**TTLed Random Walks for Collaborative Monitoring**

The MIT Faculty has made this article openly available. **Please share** how this access benefits you. Your story matters.

<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>As Published</td>
<td><a href="http://dx.doi.org/10.1109/INFCOMW.2010.5466697">http://dx.doi.org/10.1109/INFCOMW.2010.5466697</a></td>
</tr>
<tr>
<td>Publisher</td>
<td>Institute of Electrical and Electronics Engineers</td>
</tr>
<tr>
<td>Version</td>
<td>Final published version</td>
</tr>
<tr>
<td>Accessed</td>
<td>Sun Apr 07 23:48:02 EDT 2019</td>
</tr>
<tr>
<td>Citable Link</td>
<td><a href="http://hdl.handle.net/1721.1/61946">http://hdl.handle.net/1721.1/61946</a></td>
</tr>
<tr>
<td>Terms of Use</td>
<td>Article is made available in accordance with the publisher's policy and may be subject to US copyright law. Please refer to the publisher's site for terms of use.</td>
</tr>
<tr>
<td>Detailed Terms</td>
<td></td>
</tr>
</tbody>
</table>
TTLed Random Walks for Collaborative Monitoring

(Preliminary Version)

Yaniv Altshuler
Deutsche Telekom Labs
Ben Gurion University, Israel
Email: yanival@cs.technion.ac.il

Shlomi Dolev
Computer Science Department
Ben Gurion University, Israel
Email: dolev@cs.bgu.ac.il

Yuval Elovici
Deutsche Telekom Labs
Ben Gurion University, Israel
Email: elovici@bgu.ac.il

Nadav Aharony
Media Lab
MIT, Cambridge, MA, USA
Email: nadav@media.mit.edu

Abstract—In this paper we discuss the problem of collaborative monitoring of applications that are suspected of being malicious. New operating systems for mobile devices allow their users to download millions of new applications created by a great number of individual programmers and companies, some of which may be malicious or flawed. The importance of defense mechanisms against an epidemic spread of malicious applications in mobile networks was recently demonstrated by Wang et. al [21]. In many cases, in order to detect that an application is malicious, monitoring its operation in a real environment for a significant period of time is required. Mobile devices have limited computation and power resources and thus can monitor only a limited number of applications that the user downloads. In this paper we propose an efficient collaborative application monitoring algorithm called **TPP** - Time-To-Live Probabilistic Flooding, harnessing the collective resources of many mobile devices. Mobile devices activating this algorithm periodically monitor mobile applications, derive conclusion concerning their maliciousness, and report their conclusions to a small number of other mobile devices. Each mobile device that receives a message (conclusion) propagates it to one additional mobile device. Each message has a predefined **TTL**. The algorithm’s performance is analyzed and its time and messages complexity are shown to be significantly lower compared to existing state of the art information propagation algorithms. The algorithm was also implemented and tested in a simulated environment.

I. INTRODUCTION

Companies that are distributing new mobile devices operating systems had created a market place that motivates individuals and other companies to introduce new applications (such as Apple’s App Store Google’s Android Market, Nokia’s Ovi Store and others). The content of the marketplace is not verified by the marketplace operators and thus there is no guarantee that the marketplace does not contain malicious or severely flawed applications. Downloading a malicious application from the marketplace is not the only way that a mobile device may be infected by malicious code. This may also happen as a result of a malicious code that manages to exploit a vulnerability in the operating systems and applications or through one of the mobile phone communication channels such as Bluetooth, Wi-Fi, or other channels [10], [21].

In many cases, in order to detect that an application is malicious, monitoring its operation in a real environment for a significant period of time is required. The monitored data is being processed using advanced algorithms in order to assess the maliciousness of the application [9], [12], [13].

Harnessing their collective resources a large group of limited devices can achieve an efficient decentralized information propagation capability. This allows participating users to significantly improve their “defense utilization” — the ratio between the local resources required for the collaborative service, and the probability to block attack attempts.

