Enhancing the Trustworthiness of Service On-Demand Systems via Smart Vote Filtering

Christos V. Samaras(✉), Ageliki Tsioliaridou, Christos Liaskos, Dimitris Spiliotopoulos, and Sotiris Ioannidis

Foundation of Research and Technology - Hellas (FORTH), Heraklion, Greece
{csamaras,atsiolia,cliaskos,dspiliot,sotiris}@ics.forth.gr

Abstract. Service on-demand (SoD) systems allow their users to regulate the sharing of common resources via a voting process. A common application example is the collaborative scheduling of multimedia transmissions in e-radio or video streaming services. Therefore, high user commitment and participation is critical to the success of a SoD system. Securing a SoD system against common attacks, such as vote flooding, can impose client anonymity retraction, online registering and access control mechanisms. Nonetheless, such processes can degrade the users' quality of experience, discouraging user participation. The present study proposes a defense mechanism against vote flooding attacks that can operate under complete vote anonymity and without any user access restrictions. The novel scheme is implemented as a vote filtering scheme, executed prior to each service scheduling decision. The proposed scheme has linear complexity and is shown via simulations to considerably mitigate or completely negate the effects of several attacks types.

Keywords: Service on-demand · Client anonymity · Security · Query filtering

1 Introduction

Service on-demand (SoD) systems constitute a particularly attractive means of resource sharing and large-scale information dissemination. Users of SoD systems can influence how often a server supplies a service via a voting system. For instance, users of video-on-demand or e-radio systems can regulate the broadcast frequency of multimedia files [25,27]. Therefore, high and unobstructed user participation is critical to the operation and economic viability of SoD systems, accentuating the need for user-friendliness. To this end, SoD systems may need to operate on anonymous user votes and without any access control method that may degrade the users' quality of experience [26]. On the other hand, such requirements facilitate the misuse by malevolent users who may, e.g., flood the system with vast amounts of votes for personally preferred services, degrading the trustworthiness of the process.

SoD systems typically follow a centralized architecture, comprising a server and a set of clients in a virtual star topology. The server supports a set of

M. Conti et al. (Eds.): TRUST 2015, LNCS 9229, pp. 88–103, 2015.
DOI: 10.1007/978-3-319-22846-4_6

actions that are provided in a cyclic fashion. After the end of an action, the server proceeds to select the next action for execution. The selection is derived from the votes of the users, which arrive continuously at the server and are promptly enqueued. The selection process can consider the arrival times of the votes at the server, as well as the total number of votes pertaining to each supported action. Typical selection processes are the First Come-First Served, Most Requests First and the RxW scheduler which takes into account both considerations [13]. Given that the arrival time of a vote at the server cannot be tampered with, a malevolent user may seek to influence the total number of votes pertaining to one or more actions. Thus, the selection process can be forced to produce results that no longer correspond to the preferences of the normal (benevolent) users of the system.

Existing voting systems employ access control and user identification mechanisms in order to: (i) discourage or disable vote-flooding attacks and (ii) detect the perpetrator in case of a successful attack [9,20]. A commonly followed access control approach is to employ CAPTCHAs, automated challenge-response Turing tests, to disable vote flooding by bots [32,39]. However, the process is time-consuming and degrades the quality of experience of the normal users. Furthermore, the users may be requested to register to the system with an online account, compromising their anonymity. Some approaches employ an intermediate anonymization server, which removes personal information from the vote of a user prior to forwarding it to the SoD system [11]. Nonetheless, this approach simply delegates the identification and access control process to another system and still degrades the quality of experience. Furthermore, the approach requires additional equipment, increasing the capital and operational expenses of the system.

The present paper proposes a mechanism for defending against vote flooding attacks in SoD systems, which requires no access control and does not compromise the anonymity of the users, even under attack. It can be classified as a first-line, low-complexity defense mechanism that is implemented as a vote filtering mechanism. The methodology of the presented scheme comprises an attack detection and an actuation process. For the detection purposes, the votes of the users are mapped to a stream of alarm indications, each designating the presence or absence of malevolent behavior. A specially-designed, low-complexity variation of the Misra-Gries algorithm [31] deduces the most frequent indication, thus raising an alarm or deducing normal operation. In the case of an alarm, the actuation process is activated and proceeds to filter the users' votes prior to every new scheduling decision. The success of an attack is measured in terms of the increase it induces to the user query service ratio and service times. In retaliation, the proposed scheme succeeds in keeping these metrics close to their normal operation counterparts under several attack cases. Thus, it can promote the trustworthiness of a SoD system, without compromises in the users' quality of experience.

The remainder of this paper is organized as follows. The related work on trustworthy voting systems is given in Sect. 2. The prerequisites for the presentation and presentation of the novel scheme follow in Sect. 3. The scheme is detailed in Sect. 4 and evaluated via simulations in Sect. 5. Finally, the conclusion is given in Sect. 6.

2 Related Work

Research on secure service-on-demand systems has not proposed a voting mechanism that can operate on anonymous users with no access restrictions, to the best of the authors knowledge. However, there exists a considerable amount of work on electronic voting systems and polling protocols in general, which has concentrated on a diverse set of desired properties and functionality such as accuracy, privacy, verifiability, eligibility, coercion resistance, availability and fault-tolerance. A number of protocols, models, prototypes, and real-world systems have been proposed and implemented to support e-voting and polling functions.

