High-Fidelity Modeling of Computer Network Worms

Kalyan S. Perumalla kalyan@cc.gatech.edu
Srikanth Sundaragopalan srikanth@cc.gatech.edu
College of Computing, Georgia Institute of Technology, Atlanta, Georgia

Abstract

Abstract modeling, such as using epidemic models, has been the general method of choice for understanding and analyzing the high-level effects of worms. However, high-fidelity models, such as packet-level models, are indispensable for moving beyond aggregate effects, to capture finer nuances and complexities associated with known and future worms in realistic network environments. Here, we first identify the spectrum of available alternatives for worm modeling, and classify them according to their scalability and fidelity. Among them, we focus on three high-fidelity methods for modeling worms, and study their effectiveness with respect to scalability. Employing these methods, we are then able to, respectively, achieve some of the largest packet-level simulations of worm models to date; implant and attack actual worm monitoring/defense installations inside large simulated networks; and identify a workaround for real-time requirement that fundamentally constrains worm modeling at the highest fidelity levels.

1 Introduction

Abstract models such as epidemic models have so far been the general means of choice for modeling worm propagation. However, such models possess limitations due to their many simplifying assumptions. They are useful for certain studies, such as post mortem analysis, but otherwise poor in versatility. This is especially true in their inability to accommodate complex scenarios, such as sophisticated worms, elaborate defense mechanisms, rich network topologies and variety in background traffic.

An effective alternative is packet-level modeling, which is capable of capturing many fine details and scenario variants. However, packet-level simulations have so far been considered computationally expensive and hence prohibitive. Few packet-level models have been employed for large-scale simulation studies of worms. A reason behind the limited use of packet-level models is that, until recently, it has been constrained by sequential execution. Lately, with the advent of effective parallel/distributed processing techniques, packet-level network simulations are enabling the execution of very large-scale network models (a few millions of nodes) at packet-level. These parallel systems support large-scale configurations of detailed software models of routers and links of the network, loaded by synthetic traffic introduced at end-host models. By utilizing such scalable packet-level simulation environments, it is now possible to effectively simulate and analyze the propagation (and other) behaviors associated with worms, under realistic large-scale phenomena such as network congestion, feedback and rich topological layouts. Moreover, these packet-level environments can be incrementally augmented on demand, with additional models, such as of defense/quarantining mechanisms at end-hosts and/or gateway/core routers.

By exercising this new level of packet-level fidelity enabled by the state-of-the-art parallel network simulators, we undertake worm modeling at a large-scale, and explore the current (quantitative) limits of those environments. The issues, challenges and results in the development of these large-scale, *packet-level worm models* constitute the first component in our contributions.

In our second component, we explore the possibility of further increasing the fidelity afforded by the packet-level models, by focusing on substituting parts of the large-scale network with real, operational subsystems. In particular, we look at the issue of incorporating live monitoring/defense systems into a large simulated network. A honeypot system is immersed in the virtual network, yet it is made oblivious to the fact that it is operating within a virtual world. We describe issues and challenges in enabling such a capability, which we call *constructive emulation*. As will be described in greater detail later, this differs from traditional network emulation systems in a significant way.

Finally, in our third component, we present a novel architecture, namely, *full system virtualization*, which is designed to resolve the scalability problems inherent

in methods in which virtual models interact with real systems. While almost all high-fidelity systems are limited at one scale or another by real time execution constraint, this fully virtualized system is free from the constraint (hence arbitrarily scalable in theory), albeit at some cost of degraded computational efficiency.

The rest of the document is organized as follows. Section 2 presents the spectrum of worm modeling alternatives and motivates the need for high-fidelity worm modeling. Issues, challenges and results from large-scale packet-level worm modeling are described in Section 3. Constructive emulation and its application to honeypot emulation are presented in Section 4. The full system virtualization approach is outlined in Section 5. Finally, conclusions and future work are presented in Section 6.

2 Modeling Alternatives

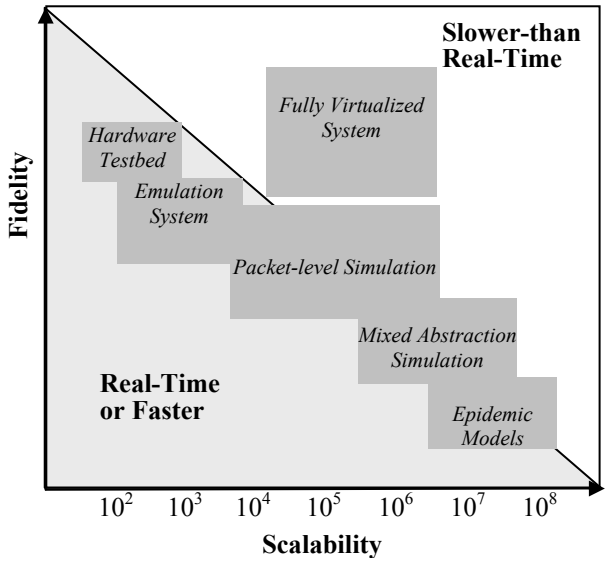


Figure 1: Range of computer worm modeling alternatives. Scalability ranges are based on current day capabilities of testbeds and tools reported in literature.

The spectrum of alternatives for worm modeling is shown in Figure 1. These alternatives roughly mimic the alternatives for network modeling in general. For example, hardware testbeds developed for broad network research can be applied for studying worms as well, since worms only represent a special case of network applications.

