Journal of Logical and Algebraic Methods in Programming 94 (2018) 45–67

Contents lists available at ScienceDirect

Journal of Logical and Algebraic Methods in Programming

www.elsevier.com/locate/jlamp

On the accuracy of formal verification of selective defenses for TDoS attacks

Marcilio O.O. Lemos [a], Yuri Gil Dantas [b], Iguatemi E. Fonseca [a], Vivek Nigam [c,d,*]

[a] Federal University of Paraíba, João Pessoa, Brazil
[b] Technische Universität Darmstadt, Darmstadt, Germany
[c] Federal University of Paraíba, João Pessoa, Brazil
[d] fortiss, Munich, Germany

ARTICLE INFO

Article history:
Received 30 November 2016
Received in revised form 31 July 2017
Accepted 12 September 2017
Available online 22 September 2017

ABSTRACT

Telephony Denial of Service (TDoS) attacks target telephony services, such as Voice over IP (VoIP), 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 previous work, we proposed to use selective strategies for mitigating HTTP Application-Layer DDoS Attacks demonstrating their effectiveness in mitigating different types of attacks. Developing such types of defenses is challenging as there are many design options, *e.g.*, which dropping functions and selection algorithms to use. Our first contribution is to demonstrate both experimentally and by using formal verification that selective strategies are suitable for mitigating TDoS attacks. We used our formal model to help decide which selective strategies to use with much less effort than carrying out experiments. Our second contribution is a detailed comparison of the results obtained from our formal models and the results obtained by carrying out experiments. We 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 line systems [1,2] and systems for emergency lines (like the American 911 system) [3]. According to the FBI, 200 TDoS attacks have been identified only in 2013 [2].

This paper investigates the use of *selective defenses* [4] for mitigating one type of TDoS attack called Coordinated Call [5] attack. 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 Bob and trying to stay in the call as long as she can. Since the server allocates resources for each call, by using enough pairs of attackers, attackers can exhaust the resources of the server denying service to honest participants. This is a simple,

* Corresponding author.
E-mail addresses: marciliolemos@ci.ufpb.br (M.O.O. Lemos), dantas@mais.informatik.tu-darmstadt.de (Y.G. Dantas), iguatemi@ci.ufpb.br (I.E. Fonseca), vivek.nigam@gmail.com (V. Nigam).

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

https://doi.org/10.1016/j.jlamp.2017.09.001
2352-2208/

but ingenious attack, as only a relatively low rate of incoming calls 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.

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 targeting web-servers. We formalized different attack scenarios in Maude [6] and since our strategies are constructed over some probability functions, we used statistical model checking [7], namely PVeStA [8], 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. Specifying scenarios using formal verification amounts to coding some few hundred lines of specification, while carrying out such experiments on the network amounts to building complex prototypes to carry out attacks, generate legitimate traffic and deploy defenses, integrating them with existing machinery, such as VoIP servers, setting and configuring the network and testing which take much more effort involving a larger team. 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 provides more evidence supporting the claim that formal methods can help specifiers in designing selective defenses. We systematically consider a number of selective defenses used for mitigating TDoS attacks. We compare the results obtained using our formal specification and the results obtained implementing such defenses and carrying out experiments on the network. Our results show a high accuracy for most of the results, specially on availability, but less accurate on results involving time measurements.

Our contributions are three-fold:

- We formalized in Maude the Coordinated Call attack and three selective defenses based on SeVen: the first using a uniform selection strategy, the second with roulette selection strategy [9], and the third with a tournament selection strategy [10]. We also considered two models for legitimate call duration: an exponential call duration which models traditional telephony [11] and lognormal call duration which models VoIP telephony [12].
  We carried out a number of simulations using PVeStA to test the efficiency of each version the defense used under the two different assumptions on call duration. Our simulation results suggest that SeVen mitigates the Coordinated Call attack;
- We implemented the different selective defenses analyzed using our formal models, and integrated them with the VoIP server Asterisk [13] using the SIP-protocol. We also implemented the Coordinated Call attack. Our experimental results demonstrate in practice that our selective defenses can mitigate the Coordinated Call attack;
- Finally, we compare the results obtained from our formal analysis with the results obtained from our experimental results to analyze the accuracy of the results obtained from our formal analysis. This comparison demonstrates that formal methods are of great value as they can be used early on to develop and evaluate new defense mechanisms for mitigating TDoS attacks with much less effort than implementing defenses and carrying out experiments on the network.

A small subset of experimental and simulation results appearing in this paper appeared in our previous work [14,15] which only considered scenarios where call duration followed a uniform probability and a single mechanism for dropping calls, namely the roulette strategy. This paper extends our previous work by considering different assumptions on call duration, namely lognormal distribution, modeling usual VoIP calls, and exponential distribution, modeling usual telephony, *i.e.*, non-VoIP calls. Moreover, we consider here different mechanisms for dropping calls, namely uniform, roulette and tournament dropping strategies. In terms of total time of experiments, the results in this paper add more than 40 hours of experimental results when compared to the results in our previous work [14,15].

This paper is organized as follows. Section 2 we review the Session Initiation Protocol (SIP) used for initiating a VoIP call and also explain the Coordinated Call attack. Section 3 describes how SeVen works, while Section 4 details its formalization in Maude. Sections 5 and 6 contain our simulation and experimental results including our main assumptions, results and discussion of the results obtained and Section 7 discusses the accuracy of our simulation results. We discuss in Section 8 related and future work. The implementation used to carry out our simulations is available for download at [16].

2. VoIP protocols and the coordinated call attack

We now review the Session Initiation Protocol [17], which is one of the main protocols used to establish Voice over IP (VoIP) connections. Fig. 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.

[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 not modeled in our formal specification. Moreover our experiments suffered from interference that cannot be controlled, such as network latency. The same is true for our results involving the VoIP server Asterisk used in our experimental results.

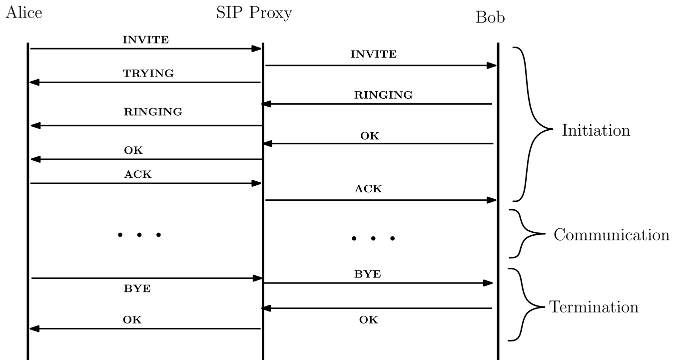


Fig. 1. Exchange of messages between the server and two users (Alice and Bob) during a normal execution of the SIP protocol.

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. The server waits for a RINGING message from Bob indicating that Bob's telephone is ringing. Bob might reject the request, in which case the server informs Alice (not shown in the Figure), or accept the call by sending the message OK. Finally, the server sends the message OK to Alice who sends an ACK message back to the server which forwards it to Bob.

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 Fig. 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.

Coordinated VoIP attack [5] 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. Many VoIP servers have an upper-bound on the number of simultaneous calls they can handle. If enough 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 follow correctly the SIP protocol and, moreover, there is no need to generate a large burst of calls, *but rather place calls in a moderate pace. Eventually, the server's capacity will be exhausted.*

There are many reasons why VoIP devices participate in a Coordinated Call attack. Pairs of legitimate users may be unsatisfied with the VoIP provider and participate in such attacks. Attackers may also use botnets with some infected malware. There has been evidence of the use of botnets in 2007 [19]. Tools that can place and receive calls (SIPp [20]) and tutorials on the Internet help automate the steps for carrying out the Coordinated Call attack.

Indeed, we have done so and as we demonstrate in Section 6, the Coordinated Call attack can reduce considerably availability to levels around 5% without generating large amount of traffic.

3. Selective strategies

We proposed in our previous work [4] a new defense mechanism, called SeVen, for mitigating Application-Layer DDoS attacks (ADDoS) using selective strategies [21]. An application using selective strategies 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. Normally, this internal buffer reflects the connections of the protected service. Assume that k is the maximum capacity of the service being protected. For VoIP servers k is the number of calls that the application can handle simultaneously. If the number of messages accumulated reaches k (the size of internal buffer) and a new incoming request R arrives, SeVen behaves as follows:

1. SeVen decides whether to process R or not based on a probability P_1. P_1 is defined using the counter `PMod` following [22]:

[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 [18] for which known solutions exists. Such amplification attacks are not, however, the main topic of this paper.

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

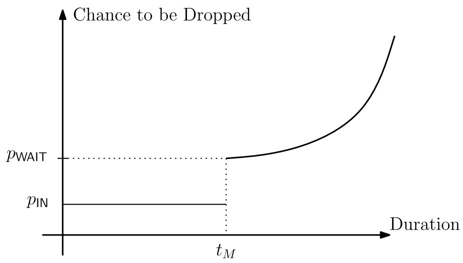


Fig. 2. Graph (not in scale) illustrating the behavior of SeVen according to the status of a call and its duration. p_{WAIT} is the probability of dropping a WAITING call, while p_{IN} the probability of dropping a INCALL call.

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

At the beginning of the round, we set $\texttt{PMod} = 0$. 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. The intuition is that P_1 reduces with the increase of incoming traffic thus reducing the impact of high numbers of request to the application;

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 temporarily unavailable.

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

SeVen mitigates attacks *only when the maximum capacity of the VoIP server is reached*. When this happens, SeVen has two mechanisms for dropping requests. The first one is by using probability P_1 and the other by using P_2. The main goal of the former is to mitigate the impact of volumetric attacks [21]. This is because whenever the defense receives a high volume attack the value of PMod increases rapidly increasing rapidly the chance of dropping a request. This is also reflected on the round time T_S which is typically in the order of some hundred milliseconds to avoid PMod from reaching too high values even under normal traffic. We used $T_S = 100$ ms.