Electronic voting schemes are mainly divided into three categories, based on the technique used to anonymize votes: (i) Homomorphic encryption allows computations to be carried out on ciphertext, thus generating an encrypted result which, when decrypted, matches the result of operations performed on the plaintext. Protocols based on homomorphic encryption generally have a complex mathematical structure thus inducing high computational costs. (ii) In blind signature approaches the content of message/vote is disguised (blinded) before it is signed, thus the signer (authenticator) is not given any knowledge about the message. The voter unblinds the signed vote and submits it to the tallier through an anonymized channel. Blind signature protocols usually exhibit the advantages of simplicity, low computational costs and being ballot independent. (iii) A mix network (mixnet) is a multistage system that uses cryptography and permutations to provide anonymity. The design of a mixnet is based on providing anonymity for a batch of inputs, by changing their appearance and removing the order of arrival information. In mix network schemes, voters authenticate and submit encrypted votes; votes are anonymized using a mix; and anonymized votes are then decrypted. Mix network protocols involve less voter's interactions, but require complex proofs of correctness.

E-voting and polling have been an active area of research posing several new challenges [2,16,19,23,34,38]. Comparison of existing voting schemes reveals common security property tradeoffs [35]. REVS [22] is an electronic voting system based on blind signatures and designed for distributed and faulty environments, which exploits server replication to allow a certain degree of failures. Sensus [11] is a secure and private system for polling that requires at least two servers, namely a validator and a tallier, for conducting an election or a survey (i.e., a generic term of polling is considered). In [3] authors propose a prototype implementation of SEAS, which is a portable and flexible system that preserves the limited number of servers of the above-mentioned Sensus, but it avoids a vulnerability that allows one of the entities involved in the election process to cast its own votes in place of those that abstain from the vote. Civitas [9] is based on mix networks and enforces verifiability (an integrity property) and coercion resistance (a confidentiality property), whereas it does not rely on trusted supervision of polling places, making it a remote voting system.

In the literature there also exist studies on polling protocols [5,14,17,20,21,37], which cover areas such as: distributed polling, privacy, secret sharing, scalability, social networks, peer-to-peer networks, and reputation systems. However, anonymity systems are of significant practical relevance because they are the best means of providing privacy for users. Further works relating to e-voting and to methods for achieving anonymity and providing privacy for users, can be seen in [4,6–8,10,15,24,28–30,41].

Given that existing systems do not cover the needs of SoD systems for complete client anonymity and unrestricted access, the authors proposed an initial solution based on early filtering of client queries [26]. The study defined probable attack types and proposed a defense mechanism based on the Dendritic algorithm, a nature-inspired process for intrusion detection based on danger and safety signals. However, being a nature-inspired heuristic, the mechanics of the Dendritic algorithm are still not well understood [18]. Particularly, it is not clear how to parametrize and map the danger and safety signals to real attributes of a given system. Thus, while the proposed Sensor Swarm Filter process was shown to efficiently defend against several attack types, it could not account for common attacks, such as random query flooding.

The present study proposes a superior query filtering process that: (i) is based on the well-studied Misra-Gries classification algorithm [31], and (ii) utilizes parameters that have an intuitive and clear meaning within the context of the voting system.

3 Prerequisites

We assume a service-on-demand system, comprising a server and a set of connected clients. The server hosts a number of service "items" (actions), each with its own service time. The clients post queries in order to vote for the next action to be taken by the server. In order to derive the next service action, the server employs the RxW scheduler without preemption support, but enhanced to handle actions with different processing times [1,36]. The RxW scheduler selects the action with the highest number of hits, multiplied by the queuing time of its oldest query.

The preferences of each client regarding the service actions are unknown to the server, and are expressed as personal probability mass functions (p.m.f.):

$$p_{c,i},\ c = 1 \ldots C,\ i = 1 \ldots N,\ :\ \sum_{i=1}^{N} p_{c,i} = 1 \tag{1}$$

which denotes the percentage of queries of client $c = 1 \ldots C$ that refer to action i.

In order to establish a dependable ground-through on the popularity of each action, an external, trusted entity provides the server with an approximate, per-action p.m.f. as:

$$\mathbb{P}_i,\ i = 1 \ldots N\ :\ \sum_{i=1}^{N} \mathbb{P}_i = 1 \tag{2}$$

For example, in the case of a video on-demand service, the popularity of each movie "item" can be derived by its ranking in online services (e.g., the Internet Movie Database), hits in social networks (e.g., Tweets) or direct polling of trustworthy, authenticated users (critics).

Each service action may be requested multiple times over the operation of the system by any user, without restrictions. On the client-side, each benevolent user poses a query for a single service action and awaits for a maximum time interval D (deadline). The server is oblivious to deadline expiration events, since such an ability would be open to extensive misuse, even by non-expert users. If D elapses and the server has not started to process the requested action, the client abandons the query. Regardless of the outcome (served or not) a client poses a new query after a random $ThinkTime$ [25]. The service ratio of the system is defined as the total number of served queries over all clients divided by the total number of posed queries.