Scalability of a method is defined as a limit on the number of network nodes modeled by that method. Network nodes include end-hosts and routers. Fidelity in general is hard to define, but it is possible to compare two methods with respect to their relative fidelity. Fidelity could be based on the amount of detail accounted for in the network (e.g., routing, network congestion, etc.), or in end-hosts (e.g., stack processing, operating system overheads, etc.).

Hardware testbeds have improved in scale with recent advancements, with network testbeds scaling to hundreds or more nodes (e.g., EmuLab[1, 2]). However, hardware testbeds cannot by themselves sustain fidelity with increasing scale. In fact, hardware testbeds resort to some form of network simulation underneath to improve fidelity when virtual configurations exceed physical resources in size (e.g., to emulate link delays or losses).

The next level of scalability is achieved via network emulation. Emulation systems for network security analysis have also scaled in size, with recent emulators capable of sustaining a few thousand nodes (e.g., NetLab[2] and ModelNet[3]), and are being used in major network security studies (e.g., the DETER project[4]). Both hardware testbeds as well as emulation systems are by definition executed in real-time.

The next logical alternative is packet-level simulation. Historically, network simulation experiments have always been done on small scale and the results of such experiments have been extrapolated to derive conclusion on large scale simulations. However, results on small scale are hard to extrapolate to larger configurations and hence can be misleading. Large scale network simulations are thus required for detailed and realistic simulations, where individual network parameters might produce significant difference in the behavior of experiments. A key impediment in running large scale simulations is the scalability of the network simulators. Packet-level simulation has lately seen great advances, especially due to parallel/distributed execution capabilities of network simulators (e.g., PDNS and GTNetS[5], and Genesis[6]). Scales of up to a few million nodes have been reported for simulation of large TCP/IP networks[7, 8].

Hybrid simulations, using a combination of fluid and packet level models have been used to scale network simulations by at least another order of magnitude[9, 10], but they have been largely constrained in generality (e.g., limited accounting for feedback effects). Further, they are restricted to core network segments (backbone links and routers), and are difficult to extend to worm application traffic at end-hosts.

Use of simplified epidemic models, such as the SIR model, is quite widespread in the literature (e.g., see [11]). They are the most scalable as they simply use a system of (differential) equations, but also exhibit the least fidelity due to their simplifying assumptions about network and traffic dynamics. Mixed abstraction simulations have also been designed that aim to

combine the fidelity of packet-level worm models (in subnets of interest), with the scalability of aggregate epidemic models[12, 13] (in other uninteresting portions of the network).

As can be seen, packet-level simulation exhibits the best tradeoff between scalability and fidelity, and holds potential to sustain Internet-scale experiments without great loss of flexibility or accuracy. Complex worms can be easily modeled in terms of their TCP/IP packet exchange behavior, and the simulation can be enhanced as needed if/when new defense mechanisms or new worm types are explored (e.g., worms based on header spoofing, or defenses based on packet header analysis[14]). For these reasons, we focus on packet-level models as a replacement for large-scale networks in worm modeling (Section 3).

We also explore ways to immerse actual defense installations into such replaced, virtual networks to test such installations under controlled, repeatable attack scenarios, as described in Section 4. High-fidelity modeling enables constructive emulation, which in turn can obviate the development of models for operational defense/monitoring installations such as honeypots[15] or other early warning apparatus[15].

At sufficiently large levels of scale (e.g., hundreds of thousands of nodes or larger), most high-fidelity systems break down, because of their inability to keep up with real-time for their "real-system" components. For such large scales, we propose a *fully virtualized system* to overcome their inherent real-time constraint, without sacrificing fidelity. The fully virtualized system has the potential for achieving the highest fidelity among all approaches, even higher than that of a general hardware testbed. Theoretically it is not limited in scalability, but it is only limited practically by the amount of computation power available. This approach is described in greater detail in Section 5.

3 Packet-level Modeling & Execution

3.1 Simulators

To develop packet-level worm models, we chose two parallel network simulators that represent the state-of-the-art: PDNS and GTNetS. The Parallel and Distributed Network Simulator (PDNS) is an extension of the popular ns-2 simulator. The Georgia Tech Network Simulator (GTNetS) is a C^{++}-based simulator developed at Georgia Tech. Both these packages are downloadable on the Web[16, 17]. We chose these packages due to our intimate familiarity with them, although one could choose another similar parallel simulator such as DaSSF[18].

The simulators allow an arbitrary subject network configuration to be specified (topology, normal user traffic, etc.) and initialized accordingly. Normal user traffic can be realized as end-applications with either customized packet-level behavior or aggregate statistical traffic. Malware (worms) can be injected, activated, and/or initialized into this network. Complex scripts of attack/detection/defense scenarios can then be enacted. Several different types of outputs can be obtained from the simulated scenarios, including the obvious ones such as the number of infected hosts at any given instant. Since the test-bed is a time-controlled software-based simulation, certain network measurements and statistics can be obtained from the scenario execution, which are difficult or impossible to obtain in a hardware test-bed (e.g., sub-millisecond granularity of network event statistics, or an accurate global snapshot of entire network). Both simulators boast demonstrated scalability, simulating several million TCP/IP packet transfers in a single wall-clock second.

We have developed worm models in both PDNS and GTNetS. Both implementations realize the generalized worm model framework described next.

3.2 Models

As a generalization of several worm types, we chose the model depicted in Figure 2. This model contains the following components:

- **Worm Node**: This represents an end-host in the network which can potentially act as a node spreading the worm.
- **Vulnerable Service**: This represents the flawed network service that is penetrated by the worms to infect the host machine.
- **Shooting Agents**: Once a worm node is compromised, Shooting Agents take over the task of propagating the worm from the current host to other vulnerable hosts.
- **Backdoor Agents**: These agents model the backdoor entry which is opened by the initial infection on a vulnerable host. The backdoor is used to transfer larger worm payload, if any.

