

Protecting VoIP Services Against DoS Using Overload Control

Dorgham Sisalem
Tekelec
Berlin, Germany
dorgham.sisalem@tekelec.com

John Floroiu
Tekelec
Berlin, Germany
john.floroiu@tekelec.com

Abstract—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 mechanisms that would deal with the attack in a manner that would not lead to a complete service interruption. In this paper, we propose a new approach called the Receiver-based SIP Overload Control Algorithm (R-SOCA) which aims at stabilizing the behavior of a VoIP server during overload situations and preventing a complete collapse of the service. R-SOCA is designed so that it takes the cause of the overload as well the nature of the overloaded resource into consideration. R-SOCA was implemented in a SIP proxy and tested under various conditions. The test results show a significant improvement in the performance of a SIP proxy under overload situations and the ability of this algorithm to stabilize the performance of SIP proxies even under a denial of service attack.

I. INTRODUCTION AND MOTIVATION

VoIP servers 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. From a technical point of view, a Voice over IP service resembles to a greater extent an Email service than a traditional telecommunication service. VoIP components use the same physical infrastructure as Email services whether fixed or wireless, the same transport protocols, e.g., TCP/IP and very often the same operating systems and hardware, e.g. PCs. Hence, it is only natural to expect that a VoIP system can suffer from the same security threats as an Email system or any other system in the Internet.

The goal of a denial of service (DoS) attack is to prevent legitimate users from using the attacked, e.g. victim, VoIP servers and hence prevent them from initiating or receiving calls. This can be achieved by flooding a VoIP server with a high number of useless requests that would deplete one or more resources of the host such as bandwidth, memory or CPU. Thereby, the VoIP server would be overloaded and would not have sufficient resources left for serving legitimate

requests. Work done in [1] suggests that overload situations not only reduce the performance of a VoIP server but can finally lead to a complete standstill of the complete VoIP service. To withstand sudden increases in traffic or temporary lack of resources, VoIP servers need to implement overload control mechanisms that aim at reducing the work load of these servers and prevent a complete depletion of their resources. In designing the overload control mechanisms for VoIP servers, the cause of the overload plays a major rule 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, most VoIP services are based on the session initiation protocol (SIP), [2]. SIP, however, does not offer sufficient mechanisms for handling overload situations. Hence, in this paper, we describe a new overload control scheme designed to be used by SIP-based VoIP servers. The scheme called Receiver-based SIP Overload Control Algorithm (R-SOCA) aims at reducing the load on VoIP servers by shedding traffic proportionally to the measured overload status. R-SOCA is designed so that it takes the cause of the overload as well the nature of the overloaded resource into consideration. R-SOCA is designed to improve the performance of the SIP proxies without the need to standardize new features of SIP or to realize some cooperation or feedback between SIP proxies.

In Sec. II we take a brief look at possible causes of overload and the current status of congestion control for SIP. R-SOCA is then presented in Sec. III. To test the performance of R-SOCA, R-SOCA was implemented as part of a SIP proxy. The test results are described in Sec. IV. In our performance investigation we investigate the capability of R-SOCA to stabilize the behavior of a SIP proxy under various traffic loads and models. Sec. V finally summarizes the paper and describes the open issues that will need to be solved in future work. Note that in the context of this paper only SIP proxies will be mentioned. However, the proposed solution applies

equally well to all other kinds of SIP components such as PSTN gateways, back-2-back user agents or softswitches.

II. BACKGROUND AND RELATED WORK

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.

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 reply.

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 reply is received for this request or a timer expires. If the proxy did not receive a reply 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 replies 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 replies. 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. Replies can either be final or provisional. Final replies can indicate that a request was successfully received and processed by the destination. Alternatively, a final reply 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.

From the brief description of SIP-based VoIP services, it is obvious that there are two main resources that might get overloaded at a SIP proxy, namely, CPU and memory. CPU resources are used for parsing incoming messages and executing service specific tasks which might include writing logging information, reading from a database or evaluating a user's profile for example. When acting in transaction statefull mode, memory will be consumed at the SIP proxy for maintaining various transaction state information related to the request. This memory will be dedicated for this transaction until either a final reply was received for this request or some timer expires. [2] specifies that a proxy is supposed to keep these state information for several seconds or even a few minutes depending on the exchanged requests and replies. Besides these two main resources several other resources are essential to the proper working of a SIP proxy. This includes the number of free processes or threads in a multi-process implementation, see [3], the number of busy ports and disk space. Due to space limitations we will not consider these resources in this paper.

CPU and memory could get overloaded at a SIP proxy due to various reasons such as:

- Denial of service (DoS) attack: DoS attacks on a SIP proxy 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 proxy, the proxy 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 of the CPU available to the proxy 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 proxy to destinations that do not answer properly. With each forwarded request the proxy will maintain some transaction state. If the destination of these requests does not answer at all, then the proxy will keep on trying for some time, 32 seconds, the so called Timer B in [2], before it can delete the state information. If the destination answers with a provisional response but not with a final one, then the proxy will have to keep the transaction state for at least 3 minutes, the so called Timer C in [2].
- 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 proxy from serving SIP requests.

The session initiation protocol (SIP) is specified in the RFC 3261 [2]. While this specification describes the behavior of a SIP proxy in general, it does not provide much guidance on how to react to overload conditions. [2] 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) reply back to the sender of the request. Additionally, the 503 reply includes a retry-after header which indicates to the sender of the request when it can try to resend the request to this server. This would signal the sender to try forwarding the rejected request to another proxy and not to use the overloaded proxy for the retry-after time. In case the overloaded server is contacted by a lot of users then using different values of retry-after would ensure that the traffic arriving at the proxy would only increase gradually. However, in a lot of deployment scenarios a proxy is contacted only by a few other proxies or gateways. In this case using the 503 approach would result in a stop and go traffic behavior at the overloaded proxy which would lead to an oscillative and instable over all network behavior. Also directing the traffic to other proxies would usually cause these proxies to become overloaded themselves. Hence, using 503 is more likely to relief the overloaded proxy only when it is facing a lot of user agents and in cases of internal errors, e.g., hardware or software problem, that would prevent the proxy from performing in the expected manner.

