draft-ietf-soc-overload-design-00

Volker Hilt, Eric Noel,
Charles Shen, Ahmed Abdelal

(volker.hilt@alcatel-lucent.com, eric.noel@att.com,
charles@cs.columbia.edu, aabdelal@sonusnet.com)

SOC Virtual Interim Meeting, June 2010

Motivation

Design considerations and models for an overload control solution for SIP.

- Frame the discussion of an SIP overload control mechanism.
- Describe possible design choices and models.
- **Does not define a solution for SIP overload control.**

Contributors are the members of the SIP overload control design team.

Overload Control

Overload occurs if a SIP server does not have sufficient resources to process all incoming SIP messages.

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.

An overload control mechanism enables a SIP server to perform close to its capacity limit during times of overload.

Implicit/Explicit Overload Control

Explicit Overload Control

An explicit overload signal is used to request a reduction in the incoming load.

Upstream neighbors adjust transmission to a level that is acceptable to the downstream server.

- 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

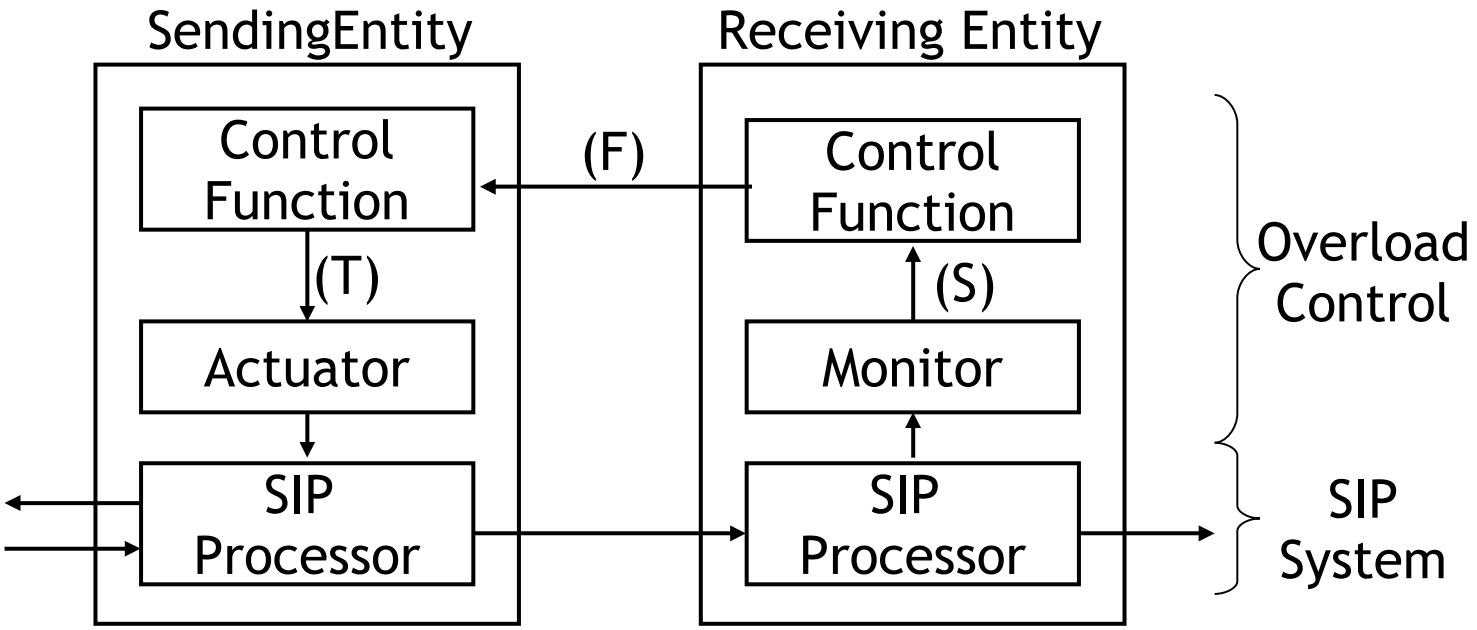
Uses the absence of responses and packet loss as an indication of overload.

A SIP server that is sensing such a condition reduces the load it is forwarding a downstream neighbor.

- Avoids that an overloaded server, which has become unable to generate overload control feedback, will be overwhelmed with requests.

The ideas of explicit and implicit overload control are complementary!

Overload Control Model



Monitor: monitors SIP load and generates load samples (S).