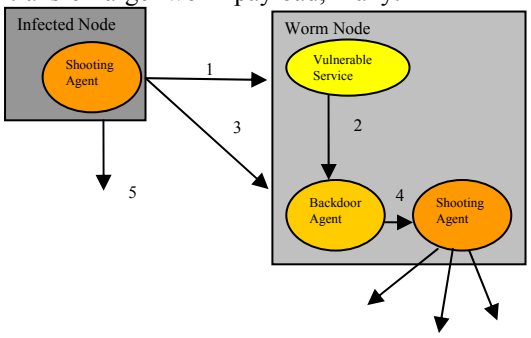


Figure 2: Models of vulnerable worm nodes, agents and their interaction sequence.

We start the simulation by marking one node as infected. A shooting agent is instantiated which starts generating random scans for spreading that infection.

As illustrated in Figure 2, the following steps are involved during the worm propagation.

1. The shooting agent finds a random host and makes a connection to that host. A vulnerable service on the worm node responds to this connection request and the shooting agent transfers a payload to it. This typically models a worm's initial step (e.g., malicious URL sent by a worm to a web server).
2. The infection triggers a backdoor port to be opened on the worm node.
3. The worm payload is (attempted to be) transferred to the worm node by initiating a connection to the opened backdoor port.
4. Once the worm payload is transferred the worm application instantiates a shooting agent on the worm node. This node is now infected and follows the preceding steps to propagate itself.
5. The original worm is finished with this infection attempt, and hence goes back to repeating the preceding steps all over again. An adjustable delay is modeled between infections at this step.

3.2.1 Model Parameters

Almost every aspect of the preceding model is customizable via a corresponding parameter. The following are examples of such parameters:

- Scan Rate: The scan rate in worm propagation could affect the overall pattern of the infection. This parameter is configurable in our system. Multi-threaded worms are modeled by instantiating one shooting agent per thread on each node.
- Topology: The worm model we have developed is completely independent of the underlying network topology. One could deploy our model in any kind of network topology by just instantiating the vulnerable nodes and attaching the corresponding agents to those nodes.
- Background traffic: During the worm propagation we inject normal traffic going through the network. The amount and pattern of background traffic could affect the propagation of a worm. In our model, one could introduce background traffic in addition to the normal worm traffic.

3.3 Unused IP Addresses

In the Internet, not all IP addresses allocated to an organization are used. "Holes" are typically present in the address space covered by each organization. Packets destined to these unused addresses usually travel all the way to the closest router of the unused address and then get dropped at that router. Such packet drops corresponding to unused IP addresses become the common case during worm propagation, and hence become especially important to model accurately.

When worm models generate random IP addresses during their scans, packets destined to unused IP addresses should not be dropped at the source, because doing so will not correctly model congestion effects that would otherwise be created by such packets further down in the network.

Unfortunately, most network simulators drop the packets at the source if their destination is not present in the simulated network topology (some simulators are even worse in that they terminate with a runtime error). We were faced with this challenge, namely, to find a way to model this correctly.

3.3.1 Maximal Prefix Match Scheme

One way to deal with this is to modify the simulator to route the packet as far as it can, similar to the Internet's operation. While an entirely accurate approach would be to model the Border Gateway Protocol (BGP), modeling BGP in fully glory is an extremely complex endeavor. A compromise is to perform "maximal prefix match" on addresses – route a packet towards the address that maximally matches the prefix of the packet's destination address[19]. A drawback of this approach is that it requires complex overhaul of the simulator. Another disadvantage is that the prefix match operation potentially needs to be performed at every hop along the packet path, incurring substantial runtime overhead.

3.3.2 Blackhole Scheme

We developed a novel approach that is an efficient alternative to the preceding approach and avoids both aforementioned drawbacks. At every intermediate router, a "blackhole" end-host is instantiated and attached to that router. The blackhole is assigned a unique unused IP address that is reachable via that router. A table is maintained that maps the subnets reachable via a router to its corresponding blackhole end-host. Instantiated (used) IP addresses simply map to themselves. When a source generates a packet, it first checks the table to determine the mapped address for the destination address. For used IP addresses, the destination remains unchanged since they map to themselves in the mapping table. For unused IP addresses, the destination is replaced by the blackhole address whose subnet maximally matches the original destination address. The blackhole end-hosts are configured to simply drop all packets destined to them.

This scheme ensures three things: (1) it forwards packets to the blackhole closest to the unused IP address (2) performs the maximal prefix mapping exactly once per packet (3) the lookup into the mapping table is optimized to reduce the table size by eliminating the identity mapping for used IP addresses if the addresses happen to be contiguous. While this

scheme pushes the modeling burden to the user, this was not a major problem for us, since the random IP address generation is quite isolated and easy to modify in our worm models.

3.4 Other Issues

3.4.1 Pre-allocation

An obvious runtime performance improvement is to minimize dynamic memory allocation. Buffers could be pre-allocated during initialization and reused at runtime. Pre-allocation was done at all possible place to reduce runtime overhead. This includes pre-allocation of a sufficient number of TCP agents for modeling incoming and outgoing connections made by the worms. Without pre-allocation, this overhead could dominate the simulation runtime, because worm models make heavy use of such agents for infections.

