

SIP Overload Control: Where are we today?

Dorgham Sisalem

Abstract Like any other Internet-based service VoIP servers can get overloaded. This overload can result from a denial of service attack, flash crowd scenario or hardware and software problems on the VoIP server itself. To be able to properly react to the overload the VoIP servers will have to monitor the incoming traffic and identify the causes of the overload. Depending on the cause of the overload and the overloaded resource different overload control mechanisms will have to be deployed. Without such mechanisms, VoIP services can fail and can hence not be trusted as replacement to PSTN services. This paper looks into the issues that need to be solved with regard to controlling the load at SIP servers, the different possibilities for controlling the load and some of the still open issues.

1 Introduction and Motivation

VoIP servers whether proxies, application servers or session border controllers constitute the core components of any VoIP infrastructure. These servers are responsible for processing and routing VoIP signaling traffic and supporting various advanced services such as call forwarding or voicemail. In order to ensure a highly available VoIP service, these servers will have to continue performing their tasks even under high load situations.

Overload can occur as the result of a denial of service attack or from software or hardware problems on the VoIP server itself. Like any other Internet-based service VoIP servers can be the target of denial of service attacks. While operators will certainly deploy various mechanisms to prevent such attacks, detecting and preventing DoS attacks is not a trivial task. Attacks can be disguised as legitimate VoIP traffic so distinguishing between a denial of service attack or a sudden surge in traffic due to some event is not always possible. Hence, VoIP servers need to incorporate

Dorgham Sisalem
Sisalem, Tekelec Germany, e-mail: dorgham.sisalem@tekelec.com

mechanisms that monitor the load and the incoming traffic, identify the overloaded resources and cause of the overload and react in a manner that will prevent a complete service interruption. Without such mechanisms, VoIP systems will fail under overload situations. Work done in [6] suggests that overload situations not only reduce the performance of a SIP server but can finally lead to a complete standstill of the entire VoIP service. The unavailability of the VoIP service due to overload would not only be inconvenient to the subscribers but would also tarnish the reputation and the trustworthiness of the provider. Actually with a reputation of being insecure and unreliable, the VoIP technology would suffer as a whole with dramatic economic consequences and will reduce the trustworthiness of VoIP as a replacement of PSTN.

In designing overload control mechanisms for VoIP servers, monitoring and identifying the cause of the overload plays a major role in deciding the system's reaction to the overload situation. On the one hand, when overload is caused for example by a denial of service attack, it would be useless to redirect the incoming traffic to another server, as this would only result in overloading that server as well. On the other hand, redirection would be an appropriate option when the overload situation is caused by software, hardware failures or unbalanced traffic load.

Currently, a lot of the offered VoIP services are based on the session initiation protocol (SIP), [9]. SIP, however, does not offer sufficient mechanisms for handling overload situations. Hence, in this paper, we look at the available solutions, categorize the different approaches and list some of the open issues.

In Sect. 2 we take a brief look at possible causes of overload and the current status of congestion control for SIP. In Sect. 3 the different design aspects of a SIP overload control scheme are discussed. For each point the different alternatives are presented and the pros and cons of each alternative are highlighted.

2 Overload Control in the SIP Specifications

In its simplest form a SIP-based VoIP service consists of user agents (UA), proxies and registrar servers. The UA can be the VoIP application used by the user, e.g., the VoIP phone or software application, a VoIP gateway which enables VoIP users to communicate with users in the public switched network (PSTN) or an application server, e.g., multi-party conferencing server or a voicemail server. User agents keep state information about each active call that they are processing for the entire duration of the call.

The registrar server maintains a location database that binds the users' VoIP addresses to their current IP addresses.

The proxy provides the routing logic of the VoIP service. When a proxy receives SIP requests from user agents or other proxies it also conducts service specific logic, such as checking the user's profile and whether the user is allowed to use the requested services. The proxy then either forwards the request to another proxy or to another user agent or rejects the request by sending a negative response.

A SIP proxy acts in either statefull or stateless mode. In the statefull mode, the proxy forwards incoming requests to their destination and keeps state information about each forwarded request until either a response is received for this request or a timer expires. If the proxy did not receive a response after some time, it will resend the request. In the stateless mode, the proxy would forward the request without maintaining any state information. In this case the user agents would be responsible for retransmitting the request if no responses were received. As the statefull behavior reduces the load on the user agents and provides the service provider with greater session control possibilities, e.g., forwarding the request to another destination if the first one did not reply, statefull SIP proxies are usually used by VoIP providers.

