A Realistic Simulation of Internet-Scale Events

Songjie Wei and Jelena Mirkovic
Department of Computer and Information Sciences
University of Delaware
Newark, DE 19716
(weis, sunshine)@cis.udel.edu

ABSTRACT

Internet-scale security incidents are becoming increasingly common, and the researchers need tools to replicate and study them in a controlled setting. Current network simulators, mathematical event models and testbed emulation cannot faithfully replicate events at such a large scale. They either omit or simplify the relevant features of the Internet environment to meet the scale challenge, thus compromising fidelity. We present a distributed worm spread simulator, called PAWS, that builds a realistic Internet model, including the AS-level topology, the limited link bandwidths, and the legitimate traffic patterns. PAWS can support diversity of Internet participants at any desired granularity, because it simulates each vulnerable host individually. Faithful replication of Internet environment, its diversity and its interaction with the simulated event, all lead to a high-fidelity simulation that can be used to study event dynamics and evaluate possible defenses. While PAWS is customized for worm spread simulation, it is a modular large-scale simulator with a realistic Internet model, that can be easily extended to simulate other Internet-scale events.

Keywords: Invasive software, worms, simulation and modeling, distributed simulation

1. INTRODUCTION

Internet stability is increasingly threatened by large-scale events, such as distributed denial-of-service (DDoS) attacks, flash-crowd attacks, worm spread, routing instabilities, botnet recruitment, spam and coordinated intrusions. To study these events and evaluate possible countermeasures, we must be able to replicate them, as faithfully as possible, in a controlled setting. This replication must include all the relevant Internet features that interact with a chosen event, such as host diversity, limited link bandwidth, routing dynamics and connectivity, background traffic, etc.

Network events are usually studied using mathematical models, emulation and simulation. All three approaches have so far failed to replicate a realistic and detailed model of the Internet environment and its interaction with the event of interest. Mathematical models of worm propagation [1],[2],[3] produce results of limited fidelity, due to heavy approximations of host diversity, and a complete lack of an Internet model. Testbed environments usually contain no more than several hundred machines, so Internet-wide events have to be scaled down to the testbed size. This scale-down results in a simplified topology and introduces a loss of fidelity [4]. Popular network simulators [5],[6],[9],[8],[7] simulate network events and protocols at a very fine level of detail and at a packet level. This leads to high memory demand and long execution time for large-scale simulation. Distributed simulators [9],[8],[7] support large scale but require powerful clusters to simulate at a reasonable speed; such clusters are not available to all researchers.

Large-scale, realistic Internet simulation is a very challenging task. It lies on a thin line between two extremes: (1) simulating too many details at too fine a granularity results in a prohibitively costly simulation with regard to memory and CPU resources, while (2) heavy approximation and oversimplification of the Internet environment and the event of interest lead to incorrect results. Floyd and Paxson discuss in [10] the additional Internet simulation challenge, that stems from the lack of necessary data about relevant Internet features, such as connectivity, link bandwidths, traffic patterns and mixes, and congestion levels. We must simulate just the right amount of detail to attain the goals of scale and speed, while maintaining fidelity. We must also build a realistic Internet model, drawing the relevant features from the real Internet observations, and superimposing the simulated event and its interactions with the underlying environment.

In this paper we describe PAWS, an Internet-scale worm spread simulator, which simulates a large number of hosts, with a reasonable speed and in a realistic Internet environment. PAWS uses distributed simulation to harvest memory and CPU resources from multiple networked machines. It replicates a realistic Internet topology at the AS level, limited inter-AS and host access link bandwidths, and legitimate traffic at the aggregate, inter-AS level. During the simulation, PAWS captures the interaction between congestion-responsive TCP traffic and aggressive worm traffic, as they travel through limited-bandwidth links. Each simulation node simulates a portion of the entire Internet, and nodes synchronize using network messages at discrete time intervals. Each vulnerable host is simulated individually; this facilitates the specification of host and network diversity at various levels. Each worm scan packet is simulated sep-

VALUETOOLS'06, October 11-13, 2006, Pisa, Italy

arately, but packet transmissions and receptions occur at specified time intervals (currently 1 s). This reduces the number of network messages exchanged between simulation nodes, and synchronization delays, leading to faster simulation. PAWS is an open-source simulator, implemented on the Emulab testbed [14], and thus is accessible to a large community of researchers. While it is currently customized for worm spread simulation, PAWS can be easily extended to simulate other Internet-scale events.

In Section 2 we survey related work in network simulation. We then describe the Internet, background traffic and worm models implemented in PAWS in Sections 3, 4 and 5. Section 6 describes PAWS implementation in the Emulab testbed, and Section 7 presents validation experiments, simulating the propagation of Code Red v2 and SQL Slammer worms. In Section 8 we present PAWS operational cost, such as simulation time and communication overhead for distributed node synchronization. We conclude in Section 9.

2. RELATED WORK

Our work on PAWS is related to three areas of network research: Internet simulation, worm modeling and worm simulation. We survey the closely related approaches below.

2.1 Internet simulation

Much work has been done on large-scale distributed simulation of network events, and several large-scale simulators are closely related to PAWS: PDNS [7], GTNetS [8] and SSFNet [9]. These simulators replicate Internet layers in greater detail than PAWS, and simulate each traffic flow and each packet separately. While this is beneficial for small-scale simulation, it results in a prohibitively slow and non-scalable simulation of Internet-wide events, as we describe in the Section 2.3. With regard to fluid (aggregate) traffic simulation, Nicol et al. [15], [16] propose to model flows at coarse time-scales. In their model, each sender and receiver are associated with a variable-rate traffic stream, and the aggregated flow on each link is computed iteratively, hop-by-hop, along the routing path until it converges. In [17] Nicol et al. simulate the interaction between some foreground traffic of interest, at the packet level, and the background flows, at the fluid level, using a discrete-event formulation. The difference between their work and our aggregate TCP traffic model is that PAWS simulates TCP's congestion response at the aggregate level, while such mechanism is lacking in [17]. PAWS also avoids iterative, hop-by-hop, traffic simulation which would prohibitively increase simulation time, while bringing a small gain in accuracy.

2.2 Worm modeling

Staniford et al. [1] use a simple epidemiological model for the spread of Code Red. Zou et al. [2] present a "two-factor" worm model, which improves the epidemiological model adding human countermeasures and a simple network congestion model. Chen et al. [3] provide a discrete-time worm model that considers effects of human patching and cleaning on worm propagation dynamics, and replicates worms with a subnet scanning strategy. All the surveyed models heavily approximate the interaction of the worm spread with the underlying topology, and do not replicate host and network diversity, nor the background traffic. These approximations and simplifications lead to inaccurate results.

2.3 Worm simulation

Savage et al. [18] model Internet topology at the AS level, along with the epidemiological worm spread model. Wagner et al. [19] model delay and bandwidth differences between vulnerable hosts by grouping Internet hosts into four categories and modeling the interactions both between and within categories. Liljenstam et al. [20], [21] design a worm model integrated with the SSFNet [9] simulator. This model adds simulation of countermeasures at a router level to the simple epidemiological model, and simulates those actions assuming a single router per AS. A realistic AS topology from the Route Views project [12] is used in a scaled-down form, due to single-machine memory constraints. Riley et al. [22] develop a packet-level worm simulator on GTNetS [8] to observe connection-level behaviors of TCP worm propagation. They approximate the Internet topology as a set of limited access links that connect vulnerable hosts to the "core" with unlimited bandwidth. Perumalla et al. [23] propose a large-scale packet-level simulation of worm propagation on PDNS [7]. All the above approaches deploy an overly simplified model of the Internet topology and do not simulate the interaction of the worm traffic with the background traffic, both of which lower simulation fidelity. In addition to this, all but [23] are single-node simulators, which limits their scalability to several thousand vulnerable hosts. [23] simulates worm propagation among 1.28 million vulnerable nodes at a dedicated 128-CPU cluster, using PDNS and GTNetS. In another paper on distributed simulation of Internet events [24] the authors note that PDNS and GTNetS can simulate about 95K packet transmissions per wall-clock second (PTS) at a single machine and 5.5M packet transmissions on a dedicated 136-CPU cluster. This is an excellent result, given that both PDNS and GTNetS simulate traffic at a flow and packet level, and must maintain connection state for each packet transmission. We note two main problems with this approach for large-scale simulation: (1) To achieve reasonable simulation speed (5.5M PTS) one needs a powerful 100+ node cluster. Not every researcher has access to such a cluster, yet many researchers need a high-fidelity Internet simulator that can run in reasonable time on multiple common PCs, to validate their ideas; (2) Packet-level simulation in PDNS and GTNetS generates a network message for each packet sent to a different simulation node which incurs huge overhead when simulating high-rate worm scans. Instead, these transmissions can be aggregated and sent in a single network message at the end of a simulated time unit, like it is done in PAWS.

