Blue-Watchdog: Detecting Bluetooth Worm Propagation in Public Areas

Guanhua Yan Leticia Cuellar Stephan Eidenbenz Nicolas Hengartner
Los Alamos National Laboratory*
{ghyan, leticia, eidenben, nickh}@lanl.gov

Abstract

The rising popularity of mobile devices, such as cellular phones and PDAs, has made them a lucrative playground for mobile malware propagation. One common infection vector exploited by these mobile malware is Bluetooth. In this paper, we propose an architecture called Blue-Watchdog that detects Bluetooth worm propagation in public areas based on statistical methods. To achieve fast and accurate Bluetooth worm detection, Blue-Watchdog monitors abrupt changes of average paging rate per Bluetooth device from both temporal and temporal-spatial perspectives. The temporal scheme relies on the CUSUM (Cumulative Sum) sequential test together with the generalized likelihood ratio (GLR), and the temporal-spatial scheme aims to identify spatial regions with abnormally frequent paging attempts. Experimental results show that Blue-Watchdog not only has low false alarm rates, but also effectively detects Bluetooth worms that spread quickly in areas where Bluetooth devices are greatly mixed due to high mobility and also those that propagate relatively slowly in a spatially constrained fashion.

Keywords: *Bluetooth, Bluetooth worms, CUSUM, temporal detection, temporal-spatial detection*

1 Introduction

The increasing popularity of mobile devices, such as cellular phones and PDAs, has created a new lucrative playground for mobile malware propagation. One common infection vector exploited by these mobile malware is Bluetooth. Bluetooth is a short-range radio technology that is aimed at connecting different wireless devices at low power consumption and at low cost. As the market for Bluetooth devices has been growing tremendously recently, malware targeted at them have been on the rise as well. Malware such as Cabir [11], Commwarrior [8] and Inqtana.A [14] have been spotted using Bluetooth to spread on cell phones and laptops.

Mobile malware such as Bluetooth worms have posed serious security threats to both mobile device users and mobile network operators, including information stealing [10] and power exhaustion of mobile devices [19]. In this work, we propose an architecture called *Blue-Watchdog* to detect propagation of Bluetooth worms in public areas such as airports and shopping malls. In Blue-Watchdog, monitors are deployed in a distributed fashion to sniff Bluetooth paging packets in the air which are used to establish connections between Bluetooth devices. To detect Bluetooth worm propagation, Blue-Watchdog looks for abrupt changes of average paging rate per Bluetooth device from both temporal and temporal-spatial perspectives. The temporal scheme relies on the CUSUM (Cumulative Sum) sequential test together with the generalized likelihood ratio (GLR), and the temporal-spatial scheme aims to identify spatial regions within which abnormally frequent paging attempts have occurred.

We evaluate both schemes with two mobility models, one the random waypoint model widely used in the adhoc networking community and the other a more realistic urban pedestrian mobility model. The experimental results reveal that the temporal scheme quickly catches the fast-spreading Bluetooth worm under the random waypoint model, and the temporal-spatial scheme performs more effectively in detecting the worm propagation under the urban pedestrian mobility model. Working in a hybrid mode, Blue-Watchdog not only detects worm propagation under both mobility models with high accuracy and low delays, but also generates very low false alarm rates. These results suggest that Blue-Watchdog is a viable candidate in practice for fast detection of Bluetooth worms that try to spread themselves aggressively in public areas.

The remainder of the paper is organized as follows. We discuss related work in Section 2. Section 3 presents the Blue-Watchdog architecture for Bluetooth worm detection. In Sections 4 and 5, we discuss how to detect Bluetooth worm propagation from the temporal and temporal-spatial perspectives, respectively. We evalu-

*Los Alamos National Laboratory Publication No. LA-UR 09-00184 ate the two detection schemes in Section 6 and make concluding remarks in Section 7.

2 Related Work

Bose and Shin gave a comprehensive survey about mobile viruses that exploit Bluetooth for spreading [5]. Early work regarding Bluetooth worms focused on applying simulation techniques to study the propagation dynamics of mobile malware including Bluetooth worms [25, 20, 9]. Some mathematical models have also been proposed to characterize the spreading process of mobile worms, including a probabilistic queueing model [17] and a detailed mathematical model specifically for Bluetooth worm propagation [26]. Existing malware detection schemes for IP networks have also been borrowed to detect malware on mobile devices, including both signature-based detection schemes [13, 4] and anomaly-based detection schemes [15]. These schemes, however, are usually device-resident and thus inevitably consume battery power, which significantly limits their practical deployment. SmartSiren [7] offloads mobile malware detection to a server, but still demands support from mobile devices. An intelligent mobile malware, like many existing malware that spread on the IP network, can disable detection functionality deployed on the mobile device. Blue-Watchdog proposed in this work, however, does not require any support from mobile devices and is thus complementary to existing malware detection schemes.

Blue-Watchdog uses the CUSUM algorithm to detect Bluetooth worms from the temporal perspective. Similar approaches were applied before to detect SYN flooding attacks [23] and instant messaging worms [27]. Integrating this method with spatial notion, which is proposed in this work to detect Bluetooth worms from the temporal-spatial perspective, has never been pursued for malware detection.

3 Blue-Watchdog Architecture

We propose an architecture called *Blue-Watchdog* to detect Bluetooth worm propagation in public areas. Blue-Watchdog, depicted in Figure 1, is comprised of two basic components: *Bluetooth monitors* and *Bluetooth worm detection center*. Bluetooth monitors are used to collect the numbers of paging attempts, which are essential to Bluetooth worm propagation. By contrast, the number of inquiry packets is not a good signal for worm detection because inquiry packets, which are used to discover neighbors, may be used for normal surveillance operations. The paging process of Bluetooth is illustrated in Fig. 2. The paging device and the paged device are also called *master* and *slave*, respectively. The master first sends a *series* of ID packets (a special type of Bluetooth packets) that contain only the access code of the slave. The access code of the slave contains the low 24 bits of the slave's Bluetooth device address. If the slave is in the *paging scanning* substate and receives an ID packet with the same frequency at which it is being transmitted, the slave sends back an ID packet that also contains its access code. The master responds with an FHS (Frequency Hop Synchronization) packet and the slave then sends back another ID packet with its own access code.