In this work we present a collaborative application monitoring algorithm that provides high efficiency, scalability and robustness. The algorithm is completely decentralized and no supervising authority is assumed, nor do any central of hierarchical tasks allocation or any kind of shared memory. Specifically, we show that by sending $O(\ln n)$ messages, the number of applications a device would have to monitor in order to become “vaccinated” (a device is considered vaccinated when it is aware of the vast majority of the malicious applications) is reduced by a factor of $O(\ln n)$. Using real-world numbers, implemented as a service executed by 1,000,000 units, assuming 10,000 new applications are released every month, Theorem 3 implies that by monitoring a single application each month and sending 4 SMS messages per day, a participating mobile device can be guaranteed to be immune for 99% of all malicious applications.

Several related works are discussed in Section II, and are analytically compared to the proposed **TPP** algorithm. The formal definition of the problem appears in Section III. The **TPP** algorithm is presented and analyzed in Section IV (details and some proofs are omitted from this preliminary version). Experimental results are presented in Section V.

II. RELATED WORK

Since the problem of finding the minimum energy transmission scheme for broadcasting a set of messages in a given network is known to be NP-Complete [2], flooding optimization often relies on approximation algorithms. For example, in [8], [16] messages are forwarded according to a set of predefined probabilistic rules, whereas in [15] a deterministic algorithm which approximates the connected dominating set within a two-hop neighborhood of each node is proposed.

In this work we applied a different approach — instead of a probabilistic forwarding of messages, we assign a **TTL** value for each message, using which we are able to guide the flooding process. Short random walks have been used in the past, albeit for different tasks [5].
The full text paper was peer reviewed at the direction of IEEE Communications Society subject matter experts for publication in the IEEE INFOCOM 2010 proceedings.

It is well known that the basic flooding algorithm, assuming a single source of information, guarantees completion in a worse case cost of $O(n^2)$ messages and time equals to the graph’s diameter, which in the case of a random graph $G(n, p)$ is approximately $O(\log n)$ [3]. Variants of flooding algorithms use various methods to improve the efficiency of the basic algorithm, such as area-based methods [14] or neighborhood knowledge methods [17]. An extremely efficient flooding algorithms in terms of completion time, is the network coded flooding algorithm, discussed in [4]. In this work, a message is forwarded by any receiving vertex $\frac{k}{d(n)}$ times, while $k$ is a parameter which depends on the network’s topology. Using this method, the algorithm achieves a completion time of approximately $O\left(\frac{n^3}{\log^2 n}\right)$. This algorithm, however, is still outperformed by our proposed algorithm. Specifically, our algorithm performs faster in graphs with average degree of less than $O\left(\sqrt{\frac{\log n}{n}}\right)$.

An alternative approach to be mentioned in this scope is the use of epidemic algorithms [19]. There exist a variety of epidemic algorithms, starting with the basic epidemic protocol [7], through neighborhood epidemics [6] and up to hierarchical epidemics [18]. In general, all the various epidemic variants has a trade-off between number of messages sent, completion time, and previous knowledge required for the protocols.

Tables I and II present a summary of the performance of the TPP algorithm and available state of the art algorithms.

### Table I. Performance Comparison between the TPP Algorithm and Available State of the Art Algorithms.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Time</th>
<th>Messages</th>
</tr>
</thead>
<tbody>
<tr>
<td>Flooding</td>
<td>$O\left(\frac{n^2}{\log n}\right)$</td>
<td>$O(n \log n)$</td>
</tr>
<tr>
<td>Network Coded Flooding [4]</td>
<td>$O\left(\frac{n}{\log n}\right)$</td>
<td>$O\left(\frac{n}{\log n}\right)$</td>
</tr>
<tr>
<td>Neighborhood Epidemics [6]</td>
<td>$O\left(\frac{n^2}{\log n}\right)$</td>
<td>$O\left(\frac{n^2}{\log n}\right)$</td>
</tr>
<tr>
<td>Hierarchical Epidemics [18]</td>
<td>$O\left(\frac{n^2}{\log n}\right)$</td>
<td>$O\left(\frac{n^2}{\log n}\right)$</td>
</tr>
<tr>
<td>LRTA* [11] in planar degree bounded graphs</td>
<td>$O(n^2)$</td>
<td>$O(n^2)$</td>
</tr>
<tr>
<td>SWEEP [20] in the $Z^2$ grid</td>
<td>$O(n^{1.5})$</td>
<td>$O(n^{1.5})$</td>
</tr>
</tbody>
</table>