With regard to the SIP messages we distinguish between requests and responses. A request indicate the user's wish to start a session (INVITE request) or terminate a session (BYE request). We further distinguish between session initiating requests and in-dialog requests. The INVITE request used to establish a session between two users is a session initiating request. The BYE sent for terminating this session would be an in-dialog request. Responses can either be final or provisional. Final responses can indicate that a request was successfully received and processed by the destination. Alternatively, a final response can indicate that the request could not be processed by the destination or by some proxy in between or that the session could not be established for some reason. Provisional responses indicate that the session establishment is in progress, e.g, the destination phone is ringing but the user did not pickup the phone yet.

SIP proxies constitute the core components of a VoIP service and will have to handle signaling traffic arriving from thousands if not millions of user agents. In this paper we will therefore concentrate on providing a solution for handling congestion scenarios that might influence the behavior of SIP proxies.

In the context of this paper we will use the term SIP server to indicate any SIP component that is expected to handle many calls in parallel. This includes SIP proxies, application servers, conferencing servers or PSTN gateways. As SIP proxies constitute the core components of a VoIP service and will have to handle signaling traffic arriving from thousands if not millions of user agents more attention and details will however be given to proxies.

A SIP server can become overloaded due to various reasons such as:

- Denial of service (DoS) attack: DoS attacks on a SIP server can take different forms and target either the memory consumption of the server or the CPU -or both [4].
 - Flooding attacks: With these kinds of attacks, an attacker generates a large number of SIP requests. Even if these requests end up being dropped by the server, the server will first have to parse and process them before deciding to either forward, reject or drop them. Depending on the number of generated requests, such attacks can misuse a large portion oft the CPU available to the server and reduce the amount of CPU available for processing valid requests.
 - Memory attacks: This is a more intelligent kind of attack in which the attacker sends valid SIP requests that are forwarded by the server to destinations that

do not answer properly. With each forwarded request the server will maintain some transaction state. If the destination of these requests does not answer at all, then the server will keep on trying for some time, 32 seconds, the so called Timer B in [9], before it can delete the state information. If the destination answers with a provisional response but not with a final one, then the server will have to keep the transaction state for at least 3 minutes, the so called Timer C in [9].

- Flash crowds: Flash crowd scenarios describe a sudden increase in the number of phone calls in the network. An example for this is when thousands of users want to vote on a TV show. This sudden increase in the number of calls will result in an increase in the required CPU and memory at the server.
- Unintended traffic: Software errors or configuration mistakes can cause one or more user agents or SIP proxies to send unintentionally multiples of the amount of the traffic they usually generate. Even though the excess traffic is not generated with malicious intent it is just as useless as DoS traffic and can just as well cause an overload situation.
- Software errors: Software errors include memory leak problems or infinite loops. Memory leak would deplete the available memory in a similar manner as a memory DoS attack. An infinite loop could block the server from serving SIP requests.

The most straightforward approach for avoiding overload is to ensure that the available processing resources of a SIP component are sufficient for handling SIP traffic arriving at the speed of the link connecting this component to the Internet. With modern access lines reaching gigabit speeds, provisioning the VoIP infrastructure of a provider to support such an amount of traffic, which is most likely several times the normal traffic can easily become rather expensive.

The SIP specifications do not provide much guidance on how to react to overload conditions. [9] indicates that a server that is not capable of serving new requests, e.g., because it is overloaded, could reject incoming messages by sending a *503 Service unavailable* response back to the sender of the request. This signals to the sender that it should try forwarding the rejected request to another proxy and not to use the overloaded proxy for some time. Further, the 503 response includes a *Retry-After* header indicating the period of time, during which the overloaded server should not be contacted. While this reduces the load on the overloaded proxy, it results in directing the traffic, which has caused the overload to another proxy, which might then get overloaded itself. Fig. 1 depicts a scenario in which a load balancer distributes the traffic to two different proxies. In the case of a DoS attack it is most likely that all the SIP servers in a SIP cluster will be affected and will be overloaded at the same time. When the first server replies with a 503, the load balancer will forward the traffic destined to that server to the other server. With the additional traffic this server will become overloaded as well and will issue 503 replies. Shifting the traffic from one server to another has only made the situation worse for this server. This shifting of traffic can also lead to an on-off behavior. That is, consider the case when an attacker is generating traffic that is causing both servers to run at 100% of their capacity. When one of them issues a 503 response, the traffic destined to it will