3. INTERNET MODEL

In this section we describe a detailed Internet model, replicated in PAWS. Such model is necessary to capture the interactions between the worm spread and the Internet connectivity, bandwidth, legitimate traffic and host diversity.

3.1 Topology

PAWS currently models the Internet at the Autonomous System (AS) level, using the data from the Route Views project [12] to reconstruct the Internet topology. The Route Views project provides periodic snapshots of the BGP routing tables of a few participating ASes. We use the as_path information from these snapshots to infer connectivity and relationships between ASes, and the ownership of IP address

ranges. The Route Views data set provides a limited picture of the Internet topology [25]. It may miss some backup links because they are not used in best path selection, but the reconstructed topology preserves many important properties of the real topology [25] and should contain majority of links used by data traffic. The Route Views project provides daily routing table snapshots spanning almost nine years. This enables us to reconstruct a realistic Internet topology for the specific day of a given worm breakout, and increases simulation fidelity.

3.2 Routing and forwarding

In addition to the topology, we must carefully reconstruct Internet routing and forwarding, to obtain a complete and correct Internet model. In our AS-level Internet model, each AS is represented by a single router. PAWS infers the forwarding table information for each router from the Route Views data. We first derive a list of BGP atoms — these are groups of prefixes that share the same AS path from any source [26]. We then allocate one entry per atom in the forwarding table of each simulated router, and populate the entries using the following steps:

1. Populate forwarding tables of the ASes participating in the RouteViews project with the next-hop information specified in the as_path fields.

2. Partially populate forwarding tables of ASes that do not participate in the Route Views project, but appear in as_path fields of other participants. For example, if AS 6 participates in Route Views project and reaches destination X via ASes 7, 8, and 9, path 6-7-8-9 will exist in BGP dump logs of AS 6. In step 1, we specify 7 as the next hop for AS 6 and destination X. In step 2 we will insert 8 as the next hop for AS 7 to X, 9 as the next hop for AS 8 to X and will mark 9 as the origin AS for X.

3. Populate remaining vacant entries by calculating shortest path route on the AS map.

The current PAWS simulator uses static routing model. We plan to implement dynamic routing response to congestion as part of our future work.

A worm may generate scans to non-routable addresses, i.e., to addresses that are not assigned to any AS. Since such addresses will not have an entry in BGP routers' forwarding tables, they will be dropped by the first BGP router they encounter. We simulate the non-routable scan drops at the exit router of the AS containing the worm.

3.3 Link bandwidth model

Worm spread events produce a huge volume of probe traffic, which frequently leads to serious network congestion. To simulate this phenomenon, we need to model the limited bandwidth of each inter-AS link. Although in reality, two ASes may have multiple physical connections we simplify this in our simulation by assuming a single limited-capacity link between any two connected ASes.

We use the Pathneck [13] project's data to guide our choice of reasonable inter-AS link bandwidth values. Pathneck project estimates the upper bound of the available bandwidth on bottleneck inter-AS links, by sending a train of packets on a path, and measuring packet inter-arrival times at routers along the path. The measurement process has some limitations: (1) It provides only an instantaneous sample of the upper bound of the available bandwidth. (2) It can only estimate the upper bound for the links preceding a bottleneck link and the *lower bound* of the available bandwidth for links after the bottleneck. (3) Some links will have no records in the Pathneck data as they have not been traversed by a measurement train, while other links may have multiple records due to multiple measurement trains traversing different paths. Finally, we were able to obtain only a single set of Pathneck samples, collected at the early 2004. This prevents time rollback for the links that we were able to perform for the connectivity using Route Views data for a chosen date. We are aware that these multiple limitations compromise fidelity of the bandwidth values inferred using the Pathneck data. Still, we believe that by providing an upper bound and the distribution of the related, *available* bandwidth values, Pathneck data allows us to make an informed guess about the distribution and the range of reasonable link bandwidth values.

Cat.	Percentage	Bandwidth	Link Type
1	95%	51.84 Mbps	OC-1
	5%	1244.16 Mbps	OC-24
2	90%	51.84 Mbps	OC-1
	10%	9953.28 Mbps	OC-192
3	80%	155.52 Mbps	OC-3
	5%	466.56 Mbps	OC-9
	15%	13271.04 Mbps	OC-256

Table 1: Link bandwidths chosen in PAWS

3.3.1 Grouping links based on the projected load

The first step in inferring the bandwidth value of each link is grouping of similar links into categories. The grouping approach enables us to infer bandwidth values for links that do not appear in the Pathneck data, using the data for other links in the same category. Two links are considered similar if they have a similar expected traffic load, which justifies the assignment of the same bandwidth value to both links. We calculate a link's expected traffic load by following paths from each AS to all other ASes in our topology, and summing the product of the source and the destination IP size (number of IPs allocated to an AS) in a score assigned to each traversed link. We call this score the "IP product"; it represents the projected number of IP pairs that can communicate over a given link and can be interpreted as the indicator of the expected traffic load on this link. For example, if AS A has 256 allocated addresses, and AS B has 512 addresses, the IP product of all links on the path from A to B will increase by $256 \cdot 512 = 131,072$.

The main limitation of estimating a traffic load from the IP product lies in the fact that many addresses assigned to an AS may not be allocated to a live host, and the well-known irregularity of host traffic patterns [28]. However, in the absence of detailed information about Internet traffic patterns and the address allocation to hosts, the IP product enables us to make an informed guess about the magnitude of the expected traffic load on each link.

We next observe the distribution of IP product values, shown in the Figure 1(a) and derive three link categories: links with a small demand ($IP_{prod} \leq 10^{10}$), medium demand ($10^{10} < IP_{prod} \leq 10^{13}$), and large demand ($IP_{prod} > 10^{13}$). We next separate the Pathneck samples into these three cat-

egories, and quantize the distribution of available bandwidth (given by Pathneck data) in each category. The outcome of this process are several quantiles for each category, and the probability of drawing a sample randomly from a given quantile. Figure 1(b) shows the distribution of the Pathneck data in the category 3 and its quantization. Finally, we use specifications of common connection speeds [27], and choose the first higher connection speed (bandwidth) value for each quantile. We then assign chosen bandwidth values to links in the same category, based on the quantiles' probabilities. We list the bandwidth values and their probabilities for each link category in the Table 1. For example, a link whose IP product falls into category 3 will be an OC-3 link with the probability 80%, the OC-9 link with the probability of 5% or the OC-256 link with the probability of 15%.

3.3.2 Host access links

Another limiting factor affecting worm propagation is the bandwidth of an infected host's Internet access link. To faithfully replicate the diversity of host connection speeds (dial-up, DSL, cable, etc.) we retrieve host bandwidth distribution from the Annual Bandwidth Report [29], for a chosen date. For example, for our Slammer simulation, described in Section 7.2, we used the bandwidth distribution data for work users in early 2003 to infer host access link bandwidths. Since Slammer was targeting a vulnerability in Microsoft's SQL server, which is not commonly used by home users, work user data was best suited for our purpose. Half of the hosts in Slammer simulation are assigned a narrowband T1 connection (3.152 Mbps), while the other half have high-speed connections of 100 Mbps.