### Table II. Performance Comparison for Random $G(n, p)$ Graphs, with $p < O(\frac{n \log n}{n})$.

For some malicious application $a_i$, let $p_{a_i}$ denote the application’s penetration probability — the probability that given some arbitrary device $v$, it is unaware of the maliciousness of $a_i$. The penetration probability of every new malicious application is 1. Our goal is to verify that at the end of the month, the penetration probability of all malicious applications released during this month would be lower than a penetration threshold $p_{\text{MAX}}$, resulting in a “vaccination” of the network with regards to these applications. Formally, we require that:

$$\forall i \text{ Malicious application } a_i \quad \text{Prob}(p_{a_i} > p_{\text{MAX}}) < \epsilon$$

The rational behind the use of the threshold $p_{\text{MAX}}$ is to increase the utilization of collective resources, focusing on the system’s main threats. Given additional resources, $p_{\text{MAX}}$ could be decreased appropriately by the network operator, resulting in a tighter defense net (albeit, using more messages). In addition, it is likely that malware kept confined to a small portion of the network would have far less hazardous potential. A demonstration of the correlation between the penetration probability of malicious applications in mobile networks and their damage potential can be seen in [21].

We assume that any device $v$ can send a message of some short content to any other device $u$. In addition, we assume that at the initialization phase of the algorithm each device is given a list containing the addresses of some $X$ random network members. This can be implemented either by the network operator, or by distributively constructing and maintaining a random network overlay service.

We assume that every device is equipped with a way of locally monitoring applications that are installed on it [1], [13]. However, this process is assumed to be rather expensive (in terms of the device’s battery), and should therefore be executed as fewer times as possible. The result of an application monitoring is a non-deterministic boolean function:

$$\text{Monitoring}(x) : A \rightarrow \{\text{true}, \text{false}\}$$

False-positive and false-negative error rates of the monitoring process shall be denoted as follows:

$$P(\text{Monitoring}(a_i) = \text{true} \mid A_i \text{ is not malicious}) = E_+$$
$$P(\text{Monitoring}(a_i) = \text{false} \mid A_i \text{ is malicious}) = E_-$$

In this work we assume that the monitoring algorithm is calibrated in such a way that $E_+$ is rather small.
As we rely on the propagation of information concerning the maliciousness of applications, this method might be abused by injection of inaccurate information. This may be the result of a deliberate attack, aimed for “framing” a benign application, or simply as a consequence of a false-positive result of the monitoring function. Therefore, in order for a unit to determine that an application \( a_i \) is malicious, it needs for one of the following to hold:

- Unit \( v \) had monitored \( a_i \) and found it to be malicious.
- Unit \( v \) had received at least \( \rho \) alerts concerning \( a_i \) from different sources (for some decision threshold \( \rho \)).

IV. ALGORITHM, CORRECTNESS & ANALYSIS

Note again that the TPP algorithm is executed by each device separately and asynchronously, where no supervised or hierarchical allocation of tasks, as well as any kind of shared memory are required. The conceptual basis of the TPP algorithm relies on the fact that although the application monitoring process is relatively expensive (in terms of battery and CPU resources), almost every application is installed on a great number of devices and therefore once identified as malicious by one (or few) devices, this information can be rapidly propagated throughout the network, resulting in a low amortized cost. This process is hereafter denoted as a vaccination of the network. A detailed implementation of the TPP algorithm appears in Algorithm 1.

At the algorithm’s initialization (lines 1 through 8), all the applications which are installed on the device are added to a list of suspected applications. In addition, another two (empty) lists of known malicious applications and known safe applications are created. Once an application is determined as malicious, it is added to the known malicious applications list. In case this application was also in the suspected applications list (namely, it is installed on the device, but has not been monitored yet), it is deleted from that list. Once a new application is encountered it is compared to the known malicious applications list, and if found, an alert is sent to the user. If the new application is not yet known to be malicious, the application is added to the suspected applications list.

