
Research on Worm Attack Prevention using Content-Based Distributed Packet Filtering, its Implementation and Benchmarking

ETRI Project: Final Report

Project Period: Nov. 15, 2002 – Dec. 15, 2005

Principal Investigator: Kihong Park

ETRI Collaborator: Ikkyun Kim

Research Assistants: Bhagya Bethala, Hyojeong Kim

Network Systems Lab
Department of Computer Sciences
Purdue University
West Lafayette, IN 47907, U.S.A.

Tel.: +1-765-494-7821

Fax.: +1-765-494-0739

E-mail: park@cs.purdue.edu

<http://www.cs.purdue.edu/~park>

<http://www.cs.purdue.edu/nsf>

Contents

1	Summary	1
1.1	Gigabit Network Processor Worm Filtering	1
1.2	Inter-domain and Intra-domain Distributed Worm Filtering	1
1.3	Zero-day Worm Attack Defense	2
2	Gigabit Network Processor Worm Filtering	2
2.1	Motivation and Background	2
2.1.1	A case for transit worm filtering	2
2.1.2	Worms and viruses are not created equal	3
2.1.3	Scope of transit worm filtering	4
2.2	New Contribution	4
3	Scalable Worm Filter Architecture	5
3.1	Network Processor Platform	5
3.1.1	System requirement	5
3.1.2	IXP1200 network processor	5
3.1.3	Incremental deployment and usage	6
3.2	Architectural Features: (f1), (f2), and (f3)	6
3.2.1	Workload sensitivity	6
3.2.2	Worm specificity	7
3.2.3	Multi-level caching	10
4	Performance Evaluation	10
4.1	Experimental Testbed	10
4.1.1	Physical set-up	10
4.1.2	Workload generation	10
4.1.3	Basic worm benchmark suite	11
4.2	End-to-end Worm Filter Performance	11
4.3	Filter Latency	12
4.4	Critical Path Analysis	13
4.4.1	Component Type	14
4.4.2	Packet hold time for traffic type	16
4.5	Trace-based Benchmarking: Active Internet Honeypot	16
4.6	Polymorphic Stress Test: Artificial Workload Generation	19
4.7	Real-time Adaptivity and Extensibility: Zero-Day Attack	20
5	Inter-domain and Intra-domain Distributed Worm Filtering	21
5.1	Motivation	21
5.2	New Contribution	22

6	Distributed Worm Filtering	23
6.1	2-level Distributed Filter Hierarchy	23
6.2	Strategic Filter Deployment	24
7	Inter-domain Internet	24
7.1	Definitions and Set-up	24
7.2	Containment: Critical Filter Density	25
7.3	Robustness	27
7.4	Load-based Filter Placement	28
7.4.1	Containment	28
7.4.2	Suburbia, Backdoors, and Placement	29
8	Intra-domain ISP Networks	31
8.1	Local and Global Protection	31
8.2	Containment	32
8.2.1	Transit ISP Topology: Rocketfuel	32
8.2.2	Transit ISP Topology: Skitter	34
9	Zero-day Worm Attack Defense Architecture	34
9.1	Cooperative Filtering: Overall Architecture	34
9.2	Detection and Signature Generation Subsystem	35
9.2.1	Detection Subsystem	35
9.2.2	Signature Generation Subsystem	37
9.3	Worm Filtering Subsystem	38
10	Optimal New Worm Attack Detection	39
10.1	Problem Definition	39
10.2	Optimal Filter Placement: Distributed Worm Filtering	39
10.3	Optimal Detector Placement: Distributed New Worm Detection	39
10.4	Optimal Joint Detector and Filter Placement	41
10.5	Effective Detector Placement	41
10.5.1	Optimal Single-source Attack Detection and Destination Address Dispersion	41
10.5.2	Influence of Source Address Dispersion	44
10.5.3	Multiple-source Attack Detection: Finding Small Detector Sets	46
11	Reactive Zero-day Attack Containment	47
11.1	Scanning, Detection, and Infection Damage	47
11.1.1	Scanning: Epidemiology and Sampling	47
11.1.2	Sampling and Detection	48
11.1.3	Sampling and Infection Damage	48
11.1.4	Impact of Network Delay and Bandwidth	51
11.1.5	Source Address Dispersion Detection: “Less Is More Paradox”	57
11.2	Large-scale Network Simulation Environment	58

11.2.1	Hardware and Software Set-up	58
11.2.2	Dynamic Network Simulation: Parameter Settings	58
11.3	Content Prevalence and Destination Address Dispersion Detection	58
11.3.1	Scale-invariant Containment: Scan Rate	59
11.3.2	Scale-invariant Containment: Delay and Bandwidth	61
11.3.3	Content Prevalence and Destination Address Dispersion Thresholds	61
11.3.4	Scan Type: Random, Local, and Hybrid	62
11.3.5	Number of Initial Attackers	63
11.3.6	Degree-based IP Address to AS Assignment	64
11.4	Dynamics under Source Address Dispersion	65
12	Related Work	66
13	Conclusion	68
14	Bibliography	69

1 Summary

The first half of the project period in 2005 has focused on completing the IXP1200 network processor worm filter design, implementation, and benchmarking, and completing performance evaluation of distributed worm filtering in both inter-domain and intra-domain Internet measurement topologies. The second half focuses on scalable protection against zero-day attacks, a critical threat facing the Internet today. The final report describes the main elements of these three parts.

1.1 Gigabit Network Processor Worm Filtering

Network solutions for protecting against worm attacks that complement partial end system patch deployment is a pressing problem. We present a scalable worm filter architecture for deployment at ingress/egress transit points on the Internet, including firewalls and gateways. Content-based packet filtering at gigabit line rates, in general, is a challenging problem due to the signature explosion problem that curtails performance in the absence of hardcoded ASIC implementation. We show that for worm malware, in particular, buffer overflow worms which comprise a large segment of recent outbreaks, scalable—accurate, cut-through, and extensible—filtering performance is feasible. This is achieved by three architectural features: (f1) workload sensitivity that affects prioritized handling of non-worm traffic, (f2) worm specificity that exploits the structured nature of worm malware, and (f3) multi-level signature caching that enables programmability and extensibility at cut-through performance. We demonstrate the efficacy of the design by implementing it on an IXP1200 network processor platform with gigabit interfaces. We benchmark the worm filter network appliance on a suite of current/past worms and their polymorphic variants, showing gigabit line speed filtering prowess with minimal footprint on end-to-end network performance.

1.2 Inter-domain and Intra-domain Distributed Worm Filtering

Current solutions for worm protection focus on the individual’s responsibility and vigilance, where each user or organization is expected to keep abreast of patch developments and diligently affect timely software updates before harm is visited. For both technical and non-technical reasons, end system solutions suffer from the Achilles’ heel of partial deployment which curtails their effectiveness at preventing large-scale outbreaks. We describe an infrastructure solution for worm attack prevention that uses distributed worm filtering to affect system-wide protection. We show that for the inter-domain Internet as well as intra-domain ISP networks, content-based worm filtering at strategically selected sites representing a mere 1.5% deployment suffices to yield effective containment. We show that containment performance undergoes a phase transition at a critical filter density whose value depends on the filter placement algorithm, with the engineering consequence that deployment below the critical filter density is ineffective and deployment above superfluous. Distributed worm filtering is potentially applicable to other forms of malware, however, we demonstrate its efficacy using a worm-specific plug-and-play filter architecture implemented as an off-the-shelf network processor appliance (cf. the results on gigabit network processor worm filtering).

1.3 Zero-day Worm Attack Defense

The inter- and intra-domain distributed worm filtering results show that scalable worm attack prevention is possible for recurrent worm attacks with known worm signatures. Since recurrent worm attacks comprise a significant fraction of today’s threat, effective protection against known worms is an important component of the overall solution. However, due to the critical threat posed by zero-day worm attacks, protection against recurrent worm attacks is necessary but not sufficient for protecting the Internet—and individual intranets—against debilitating future worm attacks. We advance a reactive zero-day attack defense architecture comprised of two subsystems: a new worm attack detection subsystem that automatically detects new worms and generates their signature, and a distributed filtering architecture that interfaces with the detection subsystem by loading newly detected worm signatures in real-time. Although the overall defense mechanism is reactive in nature—thus subject to a time window of only a few seconds before significant contamination ensues—we show that the zero-day defense architecture is able to provide scalable protection against unknown worm attacks. The cost for reactivity is an increase in filter deployment cost, from 1.5% to 3%, to achieve containment at the level for known worm attacks. The defense architecture is scalable not only in the sense of small deployment cost as a function of the size of the network, but also with respect to key attack parameters such as scanning rate and bandwidth. For example, protection performance remains invariant as per-host scanning rate is varied from 10 scans-per-second to 10000 scans-per-second and bandwidth is increased from 10 Mbps to 10 Gbps. The zero-day worm attack defense can be employed at the inter-domain network level, intra-domain network level, or firewall filter level.

2 Gigabit Network Processor Worm Filtering

2.1 Motivation and Background

2.1.1 A case for transit worm filtering

Protecting network systems against recurrent and new worm attacks is a pressing problem. Current solutions focus on end system patching, the purview of end user vigilance with system support for streamlining the patching effort. For both technical and non-technical reasons, software patches incur a time lag before they are adopted, and even then, the deployment level is partial [64]. Significant damage may have already been done, and patching becomes more a recovery effort than preventive protection. User-level end system worm filtering [79] can alleviate technically motivated resistance to patch deployment—disruption and reliability—but is still subject to the peril of partial deployment stemming from laxity and insufficient awareness. This leaves a significant segment exposed to infection, from whence hybrid malware may venture DDoS attacks targeted at patched systems. Firewalls perform a variety of functions including AAA/VPN, NAT, VLAN, and selective rule-based packet filtering—e.g., port 445 is commonly blocked by network administrators to filter Sasser and Blaster—but are not designed to carry out content-based worm filtering at gigabit line speeds.

In this report, we tackle the problem of scalable—i.e., accurate, cut-through, and extensible—

worm filtering at ingress/egress transit points in a network system. Although the benefit of worm filtering at firewalls in stub domains and access gateways at transit domains is evident, the main technical challenge to overcome is scalable content-based worm filtering. In general, scalable content-based packet filtering is an inherently difficult problem requiring hardcoded ASIC implementation to achieve accurate—low false positives and negatives—line speed performance. This is due to the signature explosion problem where rule trees, even after optimization, exact a heavy toll on processing power introducing significant slow-down vis-à-vis cut-through performance in the absence of ASIC support. As a point of reference, for computer viruses the degree of freedom availed to an attacker in embedding parasite code in executables is too large to allow scalable detection: not only is behavioral analysis not an option at gigabit gateways, sophisticated obfuscation methods can render polymorphic worm detection next to impossible [16].

2.1.2 Worms and viruses are not created equal

In the case of worm malware—parasite code that travels as part of network protocol interaction—the problem of scalable content-based packet filtering is less daunting. For buffer overflow worms (also called the “vulnerability of the decade” [21]) comprising a large segment of recent worm outbreaks, we submit that scalable worm filtering is tractable due to the structured nature of the malware. By this we mean two key traits: protocol embedding and length invariance. Protocol embedding refers to the orderly nature of client/server network protocols that worms must abide by if their aim is to take over a target system by executing parasite code in the context of privileged system processes. Doing so necessitates conforming to legacy protocol convention to pass through the protocol parsing stage unencumbered, until such point where a targeted vulnerability can be triggered. This well-formedness property constrains the malware to be localized in the protocol interaction, admitting to efficient signature-based detection, a key reason behind EarlyBird’s success in identifying and generating signatures for known and unknown worms over a 8-month period in UCSD’s production network [69]. Protocol embedding is not restricted to buffer overflow worms. Length invariance refers to the vulnerability targeted by buffer overflow exploits, where at a specific point in the protocol parsing process arguments are interpreted and copied resulting in memory overwrites impacting the program’s execution path, the canonical example being the return address in the function stack. A large number of polymorphic variants can be generated by padding, NOP substitution, interleaving and metamorphism of the parasite code that leave its function unchanged. To trigger the buffer overflow vulnerability, however, the pertinent component in the protocol embedding must exhibit a length—some after unicode expansion—that exceeds an application protocol specific bound. Length invariance also holds for encryption-based polymorphism inherited from virus obfuscation [57], where encrypted parasite code is prepended by decryption code. Although random keys and metamorphic obfuscation of decryption code produce a large family of polymorphic variants, buffer overflow must be triggered first to pass control to the decryption routine in the parasite code which exposes encrypted worms to efficient detection through length invariance.

2.1.3 Scope of transit worm filtering

The benchmark results will show that transit worm filtering utilizing protocol embedding and length invariance, augmented by additional architectural features, is able to handle a wide range of worms—buffer overflow and non-buffer overflow—and their polymorphic variants including fragmentation attacks. However, some worms fall outside the reach of our methodology. Principal among them are e-mail worms that infect by invocation of executable attachments. E-mail worms possess the same degree of obfuscation freedom as viruses which renders transit filtering inherently difficult. Some worms spread through existing backdoors, others through weak passwords, neither amenable to our approach. We envision transit worm filtering as complementing end system protection—patch deployment and run-time worm detection—representing the network side of a combined effort.

2.2 New Contribution

We advance a scalable worm filter architecture suited for network processor implementation and incremental deployment at transit points on the Internet. The platform independent architecture is characterized by three key features: (f1) workload sensitivity, (f2) worm specificity, and (f3) multi-level signature caching. The first feature, workload sensitivity, affects prioritized handling of non-worm traffic through traffic differentiation and Bloom filtering that put premium on early identification of non-worm traffic for cut-through forwarding on the fast path. The second feature, worm specificity, exploits protocol embedding and length invariance to detect worm malware and their polymorphic variants at near-cut-through speed in packets held back by (f1). This is enabled by a compact rule tree, made possible by protocol embedding and length invariance, capable of filtering a wide range of polymorphic worms without being subject to the debilitating effect of the signature explosion problem. The third feature, multi-level signature caching, allows (f1) to be carried out on-chip by maintaining Bloom filter bits in general purpose registers. Rule tree signatures are maintained in fast memory, indexed by Bloom filter bitmaps, facilitating near-cut-through filtering of (f2) packets. Feature (f3) allows programmability and extensibility by enabling dynamic, run-time update of general purpose register Bloom filter bits, rule tree signatures in fast memory, and packet forwarding/worm filter code in the instruction store.

We implement the worm filter architecture in an IXP1200 network processor platform with gigabit interfaces and analyze its filtering performance with respect to critical path measurements under varying workload conditions using a benchmark suite of 35 current/past worms. We establish the basic soundness of the worm filter architecture from its performance response in overload stress tests and footprint as a layer-2 traffic relay. To evaluate worm filtering prowess under real-world polymorphism, we benchmark the IXP1200 gigabit worm filter appliance on “in vivo” worm specimens (which includes polymorphic variants) obtained from Internet measurement traces at an active honeypot at the University of Wisconsin [58]. To further stress test the worm filter under polymorphic diversity, we generate a range of polymorphic worm attacks including mutation, IP and application layer fragmentation, metamorphic and encryption attacks using ADMmutate [37], CLET [74], and in-house tools. Our results show that within its intended scope, the worm filter network appliance is able to distill transit ingress/egress traffic of polymorphic worm malware at

gigabit line speed.

3 Scalable Worm Filter Architecture

3.1 Network Processor Platform

3.1.1 System requirement

Worm specificity, (f2), is the main feature that enables scalable worm filtering in the presence of polymorphic variants. Features (f1) and (f3) play an important supporting role, and are readily realized in a network processor platform where programmability and ASIC support meet. Network processors are designed to achieve line speed packet forwarding as long as program driven packet processing does not exceed a system dependent memory reference budget and task specific time complexity. The overall system requirement is dictated by (f3) which predicates one or more packet processing engines with programmable instruction store (MIMD or MISD), general purpose register bank, cache and slow memory, and general purpose CPU. These requirements are satisfied by most commercial network processors [19], although some (e.g., Agere's PayloadPlus) export a restrictive programming model to affect improved pipeline efficiency. The scalable worm filter architecture can be realized over any network processor platform meeting the (f3) system requirements.

3.1.2 IXP1200 network processor

Our specific implementation platform is the Intel IXP1200 network processor with two GigE interfaces. Another variant comes with four FastEthernet interfaces, our original development platform. The IXP1200 is an outdated network processor in the sense that its resources—e.g., 1 K instruction store, 6 microengines (packet processing engines), and 1 Gbps interfaces—are limited compared to newer models such as IXP2800 which has a 4 K instruction store, 16 microengines, and 10 Gbps interfaces. On the flip side, the fact that the worm filter architecture implemented in IXP1200 is able to achieve gigabit line rate filtering of polymorphic worms bodes well for a 10 Gbps port.¹ Following Intel's default programming recommendation, the 6 microengines are divided into two task groups: 2 microengines for ingress packet processing and 1 microengine for egress packet processing, for full-duplex operation on the two GigE interfaces. Each microengine supports 4 hardware threads and is equipped with 128 general purpose registers (GPRs). The IXP1200 follows a MISD parallel programming model. There is a 4 KB cache, 8 MB SRAM, and 256 MB SDRAM. Ethernet frames arriving on an GigE interface are chunked into 64 B PDUs that are shuffled to SDRAM over a 64-bit data bus for further processing by microengines. There is a StrongARM CPU which we use for system initialization, dynamic multi-level signature cache update, and coarse-granular performance measurement.

¹The actual port is straightforward but system benchmarking is not due to a manifold increase in testbed size to accommodate accurate 10 Gbps performance evaluation.

3.1.3 Incremental deployment and usage

We envision a network processor worm filter appliance to be used as a layer-2 traffic relay inserted between two switches or routers. This may be at an ingress/egress link of a firewall in an enterprise network, access router at an ISP connecting multiple stub networks, or gateway at a transit provider, where worm filtering is carried out transparent to the rest of the network including the relayed devices. For example, Purdue University connects to the Internet via a gigabit link through the Indy NOC. So do many other campus/enterprise networks through their respective service providers. Worm malware may be filtered at these gateways using an IXP1200 worm filter appliance. Tier-1 providers operate 10 Gbps backbones whose traffic may be distilled by IXP2800 implementation of the scalable worm filter architecture. Transit worm filtering also allows for the potential to affect zero-day attack prevention, if new worms are speedily detected, signatures automatically generated and uploaded to worm filters in a timely manner (e.g., EarlyBird [69] and Autograph [41]).

3.2 Architectural Features: (f1), (f2), and (f3)

We describe the details of features (f1)–(f3) and how they are implemented in the IXP1200 worm filter. Although each feature embodies different principles, their manifestation in the worm filter architecture is inter-related, resulting in synergistic performance benefits.

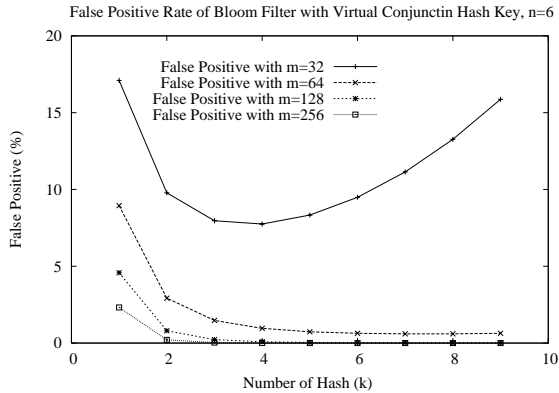
3.2.1 Workload sensitivity

Feature (f1) is comprised of two criteria: one, Amdahl’s law [2] reflecting the composition of input traffic so that attention is directed to frequent/important traffic first, and two, Bloom filtering which provides an effective means to achieve early identification of non-worm (or normal) traffic so that they may be forwarded cut-through on the fast path. Together they result in affecting minimal footprint on normal traffic that enables line rate worm filtering. The first aspect of workload sensitivity has been applied in DPF [24] for packet filtering by invoking dynamic code generation that adapt to the input. In our work, we utilize the properties of Internet traffic [60] to hardcode Amdahl’s law into microcode—e.g., the bulk of Internet traffic is TCP traffic driven by port 80 HTTP traffic—which, if so needed in special environments, may be modified at run-time by dynamically updating the microcode, courtesy of feature (f3). Bloom filters [10] allow efficient worm membership checking, amplified by the small signature set induced by the length invariance property of (f2), worm specificity. Given n worms, k independent hash functions and their range $\{1, 2, \dots, m\}$,

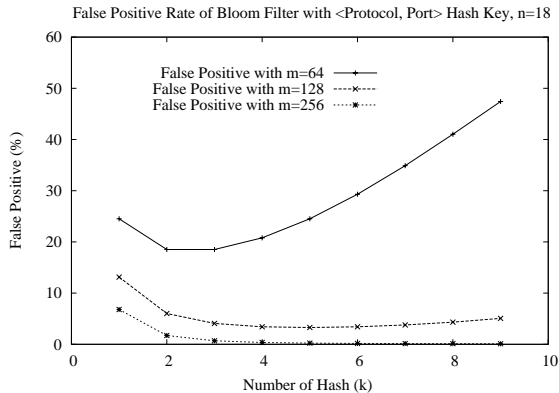
$$\left(1 - e^{-kn/m}\right)^k$$

gives an estimate of the false positive rate: the probability that a non-worm packet is further considered in the worm determination—not actual worm mis-classification—which delays its forwarding thus imparting a detrimental performance effect. For a benchmark suite of current/past worms that covers most buffer overflow, and some non-buffer overflow, worms, $n = 35$ (cf. Section 4). Given a target false positive rate ε (e.g., 5% facilitates line speed performance in IXP1200) we can find k and m that minimizes the false positive rate while achieving ε . Note that k ’s range

is determined by the size of the instruction store and m is determined by cache memory size. As cache size is 4 KB in IXP1200, m is not constrained given the small n . Due to the small instruction store (1 K), however, k is very constrained. Figure 1(a) shows the false positive rate for a range of practical k and m values for HTTP Bloom filtering. It is found that $k = 3$ and



(a) HTTP Bloom Filter



(b) (Protocol, Port) Bloom Filter

Figure 1: False positive rate of Bloom Filter for non-worm traffic.

