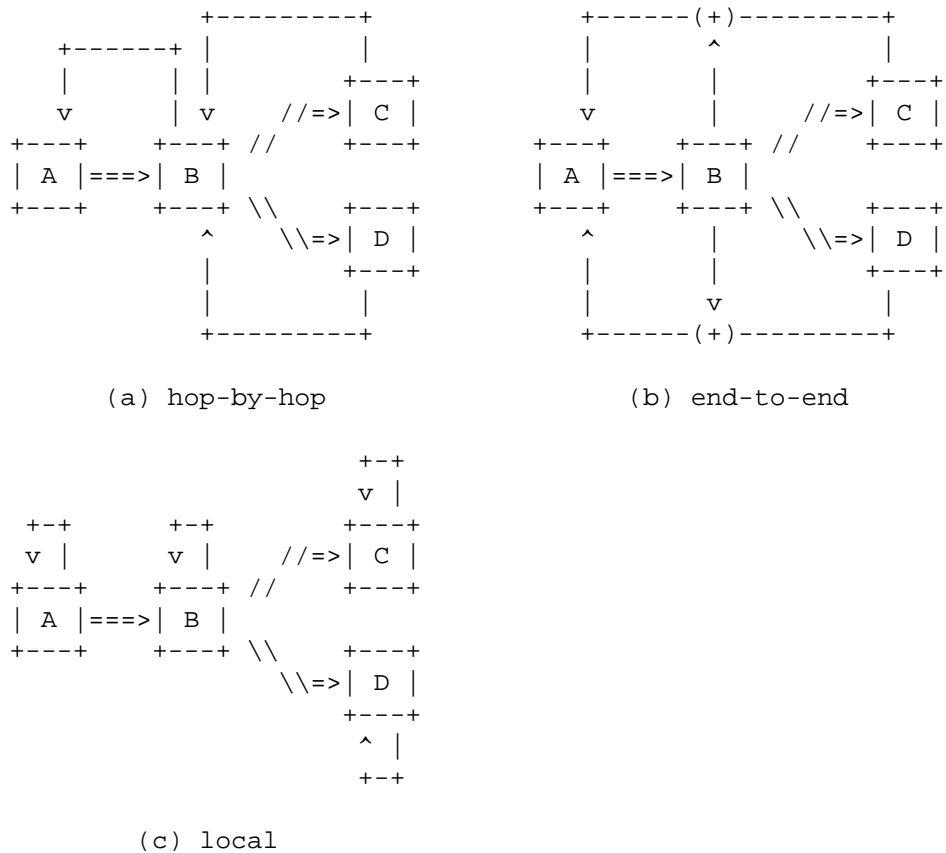
```
                      +---------+         +------(+)---------+
          +------+    |         |         |         ^        |
          |      | |  |   +---+ |         |         |   +---+ |
          v      | v  //=>| C | |         v         |   //=>| C |
        +---+    +---+ //  +---+         +---+     +---+ //  +---+
        | A |===>| B |                  | A |===>| B |
        +---+    +---+ \\  +---+         +---+     +---+ \\  +---+
          ^           \\=>| D |           ^         |   \\=>| D |
          |               +---+           |         |       +---+
          |          |                    |         v           |
          +---------+                     +------(+)---------+

          (a) hop-by-hop                     (b) end-to-end


                             +-+
                             v |
          +-+        +-+      +---+
          v |        v |  //=>| C |
        +---+      +---+  //   +---+
        | A |===>| B |  //
        +---+      +---+  \\   +---+
                          \\=>| D |
                               +---+
                               ^ |
                               +-+


              (c) local


        ==> SIP request flow
        <-- Overload feedback loop


            Figure 2: Degree of Cooperation between Servers
```

5.1. Hop-by-Hop

   The idea of hop-by-hop overload control is to instantiate a separate
   control loop between all neighboring SIP servers that directly
   exchange traffic. That is, the Actuator is located on the SIP server
   that is the direct upstream neighbor of the SIP server that has the
   corresponding Monitor. Each control loop between two servers is
   completely independent of the control loop between other servers
   further up- or downstream. In the example in Figure 2(a), three
   independent overload control loops are instantiated: A - B, B - C,
   and B - D. Each loop only controls a single hop. Overload feedback
   received from a downstream neighbor is not forwarded further
   upstream. Instead, a SIP server acts on this feedback, for example,
   by rejecting SIP messages if needed. If the upstream neighbor of a
   server also becomes overloaded, it will report this problem to its

upstream neighbors, which again take action based on the reported
feedback. Thus, in hop-by-hop overload control, overload is always
resolved by the direct upstream neighbors of the overloaded server
without the need to involve entities that are located multiple SIP
hops away.