3.4.2 TCP vs. UDP Worm Models

We have modeled both TCP and UDP versions of the worms and have found a significant difference implementation complexity between the two. TCP worms are more complex to model, due to bookkeeping complexities in connection establishment, and the need for creating a new TCP agent object for every new random connection. Such complexity is absent in UDP models, as it is sufficient to simply send a packet with the worm payload and easily mark the destination node as infected when the packet is received (e.g., for modeling SQL Slammer).

3.5 Performance Study

3.5.1 TCP Worm Models

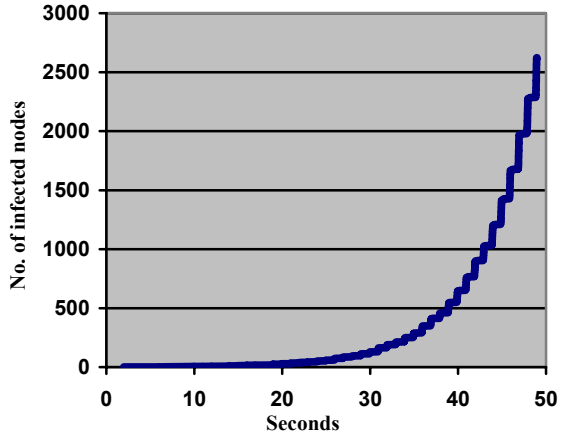


Figure 3: Initial propagation phases of a Code Red II-like worm in a 1,280,000-node network. Interestingly, exponential increase in infections is observed to start in as early as 35 seconds since initial infection. Experiment uses GTNetS at packet-level on a 128-CPU Linux cluster.

Using TCP-based worm models mimicking the Code Red II worm, we ran experiments to test the feasibility of large-scale high-fidelity worm modeling. For our TCP worm experiments, we simulated a clique network of core routers, mapped one per CPU. A two-level tree hangs off each core router, with parameterized fan-out at each level. A 128,000 node network is instantiated on 128 CPUs, with a tree of 10x100 on each CPU. Similarly, a 64,000 node network uses 64 CPUs, with 10x100 nodes mapped per CPU. The 1.28 million node network contains a 100x100-node tree per CPU.

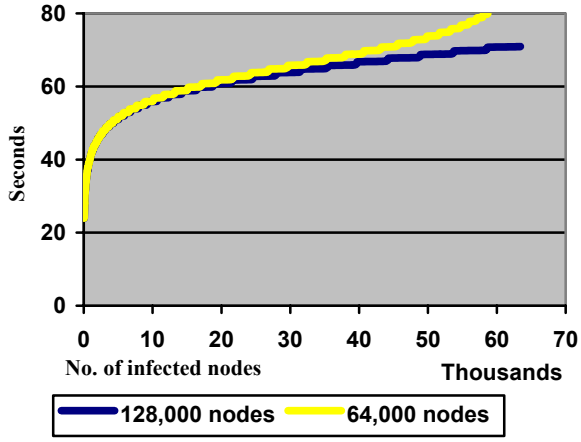


Figure 4: Propagation of a Code Red II-like worm in relatively smaller networks (packet-level GTNetS model).

Figure 3 plots the propagation of the TCP-based worm. This execution is among the largest TCP worm models simulated to date at packet-level. Figure 4 plots the propagation of the same TCP-based worm on networks of 64,000 and 128,000 nodes.

3.5.2 UDP Worm Models

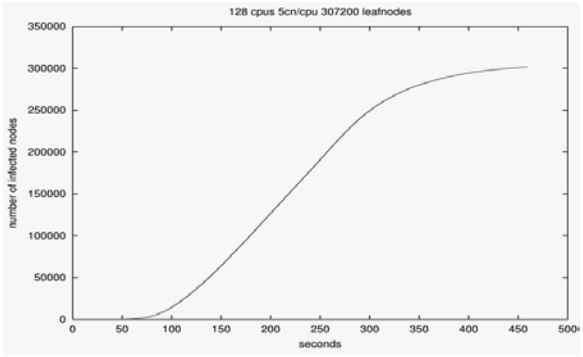


Figure 5: Propagation of Slammer-like UDP worm through 307,200 nodes of a complex network topology. Entire simulation is performed at packet-level in PDNS on a 128-CPU Linux cluster.

Using packet-level models of a UDP worm similar to the SQL Slammer, we have simulated the worm propagation on a large-scale network containing over

300,000 end-hosts. Results from a sample worm propagation experiment on the 300,000 node network are shown in Figure 5. The network topology consists of several "university campus-like" subnets connected via a network of gateway routers. Each campus subnet consists of roughly 30 routers and 508 end-hosts.

This experiment represents results from among the largest packet-level worm simulations to date that we are aware of. It is presented to illustrate the scale at which all packet-level models can be simulated using current day's computation platforms and simulation tools. It captures all network details, such as queuing and congestion at routers, etc. The worm is seen to follow the expected well-behaved trajectory of an epidemic model. More importantly, the additional power of packet-level model lies in its ability to easily accommodate complex variations to the worm behavior (e.g., intelligent hit list scanning), and dynamics of network topologies, background traffic intensities, etc.

4 Constructive Emulation

Armed with the ability to perform high-fidelity simulation of worm models, we were ready to face our next challenge. In our projects related to modeling and simulation of military networks[20], we were tasked to explore exercising actual network security installations against simulated scenarios of large-scale worm attacks. Testing the security systems against simulated attacks provides the benefits of flexible, controllable and repeatable experiments, in contrast to using live testbeds. The initial candidate installation to be tested was that of a honeypot system. While at first it appeared to be a straightforward application of traditional network emulation techniques, closer analysis revealed that such a scenario represented a higher-fidelity experiment that requires a new emulation capability, as described next.