$m = 64$ achieve the target Bloom filter performance without violating IXP1200’s resource constraints. Figure 1(b) shows the false positive rate for (protocol,port)-based Bloom filtering that checks for infrequent worms, excluding those prioritized by Amdahl’s law such as HTTP. $k = 3$ and $m = 128$ can be shown to meet the performance goal. Since both the Bloom filter bits and hash functions are kept on-chip (GPR and microcode) enabled by (f3), cut-through forwarding is achieved for more than 95% of non-worm traffic.

3.2.2 Worm specificity

Feature (f2) exploits structural properties of worm malware—protocol embedding and length invariance—to effect efficient Bloom filter implementation in (f1) and near-cut-through perfor-

mance of undetermined packets through signature matching. The first stage of (f2), which ties with the last stage of (f1), concerns the addressing scheme used to index the rule tree resident in cache memory. To reduce computation overhead, we use the computed Bloom filter bits, prepended by a $\langle \text{protocol, port} \rangle$ specific shift, to access the relevant part of the rule tree in cache memory. To minimize collision over the cache index space, we employ tier 3 header virtual conjunction that exploits signature locality afforded by protocol embedding. For most worms, the local signature characteristic (or a part of it) resides in the front part of the tier 3 application header. Bloom filtering over the virtual conjunction field introduces sufficient spread into the index map so that collisions are reduced. Table 1 shows samples of tier 3 header prefix and virtual conjunction field for frequent service ports 80, 445, 139 determined by (f1). Rules (i.e.,

Port / Worm	Tier 3 Header Prefix	VC Field
TCP 80 / Codered	“GET /default.ida?”	“GET /d”
TCP 80 / CodeGreen	“GET /default.ida?”	“GET /d”
TCP 80 / Crclean	“GET /default.ida?”	“GET /d”
TCP 80 / Welchia	“SEARCH /AAAA”	“SEARCH”
TCP 80 / WebDav	“SEARCH /AAAA”	“SEARCH”
TCP 80 / Codeblue	“GET /.”	“GET /.”
TCP 80 / Scalper	“GET / HTTP/1.1”	“GET /”
TCP 80 / Nimda	“GET /scripts”	“GET /s”
TCP 80 / Codered-II	“GET /NULL.ida”	”GET /N”
TCP 139,445 / Sasser	10F8FF “SMB” 2F	FF “SMB” 2F
TCP 139,445 / Agobot	10F8FF “SMB” 2F	FF “SMB” 2F
TCP 139,445 / Locator	07BCFF “SMB” 25	FF “SMB” 25
TCP 139,445 / Epmapper	0B98FF “SMB” 2F	FF “SMB” 2F

Table 1: Virtual conjunction: frequent service ports 80, 445, 139.

worm signatures) are represented using a declarative cell or atom structure similar to ones used in [5, 24]. A small but important syntactic distinction is the addition of an op-code field with “>” (in addition to “=”) that allows the length invariance component of feature (f2) to be checked as part of polymorphism determination. The rule tree, by virtue of compactness afforded by length invariance—in the tens of (f2) signatures as opposed to thousands in general packet filters—can be readily optimized. The focus of scalable worm filtering can be shifted away from supporting thousands of rules, an inherently difficult problem, to efficiently catching polymorphic variants aided by features (f1) and (f3).

The last item we consider under (f2)—related to length invariance—is IP fragmentation which, on the surface, seems to require statefulness for assembling packet fragments needed for content-based filtering. For example, the Blaster worm which carries parasite code of length 6176 bytes may be fragmented into 1 byte fragments (worst case) at the attack machine, which would require too much state and overhead to be feasibly carried out at transit points. The same goes for the Slammer worm which has a parasite code of 376 bytes. The Blaster worm, depending on the MTU and LAN that it traverses, may be fragmented into 5 packets if 1500 Ethernet MTUs are used. Other common MTU sizes (excluding headers) are 1492 for LLC/SNP and PPPoE, 4470 for POS/SDH, and 576 for access link IP hosts (e.g., Windows uses a default value of 576). In

order to prevent some worms from evading the length checking functionality, the worm filter has fragmentation handler functionality. Our strategy for dealing with IP fragmented worm attacks is to discard the first fragmented IP datagram if its size is less than a small threshold assuming the protocol type, port number, and tier 3 header prefix—to the extent that partial information is available—prevents outright non-worm packet classification. In [12] it is shown that although MTU sizes above 500 bytes are predominant, there are fragmented first packets of MTU sizes smaller than 400 such as 300, 124, and 44 bytes, inclusive headers. We are not aware of any IP fragmentation attacks that escape the length-based filter rule nor have we encountered well-known applications that transmit fragmented packets where the first packet is smaller than 500 bytes. Application layer fragmentation is discussed along with trace-based benchmarking in Section 4. Figure 2 depicts the worm filter control flow with focus on features (f1) and (f2).

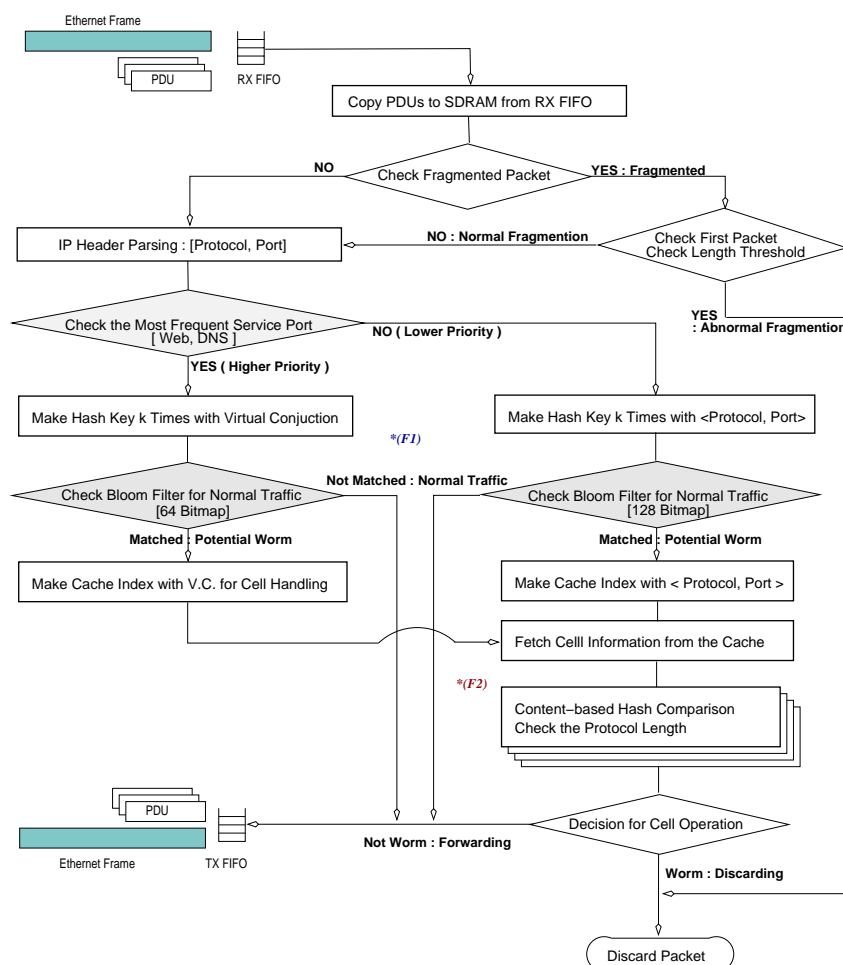


Figure 2: IXP1200-based worm filter flow diagram highlighting features (f1) and (f2).

3.2.3 Multi-level caching

Feature (f3) allows (f1) to be carried out on-chip using GPRs that hold the Bloom filter bits, facilitating cut-through performance. Upon introduction of new worms, an updated Bloom filter bit vector can be computed by the general purpose CPU (StrongARM in the case of IXP1200) that is then downloaded on-chip dynamically at run-time. Second stage worm filtering—biased towards identifying remaining normal traffic not filtered in the first Bloom filter stage—involves read access to cache memory (4 KB in Intel IXP1200) that fetches indexed rules to identify worm and non-worm traffic. Again, by supervision of the general purpose processor, the rule tree in cache memory can be dynamically updated to incorporate the new worm signature. The third and last aspect of programmability and extensibility is update of the microcode itself running on the microengines, should structural changes require (f1) related updates. The disruptive cost is on the order of a couple of seconds but we do not expect microcode updates to be a frequent occurrence.

4 Performance Evaluation

In this section, we evaluate the performance of the scalable worm filter architecture implemented in the IXP1200 network processor with GigE interfaces, first with respect to system side performance followed by filtering prowess under trace-based and artificially generated polymorphism.

4.1 Experimental Testbed

4.1.1 Physical set-up

The physical testbed is comprised of 5 workload generation hosts (i.e., attack hosts) and 5 receiver hosts (i.e., victims) connected by two 3Com GigE switches. Each host, a x86 PC running Linux 2.4.x, is connected by a GigE interface to one of the two 3Com switches which are connected to each other through a GigE link forming a classical dumbbell configuration. The IXP1200 worm filter appliance is inserted between the two 3Com GigE switches, acting as a layer-2 relay transparent to IP. Figure 3 shows the overall layout of the testbed.

4.1.2 Workload generation

We use CBR and VBR traffic generators to generate worm and non-worm workload at sender side impinging on the ingress GigE switch that acts as an aggregator. The aggregate worm and non-worm traffic is relayed through the IXP1200 worm filter whose filtering function can be turned on/off. Traffic exiting the worm filter is demuxed at the egress GigE switch to 5 receiver PCs for load balancing to prevent end system packet loss. Each host, on its own, can generate up to 800 Mbps CBR traffic with MTU 1500 B at full throttle. Accurate, user-specified transmission for controlled experiments is possible up to 400 Mbps. Traffic generators use the IP header source port field to inscribe sequence numbers and mark worm malware packets for ease of end-to-end performance analysis from receiver measurement logs. Fine-granular performance measurements are collected at the IXP1200 worm filter using special tools for critical path evaluation and

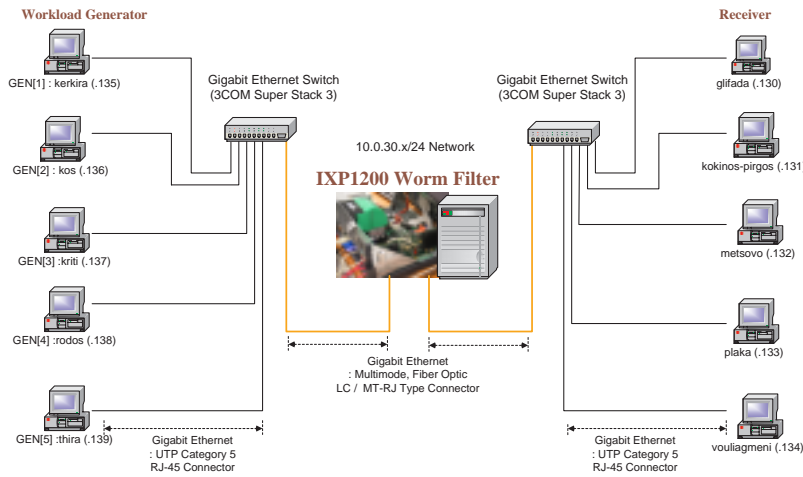


Figure 3: Worm filter benchmark environment comprised of GigE dumbbell with IXP1200 worm filter acting as layer-2 relay.

confirmation of end-to-end performance with in-core measurements. Each performance run lasts for 300 seconds.

4.1.3 Basic worm benchmark suite

We consider a canonical worm benchmark suite comprised of 32 current/past worms—principally buffer overflow worms but not exclusively so—including Blaster, CodeRed2, Dabber, Sasser, Slammer, Slapper, Welchia, and Witty. Worm traffic is mixed with port 80 TCP and port 8080 UDP non-worm traffic. Worm and non-worm traffic are generated by the configurable traffic generators, trace-based and on-line. Figure 4 shows the measured traffic profile at the receiver hosts at aggregate 984 Mbps traffic rate when the filter function at the IXP1200 worm filter was turned off. Non-worm TCP and UDP traffic make up 642 Mbps and 83 Mbps of the total traffic, respectively, with the 32 worms taking up the remainder (259 Mbps), a significant fraction for operational networks.

4.2 End-to-end Worm Filter Performance

Table 2, which serves as a reference point, shows IXP1200 worm filter performance when filtering is turned off under the worm and no-worm workload composition shown in Figure 4. Four sender hosts transmit TCP-based worm and non-worm (or normal) traffic at 220 Mbps each, and one sender PC transmits UDP-based worm/normal traffic at rate 97 Mbps, for an aggregate rate of 977 Mbps. MTU size is 1500 B resulting in an aggregate 80733 pps. All packets are received at the receiver hosts, implying no drops at the IXP1200 worm filter or anywhere else in the system. At the overload load of 999.2 Mbps, 2.9% packet loss rate ensues. Only 0.2% is contributed by the IXP1200 worm filter; the remainder is caused by Ethernet frame drops at the 3Com GigE switch.

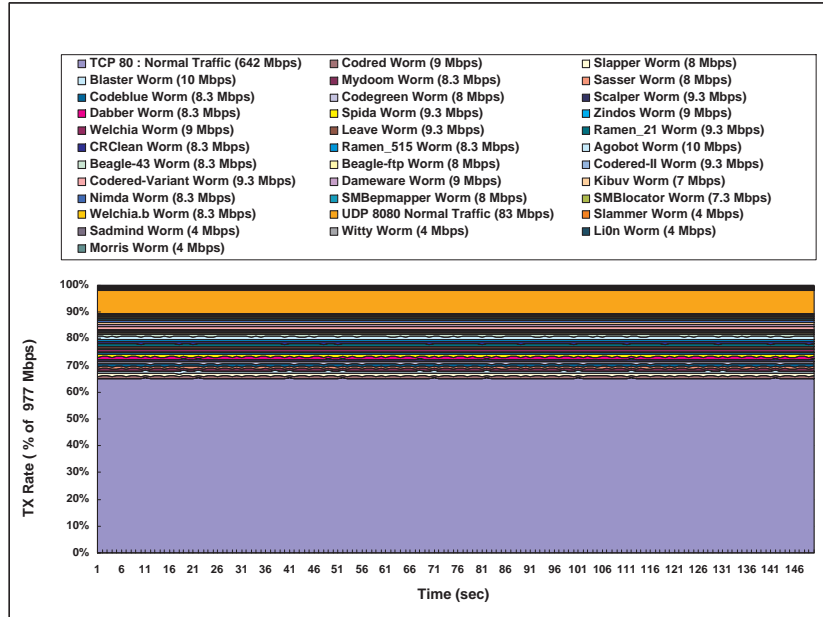


Figure 4: 984 Mbps aggregate worm workload generated by 5 PCs impinging on IXP1200 worm filter appliance.

Table 3 shows corresponding IXP1200 worm filter performance when filtering is turned on. We let the four TCP worm/non-worm workload generators operate as before at 220 Mbps, but the traffic rate of the UDP-based workload generator is throttled down to 60.5 Mbps from 97 Mbps to achieve loss free transmission of all non-worm traffic. The IXP1200 worm filter operates perfectly with no loss, no false positives or false negatives. The aggregate traffic rate is 940.5 Mbps, which is close to the 1 Gbps line rate.

Figure 5 shows IXP1200 worm filter performance with respect to loss rate for varying offered load under three non-worm vs. worm traffic composition: 0.8:0.2, 0.7:0.3, and 0.65:0.35. We observe that the larger the worm workload composition, the earlier packet losses occur at the worm filter due to increased processing overhead of worm malware, which manifests as an upward shift in the loss curve. Overall, at worm traffic composition of 20% or less—practically a significant number—gigabit line speed worm filtering is readily attained in the IXP1200 implementation of the scalable worm filter architecture.

4.3 Filter Latency

Zero packet drop does not imply that there is no adverse impact on normal traffic as latency and jitter incurred at filter relays may accumulate over multi-hop paths, degrading end-to-end QoS of VoIP and other QoS-sensitive applications. Figure 6 shows latency caused by packet processing overhead at the IXP1200 worm filter. The latency measurements were obtained using special tools that allow us to monitor in-core IXP1200 filter performance. Figure reffig:latency-

1500 Bytes-Size	Send-Rate	Recv-Rate	Loss-Rate
Gen-1 (TCP)	220 Mbps	220 Mbps	0 %
Gen-2 (TCP)	220 Mbps	220 Mbps	0 %
Gen-3 (TCP)	220 Mbps	220 Mbps	0 %
Gen-4 (TCP)	220 Mbps	220 Mbps	0 %
Gen-5 (UDP)	97 Mbps	97 Mbps	0 %
Total	977 Mbps (80,733 PPS)	977 Mbps (80,733 PPS)	0 %
Overload	999.2 Mbps (82,581 PPS)		2.9 %

Table 2: Gigabit IXP1200 worm filter performance: Turn-Off Filter

1500 Bytes-Size	Send-Rate	Recv-Rate	Loss-Rate
Gen-1 (TCP)	220 Mbps	160 Mbps	0 %
Gen-2 (TCP)	220 Mbps	160 Mbps	0 %
Gen-3 (TCP)	220 Mbps	160 Mbps	0 %
Gen-4 (TCP)	220 Mbps	160 Mbps	0 %
Gen-5 (UDP)	60.5 Mbps	48.3 Mbps	0 %
Total	940.5 Mbps (77,708 PPS)	688.3 Mbps	0 %
Overload	999.2 Mbps (82,581 PPS)		1.8 %

Table 3: Gigabit IXP1200 worm filter performance: Turn-On Filter

1(a) shows packet latency—from the moment the first PDU of an Ethernet frame is released by the ingress interface to the time when the last PDU of the same Ethernet frame is sent to the egress interface—as a function of time when offered load is 100 Mbps. Figure 6(b) shows the corresponding results when traffic rate is 900 Mbps. We observe a higher average value as well as jitter, and at high load packet processing overhead is sequentially correlated. Overall, however, maximum packet latency is below 55 μ sec, a regime that is not significant for end-to-end QoS even over multi-hop paths. Figure 7 compares IXP1200 worm filter packet processing latency with, and without, worm filtering. We observe that the absolute difference in magnitude—mean and standard deviation—is small.

4.4 Critical Path Analysis

Critical path analysis is a fine-granular in-core measurement analysis of the factors contributing to packet processing (i.e., worm filtering) overhead, with the aim of identifying key bottlenecks and their component-wise costs. Table 4 lists the key elements involved in IXP1200 packet processing for scalable worm filtering which also serve as the measurement checkpoints. Figure 8 shows the timing checkpoint components of the critical path in the worm filter. Let $T_{packet_hold}(x)$ be the packet holding time on each check point(x) as shown in Figure 8.

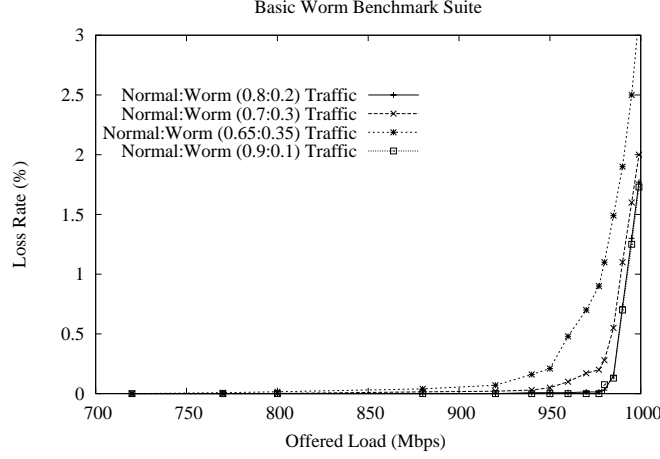


Figure 5: Worm filter performance under varying offered load for different normal-to-worm traffic ratios.

