Using Performance Signatures and Software Rejuvenation for Worm Mitigation in Tactical MANETs

Alberto Avritzer
Siemens Corporate Research
755 College Road East
Princeton, NJ, USA 08540
alberto.avritzer@siemens.com

Robert G. Cole
JHU/Applied Physics Laboratory
12000 Johns Hopkins Road
Laurel, MD, USA 20723
robert.cole@jhuapl.edu

Elaine J. Weyuker
AT&T Labs - Research
180 Park Avenue
Florham Park, NJ, USA 07932
weyuker@research.att.com

ABSTRACT

In this paper, we propose a new approach for mitigation of worm propagation through tactical Mobile Ad-Hoc Networks (MANETs) which is based upon performance signatures and software rejuvenation. Three application performance signature and software rejuvenation algorithms are proposed and analyzed. These algorithms monitor critical applications' responsiveness and trigger actions for software rejuvenation when host resources degrade due to a coresident worm competing for host resources. We analyze the effectiveness of our algorithms through analytic modeling and detailed, extensive simulation studies. The key performance metrics investigated are application response time, mean time between rejuvenations and the steady state probability of host infection. We also use simulation models to investigate several design and parameter tuning issues. We investigate the relationship between the rate at which the application performance monitors can detect out-of-specification applications and the rate of worm propagation in the network.

Categories and Subject Descriptors

C.2.3 [**Computer Communications Networks**]: Network Operations—*network monitoring*; D.2.0 [**Software Engineering**]: General—*protection mechanisms*

General Terms

Security, Performance, Measurement

Keywords

Computer worms, Mobile Ad Hoc Networks (MANETS), mitigation, software monitoring

1. INTRODUCTION

The U.S. Army is developing tactical Mobile Ad Hoc Network (MANET) technologies to support its communications needs in military situations. It is feared that these might be a ready target for people interested in disrupting the military's capability to communicate. One effective means to attack a computer network is to develop and release a computer worm into the network.

In a previous paper, we proposed an architecture for mitigating the effects of worm propagation in such an environment, making few assumptions regarding the nature of the worm propagation [3]. Our architecture, which we call *Wireless Taps*, relies on the existence of passive monitors, or Taps, which check on the health of the applications supporting the mission of the MANET. These Taps provide periodic feedback to hosts and network management systems through health reports and recommendations which are broadcast throughout the MANET. They are designed to be impervious to attack by a computer worm spreading through the MANET as they are not addressable. The role of the Taps is to make recommendations about such activities as host capacity restoration or *rejuvenation*, or host/network isolation, based upon the performance of the mission-critical applications running on the host computers.

In this paper, we describe how to apply the monitoring methodology proposed in [1] to the Wireless Taps. That work developed an application health monitor scheme to detect errant behavior in the performance of the applications and made recommendations with respect to server reboots. These health monitors tradeoff between performance degradation due to faulty server state and the loss of server capacity during server reboots. We propose to use a similar approach to detect the presence of harmful worm infections and plan to develop a system which controls the rebooting of servers into fresh states or protected states free of the influence of worms propagating through the MANET. The intent of this aspect of the Wireless Taps Architecture is to prevent mission disruption due to fast spreading, or Flash, worms. Other mechanisms of the architecture, not discussed here, are discussed in [3].

In this paper, we propose and analyze the effectiveness of three different performance signature and software rejuvenation algorithms for the detection and removal of worms infecting a tactical MANET. We study the performance of these algorithms using both analytic and detailed simulation models. We develop a continuous time high level performance model to compute the fraction of infected hosts as a function of worm propagation parameters. The analytic model assumes constant rejuvenation rates as input. In contrast, the simulation model inspects dynamic state behavior

WOSP'07, February 5–8, 2007, Buenos Aires, Argentina.

and determines the best time to perform rejuvenation to optimize application performance. We use a discrete event simulator to model the details of the rejuvenation algorithm and compute its impact on application response time and the steady state probability of infection.

The outline of the paper is as follows: In Section 2 we provide a brief literature review of worm mitigation methods and mechanisms proposed and analyzed, generally in the context of the Internet. In Section 3 we outline our Wireless Taps Architecture for worm mitigation in tactical MANETs. Our architecture concentrates on maintaining a viable communications infrastructure while under active attack by a computer worm. In Section 4 we present our analytic and simulation models. This section further contains a description of the three performance signature and software rejuvenation algorithms. In Section 5 we present our simulation results of the effectiveness of our algorithms in mitigating the impact of a worm on application performance. In Section 6 we discuss our conclusions and directions for future work.

2. LITERATURE REVIEW

There has been a great deal of interest in computer virus and worm mitigation technologies for the Internet. Proposals include network detection monitoring, host-based throttling methods, and host-based detection systems. We provide a brief literature review illustrating the breadth of work in this area, categorizing methods as being either: host-based, network-based, or combined host/network-based patching mechanisms.

There has been also a great deal of research into modeling, monitoring and restoration of software system using software rejuvenation algorithms. A detailed review of the software rejuvenation literature in included in [1] and [2].

2.1 Host-Based Defenses

Worms propagate because the host operating system (OS) has allowed modifications to be made to its active processes. Methods which can preclude this from happening are the best methods for protecting a single host against infection. These methods often require modifications to the underlying operating systems in the hosts. Often, the existence of these host-based defenses can themselves negatively impact system performance.