4.1 Emulation Architectures

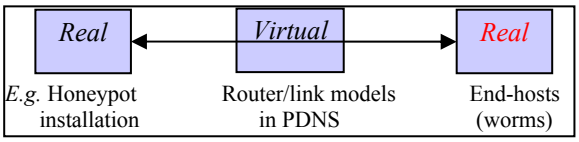


Figure 6: Traditional emulation architecture.

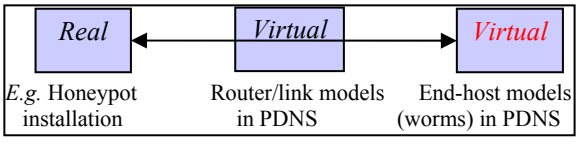


Figure 7: *Constructive* emulation architecture.

A drawback of existing emulation systems is that they require end-hosts to be realized as real end-host systems. The traditional emulation architecture is shown in Figure 6. For example, worm infections will need to originate and terminate in real hosts. However, this makes it difficult to experiment with very large number of vulnerable end-hosts in worm propagation experiments.

A majority of existing emulation systems, including the most scalable ones, such as Netbed[2] of University of Utah and MAYA[21] of UCLA, allow the simulated portion of the emulation to only act as a transport plane, without network endpoints. The LARIAT[3] system of MIT supports virtual end-host applications, but is limited by very low-fidelity network models. In our worm emulation scenarios (e.g., honeypot emulation, as described next), however, we need simulated vulnerable/infected nodes to interact directly as endpoints of worm infections with actual honeypot installations. The virtual portion of the emulation system is thus required to maintain all the state associated with every interaction endpoint that can interact with real endpoint system. For example, full TCP state machine needs to be modeled and maintained at each simulated end point. Traditional emulation systems are neither equipped to maintain such state, nor possess the necessary translation mechanisms to bridge the semantic/representational gap between modeled endpoints and real systems. In traditional emulation systems, this would require thousands of real hosts to be configured and integrated into the emulation setup, to be able to experiment with large-scale worm propagation at high-fidelity. Instead, it is desirable to have only a small subset of end-hosts realized as real hosts, while the rest of the end-hosts are instantiated virtually inside simulation, as depicted in Figure 7. For example, it is sufficient to realize the honeypot installation(s) on real hosts (say, configured and run on Linux), while the worms themselves originate from end-hosts simulated inside the network simulator. This makes it possible to achieve scalable execution of worm propagation (inside the simulator), while still retaining the ability to test honeypot installations against large-scale worm attacks.

Our *constructive emulation* approach solves precisely this problem, with its ability to interface real operational systems directly with simulated end-host applications. The objective of this approach is a setup which essentially bridges simulated network and real network in a "head-to-head" fashion. This setup can then be used to plug-in real network applications into simulated networks to measure effectiveness and other metrics. To demonstrate the capability of our emulation interface, we have integrated *honeyd*, which is a popular real honeypot implementation, with our worm simulation models. *Honeyd*[22, 23] is a low-interaction, but highly scalable, honeypot framework.

Other equivalent systems are also well-documented in the network security literature (e.g., see [24]).

4.2 Example: Honeypot Emulation

Honeypots are introduced into the simulation using the following scheme. The entire network of interest is configured with end-hosts and routers, as usual, in the simulator. However, the end-hosts where honeypots need to be inserted are marked as "emulated nodes". In these nodes, special "Emulated TCP" agents are used instead of the usual simulation models of TCP. The emulated TCP agents do not have any TCP end-point behavior by themselves, but simply act as conduits to the TCP stacks inside the actual honeypots. For example, when a SYN packet event is received at an emulated node, the emulated TCP agent performs some simple conversions to format it into a bonafide network packet, and forwards it to the honeypot (via a proxy server, as will be described later). The emulated TCP agent maintains minimal state required for translating simulation events into network packets and vice versa (e.g., translating sequence numbers). Each virtual IP address hosted by the honeypot is represented and associated with a corresponding emulated node in the simulator.

Since our network is simulated in parallel, the network is partitioned across multiple processors. We need to distinguish among the processors based on whether they hold any emulated nodes or not. If a processor does not house any emulated nodes, it simulates in an as-fast-as-possible (AFAP) mode (i.e., oblivious to emulation). Processors that house emulated nodes need to perform special initialization, to set up conduits to the honeypot installation. This is done via socket-based communication between the emulator processor and proxy server. Additionally, the simulation loop is modified to accept incoming packets and emit outgoing packets, and to pace the execution with real-time. The honeypot emulation setup is shown in Figure 8.

In our PDNS execution, two types of PDNS instances run in the public network: AFAP and Real. The PDNS-Real instances instantiate an emulator object which registers itself with the proxy server. During registration, the emulator object sends the instance ID and the IP address range it covers. Using this information the proxy maintains a connection table to route traffic to/from simulated from/to the honeypot. A TCP connection is established between the emulator and the proxy during the registration process and is held open throughout the simulation to exchange packets with the proxy server.

4.2.1 Experimental Hardware Setup

Figure 9 shows the hardware setup for honeypot emulation. The machine running the proxy server is equipped with two network interfaces to bridge the public and the private network. A DHCP daemon is started on one of the network cards, which acts as a gateway for *honeyd*. Since *honeyd* simulates virtual IP addresses, a route entry is added to forward packets destined to the virtual IP addresses on to this network card.