Finally, the attacker model of [26] is assumed. According to it, a malevolent user performs query flooding in order to tip the RxW scheduler to their favor. An attack by a malevolent user is defined as:

$$\left\{target,\, \bar{T},\, t_s,\, t_e\right\} \tag{3}$$

where t_s is the time moment when the user begins posting consecutive queries with mean interarrival $\bar{T}$ until time t_e. The *target* of these queries, i.e., the action that is being requested, defines the attack types introduced in [26]:

- **Needed action**. "Selfish" behavior which constitutes at flooding the scheduler with requests for the personally needed service action.
- **Random action**. Flooding the server with random queries.
- **Less popular action**. The scheduler is flooded with multiple requests for the less wanted service action.
- **Lengthiest action**. The scheduler is forced to yield the most time-consuming service, delaying all other actions.
- **Smallest popularity-to-size ratio** action, which combines the preceding attacks.

The last three attack types assume that a malevolent user has obtained an approximation of the the $\mathbb{P}_i$ p.m.f..

The proposed scheme seeks to improve the trustworthiness of the system by (ideally) keeping the service/expiration ratio and the mean service time unaltered, despite the presence of an increasing number of malevolent users.

4 Misra-Gries-Based Query Filtering

A service-on-demand system defines a cycle of operation given in Fig. 1. A server selects and executes an action from a given pool, based on the preceding votes of the users. While the action is executed, the server enqueues all incoming votes in a single queue, logging their arrival times as well. Once the execution

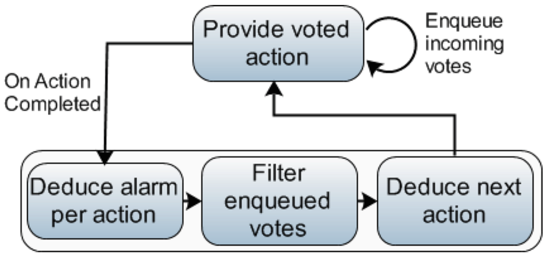

Fig. 1. State chart of the service on-demand system, combined with the proposed defense mechanism.

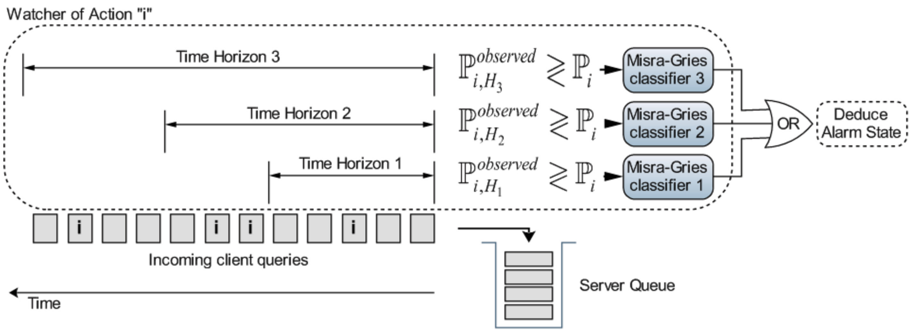

Fig. 2. Operation of an action "watcher", responsible for raising an alarm when the observed rate of incoming queries for the action, $\mathbb{P}^{observed}$, consistently surpasses the expectation $\mathbb{P}$.

of the current action is complete, the server proceeds to select the next action for execution, given the user votes and the employed scheduler (e.g., RxW). The proposed defense mechanism takes action before the execution of the scheduler, by filtering the votes accumulated at the queue of the server. In this aspect, the proposed mechanism has the added advantage of not disrupting the normal operation of the used scheduler.

The defense mechanism comprises two components: the *threat detection module* and the *actuation module* (i.e., query filtering).

The operation of the *threat detection module* is illustrated in Fig. 2. It comprises a set of N *watcher* processes running as daemons on the server. Each watcher is responsible for detecting suspicious queries pertaining to a single action offered by the server, with $1-1$ correspondence. The watcher of action i processes all incoming client queries before they enter the server's queue, and logs the running ratio of $i-$query occurrences, $\mathbb{P}^{observed}_{i,H}$, over three different time horizons, H_1, H_2, H_3. For example, if the span of time horizon H_1 is $S = 100$ incoming client votes and action i was requested $n = 10$ times within this window, then $\mathbb{P}^{observed}_{i,H} = {}^{n}/_{S} = 0.1$.

The $\mathbb{P}_{i,H}^{observed}$ logging over three time horizons makes for fast attack detection (smaller horizon, H_1), vigilance after the attack (medium horizon, H_2) and indications of long-term attacks that may call for additional security measures (large horizon, H_3), such as CAPTCHA checks and client identification requests. The span of the time horizons can be set intuitively. For example, assuming that $min\,\{\mathbb{P}_i\} = p$, H_1 can be set at $\lceil 1/p \rceil$, i.e., the span that accentuates the presence of votes for the least probable actions. H_3 can be set to a maximum allowed attack duration, and H_2 in a value within $[H_1, H_3]$.

Once the $\mathbb{P}_{i,H}^{observed}$ values have been derived for each horizon, the attack detection module proceeds to compare them to the $\mathbb{P}_i$ expectations and deduce whether they constitute threat indications. This task is accomplished by the Misra-Gries (MG) classifier, incorporated to the watcher process.

The employed variation of the MG algorithm extracts the most frequent object from a running stream [31]. MG assumes an associative array indexed by the objects, ctr_{obj}, which are initialized to zero. For each incoming *object*, MG increases ctr_{obj} by one and decreases all other counters by one unit. If a counter has become negative, it is reset to zero. After K steps, the classifier yields the most common object, obj^*, as:

$$obj^* = argmax\,\{ctr_{obj}\} \tag{4}$$

From the $(K+1)^{th}$ step and on, the MG process retains the classification result, but the ctr_{obj} counters are reset to zero and the process starts over. Thus, the classification result is updated at the $(2 \cdot K)^{th}$ step. The storage overhead of MG is $O(m)$, where m is the total number of possible objects, while its complexity is constant, $O(1)$.