One form of host-based defenses is automatic signature generation systems which are discussed in [10] [11] [14] and [17]. These focus on methods for the generation of attack signatures, but do not propose approaches to distribute the signatures to peer systems throughout the network. The detection architectures discussed in [13] and [8] detect intrusion attempts and automatically develop code to protect the hosts from the newly identified vulnerability. They assume known categories of attack and are effective against these types of attacks. It is not known whether they can protect against unknown attack types. Local hosts use this information in the form of software patches to remove the worm and prevent future infections. However, without methods to effectively and swiftly disseminate the patch in a secure way, these methods may not prevent a worm from propagating throughout the network.

Host-based throttling mechanisms have been proposed which attempt to slow the worm propagation mechanisms, but not to stop their spread. The hope here is to improve the effectiveness of other techniques by slowing the worm propagation. These have been proposed in [18] and studied in [12] and [16]. These proposals function by slowing down new connection attempts from the host in the hope of slowing the worm propagation. These techniques are often referred to as *Tarpits*.

2.2 Network-Based Defenses

Network-based defenses generally rely on the fact that worms have a significant impact on network traffic, and hence, it should be possible to detect the presence of an infection by monitoring changes in traffic patterns. Upon detection, specific network actions are proposed to block further infection by using mechanisms such as the installation of access control lists at key point within the network, such as at gateways. As an example of this approach, see [5].

Another approach to network monitoring is to look for similarities in the packet payloads generated by a common worm application. These are often referred to as "Entropy Methods", and are discussed in [15]. Worms can attempt to defeat this type of detection by padding payloads or other types of random modifications to meaningless parts of the payload. Further, false positives are possible if other network-based applications generate similar payload fields.

2.3 Combined Host/Network-Based Patching Mechanisms

An architecture consisting of automatic, centralized patch generation sites was proposed and analyzed in [13]. Patches are generated using a set of heuristics to modify vulnerable source code. This idea is advanced in the Vigilante approach [8]. This approach includes methods to a) detect code attempting to execute in the host, b) analyze the code exploit and develop an intrusion message for peer system notification, c) automatically develop a set of code execution filter rules to prevent the code intrusion, and d) provide a method to distribute the intrusion message to its peer systems. Potential limitations of this approach include problems that may prevent the host from detecting, isolating and automatically generating patches for all future unknown worms, and the ability to rapidly disseminate the patches to host peers in the network.

3. THE WIRELESS TAPS ARCHITECTURE

We briefly describe our Wireless Taps architecture [3] in this section. The architecture is based on the assumption that the primary function of the tactical MANET is to support the mission-critical, networked applications. Therefore this should be the focal point of the mitigation architecture. If worm propagation is affecting application performance, then immediate action is taken to rejuvenate the impacted host. In addition, other mechanism, e.g., a relatively low frequency process integrity checking mechanism, can be implemented to detect code modifications due to a slow propagating worm. Thus, the primary function of the Wireless Taps architecture is to provide a secure means of monitoring the performance or health of the applications, and to provide reports of health to all hosts and network managements systems. Based upon these reports, actions can be taken by hosts, network management systems, and human observers to correct any problems identified in the reports.

These potential actions, allocated to the various components comprising our mitigation architecture, include:

- *Wireless Tap Monitors* - these are network embedded agents which passively monitor the performance of applications running over the network. Based upon the observed performance of the applications in relationship to the hosts serving those applications, these embedded agents periodically broadcast concise network health reports throughout the MANET. The Taps are designed to detect out-of-specification behavior of applications residing on distributed servers and to recommend actions for recovery [1]. The performance signature algorithms proposed and analyzed in this paper reside within the Wireless Taps. Based upon network-based performance monitoring, the Taps can develop a set of recommended actions for the network components to take to repair or recover from infection.

- *Hosts* - the hosts in the architecture are required to maintain certain protected code, as discussed in [3]. The code is used to reboot the system into a protected or a fresh state. They further implement some form of rate throttles to help slow worm propagation. These Tarpits may be dynamic in the sense that they are initially turned off but once a tap detects the presence of a worm in the MANET, it can instruct the hosts to turn on their Tarpits. This allows the Taps the ability to detect a fast-spreading worm, and slow the worm down in order to flush the worm from the MANET through a series of host re-initializations. Hosts also implement a low frequency integrity check on its application code resident in memory to determine if any of its running code has been infected by a stealth worm.

  Given the presence of secure, reliable health reports from the Taps, hosts may implement additional capabilities. For example, they may act in a distributed sense to isolate an infected peer from the network in the event that it fails to respond to the recommendations of the health reports. If an identified host is unable or unwilling to reinitialize into a safe (or clean) state, the non-infected nodes can act in concert to isolate the infected, non-repairable host. They can isolate the infected host at the IP layer by refusing to forward packets from that host. They can effectively influence the physical layer to exclude the infected node from accessing the physical layer medium. Thus, even if manual intervention is not possible and the local host is unable to repair itself, there may exist distributed mechanisms to exclude the infected node from the network until later repair is possible.

- *Network Management and Human Intervention* - network management and associated human intervention allows for escalation to a higher entity. For example, the only realistic patching mechanism (which places updated application code on the hosts) may be through escalation to the network management systems which can monitor the Taps' reports and recommendations and take appropriate actions in the event that self-repair of the network seems infeasible. Further, network management systems can perform additional queries of hosts in order to determine additional actions necessary to repair the network.