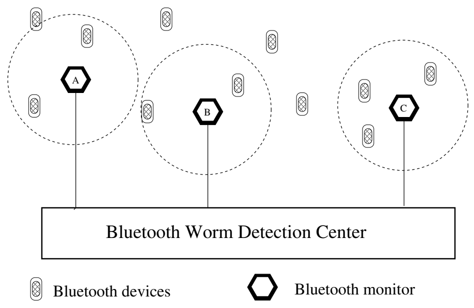

Figure 1. Blue-Watchdog architecture

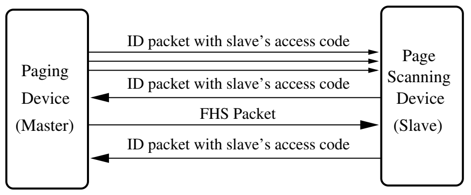
Paging Device (Master)
Page Scanning Device (Slave)
ID packet with slave's access code
ID packet with slave's access code
FHS Packet
ID packet with slave's access code

Figure 2. Paging process in Bluetooth

Blue-Watchdog discretizes time into *measurement windows* of equal length. Worm detection is performed at the end of each measurement window. A Bluetooth monitor sniffs Bluetooth ID packets in the air[1] and keeps a list of *unique* Bluetooth device addresses extracted from those ID packets that it has captured from the air during the current measurement window[2]. Note that ID packets are also used during the inquiry process when a Bluetooth device tries to discover new neighbors: an inquiry packet sent by a Bluetooth device is actually an ID packet with the Inquiry Access Code. As this Inquiry Access Code is unique, we can easily exclude these ID packets from those that are used during the paging process. In Blue-Watchdog, we count the number of *unique* Bluetooth device addresses, as opposed to the total number of ID packets, because a paging device (i.e., a master) sends out a series of ID

[1]Bluetooth RF sniffers that can capture Bluetooth traffic in the air have already been available in the market [21].

[2]To address privacy concerns, we can keep only hashed Bluetooth device addresses in Blue-Watchdog.

packets with the slave's access code until one of them is successfully received by the slave.

A high number of paging attempts with different slave addresses do not necessarily mean abnormal Bluetooth behaviors. We need to consider the total number of Bluetooth devices in the area being monitored. To obtain this information, a Bluetooth monitor in Blue-Watchdog itself uses Bluetooth to periodically discover neighbors in its vicinity. In Blue-Watchdog, we assume that the sniffing range of the monitor is roughly the same as its Bluetooth communication range. We call the sniffing area of a Bluetooth monitor its *monitoring range*. Blue-Watchdog does not require that the monitoring ranges of all Bluetooth monitors cover the whole area. Hence, some Bluetooth devices, even in a discoverable model, may not be found by any monitor.

To further explain the detection problem, we introduce a few notations here. Suppose that there are m Bluetooth monitors in the system. Let $\mathcal{M} = \{M_j\}|_{j=1}^{m}$ denote the entire set of Bluetooth monitors. Within the i-th measurement window, suppose that monitor M_j ($1 \le j \le m$) has observed $a_j(i)$ unique Bluetooth device addresses in ID packets and found $b_j(i)$ unique neighbors. Define vectors $\vec{A}(i)$ and $\vec{B}(i)$ as follows:

$$\begin{cases} \vec{A}(i) = [a_1(i), \quad a_2(i), \quad ..., \quad a_m(i)], \\ \vec{B}(i) = [b_1(i), \quad b_2(i), \quad ..., \quad b_m(i)], \end{cases} \tag{1}$$

At the end of the i-th measurement window, each monitor M_j sends both $a_j(i)$ and $b_j(i)$ to the Bluetooth worm detection center. Bluetooth monitors and Bluetooth worm detection center can be connected through wireline technology (e.g., Ethernet) or wireless connections with larger communication ranges than Bluetooth (e.g., WLANs). After collecting $\vec{A}(k)$ and $\vec{B}(k)$ up to the current measurement window, the Bluetooth worm detection center raises an alert if it finds there is a Bluetooth worm spreading in the area. We shall describe detailed detection algorithms in Sections 4 and 5.

How to respond to a worm alert is contingent on the specific environment in which Blue-Watchdog is deployed. For instance, a warning can be announced through other medias (e.g., TVs at an airport) that asks people to cautiously accept file transfer requests from unknown Bluetooth devices, put their Bluetooth devices in a non-discovery mode, or even turn off their devices if possible.

4 Temporal Detection

A Bluetooth worm that aims to spread quickly needs to aggressively scan for new victims in the neighborhood. Temporal detection techniques aim to detect such Bluetooth worms by analyzing the time series $\vec{A}(i)$ and $\vec{B}(i)$ that have been collected so far. Blue-Watchdog applies a sequential change-point detection method whose key idea is to find the point of change, if it occurs, within a time series by checking whether it is a homogeneous process. This approach combines the CUSUM algorithm [6] with the generalized likelihood ratio (GLR). Consider $X = \{X_i\}_{i=1}^{\infty}$, where

$$X_i = \frac{\sum_{j=1}^{m} a_j(i)}{\sum_{j=1}^{m} b_j(i)}. \tag{2}$$

X_i reflects the average paging rate per Bluetooth device within the i-th measurement window. Let $p_\theta(x)$ denote the probability density of X, and assume that parameter θ is equal to θ_0 before change point t_0, and becomes θ_1 after t_0 ($\theta_1 \neq \theta_0$). The CUSUM-GLR algorithm repeatedly tests the following hypothesis:

$$\mathbf{H_0} \quad : \quad \theta = \theta_0 \quad \text{(null hypothesis)} \tag{3}$$
$$\mathbf{H_1} \quad : \quad \theta = \theta_1, \tag{4}$$

with the aid of the *sequential probability ratio test* (SPRT). Let x_i be the measured value of X_i. Define g_i by:

$$g_i = \begin{cases} 0 & \text{if } i = 0 \text{ or } g_{i-1} + \ln \frac{p_{\theta_1}(x_i)}{p_{\theta_0}(x_i)} \le 0 \\ g_{i-1} + \ln \frac{p_{\theta_1}(x_i)}{p_{\theta_0}(x_i)} & \text{if } i > 0 \text{ and } g_{i-1} + \ln \frac{p_{\theta_1}(X_i)}{p_{\theta_0}(X_i)} > 0 \end{cases}$$

The stopping rule and the alarm time are defined by:

$$t_a = \min\{i : g_i \ge h\}, \tag{5}$$

where h is a predefined threshold. That is, if $g_i < h$, hypothesis H_0 holds; otherwise, hypothesis H_1 holds.