We have

$$T_{packet_hold}(SYSTEM) = \sum_{i=1}^n T_{process}(comp_i) + T_{measure_overhead}$$

where $comp = \{zero, cp_pkt, Ingress_q, pkt_access, bf_vc, bf_port, egress_q, end_of_pkt, cell_process_n, cache_access, pattern_process\}$. Let $T_{process}(y)$ denote processing time for each $comp(y)$. We have

$$T_{process}(comp_i) = T_{packet_hold}(comp_i) - T_{packet_hold}(comp_{i-1})$$

We consider and compute the costs by component type and traffic type below.

4.4.1 Component Type

Constant-time components These components are constant-time related ones. The process time of these components can be regarded as basic cost for worm filtering with (f1), (f2) features, which are described in the previous section.

$$\begin{aligned} T_{measure_overhead} &= Const_{measure_overhead} \approx 0.45usec \\ T_{process}(pkt_access) &= Const_{pkt_access} \approx 0.42usec \\ T_{process}(bf_vc) &= Const_{bf_vc} \approx 2.4usec \\ T_{process}(bf_port) &= Const_{bf_port} \approx 0.6usec \\ T_{process}(Ingress_q) &= Const_{ingress_q} \approx 0.38usec \\ T_{process}(egress_q) &= Const_{egress_q} \approx 1.9usec \\ T_{process}(cache_access) &= Const_{cache_access} \approx 0.19usec \end{aligned}$$

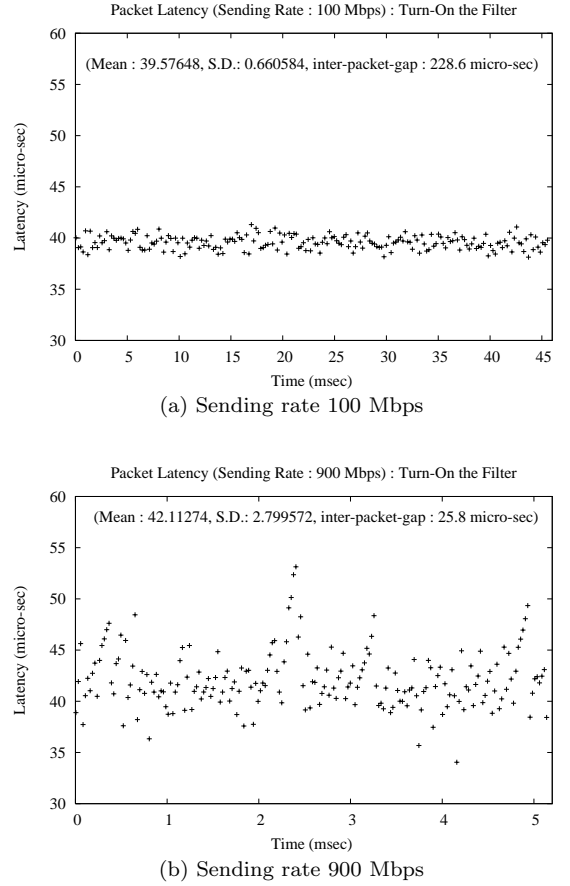


Figure 6: Packet latency: IXP1200 worm filter.

Packet-length-dependent components The process time of these components depends on the length of the packet. Larger the packet, longer the inter-packet gap. This process time does not affect the packet loss rate of the over-loaded traffic.

$$T_{process}(cp_pkt) = f_{cp}(packet) \propto length_{packet}$$

$$T_{process}(end_of_packet) = f_{eop}(packet) \propto length_{packet}$$

Pattern-length-dependent components This component is most critical one – which affects the performance degradation of the filter. As the length of the pattern in the filtering rule becomes longer, the hold time of the packet increases.

$$T_{process}(pattern_process) = f_{pp}(patern) \propto length_{pattern}$$

$$T_{process}(cell_process_n) = Const_{cache_access} + Const_{pkt_access} + f_{pp}(pattern) \propto length_{pattern}$$

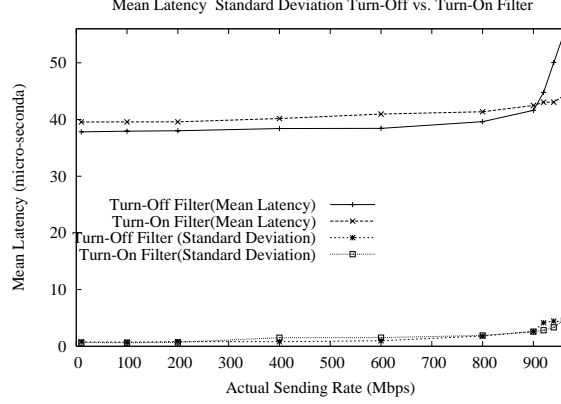


Figure 7: Packet processing mean latency & standard deviation incurred IXP1200 worm filter with filter function turned on/off.

4.4.2 Packet hold time for traffic type

Non-worm traffic The component set for the non-worm traffic is $\{cp_pkt, pkt_access, (bf_vc | bf_port), egress_q, end_of_pkt\}$.

$$T_{packet_hold}(non_worm) = f_{cp}(packet) + Const_{pkt_access} + (Const_{bf_vc} | Const_{bf_port}) + Const_{egress_q} + f_{eop}(packet) \propto length_{packet}$$

Worm traffic The component set for worm traffic is $\{cp_pkt, ingress_q, pkt_access, (bf_vc | bf_port), (cell_process)^+\}$.

$$T_{packet_hold}(worm) = f_{cp}(packet) + Const_{Ingress_q} + Const_{pkt_access} + (Const_{bf_vc} | Const_{bf_port}) + (Const_{cache_access} + Const_{pkt_access} + f_{pp}(pattern))^+ \propto (length_{packet}, length_{pattern}, number_of_cell)$$

False positive non-worm traffic The component set for non-worm traffic—which is false positive on the bloom filter—is $\{cp_pkt, pkt_access, ingress_q, (bf_vc | bf_port), (cell_process)^+, egress_q, end_of_pkt\}$.

$$T_{packet_hold}(non_worm^*) = f_{cp}(packet) + Const_{ingress_q} + Const_{pkt_access} + (Const_{bf_vc} | Const_{bf_port}) + (Const_{cache_access} + Const_{pkt_access} + f_{pp}(pattern))^+ + Const_{egress_q} + f_{eop}(pattern) \propto (length_{packet}, length_{pattern}, number_of_cell)$$

4.5 Trace-based Benchmarking: Active Internet Honeypot

We benchmark the IXP1200 worm filter on measurement traces from an active Internet honeypot at the University of Wisconsin [58] (the authors are not affiliated with the University of Wisconsin; for double-blind reviewing clarification). The network telescope captures packets destined

Components	Description
zero	Tag time-stamp whenever Start_of_packet is detected
cp_pkt	Copy PDUs from RX_FIFO to DRAM
ingress_q	Management for the Ingress queue
pkt_access	Fetch the field in IP packet from the DRAM
bf_vc	Process the bloom filter for virtual conjunction
bf_port	Process the bloom filter for (protocol, port)
egress_q	Management for the egress queue
end_of_packet	Copy PDUs from DRAM to TX_FIFO
cache_access	Fetch the cell of rule from the cache
pattern_process	Hash the pattern & Execute the operator in the cell
cell_process	Process the cell of the filter rule (cache_access, pkt_access, pattern_process)

Table 4: Micro component check points of Worm Filter.

to an unused subset of Wisconsin’s routable address space, strongly biasing the logged packets toward “anomalous” traffic aided by the probing nature of the active honeypot that elicits further responses in the protocol interaction. We classify the packets into sessions based on our catalog of known worms. We partition the packets by port number and each port group is further partitioned by applying a set of coarse protocol embedding rules. They include:

- TCP 80 (CodeRed1&2, Welchia): Payload contains a string in {GET, SEARCH, PROPFIND, OPTIONS} and payload size is greater than 1024 bytes.
- TCP 5554 (Dabber): Payload contains a string in {PORT, PASS}; payload size > 1024 bytes.
- TCP 135, 139, 445, 1025 (Blaster, Welchia, Sasser): First 100 bytes contain SMB (SMB request) or the first byte is 5 and the third byte is 0 (RPC v5 request). Payload size > 1024 bytes.
- TCP 5000 (Kibuv): Payload size is greater than 500 bytes.
- TCP 1434 (Slammer): The first byte of the payload is 4 and payload size > 300 bytes.
- UDP 32772 (Sadmind): Payload contains the value 01 87 88 00 00 in the first 100 bytes of the payload (interface number for sadmind) and payload size is greater than 1024 bytes.

The resultant grouping, due to the lax rule set, yields a superset of worms that may include normal traffic and mutated worm traffic. By casting a wider net, we are able to inspect the polymorphic variants occurring in Internet worm traffic—carried out manually—which aids in analysis of worm filter performance. The polymorphic variety observed is given by: (TCP, 80, 12), (TCP, 1433, 1), (TCP, 5554, 7), (TCP, 135, 31), (TCP, 139, 27), (TCP, 5000, 2), (UDP, 1434, 1), (UDP, 32772, 1). where the triple represents (protocol, port, polymorphic variety). For example, for TCP port 80 we find 12 different instances of Welchia (CodeRed was not contained in the measurement trace) which are comprised of length variations and TCP options where the timestamp option (NOP NOP timestamp) may have been generated by the kernel for RTT estimation. Thus although

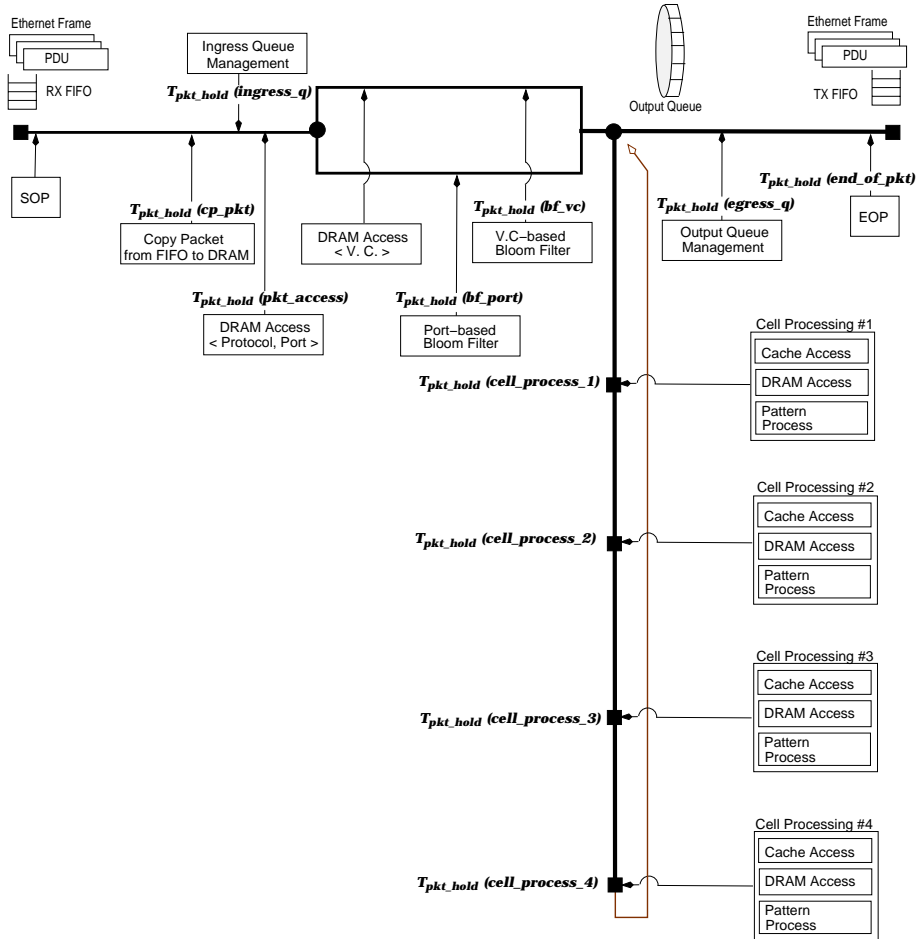


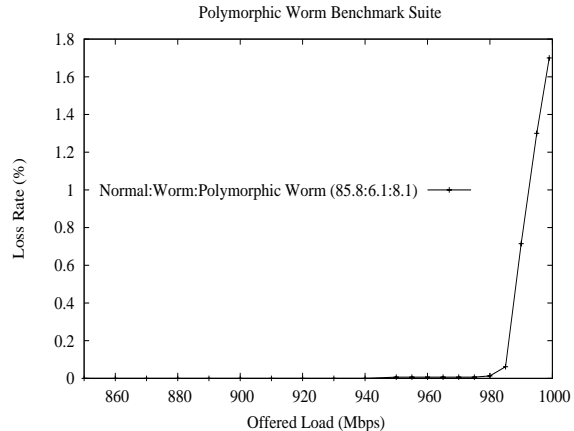
Figure 8: An example of the timing checkpoint for critical packet path.

the 12 instance types comprised actual Welchia worms, their polymorphism need not stem from intentional obfuscation given the degree of freedom admitted by Internet protocols. In the case of Blaster which exhibited 31 types, most were length variations (payload sizes vary from 1260 to 1460 bytes), some prepended a BIND packet before the actual attack packet, and others carried metamorphic parasite code following the payload segment triggering buffer overflow. We found 4 different metamorphic Blaster instances where buffer overflow is triggered by 32+ bytes following the delimiters “\\” at location 916 and 917, with NOPs preceding the metamorphic parasite code starting at sliding locations (e.g., 1102).

Figure 9(a) shows traffic composition of a trace-based polymorphic worm benchmark suite derived from the Wisconsin measurement data comprised of non-worm, worm, and polymorphic variants. Figure 9(b) shows worm filter performance on the trace-based benchmark suite as offered load is varied while the traffic ratio is kept fixed. For buffer overflow worms with polymorphic variants obtained from Internet trace data, the worm filter is able to achieve loss free near-line

Type	Traffic	Proportion
Normal	TCP Port 80	75.20 %
	UDP Port 8080	10.60 %
Worm (Wisconsin)	TCP Port 80 Welchia.B	1.79 %
	TCP Port 5554 Dabber	0.97 %
	TCP Port 135 Blaster	0.39 %
	TCP Port 139 Agobot	1.84 %
	TCP Port 5000 Kibuv	0.43 %
	UDP Port 1434 Slammer	0.14 %
	UDP Port 32772 Sadmin	0.50 %
Polymorphic Worm (Wisconsin)	TCP Port 80 Welchia.B	3.24 %
	TCP Port 5554 Dabber	1.86 %
	TCP Port 135 Blaster	1.75 %
	TCP Port 139 Agobot	0.83 %
	TCP Port 5000 Kibuv	0.35 %

(a)



(b)

Figure 9: (a) Active Internet honeypot polymorphic trace-based benchmark suite. (b) Worm filter performance under varying offered load for polymorphic trace-based benchmark suite.

speed (940 Mbps) filtering performance. Significant onset of packet loss commences at 980 Mbps offered load. The polymorphic types and variations observed fall within the scope of (f2) that makes them efficiently detectable without causing a signature explosion problem.

4.6 Polymorphic Stress Test: Artificial Workload Generation

Performance evaluation with active Internet honeypot trace data indicates that the polymorphic worm variety found in the measurement data is readily handled by the IXP1200 implementation of the scalable worm filter architecture. Although trace-based evaluation provides encouraging evidence that scalable worm filtering may be effected at gigabit line rate “in the wild,” we have also found that the Wisconsin data and in-house measurement data exhibit much less polymorphism than what more sophisticated attacks would be able to muster. To perform polymorphic stress testing, we consider several attack variants, two of which are highlighted below:

- *IP fragmentation.* One of the simplest worm attack variations is to perform IP fragmentation such that tier-3 payload needed for worm signature application is spread across multiple IP fragments. On the surface, filtering fragmented worm packets appears to be an impossible task without using statefulness which would severely impede transit filtering feasibility. Our approach to the problem is to utilize Internet workload property [12] by noting that few (if any) non-malware application traffic exhibit IP fragmentation where the first fragment—in most cases containing the buffer overflow payload in the absence IP fragmentation—is too small (e.g., below 400 bytes). An unusually small first fragment is itself an anomaly that for applicable (protocol,port)-traffic can be filtered to neutralize a fragmented worm attack without inspecting later fragments. For the few worms whose characteristic signature

occurs beyond the first fragment for a valid small fragment size (e.g., 576 bytes), the second fragment suffices to apply signature matching.

- *Polymorphic shellcode encryption.* For viruses, encryption provides a potent obfuscation method. For buffer overflow worms, the situation is different. Obfuscation tools such as ADMmutate [37] and CLET [74] provide worm shellcode encryption for existing worms using different keys each time, aiming to elude detection. For example, polymorphic shell codes generated by CLET are known to evade intrusion detection methods such as spectral analysis. When polymorphic shellcode encryption is applied to buffer overflow worms, for the decryption code to run, the buffer overflow vulnerability on the target system must be first triggered. This implies that the encrypted worm remains exposed to efficient detection by length invariance and protocol embedding: the length invariant cannot be “compressed away” since otherwise buffer overflow—needed to take control of the target system and run the parasite code (the first phase being decryption)—would not be triggered.

Other attack variants include application layer fragmentation/concatenation which, in general, need not abide by known fragmentation sizes but in practice does—some were seen in trace-based data in Section 4.5—for well-known applications, where concatenation occurs over back-to-back application message boundaries affected by TCP buffering. Thus irregular forms of fragmentation/concatenation are easily revealed. TCP option padding is defeated by TCP header parsing and forwarding.

Figure 10(a) shows IXP1200 worm filtering performance on worm workload infused with polymorphic attack variants: IP fragmentation, TCP option padding, application layer fragmentation/concatenation, and polymorphic shellcode encryption using ADMmutate and CLET. We observe no loss filtering prowess in the near-gigabit line speed range with zero false positives and negatives.

4.7 Real-time Adaptivity and Extensibility: Zero-Day Attack

Feature (f3) allows real-time adaptivity and filter extensibility by supporting three levels of worm signature representation: register-level Bloom filter encoding for cut-through forwarding of non-worm traffic (integrated with (f1)), cache-level signature representation for tier-3 worm signature matching for packets withheld by (f1), and microengine microcode updates. We envision cache-level worm signature updates—which lead to on-board register-level Bloom filter updates—to be of potential use for zero-day attack mitigation. In conjunction with an automated worm detection and signature generation system such as EarlyBird [69] and Autograph [41], signatures of new worms may be uploaded onto worm filters, perhaps in time to mitigate—if not prevent—significant damage and wide-spread infection. The rapid pace at which worms propagate [51] puts premium on responsiveness where every second counts, but the potential is there to effect runtime adaptivity to zero-day attacks. On the worm filter side, this is illustrated in Figure 10(b) where non-worm traffic and three types of worm traffic (and their polymorphic variants) are active, with the worm filter not configured with their signatures. At time instance 320 seconds, cache-level filter update is instituted through control messaging which results in an immediate

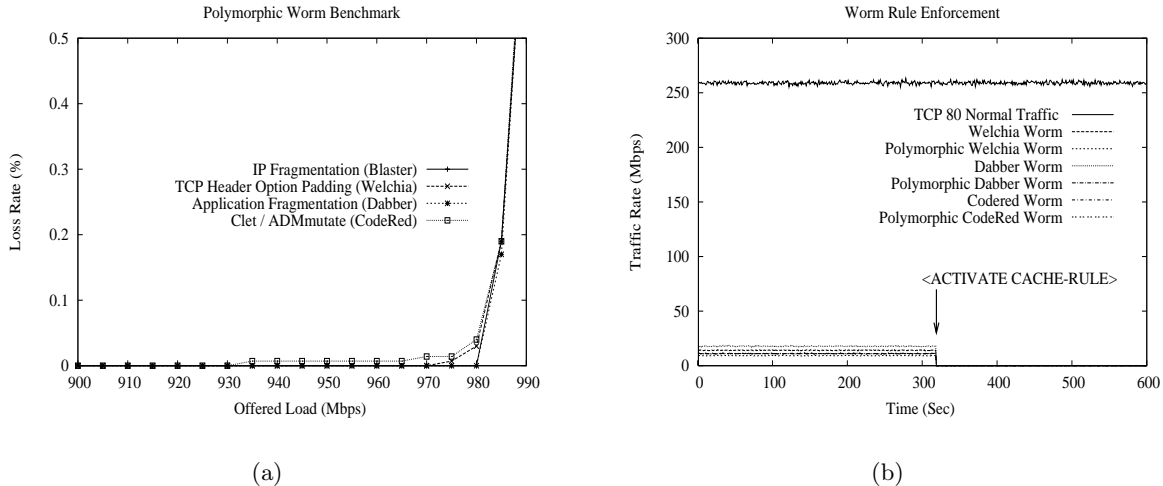


Figure 10: (a) Polymorphic stress test under artificially generated worm workload. (b) Run-time adaptive worm filter rule update.

recomputation/update of the register-level Bloom filter bits that leads to (f1) packet-withholding and (f2) filtering of the worm attack packets.

