Formal Specification and Verification of a Selective Defense for TDoS Attacks

Yuri Gil Dantas[1], Marcilio O. O. Lemos[2], Iguatemi E. Fonseca[2], and Vivek Nigam[2]

[1]Computer Science Department, TU Darmstadt, Germany
[2]Computer Science Department, UFPB, Brazil
{marciliolemos,iguatemi,vivek}@ci.ufpb.br
dantas@mais.informatik.tu-darmstadt.de

Abstract. Telephony Denial of Service (TDoS) attacks target telephony services, such as Voice over IP, not allowing legitimate users to make calls. There are few defenses that attempt to mitigate TDoS attacks, most of them using IP filtering, with limited applicability. In our recent work, we proposed to use selective strategies for mitigating HTTP Application-Layer DDoS Attacks demonstrating their effectiveness in mitigating different types of attacks. This paper demonstrates that selective strategies can also be successfully used to mitigate TDoS attacks, in particular, two attacks: the Coordinated Call Attack and the Prank Call attack. We formalize a novel selective strategy for mitigating these attacks in the computational tool Maude and verify these defenses using the statistical model checker PVeStA. When compared to our experimental results (reported elsewhere), the results obtained by using formal methods were very similar. This demonstrate that formal methods is a powerful tool for specifying defenses for mitigating Distributed Denial of Service attacks allowing to increase our confidence on the proposed defense before actual implementation.

1 Introduction

Telephony Denial of Service (TDoS) attacks is a type of Denial of Service (DoS) attack that target telephony services, such as Voice over IP (VoIP). With the increase in the popularity of VoIP services, we have witnessed an increase in TDoS attacks being used to target hospital VoIP systems [1, 2] and systems for emergency lines (like the American 911 system) [3]. Moreover, according the FBI, 200 TDoS attacks have identified only in 2013 [2].

This paper investigates the use of selective defenses [4] for mitigating two common TDoS attacks: The Coordinated Call [5] and the Prank Call [6] attacks:

The Coordinated Call attack [5] exploits the fact that pairs of attackers, Alice and Bob, can collude to exhaust the resources of the VoIP server. Assume that Alice and Bob are valid registered users.[1] The attack goes by Alice simply calling

[1] This can be easily done for many VoIP services.

Bob and trying to stay in the call as long as she can. Since the server allocates resources for each call, by using a great number of pairs of attackers, one can exhaust the resources of the server and denying service to honest participants. This is a simple, but ingenious attack, as only a small number of attackers is needed generating a small network traffic (when compared to SIP flooding attack for example). Thus it is hard for the network administrator to detect and counter-measure such attack.

The Prank Call attack [6] is similar to the usual flooding attack [7] denying service by overloading the target resources. It has been carried out to shutdown essential public services, such as the US emergency number (911) and hospital lines. The attack follows by a large number of attackers (or their bots) initiating calls to the target call-center. This causes that many, if not all, telephones in the center to ring. Once the attendant picks up the phone, he can normally notice that this is fake call and puts down the phone. However, since the number of calls is very large, the phone rings again, not allowing legitimate clients to be served.

Formal methods and, in particular, rewriting logic can help developers to design defenses for mitigating DDoS attacks. In our previous work [4] we used selective strategies in the form of the tool SeVen for mitigating HTTP Low-Rate Application-Layer DDoS attacks targetting web-servers. We formalized different attack scenarios in Maude and since our strategies are constructed over some probability functions, we used statistical model checking [8], namely PVeStA [9], to validate our defense. Due to our reasonable preliminary results, we implemented SeVen and carried out experiments over the network obtaining similar results to the ones obtained using formal methods. It took us *only 3 person months* to obtain our results using formal methods, while it took us *24 person months* to obtain our first experimental results. Although we strongly believe that systems should also be validated by means of experiments, the confidence acquired from our formal analysis was invaluable for the success of this project.[2]

This paper continues our general goal of using selective strategies for mitigating DoS attacks, in particular, here for mitigating TDoS attacks. We followed the same methodology as before, first formalizing our defense, the Coordinated Call and the Prank Call attacks in Maude and using PVeStA to validate our defense's effectiveness. While this paper explains the formal model used, in another technical report [10], we detail our initial experimental results on mitigating the Coordinated Call attack using SeVen to defend VoIP servers. The results obtained by using our formal model and our experimental results were very similar.

This paper is organized as follows. Section 2 we review Session Initiation Protocol (SIP) used for initiating a VoIP call and also explain two different TDoS attacks: Coordinated Call and Prank Call attacks. Section 3 explains how

[2] Notice that although our experiments on the network were controlled experiments, they used off-the-shelf tools, such as Apache web-servers, which implement a number of optimizations, and our experiments suffered from interferences that cannot be controlled, such as network latency.