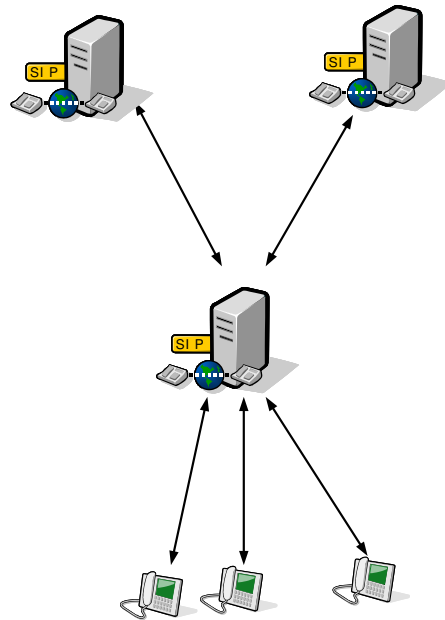


Fig. 1 Load distribution and usage of 503 replies

be forwarded to the other, which will now receive traffic at 200% of its capacity. This server will, hence, issue a 503 response. In case the *Retry-After* value of the first server expires before that of the second server, then that server will suddenly receive traffic at 200% of its capacity and will reject the traffic with another 503. This on-off behavior can actually even lead to a lower average throughput making the 503 approach not optimal for the cases in which a SIP component receives SIP messages from only a small number of other SIP instances. In case a SIP server receives requests from a large number of user agents, then the 503 approach can work much more efficiently as only the user agents that receive the 503 response will try another destination. Further, the on-off behavior would not be observed in this case as spreading out the 503 among the clients has the effect of providing the overloaded SIP instance with more fine-grained controls on the amount of work it receives. Naturally, if the senders are bots that do not respect the *Retry-After* header using 503 will not be sufficient for protecting the server from getting overloaded.

As guidelines for providing mechanisms for solving the overload control issue, in [8] the authors discuss the requirements an overload control scheme should ful-

fill. Among these requirements, is that an overload control scheme should enable a server to support a meaningful throughput under all circumstances, should prevent forwarding traffic to other servers that might be overloaded themselves and should work even if not all servers in the network support it.

Hilt describes in [3] the needed extensions for SIP in order to exchange overload related information between neighboring servers. [2] provides a general overview of the design considerations for overload control schemes and their advantages and some of the issues that need to be considered when designing a new overload control algorithm such as which network topologies to consider and fairness

3 Design Considerations for SIP Overload Mechanisms

When designing a mechanism for protecting a SIP server from overload, the designer needs to answer the following questions:

- **Monitoring of overload indicators:** What information should be monitored and used as the indication of the load status of the server?
- **Algorithm:** Once overload was observed what is the best approach for reducing the overload and what issues should be considered in this context?
- **Reaction:** How should a server reduce the amount of traffic once the overload control algorithm decides that the load must be reduced?
- **Model:** Should the mechanism work in isolation on the server or can some form of cooperation between the SIP servers in the network be assumed? In the standalone model, the SIP server monitors its own overload status and once overload is identified, an overload control algorithm is executed and the server reacts to the excess traffic. In the cooperative model, once overload is identified, the SIP server informs its neighbors about the overload status and requests them to reduce the amount of traffic they are sending to it.

3.1 *Overload Monitoring*

A system is considered to be overloaded if some pre-defined thresholds are exceeded. These thresholds might relate to either the resources used by the system for conducting its task or some values that define the capabilities of the system. By monitoring these resources a system can decide whether some reaction is needed or not. The reaction to the overload will depend on the cause of the overload. The reasons for overload can be either natural such as flash crowds or due to the failures of some servers in a cluster, accidental as is the case with software or configuration errors or malicious as is the case with DoS attacks. Hence, as part of the monitoring process, the servers will also need to determine the type and nature of the incoming traffic.

A simple value that defines the capability of a system is the number of supported active calls. As each SIP system can maximally support a predefined number of active calls, by monitoring the number of parallel calls a server can easily assess its status.