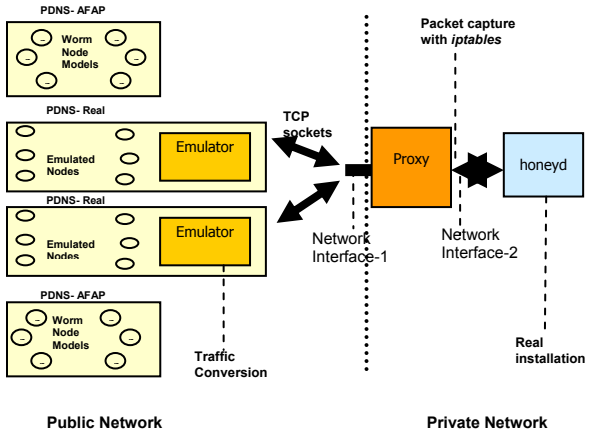


Figure 8: Constructive emulation setup of the *Honeyd* honeypot with parallel simulation of PDNS worm models.

To avoid the runtime overhead of ARP lookups at the proxy server for *honeyd*'s virtual IP addresses, we hardcoded the mapping of the virtual IP addresses to *honeyd*'s MAC address in the proxy's ARP table. The proxy server listens for packets from both sides of the network (PDNS and *honeyd*). It uses a set of *iptables*[25] rules to capture packets from *honeyd* and encapsulate them to insert into PDNS, and uses raw sockets to emit forwarded packets onto the *honeyd* side.

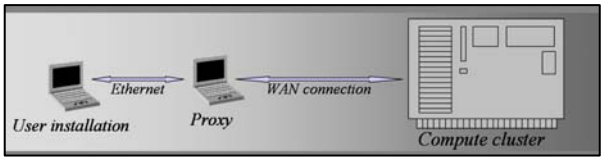


Figure 9: Hardware hosting platform for constructive emulation: user installation (*honeyd*) is connected over a wide area connection to a compute cluster that executes worm models in PDNS. The user installation of network defense could be a single end-host or an entire subnet.

4.3 Emulation Issues

Several issues arose in implementing a constructive emulation architecture. Some of these issues are discussed next.

4.3.1 Traffic Conversion

In PDNS, sequence numbers for TCP connections always start at a fixed value (zero). But, in real TCP implementations, such as used in the honeypot, TCP packets start at randomly generated sequence numbers. The sequence numbers need to be dynamically translated to match the receiver's view, to prevent

incorrect packet drops or packet buffering. We have solved this problem by bridging the gap with a conversion module inside the emulated TCP agents in PDNS. Similarly, both TCP and IP checksums are not modeled in PDNS, but are required for real packets. Again, we resolved this by generating (stripping) checksums just before (after) packet emission (reception) in the emulated TCP agent.

Note that such conversions need to be performed on a per-connection basis, and hence state needs to be maintained for each connection (e.g., starting sequence numbers). The emulated TCP agents are natural holding points for such state.

4.3.2 Real-time pacing

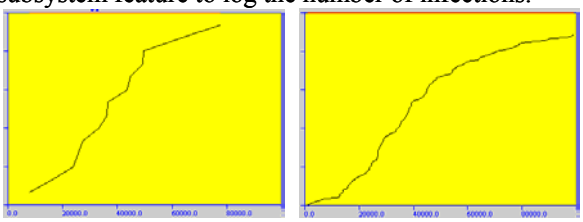
In a network simulator, packet events are processed by dequeueing the next earliest event and advancing to its timestamp. Thus, the simulator finishes processing all events in an as-fast-as-possible (AFAP) mode. But when interfaced with a real network, packet arrivals are dynamic. Packets can arrive at unpredictable times from the real network. This prevents the simulator from running as fast as possible, and instead mandates real time pacing. Real time pacing was hence added into the main event processing loop of PDNS.

Constructive emulation shares this real time aspect with traditional emulation. However, a nuance is that the simulator executes in parallel. To maximize runtime performance, we implemented a scheme by which AFAP instances are never constrained by real-time, but only the non-AFAP instances are paced with real-time. This helps one to intelligently balance the simulation load across processors, so that most of the simulated network is simulated in AFAP mode, while only the few processors containing emulated nodes are constrained by real-time.

Another sticky issue is again related to parallel execution. A factor called *lookahead* [26] (LA) is crucial for efficient parallel execution of the simulators. We expand this notion using two kinds of LA: simulation LA and emulation LA. We omit the details of these schemes here, in interest of space. Both schemes are essential to ensuring maximal parallelism in the entire system.

4.4 Sample Execution

In an experimental setup, we used two laptops (an IBM Pentium-4 ThinkPad, and an IBM Pentium-III ThinkPad), running the proxy server and *honeyd* respectively. The Code Red II worm model previously described was used to attack a network in which *honeyd* was implanted into a 2-CPU PDNS execution. As expected, *honeyd* received and logged incoming TCP malicious connections. As no other traffic is destined to the honeypot, every incoming TCP connection at *honeyd* constituted a worm scan/infection attempt. We used *honeyd*'s virtual subsystem feature to log the number of infections.



a. Infections in honeypot **b.** Infections in entire network

Figure 10: Tracking infections with honeypot emulation.

Figure 10 **a** & **b** show the propagation of the worm in a sample 200-node network (simulated on 2 CPUs) and the number of infection attempts logged by *honeyd*. As can be expected, the number of infections in the honeypot roughly tracks the total infections in the entire network, accurately reflecting the fact that uniformly random addresses are generated by our worm model. In larger experiments, we are able to run the honeypot with networks with tens of thousands of nodes (by employing suitably larger number of CPUs).