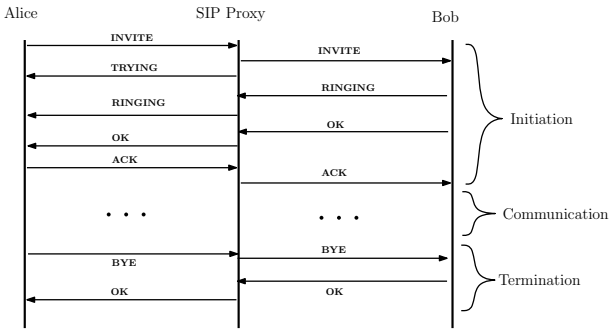


Fig. 1. Exchange of messages between the server and two users (Alice and Bob) during a normal execution of the SIP protocol.

SeVen works, while Section 4 describes its formalization in Maude. In Section 5, we explain our simulation results including our main assumptions, results and discussion of the results obtained. We comment in Section 6 related and future work. Finally, the implementation used to carry out our simulations is available for download at [11].

2 VoIP Protocols and DDoS Attacks

We now review the Session Initiation Protocol [12], which is one of the main protocols used to establish Voice over IP (VoIP) connections. Figure 1 shows the message exchanges performed to establish a connection between two registered users, Alice and Bob, where Alice tries to initiate a conversation with Bob. It also contains the messages exchanged to terminate the connection.

For initiating a call, Alice sends an INVITE message to the SIP server informing that she wants to call Bob. If Bob or Alice is not registered as valid users, the server sends a reject message to Alice. Otherwise, the server sends an INVITE message to Bob.[3] At the same time, the server sends a TRYING message to Alice informing her the server is waiting for Bob's response to Alice's invitation. Bob might reject the request, in which case the server informs Alice (not shown in the Figure), or Bob can accept the call by sending the message RINGING. Finally, the server sends the message RINGING to Alice and the parties exchange OK and ACK messages.

At this point, the communication is established and Alice and Bob should be able to communicate as long as they need/want. (This is represented by the three ellipses in Figure 1.) The call is then terminated once one of the parties (Alice) sends a BYE message to the server. The server then sends a BYE message to the other party (Bob), which then answers with the message OK, which is forwarded to Alice, and the connection is terminated.

[3] In fact, we omit some steps carried out by the server to find Bob in the network. This step can lead to DDoS amplification attacks [13] for which known solutions exists. Such amplification attacks are not, however, the main topic of this paper.

Many attacks have exploited SIP to deny the VoIP service. We detail two different attacks to the SIP protocols. The first one is the Coordinated Call attack and the second one is the Prank Call attack.

Coordinated VoIP Attack A pair of colluding attackers, A_1 and A_2, that are registered in the VoIP service,[4] call each other and stay in the call for as much time as they can. Once the call is established, the attackers stay in the call for indefinite time. They might be disconnected by some Timeout mechanism establishing some time bounds on the amount of time that two users might call. During the time that A_1 and A_2 are communicating, they are using resources of the server. If many pairs of attackers collude, then the resources of the server can be quickly exhausted. This attack is hard to detect using network analyzers because the traffic generated by attackers is similar to the traffic generated by legitimate clients. The attackers follows correctly the SIP protocol.

Our own experiments [10] replicating this attack show its effectiveness reducing the availability of the VoIP service to less than 15% of legitimate users.

Prank Call Attack The Prank Call attack is similar to a flooding attack [7] in that the attackers send a large number of requests targeting essential public resources services, such as the American 911 service. The attacker launches a high volume of calls in order to flood the target VoIP service, reducing the availability to legitimate users. Usually a particular call from the Prank Call attack does not take long, because the callee hangs up the call when he realizes that is a prank call. Prank Call attacks are difficult to track and investigate because the calls are classified as anonymous, hence using traditional traffic analysis tools might not be an efficient approach to mitigate such attacks.

3 SeVen

We proposed recently [4] a new defense mechanism, called SeVen, for mitigating Application-Layer DDoS attacks (ADDoS) using selective strategies. An application using SeVen does not immediately process incoming messages, but waits for a period of time, T_S, called a round. During a round, SeVen accumulates messages received in an internal buffer k. If the number of messages accumulated reaches the maximum capacity of the service being protected and a new incoming request R arrives, SeVen behaves as follows:

1. SeVen decides whether process R or not based on a probability P_1. P_1 is defined using the variable `PMod` following [14]:

$$\frac{k}{k + \texttt{PMod}}$$

[4] Or alternatively two honest users that have been infected to be zombies by some attacker.

At the beginning of the round, we set the variable `PMod` = 0. `PMod` is incremented whenever the application's capacity is exhausted and a new incoming request arrives reducing thus the probability of new incoming request being selected by SeVen during a round.

2. If SeVen decides to process R, then as the application is overloaded, it should decide which request currently being processed should be dropped. This decision is governed by P_2, a distribution probability *which might depend on the state of the existing request.*
3. Otherwise, SeVen simply drops the request R without affecting the requests currently being processed and sends a message to the requesting user informing that the service is temporally unavailable;

At the end of the round, SeVen processes the requests that are in its internal buffer (surviving the selective strategy) sending them to the application.