As Coordinated Calls do not generate a very large number of requests, the mechanism using P_1 is not the main mitigation mechanism used by SeVen for this attack, but rather the mechanism using P_2. There is, however, much space for specifying the probability distribution P_2 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 the Coordinated Call attack described in Section 2, we set the probability P_2, governing which call to be dropped from the internal buffer, 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 the 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 they 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 a server 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]

The dropping factor of an INCALL request increases exponentially once the call duration is greater than t_M. Fig. 2 depicts roughly the dropping factor used to drop requests. The actual function d (for drop factor) is of the form, where t is the call duration and α is a parameter:

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

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

Given this dropping factor, we consider in our analysis three ways for selecting which call to drop. Assume the server has a capacity of k simultaneous calls. Moreover, assume $c_1, \ldots, c_k$ are the calls currently being processed by the server and they have dropping factors of, respectively, $d_1, \ldots, d_k$. We considered three different selection strategies described in the literature, namely, uniform [4], roulette [9] and n-tournament [10]:

- **Uniform:** In this strategy, the dropping factor of a call is not considered. We select using uniform probability which call is going to be dropped. Thus any call independent on its duration and status can be selected to be dropped by SeVen;
- **Roulette:** In the roulette strategy [9], we select randomly a call c_i to be dropped where the probability of being dropped is proportional to its dropping factor. Thus in the roulette strategy a call c_i has twice the chance of being dropped than a call c_j if $d_i = 2 \times d_j$.
  For instance, consider $k = 4$ and that the server is serving the calls c_1, c_2, c_3, c_4 with dropping factors $2, 3, 1, 6$ respectively. We select using uniform distribution a number r between 0 and $2 + 3 + 1 + 6 = 12$. If $0 \leq r < 2$, then the call c_1 is selected, if $2 \leq r < 5$, then the call c_2 is selected, if $5 \leq r < 6$, then the call c_3 is selected, and otherwise if $6 \leq r < 12$ then the call c_4 is selected. In this way, the call c_4 has 6 times more chance to be selected than the call c_3 for example;
- ***n*-Tournament:** In the n-tournament strategy [10], we first select n calls randomly using uniform probability to be part of the *tournament*. Then, the call to be dropped will be the call with the greatest dropping factor among the n selected calls. In case there is more than one possible call with the greatest dropping factor, we select one of them at random.
  For instance, in the example above, if $n = 2$, then we would select randomly two out of the four calls c_1, c_2, c_3, c_4 to participate in the tournament. Say the calls c_2 and c_3 are chosen to be part of the tournament. In this case, the call c_2 is selected to be dropped as it has the greatest drop factor.
  Notice that if n is chosen to be too low when compared to k, the n-tournament behaves closer to the uniform dropping strategy. In fact, if $n = 1$, then the n-tournament can be shown to be equivalent to the uniform dropping strategy. On the other hand, if n is chosen to be too high, then the n-tournament behaves closer to a deterministic dropping strategy that selects the call with the greatest dropping factor. Indeed, if $k = n$, then the n-tournament strategy is deterministic. In our experiments, we used $n = k/2$, that is, a strategy between the uniform and a deterministic strategy.

While the attackers attempt to stay in a call for very long periods of time, legitimate clients do not behave so. The literature models legitimate call duration using the following distributions, where the parameters λ, σ and μ are computed accordingly to the mean call durations assumed t_M (see [23] for more details):

- **Exponential:** Typical telephony models [11], *i.e.*, not VoIP, assume that the call duration of legitimate clients follows an exponential density distribution:

$$f(x, \lambda) = \begin{cases} \lambda e^{-\lambda x} & x \geq 0 \\ 0 & x < 0 \end{cases} \tag{2}$$

  Since the coordinated call attack can also be carried out in standard telephony systems, we considered call duration following this distribution.
- **Lognormal:** While standard telephony calls are paid per duration, in VoIP calls have fixed rates or are even for free. This difference impacts legitimate call duration which in VoIP follows a lognormal density distribution [12]:

$$f(x, \mu, \sigma) = \frac{1}{x\sigma\sqrt{2\pi}} \exp\left[-\frac{(\ln(x) - \mu)^2}{2\sigma^2}\right]. \tag{3}$$

3.1. Sample execution

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

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

$\langle id, st, tm\rangle$ specifies that the call id has status st and the call started at time tm where tm is undef whenever $st = \text{WAITING}$. This buffer specifies that id_1 is waiting the responding party to answer (with a OK 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 $\langle id_1, \text{OK}\rangle$ at time 9.5 arrives specifying that the responder of the request id_1 answered the call. The buffer is updated to the following:

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

Then the message $\langle id_3, \text{INVITE}\rangle$ arrives. Since k has not yet been reached, 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 current requests id_1, id_2, id_3 have dropping factors following Fig. 2:

- id_1 has dropping factor 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 dropping factor because it is calling for twice t_M: $10.5 - 0.5 = 2 \times 5$;
- id_3 has dropping factor of p_{WAIT} as it has WAITING status.

The application decides which one to drop either using *uniform probability*, in which case the dropping factor of requests is not considered, or the *roulette strategy*, in which case id_2 has a greater probability of being dropped, or the n-tournament strategy in which case it would depend on n.

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 [4,24,25] by specifying test scenarios using actors where attackers, clients, and the 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 the clients.

We formalize all actors in Maude [6] and carry out simulations by using the statistical model checker PVeStA [8]. For 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 as in practice SeVen and the VoIP server are in the same machine and thus share a quick communication channel. In the following, we describe our Maude specification. The complete model can be found in [16].

We refer to [6,26] for a more detailed description of Maude and its underlying foundations on Rewriting Logic.

4.1. Key sorts and functions

Actor. The elements of the sort `Actor` is constructed by the operator

```
op <name:_|_> : Address AttributeSet -> Actor .
```

which takes an `Address`, which can be a string, and a set of attributes, `AttributeSet`. In our formalization, we use the following attributes:

```
op req-cnt:_ : Float -> Attribute .
op b-set:_ : NBuffer -> Attribute .
op server:_ : Address -> Attribute .
op status:_ : Status -> Attribute
```

The attribute `req-cnt` stores the value of `PMod`, `b-set` stores the internal buffer of the server of sort `NBuffer` and `server` stores the address of the server. Finally, the attribute `status` specifies the status of the call which may be any of the following state constants:

- `none` – a call that has not yet been placed;
- `invite` – a call that has been placed and is waiting for the responder to answer;
- `incall` – a call where the participants are currently communicating;
- `complete` – a call that has been completed, *i.e.*, the parties have communicated for the expected time;
- `incomplete` – a call that was interrupted while communicating by SeVen.

The SeVen buffer has sort `NBuffer` and is constructed by pairing a number and a buffer:

```
op [_|_] : Nat Buffer -> NBuffer .
```

The number specifies the elements in the buffer which is a list of elements of sort `EleBuf`. These elements are constructed using a 3-tuple of the form:

```
op <___> : Address State Float -> EleBuf .
```

The first position stores the address of the call. The second position denotes the state of the call. The third position stores the time of the first request of the call and is used to compute the call duration.

Messages. Actors process messages of sort `Msg` which are constructed using the following operator

```
op _<-_ : Address Contents -> Msg .
```

The first parameter of sort `Address` specifies to which actor the message is directed and the second parameter of sort `Contents` is the payload of the message. An `ActiveMsg` is a timestamped `Msg` constructed using the following operator:

```
op {_,_} : Float Msg -> ActiveMsg .
```

The first parameter specifies the time when the paired message is to be processed.

We assume that an active message `{gt1,msg1}` is always going to be processed before an active message `{gt2,msg2}` whenever `gt1 < gt2`. This is accomplished by using a Scheduler as in [4,24,25]. A scheduler has sort `Scheduler` and is constructed by the following operator

```
op [_|_] : Float ActiveMsgList -> Scheduler .
```

The first parameter is the global time while the second parameter contains the list of active messages ordered by their delivery time. The following function returns the scheduler obtained by inserting at the correct position, *i.e.*, according to the messages timestamp, a list of active messages into a given scheduler.

```
op insert : Scheduler ActiveMsgList -> Scheduler .
```

Configuration. A configuration of sort `Config` is a collection of `Actors` and `Scheduler`:

```
subsort Actors < Config .
subsort Scheduler < Config .
op __ : Config Config -> Config [assoc comm id: none] .
```

For example, the initial configuration is defined by the equation:

```
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 >
        [0.0 | {0.0, (attacker-generate <- spawn )} ;
               {0.0, (client-generate <- spawn )} ;
               {Ts, server <- ROUND}] .
```

It specifies a configuration with three actors: an attacker generator, a client generation and a server. The global time is `0.0` and there are three active messages in the scheduler, the first two directed to the actors `attacker-generate` and `client-generate`, respectively. Here we follow the Shared Channel Model [21] where clients and attackers share the same channel. Thus the application does not distinguish malicious traffic from legitimate one as it is usually the case, in particular, for the Coordinated Call attack.

Intuitively, the user specifies the rate at which new client and attacker actors are created. Then, for instance, when the message `client-generate` is processed a new client actor is created and a new message `client-generate` is scheduled so that a new client actor is created according to the rate given. Similarly, when processing `attacker-generate` which creates a new attacker actor and a new `attacker-generate` is scheduled to be processed according to the attacker generating rate. These rewrite rules are omitted.

The third message scheduled at the time `Ts` is directed to the `server` which is implementing the SeVen strategy establishing when a SeVen round ends.

The following function extracts the first scheduled active message in a scheduler and returns a new scheduler with the global time advanced to its delivery time.

```
 op mytick : Scheduler -> Config .
```

For instance, let `msg` be a message and `SL` an `ActiveMsgList`. Then

```
  mytick([0.0 | {1.0,msg} ; SL])
```

returns `msg [1.0 | SL]` containing the message `msg` and the scheduler `[1.0 | SL]`. Intuitively, the message `msg` is going to be processed next. Message processing is formalized by rewriting rules.

Selective strategies. Finally, we specified the three selective strategies described in Section 3, namely, Uniform, Roulette and Tournament. The function

```
op select : Float Buffer -> ActorBuffer .
```

implements one of the selective strategies. Given the global time and a buffer, this function returns a pair of sort `ActorBuffer` with the actor name of the selected element that has been selected to be removed and the resulting buffer obtained by removing this element from the given buffer.

All selective strategies we formalized use the following function:

```
op sampleUniWithInt : Nat -> Nat
```

which for a given input n returns a random natural number between 0 and n. The following function uses this function to select at random an element from a given buffer and thus to implement the uniform selection strategy.

```
op selectRandom : Buffer -> EleBuf .
```

For the roulette strategy, we compute using the dropping factor

```
op roulette : Float Buffer -> EleBuf .
```

which creates a roulette by assigning weights to the elements of the buffer according to the dropping factor and then randomly selects one.

A tournament for the n-tournament selection strategy is created by the following function:

```
op creatingTour : NBuffer Nat Buffer -> Buffer .
```

It takes an NBuffer and a natural number, specifying the size of the tournament, and accumulates the selected elements to the tournament in the third argument. Once the tournament is created, we use the following function to select the one with the greatest dropping factor:

```
op selectGreatest : Buffer -> EleBuf .
```

which takes a buffer with the tournament traversing it to find the element with the greatest dropping factor.

4.2. Rewrite rules

The rewrite rules modify elements from `Conf` and specify the operational semantics of a system. We describe next the main rewrite rules used in our formalization.

The first action we describe is when an actor receives a `poll` message indicating that it should start a call at time `gt + delay`.

```
rl [CLIENT-RECEIVE-POLL] :
  <name: c(i) | server: Ser, status: none, AS >
  {c(i) <- poll} [gt | SL]
 =>
  <name: c(i) | server: Ser , status: invite, AS >
  mytick(insert([gt | SL], { 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 *connected* and generates a message BYE, scheduled to be sent after some time. This means that all legitimate clients do not overpass the average time of the duration of calls using one of the call duration models, exponential or lognormal, described in Section 3. This means that the client called `c(i)` is expected to end its call at time `gt + callDur(tMedio)`.