3.4 Simulating congestion on inter-AS links

For a link j with bandwidth y_j, we represent the total traffic at a time i (containing both the legitimate and the worm traffic) on this link with $x_j(i)$. If $x_j(i) \leq y_j$, then all the traffic will be delivered to its destinations without congestion drops. Otherwise we simulate the congestion on this link, and each worm probe is forwarded over the link with a probability of $y_j/x_j(i)$. We call this probability the *pass ratio* of the link.

We acknowledge that this congestion model is much simpler than the real router's behavior. For example, early congestion notification mechanisms, such as RED [11], drop traffic when utilization exceeds some threshold, whereas we assume that all traffic can pass the link if link's utilization is below 100%. Further, excessive congestion on a link may crash the router, denying service to all link's traffic, whereas we assume that the link will be fully utilized in this case, while excess traffic will be dropped. PAWS could easily accommodate different router models to provide more realistic congestion simulation, but we lack detailed models of routing behavior under heavy load, and data about prevalence of RED-like routers.

4. BACKGROUND TRAFFIC MODEL

In addition to the Internet-scale compromise of vulnerable hosts, another serious threat of a worm spread is the severe congestion created by fast worms. The legitimate traffic suffers both because it loses competition for a limited bandwidth against more aggressive worm traffic, and because it responds to congestion by lowering its sending rate. A realistic worm spread simulation must reproduce both the realistic legitimate traffic patterns and volumes on each link, and the legitimate traffic's congestion response. Detailed congestion simulation further facilitates evaluation of the legitimate traffic's impairment due to a worm spread, and the detection of bottlenecks in the Internet topology.

It would be infeasible to simulate all the Internet traffic at the packet level. This simulation would require a huge amount of memory and CPU resources, resulting in a long execution time. On the other hand, even if the simulation at such granularity were practical, we lack details needed to replicate Internet communication patterns at the connection level, such as link delays, frequency and length distributions of various connections, and their packet rates, etc. For the simulation of a worm spread and other unwanted traffic incidents, we believe that it is sufficient to devise a model that accurately captures the volume of traffic at each link with and without the malicious event, and the congestion response of the TCP traffic when its packets are dropped. In PAWS we simulate the aggregate TCP traffic between each pair of ASes, taking into account the fact that this traffic is a mix of diverse connections, and simulating the connection diversity. We provide more details about the aggregate traffic simulation in the following sections.

4.1 Legitimate communication patterns

According to [28], the incoming and outgoing traffic volumes of the Internet ASes follow the $Weibull(r, \alpha, \beta)$ distribution, where r is the AS rank, according to traffic volume and α, β are the distribution's parameters. We use this traffic model in PAWS to determine the traffic offer and demand between AS pairs. We first sort ASes based on their IP size, and use this ranking as an input to the $Weibull(0.2, 1.5 \times 10^6)$ distribution for the incoming traffic generation, and the $Weibull(0.25, 1.0 \times 10^8)$ distribution for the outgoing traffic generation. The output of this process are two traffic arrays specifying the inflow and the outflow for each AS. We next transform these arrays into traffic matrices, that contain the sending rate between each pair of ASes.

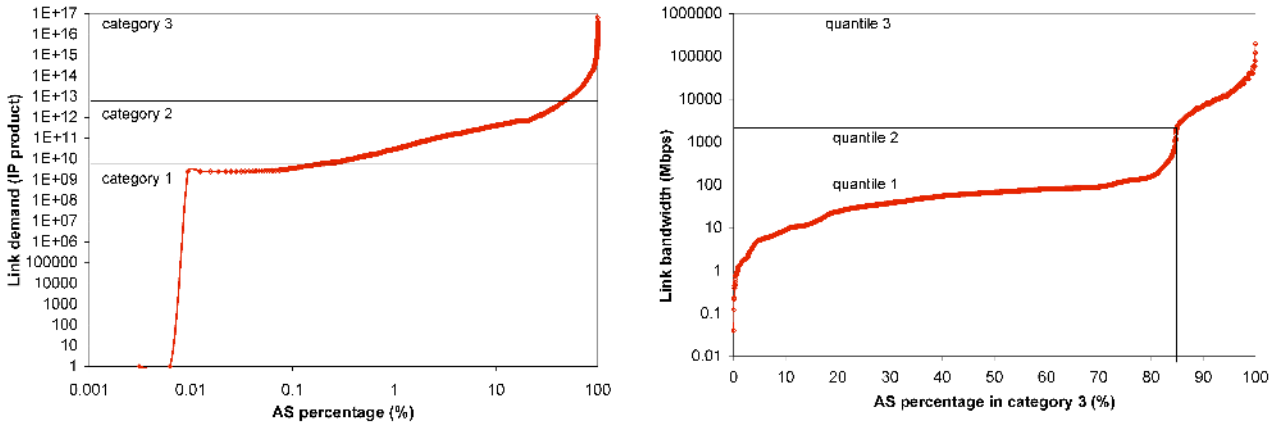
Because the $Weibull$ distribution is dominated by a few sources and destinations that account for majority of traffic flows, we can omit the simulation of small senders and receivers without jeopardizing fidelity. We only simulate traffic aggregates larger that 1 Mbps, which comprise 90% of all traffic generated by the $Weibull$ distribution. PAWS currently simulates static communication patterns, which means that the desired sending rate between a given pair of ASes is constant during the simulation, in the absence of congestion.

4.2 Simulating legitimate TCP traffic

Statistics from the Internet backbone consistently show that more than 90% of the Internet traffic uses the TCP protocol [30]. We simulate only the TCP portion of the background traffic in PAWS, and we reproduce two relevant features of TCP connections that interact with the worm spread: connection age/duration and the congestion response at the traffic aggregate level.

4.2.1 Simulating TCP connections with different ages

In case of congestion, the starting time of a TCP connection, its age and round-trip time have an impact on its congestion response, because they determine the amount of



a. Link product values distribution

b. Pathneck data distribution for link category 3

Figure 1: Grouping inter-AS links based on IP product values

Category	Traffic percentage (tp)	Connection age
c_0	15%	$\leq$3 seconds
c_1	3%	3-6 seconds
c_2	1%	6-9 seconds
c_3	3%	9-12 seconds
c_4	2%	12-15 seconds
c_5	76%	$\geq$15 seconds

Table 2: Distribution of connection ages

congestion memory that this connection will retain. Long-lived connections have a higher chance to be impaired by congestion, while short-lived ones may get "lucky" and complete before the congestion builds up. Connections that start before the congestion event achieve higher congestion window values, and are more competitive, than connections that start with the small congestion window, during the congestion event. If we periodically sample connection traffic in some mix, connections of *age* y (that have started y seconds ago) will retain congestion memory of y seconds. Another factor that influences a TCP connection's congestion response is the roundtrip time (RTT) which determines the speed of the congestion feedback. Instead of replicating a realistic RTT distribution in PAWS, we assume that each connection's retransmission timer starts at $RTO = 3$ s and that each connection learns of and responds to the congestion by the next 3s-interval. This is realistic, since the majority of RTT values observed in the Internet [31] fall well bellow 1 s.

Our first step in faithfully replicating TCP traffic at the aggregate level was to determine the distribution of connection age values in the Internet traffic, using public traffic traces collected by the MAWI project [30]. These traces contain 15-minute snapshots of traffic on the Internet backbone. We sample these connections each 3 seconds, and for each connection whose SYN packet appears within the trace, we measure the age as the time elapsed between this SYN packet and the sampling event. We then quantize the age space into six categories, shown in Table 2 and calculate the percentage of traffic carried by connections in each age category.

Table 2 shows the sample distribution of TCP connection age, derived from MAWI traffic trace for January 2003, just before the Slammer worm spread. We use this distribution for simulation of the Slammer worm spread in section 7.2.

Based on our observation of MAWI traffic traces of different dates, we find that the connection age distribution varies trivially from day to day, but it changes noticeably from year to year by more connections shifting into the long-age (c_5) category. We examined the connections in the long-age category and concluded that the majority are generated by HTTP and SMTP applications. Some connections are long because they transfer a lot of data, while others are long because they have a long RTT value.