In the case of the proposed defense mechanism, the MG objects are the Boolean outcomes of the comparisons:

$$\mathbb{P}_{i,H}^{observed} > \mathbb{P}_i \tag{5}$$

i.e., $m = 2$. In other words, MG deduces whether the votes pertaining to an action i are persistently higher than the expectations, implying that an attack may be in progress. In this case, MG is said to raise an “alarm”. Three MG instances are used within each action watcher, each deducing the alarm state over the three time horizons. The action watcher then yields an alarm state for the monitored action if any of the three MG processes is positive.

The set of watchers, one per available server action, thus yield a Boolean alarm level per action, $\mathbb{A}_i$, at any requested time moment.

The *actuation module* (query filtering process of Fig. 1, formulated as Algorithm 1) takes place before relinquishing operation to the RxW scheduler.

At first, Algorithm 1 counts the number of occurrences of each query for action i within the server queue (lines $3-5$). The Algorithm then proceeds to calculate the expected (proper) occurrences for each action with a raised alarm flag, $\mathbb{A}_i$ (lines $8-12$). However, it is possible that certain actions have presently zero occurrences within the queue (e.g., when the corresponding $\mathbb{P}_i$ is low).

Algorithm 1. Query filtering process.

INPUTS:

1. Presently Enqueued Queries $\mathbb{Q}_k$, $k = 1 \dots Q$;
2. Expectations $\mathbb{P}_i$, $i = 1 \dots N$;
3. Binary Alarm State per Action $\mathbb{A}_i$, $i = 1 \dots N$.

```
1:  $times_i \leftarrow 0, \forall i = 1 \dots N$;
2:  $proper\_times_i \leftarrow 0, \forall i = 1 \dots N$;
3:  for $k = 1 \dots Q$
4:    $times_{\mathbb{Q}_k} = times_{\mathbb{Q}_k} + 1$;
5:  end for
6:  $s \leftarrow 1 - \sum_{i:\{1 \dots N | times_i = 0\}} \mathbb{P}_i$;
7:  $proper\_times_i \leftarrow times_i$, $i = 1 \dots N$;
8:  for $i = 1 \dots N$
9:    if $times_i > 0$ and $\mathbb{A}_i$
10:     $proper\_times_i \leftarrow \lfloor \frac{\mathbb{P}_i \cdot Q}{s} \rfloor$;
11:   end if
12: end for
13: for $i = 1 \dots N$
14:   if $times_i > proper\_times_i$
15:     Remove the most recent $proper\_times_i - times_i$ queries for action $i$ from the server queue.
16:   end if
17: end for
```

The cumulative probability of these actions is logged (line 6) and is distributed to other actions with $times_i \neq 0$ within the queue (line 10). This approach ensures a less aggressive but more fair query filtering, since it takes into account that zero action occurrences within the queue are normal from time to time.

The actual query filtering then takes place at lines $13 - 17$. The occurrences of each query type are reduced to their expected values by discarding the newest queries first. Notice that the RxW scheduler schedules then next action for execution by checking the product of occurrences multiplied by the maximum query waiting time for each action. Therefore, given the importance of waiting times, discarding newest queries first ensures that older, potentially legitimate queries are not harmed by the filtering process.

The maximum storage overhead and complexity of Algorithm 1 is $O(N)$, which also represents the complexity of the complete defense mechanism, given that static requirements of the MG and $\mathbb{P}_{i,H}^{observed}$ logging sub-processes.

5 Simulations

In this Section, the performance of the proposed Misra-Gries Filtering (MGF) is compared via simulations to the Sensor Swarm Filtering (SSF) of [26]. The simulator, implemented on the Anylogic platform [40], represents a broadcast on-demand system, where “actions” correspond to “Web page items” with dynamic content. The runs evaluate the ability to maintain acceptable service ratios and mean service times while the system is under attack.

The system configuration assumes a star topology comprising a broadcast on-demand server, connected to $C = 100$ clients via $20Mbps$ links. The upstream direction (client-to-server, for posting queries) is considered trivial.

The server schedules its transmissions by employing the $R \times W$ on-demand scheduler. Each transmission pertains to an item selected from a static pool of $N = 100$ items with random sizes $l_i \in [1, 10]\, KBytes$ (uniformly distributed), representing simple Web pages.

The query deadline is set to $100\, msec$ due to the low item size/channel rate ratio. Should the deadline be exceeded, the query is dropped and global query service ratio is updated. Else, the query is answered successfully, and the average, global service time is updated. The clients' $ThinkTime$ is picked uniformly within $[0, 10]\, sec$.

The client query posing process operates as follows. Each client c has preset preferences in the form of a p.m.f. over the items, $p_{c,i}$, $i = 1 \dots N$, which is unknown to the server. $\mathbb{P}_i$ is derived from a distributed consensus process. We assume that the clients participate in a separate social network. The server also participates as a single peer. In this network, each peer has a random number of friends (other peers), which are represented as a connected graph. Each peer c assigns a random weight $g_{c,k} \in (0, 1)$ (uniformly distributed) to each member of his local network $k = 1 \dots K$, which comprises himself and his friends. A distributed consensus is a rumor propagation process and $g_{c,k}$ expresses the effect of a friendly peer on the formation of the personal opinion. A peer may also use different sets of $g_{c,k}$ weights for each data item i (i.e., $g_{c,k,i}$). The sole restriction that must hold is $\sum_{i=1}^{K} g_{c,k,i} = 1$.