To apply the CUSUM-GLR algorithm on Bluetooth worm detection, we need to have *a priori* knowledge about both θ_0 and θ_1. Recall that X_i reflects the average paging rate per Bluetooth device. Due to lack of real measurement data providing the distribution of X_i, we simply assume that it conforms to a normal distribution here based on the central limit theorem[3]. That is, $X_i \sim \mathcal{N}(\mu, \sigma^2)$, where $\mathcal{N}(\mu, \sigma^2)$ is the normal distribution with the mean μ and the standard deviation σ. Under normal circumstances without Bluetooth worms we estimate mean μ_0 and standard deviation σ_0 based on history measurement data. If there is a Bluetooth worm propagating in the network, suppose that the mean number of paging attempts within a measurement window increases up to μ_1, where $\mu_1 > \mu_0$, with standard deviation σ_1. Then g_i can be written as:

$$g_i = [g_{i-1} + \ln \frac{\sigma_1}{\sigma_0} + \frac{(x_i - \mu_0)^2}{2\sigma_0^2} - \frac{(x_i - \mu_1)^2}{2\sigma_1^2}]^+, \tag{6}$$

where $[s]^+$ denotes $\sup\{0, s\}$. When g_i reaches h, an alarm is raised that a Bluetooth worm is spreading.

[3]Our experimental results shall later show that this assumption does not need to be followed strictly in practice.

A nice feature of the CUSUM-GLR algorithm is that it does not need to remember all the measurement data collected in the past. Instead, it only needs to keep the previous value (i.e., g_{i-1}) when computing g_i in the current measurement window. When we implement the CUSUM-GLR algorithm for Bluetooth worm detection, μ_0 and σ_0 can be estimated from real-world data. It is however difficult to estimate μ_1 and σ_1 for two reasons. First, propagation speeds of Bluetooth worms rely on both device densities and their spreading strategies. Hence, the number of connection attempts generated by each Bluetooth device during a Bluetooth worm outbreak can vary significantly. Second, during the propagation of a Bluetooth worm, the average number of connection attempts generated by each Bluetooth device increases as more Bluetooth devices are infected. Hence, the random sequence $\{X_i\}$ after the change point cannot be deemed to have identical distributions any more.

In a practical setting, however, the CUSUM-GLR algorithm is still applicable for Bluetooth worm detection. As new Bluetooth devices are infected by the worm, the mean of X_i after the change point should increase monotonically, which makes g_i grow with i as well during worm propagation. As we are only interested in the change of the mean for worm detection, we assume that $\sigma_1 = \sigma_0 = \sigma$. We also choose μ_1 to be reasonably larger than μ_0. Then, Equation (6) can be rewritten as follows:

$$g_i = [g_{i-1} + \frac{(x_i - \mu_0)^2}{2\sigma^2} - \frac{(x_i - \mu_1)^2}{2\sigma^2}]^+, \tag{7}$$

Parameter setting. We now discuss how to choose parameter h. Let τ be the observation time before an alarm, either false or positive, is raised. With approximations of Wald's sequential test, the following relationship can be established [6]:

$$\mathbb{E}[\tau \mid \theta_0] \quad = \quad |e^h - h - 1|/|\phi_0|, \tag{8}$$

where $\mathbb{E}[\tau \mid H_0]$ is the average observation time before a false alarm is raised, $\mathbb{E}[\tau \mid H_1]$ is the average delay time of detection, and

$$\phi_0 \quad = \quad \mathbb{E}\left[\log \frac{p_{\theta_1}(x)}{p_{\theta_0}(x)} \mid \theta_0\right] < 0. \tag{9}$$

As $X_i \mid \theta_0 \sim \mathcal{N}(\mu_0, \sigma^2)$ and $X_i \mid \theta_1 \sim \mathcal{N}(\mu_1, \sigma^2)$, we have the following[4]:

$$\begin{aligned}\phi_0 &= \int_{-\infty}^{+\infty} [\tfrac{(x-\mu_0)^2}{2\sigma_0^2} - \tfrac{(x-\mu_1)^2}{2\sigma_1^2}] \cdot \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(x-\mu_0)^2}{2\sigma^2}} \mathrm{d}x \\ &= \tfrac{(\mu_0-\mu_1)^2}{2\sigma^2} + \tfrac{\mu_0(\mu_1-\mu_0)}{\sigma^2} = -\frac{(\mu_1-\mu_0)^2}{2\sigma^2}\end{aligned} \tag{10}$$

[4]In practice, x should be non-negative, but it is unlikely to be very large as well. We thus expect their effects cancel each other when the integration is performed.

Let κ be the desired observation time before a false alarm is raised. With (10) and (8), we have:

$$|e^h - h - 1| = \frac{\kappa(\mu_1 - \mu_0)^2}{2\sigma^2}. \tag{11}$$

As obtaining a precise solution to Eq. (11) is difficult, we use the Taylor series to approximate e^h by $1 + h + h^2/2$. Hence, parameter h is set as follows:

$$h \approx \frac{\sqrt{\kappa}(\mu_1 - \mu_0)}{\sigma}. \tag{12}$$

5 Temporal-Spatial Detection

In practice, as Bluetooth devices are often carried by human beings whose mobility is typically constrained within some small regions (e.g., office, restaurant, etc.), the spatial range of the worm propagation is limited as well. Hence, if Bluetooth monitors are deployed in a large area but a Bluetooth worm spreads into only a few small regions at its early stage, the worm may not increase the overall average paging rate significantly so that it cannot be detected by the temporal scheme.

To solve this problem, we combine the temporal algorithm with some spatial notion. In this scheme, Blue-Watchdog identifies spatial clusters with abnormally high average paging rate per Bluetooth device. For ease of explanation, we introduce a few notations here. Let Γ denote the set of all possible zones inside the entire monitored area. How to define a reasonable zone shall be discussed later. Without introducing any confusion, we abuse notation $Z \in \Gamma$ by letting it be the entire set of Bluetooth monitors within zone Z. For the j-th Bluetooth monitor, its observed average paging rate within the i-th measurement window is

$$Y_i^{(j)} = \frac{a_j(i)}{\max\{1, b_j(i)\}}, \tag{13}$$

where $a_j(i)$ and $b_j(i)$ have been introduced in Eq. (1). Note that if no neighbors were found[5], we let $Y_i^{(j)}$ simply be the number of page attempts observed. In the following discussion, we drop index i if no confusion occurs. We assume that $Y^{(j)}$, for all $j : 1 \leq j \leq m$, are independent random variables. If a Bluetooth monitor M_k is located within a region that does not have infected Bluetooth devices, we assume that $Y^{(j)} \sim \mathcal{N}(\mu_0(j), \sigma(j)^2)$, and $Y^{(j)} \sim \mathcal{N}(\mu_1(j), \sigma(j)^2)$ otherwise. We use the same variance here due to the difficulty of measuring it under worm propagation and our interest in only shift of mean for worm detection.