We note that while honeypot-based tracking is not novel, the ability to plug-in actual honeypot installations into large virtual network has not been realized before.

5 Full System Virtualization

Packet-level modeling goes a long way in accounting for many network dynamics and application characteristics that are hard to accommodate in aggregate methods. Emulations (traditional & constructive) help enhance the fidelity of packet-level models to even higher levels by incorporating actual systems into portions of simulated scenarios. However, packet-level simulations are limited in fidelity by the amount of detail incorporated into their models. Emulations are limited in scalability due to the imperative for real-time execution to keep up with their real-system components. In scenarios demanding even higher levels of fidelity at large scale, alternative methods are needed.

For example, packet-level simulations generally do not model operating system effects such as process scheduling delays. Other details in worm behavior, such as root exploits, are extremely hard to model at fine level of detail. Nevertheless, researchers would greatly benefit from the possibility of capturing full system effects at large-scale. Execution at that combination of fidelity and scale cannot be met by existing packet-level or emulation methods. Also, modeling at that level of detail entails prohibitively expensive model development efforts, which can approach the effort of building the real systems

themselves. In fact, precisely this aspect (models asymptotically approaching real systems) leads to our next modeling alternative: a fully virtualized system.

5.1 Virtualization Technology

Virtualization is an approach to enabling a software system designed for one platform to execute on a different platform. For example, while the Linux operating system (OS) is originally designed to execute directly on native hardware, virtualization technology enables the same OS to execute on top of another "host" OS. The host system provides a virtual platform layer to make the hosted system oblivious to the changed environment. Virtualization technology is recently gaining significant attention, with many systems being virtualized. For example, it is now possible to boot up multiple Windows OS instances as mere processes hosted by another Windows OS instance. Similar capabilities exist for Linux and other systems as well. With optimizations, virtualized systems are achieving acceptable performance.

5.2 High-Fidelity Models & Virtualization

In the context of high-fidelity computer worm modeling, virtualization technology can be applied to circumvent the traditional conflict between scalability and fidelity. Imagine an internet that entirely executes not by real-time clocks, but on virtual (simulation) clocks. Such a network not only retains the highest fidelity level, but also is de-linked from real-time completely. Such a virtual network can be achieved as follows. Network links and routers are modeled using traditional packet-level (parallel/distributed) network simulators. End-hosts are modeled as real systems themselves, with full blown OS's, file systems, etc. However, unlike emulation systems that have end-hosts running on real hardware, the end-hosts are executed in virtualized environments. Since the end-hosts are now under the control of a virtual host, they are not free-running anymore, and hence can be controlled at will. Since the network is executed as a (packet-level) simulation, its execution is also already controllable.

This approach leaves two issues to resolve: communication and timing. The first, namely, network communication, is easily patched between the network simulator and the virtualized end-hosts by converting packet exchanges into simulation time-stamped events. Events encapsulate actual network packets. The second issue, namely timing, is more complex. To be able to fully virtualize the entire set of network-and-endhosts, it is necessary to synchronize their execution with simulation time. Without proper synchronization, we would be faced again with the original real-time execution constraint of emulation systems. Instead, time advances would be carefully controlled by simulation clock. When a virtual OS instance queries for hardware clock value, the host OS is made to supply it the simulation clock value, rather than the real-time clock value. Elapsed time is estimated using similar techniques (e.g., Direct Execution[27]). The simulation clock is synchronized across parallel network simulator and virtual system hosts using standard parallel/distributed discrete event simulation techniques. The net effect of these arrangements is that all components of the entire system are lifted away from real-time and placed on a controllable virtual timeline.

5.3 Related Work

Recent emulation efforts are in fact moving in the direction of virtualization. Netbed/EmuLab[2] has recently added some support for virtual nodes[28] to multiplex more than one end-host or router on the same physical resource. Our work differs from virtual nodes of Netbed in that we are interested in virtualizing the entire system, including the end-hosts and routers. This involves re-mapping real-time of the end-hosts (virtualized end-hosts) to simulation time that is synchronized with the network (routers). In existing emulation/live methods, end-hosts are paced by real-time, whether they are multiplexed or not. Due to this requirement and other reasons, existing methods are typically difficult to scale to more than a few hundreds of end-hosts.

5.4 Implementation Systems

As indicated earlier, a range of virtualization alternatives is becoming available for production use, and any level of virtualization (processor, OS sandbox/jail, etc.) can be chosen. User Mode Linux can host multiple Linux instances as processes. Revirt is an optimized Linux virtualization system with very low overheads. Other examples are the VMWare virtual machine framework and the SIMICS virtual processor-simulator, both of which can boot unmodified OS's. Any of these platforms could be employed to achieve full system virtualization. When they are integrated with parallel packet-level network simulators, unprecedented level of fidelity is enabled combined with arbitrary scalability for computer worm modeling. While the virtualized systems can incur significant memory and runtime overheads, they represent the only choice available when absolute scalability and high-fidelity are essential.

6 Conclusions and Future Work

Current worm & network security modeling are done at either low fidelity or a low scale. Few are capable of simulating the effects of worms on networks of complex topologies at sufficient detail to capture effects such as congestion. Recent advances in packet-

level network simulations have enabled the possibility of modeling worms at increased fidelity. We have demonstrated this aspect by developing packet-level worm models, and performing some of the largest packet-level simulations of worm propagation to date. Using these models, we are investigating the validation of results from epidemic models, especially for pathological Malware such as Warhol worms.