The consensus process then operates as follows. Each peer initializes its estimate, $\mathbb{P}_{c,i}^{(self)}$, as his personal preferences, $p_{c,i}$, $i = 1 \dots N$. This estimate is then sent to his immediate friends. Each peer collects the incoming estimates of all his friends and updates his estimate as:

$$\mathbb{P}_{c,i}^{(self)} \leftarrow \mathbb{P}_{c,i}^{(self)} \cdot g_{c,self,i} + \sum_{k=1 \dots K,\, k \neq self} \mathbb{P}_{c,i}^{(k)} \cdot g_{c,k,i},\ \forall i \tag{6}$$

As proven in [12], the process converges iteratively, leading to $\mathbb{P}_{c,i}^{(self)} \approx \mathbb{P}_i$, $\forall c, \forall i$. Normalization is finally applied to ensure that $\sum \mathbb{P}_i = 1$. The $\mathbb{P}_{c,i}^{(self)}$ update period was set to $1\, sec$ and convergence was typically achieved in $10\, sec$. The $p_{c,i}$ preferences were set to yield a Zipfian p.m.f., $\mathbb{P}_i \propto i^{-0.9}$, which has been observed to describe client requests for Web pages [33]. Thus, both the server and the malevolent users acquire $\mathbb{P}_i$ anonymously, without knowledge of the individual $p_{c,i}$. At that point, a varying number of malevolent users, ranging from $1 - 10\,\% \cdot C$, attack with a period of $\bar{T} = 1\, msec$ each. The percentage of malevolent users is assumed not to surpass $10\,\%$ of the total users. Notice that an on-demand system serves common needs. Thus, if the malevolent users were the majority, or even a considerable minority, the system would inevitably abide

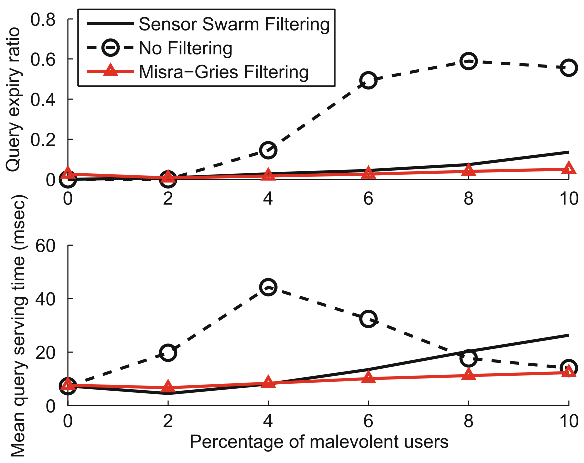

Fig. 3. Effects of the "Needed action" attack on the performance of the system.

by their preferences. Furthermore, assuming a great percentage of highly skilled users is not expected in general.

The proposed MGF uses a single *watcher* per item, monitoring the incoming client queries over three horizons, $h_1 = 10$, $h_2 = 30$ and $h_1 = 50$ (measured in number of queries). Each internal Misra-Gries process yields a classification result every $K = 20$ threshold events. Thus, the watcher process may deduce the alarm state every 200, 600 and 1000 queries. Given that $min\{\mathbb{P}_i\} = 0.02$, the horizons h_{1-3} take the values of 4, 12 and 20, which roughly correspond to ≈ 1, ≈ 10 and 20 appearances of the less popular item per threshold event. Finally, all SSF parameters are taken directly from [26].

The query service/expiration ratio and mean service time are logged and the simulation ends when a 95 % confidence has been attained. The results presented below correspond to mean values derived over 10 Monte Carlo runs, randomly varying the item sizes and the client preferences.

Figure 3 studies the robustness of the proposed MGF under a progressively aggravating "Needed action" attack. MGF is shown to surpass SSF, while essentially nullifying the attack. The query expiry ratio is kept at near-zero, while the mean service time is constant, regardless of the increasing number of attackers. On the other hand, SSF mitigates the attack for up to ≈ 5 % malevolent user percentage. From that point and on, the query expiry ratio and the service time increases steadily, with a rate double than the proposed MGF. The behavior of the system in absence of any filtering is provided to show the impact of the attack. Without any defense mechanism, approximately 60 % of the queries are dropped. The service time decreases only when a considerable amount of queries has been dropped. Therefore, MGF can be used to provide non-disrupted system performance, even under attack.

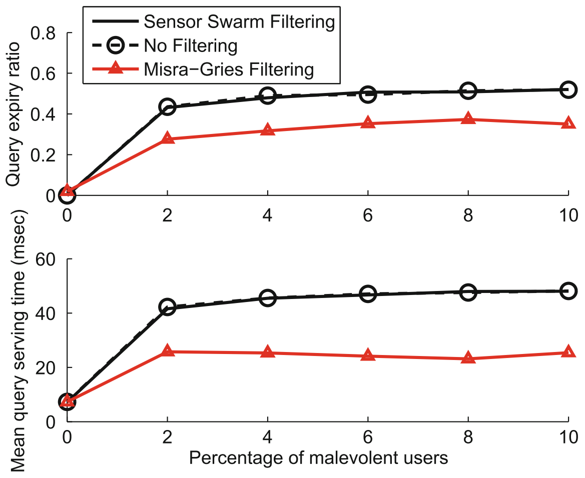

Fig. 4. The "Random action" query flooding is the most effective attack type. The proposed SSF is the only one offering a considerable degree of resilience.