Worm detection within a zone. We first discuss how to decide whether a worm spreads within a given zone Z. Define $S(Z)$ as follows:

$$S(Z) = \sum_{\forall M_j \in Z : 1 \leq j \leq m} Y^{(j)}. \tag{14}$$

[5]This can occur due to radio interference, mobility, or Bluetooth protocol dynamics.

As each $Y^{(j)}$ has a normal distribution, we have:

$$\begin{aligned} S(Z) \mid \text{no worm} &\sim \mathcal{N}\Big(\sum_{\forall j: M_j \in Z} \mu_0(j), \sum_{\forall M_j: j \in Z} \sigma(j)^2\Big) \\ S(Z) \mid \text{worm} &\sim \mathcal{N}\Big(\sum_{\forall j: M_j \in Z} \mu_1(j), \sum_{\forall j: M_j \in Z} \sigma(j)^2\Big) \end{aligned}$$

Similar to Eq. (6), define $g'_i(Z)$ for zone Z as follows:

$$\begin{aligned} g'_i(Z) &= [g_{i-1}(Z) + \frac{(s_i(Z) - \sum_{\forall j: M_j \in Z} \mu_0(j))^2}{2 \sum_{\forall M_j: j \in Z} \sigma(j)^2} \\ &\quad - \frac{(s_i(Z) - \sum_{\forall j: M_j \in Z} \mu_1(j))^2}{2 \sum_{\forall M_j: j \in Z} \sigma(j)^2}]^+, \quad (15) \end{aligned}$$

where $s_i(Z)$ is the observed value of $S(Z)$ within the i-th measurement window. Similarly, $h(Z)$ is given by:

$$h(Z) = \frac{\sqrt{\kappa}(\sum_{\forall j: M_j \in Z} \mu_1(j) - \sum_{\forall j: M_j \in Z} \mu_0(j))}{(\sum_{\forall M_j: j \in Z} \sigma(j)^2)^{1/2}},$$

where κ is the expected observation time of a false alarm.

We say a *violation* occurs when sum g exceeds a predefined threshold. For each zone Z, let $v(Z)$ be the number of violation times. With these definitions, the worm detection algorithm for zone Z is given in Algorithm 1. In it, an alert is raised only after there are at least η consecutive violations; otherwise, we keep the previous cumulative sum. The reason for this is that the average paging rate collected from each individual monitor (i.e., $Y^{(j)}$) can bear high variation. Under normal conditions, hence, the average paging rate observed by one monitor or a group of them can easily lead to a violation within a single measurement window, but it is less likely to have consecutive violations. When there is a worm outbreak, however, it is very likely to have consecutive violations because infected devices continuously attempt to infect new victims.

Algorithm 1 Worm detection of zone Z

```
Require: $g_{i-1}(Z)$ and violation times $v(Z)$.
 1: Compute $s_i(Z)$, and then $g'_i(Z)$ based on Eq. (15)
 2: if $g'_i(Z) > h(Z)$ then
 3:     $v(Z) \leftarrow v(Z) + 1$
 4:     if $v(Z) \geq \eta$ then
 5:         Alert that a worm spreads in zone $Z$
 6:         $g_i(Z) \leftarrow 0$, $v(Z) \leftarrow 0$
 7:     else
 8:         $g_i(Z) \leftarrow g_{i-1}(Z)$
 9:     end if
10: else
11:     $g_i(Z) \leftarrow g'_i(Z)$, $v(Z) \leftarrow 0$
12: end if
```

Discussion on selection of Γ. With m Bluetooth monitors, there are in total $2^m - 1$ choices for a zone Z (suppose that a zone must have one monitor in it). The exponential feature renders it impractical for actual deployment. As a Bluetooth worm spreads itself in a proximity-based fashion, we limit our choices for zone Z to only spatially continuous regions. Here, we consider a sphere-based approach to define zones: a zone is represented as the area inside a sphere. It is known that the Vapnik-Chervonenkis (VC) dimension for the set of all spheres in $\mathbb{R}^d$ is $d+1$ [12]. Let $\mathcal{S}(m)$ denote the maximum number of zones with different monitors in them. Based on the Sauer's Lemma [3], we have $\mathcal{S}(m) \leq \sum_{i=0}^{d+1} \binom{m}{i} \leq (m+1)^{d+1} \leq (\frac{me}{d+1})^{d+1}$. Hence, with m monitors, $|\Gamma| \leq (0.9m)^3$ for 2-dimension, and $|\Gamma| \leq (0.68m)^4$ for 3-dimension. It is easy to see that the sphere-based approach has a scalability issue: with 1000 monitors deployed in a 3-D space, the Sauer's Lemma can only guarantee that at most 213 billion zones appear in Γ, which is not promising in practice.

Temporal-spatial detection algorithm. With the scalability challenges shown above, we propose a light-weight algorithm. For each monitor, we keep the r closest monitors from it. The matrix $\mathcal{G}(j,k)$, where $1 \leq j \leq m$ and $1 \leq k \leq r$, stores the cumulative sum for zone $Z(j,k)$ that contains the k closest monitors from monitor j (including itself). The temporal-spatial algorithm is then summarized as follows:

Algorithm 2 Temporal-spatial worm detection

```
1: for $j = 1$ to $m$ do
2:     for $k = 1$ to $r$ do
3:         Detect worm for zone $Z(j,k)$ with Algo. 1.
           Announce a warning if Algo. 1 raises an alert.
4:     end for
5: end for
```

Algorithm 2 simplifies the set Γ by considering only spheres that are centered on each of the m monitors and contain at most r monitors. In our implementation, we precompute the closest r monitors for each monitor. As Algorithm 1 can be done within $O(1)$ time[6], the time complexity of the online temporal-spatial detection algorithm is $O(mr)$, irrespective of the dimension of the space in which the monitors are deployed. Moreover, the memory required by this algorithm is only $O(mr)$. In practice, the selection of r reflects the tradeoff between detection accuracy and computational cost: a higher r demands more computational resources but allows more zones to be considered.