The authors in [11] present a number of overload control mechanisms that use the number of active calls in the system as the overload indication. The number of active calls is determined as the difference between counted INVITE and BYE requests over a measurement interval. The results shown in [11] suggest that the discussed control mechanisms can keep the SIP systems running at their maximum capacity. However, the general applicability of the presented mechanisms requires more investigation. Determining the maximum number of supported calls at a SIP system is not a trivial task as this will depend on the call model -calls conducted according to the IMS SIP model [1] use more messages than plain SIP as defined in [9]-, the call scenario and sizes of the SIP messages and so on. Besides the difficulty in estimating the maximum capacity of a system, it is unclear how the system will react when it is attacked. A DoS attack with a high number of INVITE requests that are never terminated will mean that the number of active calls in the system will be rather high but the system will not be really overloaded as it does not have to serve these calls after processing the initial INVITE. Further, a DoS attack consisting of a flood of in-dialog requests will consume resources but will not be counted for as part of the active dialogs. Hence, when monitoring the number of active calls, the server will need to distinguish between successful and failed calls, calls that were actually completed and ones that just consume resources and are never completed and messages that are not part of a running call or do not belong to any active call.

In order to be able to receive and process a SIP request, a SIP server requires three types of resources:

- **Network Buffer:** Before handing a request to the process that is running the SIP logic at the server messages are queued in the UDP/TCP receive buffers. When a system is overloaded, i.e., the rate at which SIP messages arrive is higher than the rate at which the SIP process reads out messages from the buffer, the buffer will fill up and the incoming messages will be dropped. Setting the buffer size to a too small value could lead to the loss of messages during bursts of traffic. Setting the buffer to a too large value, in order to compensate for the effects of bursts for example, could, however, mean that some messages will not be processed in time and the sender will retransmit the request.
- **Memory:** A SIP server needs to copy each incoming request in its internal buffers to be able to process the message. The amount of buffered data and the time period the server is supposed to keep the buffered data varies depending on whether the server is working in a stateful or stateless mode. In any case, the server will at least need to maintain the buffered data while contacting another entity such as a DNS server or a database for example. Depending on the message type, the number of Via headers and the body of the message, the size of a SIP message might range from a few hundreds of bytes up to a few thousands. The buffering duration will also depend on the message type and might range from a few seconds to a few minutes.

- CPU: After receiving a SIP message, the SIP server needs to parse the message, do some processing and forward the message. Depending on the content and type of the message and server policies the actual amount of CPU resources might vary.

Monitoring the buffer size as an indication of the overload is problematic. This will require frequent reading of the buffer size. This is acceptable when UDP is used as the transport protocol as there would be only one receive buffer. When TCP is used, the SIP system might need to query the receive buffer of thousands of TCP connection which will consume a considerable amount of the CPU resources of the SIP system. Hence, schemes designed for overload control for SIP over TCP that use the buffer occupancy as an indication of overload already note that such schemes are only applicable for scenarios in which the number of communicating components is limited to a few, see [10]. Further, the socket receive buffers tend to be either full or empty. That is, while the system can serve the incoming requests the receive buffer will be empty. Once the SIP system reaches its capacity it will no longer be able to serve all the incoming requests and the receive buffer will be filled up more or less instantly. Therefore, the SIP system should react to overload before the receive buffer becomes full as by then it would be already too late. Finally, the SIP system has no knowledge of content of the buffer. Hence, it does not know whether the queued messages are responses, INVITE requests or other requests and cannot decide on the appropriate reaction.

The authors in [12] monitor the memory usage of the SIP process and use the memory occupation level as the indication of overload. The memory used by the SIP process itself can be seen as an extension of the receiver buffer memory except that there is more of it and the SIP process knows the content of saved information. This knowledge can enable the SIP system to identify what kind of calls and transactions are active and whether the maximum limit will be reached soon. However, setting an appropriate value for the maximum memory to support is just as problematic as determining the number of maximum active calls as different types of requests require different amounts of memory.