As long as the vaccination process is enabled, a periodic selection of an arbitrary application from the list of suspected applications is done, once every \( T \) steps (lines 26 through 36). The selected application is then monitored for a given period of time, in order to discover whether it is of malicious properties. In case no maliciousness traces are found, the application is removed from the list of suspected applications and is added instead to the list of known safe applications (lines 28 and 35). However, if the application is found to be malicious, it is removed from the list of suspected applications and added to the known malicious applications list (lines 28 and 31). In addition, an appropriate alert is produced and sent to another random network member, while decreasing the value of TTL by 1. Once TTL reaches zero, the forwarding process of this message stops (lines 24 and 25).

A device may also classify an application as malicious as a result of receiving an alert message concerning this application (lines 16 through 23). In order to protect begins applications from being “framed” (reported as being malicious by adversaries abusing the vaccination system), this is done only after receiving at least \( \rho \) messages concerning this application, from different sources.

<table>
<thead>
<tr>
<th>Line</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Initialization</td>
</tr>
<tr>
<td>2</td>
<td>Let ( A(v) ) be the list of installed applications</td>
</tr>
<tr>
<td>3</td>
<td>Let ( \tilde{A}(v) ) be the list of suspected applications</td>
</tr>
<tr>
<td>4</td>
<td>Let ( \tilde{A}(v) ) be the list of applications known to be safe</td>
</tr>
<tr>
<td>5</td>
<td>Let ( \tilde{A}(v) ) be the list containing known malicious applications</td>
</tr>
<tr>
<td>6</td>
<td>Initialize ( \tilde{A}(v) \leftarrow A(v) )</td>
</tr>
<tr>
<td>7</td>
<td>Initialize ( A(v) \leftarrow \emptyset )</td>
</tr>
<tr>
<td>8</td>
<td>Initialize ( \tilde{A}(v) \leftarrow \emptyset )</td>
</tr>
</tbody>
</table>

**Algorithm 1: TTL Probabilistic Propagation**

Analyzing the correctness and performance of the TPP algorithm we imagine a directed Erdos-Renyi random graph \( G(V,E) \sim G(n,p_N) \), where \( p_N = \frac{A}{n} \). The graph’s vertices \( V \) denote the network’s devices, and the graph’s edges \( E \) represent messages forwarding connections between the devices, carried out during the execution of the TPP algorithm. Observing some malicious application \( a_i \), every once in a while some device which \( a_i \) is installed on randomly selects it for monitoring. With probability \((1 - E_{\text{--}})\) the device discovers that \( a_i \) is malicious and issues a report to \( X \) random members. We treat these reports as “agents” and are interested in:

- The time it takes the graph to be explored by those agents.
- Namely, the time after which every device was visited by at least \( \rho \) agents, thus becoming immune to \( a_i \).
- Total number of messages sent during this process.
- The minimal TTL which guarantees a successful vacci-
nation of the network.

Operating in a random graph, we can assume that the initial locations of the agents are uniformly and randomly spread along the vertices of V. In compliance with the instruction of the TPP algorithm, the movement of the agents is done according to the random walk algorithm.

The expected number of new agents created at time t, denoted as \( \hat{k}(t) \), equals:

\[
\hat{k}(t) = \frac{n^2 \cdot p_{a_i} \cdot p_N}{T \cdot N} (1 - E) 
\]

and the accumulated number of agents which have been generated in a period of t time-steps is therefore:

\[
k_t = \sum_{i \leq t} \hat{k}(i) 
\]

The value of timeout (the assigned TTL) will be selected in such a way that the complete coverage of the graph, and therefore its vaccination against \( a_i \), is guaranteed (in probability greater than 1 - \( \epsilon \)). We now artificially divide the mission into two phases, the first containing the generation of agents and the second discussing the coverage of the graph. Denoting the completion time by \( T_{Vac} \) we therefore have:

\[
T_{Vac} \leq T_{Generation} + T_{Propagation} 
\]

It is easy to see that essentially \( T_{Propagation} \triangleq \) timeout. Now, let us artificially set

\[
\text{timeout} = \lambda \cdot (T_{Generation} + \text{timeout}) 
\]

\[
T_{Generation} = (1 - \lambda) \cdot (T_{Generation} + \text{timeout}) 
\]

From this we can see that:

\[
T_{Generation} = \frac{(1 - \lambda)}{\lambda} \cdot \text{timeout} 
\]