4.2.2 *Simulating TCP's congestion response*

Using congestion control [32], a TCP sender adaptively adjusts its sending rate to transmit data reliably over an unreliable network with time-varying bandwidth. In PAWS, we simulate this adaptive rate mechanism not for each individual connection but for the total TCP traffic exchanged between a pair of source and destination ASes — the TCP aggregate. The congestion response will regulate the aggregate's sending rate, rather than the sending rate of each individual connection.

When congestion occurs, a TCP sender learns about this by detecting packet loss and reduces its sending window to slow down the sending rate. This procedure continues until either the congestion is mitigated or the sending window reaches its minimum size. We replicate this response at the aggregate level by dividing the traffic on a congested link into six categories based on the connection age, shown in the Table 2, and simulating the response of each category to the congestion event separately because of their different congestion memories. Observing a macroscopic TCP behavior, if the competing malicious traffic consumes a portion of the bottleneck link, the TCP traffic volume will be gradually adjusted, over several RTT intervals, by the congestion control mechanism to fill the remaining bandwidth. When the congestion is severe, all long-lived TCP connections will, after a brief period, reduce their sending rate to the minimum. Due to the newly arriving connections, which comprise the significant portion of the Internet's traffic as shown in the

Table 2, the legitimate TCP traffic will never disappear from the links, even if the worm scans create heavy congestion. This is because new connections have no "congestion awareness." If established, they will send aggressively, using the slow-start mechanism, until they learn about the first congestion drop 3 s later.

Initially the traffic volume between each AS pair is determined using the approach described in section 4.1. This traffic volume, TV, is reported to all the links along the routing path and recorded at each link. During the simulation, if congestion occurs on a link at time i, this link will report the congestion event and its current pass ratio to all the source ASes whose traffic it carries. The source takes the smallest pass ratio reported from all the links along the routing path to some destination and uses it as its pass ratio p_i at time i for the traffic to this destination. p_i is used along with the connection's congestion memory, to adjust the source's sending rate for the next iteration.

During each simulation iteration at time i, the source AS originates some new connection traffic N_i, equal to $TV \cdot tp_0$, where tp_0 is the percentage of traffic in the new-connection category c_0. This traffic has no congestion memory and thus is not affected by the current pass ratio p_i but the connection setup depends on the congestion situation in the several previous simulation intervals, as shown in Equation (1).

$$N_i = TV \cdot tp_0 \cdot \prod_{j \in i-15s} p_j \tag{1}$$

Traffic belonging to old connections (categories c_1—c_5) is affected by the current and some previous pass ratios. For each specific category j of the old-connection traffic at time i, the traffic volume $O_{j,i}$ is computed based on its congestion memory, which depends on its age.

$$O_{j,i} = TV \cdot tp_j \cdot \prod_{k \in i-age_j} p_k, \qquad 1 \le j \le 5 \tag{2}$$

where tp_j is the percentage of traffic in the category c_j and $\prod_{k \in i-age_j} p_k$ is the product of pass ratios reported during the age of the connections in this category.

During the congestion, the traffic that fails to transmit gets accumulated in the sender's buffer and will be retransmitted after the congestion ends. We keep track of this traffic in the variable C_i for each time i and for each source, and we assume that a sender has an infinite buffer. After the congestion ends, the TCP sender retransmits the accumulated traffic and sends the newly generated traffic. This results in a sending rate higher than TV for an extended period of time. The variable W_i in our simulation expresses the retransmitted traffic volume at time i. W_i, as is shown in Equation (3), increases linearly when there is no congestion and decreases exponentially when congestion happens.

$$W_{i+1} = \begin{cases} 10,000 & i = 0 \\ W_i + 10,000 & \text{no congestion in } i^{th} \text{ interval} \\ W_i/2 & \text{congestion in } i^{th} \text{ interval} \end{cases} \tag{3}$$

The retransmission rate R_i at time i is calculated as the minimum of retransmitted traffic W_i and the accumulated dropped traffic C_i at that time: $R_i = min(W_i, C_i)$.

The total sending rate T_i at time i between an AS pair is the summation of the traffic sent from the new connections N_i, the traffic sent from the old connections $O_{j,i}$,$j = 1..5$ and the retransmitted traffic R_i: $T_i = N_i + \sum_{j=1..5} O_{j,i} + R_i$.

While it may seem that our TCP aggregate congestion model assumes that TCP connections will receive congestion notifications through timeouts, rather than through triple-duplicate acknowledgments, we do not make such assumption. Regardless of the congestion notification, highly multiplexed TCP traffic will respond to congestion by first reducing its aggregate sending rate to some level, and then increasing it gradually to fill the remaining bandwidth. In face of long-lasting congestion, such as occurs during worm spread events, sending rate of long-lived connections will reach the minimum after several RTT intervals, regardless of the congestion notification approach, because congestion drops occur in each interval. Thus, at the macroscopic level, aggregate TCP traffic scales down to volume of newly established connections in presence of long-lasting congestion, and PAWS captures this effect. Our simplified aggregate congestion response model would not be suitable for simulation of short-lived congestion events.

To validate our TCP aggregate model, we simulate several congestion scenarios in PAWS, and compare the results:

1. With the same scenarios replicated in the ns-2 [5] simulator, which models individual TCP connections and performs packet-level simulation, and

2. With live traffic experiments reenacting the same scenarios in the Emulab [14] testbed.

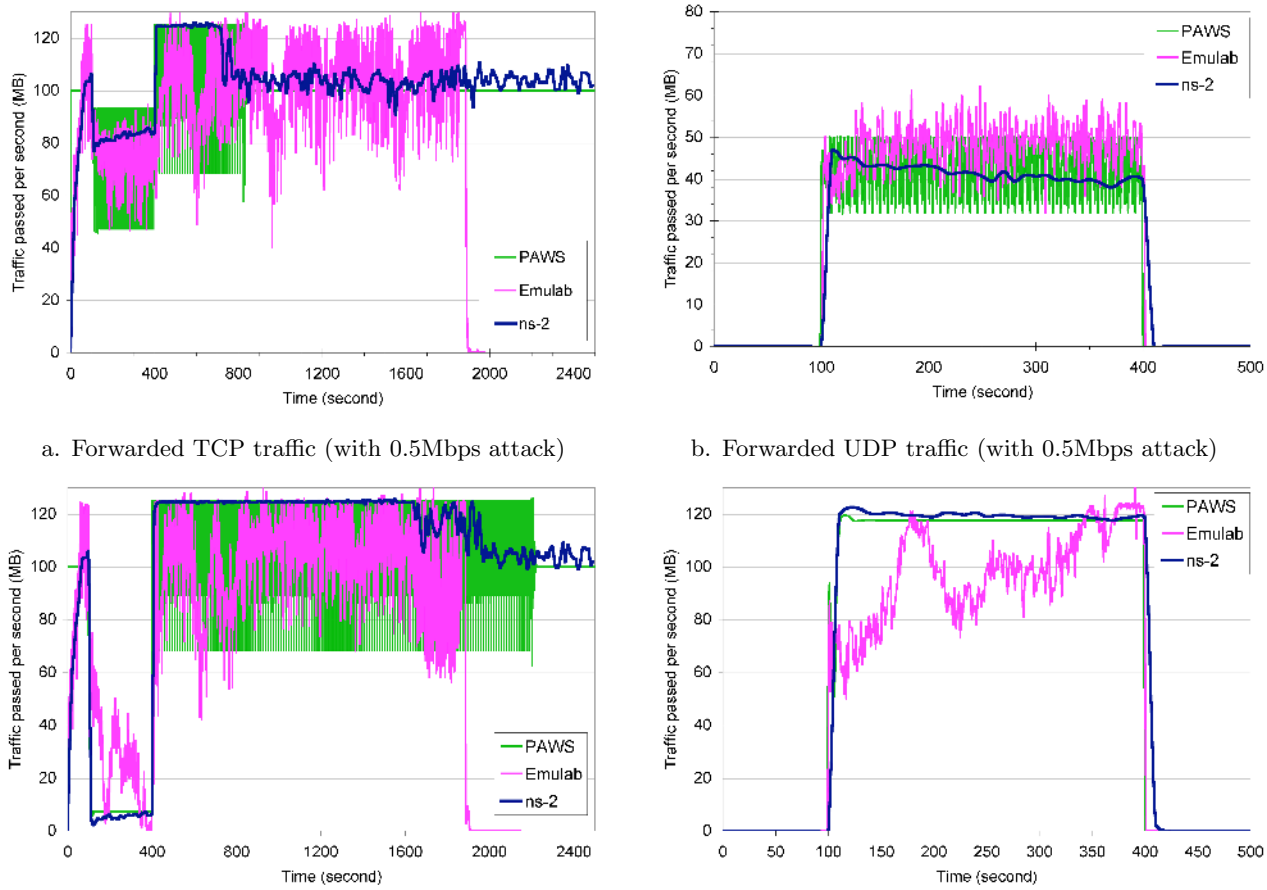
For all the three experiments (PAWS, ns-2, Emulab), we use the same topology shown in the figure 3. The bottleneck bandwidth is 1 Mbps and the legitimate TCP traffic consumes 0.8 Mbps, i.e. the link utilization is 80% in absence of the attack. The legitimate traffic is a mix of many TCP connections, whose traffic and duration follow the distribution from Table 2. During the experiment, we generate UDP traffic between the UDP sender and the receiver, at a high enough rate to create congestion on the bottleneck link. We then compare the TCP congestion response between the three tested systems. For brevity, we show here the results of two traffic scenarios: one with a medium level of congestion, with the UDP traffic rate of 0.5 Mbps, and the other with a heavy congestion, and the UDP traffic rate of 1.5 Mbps. The UDP traffic lasts for 300 seconds, and we run the experiment for 2000 seconds to track the TCP recovery. Figure 2 shows the traffic at the bottleneck link for all three test settings. The simulation result generated by PAWS matches both the ns-2 and the Emulab experiments very well, for the three distinct simulation periods: before, during and after the congestion event. While the topology and settings used in these simulations are very simple, they illustrate a representative case of the UDP worm propagation, which competes for bandwidth with highly-multiplexed, legitimate TCP traffic.