[6]In Algorithm 1, $s_i(Z)$ can be updated incrementally with $O(1)$ time.

6 Experimental Evaluation

In this section, we evaluate the performance of BlueWatchdog using the ns-2 network simulator [1], extended with the UCBT module which simulates the Bluetooth protocol in great details [2].

Behavioral model of Bluetooth worms. The behavioral model of Bluetooth worms in our experiments is depicted in Figure 3. When a Bluetooth worm is activated, it starts looking for Bluetooth neighbors. Within a certain amount of time T_{inq} (10.24 seconds by default), if it collects three responses from its neighbors, it stops the inquiry process; otherwise it stops the inquiry process after T_{inq} time units. The worm then attempts the following on each discovered neighbor: *establish a connection*, *request a file transfer*, *transmit the worm code*, and *disconnect from the victim device.*

Establishing a connection to a victim device involves the paging process in Bluetooth. In order to detect paging failures, a Bluetooth device schedules a paging (or connection establishing) timer that expires after 5.12 seconds, which is the default value in the Bluetooth specification and is used in our experiments.

If the infected device fails to establish a connection to a victim device due to, say, device mobility, it removes it from the list and attempts to infect the next one; otherwise, it sends a request to the victim device for a file transfer. When the victim device receives the request packet, it "thinks" for T_{think} time units before sending back a reply packet, either accepting or rejecting the request. In our experiments, the acceptance probability is 0.5 and the thinking time T_{think} is exponentially distributed with mean 5.0 seconds.

If the request is accepted, the worm transmits the worm code onto the victim device. The size of the worm code is 20,000 bytes in all our experiments. To detect failures of code transfer, a timer is scheduled to expire after 10 seconds; if the worm code cannot be successfully transferred when this timer expires, the transfer is canceled. If the transfer succeeds, the victim device gets infected and the channel is disconnected.

After attempting to infect all the neighbors discovered during the inquiry phase, the worm keeps inactive for 30 seconds before another infection cycle starts.

Our Bluetooth worm model closely mirrors the behavior of real-world Bluetooth worms such as CommWarrior.a!sys worm. In this paper, we do not explore how the worm model parameters affect its propagation and we refer interested readers to [25] for further insights. Instead, we focus on how the two Bluetooth worm detection schemes perform under different mobility patterns, which have significant impact on Blueworth worm propagation [24]. In the following, we consider two types of mobility models: *random waypoint mobility model* and more realistic *urban pedestrian mobility model.*

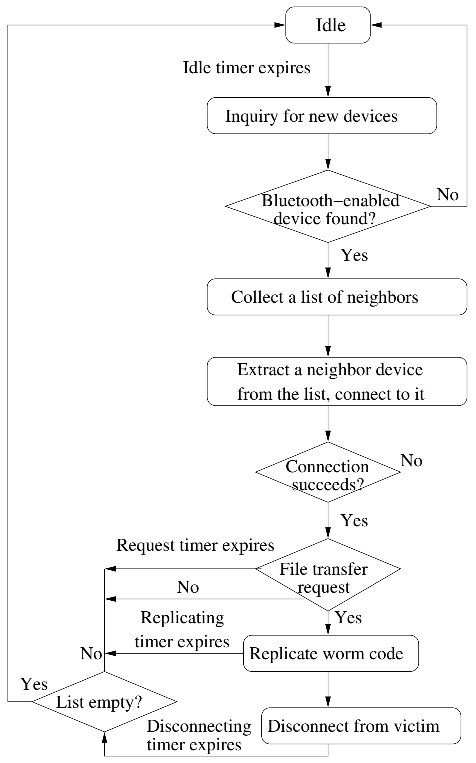

Figure 3. Behavioral model of Bluetooth worms in our experiments

6.1 Random Waypoint Mobility Model

The random waypoint mobility model has been widely used in the ad-hoc networking community, largely because of its simplicity. When this model is used in our experiments, a mobile device randomly chooses a destination within a $250 \times 250\text{m}^2$ square and moves straightly to that place at a speed uniformly drawn between 0.5 and 2.5 meters per second. Hence, the average speed of a moving mobile device is 1.5 meters per second, which mirrors the average pedestrian speed. Once a mobile device reaches its destination, it pauses for a period uniformly between 0 and 60 seconds.

For each mobile device in the network, its normal Bluetooth application behavior is modeled as follows[7]. Similar to the Bluetooth worm, this application, once activated, tries to discover at most three neighbors in its vicinity, and thereafter randomly chooses one for a file transfer. The file size is drawn from a Pareto distribution with scale (cutoff) parameter 20,000 bytes and shape parameter 1.7. Once the transfer is finished, the application sleeps for T_{sleep} time units, which is

[7] Our normal Bluetooth traffic model is motivated by the characteristics observed in Internet traffic: file sizes have long-range dependence and user sessions are Poisson [18].

exponentially distributed with mean 600 seconds.

Assuming that the coordinate of the left bottom corner of the square is (0,0), we place 100 Bluetooth worm monitors at coordinates $(250i/11, 250j/11)$, where $1 \leq i, j \leq 10$. A Bluetooth monitor tries to discover as many neighbors as possible during its inquiry phase, which lasts at most 10.24 seconds (default value in the Bluetooth protocol specification); after one inquiry phase, it sleeps for 10 seconds before another inquiry phase starts. Every five minutes, each Bluetooth monitor sends its collected information to the Bluetooth worm detection center, as shown in Figure 1. The window size is so chosen that the accumulative sum is sufficiently large for detection but the detection delay is in the order of tens of minutes.

We simulate 1,000 mobile devices for 5 hours, and the initial infected device is activated after 3 hours. We assume that all mobile devices are vulnerable because we want to have a sufficient number of vulnerable devices to show infection curves, given the scalability challenge of ns-2 simulator which takes more than one week to simulate each scenario with the machines we have. In our experiments, we simulate 20 sample runs with different random number seeds. Figure 4 shows the infection curve from each sample run[8]. From the graph, we note that the propagation speed of the Bluetooth worm varies significantly in different sample runs. Also, the simulation results reveal that neither X_i nor $Y_i^{(j)}$ strictly follows the normal distribution. Even so, our experimental results will show that both detection schemes still effectively detect the worm propagation.