```
crl [SeVen-RECEIVE-INVITE] :
    <name: Ser | req-cnt: pmod , b-set: [lenB | B],  AS >
    {Ser <- INVITE(Actor)} [gt | SL]
 => if (lenB >= lenBufSeVen) then
          if p1 then ConfAcc myTick(SchAcc)
                else ConfRej myTick(SchRej)
          fi
    else ConfAcc2 mytick(SchAcc2)
    fi
if p1 := sampleBerWithP(accept-prob(pmod))
 /\ { ActorDr, bufDr } := select(gt,B)
 /\ nBuf := add([lenB + (- 1) | bufDr], < Actor invite gt >)
 /\ ConfAcc := <name: Ser | req-cnt: (pmod + 1.0), b-set: nBuf , AS >
 /\ SchAcc := insert([gt | SL],
                     {gt, Actor <- TRYING} ; {gt, ActorDr <- poll})
 /\ ConfRej := <name: Ser | req-cnt: (pmod + 1.0), b-set: [lenB | B], AS >
 /\ SchRej := insert([gt | SL],
                     {gt + delay , Actor <- poll})
 /\ b-setNu := add( [lenB | B], < Actor invite gt > )
 /\ ConfAcc2 := <name: Ser | req-cnt: pmod , b-set: b-setNu, AS >
 /\ SchAcc2 := insert([gt | SL], {gt + delay, Actor <- TRYING}) .

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

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

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

SeVen may, however, drop a call before the call is finished. We classify such a call as an incomplete call, *i.e.* the dropped client's status is changed to `incomplete`. We omit this rule.

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.

Fig. 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 is selected to be dropped using the function `select`. It returns the name of the actor `ActorDr` and the resulting buffer `nBuf`. The incoming request is added to `nBuf` and `pmod` gets incremented, resulting in the configuration `ConfAcc`. Moreover, a `poll` message to `ActorDr` and a `TRYING` to `Actor` are created specifying that the connection is going to be terminated. 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 `WAITING` requests in SeVen's buffer are answered by the function `reply`. Then a new round starts and `pmod` is re-set.

5. Simulations

We detail our simulation results obtained from our formal specification using the statistical model checker PVeStA [8]. Our simulations are parametric in the following values:

- **Average Time of a Call –** t_M: This is the assumed average time of the calls of honest users. For the simulations, we assumed $t_M = 5$ time units;
- **Dropping Factor** – We assume the following values for the dropping factor function (Equation 1):
  - $p_{\mathsf{IN}} = 2$;

- $p_{\mathsf{WAIT}} = 8$;
- $\alpha = 1.89$.

These values were chosen so that the dropping factor increases in a reasonable fashion for calls with duration greater than t_M. Sample values are shown below, recalling that $t_M = 5$ seconds:

Call duration (mins)	Dropping factor
6	12.37
8	17.31
10	27.84

That is, dropping factor of a call with duration of 10 minutes, *i.e.*, $2 \times t_M$, is approximately 3 times greater than the dropping factor of a call whose status is WAITING (27.84/8). This is a reasonable ratio. However, according to the specific application other values can be set for p_{IN}, p_{WAIT} and α. Finally, the choice of setting $p_{\mathsf{WAIT}} = 4 \times p_{\mathsf{IN}}$ was selected so that the calls with duration less than t_M have much less chance of being dropped than the calls that are still waiting for the responder.

- **Size of Buffer –** k: This is the upper-bound on the size of the server's buffer denoting the processing capacity of the application. $k = 24$;
- **Rate of Calls (**R**)**: We fixed the total rate of calls to be R which is the result of summing the rate of legitimate calls, R_L, and the rate of attacker calls, R_A. That is, $R_L + R_A = R$.
  The value of R is computed using standard techniques[6] so that if $R_L = R$, *i.e.*, the server is not under attack, then the server will not be overloaded.
  With R fixed, we set R_L and R_A to be $R_L + R_A = R$, but considered scenarios with different proportions for R_A and R_L. This reflects the fact that Coordinated Call attack uses low traffic and therefore, can bypass usual defenses based on network traffic analysis which normally monitor the total rate of calls R.
  We considered 5 different proportions for R_L and R_A expressed in the percentage of the total number of calls R:

Legitimate calls (R_L)	Attacker calls (R_A)
83%	17%
67%	33%
50%	50%
33%	67%
17%	83%

- **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 [24];
- **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 [27,7] for more details on statistical model checking).

Quality measures. In our simulation, we use quality measures specific for VoIP services. These are specified by expressions of the QuaTEx quantitative, probabilistic temporal logic defined in [27]. We perform statistical model checking of our defense in the sense of [7]: 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 [6] and the statistical model checking is carried out by PVeStA.

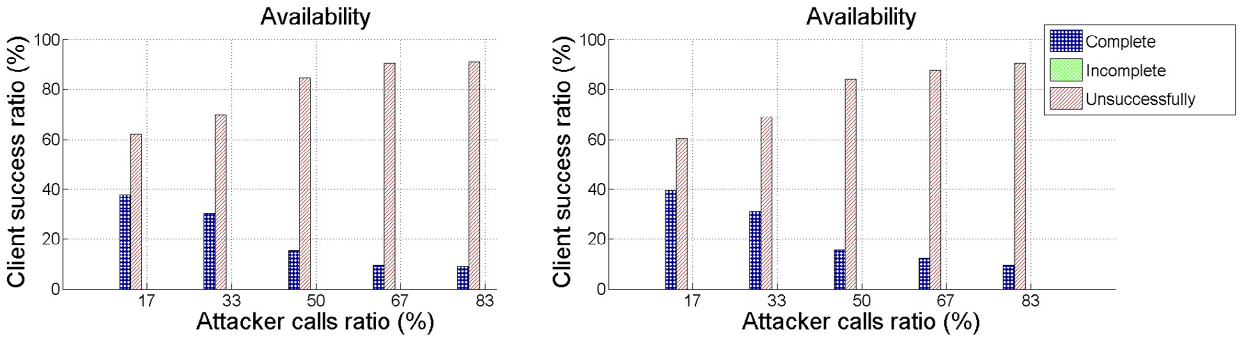
The QuaTEx formulas, *i.e.*, the quality measures, that we use in our simulations are defined below. The operator $\bigcirc$ is a temporal modality that specifies the advancement of the global time to the time of the next event (see [27] for more details).

- 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.

[6] Using the Erlang model which computes R by taking into account k and t_M.



(a) Exponential Call Duration. (b) Lognormal Call Duration.

Fig. 4. Client success ratio: simulation results when not using SeVen.

- 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.

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

  where $countUnsuccessful = countHonest - countIncall$.
- The average of client incomplete calls: We also measure the average proportion of time legitimate clients were able to talk in an incomplete call before they were dropped.

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

  where $totalTimeInCall$ is the sum of how much percent of time clients were able to talk before being interrupted and the $totalIncompleteCall$ is the total of clients that were not able to finish their call.

We carried out simulations with the three different types of dropping strategies described in Section 3, namely uniform, roulette and $\frac{k}{2}$-tournament. We also carried out simulations with a scenario without SeVen.

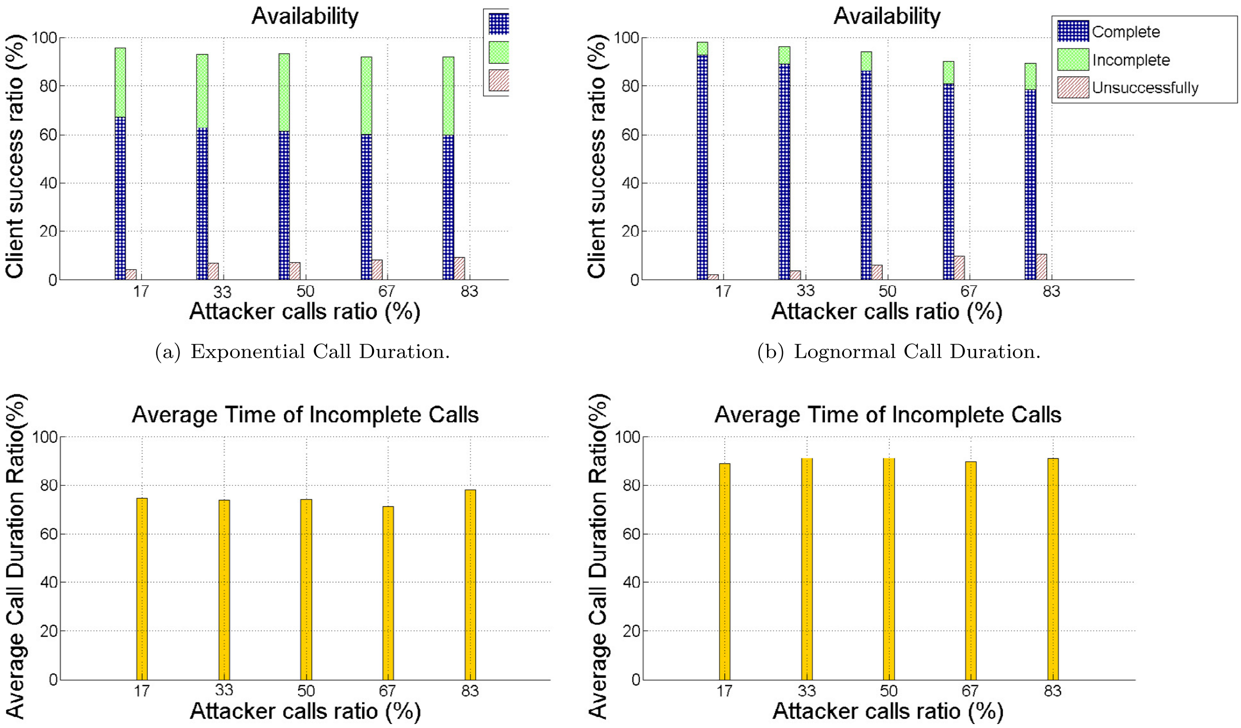
5.1. No defense

Our simulations results are depicted in Fig. 4. They suggest that the Coordinated Call attack is indeed effective in reducing the availability of a VoIP service when assuming both an exponential and a lognormal call duration. Increasing the proportion of attackers rapidly increases the proportion of Unsuccessful calls, *i.e.*, calls that did not even start a conversation, while the proportion of Complete calls falls. As expected there are no Incomplete calls as the VoIP server does not interrupt calls.

5.2. Uniform dropping strategy

Fig. 5 depicts the results when using SeVen with a uniform dropping strategy. It suggests that SeVen can indeed mitigate the Coordinated Call attack. The proportion of Complete calls remains at high levels when assuming both an exponential, above 60% of legitimate calls, and a lognormal call duration, above 80%.

As SeVen may interrupt calls in the middle of a conversation, there are Incomplete calls, *i.e.*, calls where the parties have started to communicate, but were interrupted before communicating for the expected time. For exponential call duration,



(a) Exponential Call Duration. (b) Lognormal Call Duration.