5 Inter-domain and Intra-domain Distributed Worm Filtering

5.1 Motivation

Protecting network systems against worm attacks is a pressing problem. The time between first dissemination of a new vulnerability and onset of worms implementing the exploit is shrinking—zero-day attack—and known vulnerabilities resurface in mutated form—polymorphism—by-passing detection and prevention systems alike. Despite an abundance of firewalls and intrusion detection systems, the Internet remains exposed to the peril of new and recurrent attacks. We highlight two criteria that influence the effectiveness of a worm protection solution. The first is partial deployment. It is desirable that a solution work, i.e., effect protection near the level of full deployment, even if it is only sparsely deployed. Software patches, for both technical (e.g., disruption and reliability) and non-technical (e.g., awareness and laxity) reasons [79], incur a lag before they are adopted, and even then, the deployment level is partial. Significant damage may have already been done, and patching becomes more a recovery effort than preventive action. The second criterion is global protection. A township that bolts itself shut when faced with a raging epidemic may alleviate short-term harm, but without participating in a cooperative solution that contains the outbreak may eventually succumb to its pervasive reach. The “no man is an island” metaphor applies to worm attacks: neighbors that get infected turn into enemies; if left unchecked, their numbers will grow, becoming fertile ground for further intrusion and DDoS

attacks. Global protection through cooperative synergism is needed to escape the fate of the tragedy of the commons [35].

Today’s solutions are targeted at achieving improved local protection through individual vigilance. They focus on the individual’s security sphere, where each user or organization is expected to keep abreast of patch developments and diligently apply them before harm can strike. Although a necessary component, the end system approach, on its own, is perhaps an onerous security paradigm where each entity, in the end, is left to fend for itself. This report advances an infrastructure solution that shifts emphasis from the individual to the collective. This is not at the exclusion of the former but at the recognition of the opportunities availed to the latter that complement each other. Distributed worm filtering, by deploying checkpoints at select locations, effects global protection that contains worm outbreaks near their source. Extremely sparse, partial deployment is enabled by the power-law connectivity of the Internet—inter-domain and, to some extent, intra-domain—which allows economy of placement without sacrificing protective performance.

5.2 New Contribution

Our contribution is two-fold. First, we advance a 2-level hierarchical, partial deployment, distributed filter architecture encompassing both the inter-domain Internet and intra-domain ISP networks, wherein 1.5% deployment suffices to achieve containment. We show that deployment economy is enabled by the connectivity structure of inter- and intra-domain IP internetworks which exhibit features of power-law graphs essential for scalable protection. We show that containment performance, as measured by a largest contamination, undergoes a sharp change at a critical filter density whose value, in turn, depends on the filter placement algorithm. An important engineering consequence of the “phase transition” is that placement below the critical filter density is ineffective whereas placement above is superfluous in the sense of being redundant. Knowledge of the critical filter density and its minimization are important design factors of the hierarchical distributed worm filter architecture. We show that heuristic rules such as “top 100 domains” [51] can lead to inaccurate conclusions.

Second, we demonstrate the viability of distributed worm filtering, which relies on content-based worm filters as its systems-level building block, by advancing a plug-and-play, extensible worm filter design that is implemented as a network processor appliance. The prototype worm filter focuses on buffer overflow worms—but not exclusively so—and their mutations obtained from Internet measurement traces and polymorphism tools. We show that, unlike general virus parasite code, worms targeting buffer overflow exploits are themselves vulnerable to efficient detection through innate length invariants—in addition to text-based signatures—which admit fast, accurate, and scalable detection aided by the three features of our worm filter design: workload-sensitivity, Bloom filter hashing, and multi-level caching. The first two features allow fast pass-through of non-worm traffic leaving a small footprint, whereas the third feature enables extensibility through on-line programmability. We show through comprehensive benchmarking that gigabit line speed worm filtering can be achieved on past/present worms and their mutations in an off-the-shelf Intel IXP1200 network processor with GigE interfaces.

6 Distributed Worm Filtering

6.1 2-level Distributed Filter Hierarchy

The distributed worm filtering architecture is comprised of a 2-level filter hierarchy where content-based worm filters, under strategic partial deployment, effect worm containment across both the inter-domain Internet at the granularity of autonomous systems and intra-domain ISP networks at the granularity of routers. Global Internet protection is achieved by selective filter placement in the inter-domain autonomous system (AS) graph, and local protection at large transit ISP networks is achieved through selective filter deployment in intra-domain router graphs. Filter placement at a node in the inter-domain AS graph physically means that worm filtering is carried out at all border routers belonging to the AS. Technically, for effect global protection, it suffices to do filtering only in the egress direction which prevents worms from escaping from the filter AS, be they transit or emanating from hosts inside the domain. Ingress filtering, additionally, prevents local infection from the outside. For large ISP domains such as AS 7018 (AT&T Worldnet Service) that serve as tier-1 transit and access providers for a large customer base, protecting their expansive intranet and customer networks, including stub domains, from transit and access network attacks—e.g., “patient zero” or staging hosts residing in an access network—becomes an important concern. This is addressed by intra-domain, router-level partial filter deployment.

Figure 11 illustrates strategic worm filter deployment, first, across the inter-domain Internet for global protection, and second, within intra-domain ISP networks for local protection. The inter-domain AS graph shown is but a small 300-node subgraph of a 16921-node 2004 NLANR/Oregon RouteViews Internet AS measurement graph [54, 77].

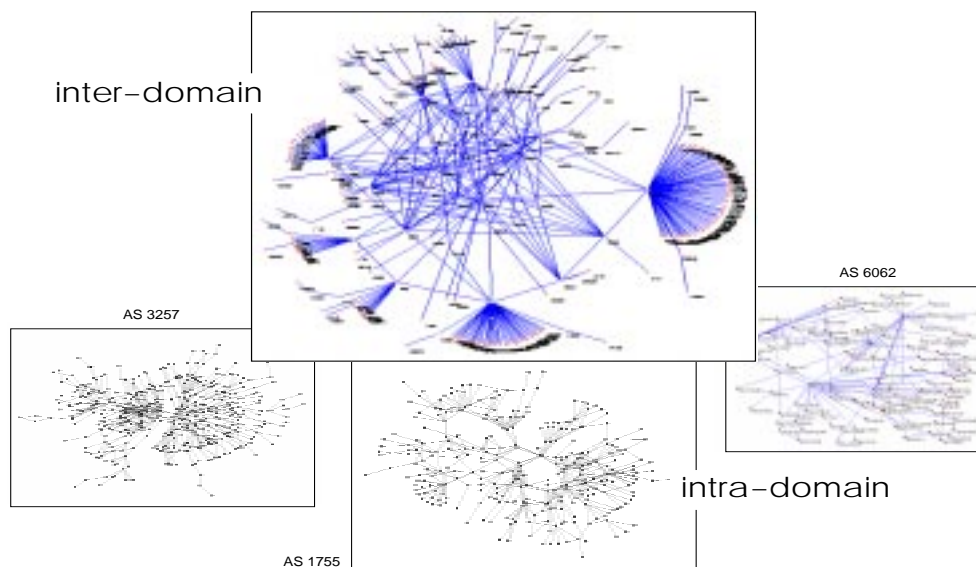


Figure 11: Strategic worm filter deployment in 2-level distributed filter hierarchy spanning the inter-domain Internet and intra-domain ISP networks.

6.2 Strategic Filter Deployment

The opportunities availed by sparse, partial filter deployment for global network security was studied in [59] in the context of spoofed DDoS attacks. It is shown that 15% deployment of route-based filters suffices to achieve proactive and reactive DDoS protection in Internet domain graphs. It is also shown that sparse deployment fails to provide significant protection if the underlying graph structure is non-power-law, e.g., random. Our work shows that for worm attacks, 1.5% filter deployment suffices to achieve containment, a deployment level that is an order of magnitude smaller than that reported for DDoS. An important technical innovation is the use of a novel filter placement algorithm that improves on the vertex cover heuristic (used in [59]) by a factor of 2. Our 2-level hierarchical distributed filter architecture also protects intra-domain ISP networks [71] through strategic partial deployment. Existence of a critical filter density (CFD), its engineering implications, dependence of CFD on the filter placement algorithm, and intra-domain protection are new features studied in this report.

7 Inter-domain Internet

In this section, we discuss performance evaluation of distributed worm filtering on the Internet AS topology.

7.1 Definitions and Set-up

We provide basic definitions and performance evaluation set-up common to both the inter-domain Internet and intra-domain ISP networks. Given a connected graph $G = (V, E)$, a routing protocol R , and a set of filter nodes $S \subseteq V$, node v is *reachable* from u , $u \rightsquigarrow v$, if there is a path from u to v in R such that no filter node is on the path. If $(u, v) \in E$ and (u, v) is a direct route from u to v under R , $u \rightsquigarrow v$ if neither u nor v belong to S . Node v is *infectable* from u , $u \rightarrow v$, if there exists a sequence of intermediate nodes x_1, x_2, \dots, x_k such that $u \rightsquigarrow x_1 \rightsquigarrow \dots \rightsquigarrow x_k \rightsquigarrow v$. Note that under Internet routing—inter-domain or intra-domain—the relation “ \rightsquigarrow ” is not transitive but “ \rightarrow ” is. For example, DoS attacks may be feasible from u to v and v to w without detection and filtering by filter nodes, but that does not imply $u \rightsquigarrow w$. In the inter-domain Internet topology, v may be a stub domain in which case packet forwarding from u to w via v is outright disallowed. Nonetheless, worm infection from u to w through v is feasible. Thus, in general, the range of nodes that can potentially harm a victim is larger under worm attack (“ \rightarrow ”) than DoS attack (“ \rightsquigarrow ”). Surprisingly, the filter density required to achieve worm protection is an order of magnitude smaller than that required to effect spoofed DDoS attack protection. For a non-filter node $u \in V \setminus S$, we denote its transitive closure under “ \rightarrow ” by $[u]$. Thus “ \rightarrow ” under the union of overlapping transitive closures, i.e., perform $[u] \cup [v]$ to form an equivalence class if $[u] \cap [v] \neq \emptyset$, induces a partition of $V \setminus S$ which we refer to as *pockets*. The size of a largest pocket gives an upper bound on the maximum damage an attacker can exact by carefully selecting a single site from which to initiate a worm attack. Pockets represent regions in the graph—small and large—wherein worm outbreaks are contained. The main performance metric, *containment*, is defined as the maximum pocket size given a triple $\langle G, R, S \rangle$.

We consider two filter placement algorithms, pruned vertex cover and load-based. A subset of nodes $V' \subseteq V$ in a graph $G = (V, E)$ is a *vertex cover* if all edges $e = (u, v) \in E$ have at least one end point incident on V' , i.e., $u \in V'$ or $v \in V'$. Given a vertex cover V' , a *pruned* vertex cover of size k , V'_k , is a subset of V' comprised of the top k nodes. The *degree* of a node u , $\deg(u)$, is the number of edges incident on u . Vertex cover was used as the filter placement heuristic for spoofed DDoS attack prevention in [59], and it was a key motivating factor when improving the Internet AS topology generator Inet from version 2.2 to 3.0 [81]. Finding the size of a smallest VC is *NP*-complete. We employ the well-known greedy heuristic which, in power-law graphs, yields tight approximations. In *load-based* filter placement, we utilize the *load* of a node, $L(u)$, in placement decisions. $L(u)$ is defined as the number of source-destination pairs in $G = (V, E)$ that under routing R traverse through u . Thus the higher the load of a node, the more central its role as a crossroad of the flow network. Load-based filter placement picks k nodes of the highest load, partitions the graph into pockets, then recursively applies the k -highest load selection rule to the largest remnant pocket.

7.2 Containment: Critical Filter Density

In this section, we show the existence of a critical filter density at which the maximum pocket size undergoes a sharp change. We also show that power-law connectivity is essential for achieving a small critical filter density (CFD). We use pruned vertex cover (VC) filter placement, which also serves as a reference point, and benchmark containment performance on a 2004 Internet AS topology with 16,921 nodes from NLANR/Oregon RouteViews measurements [54, 77]. The default routing policy is shortest-path. Robustness issues such as different measurement topologies and routing policies are discussed in Section 7.3.

Figure 12 shows the maximum pocket size as a function of filter density under pruned VC placement for the 2004 Internet AS topology and its corresponding $G_{n,p}$ random graph of the same number of nodes n and edge density p . In a $G_{n,p}$ random graph, every edge is independently

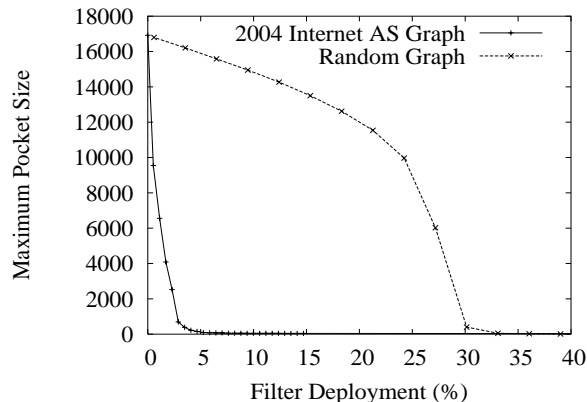


Figure 12: Critical filter density for worm containment in 16921-node NLANR/Oregon RouteViews Internet AS graph vs. corresponding random graph.

selected with probability p . We observe phase transitions in both cases but the CFD in the Internet AS graph at 4% is nearly an order of magnitude smaller than the corresponding CFD of the random graph at around 30%. This represents a significant gain in deployment economy which makes partial deployment viable. Practically, the phase transition implies that filter deployment below the CFD is ineffective—a large segment of the global Internet will get infected—and deployment above is superfluous in the sense of being redundant due to the marginal gain obtained. Thus knowledge of the critical filter density is important to engineering an effective distributed worm filter architecture.

Figure 13 shows the size of pockets, ranked by size, under varying filter density. We observe

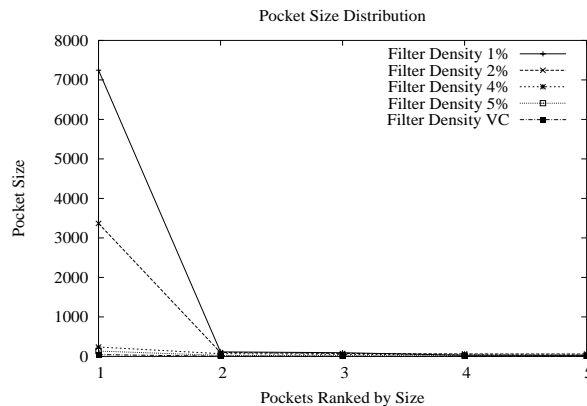


Figure 13: Pocket size distribution, ranked by size, under varying filter density.

that below the CFD the maximum pocket size is given by a singular largest pocket, with the second largest pocket being of negligible size. From an attacker’s perspective, for a given filter deployment, maximum damage with least effort is exacted by compromising a host in an AS u belonging to the maximum pocket with maximum transitive closure $[u]$. In the best case, a single infected AS in the maximum pocket will infect all other ASes in the pocket. Attempting to amplify a worm outbreak by compromising hosts belonging to domains in other pockets entails a proportional effort on the part of the attacker when engaging in known worm attacks. In the case of unknown worms under zero-day attacks, it is a race between worm propagation and worm detection/signature generation/filter update.

Figure 14 shows infection, detection, and filtering time dynamics under zero-day attack in the 2004 Internet AS topology. To facilitate large-scale network security simulation, we use an extension of DaSSF [73], a C++ based process-oriented distributed simulation environment for workstation clusters, with a full TCP/IP stack including socket API, BGP, and application layer protocols. Detection and filter modules are implemented as protocol modules between the IP and link layers. The zero-day attack worm is initiated from 10 randomly selected sites in the 16,921-node Internet AS graph, implementing non-topological scanning—uniformly random from the 16,921 AS number address space—with a 100 scans-per-second scan rate per worm application process. Detection and signature generation nodes are co-located with filter nodes. The prevalence

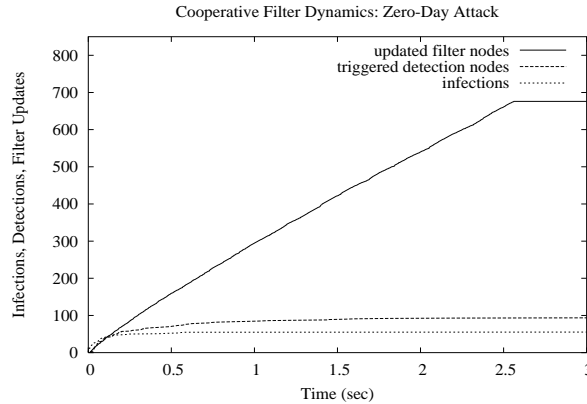


Figure 14: Infection, detection, and filtering dynamics under zero-day attack in 2004 Internet AS topology with 16,921 nodes.

threshold for detection/signature generation is set to 3, the value used in EarlyBird live tests [69]. Figure 14 shows that the head start held by infection propagation is soon overtaken by detection, signature generation, and filter update at the 4% filter sites which contains the outbreak before it has a chance to spread more widely. The battle is won—or lost—in the first few seconds.

7.3 Robustness

In this section, we show that the structural result of Section 7.2 is robust in the sense of not being significantly impacted by time evolution over a 6-year period, inaccuracies introduced in Internet topology measurement and inference, and diversity of routing policies. Figure 15(a) shows maximum pocket size under varying filter densities for NLANR/ Oregon RouteViews Internet AS topologies during 1998–2004. The BGP table dump-based measurement graph has grown from 3,222 nodes in 1998 to 16,921 nodes in 2004. We observe that the location of critical filter density remains approximately invariant during the 6-year span. Figure 15(b) shows maximum pocket size for different Internet AS topology sources, denoted CAIDA, RIPE, USC/ISI, and UMich. Internet AS topologies from CAIDA [13] and USC/ISI [49] use traceroute measurements to infer inter-domain connectivity. RIPE [65] and UMich [76] are based on BGP route table dumps. The UMich topology integrates RIPE and NLANR/RouteViews data, and has undergone corrective processing which increased the representation of medium-degree ASes. We observe more variation in the CFD values, with USC/ISI yielding the smallest CFD of 2.5% and UMich giving the largest CFD of around 5%. RIPE and CAIDA are close to that of NLANR/RouteViews. Overall, effectiveness of distributed worm filtering under sparse partial deployment continues to hold. Figure 15(c) shows containment performance under different routing policies: shortest-path, semi-random, and random routing. To reflect policy diversity of inter-domain routing, when performing BGP’s destination-rooted path-vector routing, we inject varying degrees of randomness in the route table update policy that deviate from shortest-path. The resultant average path lengths for shortest-path, semi-random, and random routing are 3.74, 3.98, and 6.93, respectively. Figure 15(c) shows

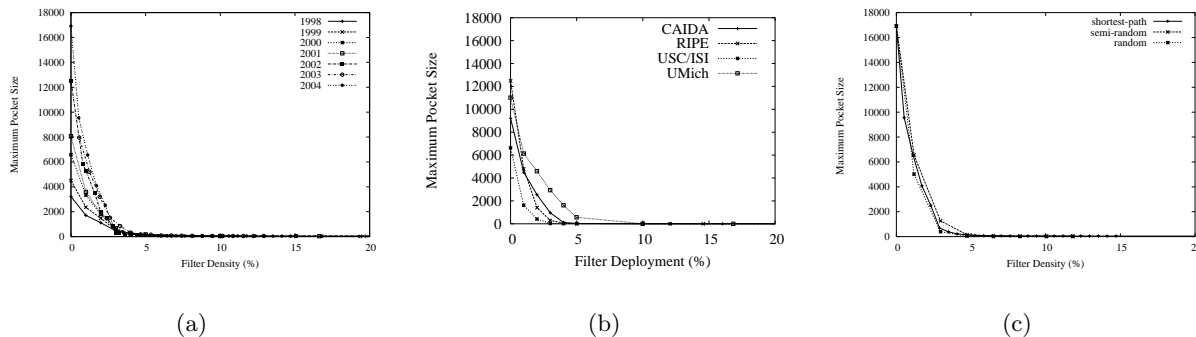


Figure 15: (a) CFD during 1998–2004 under pruned VC filter placement. (b) CFD for CAIDA, RIPE, USC/ISI, and UMich topologies. (c) CFD for shortest-path, semi-random, and random routing policies.

that there is close agreement between the three CFDs. This is due to the power-law connectivity of the Internet AS topology, which constrains the paths a packet can take from here to there without encountering a filter node.