In [5] the author discusses the drawbacks of using the 503 reply for indicating overload and the requirements an overload control scheme should fulfill. 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 [6] an approach for handling overload based on having the overloaded proxy exchanging load information with its neighboring proxies. The neighboring proxies

would then adjust the amount of traffic they send to the overloaded proxy based on this information. While this approach seems to be promising, no simulative or measured results are reported for this work. Further, this approach requires the definition of new headers and replies, making it usable mainly between proxies that implement this solution.

[7] describes an approach for avoiding overload by separating different types of SIP messages into different queues and allocating specific shares of the CPU to the different queues. This approach can help in preventing overload situations when for example the overload is caused by a certain type of messages. However, this approach does not help against memory attacks or flooding attacks that use different kinds of SIP requests. In a related manner, in [8] the author also discusses the possibility of using different queues to prioritize certain messages. The authors in [9] presented initial results comparing a SIP network without overload control, with the built-in SIP overload control and with a rate-based overload control scheme. However, their paper does not discuss issues of denial of service or resource specific overload handling mechanisms.

Using simulations, the authors in [10] compare between a number of window and rate based overload control mechanisms based on the feedback approach. While the results are very promising the applicability of the presented solutions for TCP is not discussed. Also these schemes require cooperating proxies and are not optimized to deal with denial of service attacks or to protect the server's memory. Similar to some of the approaches discussed above R-SOCA is also designed to prioritize certain traffic in case of overload. However, in addition, R-SOCA takes into consideration the causes of the overload, e.g., DoS attack or flash crowd, in its reaction to overload. R-SOCA further does not require cooperating neighbors so it could be introduced without requiring other servers to support overload mechanisms. Finally, R-SOCA can be applied to servers using TCP as well as UDP and to protect all kinds of server resources.

III. RECEIVER-BASED SIP OVERLOAD CONTROL ALGORITHM (R-SOCA)

As discussed in Sec. II, 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 [11]. Hence, the goal of any overload control scheme is to reduce the load on the resources. Reducing the load on the SIP proxy can be realized by either dropping incoming requests or rejecting them, e.g., sending back a 5xx reply. However, both options have their costs in terms of CPU usage as well. In both cases, the SIP proxy will have to receive the request and either drop it or generate a reply and send it back. Fig. 1 depicts the results of a simple measurement scenario in which a SIP proxy either forwards all received requests, rejects all received requests

with a 500 reply or drops all received requests. The results depicted in Fig. 1 suggest that dropping incoming requests, consumes slightly less CPU at the SIP proxy than rejecting them. However, dropping packets in SIP has its negative side effects. A sender that does not receive a reply to one of its requests will resend this request for a number of times. Hence, dropping requests will actually lead to an increase in the number of arriving requests at the SIP proxy. Considering that dropping one request could result in the sending of 6 or more retransmissions of the request, and that the CPU resources required for rejecting a request are only 20% higher than for dropping a request, the dropping approach will actually be more costly in terms of CPU usage at the end. Further, a sender that does not get a reply to its requests is likely to search for another proxy and resend the request to that proxy. This will finally lead to overloading other proxies as well. Therefore, R-SOCA uses the rejection approach as the main approach for reducing the load on the SIP proxy.

From Fig. 1 we can also observe that the amount of resources used for forwarding requests is much higher than that for rejecting them. For conducting the measurement, INVITE messages were sent to a SIP proxy. 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.

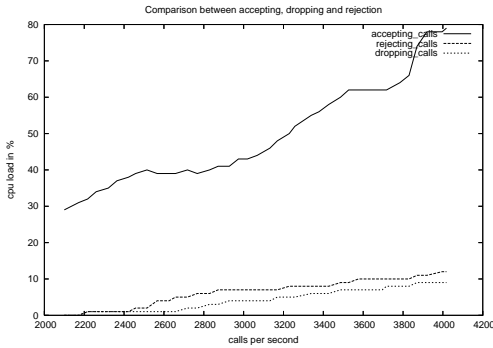


Fig. 1. CPU costs of forwarding, rejecting and dropping requests

Besides aiming at reducing the load on the overloaded resources, R-SOCA was designed with the following additional goals in mind:

- 1) Keep the amount of used resources at the SIP proxy at predictable levels.
- 2) While the stability of the SIP proxy is not endangered, aim at serving the engineered load. Engineered load (E) is defined as the number of requests the system should

be able to deal with. This includes the number of session initiating INVITE messages (E_I), number of in-dialog requests (E_D), e.g., the requests that are exchanged as part of a dialog such as BYE and re-INVITE messages, number of REGISTER messages (E_R) and the number of out-of dialog messages such as INFO, OPTIONS or MESSAGE for example (E_O). The values for the engineered traffic are defined based on the experience of the manufacturer of the SIP proxy and the expected usage scenario.

- 3) Prioritize running sessions. This is based on the assumption that a user would be more annoyed if his call gets interrupted or can't be terminated and he has to pay more because his BYE request was rejected by the SIP proxy than if he could not make a call in the first place.
- 4) Identify traffic that might be part of a DoS attack and penalize it with a higher rate than other traffic
- 5) A SIP proxy implementing R-SOCA will interoperate seamlessly with other SIP entities that have no knowledge of R-SOCA.

A. Probabilistic Message Rejection

There are two main resources that need to be protected from getting overloaded: memory and CPU. R-SOCA defines for each resource (R), e.g., memory and CPU, two thresholds; a minor (R_{min}) and a major threshold (R_{max}). Once the minor threshold is exceeded, R-SOCA starts gradually reducing the load on the SIP proxy by rejecting incoming requests with the rejection probability of P_T^R . Once the maximum threshold is exceeded, all incoming requests are rejected. The minimum threshold (R_{min}) is set to the amount of resources required to serve the so called engineered traffic (E). The engineered traffic, is the amount of calls a SIP proxy is expected to receive and process successfully under normal operational conditions, e.g., no DoS attacks or flash crowd scenarios. The maximum threshold (R_{max}) is set to the amount of resources required to handle some worst case scenarios such as flash crowds.