(c) Exponential Call Duration. (d) Lognormal Call Duration.

Fig. 5. Client success ratio and average time of incomplete calls: simulation results when a uniform dropping strategy.

around 30% of legitimate calls were interrupted, while for lognormal call duration around 10% of legitimate calls were interrupted. However, the average time of incomplete calls suggests that although these calls are interrupted, they still are able to communicate for long periods of time, above 70% of the expected time for exponential call duration and above 89% of the expected time for lognormal call duration.

5.3. Roulette dropping strategy

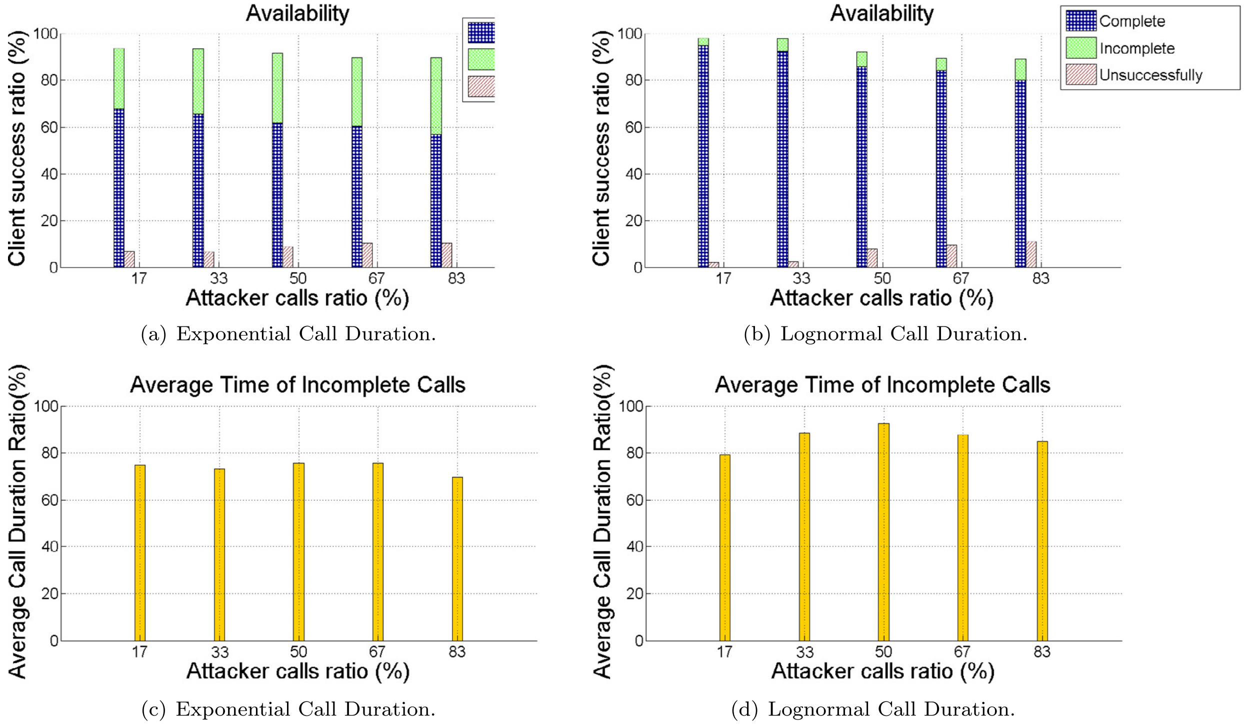
Figs. 6 depicts our simulation results when using a roulette dropping strategy. The results are similar to the results obtained with the uniform dropping strategy. For exponential call duration, above 60% of legitimate calls were completed and around 30% were interrupted by SeVen. For lognormal call duration, above 80% of legitimate calls were completed and around 10% were interrupted by SeVen. Moreover, the interrupted calls stayed communicating in average above 70% of the expected call duration for exponential call duration and above 88% of the expected call duration for lognormal call duration.

5.4. $\frac{k}{2}$-tournament

Fig. 7 depicts the simulation results obtained by using a $\frac{k}{2}$-tournament dropping strategy. They suggest that this strategy is better than the roulette and uniform strategies. The proportion of complete calls is above 62% for exponential call duration and above 86% for lognormal call duration. In other words, around 30% of legitimate calls were interrupted by SeVen when assuming exponential call duration and around 10% of legitimate calls were interrupted by SeVen when assuming lognormal call duration. Moreover, the average time of incomplete calls is always greater than 72% for exponential call duration and above 84% for lognormal call duration.

6. Experiments

In our experiments, we used Asterisk version 13.6.0 which is a SIP server widely used by small and mid size companies for implementing their VoIP services. We assume there are honest users and malicious attackers which try to make the VoIP unavailable. Both the traffic of the honest users and the attackers are emulated using the tool SIPp [20] version 3.4.1. SIPp generates calls which may be configured as the caller or the callee. Thus, in our experiments, we used pairs of SIPp, one pair for generating the honest user calls and the other pair for generating the attacker calls. Finally, we developed the SeVen proxy in C++ which implements the selective strategy described in Section 3.



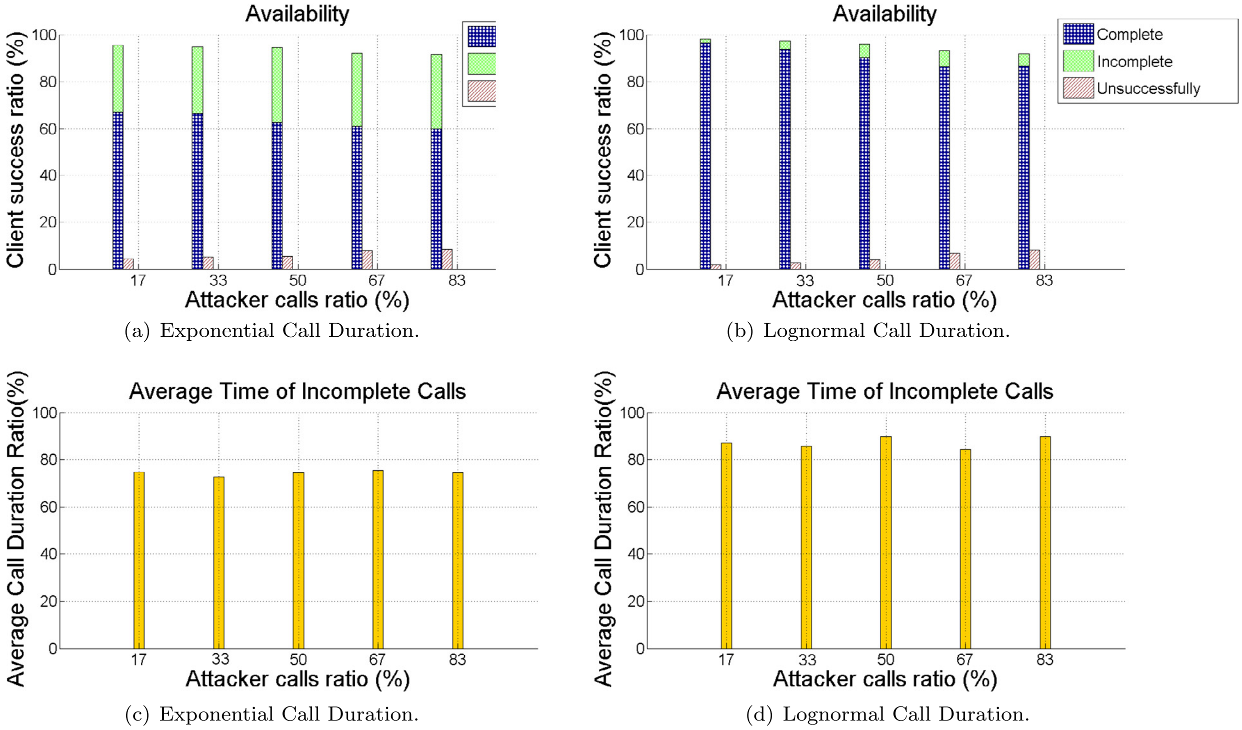
(a) Exponential Call Duration.

(b) Lognormal Call Duration.

(c) Exponential Call Duration.

(d) Lognormal Call Duration.

Fig. 6. Client success ratio and average time of incomplete calls: simulation results when a roulette dropping strategy.



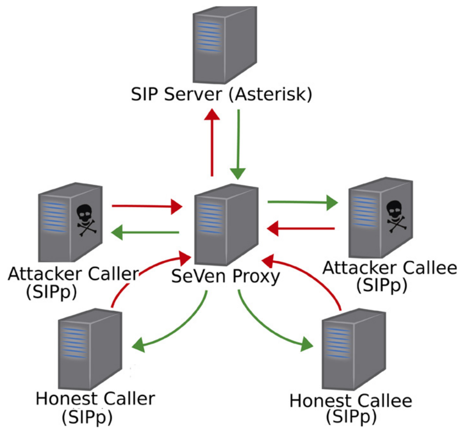
(a) Exponential Call Duration.

(b) Lognormal Call Duration.

(c) Exponential Call Duration.

(d) Lognormal Call Duration.

Fig. 7. Client success ratio and average time of incomplete calls: simulation results when using a tournament strategy.

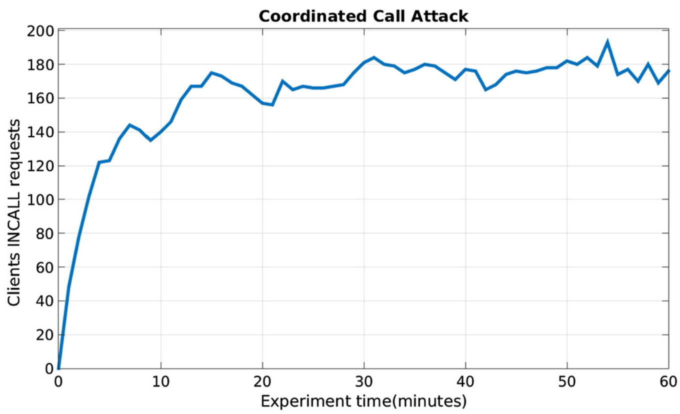


The figure above illustrates the topology of the experiments we carried out. To make a call, the pairs of SIPp send messages to the SeVen proxy which on the other hand forwards them to Asterisk. Similarly, any message generated by Asterisk is forward to the SeVen proxy which then forwards them to the corresponding users. Therefore, SeVen is acting as an *Outbound Proxy* for both Asterisk and the pairs of SIPp. For our experiments, it is enough to use a single machine. We used a machine with configuration Intel(R) Core(TM) i7-4510U CPU @ 2.00 GHz and 8 GB of RAM.