We now examine the case of \( \lambda = 0.5 \), in which case:

\[
T_{Vac} \leq 2 \cdot \text{timeout} 
\]

Let us denote the number of agents created in the first \( T_{Generation} \) time-steps by \( k = kT_{Generation} \). We now find the time it takes those \( k \) agents to completely cover the graph\( G \). and from this, derive the value of timeout. Remembering that the initial locations of the agents are random, their movements are random and the graph \( G \) is random, we can see that the locations of the agents after every step are purely random over the nodes. It can be easily shown that the number of incoming neighbors for each vertex \( v \) is at least \( \frac{1}{2} p_N \cdot n \). Therefore, for every \( v \) the expected number of agents that reside in distance 1 from \( v \) after every step is at least \( \frac{n^2 \cdot p_{a_i} \cdot p_N}{T \cdot N} \).

**Lemma 1.** For any \( v \in V \), the probability of \( v \) being notified at the next time-step that \( a_i \) is malicious is at least \( 1 - e^{-\frac{1}{2}\frac{p_{a_i} \cdot p_N}{T \cdot N}} \).

Let us denote by \( \rho \)-coverage the exploration of each vertex by at least \( \rho \) different agents.

**Theorem 1.** The time it takes \( k \) random walkers to complete a \( \rho \)-coverage of \( G \) in probability greater than \( 1 - \epsilon \) is:

\[
T(n) \geq \frac{2 (\rho - \ln \frac{n}{\epsilon})}{1 - e^{-\frac{1}{2}}} 
\]

Proof: Lemma 1 states the probability that some vertex \( v \in V \) will be reported of \( a_i \) at the next time-step. This is in fact a Bernoulli trial with \( p_{success} = 1 - e^{-\frac{1}{2}\frac{p_{a_i} \cdot p_N}{T \cdot N}} \). We bound the probability of failing this trial (not notifying vertex \( v \) enough times) after \( m \) steps. Let \( X_v(m) \) denote the number of times that a notification message had reached \( v \) after \( m \) steps, and \( F_v(m) \) the event that \( v \) was not notified enough times after \( m \) steps (i.e. \( X_v(m) < \rho \)). We additionally denote by \( F(m) \) the event that one of the vertices of \( G \) where not notified enough times after \( m \) steps (i.e. \( \bigcup_{v \in V(G)} F_v(m) \)). We use the Chernoff bound:

\[
P[X_v(m) < (1 - \delta)p_{success}m] < e^{-\delta^2 \frac{p_{success}}{2}} 
\]

in which we set \( \delta = 1 - \frac{\epsilon}{m_{p\text{success}}} \). We can now see that:

\[
P[F_v(m)] = e^{\rho - \frac{m_{p\text{success}}}{2}} 
\]

This bound is strong enough for applying the union bound:

\[
P[e_1 \cup e_2 \cup \ldots \cup e_n] \leq P[e_1] + P[e_2] + \ldots + P[e_n]
\]

on all \( n \) vertices of \( G \). Therefore we can bound the probability of failure on any vertex \( v \) (using Lemma 1) as follows:

\[
Pr[F(m)] \leq n e^{\rho - \frac{m_{p\text{success}}}{2}} \leq n e^{\rho - \frac{\ln(1 - \epsilon)}{2}} \leq \epsilon
\]

We now show how to select a value of \( \text{timeout} \) that guarantees a successful vaccination process:

**Theorem 2.** In order for the TPP algorithm to guarantee a successful vaccination process for some penetration threshold \( p_{MAX} \) in probability greater than \( 1 - \epsilon \), the value of \( \text{timeout} \) should satisfy the following expression:

\[
\text{timeout} \frac{2 (\rho - \ln \frac{n}{\epsilon})}{1 - e^{-\frac{1}{2}}} = 1
\]

Proof: The number of agents \( k \) in Theorem 1 equals:

\[
k = \sum_{i \leq T_{Generation}} \frac{n^2 \cdot p_{a_i} \cdot p_N}{T \cdot N} (1 - E)
\]

The goal of the TPP vaccination algorithm is to decrease the penetration probability of \( a_i \) below the threshold \( p_{MAX} \). Until the process is completed, we can therefore assume that this probability never decreases below \( p_{MAX} \).