5. SIMULATION OF WORM PROPAGATION

We describe the models of relevant worm features in this Section.

5.1 Vulnerable host model

A vulnerable host has a set of hardware and software features that characterize its vulnerability and usefulness to



a. Forwarded TCP traffic (with 0.5Mbps attack)

b. Forwarded UDP traffic (with 0.5Mbps attack)

c. Forwarded TCP traffic (with 1.5Mbps attack)

d. Forwarded UDP traffic (with 1.5Mbps attack)

Figure 2: Simulation of the TCP traffic aggregate's rate in presence of a mild and a strong congestion

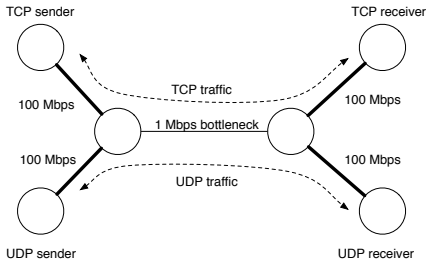


Figure 3: Topology used for the aggregate traffic model validation

the attacker. This diversity and individual actions of each host have to be reproduced for a high-fidelity worm simulation. We use an IP address to identify each vulnerable host in PAWS, and associate a user-customizable data structure with each host, to record its features. The relevant features affect worm propagation, such as the type of OS, the applications running on this host, the amount of critical resources (CPU frequency, memory, network connection speed, etc.), the presence of vulnerabilities that can be exploited by worms, etc. We also record a state of each host, which includes its infection status (vulnerable, infected, scanning, patched, etc.) and the infection statistics (e.g. how many scans it has sent and how many new hosts have been infected by this host).

5.1.1 *Vulnerable host distribution*

Many existing worm models and simulations make the simplifying assumption that vulnerable population is uniformly distributed over the IP space. Still, the collected data of high-profile worm spreads, such as Code Red and Slammer, has indicated that the vulnerable hosts have a highly non-uniform distribution [33], [34]. PAWS supports both the uniform and the non-uniform vulnerable host distribution models, by defining a vulnerable ratio for each AS or subnet and initializing it according to a distribution chosen from the following options:

1. *Uniform:* Each AS has the same vulnerable ratio.
2. *Log-normal:* Rajab et al. have found that the vulnerable host distribution in the Internet fits a log-normal distribution with different parameters for different vulnerabilities [36]. PAWS simulates this distribution by drawing the vulnerable ratios of different ASes/subnets from a log-normal distribution.
3. *Other non-uniform distributions:* PAWS can apply vulnerable ratios at different granularities, i.e., at an AS,

a BGP atom or an IP range level. If a priori knowledge exists of the distribution of vulnerable hosts for a specific worm, PAWS supports loading this information for each entity from a properly formatted file. This facilitates use of PAWS for simulation of any empirical distribution of vulnerable hosts, e.g., [37].

5.2 Worm model

A worm model defines worm scanning and infection strategy, including the delay between a vulnerable host scan and the infection completion.

TCP worms scan vulnerable hosts by initiating a TCP connection with each scanned address, via a TCP-SYN packet sent by a worm thread. Since TCP-SYN packets are small (40 B), a TCP worm seldom consumes much bandwidth and does not cause serious congestion. If a destination does not respond to a SYN packet, it will be retransmitted after a timeout of *RTO* seconds, where *RTO* value differs between TCP connections. After three unsuccessful retransmissions, each occurring after a multiplicatively increased *RTO* interval, a sender will usually give up its connection attempt. PAWS simulates this mechanism by buffering the destination address of each worm scan that is not delivered to a vulnerable destination, and maintaining a timer for this scan, initialized to a default *RTO* value of 3 s. The scans are resent after the timer expires, up to 3 times, and the timer is doubled after each unsuccessful attempt. The scanning rate of each host depends on the number of threads that can be simultaneously created by this host, which is defined by a host's OS type and version. This effect can be simulated by defining a host's OS type and version, in the set of relevant host features, and drawing the values for these parameters randomly from some measured or assumed real-world distribution.

UDP worms send their payload and the scan usually bundled in a single probe, and do not retransmit failed scans. The limiting factor of the worm scanning speed is the host's access link bandwidth and the available bandwidth en route to the destination.

5.3 Worm scan model

Large worm spread events observed in the Internet have used various scanning strategies. In addition to this, many researchers have proposed hypothetical scanning strategies that would result in much faster than observed worm propagation [1],[38],[39],[37].

5.3.1 Scanning strategies

PAWS simulates uniform scanning by having each simulated host randomly generate a 4-byte address for each scan. IP address can also be generated based on the host's address or according to some specified distribution, which facilitates simulation of subnet scanning worms, like Code Red II [33]. PAWS' modular design allows for an easy extension to integrate new scanning strategies such as hitlist scanning, routable address scanning, etc.

5.3.2 Worm scan simulation

The simulation of worm spread on PAWS is a time-discrete procedure. During every time unit, currently set to 1 s, each infected host generates a list of target addresses and sends scans to them. Each worm scan is time-stamped and buffered as an event. At the end of the time unit, all the events are carried out simultaneously and all vulnerable hosts that are scanned become infected. To enhance simulation speed, only scans to vulnerable but not yet infected hosts are delivered to the destinations. Scans to invulnerable or infected hosts are not sent, but their effects (e.g. consumption of host resources and link bandwidth) are accounted for.

6. PAWS IMPLEMENTATION ON EMULAB

PAWS can run on any group of networked PCs. To make PAWS easily accessible to other researchers, we implement it on the shared Emulab testbed [14]. While the PAWS simulation code can run on any machine with a Standard C compiler, we customized our setup scripts and data structures for the Emulab environment. To maximize simulation performance, we currently use Emulab machines with the highest CPU rate with 3GHz, 64-bit Xeon processors, 800MHz FSB and 2GB, 400 MHz RAM. One simulation node is designated as a master, which synchronizes the simulation and controls the slave nodes that carry out the simulation tasks. A 100Mbps LAN is used to connect all the simulation machines, with the latency and the loss rate both set to zero.