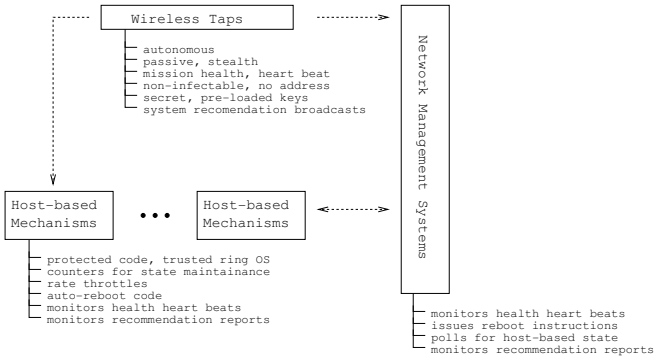

Figure 1: The Mission-Monitor System composed of embedded monitoring points.

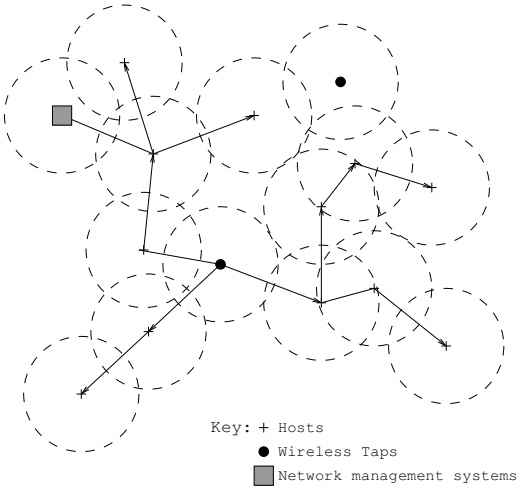

Figure 2: The MANET with embedded monitoring points.

Figure 1 depicts the various Wireless Taps functions discussed above associated with the components comprising the MANET. Figure 2 depicts a physical realization of the MANET with its associated Wireless Taps. Here a broadcast report is propagating throughout the MANET. More details of the Wireless Taps mitigation architecture is discussed in [3].

4. MODELS

In this paper, we analyze the expected performance of our detection and rejuvenation methods using simulation and analytic models. Of course neither view provides actual experience using this technology on real networks. However, at this early stage of architecture development, we are relying upon modeling and simulation to address several open issues. In this study, we provide a first investigation of the effectiveness of application monitoring and rejuvenation in maintaining good application response times in the presence of an on-going computer worm attack. As such, we are not modeling all aspects of the Wireless Taps architecture, but only aspects pertaining to monitoring algorithms and their effectiveness in recommending rejuvenations. Here we

assume that the Taps monitor all transactions and immediately send reports to rejuvenate when necessary.

First, we develop a continuous-time, high-level performance model to compute the fraction of infected hosts as a function of worm propagation parameters. The analytic model assumes constant rejuvenation rates as input. In contrast, we develop a simulation model which inspects dynamic state behavior and determines the best time to perform rejuvenation to optimize application performance. We use a discrete event simulator to model the details of the rejuvenation algorithm and compute its impact on application response time.

In both models worm propagation is parameterized by β, the effective rate at which an infected host attempts to infect other hosts, and γ, the effective rate at which the rejuvenation algorithms refresh infected hosts back into the non-infected states. The discrete event simulation further models the specifics of the rejuvenation algorithms. This level of detail is not reflected in the analytic model.

4.1 Detailed Event Driven Simulation Model

In this section, we begin the development of a simulation model of the Wireless Taps architecture. There are many design and performance issues to address with respect to this mitigation architecture. The goal of this simulation is to assess the effectiveness of the application level monitors in detecting infections and rejuvenating the infected hosts under certain reasonable conditions. In particular, we will evaluate the ability of the performance signature and software rejuvenation algorithms to purge a worm infection from the MANET environment without access to (unknown) software patches. This affects design considerations of various aspects of our mitigation architecture including the speed of passive monitoring to prompt rejuvenations, the rate of host-based throttles and low frequency process integrity checks.

We have modified the simulation model presented in [1] to incorporate worm infection, worm propagation and worm infection removal. We use the same case study for this investigation: namely 16 CPUs with a Java virtual machine (JVM) heap size of 1 Gbyte. The maximum acceptable response time is 10 seconds. The system model is also an extension of the model presented in [1] as follows:

- The simulation starts with Node 0 as the only infected node.
- If Node i is infected, and the Network Radius is R, Node i will send an infection probe to R randomly selected nodes.
- The infection probes will be issued after being delayed by a constant value, D_I, the Infection Delay. If Node i is still infected after the probes have been sent, it will schedule another R probes to randomly selected nodes.
- Threads arrive for processing at a node with an exponentially distributed inter-arrival time with arrival rate λ, and the number of active threads is incremented by one.
- The simulation models thread queuing for CPU.
- The CPU processing time of the thread is sampled from an exponential distribution with service rate $\mu = 0.2$ threads/second.
- If a node is infected, an overhead for processing is incurred for every thread executing in that node. To account for worm infection overhead, the thread processing time is multiplied by a factor of 1.5. This model assumes that the infection consumes a fixed portion of the CPU processing on the host, which is a reasonable assumption for our initial studies.
- As soon as one of the 16 CPUs is acquired, the thread attempts to allocate 10 Mbytes of memory.
- If the remaining memory heap size is less than 100 Mbytes, a full garbage collection event is scheduled and all running threads are delayed by 20 seconds, the time needed to perform the garbage collection on a 1 Gbyte heap.
- Whenever a thread completes service, the total response time of the thread consisting of waiting time plus service time is computed.
- At service completion times, the rejuvenation algorithm being modeled uses the history of the sequence of most recent response times to decide if the host should be rebooted.