Hop-by-hop overload control reduces the impact of overload on a SIP
network and can avoid congestion collapse. It is simple and scales
well to networks with many SIP entities. An advantage is that it
does not require feedback to be transmitted across multiple-hops,
possibly crossing multiple trust domains. Feedback is sent to the
next hop only. Furthermore, it does not require a SIP entity to
aggregate a large number of overload status values or keep track of
the overload status of SIP servers it is not communicating with.

5.2. End-to-End

End-to-end overload control implements an overload control loop along
the entire path of a SIP request, from user agent client (UAC) to
user agent server (UAS). An end-to-end overload control mechanism
consolidates overload information from all SIP servers on the way
(including all proxies and the UAS) and uses this information to
throttle traffic as far upstream as possible. An end-to-end overload
control mechanism has to be able to frequently collect the overload
status of all servers on the potential path(s) to a destination and
combine this data into meaningful overload feedback.

A UA or SIP server only throttles requests if it knows that these
requests will eventually be forwarded to an overloaded server. For
example, if D is overloaded in Figure 2(b), A should only throttle
requests it forwards to B when it knows that they will be forwarded
to D. It should not throttle requests that will eventually be
forwarded to C, since server C is not overloaded. In many cases, it
is difficult for A to determine which requests will be routed to C
and D, since this depends on the local routing decision made by B.
These routing decisions can be highly variable and, for example,
depend on call-routing policies configured by the user, services
invoked on a call, load-balancing policies, etc. A previous message
to a target that has been routed through an overloaded server does
not necessarily mean that the next message to this target will also
be routed through the same server.

The main problem of end-to-end overload control is its inherent
complexity, since UAC or SIP servers need to monitor all potential
paths to a destination in order to determine which requests should be
throttled and which requests may be sent. Even if this information
is available, it is not clear which path a specific request will
take.

A variant of end-to-end overload control is to implement a control
loop between a set of well-known SIP servers along the path of a SIP
request. For example, an overload control loop can be instantiated
between a server that only has one downstream neighbor or a set of
closely coupled SIP servers. A control loop spanning multiple hops
can be used if the sending entity has full knowledge about the SIP
servers on the path of a SIP message.

Overload control for SIP servers is different from end-to-end
congestion control used by transport protocols such as TCP. The
traffic exchanged between SIP servers consists of many individual SIP
messages. Each SIP message is created by a SIP UA to achieve a
specific goal (e.g., to start setting up a call). All messages have
their own source and destination addresses. Even SIP messages
containing identical SIP URIs (e.g., a SUBSCRIBE and an INVITE
message to the same SIP URI) can be routed to different destinations.
This is different from TCP, where the traffic exchanged between
routers consists of packets belonging to a usually longer flow of
messages exchanged between a source and a destination (e.g., to
transmit a file). If congestion occurs, the sources can detect this
condition and adjust the rate at which the next packets are
transmitted.

5.3. Local Overload Control

The idea of local overload control (see Figure 2(c)) is to run the
Monitor and Actuator on the same server. This enables the server to
monitor the current resource usage and to reject messages that can't
be processed without overusing local resources. The fundamental
assumption behind local overload control is that it is less resource
consuming for a server to reject messages than to process them. A
server can therefore reject the excess messages it cannot process to
stop all retransmissions of these messages. Since rejecting messages
does consume resources on a SIP server, local overload control alone
cannot prevent a congestion collapse.

Local overload control can be used in conjunction with other overload
control mechanisms and provides an additional layer of protection
against overload. It is fully implemented within a SIP server and
does not require cooperation between servers. In general, SIP
servers should apply other overload control techniques to control
load before a local overload control mechanism is activated as a
mechanism of last resort.

6. Topologies

   The following topologies describe four generic SIP server
   configurations. These topologies illustrate specific challenges for
   an overload control mechanism. An actual SIP server topology is
   likely to consist of combinations of these generic scenarios.

   In the "load balancer" configuration shown in Figure 3(a), a set of
   SIP servers (D, E, and F) receives traffic from a single source A. A
   load balancer is a typical example for such a configuration. In this
   configuration, overload control needs to prevent server A (i.e., the
   load balancer) from sending too much traffic to any of its downstream
   neighbors D, E, and F. If one of the downstream neighbors becomes
   overloaded, A can direct traffic to the servers that still have
   capacity. If one of the servers acts as a backup, it can be
   activated once one of the primary servers reaches overload.