The intuition of why such a defense works is because whenever a system is overloaded, it is very likely that it is suffering a DoS attack, which means that it is very likely that an attacker request is occupying the resources of the service. Therefore, the probability of dropping an attacker's request is higher than the probability of dropping an honest request. Thus, even under severe attack of multiple attackers, an application running SeVen can maintain fair levels of availability.

There is, however, much space for specifying these probability distributions governing SeVen. In [4], we showed that by using simple *uniform distributions* for dropping existing requests, SeVen can be used to mitigate a number of ADDoS attacks using the HTTP protocol, such as the Slowloris and HTTP POST attacks even in the presence of a large number of attackers.

For mitigating both Prank Call and Coordinated Call attacks described in Section 2, we set the probability P_2 to depend on (1) the status of the call and (2) on the duration of a call. We consider two types of call status:

- WAITING: A call is WAITING if it has already sent an INVITE message, but it is still waiting for the responder to join the call, that is, it has not completed the initiation part of the SIP protocol;
- INCALL: A call is INCALL if the messages of initiation part of SIP have been completed and initiator and the responder are already communicating (or simply in a call).

Thus, any incoming INVITE requests assume the status of WAITING, and these can change its status to INCALL once the initiation part of SIP is completed.

We assume here that it is preferable to a VoIP server, when overloaded, to drop WAITING requests than INCALL requests that are communicating not for a *very long duration.* In many cases, it is true that interrupting an existing call is considered to be more damaging to server's reputation than not allowing a user to start a new call. This could also be modeled by configuring the probability distributions of SeVen accordingly. To determine whether a call is taking too

long, we assume that the server knows what is the average duration, t_M, of calls.[5]

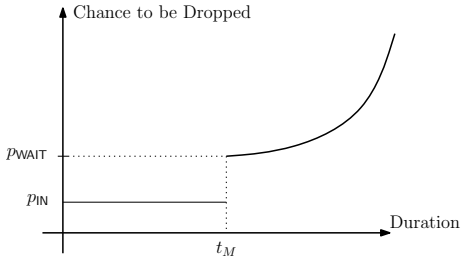


Fig. 2. Graph (not in scale) illustrating the behavior of SeVen according to the status of a call and its duration. p_{WAIT} is the probability of dropping a WAITING call, while p_{IN} the probability of dropping a INCALL call.

The probability of an INCALL request increase using a Poisson distribution[6] once this has a duration of more than t_M. Figure 2 depicts roughly the probability distribution used to drop requests. The actual function d (for drop factor) is of the form, where t is the call duration:

$$d(t) = \begin{cases} p_{\text{WAIT}} & \text{if } t = 0 \\ p_{\text{IN}} & \text{if } 0 \leq t \leq t_M \\ p_{\text{WAIT}} + e^{\alpha t / t_M} & \text{if } t > t_M \end{cases} \quad (1)$$

We use this probability distribution as an illustration of how SeVen can be used for mitigating VoIP DDoS attacks. Of course, there are many decision options for these probabilities which will depend on the intended application. For instance, one could consider that the Poisson distribution should begin only a period after t_M, or that it should be another distribution, etc. It will depend on the specific requirements of the defense. As our results in Section 5 demonstrate, the values chosen are good enough for the VoIP attacks we consider.

We also have developed SeVen as a proxy in C++ which implements the strategy explained above, *e.g.*, using the drop function 1, Poisson distribution and so on. The measures and results (which can be found in another submission [10]) were very similar to the ones detailed in Section 5. Such results confirm the success of our formal model proposed here.

3.1 Sample Execution

Consider the following buffer, $\mathcal{B}_i$, at the beginning of a round and assume that $k = 3$, `PMod` $= 0$, the initial time is 9 and the average call duration is $t_M = 5$ time units:

$$\mathcal{B}_1 = [\langle id_1, \mathsf{WAITING}, \mathsf{undef} \rangle, \langle id_2, \mathsf{INCALL}, 0.5 \rangle]$$

[5] The value of t_M can be obtained by the history of a VoIP provider's usage.

[6] We used a Poisson distribution because such distributions are normally used for modeling telephone calls arrival.

$\langle id, st, tm\rangle$ specifies that the call id has status st and the call started at time tm where tm is undef whenever st = WAITING. This buffer specifies that the id_1 is waiting the responding party to answer (with a RINGING message) his invitation request and that id_2 is currently in a call. This means that id_2 is calling already for way more than the expected average.