The major characteristics of R-SOCA can be described as follows:

- At fixed time intervals T the current load status R_T for each resource R is measured and its rejection probability (P_T^R) is determined.

$$P_T^R = \begin{cases} 0 & R_T \leq R_{min} \\ \frac{R_T - R_{min}}{R_{max} - R_{min}} & R_{min} < R_T < R_{max} \\ 1 & R_T \geq R_{max} \end{cases} \quad (1)$$

- The rejection probability to be used by the SIP proxy (P_T) is then determined as the maximum value of all P_T^R . To avoid rapid changes in the rejection probability, an averaged rejection probability (\hat{P}_T) is determined as a weighted moving average over time with

$$\hat{P}_T = \hat{P}_{T-1} \times (1 - \sigma) + P_T \times \sigma \quad (2)$$

For smoothening the rejection probability a σ of 0.6 was used. This value proved after a couple of measurements as the best choice between rapid reaction to sudden changes in the traffic and smooth changes of the rejection value. As the rejection probabilities are determined as moving averages, they will not converge to 0 or 1. Hence, \hat{P}_t is used as follows:

- If $\hat{P}_t \leq 0.05$, then the system is not considered overloaded and no incoming requests will be rejected
- With $0.05 < \hat{P}_t$, then the system is considered overloaded
- If any P_T^R had a value of 1 then the system is considered severely overloaded and \hat{P}_T is set to 1.
- when the system is in overload status INVITE requests and out of dialog requests such as REGISTER and MESSAGE will be rejected with probability of \hat{P}_T . To ensure that emergency calls are processed even under overload situations, emergency requests should not be rejected. In-dialog requests are not rejected.
- when the system is considered severely overloaded, e.g. \hat{P}_T set to 1, then all incoming requests will be rejected

The rejection is achieved by sending a 500 reply including a retry-after header indicating that the user agent should resend the request after T_{retry} seconds. The 500 replies are sent in stateless manner. That is, no state information is maintained for them and the negative reply is not retransmitted if no ACK was received. In case the 500 reply is lost the sender would resend the INVITE request. In case the sender receives the 500 reply it will send an ACK request. As the proxy will not find an appropriate transaction state for the ACK it can just drop it.

B. Denial of Service Protection

Now, it is naturally possible that a DoS attack would consist of sending large amounts of requests that look like in-dialog requests such as BYE messages for example. In such a case, the above mentioned steps would not lead to a reduction of the overload. To accommodate this scenario, for each type of in-dialog requests, the number of such requests (D) is counted and compared to the expected number of requests of this type (D_{en}). Depending on the used traffic model, see [12], each type of requests will constitute a certain share of the overall requests. So for example if 1000 calls per second are being processed successfully, e.g., the INVITE messages are getting a positive reply from the destination, then we can expect to receive on the average also 1000 BYE requests per second. If more than the expected level of a certain type of requests was received then this can be seen as an indication of a DoS attack and the requests arriving in excess of the expected traffic should be dropped. Hence, the approach of Sec.III-A is extended as follows:

- During each measurement interval T , the number of arriving data packets (D) for each type of in-dialog requests is counted.
- Each type of requests that exceeds its engineered value will be dropped with a probability of $((D - D_{en})/D)$. Re-INVITE messages will be rejected instead of dropped with a 500 reply including a retry-after header indicating that the user agent should retry the re-invite after T_{retry} seconds.

Thereby, the rejection rate of the requests which are suspected to be part of a DoS attack will be set proportionally to the number of requests sent in excess of the engineered load. This rejection probability would, however, still allow the forwarding of the engineered load.

For the case of in-dialog messages, rejecting requests can often have negative side effects. In IMS [13] the session establishment requires the exchange of an INVITE plus a number of PRACK and UPDATE requests. In case the INVITE message was accepted but the PRACK or UPDATE requests were rejected then the session establishment would be prolonged by at least the value of the retry-after indicated in the 500 reply. Also, a user agent that has sent a BYE might not even wait for the duration of the retry-after time (T_{retry}) indicated in the 500 replies.

As dropped messages will be normally retransmitted, the retransmitted messages will have some chance of passing an overloaded server. That is, with the exponential retransmission approach used by SIP, see [2], a user agent would retransmit an in-dialog request up to 9 times before quitting the session.

The re-INVITE messages can be rejected, as the user agents have to maintain the transaction state anyway and delaying will in general not have a considerable negative effects on the user's session.

The probability that all retransmitted requests sent by a user agent get dropped can be determined as p^n with p being the drop or rejection probability at the proxy and n being the number of the retransmitted request. Hence, to ensure that the probability that at least one of the retransmitted requests pass through the overloaded server with a probability of p_{pass} , p should be set in the following manner:

$$p^n < p_{pass} \quad (3)$$

$$p < \sqrt[n]{p_{pass}} \quad (4)$$

with set to 10.

D_{en} is set to different values depending on the type of the request. For requests that are sent only once in a dialog such as BYE or UPDATE, D_{en} is set to the following value:

$$D_{en} = \max(E_I, I) \times B_{en} \quad (5)$$

with I as the average value of successful session initiating INVITE requests and B_{en} as a burstiness factor used to accommodate sudden bursts of traffic.

Re-invite messages occur more often during a session. In this case, D_{en} would be set to

$$D_{en} = \max(E_I, I) \times B_{en} \times N_r \quad (6)$$

with N_r as the expected number of re-invite messages during a session and can be estimated as the average duration of a session divided by the session update timer, see [14]. The session update timer indicates the frequency with which reINVITE messages are sent.

C. Burstiness Protection For CPU

On one hand, Internet traffic is rather bursty in nature, see [15]. On the other, resources such as CPU should only be measured as average values over some time intervals. Further, continuously calculating the rejection rate and updating the system behavior would require a considerable amount of resources. Hence, it is possible that in between two measurement points, a large burst of signaling packets arrives that would require more processing resources than are available to the system. To allow for bursts of traffic an internal traffic counter is used that counts the number of received requests. Once the number of received requests of one type of engineered traffic, e.g., E_I, E_D, E_R, E_O exceeds a certain threshold, B_{en} times the engineered traffic for that kind of requests with ($B_{en} \geq 1$) the excess traffic will be rejected. That is, if for example if during the measurement interval T more than $(E_I \times B_{en})$ INVITE requests were received the proxy will start rejecting all incoming session initiating INVITE requests.