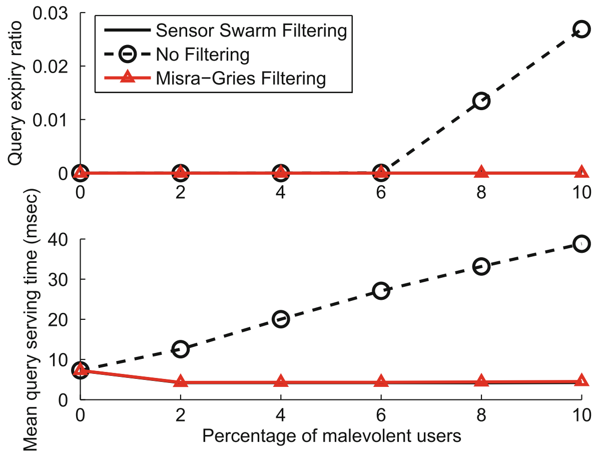

Fig. 5. The behavior of the system under a "Less popular action" attack.

Malevolent users may also attempt to flood the server with random queries. This case is examined in Fig. 4. According to [26], this type of attack is the most effective in the examined voting systems. This can be explained if we consider that a random attack of just a few users is tantamount to a high number of attackers launching a "Needed action" attack. As a result, SSF is not able to offer any defense against a "Random item" attack, even when the number of malevolent users is very low. On the other hand, the proposed MGF performs better, bounding the expiry ratio at $\approx 35\,\%$ in the worst case, while keeping the

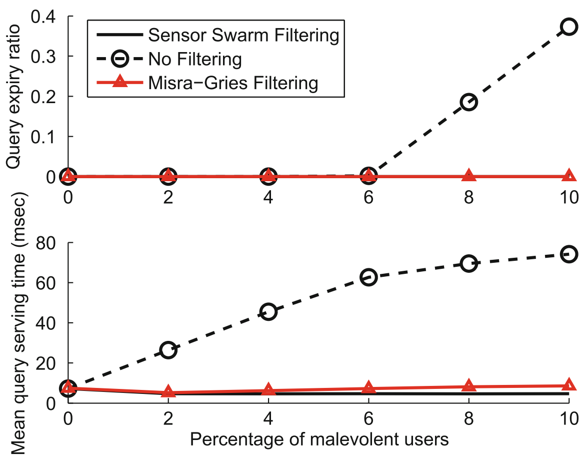

Fig. 6. Operation under false queries for the item with the lowest $\mathbb{P}_i/l_i$ ratio (popularity-to-size).

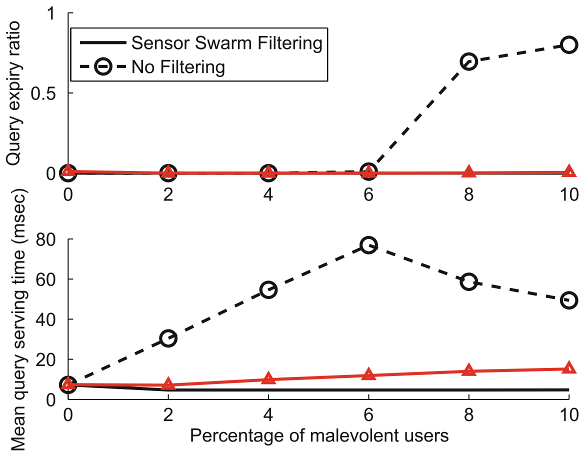

Fig. 7. Flooding the server with queries for the biggest item can yield slightly increased service times.

average service time constant at $\approx 25\, msec$. While the attack is not mitigated, the system exhibits an increased degree of robustness against this attack type. Figure 4 also accentuates the fact that the nature-inspired, Dendritic attack detector of SSF is still not well understood [18]. The present simulations can certainly not preclude that a different mapping of the nature-inspired process to real-world attributes may perform better. However, such a mapping is not straightforward and a well-defined process does not exist up to date.

Figures 5, 6 and 7 study the less probable attacks of "Less popular action", "Lowest $\mathbb{P}_i/l_i$ action" and "Lengthiest action" (i.e., biggest item). As shown in

Fig. 5, the "Less popular action" attack is easily detected and fully negated by both MGF and SSF. The fact that a very unpopular item appears multiple times in the server query queue facilitates attack detection and mitigation. The performance of the system is not affected much when item popularity and size are combined into one attack, as shown in Fig. 6. Both MGF and SSF mitigate the attack, with SSF offering slightly better servicing times. This behavior is owed to the fact that SSF takes into account the size of the items in the detection phase. The internal alarm level of SSF increases faster for big items and slower for small ones. This difference in performance is more discernible in Fig. 7, focusing on "Biggest item" attacks only. While both SSF and MGF detect the attack, the filtering of SSF is more aggressive against big items, leading to a gain in service times. However, the "Lowest $\mathbb{P}_i/l_i$ item" and "Biggest item" attacks cannot be considered as effective under presence of either MGF and SSF. Furthermore, a hacker is more likely to launch "Needed item" and "Random" attacks, since these are more impactful, as shown in Figs. 3 and 4. Therefore, the $+2\,msec$ and $+10\,msec$ service time advantage of SSF over the proposed MGF in "Lowest $\mathbb{P}_i/l_i$ item" and "Biggest item" is not deemed significant. Coupled with $O(N)$ complexity, the proposed MGF scheme can offer increased system robustness under the most significant attack types, with minimal requirements.