7.4 Load-based Filter Placement

7.4.1 Containment

Minimizing the critical filter density through improved filter placement is an important practical concern as the CFD determines the number of ASes that must participate in distributed worm filtering to effect global worm protection. At the inter-domain Internet scale, a key factor that influences network technology deployment is inter-domain policy barriers across administra-

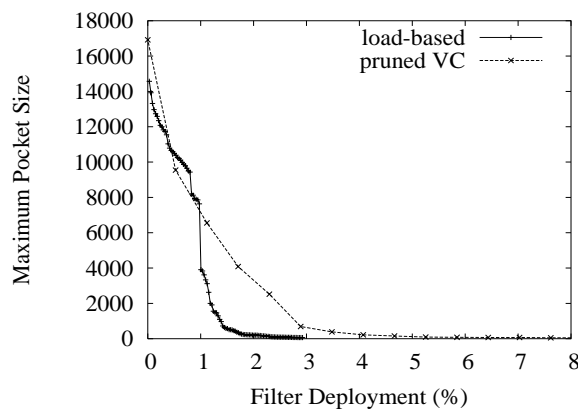


Figure 16: Critical filter density for 16,921-node 2004 NLANR/Oregon RouteViews Internet AS graph under load-based vs. pruned VC filter placement.

tive boundaries. Inter-domain relations tend to be bi-lateral and getting two organizations to cooperate is considered not an easy task. To get more than two to agree is viewed as “exponentially” harder [55]. In an ISP or enterprise network that is under the governance of a single administrative entity, deployment decisions can be instituted top-down subject to economic and technical constraints. For these reasons, the main criterion when evaluating deployment cost at the inter-domain level is the number of participating filter ASes.

Figure 16 compares maximum pocket size under varying filter densities for the 2004 Internet AS graph when pruned VC and load-based filter placement (cf. Section 7.1) are employed. We observe a more than factor-2 reduction in the CFD, from 4% down to 1.5%. For the 2004 Internet AS topology which has 16,921 domains, this implies a reduction from 677 filter ASes to 254 filter ASes, a substantial change in the absolute number. Figure 17 compares filter deployment at CFD under pruned VC and load-based filter placement for the 1998–2004 NLANR/Oregon RouteViews Internet AS topologies. From Figure 17 we can also see why the “top 100 AS” rule studied in

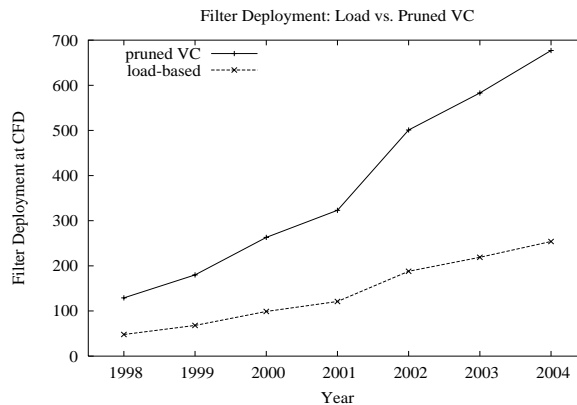


Figure 17: Number of filter nodes at CFD under load-based and pruned VC filter placement for 1998–2004 NLANR/RouteViews Internet AS topologies.

[51] is ineffective at worm containment for all of 1998–2004. The top 100 AS rule picks nodes by absolute degree, which is less efficient than pruned VC computed by greedy VC that uses remnant degrees. Under load-based filter placement, 100 ASes would have sufficed for 1998–2000 to effect global worm protection. For 2001–2004, the requirement is higher. Containment prowess of load-based filter placement also holds for CAIDA, RIPE, USC/ISI, and UMich Internet AS topologies. This is shown in Figure 18. We observe a factor-2 or more reduction in critical filter density across the board compared with pruned VC filter placement (cf. Figure 15(b)). Similar results hold for routing policy diversity.

7.4.2 Suburbia, Backdoors, and Placement

Given the practical relevance of small filter deployment at the inter-domain Internet scale, we would like to understand why load-based filter placement performs better than pruned VC filter placement, and if there is room for further improvement. A small vertex cover induces two effects:

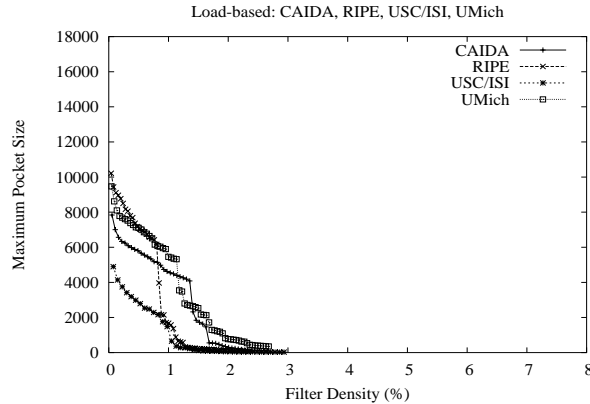


Figure 18: CFD for CAIDA, RIPE, USC/ISI, and UMich topologies under load-based filter placement.

preference for high-degree nodes—multiple edges are caught with one node—and uniform filter density along routes. In power-law networks, the probability that a node u has k edges is only polynomially small, i.e., $\Pr[\deg(u) = k] \propto k^{-\alpha}$ (in Internet topologies [25], the exponent lies in $2 < \alpha < 3$), in contrast to exponential smallness in random graphs, and a general bias toward high-degree nodes is beneficial. The second feature of VC, uniform filter density along routes, is a consequence of the vertex cover property: to cover all edges, at least every other node on a path must belong to the VC. This makes undetected entry of malware traffic, including worm traffic, into the network difficult. The problem of VC as a placement strategy for worm containment is illustrated in Figure 19, which shows a largest pocket resulting from pruned VC filter placement. Empty circles denote stub domains and transit domains are shown as filled circles. A directed edge $u \rightarrow v$ represents a provider-customer relation between ASes u and v , and an undirected edge denotes a peering relation. Pruned VC filter placement, when continued on the largest pocket subgraph to reduce the maximum pocket size, would pick node A which has the highest remnant degree 8. Note that edges that connect this pocket to other pockets must go through filter nodes which are not shown. Picking node A has a marginal effect on the size of the resultant maximum pocket. Load-based filter placement selects node B which has remnant degree 6 but a much higher load—relative to the pocket subgraph—than node A . Selection of B reduces the maximum pocket size significantly. Although high-degree nodes have a tendency to be crossroads, that is not necessarily the case. Figure 20 shows the number of common, i.e., overlapping, and different filter nodes in the ordered filter sets generated by pruned VC and load-based filter placement. Filter nodes selected by the two placement algorithms differ from the start by about 30% which persists over a wide range. This implies the filter nodes selected by the two algorithms are significantly different, not mere appearances caused by differing selection permutations.

Returning to Figure 19, we note that node C is a multi-homed stub AS of remnant degree 3 which plays an important role in enlarging the size of the pocket. Although both small transit domains and multi-homed stubs contribute to making up the branches that link the “suburbia” into a connected whole—i.e., with respect to worm propagation relation “ \rightarrow ”—under the reach-

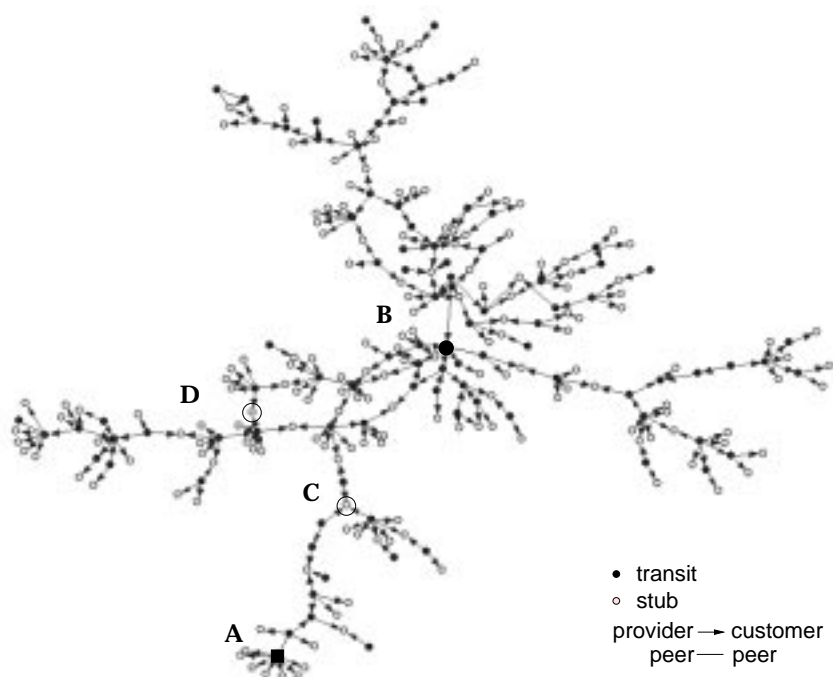


Figure 19: Connectivity structure of largest pocket under VC-based filter placement. Transit domains are shown as filled circles. Empty circles denote stub domains. A directed edge represents a provider-customer relation. Undirected edges represent peering links.

ability relation “ \rightsquigarrow ” pocket sizes would be much smaller. We call multi-homed stubs such as *C* and *D* “backdoors” as they contribute to worm propagation through non-transit channels. The load-based filter placement algorithm incorporates backdoors by computing load without differentiating transit and stub ASes. That is, for filter site selection only, stub domains are treated as if they could forward packets between other domains. A weakness of load-based filter placement is that high-load nodes tend to be clustered, which can lead to redundancies that may be eliminated to enhance deployment economy. Sensitivity of load-based placement to multi-homed backdoor stubs may be further improved which leads us to conjecture that sub-1% CFDs may be achievable.

8 Intra-domain ISP Networks

In this section, we demonstrate how distributed worm filtering within intranets can effect worm containment.

8.1 Local and Global Protection

For transit ISPs such as AS 7018 (AT&T Worldnet Service) and AS 1239 (Sprintlink) that serve as tier-1 transit and access providers to a large customer base, protecting their expansive intranet and

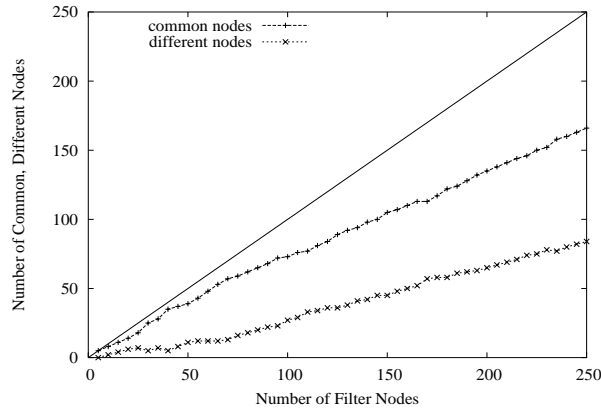


Figure 20: Number of common, i.e., overlapping, and different filter nodes between pruned VC and load-based filter sets.

customer networks, including non-BGP-speaking single-homed stubs and BGP-speaking multi-homed stubs, from transit and internally originating worm attacks is an important concern. From a global Internet worm containment perspective, protecting against zero-day attacks is a time-critical affair where local containment or wide-spread outbreak is decided in the first few seconds of an attack. For example, the zero-day attack that was thwarted by cooperative filtering described in Section 7.2 (cf. Figure 14) may, under different circumstances (e.g., higher scan rate), lead to system-wide outbreak. Such a scenario is shown in Figure 21 where the battle is lost in the first few seconds and a large segment of the global Internet is infected by the new worm. Our performance evaluation results are conservative—from the defender’s perspective—where, for example, we assume that a single scan to an AS will instantly infect the entire AS, a worst-case assumption. In reality, random scans may include unused but routable IP addresses and IP addresses of hosts that are not susceptible to the vulnerability (e.g., different OS), all of which slow down the effective scan rate. Nonetheless, the window of opportunity must be counted in seconds, and intra-domain networks that are internally equipped with strategically deployed distributed worm filters may, when updated with the new worm signature soon after a zero-day attack infection, contain propagation within small segments of its own intranet and access networks—a form of “healing” at the inter-domain scale—that then helps contain global worm infection.

8.2 Containment

8.2.1 Transit ISP Topology: Rocketfuel

We use Rocketfuel [71] transit ISP measurement topology for filter placement and containment benchmarking. Both inter- and intra-domain Internet topologies inferred from BGP dumps and traceroute measurements suffer from a range of inaccuracies [11, 15, 31]. For example, traceroute-based router-level topologies may miss connectivity information due to limited sampling, firewalls, and non-responsive routers. In stub ASes the resultant router-level graphs need not even be

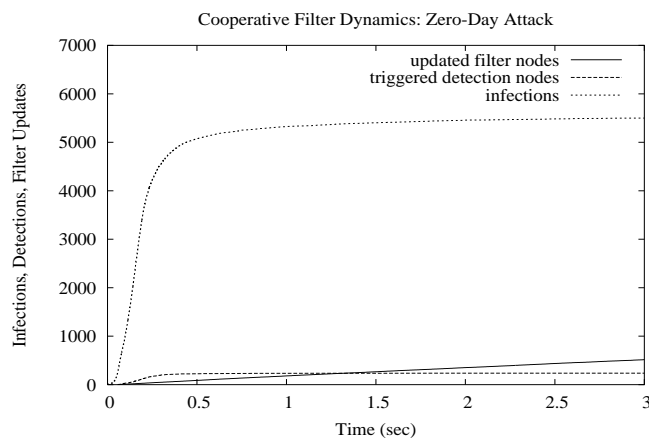


Figure 21: Infection, detection, and filtering dynamics under zero-day attack in 2004 Internet AS topology with 16,921 nodes.

connected, and distinction between routers and end systems is uncertain [70]. Rocketfuel [71] is directed at inferring transit ISP network topology. By performing traceroute through $u \rightarrow v \rightarrow w$ where v is the target transit ISP whose router-level topology is to be inferred and $u \rightarrow v \rightarrow w$ is part of the available AS-level BGP route, reasonably accurate router-level backbone and access network topologies are deduced by making use of DNS name interpretation and aliasing. We use the 10 transit ISP topologies—ASes 1221, 1239, 1755, 2914, 3257, 3356, 3967, 4755, 6461, and 7018—discussed in [71]. Figure 22 shows maximum pocket size as a function of filter density under load-based filter placement for the 10 intra-domain ISP topologies. We observe that for all transit ISP topologies, except Level3, a phase transition occurs below 3%. That is, worm containment under sparse partial deployment is feasible. Level3 (AS 3356) is a special case since, as noted in [71], Level3 operates an MPLS backbone where tunneling creates a multitude of logical links that

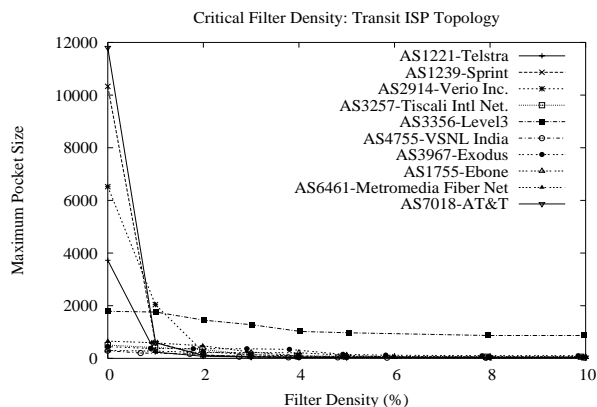


Figure 22: Maximum pocket size under varying filter density for Rocketfuel transit ISP topologies.

lead to a highly connected backbone from the viewpoint of IP. At layer 2, the physical network is expected to be close to that of other transit ISP networks such as AT&T Worldnet and Sprintlink, which is amenable to layer 2 filtering.

8.2.2 Transit ISP Topology: Skitter

Figure 23 shows containment performance for seven transit domain (AS 1221, 1273, 2828, 2379, 3292, 4355, and 6062) measurement topologies from Skitter [70] under load-based filter placement. We observe that the critical filter densities lie below 1.5% indicating that sparse partial deployment is effective. Despite recent progress in understanding of intra-domain topologies [45, 71], our knowledge of intra-domain topologies remains limited, and intra-domain topology performance evaluations must be viewed as very approximate.

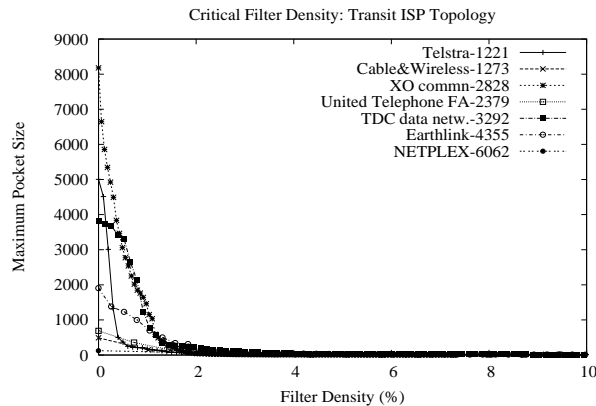


Figure 23: Containment performance for seven transit ISP networks from Skitter.

9 Zero-day Worm Attack Defense Architecture

9.1 Cooperative Filtering: Overall Architecture

The zero-day worm attack defense architecture is composed of two subsystems: a few worm attack detection subsystem that automatically detects new worms and generates their signature, and a distributed worm filtering subsystem that interfaces with the detection subsystem by loading newly detected worm signatures in real-time. Polymorphic variants of existing worms are handled by the network processor based gigabit polymorphic worm filter. Thus unknown worms correspond to newly discovered worm vulnerabilities such as different buffer overflow vulnerabilities targeted by Blaster and Slammer worms, as opposed to polymorphic variants of the Blaster worm—i.e., its family—which may include exploit instances comprised of metamorphic variants and fragmentation attacks. The division of labor is essential for curtailing the signature explosion problem. We call the combined system cooperative filtering. Figure 24 depicts a coarse-scale deployment of such a cooperative worm filter architecture. The deployment problem, now, has two

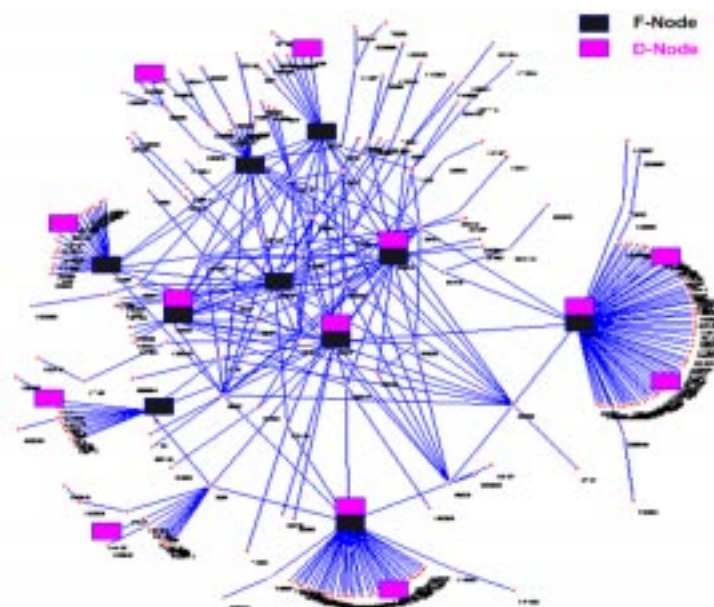


Figure 24: Cooperative worm filtering: distributed worm detection/signature generation (D) and worm filtering subsystems (F).

components: optimal distributed filter placement (discussed under distributed worm filtering of known worms) and optimal distributed detector placement. The latter is discussed in Section 10 under optimal new worm attack detection.

Responsiveness and timeliness—the interlude between initial onset and large-scale worm outbreak is counted in seconds—susceptibility of the detection subsystem to bogus attacks that generate signatures leading to false positives, and vulnerability of the signaling plane to DDoS attacks are issues investigated in on-going work.

9.2 Detection and Signature Generation Subsystem

A promising recent development is detection and automated signature generation of unknown worms [41, 69]. The speed and accuracy reported for EarlyBird [69] in UCSD’s production network indicates that distributed worm detection and signature generation—an aspect emphasized by Autograph [41]—may be feasible. By coupling the worm filtering and worm detection/signature generation subsystems through a secure signaling plane, an adaptive worm containment system—cooperative filtering—is constructed that, despite its reactivity, is able to defend against zero-day attacks.

9.2.1 Detection Subsystem

We assume a threshold based new worm detection subsystem such as EarlyBird that exploits two features—content prevalence and address dispersion—to detect new worms. Content prevalence

relies on the heuristic that worm payloads tend to share one or more substrings, and due to their rapid propagation, watch posts can be primed to detect frequent occurrence of packets with common substrings within a given time period. In contrast, most non-worm traffic do not share common substrings. For example, in [69] it is shown that when applying CRC over whole packets, during a 60 second time interval, 97%+ of all signatures occur less than 3 times, with most occurring only once. However, some non-worm traffic such as certain client/server traffic do exhibit frequent overlap due to the nature of the underlying application, and address dispersion is needed to filter out these packets to reduce false positives.