   If A can reliably determine that D, E, and F are its only downstream
   neighbors and all of them are in overload, it may choose to report
   overload upstream on behalf of D, E, and F. However, if the set of
   downstream neighbors is not fixed or only some of them are in
   overload, then A should not activate an overload control since A can
   still forward the requests destined to non-overloaded downstream
   neighbors. These requests would be throttled as well if A would use
   overload control towards its upstream neighbors.

   In some cases, the servers D, E, and F are in a server farm and are
   configured to appear as a single server to their upstream neighbors.
   In this case, server A can report overload on behalf of the server
   farm. If the load balancer is not a SIP entity, servers D, E, and F
   can report the overall load of the server farm (i.e., the load of the
   virtual server) in their messages. As an alternative, one of the
   servers (e.g., server E) can report overload on behalf of the server
   farm. In this case, not all messages contain overload control
   information, and all upstream neighbors need to be served by server E
   periodically to ensure that updated information is received.

   In the "multiple sources" configuration shown in Figure 3(b), a SIP
   server D receives traffic from multiple upstream sources A, B, and C.
   Each of these sources can contribute a different amount of traffic,
   which can vary over time. The set of active upstream neighbors of D
   can change as servers may become inactive, and previously inactive
   servers may start contributing traffic to D.

   If D becomes overloaded, it needs to generate feedback to reduce the
   amount of traffic it receives from its upstream neighbors. D needs
   to decide by how much each upstream neighbor should reduce traffic.
   This decision can require the consideration of the amount of traffic

sent by each upstream neighbor and it may need to be re-adjusted as
the traffic contributed by each upstream neighbor varies over time.
Server D can use a local fairness policy to determine how much
traffic it accepts from each upstream neighbor.

In many configurations, SIP servers form a "mesh" as shown in Figure
3(c). Here, multiple upstream servers A, B, and C forward traffic to
multiple alternative servers D and E. This configuration is a
combination of the "load balancer" and "multiple sources" scenario.

```
             +---+                  +---+
          /->| D |                  | A |-\
         /   +---+                  +---+  \
        /                                  \   +---+
  +---+-/      +---+           +---+     \->|   |
  | A |------>| E |           | B |------>| D |
  +---+-\      +---+           +---+    /->|   |
         \                              /   +---+
          \   +---+                  +---+  /
           \->| F |                  | C |-/
             +---+                  +---+


     (a) load balancer              (b) multiple sources

  +---+
  | A |---\
  +---+-\  \---->+---+              a--\
         \/----->| D |                  \
  +---+--/\   /->+---+              b--\ \--->+---+
  | B |     \/                       \--->|   |
  +---+---\/\--->+---+              c-------->| D |
         /\---->| E |                        |   |
  +---+--/   /->+---+              ...   /--->+---+
  | C |-----/                           /
  +---+                            z--/


          (c) mesh                     (d) edge proxy
```
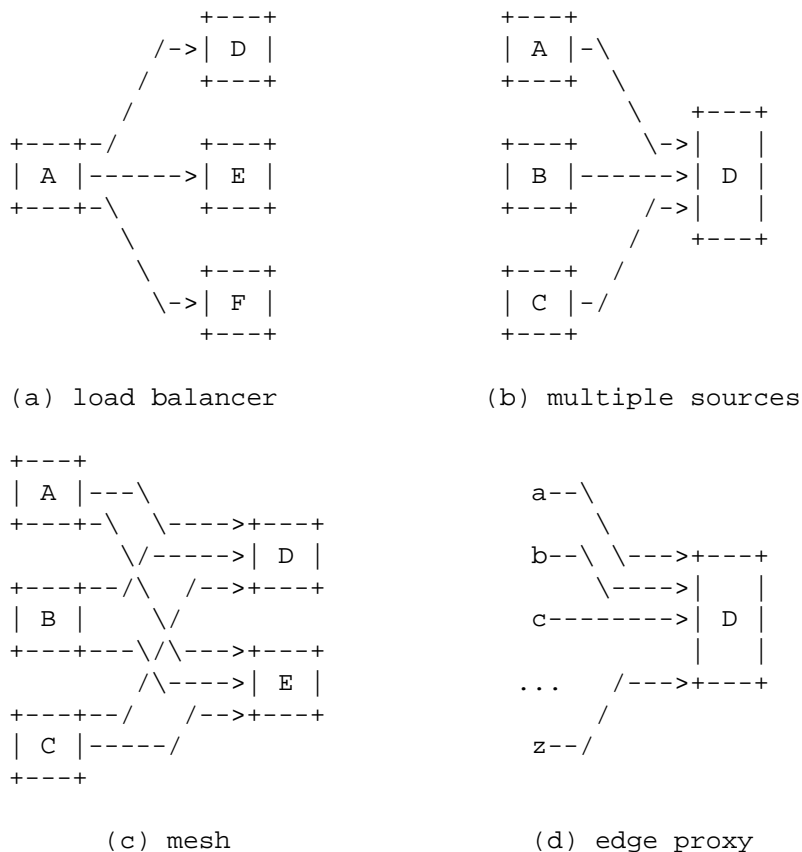
                     Figure 3: Topologies