6.1 Sharing the simulation load

To balance the workload on the simulation nodes and minimize inter-machine communication, we apply a heuristic to distribute the AS-level Internet topology among multiple machines with two goals: (1) The number of the vulnerable hosts simulated by each node is roughly the same. This requirement balances the processing load of each machine. (2) Each machine is assigned a connected portion of the Internet AS map. This requirement minimizes the inter-machine communication as many worm packets and legitimate flows can be simulated with internal function calls. A simulator node only simulates the propagation of worm traffic within its own portion of the Internet, and then delivers partially processed traffic to the simulation node responsible for its next hop, using network communication at the end of each simulation interval. We also distribute the forwarding tables across the simulation machines, so that each machine stores only the tables for the ASes it simulates and thus minimizes its memory requirements.

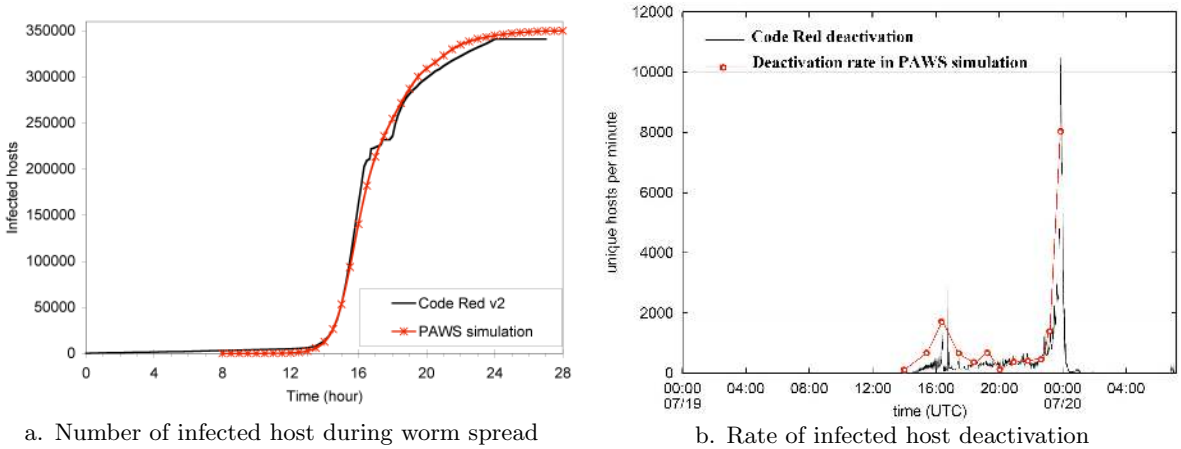
6.2 Inter-node communication

PAWS simulator nodes communicate with each other in two ways: (1) We use the shared NFS directory in the Emulab, to initialize each simulation node with the worm specification, and its portion of the Internet topology and forwarding data. (2) Simulator nodes use stream sockets to exchange simulated traffic at the end of each time unit, and to report the worm propagation status. The sockets are set up between each pair of the simulation nodes, during the initialization stage.

6.3 Time unit scaling

A significant portion of the simulation time is used for the inter-machine communication, to synchronize machines at the end of each time unit. To reduce this overhead, PAWS deploys a dynamic communication interval, whose value is adjusted based on the dynamics of the simulated event. We call this approach "time unit scaling."

In case of a worm spread, communication interval is set to a reasonably high value (20 seconds) at the early and the



a. Number of infected host during worm spread

b. Rate of infected host deactivation

Figure 4: Simulation of Code Red v2

final stages of the worm propagation, when the successful infections are a few and the loss of fidelity is small if we skip a few synchronization points. The interval is decreased when the infection rate increases. The Equation (4) shows how PAWS calculates the current communication interval C_i at time i, where N is the size of the vulnerable population, M_{i-1} is the number of newly infected hosts during the previous time interval $i-1$, and s is a scaling factor (currently set at 1000).

$$C_i = \left\lceil \frac{N}{s \cdot M_{i-1}} \right\rceil, 1 \leq C_i \leq 20 \quad (4)$$

7. SIMULATION OF WORM EVENTS

To validate PAWS simulation of worm spread events, we simulate two well-known events: Code Red v2 on July 19, 2001 [33] and SQL Slammer on January 25, 2003 [34].

7.1 Simulation of Code Red v2

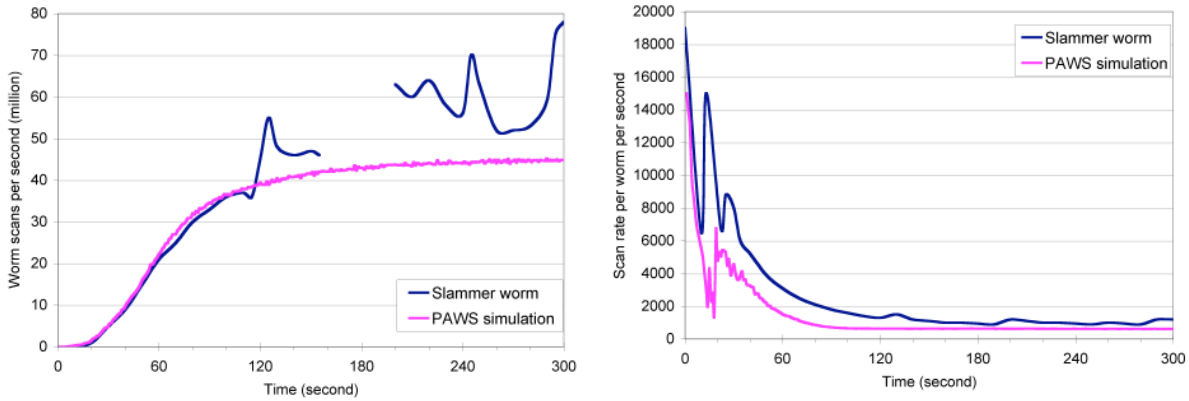
During the outbreak of Code Red v2 worm, more than 359,000 infected hosts were observed by CAIDA [33]. In the simulation, we set the vulnerable population $N = 360,000$. The distribution of vulnerable hosts follows $log{-}normal(6, 4)$ distribution among ASes. Each infected host tries to infect a different list of randomly generated IP addresses at a peak rate of 11 probes per second. Initially, there are 2 infected hosts when simulation starts. We reproduce AS-level Internet topology from July 19, 2001 at 10 am in this simulation. To faithfully replicate the worm propagation event, we also consider the deactivation of the infected hosts, reported in [33]. Such deactivation happens when a previously infected host stops sending out scans, possibly because it was patched or disconnected from the network. Deactivation starts after the 14th hour of worm outbreak. Figure 4(a) shows our simulation results compared with the CAIDA's observation of the real worm propagation in the Internet [33]. We retrieve the real worm data from the CAIDA's animation of the geographic spread of Code Red, in five minute intervals [43]. Worm behavior in PAWS simulation exactly matches the Code Red v2 spread in the real Internet. In Figure 4(b) we show the deactivation rate used by PAWS (red circles) compared to the observed deactivation rate reported in [33] (lines).

7.2 Simulation of SQL Slammer

SQL Slammer [34] propagated on January 25, 2003, starting from 5:30 am UTC and infected more than 90% of the vulnerable hosts within 10 minutes. Totally at least 75,000 hosts were infected and a large portion of the Internet was congested by worm probes.

In this experiment, we simulate the Slammer spread with a vulnerable population of 75,000 hosts. These vulnerable hosts are distributed among ASes using the $log{-}normal(6, 4)$ distribution. Slammer worm probes were carried by the User Diagram Protocol (UDP), which facilitated its high scanning speed. The largest scanning rate of an infected host is restricted at 26,000 scans per second, as observed by Moore et al. [34], and is further limited by the limited bandwidths of the inter-AS links and the hosts' access links. We initialize host access link bandwidths using the Annual Bandwidth Report for 2003 [29]. According to the worm code analysis [34], there are two flaws in Slammer's random-number generator, which reduce the quality of scan distribution because all the scanned addresses from any worm instance share the same two bits. In PAWS experiment, we simulate this worm flaw by restricting each worm's scan space to 2^{30} addresses, instead of 2^{32} addressees.