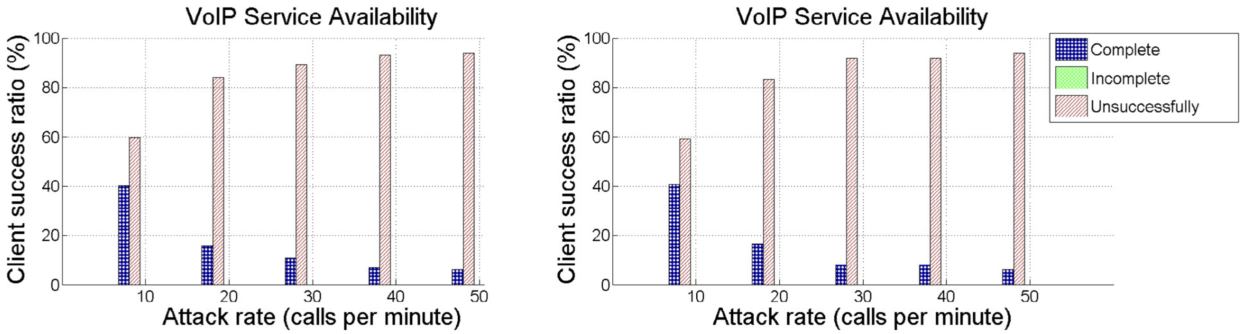
Parameters. We use the following parameters to configure our experiments:

- **Average Call Duration** (t_M) – We assume known what is the average duration of calls. This can be determined in practice by analyzing the history of calls. We assume in our experiments that $t_M = 160$ seconds (approximately 2.6 minutes);
- **SIP Sever Capacity** (k) – This is the number of simultaneous calls the SIP server can handle. We set $k = 200$ which is a realistic capacity for a small company allowing 400 users (2×200) to use the service at the same time.
- **Experiment Total Time** (T) – Each one of our experiments had a duration of 60 minutes which corresponds to 3600 calls in each experiment. With this duration, it was already possible to witness the damage caused by the Coordinated Call Attack as well as the efficiency of our solution for mitigating this attack.
- **Traffic Rate** (R) – Using a server with capacity of k, we calculated using standard techniques [23] what would be a typical traffic of such a server. It is $R = 60$ calls per minute. This value is computed using traditional techniques [23] (Erlang model) taking into account $k = 200$ and $t_M = 160$. Thus the service can handle R legitimate calls per second. However, as the attacker does not behave as legitimate placing calls with much greater durations, the server can be subject to this attack.
  In our experiments, we split this rate among clients and attackers. This is because we want to emulate the fact that Coordinated Call attack can deny service using low traffic thus bypassing usual defenses [28–33] based on network traffic analysis or monitoring the number of incoming calls. This means that the total traffic (attacker + client) is always less or equal than R.

The following graph illustrates client usage in normal conditions, *i.e.*, without suffering an attack, using the parameters as specified above:



It shows that the server is indeed well dimensioned for this rate of (legitimate) calls. Clients occupy in average approximately 85% of Asterisk's capacity thus not overloading the server, but still being heavily used.



(a) Exponential Call Duration. (b) Lognormal Call Duration.

Fig. 8. Client success ratio: experimental results when not using SeVen.

Finally, we set the duration of the calls generated by SIPp as follows:

- **Total Call Duration of Clients:** As described in Section 3, we used two models for the call duration of legitimate client calls, namely the exponential model suitable for non-VoIP calls and the lognormal model suitable for VoIP calls. The parameters, λ, μ, σ, in Equations 2 and 3 were computed as described in [11,12] using the average call duration t_M. Whenever we generate a new call, we generate the call duration randomly according to the used model (exponential or lognormal). SIPp ends the call when its corresponding call duration is reached.
- **Call Duration of Attackers:** Following the Coordinated Call Attack, we do not limit the call duration of an attacker call. His calls communicate for indefinite time.

Quality measures. For our experiments, we used the following three quality measures for our calls:

- **Complete Call:** A call is complete whenever its status changed from WAITING to INCALL and *it is able to stay in status* INCALL *for its corresponding call duration*. That is, the caller was able to communicate with the responder for all the prescribed duration;
- **Incomplete Call:** A call is incomplete whenever its status changed from WAITING to INCALL, but *it was not able to stay in status* INCALL *for its corresponding call duration*. That is, the caller was interrupted before completing the call;
- **Unsuccessful Call:** A call is unsuccessful if it did not even change its status from WAITING to INCALL. That is, the caller did not even have the chance to speak with the responder.

Intuitively, complete calls are better than incomplete calls which are better than unsuccessful calls. In order to support this claim, we also computed the average duration call of the incomplete calls, that is, the time that users in average were able to stay communicating before they were interrupted by SeVen.

6.1. Experimental results

We carried out the corresponding experiments to the scenarios used in Section 5. That is, we tested the efficiency of the Coordinated Call attack when the server is not running any defense. We also carried out experiments with scenarios using an exponential and lognormal call duration with the uniform, roulette and 100-tournament dropping strategies.

6.1.1. No defense

Figs. 8 and 9 illustrate our main results when assuming exponential and lognormal call duration and not using any defense mechanism. Our results demonstrate the efficiency of the Coordinated Call attack. We observed that the VoIP availability decreases considerably when increasing the proportion of attacker calls in the rate R (Fig. 8). In particular, the number of unsuccessful call increases to level near 100%, while the number of completed calls falls to close to 0%. Moreover, there are no incomplete calls, which is expected since no calls are interrupted.

We also measured the number of attacker calls that the server serves during the experiment (Fig. 9). As expected from the profile of the Coordinated Call attack, the attacker is able to deny service by slowly (after 10 minutes) occupying all the available calls in the server and therefore deny its service to legitimate clients.

6.1.2. Uniform defense

We carried out experiments to test the efficiency of SeVen when using a uniform dropping strategy. Our results are summarized in Figs. 10 and 11.

The graphs depicted in Fig. 10 show that SeVen when using a uniform dropping strategy can mitigate the Coordinated Call attack. The results assuming a lognormal call duration is slightly better than the results when assuming an exponential

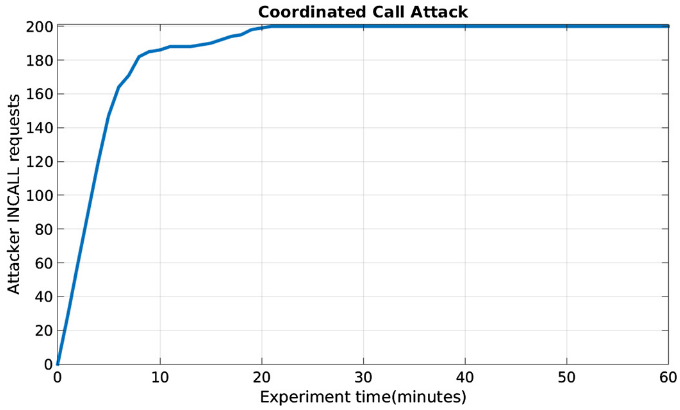
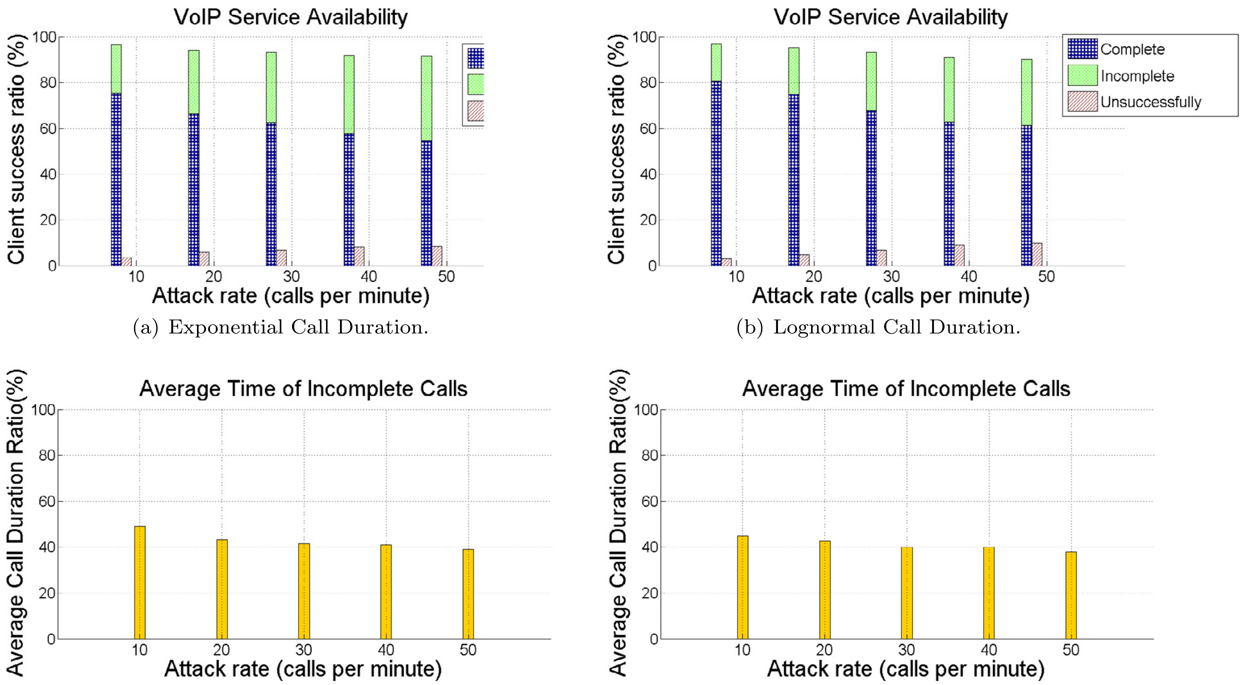


Fig. 9. Attacker call occupancy in buffer: experimental results when not using SeVen.



(a) Exponential Call Duration. (b) Lognormal Call Duration.

(c) Exponential Call Duration. (d) Lognormal Call Duration.

Fig. 10. Client success ratio and average time of incomplete calls: experimental results when SeVen and a uniform dropping strategy.

call duration. In both cases, the proportion of completed calls stayed above 50% levels even when the attacker call rate is 5 times more than the client call rate. The proportion of incomplete calls was more affected by the call duration model. Under the exponential call duration assumption, the proportion of incomplete call was around 35% of all legitimate calls, while under the lognormal call duration assumption, the proportion of incomplete calls was of around 28%. The proportion of unsuccessful calls stays below 10% under both assumptions of call duration. We also measured the average time of incomplete calls, that is, the proportion of time that incomplete calls were able to stay in a call before they were dropped by the SeVen defense strategy. When assuming both an exponential call duration and a lognormal call duration, the incomplete calls were in average around 40% of the expected call time.

Fig. 11 illustrates how the attacker is able to occupy the resources of the server. While when not running SeVen the attacker was able to occupy all the server's resources (Fig. 9), when using SeVen with a uniform dropping strategy, the attacker is only able to occupy around 70% of the server's resources. This may seem to be a high value, but the graph hides the fact that attackers are dropped by the defense strategy and thus the high levels of availability obtained.