Control Function: implements overload control algorithm that decides when to throttle and to which extent.

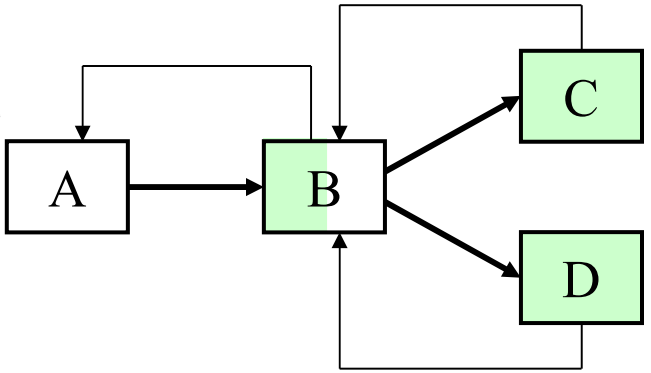
- Uses load samples (S) and generates throttles (T).
- Receiving entity sends load feedback (F) to sending entity.

Actuator: Implements well-defined behavior for throttles (T).

Hop-by-hop vs. end-to-end

Hop-by-hop overload control

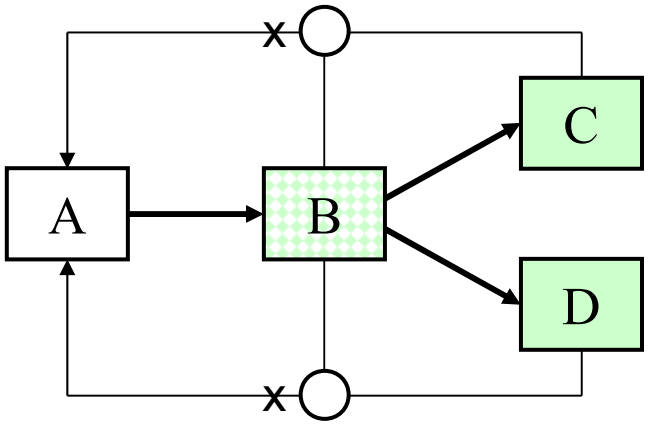
- Server provides overload control feedback to its direct upstream neighbor.
  - No knowledge about routing policies of neighbors needed.
- Neighbor processes feedback and rejects/retries excess requests if needed.



Hop-by-hop

End-to-end overload control

- A single control loop for each source-destination pair.
  - Endpoints need to track load of all servers on all possible paths to a target.
- SIP requests for the same source/destination pair can travel along different paths, depending on policies, services, forwarding rules, forking, load, etc.
  - A SIP proxy often cannot make assumptions about which downstream proxies will be on the path of a SIP request.
- Can be applicable in limited, tightly controlled environments.



End-to-end

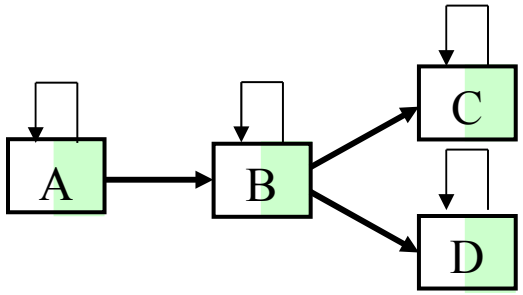
Local Overload Control

Servers locally reject messages that exceed their processing capacity.

- Assumption: rejecting messages is less expensive than processing them and stops retransmissions.
- Fully implemented within a SIP server and does not require cooperation between servers.

Can be used in conjunction with other mechanisms and provides an additional layer of protection.

Local overload control mechanism can act as a mechanism of last resort that is activated if other mechanisms do not provide adequate results.



Local Overload control

Client-to-Server vs. Server-to-Server Communication

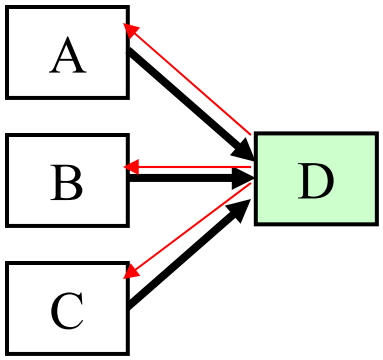
Server-to-Server Communication

A server sends a stream of SIP requests to other servers.

SIP request streams between servers are dynamic.

- Load between servers can be reduced gradually by rejecting/retrying some of the requests.

Overload control can use feedback to request that an upstream server reduces traffic to a desired amount.