Figure 5 shows our simulation result, compared with the Slammer's propagation as observed at the University of Wisconsin's tarpit network [44] and scaled to the whole Internet. Since this measurement only captured successfully received scans, we compare its results with the successfully received scans in the PAWS simulation, in the Figure 5(a). Our simulation matches well the total number of scans observed during the propagation. We also show the observed and simulated per-worm scanning rate in the Figure 5(b). The scanning rate per worm first increases sharply, and then falls slowly to a flat horizontal line around 1,000 scans per second. This can be interpreted as follows: Initially there are only a few infected hosts that send out scans as fast as their Internet connections allow. This leads to the high initial per-worm scanning rate, especially if the early infected hosts have a high-speed Internet connection. As spread continues and the congestion builds up, many worm scans get dropped while the number of infected hosts increases. Jointly, this leads to a slow decline of the per-worm scanning rate. This effect cannot be observed using mathematical worm spread



a. Total successful scans in the Internet

b. Slammer's per-worm scanning rate

Figure 5: Simulation of SQL Slammer propagation

models or using simulations that do not replicate limited Internet link bandwidths. The simulated per-worm scanning rate matches the dynamics of the observed rate, but the peak heights are lower than the observed ones. We attribute this to the real worm compromising several well-connected hosts in its early propagation, that account for the observed scanning rate peaks. Since we randomly assign host access link speeds from a uniform distribution, we do not observe this event in the simulation.

7.3 Realistic model's influence on worm simulation fidelity

PAWS strives to replicate many realistic Internet features to increase the fidelity of the worm spread simulation. In this section we demonstrate how a failure to simulate these features compromises simulation fidelity. We compare the results of the PAWS Slammer simulation, from the previous section, with a PAWS simulation that applies one of the following assumptions and simplifications that are widely used in worm research:

1. *Infinite bandwidth of Internet links:* there is no limit on the inter-AS link capacity and no congestion occurs. Each infected host can send out scans at the maximum rate of 4,000 per second. This was the average observed scanning rate during the real event [33].

2. *Infinite bandwidth of host access links:* each infected host scans at 4,000 scans per second.

3. *Uniform distribution of vulnerable hosts:* the vulnerable hosts are evenly distributed in the IP address space, they scan at 26,000 scans per second.

4. *No simulation of the legitimate traffic:* the background traffic and its interaction with the worm spread is not simulated, but the worm scans are dropped when they exhaust limited access and Internet link bandwidth. Infected hosts scan at 26,000 scans per second.

We present the results in Figure 6. The infinite link bandwidth assumption leads to a very fast spread, almost five times faster than observed, even though we use more than six times lower scanning rate. The infinite access link bandwidth results in a slower spread; this is the effect both of the lower-than-observed scanning rate and the congestion build-up at the inter-AS links. The uniform distribution of vulnerable hosts speeds up the spread almost twice, as compared to the non-uniform distribution and using the same scanning rate. This speedup occurs because each scan has a higher chance of leading to a successful infection. The legitimate traffic simulation only slows down the worm propagation in the early stages, when the volume of the legitimate traffic is comparable to the worm scan traffic and it can fairly compete for the limited bandwidth. While the worm spread dynamics is not significantly affected by the legitimate traffic simulation, we believe that this simulation is important to understand the damage inflicted by the worm spread to the Internet users.

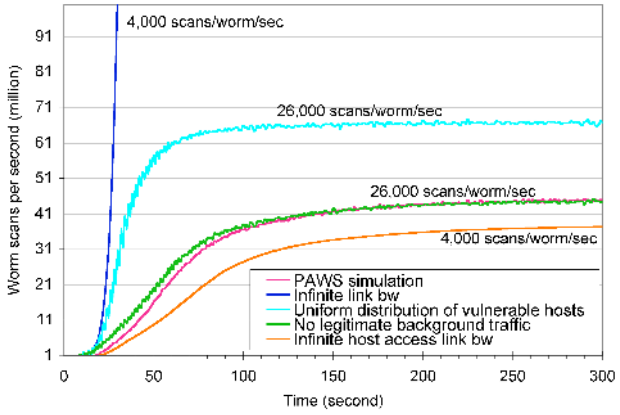


Figure 6: Simulation of the Slammer spread with different simplifications

8. PAWS PERFORMANCE

Two major PAWS's activities during worm simulation are the processing of the worm scans and the simulation synchronization via network communication. *Worm scan processing* encompasses the operation of sending a worm scan from each infected host, traversing the routing path of this scan hop by hop, and receiving this scan on the target host. This is the major task of the PAWS simulator and it consumes most of the CPU time. In the simulations with a

higher number of vulnerable hosts or higher scanning rate, PAWS spends more time on scan processing. *Simulation synchronization* includes the transfer of worm scans and worm spread reports among simulation nodes, and introduces synchronization delay. As there are more scans in the simulation, this communication overhead grows. Since PAWS uses distributed simulation we can reduce the overall processing time by engaging more simulation nodes and thus lowering the simulation demand on each CPU, but this increases synchronization overhead. Time unit scaling, described in Section 6.3 can help us amortize the effect of the increased synchronization overhead, while benefiting from a faster simulation when we engage more simulation nodes.

We illustrate the effect of the simulation distribution on the execution speed, by running the Slammer spread simulation using different numbers of simulation nodes. For each experiment, we measure execution time (with time unit scaling) and the average PTS (simulated packet transmissions per second) for the first 5 minutes of worm spread and present them in the Figure 7. We see that the simulation time declines as more machines are used for simulation. This decline is fast at first, as the processing load is being shared, but then it slows down because the communication overhead increases. The average PTS grows almost linearly with the simulation distribution. We note that our simulation speed of 5M PTS with four common PCs is comparable to the reported 5.5M PTS achievable by PDNS and GTNetS on a dedicated 136-CPU cluster [24]. We achieve comparable performance with much fewer machines because we aggregate scan transmissions in one time unit and simulate them together, unlike PDNS and GTNetS that simulate each scan separately.

Finally, we investigate the effect of the time unit scaling, described in the Section 6.3 on simulation fidelity. In the Figure 8 we depict the number of infected hosts during the Slammer worm propagation, with and without time unit scaling. We observe that the dynamics curves look very similar and that the curve with scaling is slightly faster than the curve without scaling, because it misses some synchronization events and overestimates the speed of the worm infection. This difference, however, is very small and does not affect the overall spread dynamics, while the simulation with scaling is more than four times faster than that without scaling.

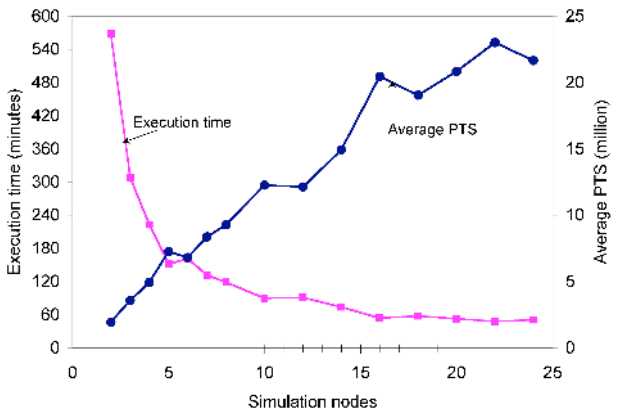


Figure 7: PAWS performance with different number of simulation nodes

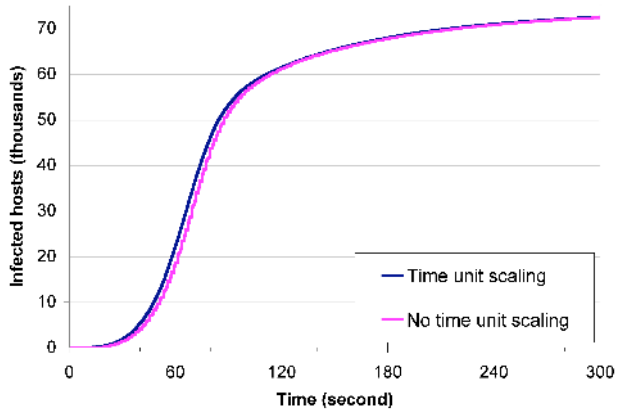


Figure 8: Number of infected hosts with and without time unit scaling

9. CONCLUSIONS

We present the design and implementation of a distributed worm simulator, PAWS, that replicates a realistic Internet environment and its interaction with a simulated worm. This detailed and realistic Internet simulation leads to high-fidelity reconstruction of the worm spread events. By using PAWS, researchers can investigate congestion effects of Internet worm spread and its interactions with the background traffic. PAWS further supports various user-customizable parameters that can be specified for each simulated host, which facilitates testing of different host and network diversity models, worm scanning strategies and Internet topologies. Our simulation results of the propagation of Code Red and Slammer validate the correctness of PAWS, and support our claim for faithful simulation of complex Internet-scale events.