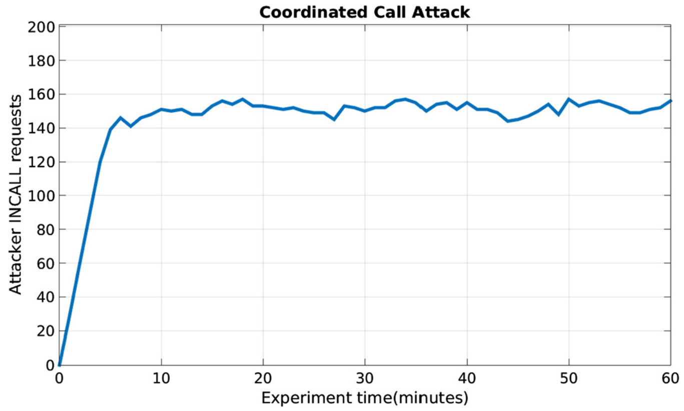
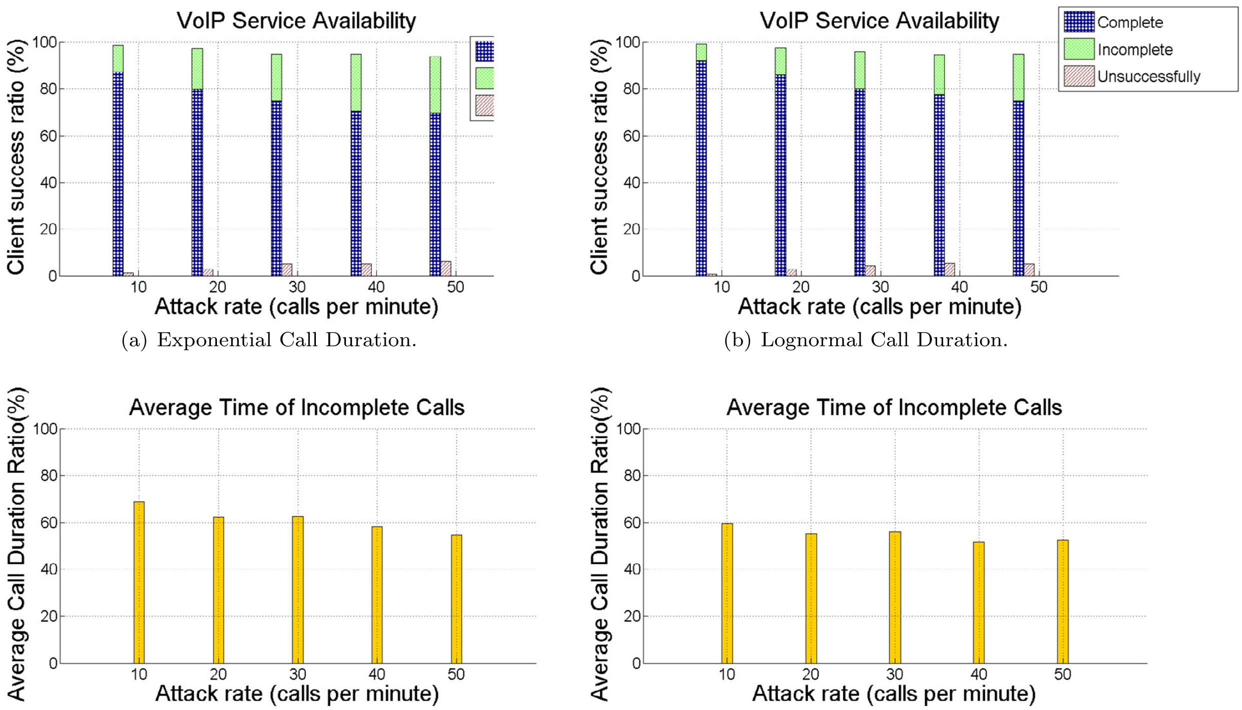


Fig. 11. Attacker call occupancy in buffer: experimental results when using SeVen with a uniform dropping strategy.



(a) Exponential Call Duration.

(b) Lognormal Call Duration.

(c) Exponential Call Duration.

(d) Lognormal Call Duration.

Fig. 12. Client success ratio and average time of incomplete calls: experimental results when using SeVen with a roulette dropping strategy.

6.1.3. Roulette defense

Figs. 12 and 13 depict our experimental results when using SeVen with the roulette dropping strategy. As with the uniform dropping strategy, the defense was able to mitigate the Coordinated Call attack. Furthermore, the roulette strategy performed better than the uniform strategy.

The availability depicted in Fig. 12 remained at high levels under both assumptions on call duration (exponential and lognormal). The proportion of completed calls was above 70% percent when assuming an exponential call duration and above 75% when assuming a lognormal call duration. In both cases, the proportion of incomplete calls was less than 6%. We also measured the average time of incomplete calls. They show that these calls were able to communicate for more than 50% of the expected time. These results are better than the results obtained using a uniform dropping strategy.

Despite the availability results using the roulette strategy being better than the availability results obtained using the uniform strategy, the attacker was still able to occupy a similar proportion of the server's resource as depicted in Fig. 13. It occupied at most 70% of the server's resources. Intuitively, the difference in the availability between the uniform and roulette strategies is because the roulette strategy tends to drop calls with greater duration. This means that the attacker calls are more likely to be selected leaving more chance for a legitimate call to access the service.

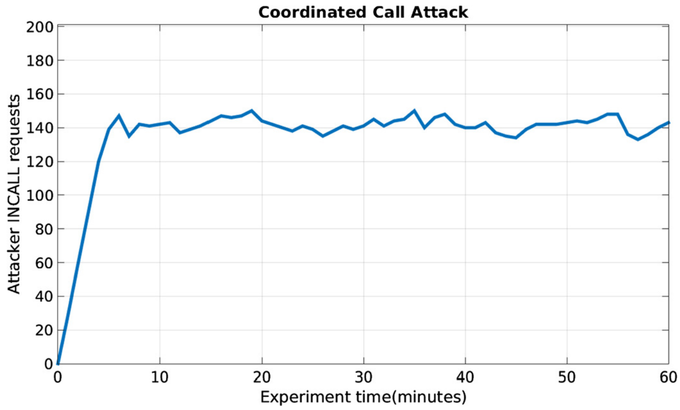


Fig. 13. Attacker call occupancy in buffer: experimental results when using SeVen with a roulette dropping strategy.

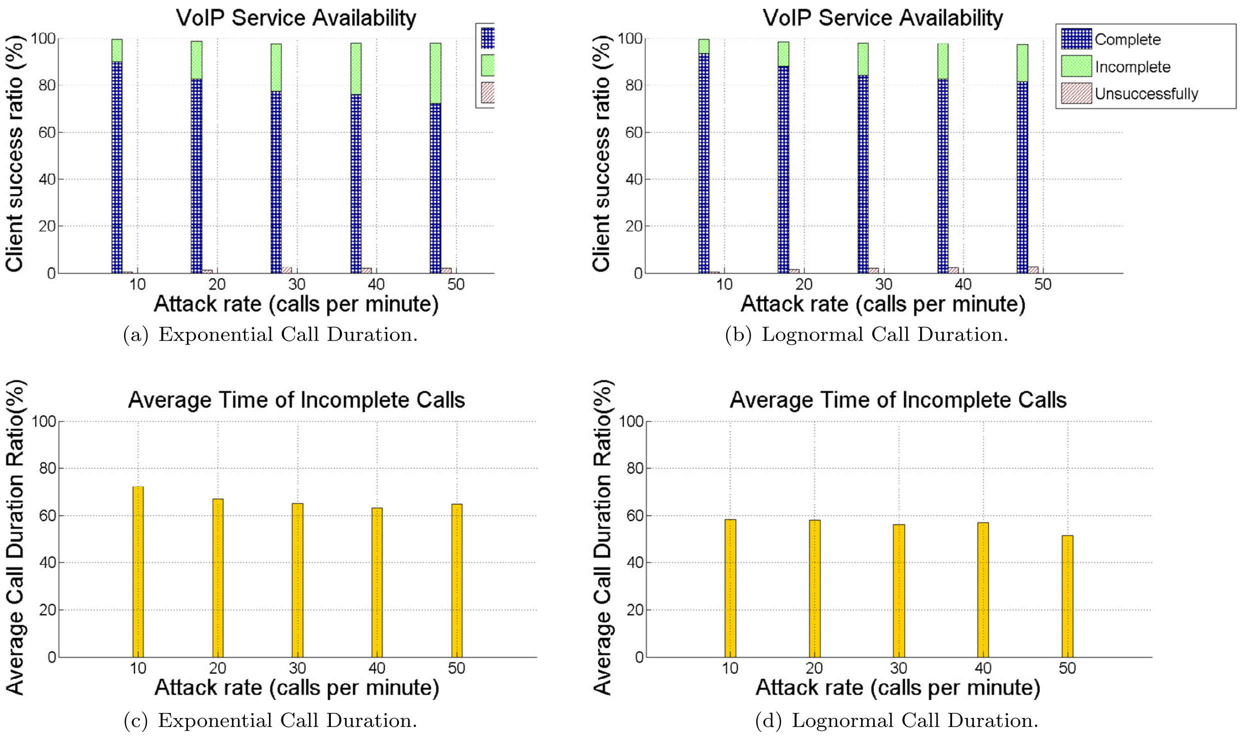


Fig. 14. Client success ratio and average time of incomplete calls: experimental results when using SeVen with a 100-tournament dropping strategy.

6.1.4. 100-tournament defense

Our last set of experiments evaluated the efficiency of SeVen with a 100-tournament dropping strategy. Figs. 14 and 15 depict our main results. The 100-tournament dropping strategy resulted in the best results when compared with the uniform and roulette strategies.

The attacker was still able to use roughly the same amount of resources of the server as when using the uniform and roulette strategy, namely around 70% of its resources as depicted in Fig. 13. Moreover, the availability results depicted in Fig. 14 were slightly better than the results obtained with the roulette strategy (Fig. 12). In both assumptions of call duration (following an exponential and a lognormal distributions), SeVen was able to maintain high levels of availability. More than 70% (respectively, 80%) of calls were completed when assuming call duration following an exponential distribution (respectively, lognormal distribution). The proportion of incomplete calls reached levels around 25% of all calls when assuming an exponential cal duration and reached 16% of all calls when assuming lognormal call duration. Thus, more than 95% of all legitimate calls were able to reach the incall status, which means that they were able to communicate.

Finally, incomplete calls were interrupted by SeVen after communicating more than 60% of the expected time when assuming exponential call duration and more than 50% when assuming lognormal call duration.

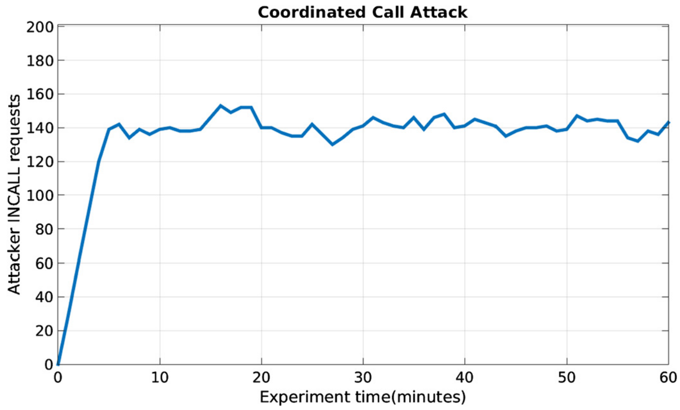


Fig. 15. Attacker call occupancy in buffer: experimental results when using SeVen with a 100-tournament dropping strategy.