Overload control that is based on reducing the number of messages a
sender is allowed to send is not suited for servers that receive
requests from a very large population of senders, each of which only
sends a very small number of requests. This scenario is shown in
Figure 3(d). An edge proxy that is connected to many UAs is a
typical example for such a configuration. Since each UA typically
infrequently sends requests, which are often related to the same
session, it can't decrease its message rate to resolve the overload.

A SIP server that receives traffic from many sources, which each
contribute only a small number of requests, can resort to local
overload control by rejecting a percentage of the requests it
receives with 503 (Service Unavailable) responses. Since it has many
upstream neighbors, it can send 503 (Service Unavailable) to a
fraction of them to gradually reduce load without entirely stopping
all incoming traffic. The Retry-After header can be used in 503
(Service Unavailable) responses to ask upstream neighbors to wait a
given number of seconds before trying the request again. Using 503
(Service Unavailable) can, however, not prevent overload if a large
number of sources create requests (e.g., to place calls) at the same
time.

Note: The requirements of the "edge proxy" topology are different
from the ones of the other topologies, which may require a different
method for overload control.

7. Fairness

There are many different ways to define fairness between multiple
upstream neighbors of a SIP server. In the context of SIP server
overload, it is helpful to describe two categories of fairness: basic
fairness and customized fairness. With basic fairness, a SIP server
treats all requests equally and ensures that each request has the
same chance of succeeding. With customized fairness, the server
allocates resources according to different priorities. An example
application of the basic fairness criteria is the "Third caller
receives free tickets" scenario, where each call attempt should have
an equal success probability in connecting through an overloaded SIP
server, irrespective of the service provider in which the call was
initiated. An example of customized fairness would be a server that
assigns different resource allocations to its upstream neighbors
(e.g., service providers) as defined in a service level agreement
(SLA).

8. Performance Metrics

The performance of an overload control mechanism can be measured
using different metrics.

A key performance indicator is the goodput of a SIP server under
overload. Ideally, a SIP server will be enabled to perform at its
maximum capacity during periods of overload. For example, if a SIP
server has a processing capacity of 140 INVITE transactions per
second, then an overload control mechanism should enable it to
process 140 INVITEs per second even if the offered load is much
higher. The delay introduced by a SIP server is another important
indicator. An overload control mechanism should ensure that the

delay encountered by a SIP message is not increased significantly
during periods of overload. Significantly increased delay can lead
to time-outs and retransmission of SIP messages, making the overload
worse.

Responsiveness and stability are other important performance
indicators. An overload control mechanism should quickly react to an
overload occurrence and ensure that a SIP server does not become
overloaded, even during sudden peaks of load. Similarly, an overload
control mechanism should quickly stop rejecting requests if the
overload disappears. Stability is another important criteria. An
overload control mechanism should not cause significant oscillations
of load on a SIP server. The performance of SIP overload control
mechanisms is discussed in [Noel], [Shen], [Hilt], and [Abdelal].

In addition to the above metrics, there are other indicators that are
relevant for the evaluation of an overload control mechanism:

Fairness: Which type of fairness does the overload control mechanism
   implement?

Self-limiting: Is the overload control self-limiting if a SIP server
   becomes unresponsive?

Changes in neighbor set: How does the mechanism adapt to a changing
   set of sending entities?

Data points to monitor: Which and how many data points does an
   overload control mechanism need to monitor?

Computational load: What is the (CPU) load created by the overload
   "Monitor" and "Actuator"?

9. Explicit Overload Control Feedback

   Explicit overload control feedback enables a receiver to indicate how
   much traffic it wants to receive. Explicit overload control
   mechanisms can be differentiated based on the type of information
   conveyed in the overload control feedback and whether the control
   function is in the receiving or sending entity (receiver- vs. sender-
   based overload control), or both.

9.1. Rate-Based Overload Control

   The key idea of rate-based overload control is to limit the request
   rate at which an upstream element is allowed to forward traffic to
   the downstream neighbor. If overload occurs, a SIP server instructs

each upstream neighbor to send, at most, X requests per second. Each
upstream neighbor can be assigned a different rate cap.