The CPU value is measured every T seconds. As the CPU values C_T oscillate, we use an averaged value of the CPU \hat{C}_T and is determined as

$$\hat{C}_T = \hat{C}_{T-1} \times (1 - \sigma) + C_T \times \sigma \quad (7)$$

For our measurements we used a σ value of 0.6.

D. Memory Specific Overload Control Procedure

Besides CPU, memory constitutes the second critical resource that can become overloaded in a SIP proxy. When acting in transaction stateful mode, a SIP proxy needs to keep some state information describing on-going transactions for a certain period of time. If no protective measures are taken, then during an overload phase, all of the memory available to the proxy might be occupied. In such a case the proxy would no longer be able to serve new calls. The average period of time during which the transaction state information must be kept at the proxy depends on the type of the requests:

- A session initiating INVITE transaction would generally last for a duration of T_I . This value includes a ringing period besides the processing delays at the traversed proxies and the round trip delay.
- A REGISTER transaction usually lasts for a duration of T_R . This value includes two rounds of message exchanges

which are caused by the authentication procedure and the delay at the registration server which includes the delay caused by the database lookup and the delay needed for conducting the authentication procedure.

- As the other kinds of SIP transactions generally do not include user interaction or authentication delay, the transaction delay T_O consists mainly of the processing and transport delay.

Hence, the minimum amount of memory (M_{min}) needed to support the engineered traffic can be determined as follows:

$$M_{min} = S \times (E_I \times T_I + E_D \times T_O + E_O \times T_O + E_R \times T_R) \quad (8)$$

with S as the number of bytes used by a SIP proxy to keep the state information of a transaction.

The maximum threshold M_{max} is then set to

$$M_{max} = M_{min} \times B \quad (9)$$

with B as a factor that allows us to accommodate cases of higher delays in the network or bursts of traffic and is ($B > B_{en}$).

As described in Sec. III-A while the value of the used memory is between M_{min} and M_{max} the proxy would start rejecting incoming requests, i.e., session initiating INVITE, REGISTER and out of dialog requests, in a probabilistic manner.

A simple denial of service attack can be realized by sending SIP requests to destinations that do not reply. This would cause the SIP proxy to start retransmitting the requests and would prolong the duration over which the proxy needs to keep the transaction state information considerably. Actually, non-INVITE transactions would reside at the proxy for 32 seconds, INVITE transactions might have to be kept at the proxy for several minutes. Such an attack would deplete the memory resources at the proxy and would prevent it from serving the engineered traffic. The probabilistic rejection approach would not be sufficient by itself, as the memory depletion is not caused by a high call arrival rate but due to the excessive transaction delays. To accommodate this case, once the consumed memory value reaches M_{min} , all active transactions will be continuously checked and

- All session initiating INVITE transactions that are older then $T_I \times B$ are then terminated and the sender of the request is sent a 500 reply.
- All in-dialog transactions that are older then $T_O \times B$ are terminated. No replies are sent to the originator of the request. Thereby, if the request has not originated from a malicious user, the request would be retransmitted and would still have a chance of reaching its destination.
- All REGISTER transactions that are not destined to the proxy and are older then $T_R \times B$ are terminated and the originator of the request is sent a 500 reply.

- All other requests that are older than $T_O \times B$ are terminated and the originator of the request is sent a 500 reply.

While terminating a transaction prematurely deviates from the specifications of SIP, see [2], this approach prevents the depletion of the memory and enables the proxy to continue serving non-malicious traffic, e.g., requests that are answered by the end destinations in the average response times multiplied by B .

IV. PERFORMANCE EVALUATION

In this section we test the performance of R-SOCA under different traffic conditions.

A. Test Environment

To test the performance of R-SOCA we extended a SIP proxy¹ with the required logic. The testbed itself consists of a number of senders initiating calls to a number of receivers through. The SIP requests are sent from the senders to two forwarding SIP proxies which then forward the traffic to the target SIP proxy. The SIP proxies are connected to the network over a gigabit link, see Fig. 2. The senders and receivers are emulated using the SIPP tool². To be able to test R-SOCA the target SIP proxy was extended with the following features:

- Measure the CPU load each T seconds
- Continuously measure the memory used by the proxy
- Count the amount of each type of incoming requests (D)
- Execute the R-SOCA logic, e.g., determine the rejection probability (\hat{P}_T) and the drop and rejection probability of in-dialog messages
- Probabilistically reject or drop incoming requests if required so by R-SOCA. When there is the need to decide if a request is to be forwarded or rejected a random number between 0 and 1 is generated. If this number is lower than the rejection probability then the request is rejected.

The proxy was configured to support an engineered load (E_I) of 2000 calls per second. The burstiness factor (B_{en}) was set to 1.7.

B. CPU Performance Evaluation of R-SOCA

The minimum threshold (C_{min}) was set to 20% of the available CPU. This was determined by measuring the CPU needed for handling the engineered traffic. The maximum threshold (C_{max}) was set to 40%.

¹The SIP Express Router was used. See www.iptel.org for more information
²SIPP is an open source traffic generator available under <http://sipp.sourceforge.net/>

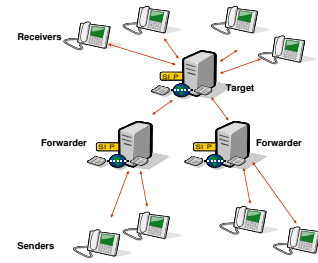


Fig. 2. Test environment

a) *Performance of R-SOCA under Flash Crowd and DoS Scenarios:* In this test, the performance of R-SOCA is tested when the proxy faces an increasing number of session initiating INVITE requests. This can be due either to a flash crowd scenario or an attacker generating large numbers of session initiating INVITES.

The arrival rate of the requests at the proxy is increased in a linear manner up to a maximum of 5000 calls per second and then is reduced again to the level of the engineered load (E_I).