Therefore, we can bound the number of agents as follows:

\[
k \geq \text{timeout} \cdot \frac{n^2 \cdot p_{MAX} \cdot p_N}{T \cdot N} (1 - E)
\]

Assigning \( timeout = m \) we can now write:

\[
\text{timeout} = \frac{2 (\rho - \ln \frac{n}{\epsilon})}{1 - e^{-\frac{1}{2}}} \leq \frac{2 (\rho - \ln \frac{n}{\epsilon})}{1 - e^{-\frac{\ln(1 - \epsilon)}{2}}} = 1
\]

From the value of \( \text{timeout} \) stated in Theorem 2, the vaccination time \( T_{Vac} \) as well as the overall cost of the TPP algorithm can now be extracted. The cost of the algorithm is measured as a combination of the overall number of messages sent during its execution and the total number of monitoring
Corollary 1. For any timeout = τ which satisfies Theorem 2, the TPP algorithm’s time and cost can be expressed as:

\[ T_{vac} = O(\tau) \quad M = O(k \cdot \tau \cdot C_S + \frac{k}{X} \cdot C_M) = O\left(\frac{p_{MAX} \cdot p_N}{n-2T \cdot N} \cdot (1 - E_-) \cdot \left(\frac{\tau^2 \cdot C_S + \frac{\tau}{n \cdot p_N} \cdot C_M}{(1 - E_-)}\right)\right) \]

We assume that \( \epsilon \) is polynomial in \( \frac{1}{n} \), namely: \( \epsilon = n^{-\alpha} e^{2\tau} \).

Using the bound \( (1 - x) < e^{-x} \) for \( x < 1 \) we can see that when assuming that:

\[\text{timeout} \cdot \frac{n \cdot p_{MAX} \cdot p_N}{2T \cdot N} \cdot (1 - E_-) < 1\]

Theorem 2 can be written as:

\[ \rho + (\alpha + 1) \ln n \geq \text{timeout}^2 \cdot \frac{n \cdot p_{MAX} \cdot p_N}{4T \cdot N} \cdot (1 - E_-)^{-1}\]

and therefore:

\[ \text{timeout} \leq \sqrt{\frac{4T \cdot N (\rho + (\alpha + 1) \ln n)}{n \cdot p_{MAX} \cdot p_N \cdot (1 - E_-)}} \]

Assigning this approximation of timeout into the assumption above results with the sparse connectivity assumption:

Definition 1. Let a network be sparsely connected when:

\[ p_N < \frac{T \cdot N}{n \cdot p_{MAX} \cdot (\rho + (\alpha + 1) \ln n)(1 - E_-)} \]

We can now obtain the algorithm’s completion time and cost:

Theorem 3. Under the sparse connectivity assumption, the completion time of the TPP algorithm is:

\[ T_{vac} \leq 4 \sqrt{\frac{T \cdot N (\rho + (\alpha + 1) \ln n)}{n \cdot p_{MAX} \cdot p_N \cdot (1 - E_-)}} \]

Theorem 4. Under the sparse connectivity assumption, the overall cost of the TPP algorithm is:

\[ M \leq k \cdot \text{timeout} \cdot C_S + \frac{k}{X} \cdot C_M \leq 4n (\rho + (\alpha + 1) \ln n) C_S + 2CM \sqrt{\frac{n (\rho + (\alpha + 1) \ln n) p_{MAX}}{p_N \cdot T \cdot N(1 - E_-)^{-1}}} \]

Proof: By assigning the approximated value of timeout into Corollary 1.