[12] also discuss the possibility of using an averaged value of the CPU usage as an indication of overload. That is, reaction to overload is triggered once the CPU usage value reaches a certain threshold. This approach has the advantage that it does not really need to know the maximum call processing capacity of the SIP system as is the case when using memory or number of active calls. However, CPU usage values can be rather oscillatory in behavior. Most used SIP systems use multi tasking operating systems in which the CPU is used by multiple processes. Hence, a sudden increase in the CPU usage value could imply an increase in the SIP processing but could also stem from a background process doing some maintenance tasks. This can lead to an oscillatory overload control behavior.

3.2 *Overload Control Algorithms*

Designing congestion control algorithms was and still is one of the major PhD topics in the area of networking and communication. Over the past forty years or so hundreds if not thousands of papers were published describing all kinds of overload control schemes. While these algorithms differ in their details, they all share the same concept: identify some value as the overload indication and react to the congestion in some manner.

In theory, most of the overload control schemes published for all kinds of IP-based service could be used for SIP servers as well. This is actually the approach taken by a lot of the papers discussed here. However, congestion control schemes usually assume that all incoming events require the same processing resources. Thereby, reducing the number of processed events by a certain value will release occupied system resources by a predictable value. This is, however, not so straightforward with SIP as different SIP messages and events can require different resources. For example:

- Requests vs. responses: Successfully processing a response means that the resources that were occupied by a transaction can be freed. On the other hand accepting a new request will occupy some new resources. Hence, processing of responses should have a higher priority than the processing of requests.
- Call initiation vs. in-dialog requests: Accepting a new call means that the server will have to also be ready for all other in-dialog requests that will come as part of this new call, e.g., BYE or updates of the session. On the other hand, accepting a BYE would mean that a call can be successfully terminated and the resources blocked by the call can be freed. From a user perspective, a successful session termination is also more important than a successful session initiation. As a BYE request usually indicates to the billing system that it should stop accounting a certain call, losing a BYE could lead to open-ended call records and angry customers.
- IMS vs. IETF call models: The call models defined for IETF based SIP are simpler than those defined for IMS calls and require fewer messages. If a server is handling both types of calls then accepting a call that is generated by an IMS subscriber will result in overall higher load than if a SIP call was accepted.
- Call priority: Similar to the PSTN, a call by some governmental agencies or an emergency call must have higher priority than normal calls. There can also be a distinction between business subscribers and others.
- User profile: The amount of resources that will be consumed by an incoming call might differ depending on the complexity of the user's profile, e.g., whether call forwarding or call forking for example is set-up.

Thereby, when designing a congestion control scheme for SIP there are various levels of priorities that should be taken into account. The type and importance of these priorities will depend very much on the server type and deployment scenario. An optimal overload control scheme would enable a server to achieve a maximal throughput by first rejecting or dropping the events with the lowest priority and

highest resource consumption. However, this can only be achieved by first parsing the incoming messages which already consumes some resources. Hence, some balance must be found between spending resources for parsing incoming messages and risking the loss of high priority events, which in the worst case can lead to the loss of further resources. For example, blindly rejecting a BYE for a certain call would mean that the server would have to maintain all resources that were blocked by the call.

3.3 Reaction to Overload

As discussed in Sect. 3.1, a SIP proxy is said to be overloaded if one or more of its resources is having a value above some maximal limits. Going above these limits can destabilize the system and even lead to complete failure, see [5]. Hence, the goal of any overload control scheme is to reduce the usage of these resources. Reducing the load on the SIP server can be realized in one of the following ways:

- **Drop excess traffic:** In this case all the available resources are used for processing incoming traffic and any requests that can not be processed are dropped. As SIP user agents retransmit requests for which they did not receive a response, dropping requests will actually even lead to more traffic in the network. That is, when a SIP client using UDP does not receive a reply after a period of time (T_1) it retransmits the request. If no answer was received after $2T_1$ seconds then the request is retransmitted again and so on. Thereby a dropped request can cause the retransmission of up to 10 requests. To demonstrate the effects of dropping requests, we conducted a test in which a SIP proxy was overloaded by sending more traffic than it can process. The proxy had a processing rate of 1 request per second and we generated a continuous stream of requests at rates of 1500, 2000 and 2500 requests. All requests arriving on top of the proxy's processing rate were dropped. The results presented in Fig. 2 show that even though the requests are generated at a rate of only 1500 requests per second, due to the retransmissions the proxy will end up having to deal with 10000 requests per second.
- **Reject excess traffic with 503:** When a server is overloaded it can reject incoming requests with 503 to indicate when the sender can retry using this server. The overloaded server would then not be used by the sender for the retry period indicated in the 503 message. As this can lead to an oscillatory behavior as described in Sect. 2 this should only be used for rejecting traffic from end systems and not from other SIP servers.
- **Reject excess traffic with 5xx:** Instead of dropping excess traffic or requesting not to receive any traffic for some period of time, an overloaded server can reject incoming requests by sending a 500 response for example. This is based on the assumption that rejecting a request is less costly in terms of processing resources than accepting and forwarding it. To test this assumption we ran a number of measurements in which INVITE requests were sent to a SIP server. The SIP server was configured to either forward, reject or drop all incoming requests.