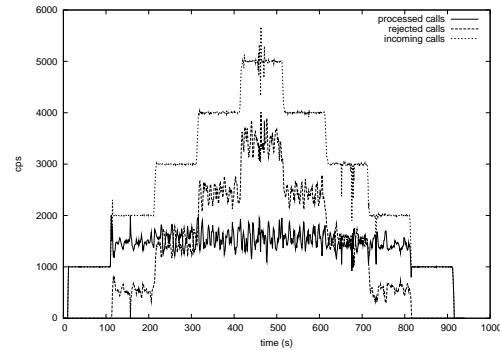


Fig. 3. Relation between received and accepted calls for linear traffic when using R-SOCA

Fig. 3 shows that even though the amount of incoming calls is increasing the SIP proxy is still capable of serving at least the engineered load ($E_I = 1000$). The CPU usage depicted in Fig. 4 shows that the CPU usage is bounded by the maximum and minimum thresholds.

The same test was repeated with the target proxy using an overload control mechanism based on sending a 503 reply when the CPU load exceeds the maximum threshold (C_{max}). The 503 replies include a retry-after with a value that is randomly chosen between 1 and 3 seconds. As for the case of R-SOCA the CPU load was determined as a weighted moving average.

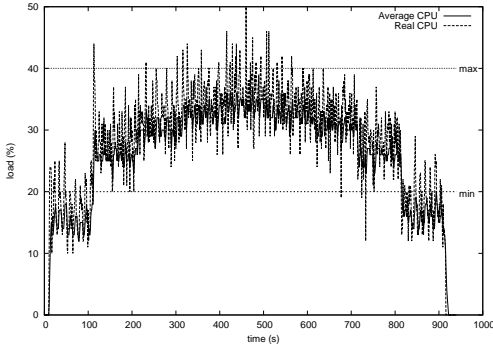


Fig. 4. Relation between CPU and rejection probability for linear traffic when using R-SOCA

Fig. 5 depicts the relation between the incoming and accepted calls. Especially in the phase of a high number of calls, the 503 based overload control schemes can achieve a better throughput in terms of accepted number of calls than R-SOCA. This stems from the different goals and control strategies deployed by the two approaches. The goal of R-SOCA is to keep the resource usage in predictable ranges. The 503 approach aims at using all the possible resources for serving the incoming requests. Therefore, with R-SOCA the average CPU usage is in general below the maximum threshold and the actual one exceeds the maximum threshold only rarely. Fig. 6 depicts that when using 503 for overload control the average CPU usage regularly exceeds the maximum threshold and the actual value is often much higher. This not only leads to the oscillative behavior of the system but can also lead to instability as these CPU usage peaks can temporarily block other processes on the system.

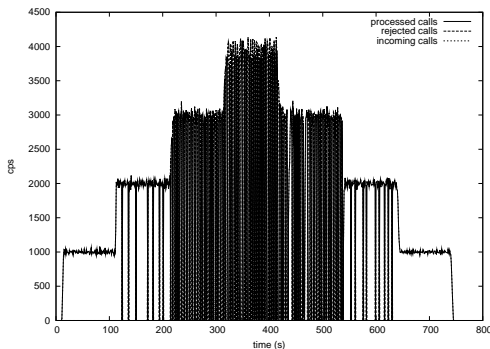


Fig. 5. Relation between received and accepted calls for linear traffic when using 503

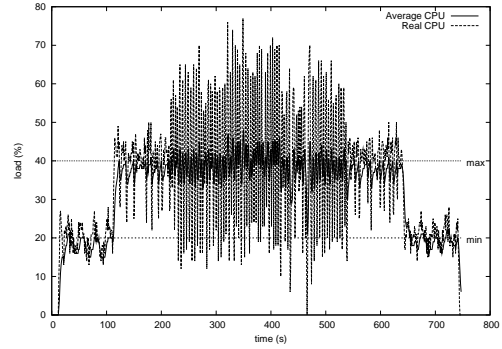


Fig. 6. Relation between CPU and rejection probability for linear traffic when using 503

b) Performance of R-SOCA under Heavy Load: In this scenario the performance of R-SOCA is tested when subjected to an incoming call rate of around 10000 session initiating INVITE requests per second which is much higher than the supported engineered load ($E_I=1000$). Actually the amount of received traffic results in exceeding both the minimum and maximum thresholds. As seen in Fig. 7 R-SOCA manages to keep the CPU load around the maximum threshold (C_{max}). From Fig. 8 we can observe that the CPU load is distributed between processing and rejection of the incoming calls. On the average around 1500 calls per second are still successfully processed and maximally 1700 which is the maximum accepted burstiness level.

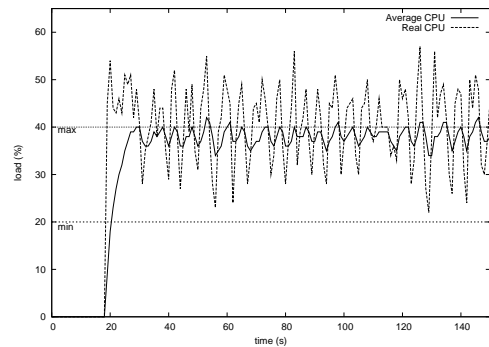


Fig. 7. CPU consumption of R-SOCA under heavy load

c) Performance of R-SOCA under BYE DoS Attacks: In the previous tests DoS attacks were initiated by sending a large number of INVITES. In the scenario tested here, the emulated DoS attack consists of a sender sending a constant load of 12000 BYE messages per second. With the DoS protection mechanism of R-SOCA, BYE requests that arrive in excess of

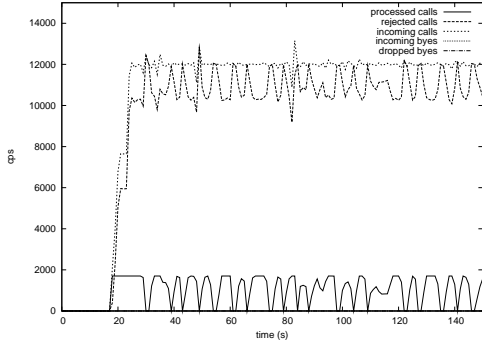


Fig. 8. Call processing behavior of R-SOCA under heavy load

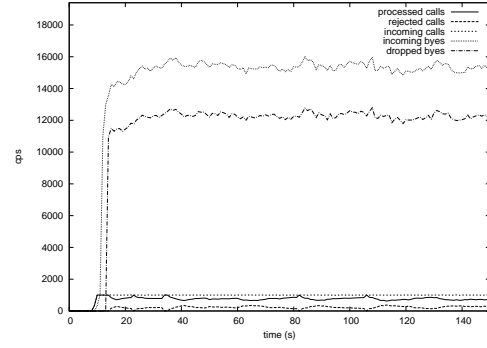


Fig. 9. Performance of R-SOCA under BYE DoS attack

the expected values are dropped. With an engineered traffic of 1000 calls per second and the burstiness factor set to 1.7, the expected number of BYEs would be 1700. In general a dropped BYE would be retransmitted by the user agent. However, as it is simpler for an attacker to generate multiple BYE requests instead of keeping a state machine that would be needed for conducting the SIP retransmission behavior, it is assumed here that an attacker will not retransmit a dropped BYE.