We have also created a unique ability to interface simulated nodes with real nodes, to exercise actual defense installations with large-scale simulated attacks. As an application of this approach, we are able to subject actual honeypot installations to large-scale packet-level models of worm propagation. We are exploring using this system to test defenses such as automated patching of infected hosts.

For the highest levels of modeling fidelity, such as those employing full-fledged OS functionality on end-hosts, we propose an approach using a fully virtualized system. In this direction, we are exploring the use of User Mode Linux to run entire Linux instances in large-scale worm experiments.

7 Acknowledgements

This work has been supported in part by DARPA contract N66001-00-1-8934. We would like to thank Dr. Richard Fujimoto for supporting this work, and to Ram Kumar Gandhapuneni for initiating the million-node TCP scenario execution.

8 References

[1] A. Vahdat, *et al.*, "Scalability and Accuracy in a Large-Scale Network Emulator," Operating System Design and Implementation (OSDI), 2002.

[2] B. White, *et al.*, "An Integrated Experimental Environment for Distributed Systems and Networks," Fifth Symposium on Operating Systems Design and Implementation, 2002.

[3] R. Durst, *et al.*, "Testing and evaluating computer intrusion detection systems," in *Communications of the ACM*, vol. 42, 1999, pp. 53-61.

[4] DETER-Project, "Cyber Defense Technology Experimental Research (DETER) Network", 2004/05/31, www.isi.edu/deter/.

[5] G. F. Riley, *et al.*, "A Federated Approach to Distributed Network Simulation," *ACM Transactions on Modeling and Computer Simulation*, vol. 14, pp. 116-148, 2004.

[6] B. K. Szymanski, Y. Liu, and R. Gupta, "Parallel Network Simulation under Distributed Genesis," 17th Workshop on Parallel and Distributed Simulation, 2003.

[7] K. S. Perumalla, *et al.*, "Scalable RTI-based Parallel Simulation of Networks," Workshop on Parallel and Distributed Simulation, 2003.

[8] R. M. Fujimoto, *et al.*, "Large-Scale Network Simulation -- How Big? How Fast?," IEEE/ACM International Symposium on Modeling, Analysis and Simulation of Computer Telecommunication Systems (MASCOTS), 2003.

[9] C. Kiddle, *et al.*, "Hybrid Packet/Fluid Flow Network Simulation," IEEE/ACM Workshop on Parallel and Distributed Simulation (PADS), 2003.

[10] Y. Guo, W. Gong, and D. Towsley, "Time-Stepped Hybrid Simulation for Large Scale Networks," INFOCOM, 2000.

[11] S. Staniford, V. Paxson, and N. Weaver, "How to Own the Internet in Your Spare Time," USENIX Security Symposium, 2002.

[12] M. Liljenstam, *et al.*, "A Mixed Abstraction Level Simulation Model of Large-scale Internet Worm Infestations," International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS), 2002.

[13] D. Nicol, M. Liljenstam, and J. Liu, "Multiscale Modeling and Simulation of Worm Effects on the Internet Routing Infrastructure," International Conference on Modeling Techniques and Tools for Computer Performance Evaluation (Performance TOOLS), 2003.

[14] A. Hussain, J. Heidemann, and C. Papadopoulos, "A framework for classifying denial of service attacks," Applications, technologies, architectures, and protocols for computer communications, 2003.

[15] C. C. Zou, *et al.*, "Monitoring and Early Warning for Internet Worms," ACM Conference on Computer and Communication Security (CCS), 2003.

[16] G. F. Riley, "The Georgia Tech Network Simulator (GTNetS)", 2004/05/31, www.ece.gatech.edu/research/labs/MANIACS/GTNetS/.

[17] G. F. Riley, "PDNS - Parallel/Distributed NS", 2004/05/31, www.cc.gatech.edu/computing/compass/pdns/.

[18] D. Nicol and J. Liu, "Dartmouth Scalable Simulation Framework (DaSSF)", 2004/05/31, http://www.cs.dartmouth.edu/~jasonliu/projects/ssf/.

[19] Monirul Islam and George F. Riley, Personal Communication.

[20] K. S. Perumalla, *et al.*, "Experiences Applying Parallel and Interoperable Network Simulation Techniques in On-Line Simulations of Military Networks," 16th Workshop on Parallel and Distributed Simulation, 2002.

[21] J. Zhou, *et al.*, "MAYA: Integrating hybrid network modeling to the physical world," *ACM Transactions on Modeling and Computer Simulation*, vol. 14, pp. 149-169, 2004.

[22] N. Provos, "A Virtual Honeypot Framework," USENIX Security Symposium, 2004.

[23] N. Provos, "Developments of the Honeyd Virtual Honeypot", 2004/05/31, http://www.honeyd.org.

[24] Honeypots.Net, "Honeypots, Intrusion Detection, Incident Response", 2004/05/31, http://www.honeypots.net.

[25] R. Russell, "Netfilter: Firewalling, NAT and Packet-Mangling for Linux 2.4", 2002/09/20, www.netfilter.org.

[26] R. M. Fujimoto, "Parallel Discrete Event Simulation," *Communications of the ACM*, vol. 33, pp. 30-53, 1990.

[27] P. Dickens, P. Heidelberger, and D. M. Nicol, "Parallelized Direct Execution Simulation of Message-Passing Programs," *IEEE Transactions on Parallel and Distributed Systems*, vol. 7, pp. 1090-1105, 1996.

[28] S. Guruprasad, *et al.*, "Scaling Network Emulation with Multiplexed Virtual Resources," SIGCOMM 2003 (Poster Abstracts), 2003.