Our simulation models three specific performance signature and software rejuvenation algorithms. These are separately discussed in the remainder of this section.

Dynamic Rejuvenation Algorithm

The *Dynamic Rejuvenation Algorithm (DRA)* is taken from [1] and is reproduced below for completeness:

The algorithm tracks the estimate of the system performance, T, in terms of the response time, by maintaining a history of up to $K \times D$ recent response times. We use the notation T to denote the point estimate of system response time. The algorithm divides the history of recent response times into K buckets of depth D. B is the pointer to the current bucket. d represents the number of recent response times stored in the current bucket. $\overline{x}$, the average response time objective, and σ, the objective standard deviation, are derived from the system performance requirements. Specifically,

1. if ($B == K$) execute the rejuvenation routine;
2. if ($T > \overline{x} + B\sigma$)then
   do { $d := d + 1$; } else do { $d := d$ - 1; }
3. if ($d > D_B$) then
   do { d:=0; $D_{B+1} = D_{MAX}/(T - (\overline{x} + B\sigma))$; B:=B+1; }
4. if (($d < 0$) AND ($B >$0)) then
   do { $d := D_{MAX}$; $B := B - 1$; }
5. if (($d < 0$) AND ($B ==$0)) then
   do { $d := 0$; }

The *Dynamic Rejuvenation Algorithm* monitors a sequence of response times and performs software rejuvenation whenever the estimated system response time has deteriorated. The expectation is that software rejuvenation will usually

lead to better response times. This must be weighed against the cost of rejuvenation measured as the percentage of transactions lost during rejuvenation events.

However, in a MANET environment, cost is measured as a result of operational disruption due to worm infection, and the benefit of rejuvenation is reflected in the prompt detection of worm infection, and quick restoration of infected hosts to normal operation. Therefore, in this paper we use the average fraction of infected nodes as a performance metric.

We now introduce our new rejuvenation algorithm designed to identify and mitigate worm infection.

Multiple Warning - Worm Rejuvenation Algorithm

The *Multiple Warning - Worm Rejuvenation Algorithm (MW-WRA)* modifies the *Dynamic Rejuvenation Algorithm* by increasing the sensitivity of monitoring based upon rejuvenations occurring at peer hosts in the network. Specifically,

1. if ($B == K$) execute the rejuvenation routine and send an alert to all neighbor nodes;
2. if ($T > \overline{x} + B\sigma$)then
   do { $d := d + 1$; } else do { $d := d$ - 1; }
3. if ($d > D_B$) OR (alert received from neighbor host) then
   do { d:=0; $D_{B+1} = D_{MAX}/(T - (\overline{x} + B\sigma))$; B:=B+1; }
4. if (($d < 0$) AND ($B >$0)) then
   do { $d := D_{MAX}$; $B := B - 1$; }
5. if (($d < 0$) AND ($B ==$0)) then
   do { $d := 0$; }

Besides monitoring response times and performing software rejuvenation, the MW-WRA sends alerts to neighbor hosts whenever a rejuvenation event has been triggered. The goal of this alerting is to speed up software rejuvenation of neighbors when a significant degradation in performance is detected in any host. Within the Wireless Taps architecture, this information would be picked up through the periodic reports broadcast by the Taps.

Single Warning - Worm Rejuvenation Algorithm

A third, more conservative warning algorithm, is implemented. The *Single Warning - Worm Rejuvenation Algorithm (SW-WRA)* modifies the MW-WRA by decreasing monitoring sensitivity based upon rejuvenations occurring at peer hosts in the network. In this variation, a host acts only upon the receipt of the first warning message, and not whenever a message has been sent as was the case in MW-WMA. Hence, the SW-WMA modifies Step 3 in the MW-WMA algorithm above to be:

3 if ($d > D_B$) OR (first alert received from neighbor host) then
   do { d:=0; $D_{B+1} = D_{MAX}/(T - (\overline{x} + B\sigma))$; B:=B+1; }

The remainder of this algorithm is the same as MW-WMA.

4.2 High Level Continuous Time Analytic Model

In this section we discuss the Kermack-Mckendrick Model [9] as applied to our rejuvenation system. This model extends the Standard Epidemic Model of the propagation of infection to incorporate the process of the removal of the infection. The Kermack-Mckendrick Model is relatively simple and we do not expect that it will model exactly our detailed, event-driven simulation model. However, we do hope to obtain some insights into the underlying competition between the worm propagation and the rejuvenation process which removes the infection.

The simulation model described above is parameterized by

D_I = the time between a host receiving an infection and the time it sends its R infection probes, and

R = the number, or radius, of probes sent to neighboring hosts per D_I.