Server-to-Server Communication

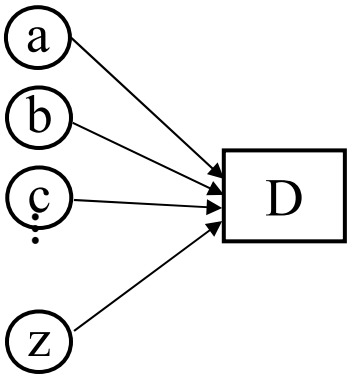
Client-to-Server Communication

UAs typically only initiate a single request at a time.

- A UA can be told to wait a certain time before re-sending the request.

Problem: a large number of UAs can cause overload even if all UAs are told to back-up.

Feedback-based overload control does not prevent overload in the server.



UA-to-Server Communication

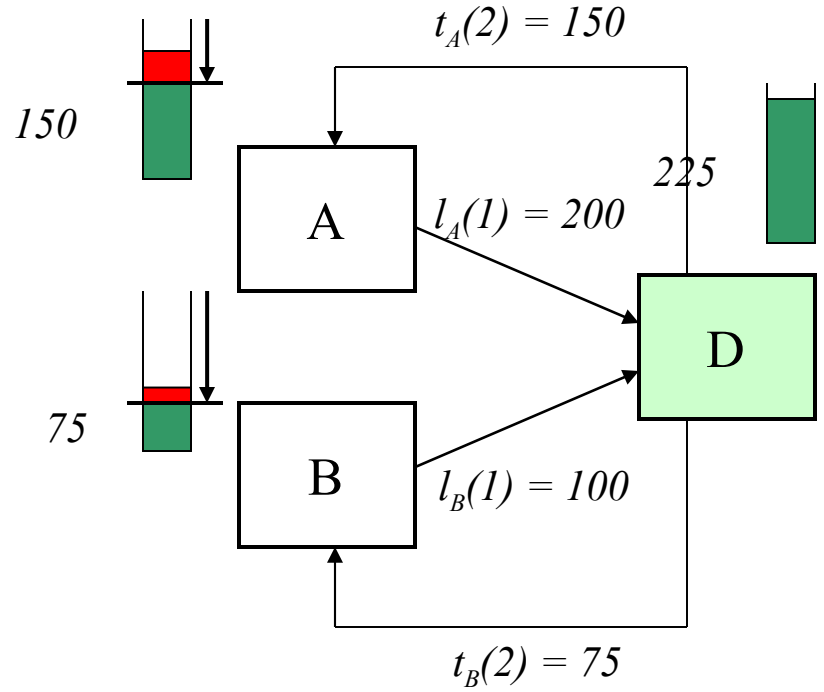
Rate-based Overload Control

Server sets a rate cap of *t* requests per second for each client.

- Rate cap *t* is determined by a control algorithm executed by D.
- Client throttles load, e.g., using request gapping.
- Rate to D never increases beyond the sum of all rate caps.

Requires server to assign a share of its capacity to each upstream neighbor.

- Servers need to track neighbors and adjust shares to server arrivals and departures.
- Shares needs to be large enough to avoid starvation and low enough to protect against overload.