In our future work, we plan to extend PAWS for simulation of other Internet-scale events, like distributed denial-of-service (DDoS) attacks, flash-crowd events, botnet recruitment and organization, spam, etc. We also plan to improve our Internet model with more realistic data and dynamics, such as the simulation of the routing dynamics and routing interaction with the congestion events, and the simulation of the time-variable traffic demand and offer between the AS pairs.

10. ACKNOWLEDGMENTS

The authors would like to thank the members of the Pathneck project for sharing their data with us, and the Emulab staff for accommodating our needs for lengthy simulations involving many of their machines. This project was supported by a grant from the University of Delaware Research Foundation and we are grateful for this support. We also thank anonymous reviewers for their helpful feedback on our work.

11. REFERENCES

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

[2] C. Zou, W. Gong and D. Towsley, Code Red Worm Propagation Modeling and Analysis, *ACM CCS 2002*.

[3] Z. Chen, L. Gao and K. Kwiat, MOdeling the Spread of Active Worms, *IEEE INFOCOM 2003*.

[4] N. Weaver, I. Hamadeh, G. Kesidis and V. Paxson, Preliminary Results Using ScaleDown to Explore Worm Dynamics, *ACM CCS WORM 2004*.

[5] The Network Simulator – ns-2, `http://www.isi.edu/nsnam/ns/`.

[6] Scalable Network Technologies (SNT), QualNet, `http://www.qualnet.com`.

[7] Parallel/Distributed NS, `http://www-static.cc.gatech.edu/computing/compass/pdns/`.

[8] Georgia Tech Network Simulator (GTNetS), `http://www.ece.gatech.edu/research/labs/MANIACS/GTNetS/`.

[9] Scalable Simulation Framework, `http://www.ssfnet.org`.

[10] S. Floyd and V. Paxson. Difficulties in Simulating the Internet, *IEEE/ACM Transactions on Networking*, Vol.9, No.4, pp.392-403, August, 2001.

[11] S. Floyd and V. Jacobson, Random early detection gateways for congestion avoidance. *IEEE/ACM Transactions on Networking (TON)*, Vol. 1 , No. 4, pp 397 - 413, August 1993.

[12] University of Oregon Route Views Project, `http://www.routeviews.org`.

[13] N. Hu, L. Li, Z. Mao, P. Steenkiste and J. Wang, Locating Internet Bottlenecks: Algorithms, Measurements, and Implications, *SIGCOMM 2004*.

[14] Emulab – Network Emulation Testbed, `http://www.emulab.net`.

[15] D. Nicol, J. Liu, M. Liljenstam and G. Yan, Simulation of Large-scale Networks Using SSF, *the 2003 Winter Simulation Conference (WSC'03)*,

[16] D. Nicol and G. Yan, Simulation of Network Traffic at Coarse Time-Scales, *PADS 2005*.

[17] D. Nicol and G. Yan, Discrete-event Fluid Modeling of TCP Background Traffic, *ACM Transactions on Modeling and Computer Simulation*, Vol. 14, No. 3, July 2004.

[18] D. Moore, C. Shannon, G. M. Voelker and S. Savage, Internet Quarantine: Requirements for Containing Self-Propagating Code, *IEEE INFOCOM 2003*.

[19] A. Wagner, T. Dubendorfer, B. Plattner adn R. Hiestand, Experiences with Worm Propagation Simulations, *WORM 2003*.

[20] M. Liljenstam, Y. Yuan and B. J. Premore, A Mixed Abstraction Level Simulation Model of Large-scale Internet Worm Infestations, *IEEE/ACM MASCOTS 2002*.

[21] M. Liljenstam, D. Nicol, V. Berk and R. Gray, Simulating Realistic Network Worm Traffic for Worm Warning System Design and Testing, *WORM 2003*.

[22] G. F. Riley, M. I. Sharif and W. Lee, Simulating Internet Worms, *IEEE/ACM MASCOTS 2004*.

[23] K. S. Perumalla and S. Sundaragopalan, High-fidelity Modeling of Computer Network Worms, *ACSAC 2004*.

[24] R. Fujimoto, K. Perumalla, A. Park, H. Wu, M. Ammar and G. Riley, Large-Scale Network Simulation — How Big? How Fast?, *IEEE/ACM MASCOTS 2003*.

[25] H. Chang, R. Govindan, S. Jamin, S. Shenker and W. Willinger, Towards Capturing Representative AS-Level Internet Topologies, *Computer Networks Journal*, Vol.44, No.6, pp. 737—755, Elsevier Publisher, April 2004.

[26] A. Broido and K. Claffy, Analysis of RouteViews BGP Data: Policy Atoms. *ACM SIGMOD/PODS Workshop on Network-related Data Management* 2001.

[27] Network Connection Speeds Reference, `http://www.ertyu.org/steven\_nikkel/netspeeds.html`.

[28] A. Broido, Y. Hyun, R. Gao and kc claffy, Their Share: Diversity and Disparity in IP Traffic, *PAM 2004*.

[29] The Bandwidth Report, `http://www.websiteoptimization.com/bw/`.

[30] MAWI Working Group Traffic Archive, `http://tracer.csl.sony.co.jp/mawi/`.

[31] S. Shakkottai and R. Srikant and N. Brownlee and A. Broido and kc claffy, The RTT Distribution of TCP Flows in the Internet and its Impact on TCP-based Flow Control, *CAIDA Technical Report, TR-2004-02*.

[32] M. Allman, V. Paxson and W. Stevens, TCP Congestion Control, *RFC 2581*.

[33] D. Moore, C. Shannon and J. Brown, Code-Red: a Case Study on the Spread and Victims of an Internet Worm, *ACM SIGCOMM/USENIX Internet Measurement Workshop*, 2002.

[34] D. Moore, V. Paxson, S. Savage, C. Shannon, S. Staniford and N. Weaver, The Spread of the Sapphire/Slammer Worm, *IEEE Security and Privacy*, Vol. 1, No. 4, pp. 33—39, July, 2003.

[35] Distributed Intrusion Detection System (DShield), `http://www.dshield.org/`.

[36] M. Rajab, F. Monrose and A. Terzis, On the Effectiveness of Distributed Worm Monitoring, *14th Usenix Security Symposium*, 2005.

[37] Z. Chen and C. Ji, Importance-scanning Worm using Vulnerable-host Distribution, *IEEE GLOBECOM 2005*.

[38] C. Zou, D. Towsley and W. Gong, Email Worm Modeling and Defense, *ICCCN'04*.

[39] C. Zou, D. Towsley, W. Gong and S. Cai, Routing Worm: a Fast, Selective Attack Worm based on IP Address Information, *PADS 2005*.

[40] N. Sprint, R. Mahajan and D. Wetherall, Measuring ISP Topology with Rocketfuel, *SIGCOMM 2002*.

[41] H. Chang, S. Jamin, Z. Mao and W. Willinger, An Empirical Approach to Modeling Inter-AS Traffic Matrices. *ACM Internet Measurement Conference 2005*.

[42] Transmission Control Protocol. *RFC 793*.

[43] Animation of the geographic spread of Code Red version 2, `http://www.caida.org/analysis/security/code-red/\#animations`.

[44] Wisconsin Advanced Internet Laboratory, `http://wail.cs.wisc.edu/`.