Further, p_{pass} was set to 10% which results in a maximal allowed drop rate of 79.5%. Thereby, only 10% of the sent BYE transactions will not be able to terminate successfully.

Fig. 9 depicts the behavior of the proxy when it is receiving 12000 BYE messages and 1000 INVITE requests per second. The DoS protection mechanism will result in dropping BYE requests both from the attacker as well as from legitimate users. The dropped BYE requests which were sent by legitimate users are retransmitted leading to an overall rate of incoming BYE requests of around 18000 BYEs per second. From Fig. 10 we can observe that even under this heavy load of incoming messages, the overall CPU rate is at around 23% which is only minimally higher than the used C_{min} .

Tab. I summarizes the results of the measurement. Out of nearly 200000 initiated calls around 15% of the calls were rejected. Out of the successful calls 94% were successfully terminated. 6% of the calls could not be terminated successfully as the sent BYE requests and their retransmissions were dropped by the overloaded proxy. Note that by increasing the value of the maximally expected BYE requests or using a smaller p_{pass} values would reduce the number of unsuccessful terminations. However, this would mean that a larger number of malicious BYE requests are processed by the proxy and hence would lead to a higher CPU usage which would then increase the call rejection rate.

Fig. 11 depicts the behavior of the proxy when it is receiving 12000 BYE messages and 2000 INVITE requests per second,

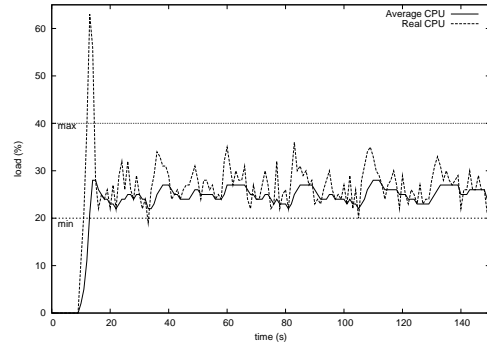


Fig. 10. CPU performance of R-SOCA under BYE DoS attack

e.g. double the engineered traffic. We can observe that the proxy in this case will allow for the engineered traffic to pass through. From Fig. 12 we can observe that even under this heavy load of incoming messages, the overall CPU rate is at around 30% which is between the thresholds used for controlling the CPU usage.

Tab. II summarizes the results of this measurement. Out of nearly 330000 initiated calls around 50% of the calls were rejected. Out of the successful calls 92% were successfully terminated. 8% of the calls could not be terminated successfully as the sent BYE requests and their retransmissions were dropped by the overloaded proxy.

The same attack was launched on a server using 503 replies for indicating overload. In this scenario 12000 BYE and 1000

| Sent Invites | Successful calls | Successful termination |
|--------------|------------------|------------------------|
| 196911 | 161193 | 150150 |

TABLE I
OVERALL PERFORMANCE OF R-SOCA UNDER BYE DoS ATTACK

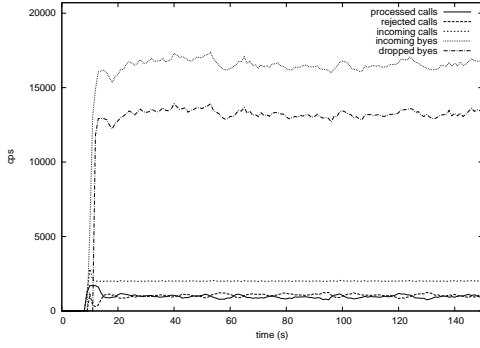


Fig. 11. Performance of R-SOCA under BYE and INVITE DoS attack

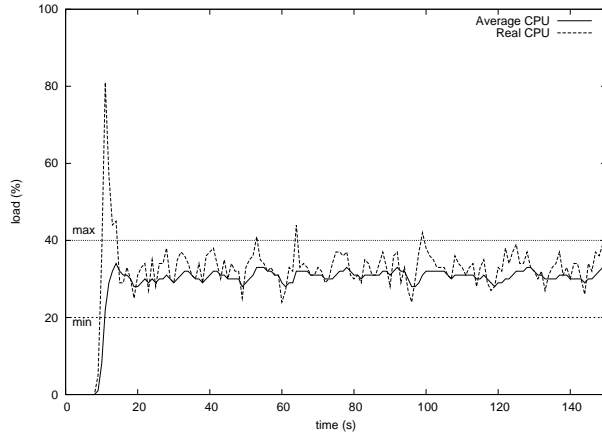


Fig. 12. CPU Performance of R-SOCA under BYE and INVITE DoS attack

INVITE requests per second were sent to the targeted server. Once the CPU load of the server exceeds the maximum threshold (C_{max}) the proxy replies with a 503 to the incoming INVITE requests. Upon that, the forwarding proxies stop sending any new requests to the server for the retry-after period.

The results depicted in Fig. 13 and Tab. III suggest that only around 5% of the initiated calls are successful. From Fig. 14 it can be observed that the CPU load is nearly constantly around the maximum threshold which is caused by the high number

| Sent Invites | Successful calls | Successful termination |
|--------------|------------------|------------------------|
| 363662 | 184290 | 168857 |

TABLE II
OVERALL PERFORMANCE OF R-SOCA UNDER BYE AND INVITE DoS ATTACK

| Sent Invites | Successful calls | Successful termination |
|--------------|------------------|------------------------|
| 278938 | 46763 | 46763 |

TABLE III
OVERALL PERFORMANCE OF 503 UNDER BYE AND INVITE DoS ATTACK

of the BYE requests. Hence, while using 503 as the basis for overload control might be sufficient for dealing with a high number of incoming INVITE requests, further mechanisms are needed for handling of DoS attacks.