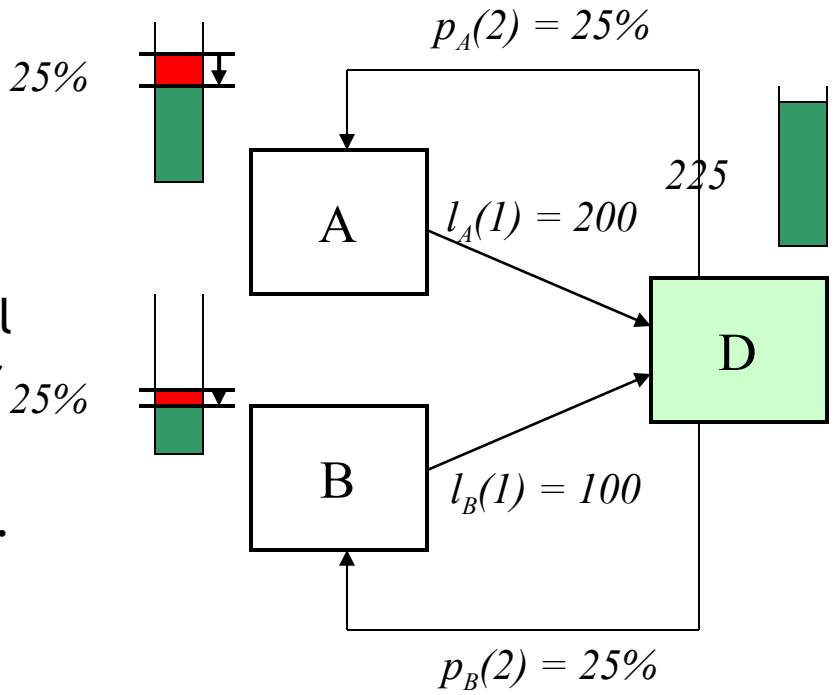
Loss-based Overload Control

Server sets a “loss” rate of p% requests in case of overload.

- Loss rate *p* is determined by a control algorithm executed by D.
- Client throttles load, e.g., by drawing a random number between 0-100.
- Server can send the same loss rate to all neighbors, independent of their number and load contribution.

Loss rate needs to be adjusted if load varies.

- Servers need to adjust loss percentage depending on the incoming load.
- Does not guarantee upper limit of load for D.



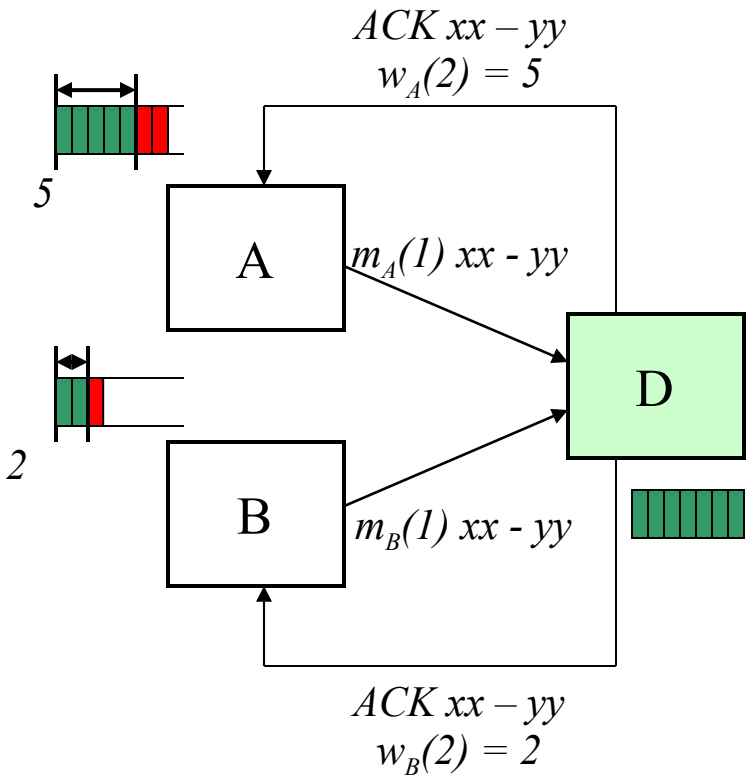
Window-based Overload Control

A client is allowed to send a certain number of requests before it needs to receive a confirmation from the server.

- Implicit throttling since clients stop sending if no feedback is received.
- Ensures that input buffer never overflows if number of clients is constant.

Requires server to assign a share of its input buffer to each neighbor.

- Window size needs to be adjusted to the load contributed by each neighbor and the number of neighbors.
- Once the window size is zero, an out-of-band message is required to restart.



Signal-based Overload Control

Use the transmission of an overload indication (e.g., a 503 (Service Unavailable) response without Retry-After header) as a signal for overload.

- After receiving an indication, the sender reduces the load to the downstream neighbor until no more indications are received.
- A sender which has not received an overload indication 503 for a while starts to increase the offered load until a 503 response is received or it is forwarding at full capacity.
- A possible algorithm for adjusting traffic is additive increase/multiplicative decrease (AIMD).

Message Prioritization

Overload control requires a SIP server to select messages that need to be rejected or redirected in cases of overload.

While the selection is largely a matter of local policy the following general rules should apply:

- Prioritize messages for ongoing transactions over messages for new transactions.
- Preserve high-priority requests (e.g., emergency service requests) possibly as indicated by the Resource-Priority header.
- Prioritize requests for ongoing sessions over requests that set up a new session.

Conclusion

Draft provides a framework for the discussion of SIP overload control mechanisms.

Product of the SIP overload control design team.

Minor update needed:

- Add reference to paper by Ahmed.

Ready for WGLC?