Assume that a message $\langle id_1, \text{RINGING}\rangle$ at time 9.5 arrives specifying that the responder of the request id_1 answered the call. The buffer is updated to the following:

$$\mathcal{B}_2 = [\langle id_1, \mathsf{INCALL}, 9.5\rangle, \langle id_2, \mathsf{INCALL}, 0.5\rangle]$$

Then the message $\langle id_3, \text{INVITE}\rangle$ arrives. Since the buffer is not yet full, a new request is inserted in the buffer and the message TRYING is sent to the requesting user. Notice that the RINGING message is not yet sent to the responding user. The buffer changes to:

$$\mathcal{B}_3 = [\langle id_1, \mathsf{INCALL}, 9.5\rangle, \langle id_2, \mathsf{INCALL}, 0.5\rangle, \langle id_3, \mathsf{WAITING}, \mathsf{undef}\rangle]$$

Suppose now that another message $m_1 = \langle id_4, \text{INVITE}\rangle$ arrives at time 10.5. As the buffer is now full, it sets `PMod` to 1 and the application has to decide whether it will keep m_1. SeVen generates a random number in the interval [0,1] using uniform distribution. Say that this number is less than $(3/3 + 1)$, which means that it will select to process m_1. However, it has to drop some existing request. The probabilities of dropping one the request in the buffer are as follows (see Figure 2):

- id_1 has probability p_{IN} to be dropped because it is calling for a duration less than t_M: $10.5 - 9.5 < 5$;
- id_2 has a much higher probability to be dropped because it is calling for twice t_M: $10.5 - 0.5 = 2 \times 5$;
- id_3 has probability p_{WAIT} to be dropped because it has WAITING status.

Suppose that the application decides to drop id_2, which means that the call is interrupted by the application. The resulting buffer is:

$$\mathcal{B}_4 = [\langle id_1, \mathsf{INCALL}, 9.5\rangle, \langle id_4, \mathsf{WAITING}, \mathsf{undef}\rangle, \langle id_3, \mathsf{WAITING}, \mathsf{undef}\rangle]$$

Assume that now the round time is elapsed. The application sends a RINGING message to the responder of the requests id_3 and id_4.

4 Formal Specification

Our specification follows [4,15,16] specifying the attack scenarios using the Actor Model where attackers, clients, and server send and receive messages. These messages are stored in a scheduler that maintains a queue of messages. The attackers do not take control over the channel. Instead they share a channel with clients.

We formalize all actors in the computational system Maude and carry out simulations by using the statistical model checker PVeStA. For sake of simplicity,

we considered the server and SeVen as one actor, which means that SeVen is also able to operate as a normal SIP Server, *e.g.*, processing and establishing call connections. Such decision does not affect the analysis of our results, which are similar to the ones of our experiments over the network [10]. In the following, we describe our Maude specification.[7]

```
eq initState =
<name: server | req-cnt: 0.0 , b-set: [0 | none], none >
<name: client-generate | server: server, cnt: 0 , none >
<name: attacker-generate | server: server, cnt: 0 , none >
        {initActor, (attacker-generate <- spawn )}
        {initActor, (client-generate <- spawn )}
        {Ts, server <- ROUND} .
```

The equation for *initState* specifies the initial state of our model, which contains three actors, an attacker generator, a client generation and SeVen. Each actor has an ID and a set of attributes. For example, SeVen is called `server` with attributes `req-cnt` storing the value of `PMod` and `b-set` the internal buffer with the current call connections. The attributes `cnt` in the other two actors stores how many clients and attackers have been created.

Finally, we also formalize the message configuration between actors that are going to be added in our scheduler. Each message configuration has the parameters delivery time and the message itself. For instance, we use the same `initActor` delivery time to initialize both actors clients and attackers with a message *spawn*. Besides that, we also create a periodically `ROUND` message to control the SeVen's round explained in Section 3, which is scheduled to be sent after `Ts` time units.

The equations for generating both clients and attackers are omitted here. Instead, we show their main rewrite rules. The clients have an attribute `status` specifying their call state. Its status is `none` before sending an `INVITE` message to the server which happens when it receives a message `pool` from the equation generating clients[8] as specified by the following rewrite rule:

```
rl [CLIENT-RECEIVE-POOL] :
        <name: c(i) | server: Ser, status: none, AS >
        {c(i) <- poll}
     =>
        <name: c(i) | server: Ser , status: invite, AS >
        { gt + delay, (Ser <- INVITE(c(i)))} .
```

The following rewrite rule specifies the behavior of a client upon receiving a RINGING message from the server. It changes the client's state from *invite* to

[7] For the sake of presentation, we simplified here some aspects such as the use of the scheduler appearing in the complete model which can be found in [11].