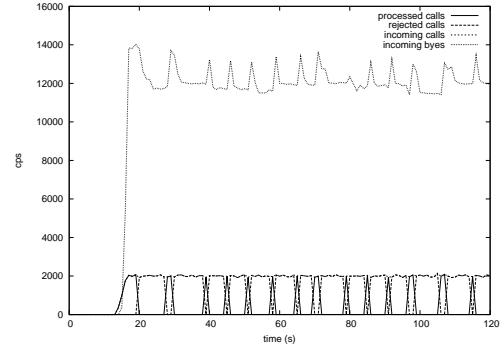


Fig. 13. Performance of 503 under BYE and INVITE DoS attack

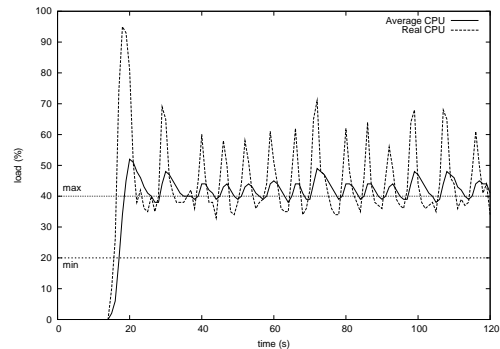


Fig. 14. CPU Performance of 503 under BYE and INVITE DoS attack

C. Memory Control Performance of R-SOCA

With each received request, a stateful SIP proxy maintains some transaction state until the transaction is terminated. A simple yet effective DoS attack would consist of sending a lot of requests that are not answered. This can be realized by manipulating a receiver or creating a fake SIP account and bounding some address to it on top of which no SIP software is

running. In this case the SIP proxy will retransmit the request a number of times before timing out the transaction.

In this section the behavior of R-SOCA is tested in the case overload is caused by high memory consumption. The used SIP proxy consumes around 8.5 KBytes per transaction and the used average call establishment time (T_I) is set to 10 seconds and the engineered traffic is set to 1000 calls per second. This results in a minimal threshold (M_{min}) of 85 MBytes and with the maximum allowed transaction duration set to 20 seconds, e.g., $B = 2$, the maximum threshold (M_{max}) is set to 170 MBytes.

Figure 15 shows the memory consumption of the SIP proxy for the case when the proxy is receiving 1000 calls per second. The INVITE requests are addressed to receivers that do not reply. The figure shows that the memory consumption increases over time. After passing the minimum threshold (M_{min}) the proxy starts rejecting calls and once the maximum transaction time expires, e.g. 20 seconds, the proxy will start releasing memory and the memory consumption stabilizes at a level that is between the maximum and minimum thresholds. The number of calls stabilizes at around 720 calls per second, see Fig. 16. Thereby, the combination of call rejection and transaction termination manages to keep the memory at predictable levels.

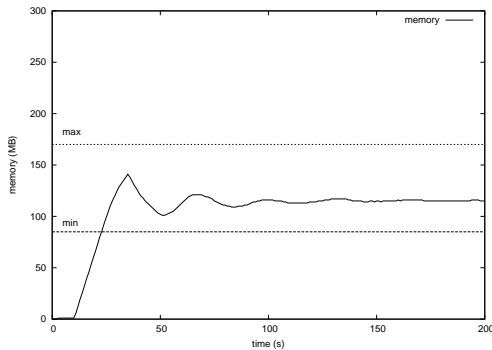


Fig. 15. R-SOCA memory control

Figures 17 and 18 depict the performance of the system under a higher load of incoming calls, namely 2000 calls per second. As previously the calls are destined to receivers that do not answer. From the figures we can observe that with R-SOCA the memory usage is still bounded between the maximum and minimum thresholds. The call throughput is around 780 calls per second.

To test the behavior of a system that uses 503 for regulating overload status the previously run tests were repeated with the target proxy rejecting calls with a 503 reply as soon as the memory reaches the M_{max} . The retry-after value was set to 1 second. Further, the proxy terminates transactions that are

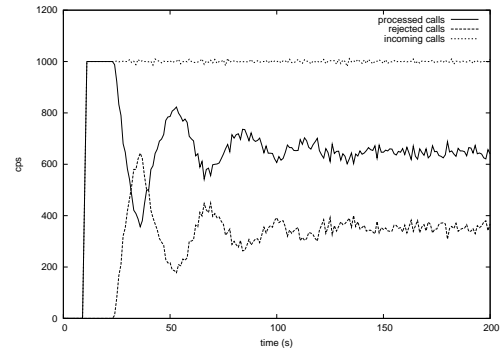


Fig. 16. Call throughput in case of memory overload

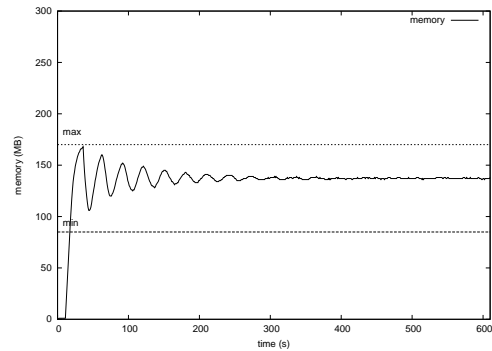


Fig. 17. R-SOCA memory control in case of a flash crowd

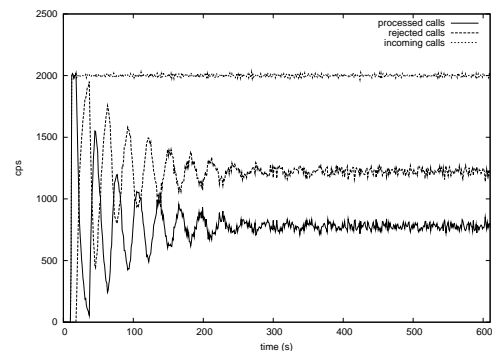


Fig. 18. Call throughput in case of memory overload and flash crowd

older than then the maximum transaction time which was set here to 20 seconds. The incoming call rate was set to 2000 calls per second.