In the simulation, the infected nodes continue to send R probes each D_I seconds until the system is rejuvenated, causing it to enter the non-infected state. Hence, in this model a host can either be in a non-infected state which is susceptible to infections, or an infected and probing state. Define:

β = the mean probing rate per infected (and probing) host. This mean rate is related to simulation parameters through the relationship $\beta = R/D_I$.

γ = the mean rate of transition from an infected host to a non-infected (susceptible) host. This mean rate is related to the mean rejuvenation time through the relationship $\gamma = 1/\overline{T_R}$. The mean rejuvenation time is defined as the mean time the host remains in the infected state prior to rejuvenation.

For the two host states we define:

$S(t)$ = the number of susceptible, non-infected hosts in the simulation at time t, which equals $N \times s(t)$ where N is the number of hosts in the system.

$I(t)$ = the number of probing, infected hosts in the simulation at time t, which equals $N \times i(t)$.

The capitalized variables represent the number of instances and their small case counterparts represent the probability of the instances. In addition, $N = S(t)+I(t)$. So the system is determined by the evolution of $I(t)$. We can write the time evolution of this quantity as follows:

$$I(t + \Delta t) \approx I(t) + (\beta \Delta t) I(t)[S(t)/N] - (\gamma \Delta t) I(t) \quad (1)$$

The second term on the right hand side (RHS) of the equation represents the increase in I due to infection probes reaching susceptible, non-infected hosts. The third term on the RHS represents the loss to $I(t)$ due to rejuvenation, causing the host to enter the non-infected state. Taking the limit as $\Delta t \to 0$ and dividing both sides of the expression by N, we get

$$\frac{di(t)}{dt} = \beta i(t)[1 - i(t)] - \gamma i(t) \quad (2)$$

This differential equation, along with the initial conditions and parameter relations between β and γ and the simulation parameters define the analytic model.

Steady State Predictions

We now analyze the steady state behavior of our analytic model. The steady state results are found by setting the time derivatives in Eq. (2) to zero. This yields:

$$\lim_{t\to\infty} \beta[1 - i(t)]i(t) = \gamma i(t) \tag{3}$$

This has two solutions; one solution is $i(\infty) = 0$ and the other solution is

$$i(\infty) = 1 - \frac{\gamma}{\beta} \tag{4}$$

Therefore, when $\beta > \gamma$ the system evolves to a steady state probability of infection which is non-zero. The ratio of γ/β can be written as x/R, where x is the ratio of the infection delay to the mean rejuvenation time: $D_I/\overline{T_R}$. Hence, Eq.(4) becomes,

$$i(\infty) = 1 - \frac{x}{R} \tag{5}$$

We see that as x approaches R from below, the infection probability approaches zero in the non-trivial steady state solution.

The steady state solution having a value for the probability of infection between zero and one represents a balance between infected hosts propagating the infection to other hosts versus the rejuvenation detecting infected hosts and rejuvenating them back into non-infected hosts. So the dynamics of a given host represents a life cycle from non-infected to infected and rejuvenation back to non-infected. We have defined the mean rejuvenation time, $\overline{T_R}$, as the mean time from becoming infected to becoming rejuvenated back to a non-infected host. The counter part is the mean time to infection, $\overline{T_I}$, which is the mean time from being rejuvenated to the time the host becomes infected. The average proportion of time a host spends in the infected state is the ratio of $\overline{T_R}$ to the total cycle time, $\overline{T_R} + \overline{T_I}$. Equating this time proportion to the mean probability or infection, $i(\infty)$, results in the relationship between these metrics

$$i(\infty) = \frac{\overline{T_R}}{\overline{T_R} + \overline{T_I}} = \frac{\overline{T_R}}{\overline{T_{BR}}} \tag{6}$$

where we have defined $\overline{T_{BR}}$ as the total cycle time or the mean time between rejuvenations. We have already derived the relationships between $i(\infty)$, $\overline{T_R}$ and the model parameters, β and γ. Using these relationships in the above expression and solving for $\overline{T_I}$ yields

$$\overline{T_I} = \overline{T_R}\left[\frac{1 - i(\infty)}{i(\infty)}\right] = \frac{1}{\gamma}\left[\frac{\gamma/\beta}{1 - \gamma/\beta}\right] = \frac{1}{\beta - \gamma} \tag{7}$$

We have introduced these different metrics, $\overline{T_R}$ and $\overline{T_{BR}}$, because they naturally arise in the analytic model and in the simulation results. Eq.(5) presents a natural scaling result from the analysis in terms of $\overline{T_R}$. Whereas, the simulation results suggest a similar scaling, but in terms of $\overline{T_{BR}}$ instead of $\overline{T_R}$ (see discussion of the simulation results below).

5. EMPIRICAL RESULTS

The model presented in Section 4.1 was implemented in a discrete event simulator. The simulator was run with five replications and the 95% confidence intervals were computed. Each replication was run for 10,000 threads per node. Therefore, for the 50 node example, 2,500,000 threads were simulated.

For each experiment we evaluate the following metrics of interest to assess the performance of the three rejuvenation algorithms under study in this paper:

- the average end-to-end transaction response time, $\overline{T_T}$,
- the mean time between rejuvenations, $\overline{T_{BR}}$, and
- the average fraction of hosts infected, $\overline{i(\infty)}$.

Our simulation results are presented for an infection overhead of 50%, for 50 nodes and for a radius of 10 nodes. The simulation parameters which we vary are the infection delay, D_I, the offered load and the rejuvenation algorithm used (DRA, MW-WRA, and SW-WRA).