An example algorithm for an Actuator in the sending entity is request
gapping. After transmitting a request to a downstream neighbor, a
server waits for 1/X seconds before it transmits the next request to
the same neighbor. Requests that arrive during the waiting period
are not forwarded and are either redirected, rejected, or buffered.
Request gapping only affects requests that are targeted by overload
control (e.g., requests that initiate a transaction and not
retransmissions in an ongoing transaction).

The rate cap ensures that the number of requests received by a SIP
server never increases beyond the sum of all rate caps granted to
upstream neighbors. Rate-based overload control protects a SIP
server against overload, even during load spikes assuming there are
no new upstream neighbors that start sending traffic. New upstream
neighbors need to be considered in the rate caps assigned to all
upstream neighbors. The rate assigned to upstream neighbors needs to
be adjusted when new neighbors join. During periods when new
neighbors are joining, overload can occur in extreme cases until the
rate caps of all servers are adjusted to again match the overall rate
cap of the server. The overall rate cap of a SIP server is
determined by an overload control algorithm, e.g., based on system
load.

Rate-based overload control requires a SIP server to assign a rate
cap to each of its upstream neighbors while it is activated.
Effectively, a server needs to assign a share of its overall capacity
to each upstream neighbor. A server needs to ensure that the sum of
all rate caps assigned to upstream neighbors does not substantially
oversubscribe its actual processing capacity. This requires a SIP
server to keep track of the set of upstream neighbors and to adjust
the rate cap if a new upstream neighbor appears or an existing
neighbor stops transmitting. For example, if the capacity of the
server is X and this server is receiving traffic from two upstream
neighbors, it can assign a rate of X/2 to each of them. If a third
sender appears, the rate for each sender is lowered to X/3. If the
overall rate cap is too high, a server may experience overload. If
the cap is too low, the upstream neighbors will reject requests even
though they could be processed by the server.

An approach for estimating a rate cap for each upstream neighbor is
using a fixed proportion of a control variable, X, where X is
initially equal to the capacity of the SIP server. The server then
increases or decreases X until the workload arrival rate matches the
actual server capacity. Usually, this will mean that the sum of the
rate caps sent out by the server (=X) exceeds its actual capacity,

but enables upstream neighbors who are not generating more than their
fair share of the work to be effectively unrestricted. In this
approach, the server only has to measure the aggregate arrival rate.
However, since the overall rate cap is usually higher than the actual
capacity, brief periods of overload may occur.

9.2. Loss-Based Overload Control

   A loss percentage enables a SIP server to ask an upstream neighbor to
   reduce the number of requests it would normally forward to this
   server by X%. For example, a SIP server can ask an upstream neighbor
   to reduce the number of requests this neighbor would normally send by
   10%. The upstream neighbor then redirects or rejects 10% of the
   traffic that is destined for this server.

   To implement a loss percentage, the sending entity may employ an
   algorithm to draw a random number between 1 and 100 for each request
   to be forwarded. The request is not forwarded to the server if the
   random number is less than or equal to X.

   An advantage of loss-based overload control is that the receiving
   entity does not need to track the set of upstream neighbors or the
   request rate it receives from each upstream neighbor. It is
   sufficient to monitor the overall system utilization. To reduce
   load, a server can ask its upstream neighbors to lower the traffic
   forwarded by a certain percentage. The server calculates this
   percentage by combining the loss percentage that is currently in use
   (i.e., the loss percentage the upstream neighbors are currently using
   when forwarding traffic), the current system utilization, and the
   desired system utilization. For example, if the server load
   approaches 90% and the current loss percentage is set to a 50%
   traffic reduction, then the server can decide to increase the loss
   percentage to 55% in order to get to a system utilization of 80%.
   Similarly, the server can lower the loss percentage if permitted by
   the system utilization.

   Loss-based overload control requires that the throttle percentage be
   adjusted to the current overall number of requests received by the
   server. This is particularly important if the number of requests
   received fluctuates quickly. For example, if a SIP server sets a
   throttle value of 10% at time t1 and the number of requests increases
   by 20% between time t1 and t2 (t1<t2), then the server will see an
   increase in traffic by 10% between time t1 and t2. This is even
   though all upstream neighbors have reduced traffic by 10%. Thus,
   percentage throttling requires an adjustment of the throttling
   percentage in response to the traffic received and may not always be
   able to prevent a server from encountering brief periods of overload
   in extreme cases.