Detection performance. Before the initial infected device starts spreading the worm, each Bluetooth monitor has already sent its collected information for 35 times (recall that each Bluetooth monitor collects data every 5 minutes). Among these states, we use the first 20 of them for training purpose, and the remaining 15 to test the false alarm rates.

(1) *Temporal detection.* In this approach, we calculate the cumulative sum g_i as in Eq. (7) at the end of each measurement window. Parameters μ_0 and σ can be directly derived from the training data. We let μ_1 be $\alpha\mu_0$. Recall that there is another input parameter κ in Eq. (12) for this detection scheme. We vary κ among 1, 2, 3 and 4, and α among 2, 3, 4 and 5. The results show that for every combination of κ and α, no false alarms have been raised and the detection ratio is 100% among the 20 sample runs.

Another measure of detection performance is *detection delay*, which is the number of measurement windows required to detect the worm. The detection delays are depicted in Figure 5(1) (with only $\kappa = 1$ and 4). We note that the impact of parameter κ is very small. This is because threshold h is linear with the square root of κ, as seen in Eq. (12). On the other hand, the detection delay increases more prominently with the growth of parameter α, because threshold h increases linearly with α. The fraction of infected devices when the worm is detected is illustrated in Figure 5(2). Similar patterns can be observed with respect to the effects of κ and α. Apparently, a small α is desirable if we want to detect the Bluetooth worm at its early stage. For instance, if α is 2, only 10% of the devices have been infected when the worm is detected, regardless of κ.

(2) *Temporal-spatial detection.* We estimate the mean and standard deviation of $Y^{(j)}$ for each Bluetooth monitor from the training dataset. Moreover, we let $\mu_1(Y^{(j)})$ be $\min\{\beta\mu_0(Y^{(j)}), C_{pr}^{min}\}$. We bound $\mu_1(Y^{(j)})$ from the lower side by C_{pr}^{min} because $Y^{(j)}$, different from X_i in Eq. (2), may bear high variation. Consider, for instance, a monitor in whose monitoring range Bluetooth devices normally seldom page their neighbors; hence, its observed average paging rate is close to 0. When a device pages a neighbor, the measured $Y^{(j)}$, which is still high relative to the low paging rate observed in the past, may trigger a false alarm if $\mu_1(Y^{(j)})$ is not bounded from the lower side. In our experiments, we let C_{pr}^{min} be one paging attempt per discovered neighbor within each measurement window.

Recall that in Algorithm 1, parameter η specifies the maximum number of violation times before an alert is triggered. Apparently, a higher η helps reduce false alarm rates, but it also elongates the detection delay if there is a Bluetooth worm outbreak. We let η be 2 in all our experiments. In this set of experiments, we vary κ among 1, 2, 3, and 4, and α among 2, 3, 4, and 5. As we only have 100 monitors in our experiments, we simply let parameter r in Algorithm 2 be 100.

Like the temporal detection scheme, the temporal-spatial detection scheme catches the propagation of Bluetooth worms among all 20 sample runs without invoking any false alarms. The detection delay and the fraction of infected devices when the worm is detected are also shown in Figure 5. We observe that the effects of parameters κ and α are similar to those for the temporal detection scheme. By contrast, the detection delay of the temporal-spatial detection scheme is longer than that under the temporal detection scheme. This is expected because, to reduce false alarm rates, the temporal-spatial scheme waits for an extra violation before an alert is triggered and it also imposes a lower bound on $\mu_1(Y^{(j)})$. If we let C_{pr}^{min} be 0 and η be 1,

[8]Due to some logistic issues, a couple of sample runs did not finish until the predefined simulation time and their infection curves thus stop at around simulation time 13,800 in Figure 4.

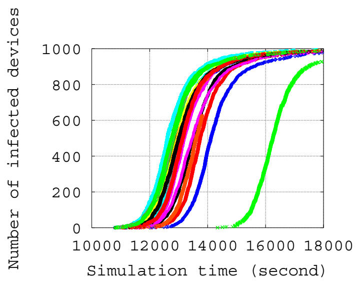

Figure 4. Infection curves for 20 sample runs

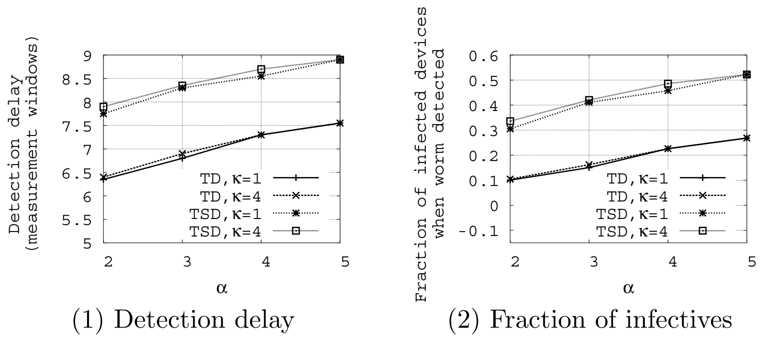

Figure 5. Detection performance (random waypoint model, TD: temporal detection, TSD: temporal-spatial detection)

the detection delays of the temporal-spatial detection scheme are comparable to those of the temporal detection scheme, although some false alarms are raised.

6.2 Urban Pedestrian Mobility Model

In another set of experiments, we simulate mobile devices carried by pedestrians moving in the Chicago downtown area with the *UDel model* [22], which is a 3-level parameterized model: a top-level model that characterizes high-level human activities such as working, shopping and at home based on surveys from US Department of Bureau of Labor, a middle-level task model that determines specific tasks associated with each activity, and a low-level agent model that decides how a person moves between different locations.

We simulate around 1,500 pedestrians, each of whom carries a Bluetooth device, in the Chicago downtown area from 9:00am to 12:00pm. Some of these people may come to this area from the outside or leave this area during the three hour period. The walking speed is set between 1.57 and 4.0 miles per hour. We generate one mobility trace to decide where to put Bluetooth monitors. This is done as follows. We first divide the whole 3-dimension space into $1\times1\text{m}^3$ cubes, each of which has a counter initialized to 0. From the training mobility trace, we take a snapshot of each pedestrian's location every 10 seconds, and for each pedestrian in a snapshot, we increase by one the counter of each cube whose center is within 10 meter range of the pedestrian. After updating the counters over all snapshots, we use the *kmlocal* k-means clustering software [16] to cluster all the cubes into 100 points in 3-dimension space. We then place a Bluetooth monitor, which collects data every 10 minutes, at each of these 100 points. In our experiments, we use two different procedures provided by the software to get two sets of monitor locations.