[8] Note that there is a value `delay` inserted when a message is sent in order to have a more realistic model.

connected and generates a message BYE, scheduled to be sent after some time in the interval $]0, \texttt{tMedio}]$. This means that all legitimate clients do not overpass the average time of the duration of calls. We omit the rule specifying when client receives a drop message.

```
rl [CLIENT-RECEIVE-RINGING] :
        <name: c(i) | server: Ser, status: invite, AS >
        {c(i) <- RINGING}
     =>
        <name: c(i) | server: Ser , status: connected, AS >
        { gt + randomNumber(0,tMedio), (Ser <- BYE(c(i)))} .
```

The rewrite rules for the attackers are similar to the client rules. The only difference is that no BYE message is generated, thus, specifying the Coordinated Call attack where attackers attempt to stay in the call for indefinite time. We elide these rules.

```
crl [SeVen-RECEIVE-INVITE] :
        <name: Ser | req-cnt: pmod , b-set: [lenB | B],  AS >
        {Ser <- INVITE(Actor)}
     => if (float(lenB) >= floor(lenBufSeVen)) then
              if p1 then ConfAcc {gt, Actor <- TRYING} {gt, ActorDr <- poll}
                    else ConfRej {gt + delay , Actor <- poll}
              fi
        else ConfAcc2 {gt + delay, Actor <- TRYING}
        fi
if p1 := sampleBerWithP(accept-prob(pmod))
   /\ { ActorDr, bufDr } := remUser(altPBuf(B, gt), sampleUniWithInt(altPBufLen(B, gt)))
   /\ nBuf := add([lenB + (- 1) | bufDr], < Actor gt INVITE >)
   /\ ConfAcc := <name: Ser | req-cnt: (pmod + 1.0), b-set: nBuf , AS >
   /\ ConfRej := <name: Ser | req-cnt: (pmod + 1.0), b-set: [lenB | B], AS >
   /\ b-setNu := add( [lenB | B], < Actor gt INVITE > )
   /\ ConfAcc2 := <name: Ser | req-cnt: pmod , b-set: b-setNu, AS > .

rl [SeVen-APP-ROUND] :
        <name: Ser | req-cnt: pmod , b-set: [lenB | B], AS >
        {Ser <- ROUND}
    =>
        <name: Ser | req-cnt: 0.0, b-set: [lenB | B], AS >
        {gt, reply(Ser, B, gt)} {gt + Ts, Ser <- ROUND} .
```

Fig. 3. Rewrite rules specifying SeVen's selective strategy.

Figure 3 depicts the rules implementing SeVen's strategy. For each INVITE message received by some actor `Actor`, the rule `SeVen-RECEIVE-INVITE` checks whether the buffer of the server reached its maximum. If not, then the incoming request is added to the server's buffer (`ConfAcc2`) and a message TRYING to the corresponding actor is created. Otherwise, SeVen throws a coin (`p1`) to decide

whether the incoming request will be processed using `pmod`. If SeVen decides to process the incoming request, then some request being processed (the one sent by `ActorDr`) is swapped with the new incoming request resulting in the buffer `nBuf` and `pmod` gets incremented, resulting in the configuration `ConfAcc`. Moreover, a `poll` message to `ActorDr` and a `TRYING` to `Actor` are created. Otherwise, the incoming request is rejected and `pmod` is incremented without affecting the server's buffer resulting in the configuration `ConfRej`. A poll message to `Actor` is also created.

The rule `SeVen-APP-ROUND` specifies that when the round finishes, all surviving requests in the server's buffer are answered, where a new round starts and `pmod` is re-set.

5 Simulation Results

We detail our simulation results obtained from our formal specification using the statistical model checker PVeStA [9]. Our simulations are parametric in the following values:

- Average time of a call – t_M: This is the assumed average time of the of calls of honest users. We assume $t_M = 5$ time units;
- Probability distribution parameters (Func 1) – $p_{\mathsf{IN}}, p_{\mathsf{WAIT}}$ and α: These are the constants used to configure the distribution probability for dropping requests as shown in Figure 2;
- SeVen Round Time – t_S: This is the time that SeVen waits accumulating requests, as described in Section 3. In our simulations, we use 0.4 time units.
- Size of Buffer – k: This is the upper-bound on the size of $\mathcal{B}$, denoting the processing capacity of the application. $k = 24$;
- Number of calls among honest participants ($countHonest$) and among colluding attackers ($countAttacker$). In all our simulations, we fixed the number of clients to $countHonest = 24$ requests. Whenever we create an honest request, we specify how long the users want to talk, *i.e.*, have the INCALL status;
- Total time of the simulation - *total*: This is the total time of the simulation using PVeStA. We used in our simulations *total* equal to 40 time units, similar to the time used in [15];
- Delay of the Network: We also assumed a delay of 0.1 time units of message in the network;
- Degree of confidence for the simulation: Our simulations were carried out with a degree of confidence of 99% (see [8,17] for more details on probabilistic model checking).

Quality Measures In our simulation, we use novel quality measures specific for VoIP services. These are specified by expressions of the QuaTEx quantitative, probabilistic temporal logic defined in [17]. We perform statistical model checking of our defense in the sense of [8]: once a QuaTEx formula and desired degree

of confidence are specified, a sufficiently large number of Monte Carlo simulations are carried out allowing for the verification of the QuaTEx formula. The Monte Carlo simulations are carried out by the computational tool Maude [18] and the statistical model checking is carried out by PVeStA.