Using the sparse connectivity assumption as an upper bound for \( p_N \), and \( \frac{\ln n}{n} \) as a lower bound for \( p_N \) which guarantees connectivity [3], the following corollaries are obtained:

Corollary 2. The completion time of the TPP algorithm is:

\[ T_{vac} = O\left(\rho + \ln n + \frac{T \cdot N(1 - E_-)^{-1}}{p_{MAX} \cdot \ln n}\right) \]

Note that although \( O\left(\frac{T \cdot N}{p_{MAX}(1 - E_-)}\right) \) is allegedly independent of \( n \), by assigning the lower bound \( p_N > \frac{\ln n}{n} \) into the sparse connectivity assumption, we can see that:

\[ \zeta_{T,N,P,M,E_-} \triangleq \frac{T \cdot N}{p_{MAX}(1 - E_-)} = \Omega(\rho \ln n + \ln^2 n) \]

The algorithm’s cost can similarly be approximated as:

Corollary 3. The overall cost of the TPP algorithm is:

\[ M = O\left(\rho \cdot \text{timeout} \cdot C_S + \frac{k}{X} \cdot C_M\right) = O\left(\frac{(n \rho + n \ln n) C_S + \left(\frac{n}{\ln n} + n(\rho + \ln n)\right)^{-1}}{p_{MAX} \cdot E_M} C_M\right) \]

In networks of \( E_- < 1 - o(1) \), provided that \( \rho = O(\ln n) \), and remembering that in this case \( \zeta_{T,N,P,M,E_-} = \Omega(\ln^2 n) \) we can see that the dominant components of Corollary 3 become:

\[ M = O\left(n \ln n C_S + \frac{n}{\ln n} C_M\right) \]

V. EXPERIMENTAL RESULTS

In order to examine its performance, we have implemented the TPP algorithm and conducted extensive simulations using various scenarios. In this section we describe one example, due to space considerations. This example concerns a network of \( n = 1000 \) units, having access to \( N = 100 \) applications, one of which was malicious 1. Each unit is assumed to download 30 random applications, monitoring 1 application every week, and allowed to send notification messages to 10 random network members (namely, \( p_N = 0.01 \)). Upon completion, at least 990 network members are required to become aware of the malicious application (namely, \( p_{MAX} = 0.01 \)), and that this would hold in probability of 0.999. In addition, we assumed that among the network members there are 100 adversaries, whose goal is to mislead at least 50 of the network’s members to believe that some benign application is malicious.

Figure 1 shows the time (in days) and messages required in order to complete this mission, as a function of the decision threshold \( \rho \). We can see that whereas the adversaries succeed in probability 1 for \( \rho < 3 \), they fail in probability 1 for any \( \rho \geq 3 \). Note the extremely efficient performance of the algorithm, with completion time of \( \sim 260 \) days using only 5 messages and at most 30 monitored applications per user. The same scenario would have resulted in 100 messages per user using the conventional flooding algorithm, or alternatively, in 700 days and 100 monitored applications per user using a non-collaborative scheme. Figure 2 demonstrates the decrease in completion time and messages requirement as a result of decreasing the penetration threshold \( p_{MAX} \). A similar example concerning the effect of changing the graph’s density \( p_N \) is given in Figure 3. Figure 4 demonstrates the evolution in the malicious application’s penetration probability throughout the vaccination process.

1 Note that the number of malicious applications does not influence the completion time of algorithm, as monitoring and notification is done in parallel. The number of message, however, grows linearly with the number of malicious applications.
Fig. 1. An experimental result of a network of $n = 1000$ members, with $N = 100$ applications, $p_{\text{MAX}} = 0.01$, $p_N = 0.01$ and 100 adversaries that try to mislead at least 5% of the network into believing that some benign application is malicious. Notice how changes in $\rho$ dramatically effect the adversaries’ success probability, with almost no effect on the completion time.

Fig. 2. The effect of decreasing the penetration threshold $p_{\text{MAX}}$ on the algorithm’s completion time and number of messages ($\rho = 1$).

Fig. 3. The effect of decreasing the graph density $p_N$ on the algorithm’s completion time and number of messages ($\rho = 1$).

Fig. 4. The penetration probability of the malicious application, as a function of the time, with $\rho = 1$ (on the left) and $\rho = 20$ (on the right).

VI. CONCLUSIONS AND FUTURE WORK

In this work we have presented the TPP collaborative application monitoring algorithm, and analytically shown its superior performance compared to the state of the art in this domain. These results were also demonstrated experimentally. Future versions of this work should investigate the selection of the decision threshold $\rho$, aiming for analytical bounds on the probability of adversarial attacks. In addition, the effect of additional kinds of adversarial attacks should be studied.

REFERENCES