6.2. Impact of using SeVen when not suffering an attack

We investigate additionally the impact of using SeVen as a proxy and implementing the selective strategy on the performance of Asterisk. Asterisk modules provide many statistics on the performance of the system. The following tables contain the number of requests according to the time intervals for the time to respond (TTR) when using and not using SeVen during normal situation, that is, when only receiving legitimate calls.

Not using SeVen					
TTR (ms)	[0, 1]	[3, 4]	[4, 5]	[8, 9]	[10, 20]
Num requests	1837	1	960	2	2

Using SeVen										
TTR (ms)	[0, 1]	[3, 4]	[4, 5]	[7, 8]	[10, 20]	[20, 30]	[30, 40]	[40, 50]	[50, 100]	[100, 150]
Num requests	19	2	404	341	657	434	148	96	70	8

As one can observe, there is an impact to TTR when using SeVen. While without SeVen most of the requests are responded within 5 ms, with SeVen, most of the requests are responded within 100 ms.

Such a delay does not greatly impact user experience as 100 ms is negligible with respect to the time users wait until establishing a call, *e.g.*, waiting until Bob accepts the call which normally takes some seconds to happen. Moreover, as SeVen only acts on SIP messages (Initiation and Termination phases of Fig. 1), SeVen does not affect user experience when the parties are in a call (Communication Phase in Fig. 1).

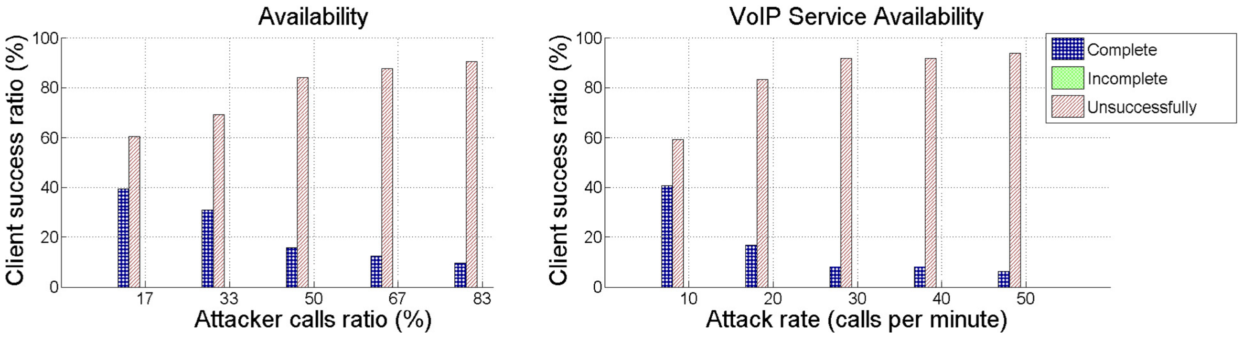
Finally, it seems possible to improve SeVen's performance by improving our implementation, *e.g.*, implementing it as a module instead of a proxy. This is left, however, to future work.

7. Comparison between simulation and experimental results

7.1. Differences between simulations and experiments

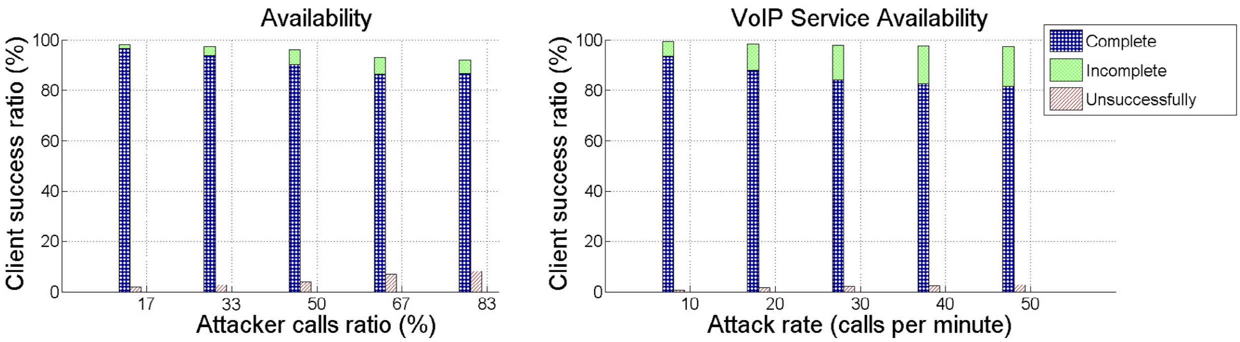
There are some important differences between the formal model and the experimental set-up. For a starter, the formal specification abstracts several aspects present in the experimental set-up. For example:

- We did not model how Asterisk actually manages its workers/threads. Asterisk has a number of modules that among other things, maintain call statistics, convert calls encoded some codex to another, etc.;
- In our experiments, there are other applications running in parallel with Asterisk that have to compete for resources (CPU and network interface for example). Our formal model does not incorporate this;
- We use a simplified model for network latency with constant latency time;
- While the parameters, *e.g.*, k, incoming client and attacker traffic, used in the simulations were proportional to the parameters used in the experiments, they were much lower to the ones used in the experimental results. If we used the actual values for these parameters, simulations would have taken much longer;
- In our experiments, SeVen is used as a proxy, while in our formal model the defense was incorporated into the application.



(a) Simulation. (b) Experiment.

Fig. 16. Client success ratio: comparison between simulation and experimental results for lognormal call duration.



(a) Simulation. (b) Experiment.

Fig. 17. Client success ratio: comparison between simulation and experimental results for lognormal call duration when using SeVen with a tournament dropping strategy.

Despite these important differences/abstractions, as we compare in more detail next, the simulation results corresponded to many of the experimental results. For example in terms of availability, *i.e.*, proportion of completed, incomplete and unsuccessful calls, the simulation results correctly indicated the power of the attack and the efficiency of SeVen mitigating this attack. They also correctly indicated which dropping strategy is better and how availability changes with the increase on the attack rate. The simulation results were less accurate in predicting the time of incomplete calls reaching a difference of 30%. The reasons for this discrepancy are not completely clear, but we believe it has to do with the abstractions mentioned above. We leave this investigation to future work.

7.2. Detailed comparisons

Typically each simulation takes about 30 seconds to be completed. In contrast, each experiment carried out on the network took 60 minutes. This means that specifiers can quickly test different selective strategies using formal verification before implementing the necessary machinery and carrying out experiments. Once a selective strategy is shown by formal verification to have reasonable results, experiments can be carried out to validate the chosen defense.

In this section, we compare the results obtained through formal verification detailed in Section 5 and the results obtained by carrying out experiments detailed in Section 6. In general, availability results obtained through formal verification indeed corresponded to our experimental results showing a high degree of accuracy for our simulation results.

Efficiency of the coordinated call attack: Our simulation results with the scenario without defense showed that the Coordinated Call is effective if the server does not have any defense mechanism under both assumptions of duration calls (exponential and lognormal). This result was also observed in our experimental results. Fig. 16 on the service availability illustrates this correspondence.

Efficiency of SeVen: All our simulations results using scenarios with SeVen indicated that SeVen is indeed a good defense for mitigating the Coordinated Call attack. The greater proportion of calls were completed calls, while a smaller proportion of the calls were incomplete and a minority of calls were unsuccessful. The same behavior was observed by our experiments. This is illustrated by Fig. 17.

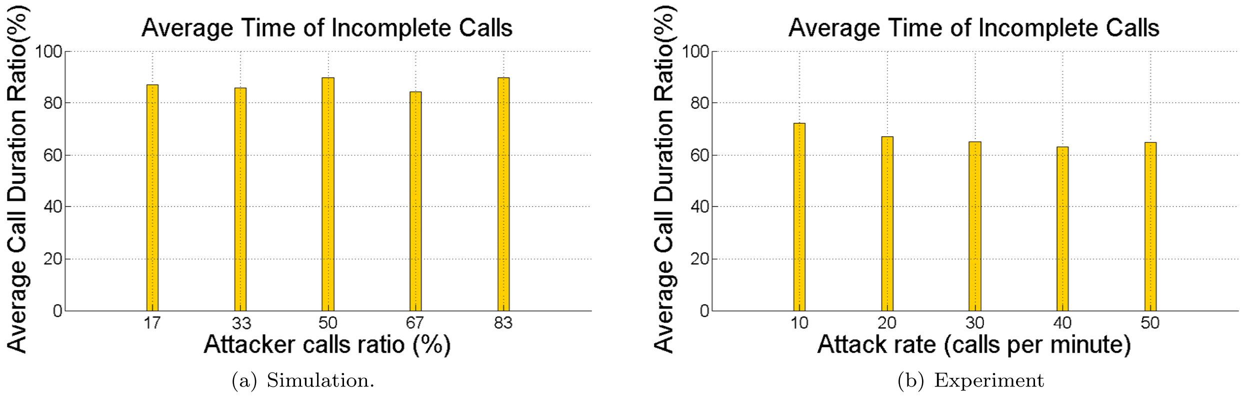


Fig. 18. Average time of incomplete calls: comparison between simulation and experimental results when using SeVen with the $\frac{k}{2}$-tournament dropping strategy.

Dropping strategies evaluation: Our simulations were able to predict that the tournament dropping strategy would perform best. However, it was not able to forecast that the roulette strategy would perform better than the uniform strategy. It is not clear to us why this was the case.

Better performance for lognormal call duration: Our simulations results also predicted that selective strategies would perform better in scenarios where call duration of legitimate clients follows a lognormal distribution, such as in VoIP communication, than scenarios with exponential call duration, such as in traditional telephony. This was the case independent on the dropping strategy used (uniform, roulette or $\frac{k}{2}$-tournament). The same behavior was observed in our experimental results.

Average time of incomplete calls: Our simulations results also indicated that the average time that incomplete calls stayed communicating before being drop by SeVen was relatively high: more than 80% of the expected time when using the tournament strategy under a lognormal call duration. This results diverged from the experimental results which observed an average time of incomplete calls of around 60%. This is illustrated by Fig. 18.

However, for other scenarios, the difference between simulation and experimental results was greater reaching 30% of difference. It is not clear to us what are the reasons for this difference. We suspect that the modeling of the time delays should be improved. In any case, our results suggest that while simulations provide quite accurate results on availability, it is less accurate on specific timing analysis. We observed a similar behavior during our experiments and simulations when modeling our defense for mitigating Application-Layer DDoS attacks [4].