The QuaTEx formulas, *i.e.*, the quality measures, that we use in our simulations are defined below. The operator $\bigcirc$ is a temporal modality that specifies the advancement of the global time to the time of the next event (see [17] for more details).

- Complete: How many honest calls were able to stay in the INCALL status for the expected duration.

  $$complete(total) = \text{if } time > total \text{ then } \frac{countComplete}{countHonest} \text{ else } \bigcirc complete(total)$$

  where $countComplete$ is a counter that is incremented whenever an honest call is completed.
- Incomplete: How many honest calls were able to have the INCALL status but were dropped before completing the call, *i.e.* not staying in INCALL status for the expected duration;

  $$incomplete(total) = \text{if } time > total \text{ then } \frac{countIncomplete}{countHonest} \text{ else } \bigcirc incomplete(total)$$

  where $countInComplete = countIncall - countComplete$ and $countIncall$ is a counter that is incremented whenever an honest calls changes from status WAITING to INCALL.
- Unsuccessful: How many honest calls were not even able to reach the INCALL status. That is, how many calls were not even able to start talking between each other.

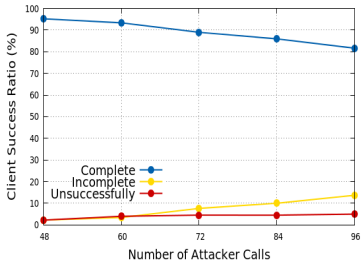
  $$unsuccessful(total) = \text{if } time > total \text{ then } \frac{countUnsuccess}{countHonest} \text{ else } \bigcirc unsuccessful(total)$$

  where $countUnsuccessful = countHonest - countIncall$.
- The average of clients incomplete calls: We also measure how many percent in average legitimate clients were able to talk in an incomplete call.

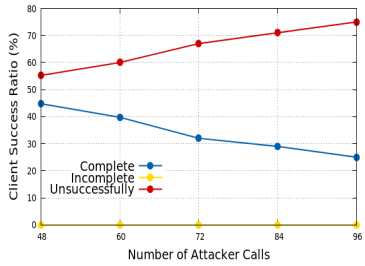
  $$avgInCall(total) = \text{if } time > total \text{ then } \frac{totalTimeInCall}{totalIncompleteCall} \text{ else } \bigcirc avgInCall(total)$$

  where $totalTimeInCall$ is the sum of how many percent of time clients were able to talk before being interrupted and the $totalIncompleteCall$ is the total of clients the were not able to finish their call as explained in Incomplete.

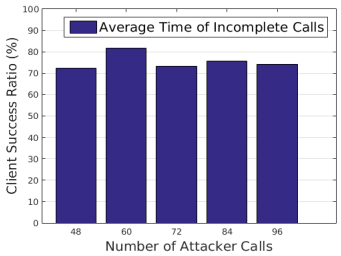
We consider that an honest request that was completed has a better performance than an honest request that was interrupted in the middle of a call which, on the other hand, has a better performance than an honest request that did not even succeed in starting a call, *i.e.*, never reached the INCALL status.



(a) Client Success Ratio when running SeVen.



(b) Client Success Ratio when *not* running SeVen.



(c) Average Duration of Incomplete Calls when using SeVen

Fig. 4. Simulation Results for when the application suffers a Coordinated Call Attack.

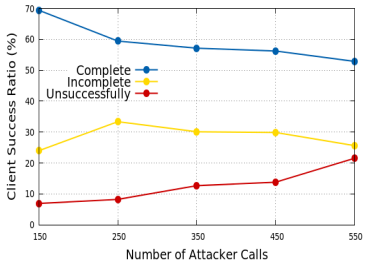
5.1 Coordinated Call Attack

Figure 4 contains our main results for the scenarios where the application is under the Coordinated Call attack discussed in Section 2. We considered scenarios where the application is using SeVen (Figure 4(a)) and not using any defense mechanism (Figure 4(b)).

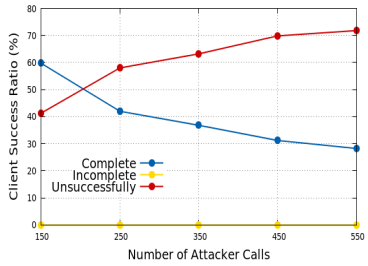
Figure 4(a) shows that the application maintains great levels of availability when using SeVen. The Complete calls reduce from 95% to 81% of the legitimate calls when the number of attackers increase. The difference is distributed between Incomplete and Unsuccessful calls, where the former is around 13% and the latter is around 5% of the legitimate calls. On the other hand, when SeVen is not running, in all simulations, most legitimate users are not able to start a call: When the number of attackers increase, the rate of Unsuccessful call increase from 55% to 75% and the rate of Successful call decrease from 45% to 25%, which means that when SeVen is running, there is an increase of availability by a factor of 3.