In this set of experiments, each Bluetooth device generates the same type of normal traffic as in those with the random waypoint mobility model. Also, we use the same behavioral model of Bluetooth worms. We use 10 mobility traces to simulate scenarios without worms, and 5 others with worms. For each mobility scenario with worms, we simulate 5 cases with different initial infected devices. Considering that there are two different sets of Bluetooth monitor locations, we simulate 20 cases without worms and 50 cases with worms.

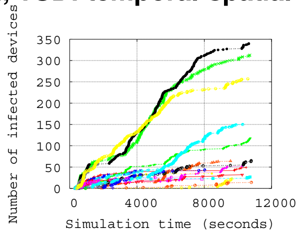

Figure 6. Infection curves of sample runs under UDel mobility model

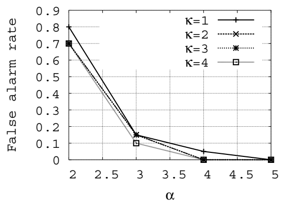

Figure 7. False alarm rates (UDel mobility model, temporal-spatial)

Figure 6 depicts the infection curves of some sample runs under the UDel mobility model. It is noted that the worm propagation speed varies significantly. Let H_s denote the set of cases with worms in which at least s devices have been infected when the simulation finishes. Among all 50 cases with worm propagation, we have $|H_{10}| = 39$ and $|H_{50}| = 19$. Clearly, the Bluetooth worm spreads much more slowly under the UDel mobility model than it under the random waypoint model, because devices in it are mixed to a much lesser degree.

Detection performance. To evaluate false alarm rates, for every case without worms, we use the remaining 19 ones for training and then use it for testing; to evaluate detection rates, we use each of the 50 cases with worms for testing and the 20 cases without worms for training.

(1) *Temporal detection.* We set the parameters in the same way as previously in the experiments under the random waypoint mobility model. We vary κ among 1, 2, 3, and 4, and α among 2, 3, 4, and 5. The temporal detection scheme does not raise any false alarms, regardless of κ and α. Figures 8(1) and 8(2) de-

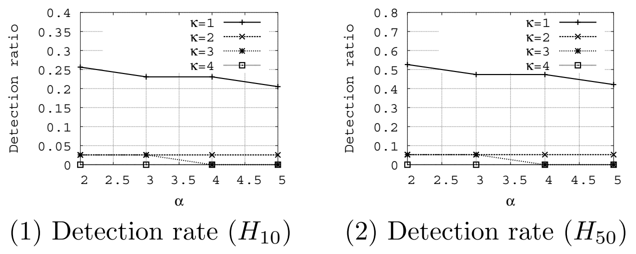
Figure 8. Detection performance (UDel mobility model, temporal detection)

pict the fraction of cases in which the temporal scheme successfully detects the worms among those that have at least 10 and 50 devices infected when the simulation finishes, respectively. Obviously, the temporal detection scheme performs poorly on worm detection: even with the best combination of parameters κ and α, the scheme can detect only about 25% of the cases in H_{10} and 50% of the cases in H_{50}. Such low detection rates attribute to the slow worm propagation in most cases under the urban pedestrian mobility model, which thus does not lead to abrupt increase of the average paging rate over the entire area.

(2) *Temporal-spatial detection.* Similar to the experiments for the random waypoint mobility model, we set $\mu_1(Y^{(j)})$ be $\min\{\beta\mu_0(Y^{(j)}), C_{pr}^{min}\}$, where C_{pr}^{min} is one paging attempt per measurement window. Also, we let η be 2 and r be 100 in Algorithms 1 and 2.

The false alarm rates under different combinations of κ and α values are shown in Figure 7. We note that the impact of κ on the false alarm rate is small, but α affects the false alarm rate significantly. This still results from the fact that threshold h in Eq. (12) is linear with α and the square root of κ. Moreover, when α is small (i.e., 2), the false alarm rate can be as high as 70%, which makes it impractical for real-world deployment. This suggests that if we want to reduce the number of false alerts triggered, we should use a relatively large α.

On the other hand, increasing parameter α leads to lower detection rates. This is confirmed in Figures 9(1) and 9(2), which show the detection rates among the cases in H_{10} and H_{50}, respectively. We note that the detection rate decreases with the growth of parameter α. Hence, to strike a balance between false alarm rates and detection rates, parameter α should be neither too large nor too small. A good operating point, for instance, can be: $\alpha = 4$ and $\kappa = 2$. Under this setting, the false alarm rate is 0, and the detection ratios for the cases in H_{10} and H_{50} are 67% and 90%, respectively. This suggests that if there are a fair number of devices infected, the worm can be detected by the temporal-spatial detection scheme with a very high probability.

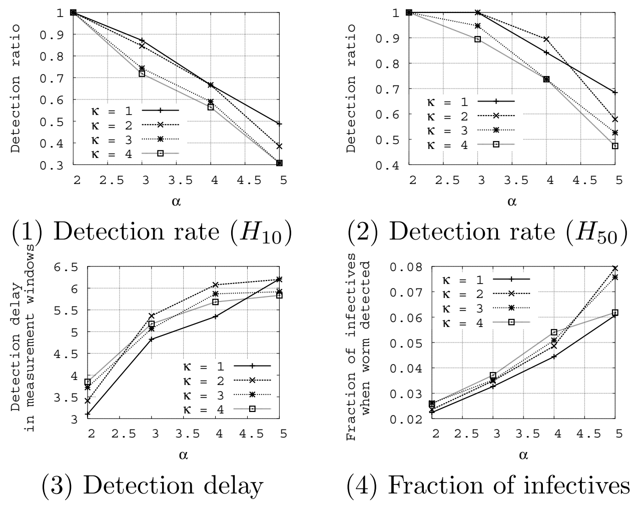
Figure 9. Detection performance (UDel mobility model, temporal-spatial detection)

Figures 9(3) and 9(4) further depict the detection delay in measurement windows and the fraction of infected devices when the worm is detected, respectively. Generally speaking, the temporal-spatial detection scheme is quite effective because when the worm is detected, less than 8% of the devices are infected, regardless of parameters κ and α. Under the aforementioned operating point (i.e., $\alpha = 4$ and $\kappa = 2$), the average detection delay is 6 measurement windows, which is about one hour, and when the worm is detected, slightly fewer than 5% of the devices have been infected. Such a low infection ratio when the worm is detected makes the temporal-spatial scheme a viable solution in practice.