8. Related and future work

This paper formalized a new selective defense, called SeVen, for mitigating Coordinated 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. We proposed three defenses based on the dropping strategy method (uniform, roulette and $\frac{k}{2}$-tournament). We carried out simulations and experiments assuming traditional telephony and VoIP communications. In both cases, we observed that our SeVen was able to mitigate the Coordinated Call attack. Finally, we compared the results obtained using our formal analysis with the results obtained by experimentation obtaining a high accuracy. This further supports the value of formal analysis during the development of selective defenses for mitigating DoS attacks.

Most of the existing work [28–33] on mitigating DoS attacks on VoIP services focuses on flooding attacks, such as the SIP-Flooding attack. They analyze the network traffic and whenever they observe an abrupt increase in the traffic load, they activate their defenses. The network traffic is usually modeled using some statistical approach, such as correlating the number of INVITE requests and the number of requests that completed the SIP initiation phase [28] or using more complicated metrics such as Helling distance to monitor traffic probability distributions [29–31]. Other solutions place a lower priority on INVITE messages, which are only processed when there are no other types of request to be processed [32, 33].

As the Coordinated Call Attack emulates legitimate client traffic not causing an unexpected sudden increase in traffic, all these defenses are not effective in mitigating the Coordinated Call Attack. The few solutions we found in the literature for this type of attack are commercial tools that act as a firewall which monitor all the call traffic and the signaling [34, 35] or analyze audio samples [36] in order to differentiate the fraudulent calls from the legitimate ones. Less sophisticated mechanisms [37] monitors all the incoming requests and reject those whose IPs do not belong to a list of trusted IPs. Clearly such approaches does not work well when the attackers are malicious users whose IPs are in the trusted list and are not using automation to make the calls. In addition, these commercial tools can be expensive for small businesses to purchase and maintain, and they require technical expertise for proper installation.

One main advantage of our proposed solution is that it is not tailored using many specific assumptions on type of service. The only assumption used is a previous knowledge of the average call duration, which can be easily inferred from the service call history. Moreover, our solution can be easily integrated with other mechanisms such as the IP filtering approach used in [37].

[38] 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. Wu et al. [39] have proposed a 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 [40], while others use branching temporal logics [41]. This paper takes the approach used in [42,25,24], where one formalizes the system in Maude and uses the Statistical Model Checker PVeStA to carry out analyses. While [42,25,24] 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 going to investigate the mitigating of other types of attacks, such as volumetric and amplification attacks. We are also thinking on intrusion detection mechanisms. We are also interested in building defenses for mitigating amplification attacks [18]. We have also been using SeVen for mitigating High-Rate ADDoS attacks using Software Defined Networks [43]. We are also investigating ways to improve simulation accuracy by improving the modeling of timing aspects of the system, such as processing and network delay.

Acknowledgements

This work has been funded by the DFG as part of the project Secure Refinement of Cryptographic Algorithms (E3) within the CRC 1119 CROSSING, by RNP project GT-ACTIONS, by Capes Science without Borders grant 88881.030357/2013-01 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 27 September 2015).
[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 27 September 2015).
[3] Situational advisory: recent telephony denial of services (TDoS) attacks, http://voipsecurityblog.typepad.com/files/ky-fusion_tdos_3-29-13-2.pdf/ (Accessed 27 September 2015).
[4] Y.G. Dantas, V. Nigam, I.E. Fonseca, A selective defense for application layer DDoS attacks, in: JISIC 2014, 2014, pp. 75–82.
[5] The surging threat of telephony denial of service attacks, http://voipsecurityblog.typepad.com/files/tdos_paper_4-11-13.pdf (Accessed 28 September 2015).
[6] M. Clavel, F. Durán, S. Eker, P. Lincoln, N. Martí-Oliet, J. Meseguer, C. Talcott, All About Maude: A High-Performance Logical Framework, LNCS, vol. 4350, Springer, 2007.
[7] K. Sen, M. Viswanathan, G. Agha, On statistical model checking of stochastic systems, in: CAV, 2005, pp. 266–280.
[8] M. AlTurki, J. Meseguer, PVeStA: a parallel statistical model checking and quantitative analysis tool, in: CALCO, 2011, pp. 386–392.
[9] A. Lipowski, D. Lipowska, Roulette-wheel selection via stochastic acceptance, CoRR, arXiv:1109.3627.
[10] T. Blickle, L. Thiele, A mathematical analysis of tournament selection, in: Proceedings of the 6th International Conference on Genetic Algorithms, San Francisco, CA, USA, 1995, pp. 9–16.
[11] L. Brown, N. Gans, A. Mandelbaum, A. Sakov, H. Shen, S. Zeltyn, L. Zhao, Statistical analysis of a telephone call center: a queueing-science perspective, J. Am. Stat. Assoc. 100 (2005) 36–50.
[12] P. Galiotos, T. Dagiuklas, S. Kotsopoulos, Call-level VoIP traffic modelling based on data from a real-life VoIP service provider, in: 2015 IEEE Globecom Workshops (GC Wkshps), 2015, pp. 1–7.
[13] I. Digium, SAsterisk private Branch exchange, http://www.asterisk.org/, 2015 (Accessed 28 September 2015).
[14] M.O.O. Lemos, Y.G. Dantas, I. Fonseca, V. Nigam, G. Sampaio, A selective defense for mitigating coordinated call attacks, in: 34th Brazilian Symposium on Computer Networks and Distributed Systems, SBRC, 2016.
[15] Y.G. Dantas, M.O.O. Lemos, I. Fonseca, V. Nigam, Formal specification and verification of a selective defense for TDoS attacks, in: 11th International Workshop on Rewriting Logic and Its Applications, WRLA, 2016.
[16] SeVen, https://github.com/ygdantas/SeVen.git, 2016.
[17] Session initiation protocol, http://www.ietf.org/rfc/rfc3261.txt.
[18] R. Shankesi, M. AlTurki, R. Sasse, C.A. Gunter, J. Meseguer, Model-checking DoS amplification for VoIP session initiation, in: ESORICS, 2009, pp. 390–405.
[19] M. Hines, Attackers get chatty on VOIP, http://www.pcworld.com/article/132389/article.html, 2007 (Accessed 27 September 2015).
[20] R. Gayraud, O. Jacques, SIPp – SIP traffic generator, http://sipp.sourceforge.net, 2014 (Accessed 28 September 2015).
[21] S. Khanna, S. Venkatesh, O. Fatemieh, F. Khan, C. Gunter, Adaptive selective verification: an efficient adaptive countermeasure to thwart DoS attacks, IEEE/ACM Trans. Netw. 20 (3) (2012) 715–728.
[22] S. Khanna, S.S. Venkatesh, O. Fatemieh, F. Khan, C.A. Gunter, Adaptive selective verification, in: INFOCOM, 2008, pp. 529–537.

[23] J. Jewett, J. Shrago, B. Yomtov, Designing Optimal Voice Networks for Businesses, Government, and Telephone Companies, 1980.
[24] J. Eckhardt, T. Mühlbauer, M. AlTurki, J. Meseguer, M. Wirsing, Stable availability under denial of service attacks through formal patterns, in: FASE, 2012, pp. 78–93.
[25] J. Eckhardt, T. Mühlbauer, J. Meseguer, M. Wirsing, Statistical model checking for composite actor systems, in: WADT, 2012, pp. 143–160.
[26] J. Meseguer, Twenty years of rewriting logic, J. Log. Algebraic Program. 81 (7–8) (2012) 721–781.
[27] G. Agha, J. Meseguer, K. Sen, PMaude: rewrite-based specification language for probabilistic object systems, Electron. Notes Theor. Comput. Sci. 153 (2) (2006) 213–239.
[28] D.-Y. Ha, H.-K. Kim, K.-H. Ko, C.-Y. Lee, J.-W. Kim, H.-C. Jeong, Design and implementation of SIP-aware DDoS attack detection system, in: ICIS '09, ACM, New York, NY, USA, 2009, pp. 1167–1171.
[29] J. Tang, Y. Cheng, C. Zhou, Sketch-based SIP flooding detection using Hellinger distance, in: Global Telecommunications Conference, 2009, GLOBECOM 2009, IEEE, 2009, pp. 1–6.
[30] J. Tang, Y. Cheng, Y. Hao, Detection and prevention of SIP flooding attacks in voice over IP networks, in: INFOCOM, 2012 Proceedings, IEEE, 2012, pp. 1161–1169.
[31] J. Tang, Y. Cheng, Y. Hao, W. Song, SIP flooding attack detection with a multi-dimensional sketch design, IEEE Trans. Dependable Secure Comput. 11 (6) (2014) 582–595.
[32] X.-Y. Wan, Z. Li, Z.-F. Fan, A SIP DoS flooding attack defense mechanism based on priority class queue, in: WCNIS 2010, 2010, pp. 428–431.
[33] F. Zi-Fu, Y. Jun-Rong, W. Xiao-Yu, A SIP DoS flooding attack defense mechanism based on custom weighted fair queue scheduling, in: ICMT 2010, 2010, pp. 1–4.
[34] SecureLogix: telephony denial of service (TDoS) solutions, http://www.securelogix.com/solutions/telephony-denial-of-service-TDoS.html (Accessed 27 September 2015).
[35] TransNexus NexOSS, http://transnexus.com/telephony-denial-service-attacks/ (Accessed 27 September 2015).
[36] PINDROP: protecting your call centers against phone fraud & social engineering, https://www.pindrop.com/wp-content/uploads/2016/01/pindrop_overview_whitepaper_fi_20141121_v2.pdf (Accessed 27 September 2015).
[37] TDoS attack mitigation, http://www.cisco.com/c/en/us/td/docs/ios-xml/ios/voice/cube_proto/configuration/15-mt/cube-proto-15-mt-book/voi-cube-tdos-attack-mitigation.pdf (Accessed 27 September 2015).
[38] F. Huici, S. Niccolini, N. D'Heureuse, Protecting SIP against very large flooding DoS attacks, in: GLOBECOM'09', IEEE Press, Piscataway, NJ, USA, 2009, pp. 1369–1374.
[39] Y.-S. Wu, S. Bagchi, S. Garg, N. Singh, T. Tsai, SCIDIVE: a stateful and cross protocol intrusion detection architecture for voice-over-IP environments, in: DSN'04, IEEE Computer Society, Washington, DC, USA, 2004, p. 433.
[40] C. Meadows, A formal framework and evaluation method for network denial of service, in: CSFW, 1999, pp. 4–13.
[41] A. Mahimkar, V. Shmatikov, Game-based analysis of denial-of-service prevention protocols, in: CSFW, 2005, pp. 287–301.
[42] M. AlTurki, J. Meseguer, C.A. Gunter, Probabilistic modeling and analysis of DoS protection for the ASV protocol, Electron. Notes Theor. Comput. Sci. 234 (2009) 3–18.
[43] J. Henrique, I.E. Fonseca, V. Nigam, SHADE: Uma Estratégia Seletiva para Mitigar Ataques DDoS na Camada de Aplicação em Redes Definidas por Software, in: XXXIV Simpósio Brasileiro de Telecomunicações e Processamento de Sinais, SBrT 2016, 2016.