Moreover, the average duration of incomplete calls (Figure 4(c)) stayed around 70%, that is, in average, a call that was dropped before completing its duration was able to stay communicating for around 70% of the intended time.

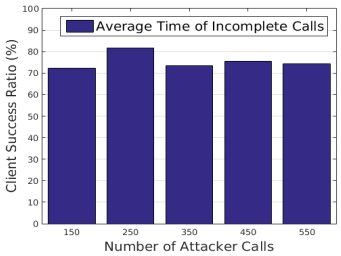
Comparison with our Experimental Results We implemented [10] the Coordinated Call attack and SeVen as described here. Our experimental results were



(a) Client Success Ratio when running SeVen.



(b) Client Success Ratio when *not* running SeVen.



(c) Average Duration of Incomplete Calls

Fig. 5. Simulation Results for when the application suffers Prank Call attack.

very similar to the ones we obtained here. In particular, when not running SeVen, we observed very high levels of Unsuccessful calls (85% of legitimate calls), low levels of Complete calls (15% of legitimate calls) and no Incomplete calls. These results are very similar to the ones in Figure 4(b). When running SeVen, our experiments showed a high rate of Successful calls (65% of legitimate calls), a low rate of Incomplete calls (25% of legitimate calls) and very low rate of Unsuccessful calls (10% of legitimate calls). These results are very similar to the ones in Figure 4(a). Finally, we also measured the average duration of Incomplete calls with an average of 60% of total call duration being very close to the results in Figure 4(c).

These results support our claim that Formal Methods can be used for proposing novel selective defenses for mitigating DDoS attacks. We leave to future work the comparison of formal methods results with experimental results for other attacks such as the Prank Call attack.

5.2 Prank Call Attack

Figure 5 shows the results obtained when the application is under a Prank Call attack. We observe that when SeVen is running, the application maintains fair levels of availability. Whereas without SeVen the number of calls that have been completed drops to around 28% of legitimate calls, it drops to 53% when SeVen is running, which means an improve of availability by a factor of almost 2. Moreover, the number of calls that is not even able to reach an INCALL status, the

Unsuccessful calls, only increases more sharply to 21% in the presence of 550 attackers when using SeVen, while it increases considerably when not using SeVen to around 71%. Finally, the average duration of Incomplete calls (Figure 5(c)) remains at around 70% of their intended duration.

These results provide us with strong evidence that SeVen can be used to mitigate the Prank Call attack. We are currently implementing the machinery to carry out experiments on the network.

6 Related and Future Work

This paper formalized a new selective defense to mitigating Coordinated Call and Prank Call attacks. We have shown that using state-dependent probability distributions for selecting which calls are to be processed results in high levels of availability even in the presence of a great number of attackers. The results obtained by our formal model using statistical model checking tools were very similar to the results we obtained running experiments at least for the Coordinated Call attack scenarios.

For VoIP protocols, there have been some defense proposals. For example [19] proposes a filtering mechanism for SIP flooding attacks. It is not clear whether such mechanisms will be enough for mitigating the Coordinated VoIP attack, as the number of messages needed to carry out such attack is much less, a feature of ADDoS. Wu *et al.* [20] have proposed mechanism to identify intruders using SIP by analyzing the traffic data. Although we do not tackle the identification of intruders problem, we find it an interesting future direction.

The formalization of DDoS attacks and their defenses has been subject of other papers. For example, Meadows proposed a cost based model in [21], while others use branching temporal logics [22]. This paper takes the approach used in [15,16,23], where one formalizes the system in Maude and uses the Statistical Model Checker PVeStA to carry out analyses. While [15, 16, 23] modeled traditional DDoS attacks exploiting stateless protocols on the transport/network layers, we are modeling stateful Application Layer DDoS attacks. Moreover, the quality measures used for VoIP services under TDoS attacks, described in Section 3, are different to the quality measures considered in the previous work.

More recently [4], we proposed SeVen showing that it can be used to mitigate ADDoS attacks that exploit the HTTP protocol. This paper shows that SeVen can also be used to mitigate DDoS attacks in VoIP protocols, but in order to do so one needs state-dependent probabilistic distributions. This is because of the quality requirements that we need in VoIP communications. We would like to give a priority to the types of call that should be given more chances to keep using resources of the server. In particular, we give preference to calls that do not take more than the average duration time. Such quality measures are not present in HTTP protocols that we analyzed in [4].

For future work, we are currently implementing controlled experiments with the Prank Call attack carried out on the network. We expect that these experiments also validate our simulation results for this attack. We are also thinking

on intrusion detection mechanisms. We are also interested in building defenses for mitigating amplification attacks [13]. We have also been using SeVen for mitigating High-Rate ADDoS attacks using Software Defined Networks [24].

Acknowledgments This work was supported by the Hessian excellence initiative LOEWE at the Center for Advanced Security Research Darmstadt (CASED), by RNP, by Capes and CNPq.