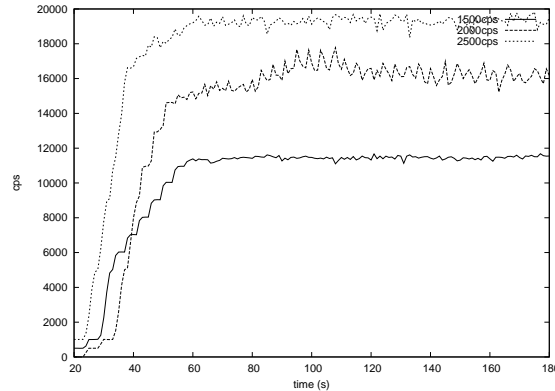


Fig. 2 Effects of retransmission on amplification of traffic

Fig. 3 depicts the CPU resources used in each case. From Fig. 3 we can observe that the amount of resources used for forwarding requests is much higher than that for rejecting them. INVITE messages that are successfully forwarded will be followed by another two requests at least, namely ACK and BYE. Thereby, rejecting an INVITE message will not only save the resources needed for processing the INVITE message itself but would also save the resources that would have been needed for processing other in-dialog requests as well. Rejecting a request is also more resource intensive than just dropping it as it requires processing of the request first. However, when considering that the amount of traffic generated through the retransmissions it would still be preferable.

- **Explicit congestion information:** In this case the overloaded server would actually indicate in its replies its overload status and indicate either the explicit transmission rate the sender should be taking or by which percentage the sender should reduce its transmission rate. The authors in [3] describe different possibilities for exchanging information between neighboring SIP systems. These include adding the information in a SIP header or using the SUBSCRIBE/NOTIFY mechanism of SIP [7]

Sending explicit congestion information does not reduce the load at the SIP system by itself. Hence, this approach assumes a cooperative model in which the receivers of the congestion information use this information to actively reduce the amount of traffic they send to the overloaded system. Rejecting or dropping traffic reduces the load at the SIP system as it decreases the number of active calls the system has to deal with. By monitoring the number of failed or rejected calls a SIP server can also have some indication of the load status of its neighbors, see . While not as accurate as the explicit congestion information it can be helpful in reducing the

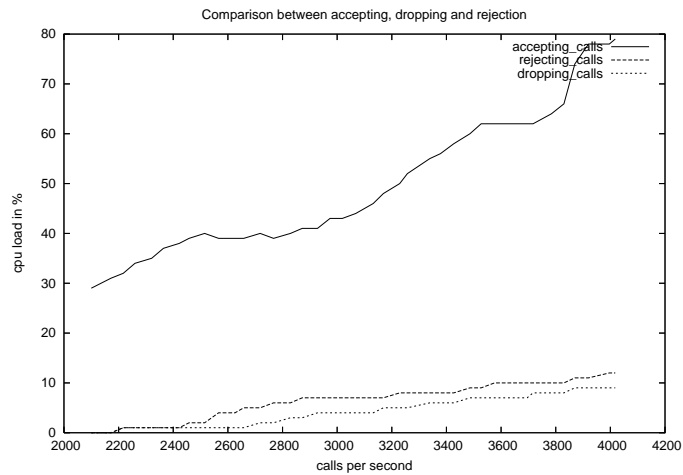


Fig. 3 CPU costs of forwarding, rejecting and dropping requests

overload in the network and does not require the sending and receiving SIP systems to agree on the format used for exchanging congestion information [13].

3.4 Standalone or Cooperative Control Model

In general one can distinguish between standalone and cooperative overload mechanisms. In the standalone approach, an overload control mechanism is implemented at the SIP server to be protected. This server monitors its own resources, e.g., memory, CPU, buffer and capacity. Based on the monitored resources the server will recognize when it starts to become overloaded and will have to deal with the incoming traffic by either rejecting new calls or dropping them.