9.3. Window-Based Overload Control

   The key idea of window-based overload control is to allow an entity
   to transmit a certain number of messages before it needs to receive a
   confirmation for the messages in transit. Each sender maintains an
   overload window that limits the number of messages that can be in
   transit without being confirmed. Window-based overload control is
   inspired by TCP [RFC0793].

   Each sender maintains an unconfirmed message counter for each
   downstream neighbor it is communicating with. For each message sent
   to the downstream neighbor, the counter is increased. For each
   confirmation received, the counter is decreased. The sender stops
   transmitting messages to the downstream neighbor when the unconfirmed
   message counter has reached the current window size.

   A crucial parameter for the performance of window-based overload
   control is the window size. Each sender has an initial window size
   it uses when first sending a request. This window size can be
   changed based on the feedback it receives from the receiver.

   The sender adjusts its window size as soon as it receives the
   corresponding feedback from the receiver. If the new window size is
   smaller than the current unconfirmed message counter, the sender
   stops transmitting messages until more messages are confirmed and the
   current unconfirmed message counter is less than the window size.

   Note that the reception of a 100 (Trying) response does not provide a
   confirmation for the successful processing of a message. 100
   (Trying) responses are often created by a SIP server very early in
   processing and do not indicate that a message has been successfully
   processed and cleared from the input buffer. If the downstream
   neighbor is a stateless proxy, it will not create 100 (Trying)
   responses at all and will instead pass through 100 (Trying) responses
   created by the next stateful server. Also, 100 (Trying) responses
   are typically only created for INVITE requests. Explicit message
   confirmations do not have these problems.

   Window-based overload control is similar to rate-based overload
   control in that the total available receiver buffer space needs to be
   divided among all upstream neighbors. However, unlike rate-based
   overload control, window-based overload control is self-limiting and
   can ensure that the receiver buffer does not overflow under normal
   conditions. The transmission of messages by senders is clocked by
   message confirmations received from the receiver. A buffer overflow
   can occur in extreme cases when a large number of new upstream

neighbors arrives at the same time. However, senders will eventually
stop transmitting new requests once their initial sending window is
closed.

In window-based overload control, the number of messages a sender is
allowed to send can frequently be set to zero. In this state, the
sender needs to be informed when it is allowed to send again and when
the receiver window has opened up. However, since the sender is not
allowed to transmit messages, the receiver cannot convey the new
window size by piggybacking it in a response to another message.
Instead, it needs to inform the sender through another mechanism,
e.g., by sending a message that contains the new window size.

9.4. Overload Signal-Based Overload Control

The key idea of overload signal-based overload control is to use the
transmission of a 503 (Service Unavailable) response as a signal for
overload in the downstream neighbor. After receiving a 503 (Service
Unavailable) response, the sender reduces the load forwarded to the
downstream neighbor to avoid triggering more 503 (Service
Unavailable) responses. The sender keeps reducing the load if more
503 (Service Unavailable) responses are received. Note that this
scheme is based on the use of 503 (Service Unavailable) responses
without the Retry-After header, as the Retry-After header would
require a sender to entirely stop forwarding requests. It should
also be noted that 503 responses can be generated for reasons other
than overload (e.g., server maintenance).

A sender that has not received 503 (Service Unavailable) responses
for a while but is still throttling traffic can start to increase the
offered load. By slowly increasing the traffic forwarded, a sender
can detect that overload in the downstream neighbor has been resolved
and more load can be forwarded. The load is increased until the
sender receives another 503 (Service Unavailable) response or is
forwarding all requests it has. A possible algorithm for adjusting
traffic is additive increase/multiplicative decrease (AIMD).

Overload signal-based overload control is a sender-based overload
control mechanism.

9.5. On-/Off Overload Control

On-/off overload control feedback enables a SIP server to turn the
traffic it is receiving either on or off. The 503 (Service
Unavailable) response with a Retry-After header implements on-/off
overload control. On-/off overload control is less effective in
controlling load than the fine grained control methods above. All of

   the above methods can realize on-/off overload control, e.g., by
   setting the allowed rate to either zero or unlimited.

10. Implicit Overload Control

   Implicit overload control ensures that the transmission of a SIP
   server is self-limiting. It slows down the transmission rate of a
   sender when there is an indication that the receiving entity is
   experiencing overload. Such an indication can be that the receiving
   entity is not responding within the expected timeframe or is not
   responding at all. The idea of implicit overload control is that
   senders should try to sense overload of a downstream neighbor even if
   there is no explicit overload control feedback. It avoids an
   overloaded server, which has become unable to generate overload
   control feedback, from being overwhelmed with requests.