Figure 3 shows the first set of simulation results for the mean response time versus offered load for the case of no rejuvenation monitoring within the network. For low offered loads, the mean response times are relatively small in the presence of the worm infection, due to the fact that the overall load on the processors is small and hence do not result in a strong negative impact on the transaction times. As the offered load increases, we see a rapid degradation in the transaction times. Four separate results are shown in this figure reflecting four values for the infection delay ranging from a high of 200 seconds to a low of 50 seconds. Within the range of response times plotted, the different infection delays show very little difference in results.

Figure 4 shows the impact of rejuvenation monitoring on the mean transaction times plotted against load. In this case, as the load increases, the rejuvenation detection algorithms detect the "out-of-specification" transaction time performance and invoke the rejuvenation process which brings the infected host back into a fresh uninfected state. Recall that in this state, the host is susceptible to later infection. We give the results for the three rejuvenation algorithms outlined earlier. All algorithms show exceptional performance, when compared to behavior without rejuvenation. As might be expected, MW-WRA shows the best performance, followed by SW-WRA, when compared to DRA and no rejuvenation at all. Two different values for the infection delay are also given in this plot, namely $D_I = 100$ and 200 seconds. We see little difference in the results for these two values.

In Figures 5 and 6 we plot the mean time between rejuvenations, $\overline{T_{BR}}$ versus the offered load. These results reinforce our understanding of the dynamics in the simulations. For small offered loads, the mean response times are relatively small compared with the target expected by the rejuvenation monitoring algorithms. Therefore the mean times between rejuvenation are quite large. At the low load range, there is little need for rejuvenation as the mean response times are still within mission targets. However, as the transaction loads increase, the mean time between rejuvenations decrease as the need for rejuvenation increases. Figure 5 shows results for the three rejuvenation algorithms with two different infection delays. Here the differences in

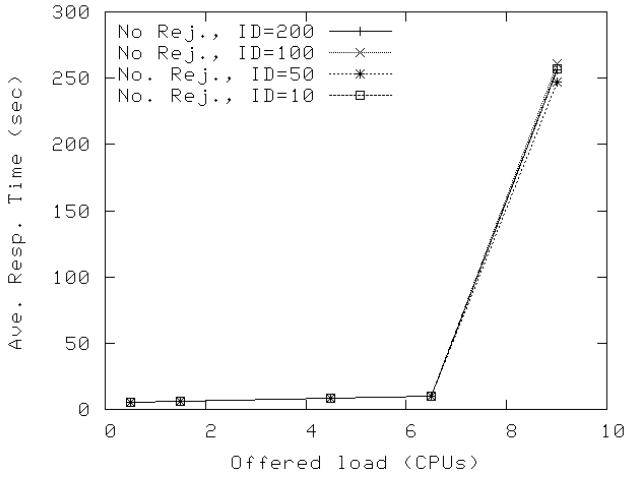

Figure 3: Response time in seconds versus load with no rejuvenation.

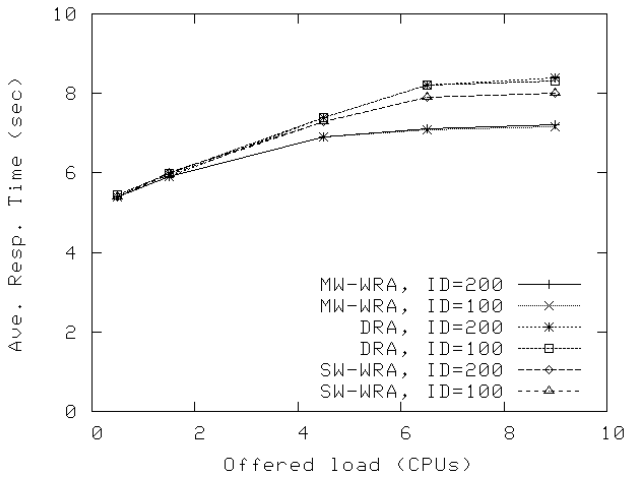

Figure 4: Response time in seconds versus load with various rejuvenation algorithms.

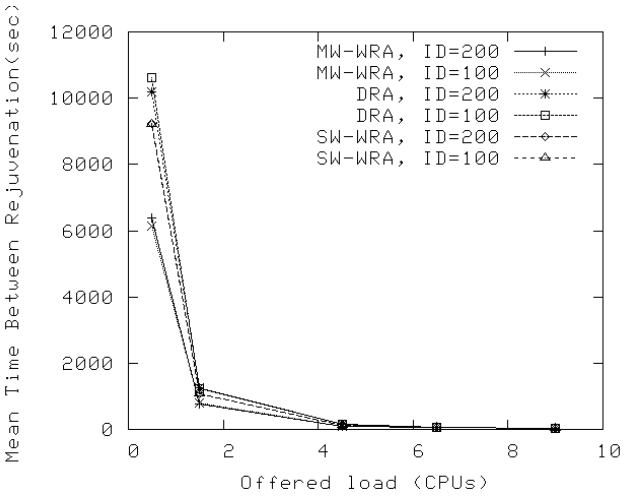

Figure 5: Mean time between rejuvenation in seconds versus load with multi-warning, worm rejuvenation.

the mean times between rejuvenation seem to be caused by the differences in the performance of the different rejuvenation algorithms. As before, the difference between the DRA and the SW-WRA is slight. The impact of the infection delay seems to be minimal. Figure 6 shows an expanded view of the DRA and the MW-WRA algorithms. We omit the results for the SW-WRA to improve readability since its performance is similar to the DRA results.