The other approach for overload control is more of a cooperative process. In this scenario the overloaded server would regulate the amount of traffic it is receiving from its neighbors by informing them about its current load. The neighboring servers will then adapt the amount of traffic that they are sending to the overloaded server. In case they have to reduce the amount of traffic they want to send to the overloaded server then they could also inform their neighbors to send less traffic to them.

Both approaches have their pros and cons. The standalone approach can be deployed without having to rely on other SIP components in the network to also support overload handling. It also does not require the standardization of how to exchange status information. This makes this approach the ideal one to start with.

However, such a mechanism does not cause the overall load of the SIP network to go down and could lead to situations in which the overloaded SIP system will use all of its resources to just reject or drop incoming requests and will stop serving any calls.

The cooperative approach can adapt the number of calls in the network to the actual available resources and could push the overload to the borders of the network. In this way, excess calls will be prevented from even reaching the overloaded servers and access points can consider using non-overloaded paths for establishing the calls. On the negative side, a server that would ignore the feedback information would still cause overload and packet drops. Actually not deploying the overload control mechanism would give the non-conforming system an unfair advantage.

Another issue with the cooperative approach, is that the neighbors will need to have the means to reduce the amount of traffic they are sending to the overloaded servers. To achieve this they will either have to drop or reject the traffic themselves by using some standalone overload control scheme or by informing their neighbors to reduce the traffic that is destined to the overloaded server. Informing the neighbors

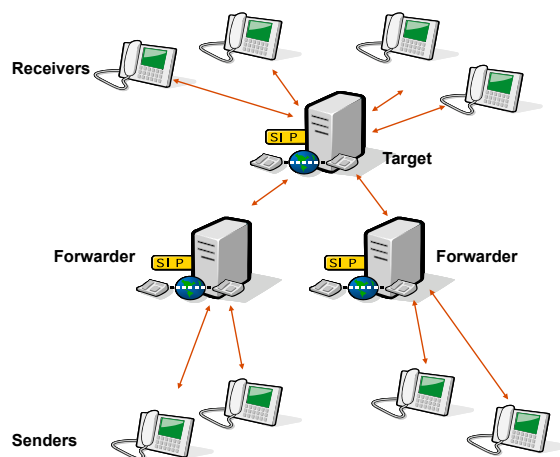


Fig. 4 Multi hop communication

to reduce the amount of sent traffic is theoretically more favorable as it would push the excess traffic even further from the overloaded server. However, SIP servers do not usually keep track of how much of their traffic is traversing some server. Assume server C in Fig.4 is overloaded and that it informs server B to reduce the amount of traffic it is sending. Server B can reduce the amount of traffic sent to server C by either using a standalone overload control scheme by informing server A to reduce the traffic sent from A to C. However, this would mean that server B needs to keep track of which upstream neighbor is sending how much traffic to each downstream

neighbor. While this already makes the logic of a SIP proxy more complex it is still not sufficient. Consider the scenario in Fig. 4 . Server A is sending traffic to servers C and D through server B. When Server C gets overloaded it asks server B to reduce its transmission rate by X. Assume X was an absolute number, i.e., say 10 calls. Server B will in its turn ask server A to also reduce its transmission rate by X, i.e., 10 calls/sec. Now, server A was sending 20 calls/sec to server B with half of the calls going to server C and the other half going to server D. If A reduced its transmission rate by 10 calls/sec then the actual load reduction on server C will only be 5 calls/sec. Now consider X was a percentage value, i.e., server C asks server B to reduce its transmission rate by 10%. When server B asks server A to reduce its transmission rate by 10%, the rate of calls arriving at server C will be reduced by the needed amount. However, server A will also reduce the amount of calls arriving at server D by 10%. Hence to actually achieve the needed reduction at server C, server A will need to keep track of which calls actually traverse server C. This is only readily available if servers A and C are end systems, i.e., voicemail servers, PSTN gateways, conferencing server or a B2BUA.

So in short, the cooperative model is only effective in pushing the overload one hop upstream. Pushing the traffic further upstream is only possible in certain scenarios and could make the SIP components rather complex otherwise.