References

1. Cyber threat bulletin: Boston hospital TDoS attack. http://voipsecurityblog.typepad.com/files/cyber-threat-bulletin-13-06-boston-hospital-telephony-denial-of-service-attack.pdf. Accessed: 2015-27-09.
2. TDoS- extortionists jam phone lines of public services including hospitals. https://nakedsecurity.sophos.com/pt/2014/01/22/tdos-extortionists-jam-phone-lines-of-public-services-including-hospitals/. Accessed: 2015-27-09.
3. Situational advisory: Recent telephony denial of services (tdos) attacks. http://voipsecurityblog.typepad.com/files/ky-fusion_tdos_3-29-13-2.pdf/. Accessed: 2015-27-09.
4. Yuri Gil Dantas, Vivek Nigam, and Iguatemi E. Fonseca. A selective defense for application layer ddos attacks. In *JISIC 2014*, pages 75–82, 2014.
5. *The Surging Threat of Telephony Denial of Service Attacks*, (accessed Setember 28, 2015). http://voipsecurityblog.typepad.com/files/tdos_paper_4-11-13.pdf.
6. *TDoS extortionists jam phone lines of public services, including hospitals*, (accessed November 28, 2015). https://nakedsecurity.sophos.com/2014/01/22/tdos-extortionists-jam-phone-lines-of-public-services-including-hospitals/.
7. Saman Taghavi Zargar, James Joshi, and David Tipper. A survey of defense mechanisms against distributed denial of service (DDoS) flooding attacks. *IEEE Communications Surveys and Tutorials*, 15(4):2046–2069, 2013.
8. Koushik Sen, Mahesh Viswanathan, and Gul Agha. On statistical model checking of stochastic systems. In *CAV*, pages 266–280, 2005.
9. Musab AlTurki and José Meseguer. Pvesta: A parallel statistical model checking and quantitative analysis tool. In *CALCO*, pages 386–392, 2011.
10. Marcílio O. O. Lemos, Yuri Gil Dantas, Iguatemi E. Fonseca, and Vivek Nigam. A selective defense for mitigating coordinated call attacks. Submitted.
11. Seven https://github.com/ygdantas/SeVen.git. 2013.
12. Session initiation protocol. http://www.ietf.org/rfc/rfc3261.txt.
13. Ravinder Shankesi, Musab AlTurki, Ralf Sasse, Carl A. Gunter, and José Meseguer. Model-checking DoS amplification for VoIP session initiation. In *ESORICS*, pages 390–405, 2009.
14. Sanjeev Khanna, Santosh S. Venkatesh, Omid Fatemieh, Fariba Khan, and Carl A. Gunter. Adaptive selectiveverification. In *INFOCOM*, pages 529–537, 2008.
15. Jonas Eckhardt, Tobias Mühlbauer, Musab AlTurki, José Meseguer, and Martin Wirsing. Stable availability under denial of service attacks through formal patterns. In *FASE*, pages 78–93, 2012.
16. Jonas Eckhardt, Tobias Mühlbauer, José Meseguer, and Martin Wirsing. Statistical model checking for composite actor systems. In *WADT*, pages 143–160, 2012.

17. Gul Agha, José Meseguer, and Koushik Sen. Pmaude: Rewrite-based specification language for probabilistic object systems. *Electron. Notes Theor. Comput. Sci.*, 153(2):213–239, May 2006.
18. Manuel Clavel, Francisco Durán, Steven Eker, Patrick Lincoln, Narciso Mart´-Oliet, José Meseguer, and Carolyn Talcott. *All About Maude: A High-Performance Logical Framework*, volume 4350 of *LNCS*. Springer, 2007.
19. Felipe Huici, Saverio Niccolini, and Nico D'Heureuse. Protecting sip against very large flooding dos attacks. In *GLOBECOM'09'*, pages 1369–1374, Piscataway, NJ, USA, 2009. IEEE Press.
20. Yu-Sung Wu, Saurabh Bagchi, Sachin Garg, Navjot Singh, and Tim Tsai. Scidive: A stateful and cross protocol intrusion detection architecture for voice-over-ip environments. In *DSN'04*, pages 433–, Washington, DC, USA, 2004. IEEE Computer Society.
21. Catherine Meadows. A formal framework and evaluation method for network denial of service. In *CSFW*, pages 4–13, 1999.
22. Ajay Mahimkar and Vitaly Shmatikov. Game-based analysis of denial-of-service prevention protocols. In *CSFW*, pages 287–301, 2005.
23. Musab AlTurki, José Meseguer, and Carl A. Gunter. Probabilistic modeling and analysis of dos protection for the asv protocol. *Electr. Notes Theor. Comput. Sci.*, 234:3–18, 2009.
24. Joao Henrique, Iguatemi E. Fonseca, and Vivek Nigam. Mitigating high-rate application layer ddos attacks in software defined networks. Submitted.