Figure 7 shows the collected simulation results plotted as the probability of infection, $\overline{i(\infty)}$, versus the ratio of the infection delay and the mean time between rejuvenations, $D_I/\overline{T_{BR}}$. We see that the simulation results collectively fall roughly on a line given by $0.9(1 - x'/2.5)$ where here $x' = D_I/\overline{T_{BR}}$. This is similar to the scaling behavior suggested by the analytic model earlier, although the analytic scaling was based upon the mean rejuvenation time, $\overline{T_R}$ (see Eq.(5)), rather than the mean time between rejuvenations, $\overline{T_{BR}}$ shown here. It is not surprising that there are differences between our simulation and analytic models since the simulation model is far more detailed, including response times, rejuvenation algorithms and worm propagation throughout a 50 node network. In contrast, the analytic results are based on a very simple two parameter differential equation of the mean infection probability. The simulation model captures the detailed statistical fluctuations of a relatively complex system while the analytic equation models the mean infection probability given an average rejuvenation rate.

Although neither the analytic model nor simulation model reflect any actual experience, we are still encouraged as they indicate that by rejuvenating a host into a fresh, but still susceptible state, the worm infection has a higher likelihood of being extinguished than predicted by differential equations modeling the mean value of the infection probability. Of course only a series of empirical studies using systems implementing these algorithms can provide actual evidence of the effectiveness of rejuvenation at arresting worm propagation.

Finally, in this initial performance study of aspects of our Wireless Taps mitigation architecture, we have used a rather

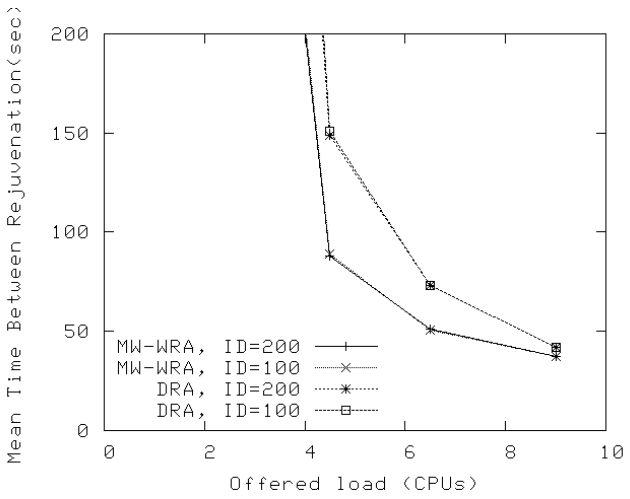

Figure 6: Mean time between rejuvenation in seconds versus load with multi-warning, worm rejuvenation.

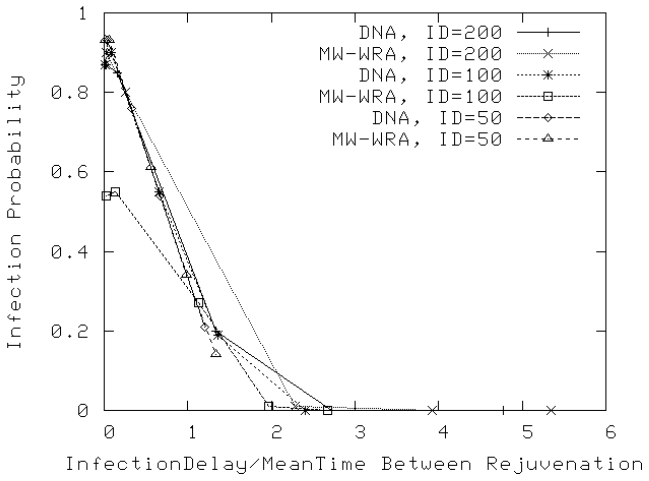

Figure 7: Average probability of infection versus the ratio of the infection delay to the mean rejuvenation time.

simplified and somewhat conservative worm model with respect to its local impact on host resources. We have assumed that the impact of worm infection on the host is to increase proportionally the transaction processing times. Hence, the worm impact on the host is proportional to the transaction load. However, typically worm infections impose their own processing load on the host, independent of any other transaction load. Also, our architecture adopts the notion of Tarpits which are host-based throttles. These should interact with worm processing to lessen its impact on host resources. These effects are also to be addressed in future studies.

6. CONCLUSIONS AND FUTURE WORK

In [3], we proposed a worm mitigation architecture for tactical, MANET environments. In this paper we have extended this architecture by proposing and analyzing three performance signature and software rejuvenation algorithms designed explicitly to mitigate the effects of a worm attack. We also performed both analytic and extensive simulation studies to see whether it is likely that this architecture coupled with rejuvenation will be effective against worm attacks. Preliminary results presented in this paper indicate that there is reason to further pursue this line of research.

This worm mitigation architecture is unique in the following ways:

- It focuses on detecting performance deviations from expected performance signatures of secured applications.
- It relies on the presence of embedded, non-addressable application-level monitors, called Wireless Taps, to provide periodic application performance reports to the network and recommended actions for rejuvenation of hosts.
- The Wireless Taps compliment a range of host-based, network-based and network management-based mechanisms previously described in the literature.
- It makes few assumptions regarding the nature of the vulnerability and focuses instead on the mission-level application performance.