Given a sequence of packets x_1, x_2, \dots , threshold based content prevalence detection is captured by assuming a pattern matching function f such that

$$f(x_{i_1}, x_{i_2}, \dots, x_{i_k}) = 1, \quad i_1 < i_2 < \dots < i_k,$$

for some k , if $x_{i_1}, x_{i_2}, \dots, x_{i_k}$ share a common substring (in general, a “pattern” including subsequence) of suitable length, and 0, otherwise. We define the counter function F as

$$F(x_i) = k$$

if for some subsequence $i_1 < i_2 < \dots < i_k = i$ we have $f(x_{i_1}, x_{i_2}, \dots, x_{i_k}) = 1$, and the subsequence is maximal. Given a content prevalence threshold θ , we say that content prevalence detection has occurred at packet x_i if $F(x_i) > \theta$. Source address dispersion is defined analogously by a source address matching function f_S where the common substring is the IP packet’s source address with its counter function F_S . The same holds for destination address dispersion with functions f_D and F_D , respectively. Finally, given thresholds θ , θ_S , and θ_D , we say that a (new) worm is detected at packet x_i if

$$F(x_i) > \theta, F_S(x_i) > \theta_S, \text{ and } F_D(x_i) > \theta_D, \quad (1)$$

and for some subsequence $i_1 < i_2 < \dots < i_k = i$ we have

$$f(x_{i_1}, x_{i_2}, \dots, x_{i_k}) = f_S(x_{i_1}, x_{i_2}, \dots, x_{i_k}) = f_D(x_{i_1}, x_{i_2}, \dots, x_{i_k}) = 1 \quad (2)$$

where $k > \max\{\theta, \theta_S, \theta_D\}$. The focus of our work is not in finding efficient f , f_S , and f_D with small space and time complexity, but in utilizing existing detection systems such as EarlyBird that exhibit reasonable performance for zero-day worm attack protection.

The issue of optimal worm filter placement has been discussed in the second part under distributed worm filtering. The two main issues that arise newly in cooperative filtering for zero-day attack defense are:

- optimal detector placement
- efficacy of cooperative filtering

They are discussed in the next two sections. First we discuss some issues related to automated signature generation as they pertain to polymorphic worm filtering by the network processor based gigabit worm filter.

9.2.2 Signature Generation Subsystem

Worm detection systems such as EarlyBird are capable of automated signature generation. EarlyBird, due to its common substring matching based heuristic, generates string matching signatures—a specific form of pattern matching based packet filtering—which predicate that a string s (in general, a set of strings that may be combined through conjunctions and disjunctions when defining successful pattern matching rules) be matched in packet payloads. For example, in Snort [22], the rule header would specify the rule action `pass` for discarding a packet, with the rule option `content` that specifies the string s to be matched. Of course, protocol, port, and any address related constraints are specified as part of the rule header. The resultant Snort rule may be uploaded onto worm filters using a secure signaling protocol.

The preceding framework for automated signature generation, however, is insufficient for scalable protection against zero-day attacks. First, we require polymorphic protection against new (and known) worms. Although not many instances of polymorphic worms are known to occur in Internet traffic in the recent past [41], polymorphism cannot be ignored, especially as it pertains to future worm attacks. Line speed worm filtering is not feasible if there is a separate signature for every polymorphic variant of a worm. As indicated earlier, in our definition of new worm, polymorphic variants—including metamorphic variants and fragmentation attacks—are excluded, i.e., they do not count as new worms. Our work in polymorphic worm filtering (part I) for carrying out scalable worm filtering using network processors deals with the signature explosion problem. But the signatures generated by existing automated detection and signature generation systems do not sufficiently incorporate polymorphic invariance.

We can approach the problem in three ways. In the first approach, we take the signatures generated by EarlyBird and perform polymorphic compaction in the worm filter subsystem. Although modular, the main problem with this approach is that the degree of polymorphic compaction achievable at the worm filter subsystem is limited by the information provided by the automated detection and signature generation subsystem through the worm signatures. As a case in point, for Sadmind, the payload contains the value `01 87 88 00 00` in the first 100 bytes of the payload but EarlyBird need not convey this information in the generated signatures. Without further information, polymorphic compaction carried out at the filter subsystem cannot infer this additional structure that is critical for gigabit line speed filtering. In the second approach, EarlyBird, in addition to generating an initial signature, provides sample packets so that the filter subsystem can perform polymorphic compaction that generates enhanced polymorphic rules that include `content` rule option modifiers such as `offset` and `depth`. Although still modular, a drawback of the approach is that payload information needs to be logged at the detection system—a nontrivial cost given the premium on space complexity placed by the detection system—and transmitted over the network which can slow update where every second counts.

The third option—and likely the most effective option—is to enhance the capabilities of the detection and signature generation subsystem so that sensitivity to `offset` and `depth` are incorporated in the detection and signature generation process. Furthermore, a critical component of polymorphic filtering for buffer overflow worms is length invariance, which, to infer in an automated fashion, requires assistance from the detection subsystem. In particular, when a new worm has been detected of a protocol type (e.g., TCP, UDP) and port number, then detection of non-

worm packets of the same protocol type and port number must be instituted—i.e., packets not matching the substring signature(s) detected—so that approximate length invariance information can be inferred. The latter, at packet granularity in Snort, may be incorporated by `dsize` in the rule option. Due to timeliness constraints, an imperfect polymorphic new worm signature—perhaps not containing `dsize` information even if it is a buffer overflow worm—may be uploaded to the worm filter subsystem to combat the zero-day attack, with improved polymorphic rules updating (i.e., overwriting) the initial rules as length invariance information becomes available. Ultimately inferred length invariance information must be confirmed by human analysis of the newly discovered buffer overflow (BO) vulnerability, in case of BO worms.

9.3 Worm Filtering Subsystem

The worm filtering subsystem is a distributed worm filter architecture where each worm filter is comprised of the network processor based gigabit polymorphic worm filter appliance. A secure signaling protocol exports a well-defined interface to the detection and signature generation subsystem from which new worm signatures are uploaded. As discussed in part I, feature (f3) allows even the microcode to be updated at run-time. We do not view the latter as a frequent occurrence.

The steps in the overall zero-day worm defense are depicted in Figure 25. Note that distributed new worm detection does not merely mean collection of traffic information from multiple detector nodes for joint analysis [41] but distributed placement of worm detectors on the global inter-domain Internet so that new attacks are most speedily detected. The issue of optimal worm detector placement for effective distributed worm detection is discussed next.

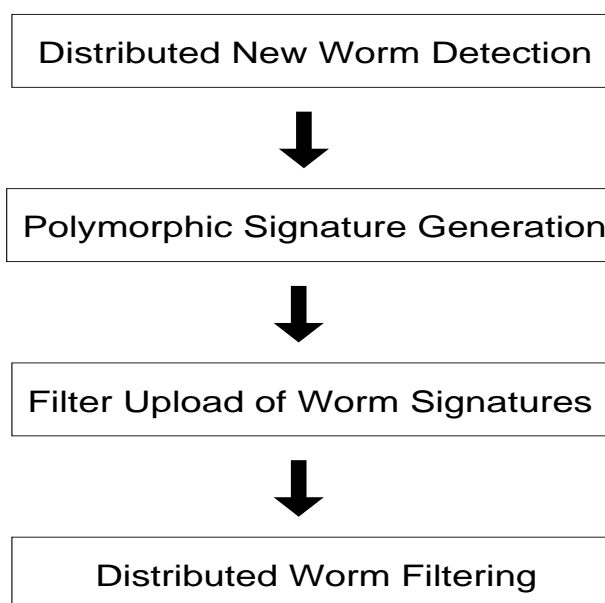


Figure 25: Main steps in overall zero-day worm defense architecture.

10 Optimal New Worm Attack Detection

10.1 Problem Definition

Given a graph $G = (V, E)$, by default, the Internet AS topology, we are interested in the optimal (distributed) detector placement problem: Given a budget of k detectors, find a placement $D \subseteq V$, $|D| \leq k$, of detectors onto nodes such that detection time is minimized. Unless otherwise specified, we assume a new worm detector uses content prevalence and source/destination address dispersion as in EarlyBird. Zero-day worm attack is assumed to commence from a subset $S \subseteq V$ of attack sites. Bandwidth, link latency, and other network parameters are assumed given as well. Given a placement D of detection nodes, the earliest detection time is defined as the first time instance when conditions (1) and (2) are satisfied. In EarlyBird, the content prevalence threshold must be first triggered before source/destination address dispersion threshold counting is started, but these are secondary details of limited relevance.

The optimum detector placement problem can be couched in multiple ways. For example, given a target detection time t^* , find a smallest $D \subseteq V$ such that t^* is satisfied. Note, for sufficiently small t^* , a solution need not exist, i.e., $D = \emptyset$. Although we have not formally proved it yet, the optimal detector placement problem is expected to be NP-hard. Following is a brief overview of what is known about optimal detector and filter placement for worm and DDoS attack protection, which are not unrelated.

10.2 Optimal Filter Placement: Distributed Worm Filtering

Given a graph G , consider the optimal filter placement where a smallest filter set $F \subseteq V$ is sought so that containment is achieved within pockets of size s . Modulo the detailed specification of G (its networking related specifications), the problem is captured by the s -SIZE CONNECTED COMPONENTS problem [30]: Given an undirected graph $G = (V, E)$ and integer $1 \leq s < |V|$, find a minimum subset of vertices $F \subseteq V$ such that the graph G_F induced by $V \setminus F$ is comprised of connected components of size at most s . In general graphs, an old result by Yannakakis [82] shows that the problem is NP-complete (reduction from 3SAT). Note that a special case of the problem, 1-SIZE CONNECTED COMPONENTS ($s = 1$), is equivalent to MINIMUM VERTEX COVER, which means that we can view the former as a generalization of the latter. In a recent result [28] (joint work with Ferrante and Pandurangan), we prove that MINIMUM VERTEX COVER for power-law graphs is NP-hard, we believe the first known hardness result for power-law graphs. Therefore, in a trivial sense, optimal filter placement in power-law graphs (certain deterministic restriction of Aiello et al.'s power-law random graph model [1]) for worm containment is also NP-hard. We remark that in a recent result [4] (joint work with Armbruster and Smith), we show that optimal route-based filtering for perfect spoofed DDoS protection is NP-complete.

10.3 Optimal Detector Placement: Distributed New Worm Detection

The preceding optimal filter placement results are of some relevance to optimal detector placement in power-law graphs. The first connection can be made by considering 1-SIZE CONNECTED COMPONENTS (i.e., MINIMUM VERTEX COVER). Given a detector set $D \subseteq V$ that is a

vertex cover (VC), we can consider a static worm attack detection problem focusing only on content prevalence. That is, a set of $S \subseteq V$ attacker nodes transmits worm traffic but we ignore the impact of new infections on detection and we only count how many worm packets are seen (i.e., address dispersion is ignored). Assuming collaborative detection by distributed detection nodes is ignored, then D being a vertex cover assures earliest possible worm detection (more precisely, within an additive hop count of one which, in the absence of full deployment, is optimal) for any attack configuration $S \subseteq V$, and the optimization problem (under the worst-case attacker model) reduces to finding a smallest such D . That is, an optimal detector set D , in order to achieve earliest detection under worst-case attack, must be a vertex cover. When D is a VC, earliest detection should hold even under source address dispersion, dynamic infection, and detection node collaboration, but an expanded formulation is needed to make the system components mathematically well-defined. Since we have shown that MINIMUM VERTEX COVER in power-law graphs is NP-hard [28], as in optimal filter placement, optimal detector placement is also NP-hard.

In general, for detector set D satisfying the s -SIZE CONNECTED COMPONENTS property that yields disconnected subgraphs (i.e., pockets) with vertex sets U_1, \dots, U_m (for some m) where $|U_i| \leq s$ for $i \in \{1, \dots, m\}$, we arrive at an upper bound on the earliest detection (given D) which is s (in unit of hop count). For pockets with logarithmic diameter, earliest detection is upper bounded by $\log s$. Figure 26 depicts a pocket A (with respect to D) and its detection nodes a, b, c , and d for worm traffic emanating from A . Because the pocket is “multi-homed,” for different

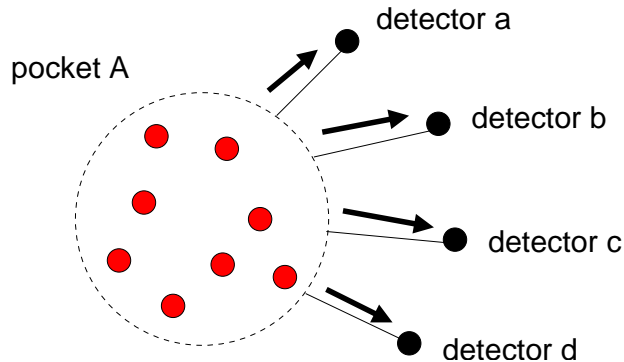


Figure 26: Detection nodes a, b, c , and d for pocket A for worm traffic emanating from A .

destination addresses for a single attack host in A , or for multiple attack hosts in A with a common destination (and, of course, their combination), egress traffic may exit on different links. By the s -SIZE CONNECTED COMPONENTS property, all egress/ingress links are incident on a detection node belonging to D . Hence by collaboration, earlier detection—content prevalence based and address dispersion based—is possible. In the above example, detectors a, b, c , and d may pool their information to reach θ, θ_S , and θ_D faster than is the case with a single detector. Collaboration can extend across a larger detector set, and the delay associated with multicast information dissemination must be considered when evaluating timeliness. At the coarsity of pockets of sizes up to s , the detection nodes in D form a VC of the coarsified graph where the i 'th

pocket U_i is treated as a single (super-) node. Hence, with respect to super-nodes U_1, \dots, U_m , D assures earliest detection by the VC property.

10.4 Optimal Joint Detector and Filter Placement

Consider the single attacker case (i.e., $|S| = 1$). For distributed filtering of known worms, maximum infection damage with respect to a filter set F is given by the maximum pocket size under worst-case attack. The maximum damage, if the attack node belongs to pocket U_i , is $|U_i|$. Under multiple attackers, maximum damage is given by the sum of the pocket sizes that the attack nodes belong to. In the single attacker case, optimal filter placement given maximum damage budget s reduces to the s -SIZE CONNECTED COMPONENTS problem. Consider the joint optimization problem: Given a maximum damage budget s , find a minimum joint detector/filter set H such that H satisfies s and achieves earliest possible detection. For $s = 1$, satisfying the damage budget involves solving the MINIMUM VERTEX COVER problem. At the same time, note that $s = 1$ implies that the earliest possible detection time $t^* = 1$, which can only be achieved by a vertex cover. Hence joint optimization reduces to the MINIMUM VERTEX COVER problem, and it suffices to solve the optimum filter placement problem whose solution is also a solution to the optimum detector placement problem (and vice versa). An analogous relationship holds for $s > 1$ with respect to s -SIZE CONNECTED COMPONENTS since a pocket may be of size up to s , and, if so, the earliest detection time (under worst-case attack) is bounded below by s .

Complications arise due to time dynamics, multiple attacker sites, non-worst-case attack scenarios, and address dispersion. These are studied in comprehensive dynamic network simulations using DaSSF [73].

10.5 Effective Detector Placement

10.5.1 Optimal Single-source Attack Detection and Destination Address Dispersion

Consider a source $u \in V$, viewed as a potential attacker, and the problem of most speedily detecting a new worm attack emanating from u with respect to content prevalence and destination address dispersion. Consider a node $d \in V$, $d \neq u$, interpreted as a potential detector node, and define $L_u(d)$ as the total number of destinations whose paths (given by a routing system) go through d . Thus $L_u(d) \leq |V| - 1$. Supposing that all destination addresses are generated equally likely (e.g., random scanning), $L_u(d)/(|V| - 1)$ represents the probability that detector d would observe a packet emitted from u . Define

$$L_u = \max_d L_u(d).$$

L_u captures how well an optimum detector for attack source u is at detecting worm packets emanating from u under random scanning. Figure 27 shows L_u as a function of u for the 2004 Internet AS topology with 16921 nodes and 36767 links from NLANR/Oregon RouteViews measurements [54, 77]. For close to 8000 nodes, there is at least one detector node—for each attacker node—through which all traffic from that attacker node, irrespective of destination, must go through. Note that we are taking a sampling viewpoint when considering the meaning of “early

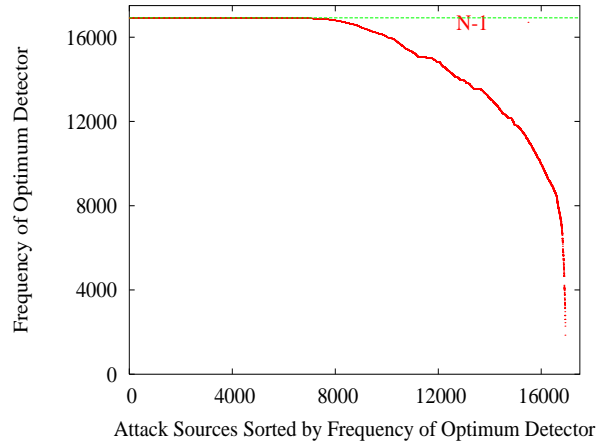


Figure 27: Detection capability of a maximum detector L_u for different sources u sorted (non-increasing order) by L_u .

detection”; that is, if an attack source u generates a sequence of packets x_1, x_2, \dots, x_n under random scanning, a detector that sees the most number of packets (in probabilistic term, expected number of packets) is considered an optimum detector. The number of samples needed to reach a content prevalence threshold θ is smaller, which, translated to time, means that elapsed time is shorter: the packet generation process is treated as stochastic point process, where x_1, x_2, \dots, x_n are associated with arrival (i.e., generation) time instances $t_1 < t_2 < \dots < t_n$. For a source u with optimum detection probability $p_u = L_u/(|V| - 1)$ and optimum detector d , d requires that u generate on the average $1/p_u$ packets before it sees a single packet. Hence the smaller p_u , the more worm packets will go undetected by u , causing correspondingly higher infection damage before a new worm attack is detected and containment measures—in our case, through packet filtering—instituted. In Figure 27, the node u ranked 16580 has d_u value 8463 which is slightly more than half of 16921, the total number of nodes in the AS topology. That means that when attacks are ventured from u , its optimum detector, on average, sees every other packet generated by u .

We remark that the definitions $L_u(d)$ and L_u incorporate destination address dispersion on a per attack source basis. That is, since $L_u(d)$ represents the number of distinct destination addresses that detector d sees for source u and L_u is its maximum, destination address dispersion, as resulting from packets emitted by a fixed source u , is captured by L_u . Of course, we are interested in finding a small set of detectors, not necessarily optimum for all attack sources, that are capable of detecting destination address dispersion across all potential attack sources. This issue is studied in the next section.

Figure 28(a) shows the degree distribution of attack source nodes, ranked in the same non-increasing order as Figure 27. The first 6000 nodes have degrees in the range 1–3, all of them single-homed or multi-homed stub nodes that are perfectly (i.e., with probability 1) detected by their optimum detectors. Many are directly connected to one or more high-degree nodes which act

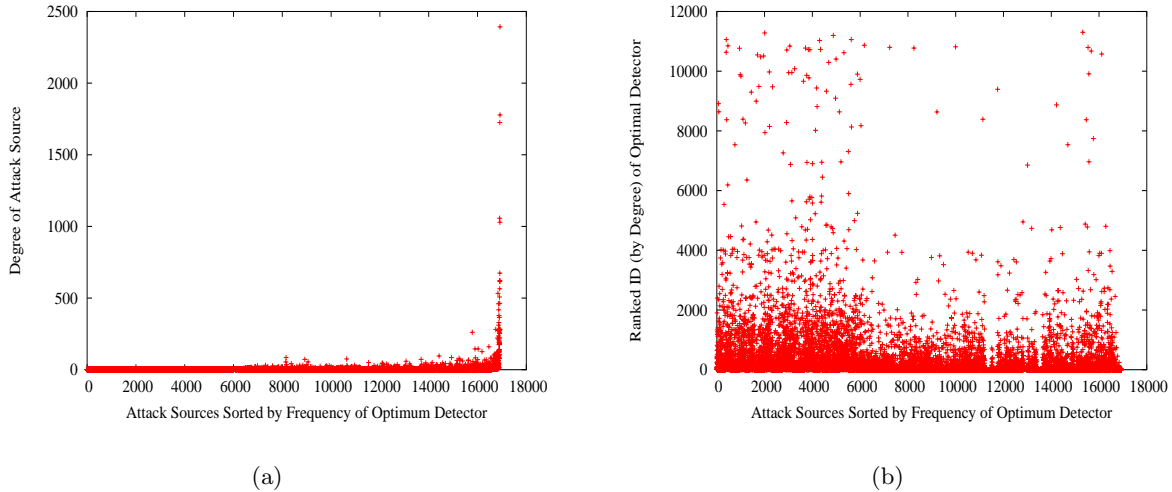


Figure 28: (a) Degree distribution of attack sources u sorted by frequency L_u of their optimum detector. (b) Ranked ID (by degree) of optimum detector for the corresponding sources.

as optimum detectors for these stub attack nodes. Nodes in the 6000–16921 range are a mixture of stub (mostly multi-homed) and transit nodes, with large transit nodes (tier-1 providers) of high-degree concentrated toward the tail end. Of course, attacks ventured from these high-degree nodes are inherently difficult for other nodes to detect due to the large spread induced by their high degrees. Figure 28(b) shows ranked ID, sorted by degree, of optimum detectors of the sources ordered by frequency of their optimum detector. We observe that the first 6000 sources covered by high-degree nodes, mixed with intermediate- and low-degree nodes. The general trend also holds for nodes in the 6000–16921 range, but with markedly fewer intermediate- and low-degree nodes acting as optimum detectors. For most sources u , we know that the number of detectors d for which $L_u(d) > 0$ is small, and the detector set for which $L_u(d) \gg 0$ even smaller. As we will see, high-degree nodes tend to be good detectors, which is consistent with the results in Figure 28(b). Since a detector node is assumed to perform both ingress and egress traffic monitoring when carrying out worm detection, a new worm attack ventured from a host within a detector AS will be detected with probability 1 (intra-AS granularity issues such as collaboration between border routers of an AS are considered separately).