   Window-based overload control is inherently self-limiting since a
   sender cannot continue to pass messages without receiving
   confirmations. All other explicit overload control schemes described
   above do not have this property and require additional implicit
   controls to limit transmissions in case an overloaded downstream
   neighbor does not generate explicit feedback.

11. Overload Control Algorithms

   An important aspect of the design of an overload control mechanism is
   the overload control algorithm. The control algorithm determines
   when the amount of traffic to a SIP server needs to be decreased and
   when it can be increased. In terms of the model described in Section
   4, the control algorithm takes (S) as an input value and generates
   (T) as a result.

   Overload control algorithms have been studied to a large extent and
   many different overload control algorithms exist. With many
   different overload control algorithms available, it seems reasonable
   to suggest a baseline algorithm in a specification for a SIP overload
   control mechanism and allow the use of other algorithms if they
   provide the same protocol semantics. This will also allow the
   development of future algorithms, which may lead to better
   performance. Conversely, the overload control mechanism should allow
   the use of different algorithms if they adhere to the defined
   protocol semantics.

12. Message Prioritization

   Overload control can require a SIP server to prioritize requests and
   select requests to be rejected or redirected. The selection is
   largely a matter of local policy of the SIP server, the overall
   network, and the services the SIP server provides.

   While there are many factors that can affect the prioritization of
   SIP requests, the Resource-Priority Header (RPH) field [RFC4412] is a
   prime candidate for marking the prioritization of SIP requests.
   Depending on the particular network and the services it offers, a
   particular namespace and priority value in the RPH could indicate i)
   a high priority request, which should be preserved if possible during
   overload, ii) a low priority request, which should be dropped during
   overload, or iii) a label, which has no impact on message
   prioritization in this network.

   For a number of reasons, responses should not be targeted in order to
   reduce SIP server load. Responses cannot be rejected and would have
   to be dropped. This triggers the retransmission of the request plus
   the response, leading to even more load. In addition, the request
   associated with a response has already been processed and dropping
   the response will waste the efforts that have been spent on the
   request. Most importantly, rejecting a request effectively also
   removes the request and the response. If no requests are passed
   along, there will be no responses coming back in return.

   Overload control does not change the retransmission behavior of SIP.
   Retransmissions are triggered using procedures defined in RFC 3261
   [RFC3261] and are not subject to throttling.

13. Operational Considerations

   In addition to the design considerations discussed above,
   implementations of a SIP overload control mechanism need to take the
   following operational aspects into consideration. These aspects,
   while important, are out of scope for this document and are left for
   further discussion in other documents.

    Selection of feedback type: A SIP overload control mechanism can
       support one or multiple types of explicit overload control
       feedback. Using a single type of feedback (e.g., loss-based
       feedback) has the advantage of simplifying the protocol and
       implementations. Supporting multiple types of feedback (e.g.,
       loss- and rate-based feedback) provides more flexibility; however,
       it requires a way to select the feedback type used between two
       servers.

   Event reporting: Overload is a serious condition for any network of
      SIP servers, even if it is handled properly by an overload control
      mechanism. Overload events should therefore be reported by a SIP
      server, e.g., through a logging or management interface.

14. Security Considerations

   This document presents an overview of several overload control
   feedback mechanisms. These mechanisms and design consideration are
   presented as input to other documents that will specify a particular
   feedback mechanism. Specific security measures pertinent to a
   particular overload feedback mechanism will be discussed in the
   context of a document specifying that security mechanism. However,
   there are common security considerations that must be taken into
   account regardless of the choice of a final mechanism.

   First, the rate-based mechanism surveyed in Section 9.1 allocates a
   fixed portion of the total inbound traffic of a server to each of its
   upstream neighbors. Consequently, an attacker can introduce a new
   upstream server for a short duration, causing the overloaded server
   to lower the proportional traffic rate to all other existing servers.
   Introducing many such short-lived servers will cause the aggregate
   rate arriving at the overloaded server to decrease substantially,
   thereby affecting a reduction in the service offered by the server
   under attack and leading to a denial-of-service attack [RFC4732].

   The same problem exists in the windows-based mechanism discussed in
   Section 9.3; however, because of the window acknowledgments sent by
   the overloaded server, the effect is not as drastic (an attacker will
   have to expend resources by constantly sending traffic to keep the
   receiver window full).