Our initial set of modeling and simulation studies related to key aspects of the Wireless Taps architecture and focused on the detection and rejuvenation algorithms to be placed in the embedded Wireless Taps in the MANET. We leveraged and extended the work in [1] on application performance monitoring algorithms. Our modeling and simulation studies found:

- There are indications that the application level monitoring and rejuvenation algorithms are capable of maintaining good performance in the presence of a propagating worm infection in the network.
- The rejuvenation algorithms do a good job at trading off good response time performance versus the host downtime during rejuvenations.
- Both the differential equation analysis and simulation results indicate that there is a scaling behavior, although they are somewhat different in form.

- Both the differential equation analysis and simulation results indicate regions where the rejuvenation algorithms totally extinguish the worm infection from the network.

As this is our initial performance investigation of the Wireless Taps architecture, there is still a lot of work remaining to improve our understanding of the design and parameter tuning as well as the effectiveness of the architecture. Specifically, we plan the following future investigations:

- Incorporate periodic local code integrity checks and rate throttles and investigate their impact of the architecture and rejuvenation algorithms in terms of the mitigation performance.
- Explore the state space by considering worm spread parameters, other monitoring algorithms, local throttles and rate of periodic or random functions.
- Incorporate network mobility, Wireless Taps nodes and design considerations such as the number and location of Taps and their overall effectiveness in monitoring.
- Design algorithms to isolate hosts during worm infection and assess their performance. When hosts notice their neighbors ignoring Taps performance reports and recommendations, what actions should they take to isolate offending hosts from the network layer, from the MAC layer and from the physical layer.
- We have modeled a small, homogeneous set of hosts. It would also be interesting to extend our modeling to large sets of heterogeneous hosts that are more likely representative of the diversity in real deployed tactical systems.

7. REFERENCES

[1] A. Avritzer, A. Bondi and E. J. Weyuker, *Ensuring Stable Performance for Systems that Degrade*, Proc. Fifth International Workshop on Software and Performance 2005, Palma de Mallorca, Spain, July, 2005, pp. 43–51.

[2] A. Avritzer, A. Bondi, M. Grottke, K. Trivedi and E. J. Weyuker *Performance Assurance via Software Rejuvenation: Monitoring, Statistics and Algorithms*, Proc. of the International Conference on Dependable Systems and Networks 2006, Philadelphia, PA, June 2006.

[3] A. Avritzer, R. G. Cole, N. Phamdo and A. Terzis, *The Wireless Taps Worm Mitigation Architecture for Tactical MANETS*, JHU / Applied Physics Laboratory Technical Report, July, 2006.

[4] N. T. Bailey, *The Mathematical Theory of Infectious Diseases and its Applications*, Hafner Press, New York, 1975.

[5] L. Briesemeister and P. Porras, *Microscopic Simulation of a Group Defense Strategy*, ACM/IEEE Parallel and Distributed Simulation (PADS), Monterey CA, June, 2005.

[6] R. G. Cole, N. Phamdo, M. A. Rajab and A. Terzis, *Requirements on Worm Mitigation Technologies in MANETS*, ACM/IEEE Parallel and Distributed Simulation (PADS), Monterey CA, June, 2005.

[7] R. G. Cole, *Studies of Worm Propagation in Mobile Ad-Hoc Networks for Future Combat Systems*, Army Science Conference 2004, Orlando FL, December, 2004.

[8] M. Costa, et.al, *Vigilante: End-to-End Containment of Internet Worms*, SOSP'05, Brighton, United Kingdom, October 2005.

[9] J. C. Frauenthal, *Mathematical Modeling in Epidemiology*, Springer-Verlag, New York, 1980.

[10] H. Kim and B. Karp, *Autograph: Toward automated, distributed worm signature detection*, USENIX Security Symposium August 2004.

[11] C. Kreibich and J. Crowcroft, *Honeycomb - creating intrusion detection signatures using honey pots*, In HotNets, November 2003.

[12] V. Paxson, *Bro. a system for detecting network intruders in real time*, Computer Networks 31, 23-24, pp 2435-2463, December, 1999.

[13] S. Sidiroglou and A. Keromytis, *A Network Worm Vaccine Architecture*, in Proceedings of the 12th IEEE WET ICE / STCA Security Workshop, June, 2003.

[14] P. K. Singh and A. Lakhotia, *Analysis and detection of computer viruses and worms: An annotated bibliography*, ACM SIGPLAN Notes, vol. 37, no. 2, pp 29-35, February, 2002.

[15] A. Wagner and B. Plattner, *Entropy Based Worm and Anomaly Detection in Fast IP Networks*, in Proceedings of the 14th IEEE WET ICE / STCA Security Workshop, June, 2005.

[16] N. Weaver, S. Staniford and V. Paxson, *Very fast containment of scanning worms*, In USENIX Security Symposium, August, 2004.

[17] J. Wilander and M. Kamkar, *A comparison of publicly available tools for dynamic buffer overflow prevention*, NDSS, February 2003.

[18] M. Williamson, *Throttling Viruses: Restricting propagation to defeat malicious mobile code*, in the Annual Computer Security Applications Conference, 2002