6 Conclusion

The present paper proposed a mechanism for defending against vote flooding attacks in service on-demand systems. The novel scheme was shown to suppress the effects of such attacks, even when a considerable percentage of the users are malevolent. Furthermore, the proposed scheme does not compromise the anonymity of the users and imposes no access control that could degrade the users' quality of experience. Combining non-disrupted user-friendliness and non-obstructed operation even under considerable attacks, the propose scheme can constitute an attractive add-on for trustworthy on-demand systems.

Acknowledgment. This work was partially supported by EU FP7 project OPTET (Grant no.317631).

References

1. Aksoy, D., Franklin, M.: R×W: a scheduling approach for large-scale on-demand data broadcast. IEEE/ACM Trans. Network. **7**(6), 846–860 (1999)
2. Backes, M., Hritcu, C., Maffei, M.: Automated verification of remote electronic voting protocols in the applied pi-calculus. In: Proceedings of the 21st IEEE Computer Security Foundations Symposium (CSF 2008), pp. 195–209, Pittsburgh, 23–25 June 2008 (2008). http://doi.ieeecomputersociety.org/10.1109/CSF.2008.26
3. Baiardi, F., Falleni, A., Granchi, R., Martinelli, F., Petrocchi, M., Vaccarelli, A.: Seas, a secure e-voting protocol: design and implementation. Comput. Secur. **24**(8), 642–652 (2005). http://dx.doi.org/10.1016/j.cose.2005.07.008

4. Benkaouz, Y., Erradi, M.: A distributed protocol for privacy preserving aggregation with non-permanent participants. Computing. J. **3**, 1–20 (2014). doi:10.1007/s00607-013-0373-6
5. Benkaouz, Y., Guerraoui, R., Erradi, M., Huc, F.: A distributed polling with probabilistic privacy. In: IEEE 32nd Symposium on Reliable Distributed Systems (SRDS 2013), pp. 41–50, Braga, 1–3 October 2013 (2013). http://dx.doi.org/10.1109/SRDS.2013.13
6. Camenisch, J.L., Lysyanskaya, A.: An efficient system for non-transferable anonymous credentials with optional anonymity revocation. In: Pfitzmann, B. (ed.) EUROCRYPT 2001. LNCS, vol. 2045, pp. 93–118. Springer, Heidelberg (2001)
7. Chaum, D.: Security without identification: transaction systems to make big brother obsolete. Commun. ACM **28**(10), 1030–1044 (1985). http://doi.acm.org/10.1145/4372.4373
8. Chen, Y., Jan, J., Chen, C.: The design of a secure anonymous internet voting system. Comput. Secur. **23**(4), 330–337 (2004). http://dx.doi.org/10.1016/j.cose.2004.01.015
9. Clarkson, M.R., Chong, S., Myers, A.C.: Civitas: Toward a secure voting system. In: 2008 IEEE Symposium on Security and Privacy (S&P 2008), pp. 354–368, Oakland, 18–21 May 2008. http://dx.doi.org/10.1109/SP.2008.32
10. Cortier, V., Smyth, B.: Attacking and fixing helios: an analysis of ballot secrecy. J. Comput. Secur. **21**(1), 89–148 (2013)
11. Cranor, L.F., Cytron, R.: Sensus: a security-conscious electronic polling system for the internet. In: 30th Annual Hawaii International Conference on System Sciences (HICSS-30), pp. 561–570, Maui, 7–10 January 1997. http://doi.ieeecomputersociety.org/10.1109/HICSS.1997.661700
12. Degroot, M.H.: Reaching a consensus. J. Am. Stat. Assoc. **69**(345), 118–121 (1974)
13. Dykeman, H.D., Ammar, M.H., Wong, J.W.: Scheduling algorithms for videotex systems under broadcast delivery. In: Proceedings of the International Conference on Communications (ICC 1986), pp. 1847–1851, Toronto, June 1986
14. Englert, B., Gheissari, R.: Multivalued and deterministic peer-to-peer polling in social networks with reputation conscious participants. In: 12th IEEE International Conference on Ubiquitous Computing and Communications (IUCC-2013), pp. 895–902, Melbourne, July 16–18 2013. http://dx.doi.org/10.1109/TrustCom.2013.109
15. Fan, C., Sun, W.: An efficient multi-receipt mechanism for uncoercible anonymous electronic voting. Math. Comput. Model. **48**(9–10), 1611–1627 (2008). http://dx.doi.org/10.1016/j.mcm.2008.05.039
16. Frith, D.: E-voting security: hope or hype? Netw. Secur. **2007**(11), 14–16 (2007)
17. Gambs, S., Guerraoui, R., Harkous, H., Huc, F., Kermarrec, A.: Scalable and secure polling in dynamic distributed networks. In: IEEE 31st Symposium on Reliable Distributed Systems (SRDS 2012), pp. 181–190, Irvine, 8–11 October 2012. http://dx.doi.org/10.1109/SRDS.2012.63
18. Greensmith, J., Aickelin, U., Tedesco, G.: Information fusion for anomaly detection with the dendritic cell algorithm. Inf. Fusion **11**(1), 21–34 (2010)
19. Gritzali, D.: Principles and requirements for a secure e-voting system. Comput. Secur. **21**(6), 539–556 (2002). http://dx.doi.org/10.1016/S0167-4048(02)01014–3
20. Guerraoui, R., Huguenin, K., Kermarrec, A., Monod, M., Vigfusson, Y.: Decentralized polling with respectable participants. J. Parallel Distrib. Comput. **72**(1), 13–26 (2012). http://dx.doi.org/10.1016/j.jpdc.2011.09.003
21. Hoang, B., Imine, A.: Efficient polling protocol for decentralized social networks. CoRR abs/1412.7653 (2014). http://arxiv.org/abs/1412.7653