Figure 29(a) shows detector nodes ranked by degree and their detection capability in terms of the number of sources for which they are an optimum detector. We observe that high-degree nodes, overall, tend to act as optimum detector nodes for multiple potential attack sources. The top 10 highest-degree nodes collectively cover 6655 attack sources as their optimum detectors, which is about 40% of all nodes. Thus the highest-degree nodes, although important, do not suffice to cover the bulk of potential attack sources. Figure 29(b) shows the size of the cumulative optimum detector set obtained by incrementally adding the optimum detector for attack sources sorted by frequency of optimum detector (i.e., L_u). For the first 6000 attack sources which are

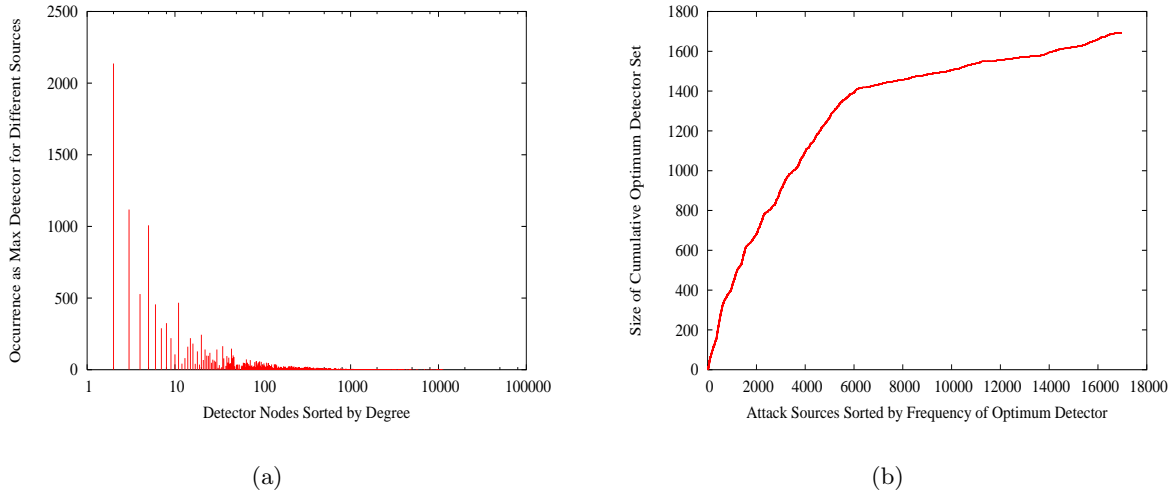


Figure 29: (a) Detectors ranked by degree and their detection capability with respect to acting as maximum detectors for distinct sources. (b) Size of cumulative optimum detector set for attack nodes sorted by frequency of optimum detector.

perfectly detectable by an optimum detector (cf. Figure 27), the cumulative set comprised of their optimum detectors forms a detector set of size 1395. The remaining 10921 potential attack sources are covered by an additional 296 optimum detectors for a total detector set of size 1691. The reason for the “bend in the knee” at around 6000 in Figure 29(b) is due to the higher frequency of sources in the first 6000 sources that are covered by maximum detectors of intermediate- and low-degrees (cf. Figure 28(b)). Intermediate- and low-degree optimum detectors, although best for a specific attack source, tend to be ineffective outside their specific scope. Thus their reusability as good detectors is limited, which leads to a robust climb and increased slope in the 1–6000 interval. For higher degree stub and transit sources in the 6000–16921 range, the frequency of intermediate- and low-degree optimum detectors tends to be significantly reduced—due to the higher degree of freedom of intermediate- and high-degree nodes, it is less likely that smaller degree nodes can do the job of detecting outgoing packets as effectively as higher degree nodes—which leads to a dampened slope.

10.5.2 Influence of Source Address Dispersion

We may define destination- or victim-centric attack detection that captures source address dispersion by “turning the tables around.” For a given destination (i.e., potential victim) $v \in V$, let $M_v(d)$ denote the number of sources u ($u \neq v$) whose path to v goes through detector d . Let

$$M_v = \max_d M_v(d).$$

For a specific node v , M_v quantifies detection power of an optimum detector that catches traffic destined to v from the most number of different sources. Figure 30(a) shows frequency of optimum detector M_v for attack victims v sorted (non-increasing order) by M_v . We also show the corresponding L_u for comparison. Figure 30(a) shows that there is little difference between source and destination address dispersion captured by M_v and L_u . Figure 30(b) shows detection

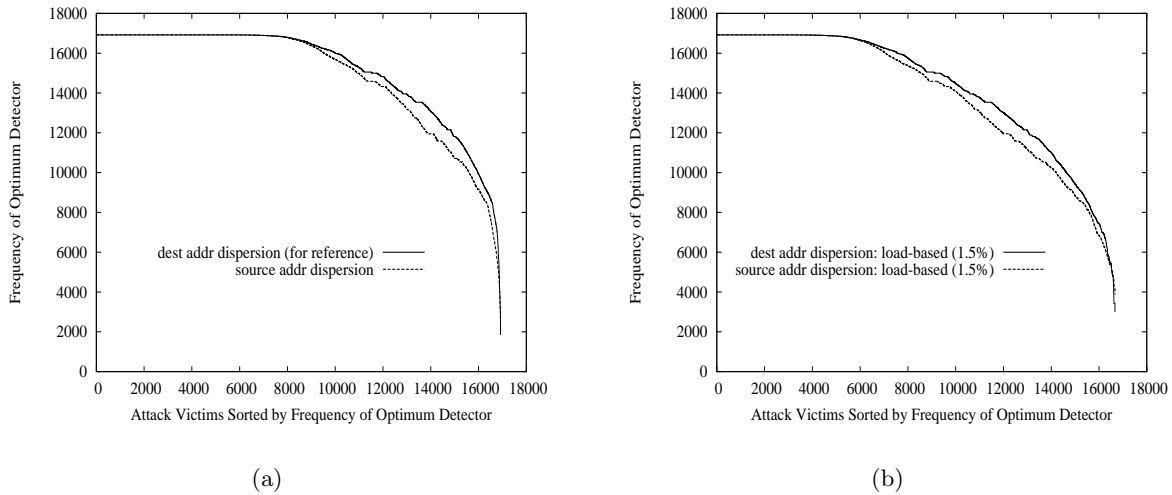


Figure 30: (a) Detection capability of a maximum detector M_v for different victims v sorted by M_v . Corresponding L_u shown for reference. (b) Detection capability of a maximum detector M_v under 1.5% load-based detector placement. Corresponding L_u under 1.5% load-based detector placement shown for reference.

capability under source address dispersion for 1.5% load-based detector placement. We also show the corresponding detection capability under destination address dispersion. We observe only a minor difference between the two. Thus, from a static perspective, destination and source address dispersion have similar quantitative detectability properties.

The picture changes significantly, however, when source and destination address dispersion are viewed from a dynamic perspective. Their fundamental difference, under worm attack, is: assuming zero-day worm attacks are initiated from a few out-of-band compromised attack sites, destination address dispersion, under random scanning, is readily detected by detectors attuned to traffic emanating from the few initial attack sites. Due to the large IP address space, scans are unlikely, during initial ramp up, to repeat, i.e., contain the same destination address. Hence every scan, with high likelihood, contributes to some detector’s destination address dispersion threshold count. This is not the case for source address dispersion. Since the number of initial attack sites is small—an attacker seeks maximum damage using minimum effort through dynamic infection—source address dispersion requires a threshold θ_S number of dynamic infections before packets emanating from θ_S sources can be detected. The need to incur θ_S infection damage before detection can occur injects a period of paralysis during which worm traffic may “fill the pipes”

as in-flight traffic—especially severe in broadband WANs where the delay-bandwidth product is large—laying the grounds for nontrivial residual infection. The performance impact of source address dispersion is discussed in later sections.

10.5.3 Multiple-source Attack Detection: Finding Small Detector Sets

We have seen that picking optimum per-source detectors yields an optimum detector set for multiple-source attacks whose size is 1691 (about 10% of the total number of nodes). To further reduce deployment without significantly sacrificing detection prowess, we consider a number of placement strategies and their detection performance. Figure 31(a) compares optimal detector placement with load-based detector placement (inherited from load-based filter placement) for deployment levels 1, 1.5, 2.5, 3, and 4%. There is a performance gap from optimal to load-based with 4% deployment for sources in the 6000–16921 range, which translates to a decrease of 0.05–0.1 detection probability. Detection performance at 2.5 and 3% deployment nearly overlap with

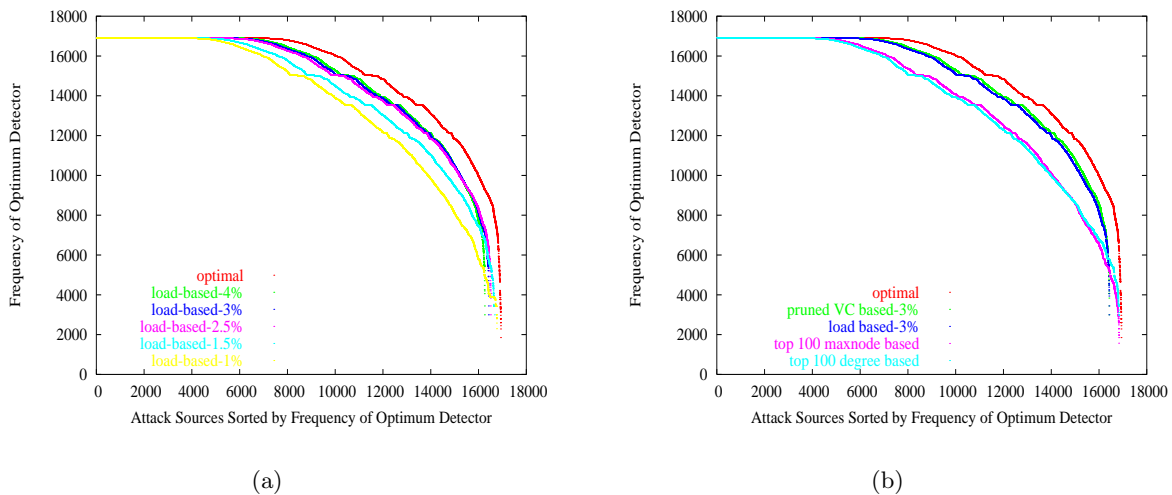


Figure 31: (a) Comparison of optimal and load-based detector placement for 1, 1.5, 2.5, 3, and 4% deployment. (b) Comparison of optimal, pruned VC-based (3%), load-based (3%), top 100 optimum detection capacity-based, and top 100 degree-based detector placement.

that of 4% deployment. Additional incremental drops are observed when decreasing deployment down to 1.5 and 1%. Given the small decrease in absolute detection probability, deployment savings using load-based joint detector and filter placement becomes viable. Figure 31(b) compares different filter placement strategies: optimal detector, load-based (3% deployment), pruned VC-based (3% deployment), top 100 optimum detection capacity-based (Figure 29(a) with detectors resorted by their detection capability), and top 100 degree-based (Figure 29(a)). Pruned VC-based placement yields detection performance almost overlapping with load-based detection performance. “Top 100” rules (e.g., degree-based top 100 advocated for containment in [51]) show

reduced but reasonable performance. Their performance drawbacks are more pronounced under worm filtering when compared with load-based filter placement.

11 Reactive Zero-day Attack Containment

11.1 Scanning, Detection, and Infection Damage

In this work, we track system behavior at multiple scales: one, at host granularity at which scanning and infection occur, two, at AS granularity where there is a many-to-one mapping from hosts to ASes, and third, at pocket granularity where ASes are partitioned into pockets through filter deployment. Although zero-day worm protection is feasible within an intranet (cf. distributed worm filtering in intra-domain topologies), our focus is on the global inter-domain Internet. Hence a single AS, even if only a single host within the AS is infected, is classified as infected. A similarly worst-case damage assessment is imposed at pocket granularity where a pocket is considered infected, even if only a single AS within the pocket is infected. Thus if at some time instance only 1 out of 10 ASes within a pocket is compromised, when tallying the infection damage we will count the cost as 10, reflecting the potential worst-case future spread within the pocket.

11.1.1 Scanning: Epidemiology and Sampling

Let $Q(t)$ denote the number of infected hosts at time t , and let $q(t)$ represent the corresponding fraction of infected hosts given a fixed population size N (i.e., $q(t) = Q(t)/N$). The classic SIR epidemic model [36], which has its origins in the seminal work of Kermack and McKendrick [39], relates susceptibles, infectives, and removed—i.e., the dead or otherwise inert—in a population through a system of nonlinear differential equations. In the simplified SI model where there are no “removed” hosts, assuming a scan rate β (in unit scans-per-second), dynamic infection can be described by

$$\frac{dq(t)}{dt} = \beta(1 - q(t))q(t) \quad (3)$$

which yields the typical “S” curve associated with macroscopic population dynamics (microscopic infection dynamics is described by percolation theory [32]). The differential equation assumes that an infected host infects β uninfected hosts, i.e., $\beta(N - Q(t))$, and since there are $Q(t)$ infected hosts we arrive at the product form (3). Note that due to network delays a time lag is incurred in the update equation, $dq(t)/dt = \beta(1 - q(t - \tau))q(t - \tau)$, where $\tau > 0$ is a fixed delay term (a simplification), which leads to a functional (or delay) differential equation [34]. The impact of time lag is considered separately.

The integral of (3) is given by

$$q(t) = \frac{e^{\beta(t-t_0)}}{1 + e^{\beta(t-t_0)}}, \quad (4)$$

and during the initial infection stage $q(t)$ increases exponentially following $q(t) \approx ce^{\beta t}$ for a sufficiently large host address space N . Furthermore, the total number of scan (i.e., worm)

packets, the number of distinct destination addresses contained in the scan packets, and the total number of infected hosts approximately coincide with $q(t)$. We are ignoring the impact of network delay τ .

11.1.2 Sampling and Detection

A necessary condition for source address dispersion detection with threshold θ_S is $ce^{\beta t} > \theta_S$ which yields the lower bound

$$t > \frac{1}{\beta} \log \theta_S + t_0. \quad (5)$$

Thus source dispersion detection time t decreases inversely proportionally with scan rate β and increases logarithmically slowly with dispersion threshold θ_S . The same holds for destination address dispersion with detection threshold θ_D . Actual detection time is further impeded by the detectors' detection capability. For a given detector set $D \subseteq V$ and initial attacker set $S \subseteq V$, L_u gives the quantitative impact of content prevalence and destination address dispersion on detection time. Let

$$p_{D,S} = \min_{u \in S} \frac{L_u}{N-1}.$$

Then if $Q(t)$ worm packets are generated up to time t , at least $p_{D,S}Q(t)$ packets, on average, will be seen by detectors in D . For worst-case attacks, $p_D = \min_{u \in V} L_u / (N-1)$. Collective detection by group of detectors through information sharing incurs additional costs which are ignored here. Assuming the process $Q(t)$ is rapidly mixing, content prevalence detection occurs when

$$Q(t) > \frac{\theta_D}{p_D}$$

Similarly for destination and source address dispersion detection with thresholds θ_D and θ_S . In EarlyBird, the specific condition is

$$Q(t) > \frac{1}{p_D} (\theta + \max\{\theta_D, \theta_S\}).$$

If dependencies from an initial attacker set S survive, then the average number of scans that are generated before detection occurs may be closer to $(\theta + \theta_D + \theta_S) / p_D$. We will denote the average number of worm packets that are generated before detection Q^* . The impact of Q^* on AS- and pocket-level infection is discussed next.

11.1.3 Sampling and Infection Damage

Worm propagation and infection occur at the individual host level. Hosts, in turn, are grouped into autonomous systems based on the IP address space assigned to an AS. An AS may have unused prefixes in its allocated address space that is used by network telescopes [52] to detect potential anomalous traffic including worm malware. Scans carrying routable but unused destination addresses aid in content prevalence and destination address dispersion detection, but have no

beneficial influence on source address dispersion detection. ASes are further grouped into pockets, which is the granularity at which distributed worm filtering acts to achieve containment.

Let $1, 2, \dots, N$ denote the host address space and let $1, 2, \dots, n$ denote the number of ASes. Thus in the Internet AS graph $G = (V, E)$, $n = |V|$. Given a filter set F , let U_1, U_2, \dots, U_m denote the pockets induced by F . Typically,

$$m < n \ll N.$$

For example, for the 2004 Internet AS topology, $N = 2^{32}$, $n = 16921$, and $m = 11418$ (under 3% filter deployment inclusive filters which form singleton sets). Let A_1, A_2, \dots, A_n represent a partition of the host address space $\{1, \dots, N\}$ corresponding to the n ASes. Little is known about the actual IP address space to AS address space mapping, although the number of routers in an AS exhibits positive correlation with the degree of the AS [75]. When Q^* worm packets are generated before content prevalence, destination address dispersion, and source address dispersion based new worm detection, we are interested in knowing the damage the Q^* packets have exacted at AS granularity. Given partition A_1, \dots, A_n (denote their sizes $a_i = |A_i|$), under random scanning the expected number of hits h_i in AS i is given by

$$h_i = \frac{a_i}{N} Q^*.$$

As discussed earlier, we will say that AS i is infected if one or more hosts in the AS are infected. Our primary metric for tabulating infection damage is the number of ASes infected at the end of the last filter update, inclusive any future damage caused by in-flight worm packets. In fact, our metric is even more conservative than that, considering all ASes within a pocket infected (due to potential future infection) if one or more ASes within the pocket is infected.

At the AS granularity, the larger a_i , the greater the probability that AS i gets infected. The worst-case host address space to AS address space mapping with respect to the total number of infected ASes is when the host address space is equi-partitioned:

$$a_1 = a_2 = \dots = a_n.$$

Under this worst-case host-to-AS mapping, given Q^* random scans, we are interested in how many ASes remain uninfected. This problem can be cast as a classical occupancy problem where r ($= Q^*$) indistinguishable balls are being placed into n cells and we are interested in knowing the probability that k cells are empty. It can be shown [26] that this probability has the form

$$p_k(r, n) = \binom{n}{k} \sum_{i=0}^{n-k} (-1)^i \binom{n-k}{i} \left(1 - \frac{k+i}{n}\right)^r.$$

For large n and small r/n , which is the case of interest to us since the AS topology is large (16921), scanning is in the initial stages (below 500), detection thresholds are small (e.g., $\theta_D = \theta_S = 30$), and detection capability of load-based detectors high (p_D value), $p_k(r, n)$ can be approximated by

$$p_k(r, n) \approx e^{-\mu} \frac{\mu^k}{k!} \quad (6)$$

where $\mu = ne^{-r/n}$. That is, the number of empty cells k , i.e., the number of uninfected ASes, obeys a Poisson distribution. Thus the expected number of uninfected ASes is given by μ , and the expected damage after Q^* scans is given by

$$n\left(1 - e^{-\frac{Q^*}{n}}\right). \quad (7)$$

Figure 32 shows expected AS-level infection damage (i.e., number of infected ASes) as a function of the number of scans for $n = 16921$ and Q^* in the range 1–1000 using (7). For $Q(t)$ values needed to achieve content prevalence, destination and source address dispersion, damage increases almost linearly. Since $Q(t)$ increases exponentially as a function of time during initial ramp up, AS-level infection also increases exponentially with time.

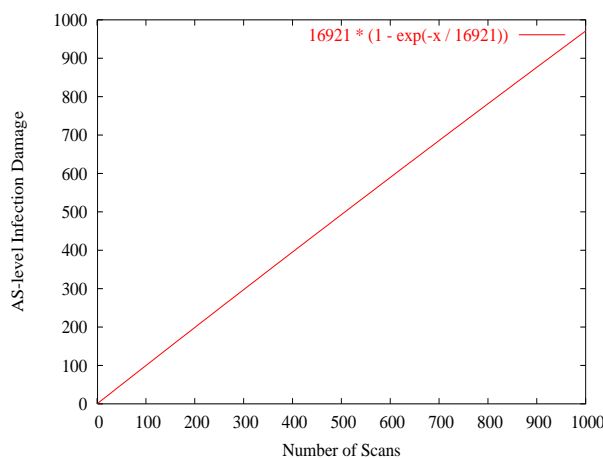


Figure 32: Estimated AS-level infection as a function of the number of scan packets.