   All mechanisms assume that the upstream neighbors of an overloaded
   server follow the feedback received. In the rate- and window-based
   mechanisms, a server can directly verify if upstream neighbors follow
   the requested policies. As the loss-based mechanism described in
   Section 9.2 requires upstream neighbors to reduce traffic by a
   fraction and the current offered load in the upstream neighbor is
   unknown, a server cannot directly verify the compliance of upstream
   neighbors, except when traffic reduction is set to 100%. In this
   case, a server has to rely on heuristics to identify upstream
   neighbors that try to gain an advantage by not reducing load or not
   reducing it at the requested loss-rate. A policing mechanism can be
   used to throttle or block traffic from unfair or malicious upstream
   neighbors. Barring such a widespread policing mechanism, the
   communication link between the upstream neighbors and the overloaded
   server should be such that the identity of both the servers at the
   end of each link can be established and logged. The use of Transport

Layer Security (TLS) and mutual authentication of upstream neighbors
[RFC3261] [RFC5922] can be used for this purpose.

If an attacker controls a server, he or she may maliciously advertise
overload feedback to all of the neighbors of the server, even if the
server is not experiencing overload. This will have the effect of
forcing all of the upstream neighbors to reject or queue messages
arriving to them and destined for the apparently overloaded server
(this, in essence, is diminishing the serving capacity of the
upstream neighbors since they now have to deal with their normal
traffic in addition to rejecting or quarantining the traffic destined
to the overloaded server). All mechanisms allow the attacker to
advertise a capacity of 0, effectively disabling all traffic destined
to the server pretending to be in overload and forcing all the
upstream neighbors to expend resources dealing with this condition.

As before, a remedy for this is to use a communication link such that
the identity of the servers at both ends of the link is established
and logged. The use of TLS and mutual authentication of neighbors
[RFC3261] [RFC5922] can be used for this purpose.

If an attacker controls several servers of a load-balanced cluster,
he or she may maliciously advertise overload feedback from these
servers to all senders. Senders with the policy to redirect traffic
that cannot be processed by an overloaded server will start to
redirect this traffic to the servers that have not reported overload.
This attack can be used to create a denial-of-service attack on these
servers. If these servers are compromised, the attack can be used to
increase the amount of traffic that is passed through the compromised
servers. This attack is ineffective if servers reject traffic based
on overload feedback instead of redirecting it.

15. Informative References

   [Abdelal]   Abdelal, A. and W. Matragi, "Signal-Based Overload
                Control for SIP Servers", 7th Annual IEEE Consumer
                Communications and Networking Conference (CCNC-10), Las
                Vegas, Nevada, USA, January 2010.

   [Hilt]   Hilt, V. and I. Widjaja, "Controlling overload in
                networks of SIP servers", IEEE International Conference
                on Network Protocols (ICNP'08), Orlando, Florida, October
                2008.

   [Noel]   Noel, E. and C. Johnson, "Novel Overload Controls for SIP
                Networks", International Teletraffic Congress (ITC 21),
                Paris, France, September 2009.

   [RFC0793]   Postel, J., "Transmission Control Protocol", STD 7, RFC
                793, September 1981.

   [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.

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

   [RFC4732]   Handley, M., Rescorla, E., and IAB, "Internet Denial-of-
                Service Considerations", RFC 4732, December 2006.

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

   [RFC5922]   Gurbani, V., Lawrence, S., and A. Jeffrey, "Domain
                Certificates in the Session Initiation Protocol (SIP)",
                RFC 5922, June 2010.

   [Shen]   Shen, C., Schulzrinne, H., and E. Nahum, "Session
                Initiation Protocol (SIP) Server Overload Control: Design
                and Evaluation, Principles", Systems and Applications of
                IP Telecommunications (IPTComm'08), Heidelberg, Germany,
                July 2008.

Appendix A. Contributors

   Many thanks for the contributions, comments, and feedback on this
   document to: Mary Barnes (Nortel), Janet Gunn (CSC), Carolyn Johnson
   (AT&T Labs), Paul Kyzivat (Cisco), Daryl Malas (CableLabs), Tom
   Phelan (Sonus Networks), Jonathan Rosenberg (Cisco), Henning
   Schulzrinne (Columbia University), Robert Sparks (Tekelec), Nick
   Stewart (British Telecommunications plc), Rich Terpstra (Level 3),
   Fangzhe Chang (Bell Labs/Alcatel-Lucent).

Authors' Addresses

   Volker Hilt
   Bell Labs/Alcatel-Lucent
   791 Holmdel-Keyport Rd
   Holmdel, NJ 07733
   USA

   EMail: volker.hilt@alcatel-lucent.com

   Eric Noel
   AT&T Labs

   EMail: eric.noel@att.com

   Charles Shen
   Columbia University

   EMail: charles@cs.columbia.edu

   Ahmed Abdelal
   Sonus Networks

   EMail: aabdelal@sonusnet.com