Hence, to be on the safe side a SIP server should implement a combination of both standalone and cooperative approaches. The SIP system will inform its neighbors about its overload situation. Monitoring the neighbors and assigning a certain share to them is also necessary in order to maintain fairness between the neighbors. Simply asking all neighbors to reduce their load by a certain percentage for example might be the simplest kind of explicit notification information to use but will mean that all neighbors will reduce the amount of traffic they are sending to the overloaded system. This would be unfair towards neighbors who are not sending a lot of traffic to the overloaded systems and are hence not the cause of the overload.

4 Summary

Handling overload at SIP servers is an issue that is being discussed and researched both in academia and standardization bodies. This paper looks at the main ingredients of overload control algorithms and discusses the major pros and cons of the currently investigated schemes in the literature. With first discussions and publications already more than four years old the topic is no longer fresh. However, till now there has been no single solution that provides a proper answer to all the different facets and scenarios of overload control. It is actually questionable if one single approach can solve all related issues. Algorithms dealing with the communication between user agents and proxies will be different than those dealing with the communication between proxies. Overload control when UDP is used as the transport protocol will often look differently from the cases when TCP or SCTP is used. While standardization bodies can provide the proper mechanisms for enabling a cooperative kind

of overload handling, the logic used at different SIP components will most likely be different depending on the type of the component, e.g., proxy vs. application server, the deployment scenario and the vendor's preferences. Further, most of the approaches in the literature assume that overload was caused by non-malicious reasons, e.g., flash crowds. The design of the overload control scheme should however include monitoring mechanisms that enable it to distinguish between malicious and non-malicious overload reasons. Without such distinction both malicious and non-malicious users will be punished in the same manner which will lead to unsatisfied subscribers.

References

1. Signalling flows for the ip multimedia call control based on session initiation protocol (SIP) and session description protocol (sdp). Technical specification group core network and terminals, 3rd Generation Partnership Project (2007)
2. Hilt, V., Noel, E., Shen, C., Abdelal, A.: Design considerations for session initiation protocol (SIP) overload control. Internet Draft, Internet Engineering Task Force (2010). Work in progress
3. Hilt, V., Schulzrinne, H.: Session initiation protocol (SIP) overload control. Internet Draft, Internet Engineering Task Force (2010). Work in progress
4. Kuthan, J., Ehlert, S., Sisalem, D.: Denial of service attacks targeting a SIP VoIP infrastructure - attack scenarios and prevention mechanisms. 'IEEE Networks Magazine **20**(5) (2006)
5. Nahum, E., Tracey, J., Wright, C.: Evaluating SIP server performance. Research report RC24183, IBM T. J. Watson Research Center (2007)
6. Ohta, M.: Simulation study of SIP signaling in an overload condition. In: M.H. Hamza (ed.) Communications, Internet, and Information Technology, pp. 321–326. IASTED/ACTA Press (2004)
7. Roach, A.B.: Session Initiation Protocol (SIP)-Specific Event Notification. RFC 3265 (Proposed Standard) (2002). URL <http://www.ietf.org/rfc/rfc3265.txt>
8. Rosenberg, J.: Requirements for Management of Overload in the Session Initiation Protocol. RFC 5390 (Proposed Standard) (2008). URL <http://www.ietf.org/rfc/rfc5390.txt>
9. Rosenberg, J., Schulzrinne, H., Camarillo, G., Johnston, A., Peterson, J., Sparks, R., Handley, M., Schooler, E.: SIP: Session Initiation Protocol. RFC 3261 (Proposed Standard) (2002). URL <http://www.ietf.org/rfc/rfc3261.txt>. Updated by RFCs 3265, 3853, 4320, 4916
10. Charles Shen, Henning Schulzrinne: On TCP-based SIP Server Overload Control. In: IPT-Comm 2010 (2010)
11. Shen, C., Schulzrinne, H., , Nahum, E.: SIP server overload control: Design and evaluation. In: Conference on Principles, Systems and Applications of IP Telecommunications (IPT-COMM08. Heidelberg, Germany (2008)
12. Sisalem, D., Floroiu, J.: Protecting voip services against DoS using overload control. In: NorSec 2008. Technical University of Denmark (2009)
13. Sisalem, D., Floroiu, J.: Voip overload, a senders burden. In: CCN10. Orlando, USA (2010)