6.3 Hybrid detection

It is noted that when we use the random waypoint mobility model under which the worm spreads very fast, both the temporal and temporal-spatial detection schemes lead to 100% detection rate without false alarms, but the former outperforms the latter due to its small detection delay; in contrast, when the urban pedestrian mobility model is used, the temporal-spatial detection scheme achieves much higher detection rates. In a hybrid mode, Blue-Watchdog uses both methods simultaneously: If *either* of them flags an alert, a worm propagation warning is raised. With this hybrid approach, Blue-Watchdog achieves high detection rates without false alarms under both mobility patterns.

Although we do not have rigorous results regarding the computational cost of Blue-Watchdog, we observe from our experiments that the overhead is insignificant: the execution time of each scheme is ignorable relative to the length of each measurement window, and the memory footprints under both schemes are very low on the commodity PCs we used for the experiments.

7 Conclusions and Future Work

Malware that spread on mobile devices have been on the rise recently. In this work, we propose an architecture called Blue-Watchdog to detect mobile malware that exploit Bluetooth for propagation. Blue-Watchdog detects Bluetooth worm propagation from both temporal and temporal-spatial perspectives. Experimental results with both random waypoint mobility model and a more realistic urban pedestrian mobility model show that Blue-Watchdog, when operating under a hybrid mode, can effectively detect Bluetooth worm propagation without false alarms.

In Blue-Watchdog, we applied the CUSUM-GLR algorithm to detect Bluetooth worm propagation from both temporal and temporal-spatial perspectives. To derive its parameters, we make assumptions such as normal distributions for both X_i in Eq. (2) and $Y_i^{(j)}$ in Eq. (13) to simplify analysis. Due to lack of real-world Bluetooth traces, we could not validate the feasibility of such assumptions rigorously. If these assumptions cannot be verified by future measurement results, we can instead use the non-parametric, but usually less efficient, version of the CUSUM algorithm, which does not require a priori knowledge of the distributions [27]. Moreover, due to the high computational cost of Bluetooth worm simulation, in this study we cannot evaluate the effectiveness of Blue-Watchdog comprehensively under diverse parameter settings. We plan to explore this further in our future work.

Admittedly, Blue-Watchdog cannot detect all Bluetooth worms, especially those that intelligently constrain their paging attempts to evade detection. Such shortcomings are generally shared by detection schemes based on statistical anomaly detection. In the future, we plan to develop other complementary techniques, such as Bluetooth packet inspection and behavioral analysis, to further improve detection accuracy.

References

[1] `http://www.isi.edu/nsnam/ns/index.html`.

[2] `http://www.ececs.uc.edu/~cdmc/ucbt/ucbt.html`.

[3] M. H. G. Anthony and N. Biggs. *Computational Learning Theory*. Cambridge University Press, 1997.

[4] A. Bose, X. Hu, K. Shin, and T. Park. Behavioral detection of malware on mobile handsets. In *Proceedings of MobiSys*, June 2008.

[5] A. Bose and K. G. Shin. On mobile viruses exploiting messaging and bluetooth services. In *Proceedings of SecureComm*, 2006.

[6] B. E. Brodsky and B. S. Darkhovsky. *Nonparametric Methods in Change Point Problems*. Kluwer Academic Publishers, 1993.

[7] J. Cheng, S. Wong, H. Yang, and S. Lu. Smartsiren: Virus detection and alert for smartphones. In *Proceedings of ACM MobiSys*, 2007.

[8] `http://www.f-secure.com/v-descs/commwarrior.shtml`.

[9] C. Fleizach, M. Liljenstam, P. Johansson, G. M. Voelker, and A. Mehes. Can you infect me now?: malware propagation in mobile phone networks. In *Proceedings of ACM WORM'07*, 2007.

[10] http://www.theregister.co.uk/2006/03/30/flexispy.

[11] http://www.f-secure.com/v-descs/cabir.shtml.

[12] P. Gupta and P. R. Kumar. Internet in the sky: Capacity of 3-d wireless networks. In *Proceedings of CDC'02*.

[13] G. Hu and D. Venugopal. A malware signature extraction and detection method applied to mobile networks. In *Proceedings of IPCCC*, April 2007.

[14] http://www.f-secure.com/v-descs/inqtana_a.shtml.

[15] H. Kim, J. .Smith, and K. Shin. Detecting energy-greedy anomalies and mobile malware variants. In *Proceedings of MobiSys*, June 2008.

[16] `http://www.cs.umd.edu/~mount/Projects/KMeans/`.

[17] J. W. Mickens and B. D. Noble. Modeling epidemic spreading in mobile environments. In *Proceedings of ACM WiSe'05*, 2005.

[18] V. Paxson and S. Floyd. Wide-area traffic: the failure of poisson modeling. *SIGCOMM Comput. Commun. Rev.*, 24(4):257–268, 1994.

[19] R. Racic, D. Ma, and H. Chen. Exploiting mms vulnerabilities to stealthily exhaust mobile phone's battery. In *Proceedings of SecureComm*, August 2006.

[20] J. Su, K. K. Chan, A. G. Miklas, K. Po, A. Akhavan, S. Saroiu, E. D. Lara, and A. Goel. A preliminary investigation of worm infections in a bluetooth environment. In *Proceedings of ACM WORM'06*, 2006.

[21] `http://www.commsdesign.com/design_corner/OEG20020313S0049`, 2002.

[22] `http://udelmodels.eecis.udel.edu`.

[23] H. Wang, D. Zhang, and K. G. Shin. Detecting SYN flooding attacks. In *Proceedings of INFOCOM'02*.

[24] G. Yan, L. Cuellar, S. Eidenbenz, H. D. Flores, N. Hengartner, and V. Vu. Bluetooth worm propagation: mobility pattern matters! In *Proceedings of ASIACCS'07*.

[25] G. Yan and S. Eidenbenz. Bluetooth worms: Models, dynamics, and defense implications. In *Proceedings of ACSAC'06*.

[26] G. Yan and S. Eidenbenz. Modeling propagation dynamics of bluetooth worms. In *Proceedings of ICDCS'07*, 2007.

[27] G. Yan, Z. Xiao, and S. Eidenbenz. Catching instant messaging worms with change-point detection techniques. In *Proceedings of LEET'08*.