22. Joaquim, R., Zúquete, A., Ferreira, P.: Revs-a robust electronic voting system. IADIS Int. J. WWW/Internet **1**(2), 47–63 (2003)
23. Jonker, H., Mauw, S., Pang, J.: Privacy and verifiability in voting systems: Methods, developments and trends. Comput. Sci. Rev. **10**, 1–30 (2013). http://dx.doi.org/10.1016/j.cosrev.2013.08.002
24. Li, C., Hwang, M., Liu, C.: An electronic voting protocol with deniable authentication for mobile ad hoc networks. Comput. Commun. **31**(10), 2534–2540 (2008). http://dx.doi.org/10.1016/j.comcom.2008.03.018
25. Liaskos, C., Petridou, S., Papadimitriou, G.: Towards realizable, low-cost broadcast systems for dynamic environments. IEEE/ACM Trans. Netw. **19**(2), 383–392 (2011)
26. Liaskos, C., Papadimitriou, G., Douligeris, C.: Sensor swarm query filtering: heightened attack resilience for broadcast on-demand services. In: IEEE Symposium on Computers and Communications (ISCC 2013), pp. 000312–000317. IEEE (2013)
27. Liaskos, C., Tsioliaridou, A., Papadimitriou, G., Nicopolitidis, P.: Minimal wireless broadcast schedules for multi-objective pursuits. IEEE Transactions on Vehicular Technology p. preprint (2014)
28. Malkhi, D., Margo, O., Pavlov, E.: E-voting without cryptography. In: Financial Cryptography, 6th International Conference (FC 2002), pp. 1–15, Southampton, 11–14 March 2002. http://dx.doi.org/10.1007/3-540-36504-4_1
29. Meng, B.: A critical review of receipt-freeness and coercion-resistance. Inf. Technol. J. **8**(7), 934–964 (2009)
30. Meng, B., Li, Z., Qin, J.: A receipt-free coercion-resistant remote internet voting protocol without physical assumptions through deniable encryption and trapdoor commitment scheme. J. Softw. **5**(9), 942–949 (2010)
31. Misra, J., Gries, D.: Finding repeated elements. Sci. Comput. Program. **2**(2), 143–152 (1982)
32. Pardede, E., Taniar, D., Awan, I., Al-Sudani, W., Gill, A., Li, C., Wang, J., Liu, F.: Protection through multimedia CAPTCHAs. In: Proceedings of the 8th International Conference on Advances in Mobile Computing and Multimedia (MoMM 2010), p. 63. ACM Press (2010)
33. Pietronero, L., Tosatti, E., Tosatti, V., Vespignani, A.: Explaining the uneven distribution of numbers in nature: the laws of Benford and Zipf. Physica A **293**(1–2), 297–304 (2001)
34. Qadah, G.Z., Taha, R.: Electronic voting systems: Requirements, design, and implementation. Computer Standards Interfaces **29**(3), 376–386 (2007). http://dx.doi.org/10.1016/j.csi.2006.06.001
35. Sampigethaya, K., Poovendran, R.: A framework and taxonomy for comparison of electronic voting schemes. Comput. Secur. **25**(2), 137–153 (2006). http://dx.doi.org/10.1016/j.cose.2005.11.003
36. Sharaf, M.A., Chrysanthis, P.: On-Demand Broadcast: new Challenges and Scheduling Algorithms. In: Proceedings of the 1st Hellenic Conference on the Management of Data (2002)
37. Sieka, B., Kshemkalyani, A.D., Singhal, M.: On the security of polling protocols in peer-to-peer systems. In: 4th International Conference on Peer-to-Peer Computing (P2P 2004), pp. 36–44, Zurich, 15–17 August 2004. http://doi.ieeecomputersociety.org/10.1109/PTP.2004.1334929
38. Smart, M., Ritter, E.: True trustworthy elections: remote electronic voting using trusted computing. In: Calero, J.M.A., Yang, L.T., Mármol, F.G., García Villalba, L.J., Li, A.X., Wang, Y. (eds.) ATC 2011. LNCS, vol. 6906, pp. 187–202. Springer, Heidelberg (2011)

39. Tsioliaridou, A., Zhang, C., Liaskos, C.: Fast and fair handling of multimedia captcha flows. International Journal of Interactive Mobile Technologies (2015). (To appear)
40. XJ Technologies: The AnyLogic Simulator (2013). http://www.xjtek.com/anylogic/
41. Zwierko, A., Kotulski, Z.: A light-weight e-voting system with distributed trust. Electr. Notes Theor. Comput. Sci. **168**, 109–126 (2007). http://dx.doi.org/10.1016/j.entcs.2006.12.004