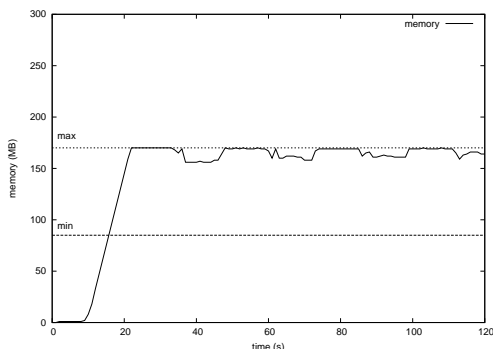


Fig. 19. R-SOCA memory control in case of a flash crowd

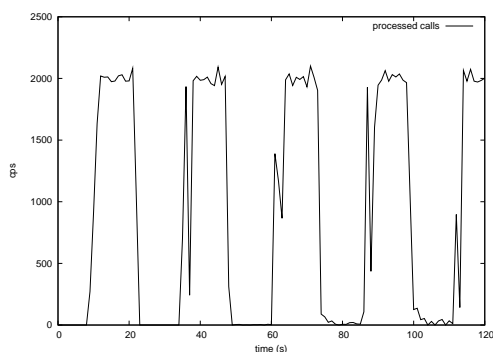


Fig. 20. Call throughput in case of memory overload and flash crowd

From figures 19 and 20 we can observe that the system shows a rather oscillative behavior with the periods of high call throughput and then periods of call rejection. The memory is nearly constantly at M_{max} . While the average rate -around 920 calls per second- is higher than with R-SOCA, the overall behavior is in a top and go manner with periods of call acceptance and then call rejection. Also the memory is used all of the time at the maximum possible value.

V. SUMMARY AND FUTURE WORK

In this paper we presented a novel scheme for dealing with overload situations in SIP proxies. The measurement results suggest that R-SOCA can achieve its goals and enable an overloaded SIP proxy to continue serving SIP traffic under high overload situations and denial of service attacks and still keep the resource usage at a pre-defined and expected level.

Compared to the basis mechanism of SIP, e.g., sending a 503 reply when in overload status, R-SOCA achieves a more predictable and stable behavior. While with 503 it is possible to achieve a higher throughput in some scenarios, this is only achieved at the cost of higher resource usage and a more oscillative behavior.

While already offering very promising results, R-SOCA will have to be extended to support additional features especially in the area of dynamic parameter setting. Currently, the thresholds used for determining the overload status of the SIP proxy are pre-defined and are set based on measurement of the system. Detecting and setting these thresholds dynamically would enhance the efficiency of R-SOCA and reduce the configuration overhead. Measuring the distribution of request types and setting the expected values for each request type dynamically, would further improve the DoS detection and enable R-SOCA to react faster to such attacks.

Finally, besides optimizing the schemes, future work will investigate the performance of R-SOCA in real-live environments and under different usage scenarios.

REFERENCES

- [1] M. Ohta, "Simulation study of sip signaling in an overload condition." in *Communications, Internet, and Information Technology*, M. H. Hamza, Ed. IASTED/ACTA Press, 2004, pp. 321–326.
- [2] J. Rosenberg, H. Schulzrinne, G. Camarillo, A. Johnston, J. Peterson, R. Sparks, M. Handley, and E. Schooler, "SIP: Session Initiation Protocol," RFC 3261 (Proposed Standard), Jun. 2002, updated by RFCs 3265, 3853, 4320, 4916. [Online]. Available: <http://www.ietf.org/rfc/rfc3261.txt>
- [3] G. Zhang, S. Ehlert, T. Magedanz, and D. Sisalem, "Denial of service attack and prevention on sip voip infrastructures using DNS," in *Principles, Systems and Applications of IP Telecommunications (IPTCOMM)*, NY, USA, July 2007.
- [4] J. Kuthan, S. Ehlert, and D. Sisalem, "Denial of service attacks targeting a SIP VoIP infrastructure - attack scenarios and prevention mechanisms," *IEEE Networks Magazine*, vol. 20, no. 5, Sep. 2006.
- [5] J. Rosenberg, "Requirements for management of overload in the session initiation protocol," Internet Engineering Task Force, Internet Draft, Oct. 2006, work in progress.
- [6] V. Hilt, I. Widjaja, D. Malas, and H. Schulzrinne, "Session initiation protocol (sip) overload control," Internet Engineering Task Force, Internet Draft, Mar. 2007, work in progress.
- [7] A. Acharya, D. Kandlur, and P. Pradhan, "Differentiated handling of SIP messages for VoIP call control," United States Patent 20050105464, May 2005.
- [8] M. Ohta, "Overload protection in a sip signaling network," in *International Conference on Internet Surveillance and Protection (ICISP'06)*, 2006.
- [9] E. C. Noel and C. R. Johnson, "Initial simulation results that analyze sip based voip networks under overload," in *International Teletraffic Congress*, ser. Lecture Notes in Computer Science, L. Mason, T. Drwiega, and J. Yan, Eds., vol. 4516. Springer, 2007, pp. 54–64.
- [10] C. Shen, H. Schulzrinne, and E. Nahum, "Sip server overload control: Design and evaluation," in *Conference on Principles, Systems and Applications of IP Telecommunications (IPTCOMM08)*, Heidelberg, Germany, Jul. 2008.
- [11] E. Nahum, J. Tracey, and C. Wright, "Evaluating SIP server performance," IBM T. J. Watson Research Center, Research report RC24183, Feb. 2007.
- [12] H. Schulzrinne, S. Narayanan, J. Lennox, and M. Doyle, "Sipstone - benchmarking sip server performance," Apr. 2002.

- [13] “Signalling flows for the ip multimedia call control based on session initiation protocol (sip) and session description protocol (sdp),” 3rd Generation Partnership Project, Technical Specification Group Core Network and Terminals, 2007.
- [14] S. Donovan and J. Rosenberg, “Session Timers in the Session Initiation Protocol (SIP),” RFC 4028 (Proposed Standard), Apr. 2005.
- [15] M. Grossglauser and J.-C. Bolot, “On the relevance of long-range dependence in network traffic,” *IEEE/ACM Transactions on Networking*, vol. 7, no. 5, pp. 629–640, 1999. [Online]. Available: citeseer.ist.psu.edu/article/grossglauser96relevance.html