The preceding analysis yielded a conservative estimate under the assumption of equi-partitioned IP address space into ASes. Pocket-level damage may be estimated by taking the probability estimate of the number of uninfected ASes, then using the pocket sizes $|U_1|, \dots, |U_m|$ ($m = 11418$ under 3% deployment) to determine additional damage due to intra-pocket infection. Figure 33(a) shows pocket size distribution under 3% load-based detector/filter placement where pockets are ranked by size. Over 90% of the pockets are of size 1, i.e., contain a single AS, with the largest pocket of size 58. Figure 33(b) shows a log scale plot of the abscissa of the same distribution which conveys details of the larger pockets. The amplification factor in infection damage when switching to pocket-level granularity is nontrivial (we omit the calculations here). In a worst-case attack, a smart attacker might selectively target hosts belonging to the IP prefix space of ASes in the largest pockets. For example, the top 10 pockets contain 516 ASes. Hence, conceivably, an attacker could send out 10 worm packets and bring down 516 ASes. Targeting the 100 largest pockets can potentially bring down 2740 ASes; targeting the top 500 pockets can infect 5296 ASes, 31% of all ASes in the 2004 Internet AS topology. Figure 34 shows the cumulative distribution of pocket sizes which is relevant for evaluating worst-case attacks with respect to destination

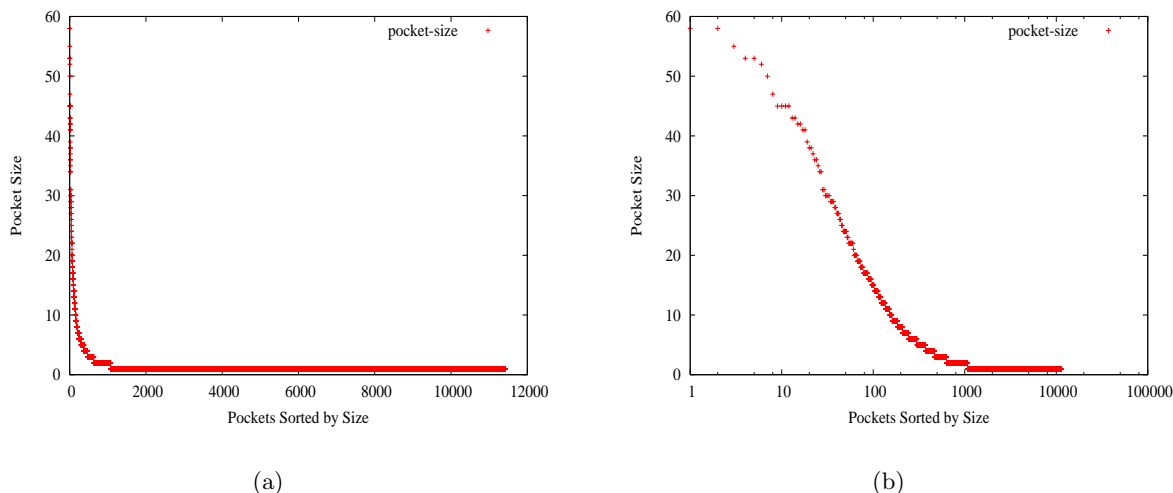


Figure 33: (a) Pocket size distribution under 3% load-based detector/filter placement where pocket IDs are ranked by size. (b) Pocket size distribution under log scale of the x -axis.

address selection. We observe a knick near the 1000 point which stems from the fact that pockets thereafter are mostly singleton sets with a very few of sizes 2 and 3. Of course, individual IP addresses, if selected randomly from the target pockets' collective IP prefix space, may not belong to hosts with the vulnerability that a zero-day worm is exploiting. Due to this and other factors, even in worst-case attacks, the actual number of scans needed to infect an AS within a larger pocket is going to be higher.

11.1.4 Impact of Network Delay and Bandwidth

The preceding scanning, detection, and infection damage results hold when network delay is ignored. The $Q(t)$ process approximately coincides with scanning (i.e., generated worm traffic), destination address diversity (likelihood of destination address repeats is small), host-level infection damage and source address diversity. There is a significant performance difference, however, when the influence of network delay (propagation delay, transmission delay, and queueing delay) and bandwidth are considered: scanning, detection, and infection dynamics remain invariant under content prevalence and destination address dispersion, but change under source address dispersion as key parameter of the underlying network such as scan rate and link latency are varied. We treat the two cases separately.

Case I: Scale-invariance under content prevalence and destination address dispersion. We illustrate the key idea that underlies scale-invariance with respect to detection performance and infection damage under reactive containment through packet filtering using a star topology. High-degree nodes in the Internet AS topology locally resemble a star topology (e.g., many single-homed stub ASes connect through a common provider AS), so the example has also practical relevance.

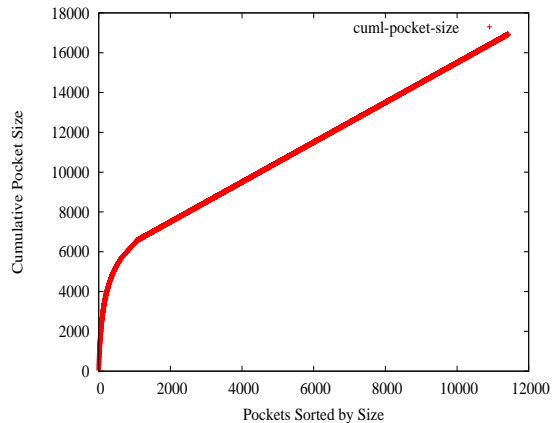


Figure 34: Cumulative pocket size in terms of number of ASes.

We assume that detection—and subsequent reactive filtering—is carried out by the center node in a star topology.

Figure 35(a) shows a 14-node star graph where r is the center node and a, b, \dots, l, m are (stub) nodes dangling from r . The 14-node topology may be a subgraph of a larger network where node m may be a transit node through which external traffic are forwarded. In the following discussion, graph embeddings and related consequences are ignored. We assume a single initial attacker configuration—multiple initial attackers can be handled by a superposition argument—where node a transmits worm packets with distinct destination addresses within the given address space. Our analysis does not require distinct destination addresses, in fact, holds for any addresses (hence holds for random, local, and hybrid scanning). The restriction is for simplicity of exposition. Figure 35(a) shows attack node a transmitting three consecutive packets $a|b$, $a|c$, and $a|d$ where $x|y$ means that a packet originates at x and is destined to y (source address spoofing issues are ignored). We assume equal-sized packets, and bandwidth, link latency, and packet size are such that 3 consecutive packets can be in-flight concurrently. That is, the delay-bandwidth product satisfies

$$\text{delay} \times \text{bandwidth} = \text{const} \quad (8)$$

where the constant, in the example, is 3. Attacker a transmits at the fastest scan rate allowed by its bandwidth, say, 3 scans (i.e., packets) per second. Thus a single packet takes up $1/3$ second, which we will refer to as a tick. We assume that the content prevalence threshold or destination address dispersion threshold are 3 ($\theta = \theta_D = 3$). For simplicity of exposition, we assume that content prevalence checking and destination address dispersion checking are carried out concurrently.

Figure 35(b) shows the network configuration after 3 ticks. Packet $a|b$ is almost at its destination, $a|c$ mid-way, and $a|d$ at the beginning, on their respective final links. After the third packet, $a|d$, passes through router—and detector node— r , destination address dispersion threshold θ_D (and, of course, content prevalence dispersion threshold θ) is reached, i.e., new worm detection has occurred. Assuming the worm filter at r is immediately updated, the next three packets on

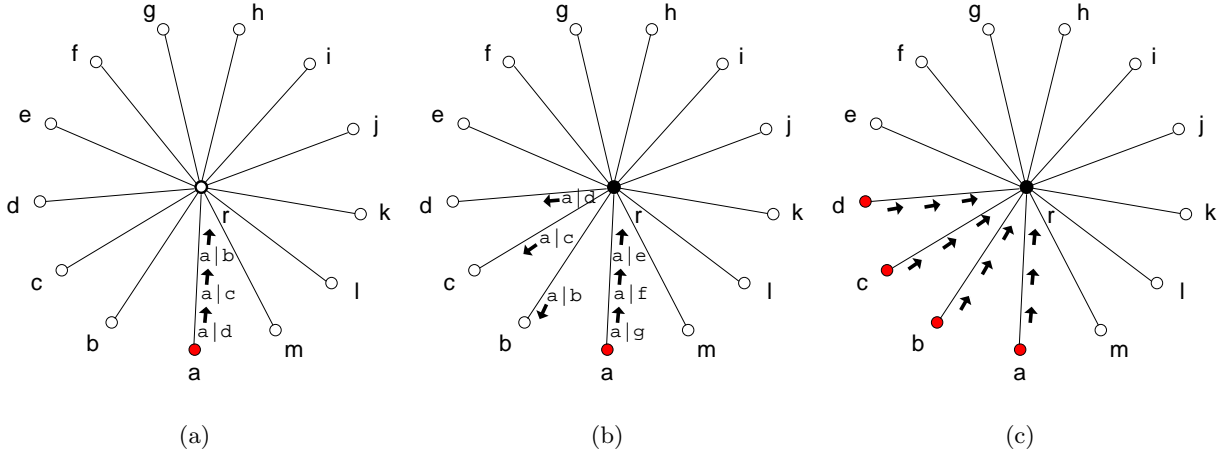


Figure 35: (a) Attacker a sends out 3 packets destined to b , c , and d . (b) With content prevalence and destination address dispersion threshold 3, router r detects worm after seeing the third packet and updates filter at r . Packets destined to b , c , and d have escaped and eventually infect their destinations. The next three packets destined to e , f , and g will get discarded by r . (c) Steady-state configuration with 4 infected nodes.

the wire $a|e$, $a|f$, and $a|g$ will get discarded at r . Figure 35(c) depicts the steady-state network configuration where a total of 4 nodes are infected, including the initial attacker a , but the remaining 11 nodes are shielded from infection. That is, the worm attack and its AS-level damage is contained within 4 singleton pockets $U_1 = \{a\}$, $U_2 = \{b\}$, $U_3 = \{c\}$, and $U_4 = \{d\}$.

For content prevalence and destination address dispersion based worm detection we arrive at the following scale-invariance properties:

- *Scan rate.* For a given delay-bandwidth product, normalized detection time—detection time is tabulated in tick units but normalized detection time is event-driven, i.e., tabulated in units of scan packets—and infection damage stay invariant under different scan rates. Normalized detection time and damage equal θ_D (and θ).
- *Delay and bandwidth.* Normalized detection time and infection damage remain invariant at θ_D under varying link latency. The same holds for bandwidth.
- *Number of initial attack sites.* Normalized detection time and infection damage (here we mean net infections which exclude the initial infected attack sites) remain invariant at θ_D under varying number of initial attack sites.

Scale-invariance of normalized detection time and (net) infection damage is a powerful property that is conducive to zero-day worm attack protection, even though the defense mechanism is reactive in nature. Since optimal joint detector and filter placement predicates co-location of detectors and filters, this is conducive to speedy update of worm filters with newly detected worm

signatures. Destination address dispersion is introduced to combat false positives, a perennial problem of intrusion detection systems. Increasing θ_D (or θ) may further reduce the false positive rate, although results from EarlyBird [69] performance in UCSD’s production network indicate that thresholds need not be large to be effective. Our analysis shows that infection damage equals θ_D , hence increases proportionally with θ_D (or θ) which aids scalable protection.

The preceding results show that reactive zero-day worm attack protection under content prevalence and destination address dispersion worm detection is effective, in the sense of being scale-invariant with respect to key attack and network system parameters such as scan rate, number of initial attack sites, delay, and bandwidth. The reactive cost due to increase in prevalence/dispersion threshold is linear. However, these results need to be qualified by system level details that further modulate protective performance. First, the aforementioned analysis is for a star graph. Although large star graphs are embedded as subgraphs in power-law networks, overall performance cannot be determined from local analysis alone. Our comprehensive dynamic simulation results using DaSSF address this issue. Second, co-location of detector and filter nodes at the granularity of AS topologies does not necessarily imply physical co-location. Even for detection alone, the physical meaning of detector deployment at a transit AS entails, by default, that all border routers of an AS be equipped with detection (and filtering) capabilities. Since border routers are geographically separated—some across continents—delays introduced by their cooperation needs to be incorporated. Third, when a new worm has been detected, generating a signature (in our case a polymorphic worm signature for use in the scalable polymorphic worm filter architecture²) and uploading it incur a time lag. The time incurred by remote signature upload (i.e., a detector multicasts a new worm signature to all filter nodes) is part and parcel of the dynamic network simulation set-up, therefore its effect is incorporated in the performance results. The time lag associated with local signature upload can be captured by δ which represents delay cost above-and-beyond communication cost of remote uploads. In the star topology, additional infection damage due to δ is given by $\lambda\delta$ where λ is the scan rate (in pps unit), assuming bandwidth is sufficiently high to accommodate λ . Thus total net infection is given by

$$\theta_D + \lambda\delta$$

in a single-source attack. In a k -source attack, damage is bounded above by

$$\theta_D + k\lambda\delta.$$

Here we assume that signature upload time δ is smaller than round-trip time. Fourth, content prevalence based worm detection may not be able to keep up with peak packet rates approaching wire speed of today’s 1 Gbps and 10 Gbps access and backbone links. Let ρ denote the fraction of packets missed by the worm detector due to detection overhead which incorporates dependence on traffic rate. The revised infection bound is given by

$$\frac{\theta_D}{1 - \rho} + k\lambda\delta. \quad (9)$$

²In practice, the non-polymorphic signature may be uploaded first, followed by an updated polymorphic signature for timeliness.

Case II: Impact of source address dispersion. Consider a single-source zero-day worm attack in a star topology. Unless the attack source is employing source address spoofing, for source address dispersion at threshold θ_S to be recognized, at a minimum $\theta_S - 1$ hosts need to get infected so that they may transmit scan packets with $\theta_S - 1$ distinct source addresses. This is a fundamental difference compared with destination address dispersion (and content prevalence) detection. Let τ denote the time lag before θ_S hosts get infected and a worm packet from the earliest infected host arrives at center node r . We measure τ from the moment that the first scan packet arrives at r . Let \mathcal{L} denote link latency and \mathcal{B} bandwidth. Let κ denote packet size in bits. In the star topology, $\tau = 2(\mathcal{L} + \kappa/\mathcal{B})$. The expected detection time t^* , measured from the moment when the first scan packet arrives at r , has the upper bound

$$t^* < \tau + \frac{1}{\lambda} \frac{\theta_S}{1 - \rho} = 2\left(\mathcal{L} + \frac{\kappa}{\mathcal{B}}\right) + \frac{1}{\lambda} \frac{\theta_S}{1 - \rho} \quad (10)$$

since packet inter-arrival time is $1/\lambda$ and $\theta_S/(1 - \rho)$ scans must be seen before θ_S is reached. The bound may be tightened a little by noting that infected hosts, with some staggering, transmit scan packets in parallel to r . Given small threshold values, we take a conservative approach and stay with the slightly laxer bound.

The eventual infection damage $\xi(t^*)$ at detection time t^* —eventual infection includes the future infections resulting from in-flight worm packets that have eluded filtering—is comprised of two components: first-order infections arising from worm packets originating at the single-source attacker, and second-order infections stemming from worm packets transmitted by hosts from the first-order infection. We assume that $\theta_S/\lambda(1 - \rho) < \tau$ so that third-order infections do not arise. $\xi(t^*)$ is given by

$$\xi(t^*) = \lambda \left(2\left(\mathcal{L} + \frac{\kappa}{\mathcal{B}}\right) + \frac{1}{\lambda} \frac{\theta_S}{1 - \rho} \right) + \frac{\theta_S}{1 - \rho} = 2\lambda \left(\mathcal{L} + \frac{\kappa}{\mathcal{B}}\right) + \frac{2\theta_S}{1 - \rho} \quad (11)$$

Thus equation (11) shows that infection damage is proportional to the delay-bandwidth product (note that scan rate λ is upper bounded by bandwidth), which dominates actual detection cost $2\theta_S/(1 - \rho)$ incurred by destination address dispersion (and content prevalence) detection.

Additional damage due to signature generation time lag δ is bounded by

$$\delta \lambda \left(2\lambda \left(\mathcal{L} + \frac{\kappa}{\mathcal{B}}\right) + \frac{2\theta_S}{1 - \rho} \right) = 2\delta \lambda^2 \left(\mathcal{L} + \frac{\kappa}{\mathcal{B}}\right) + \frac{2\delta \lambda \theta_S}{1 - \rho}$$

The quadratic dependence of the first term on λ predicates that local signature upload time be small.

Figure 36 illustrates the dynamics underlying scanning, infection, detection, and containment in the star topology. Figure 36(a) depicts the initial single-source attack configuration which is the same as in Figure 35(a) for destination address dispersion. The snapshot shown in Figure 36(b) is the same as in Figure 35(b), except that, unlike destination address dispersion with $\theta_D = 3$, detection has not yet occurred under source address dispersion with $\theta_S = 3$ (the detector node is shown filled when detection has occurred). In Figure 36(c), after three ticks, nodes b , c , and d are infected, and they send out their own scans $b|k$, $c|l$, and $d|m$.

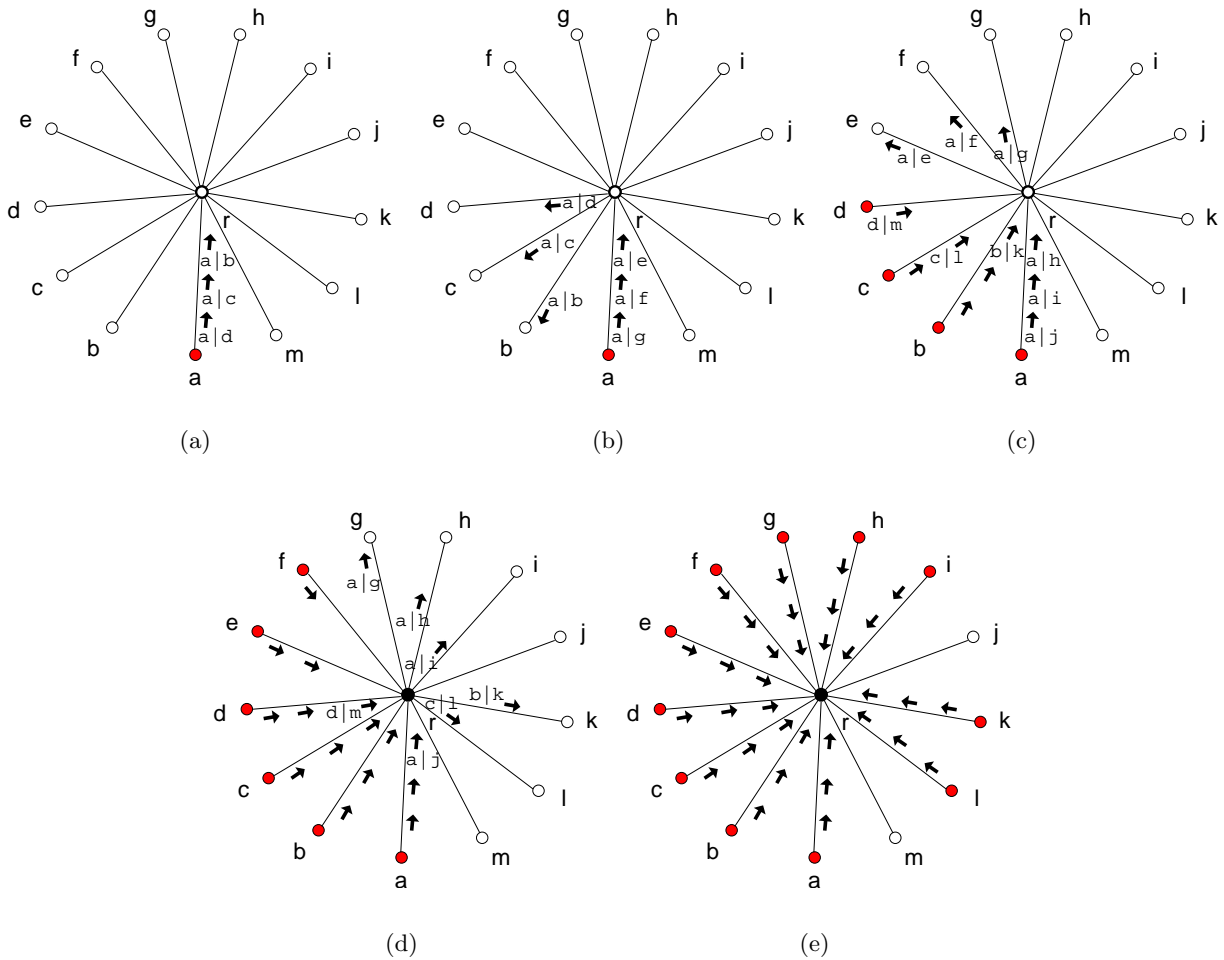


Figure 36: (a) Initial attack configuration. (b) With $\theta_S = 3$, r has not detected the worm after seeing the third packet. (c) b , c , and d are infected, and additional scans $a|e$, $a|f$, and $a|g$ have escaped. Infected hosts b , c , and d send out their own scans $b|k$, $c|l$, and $d|m$. (d) Additional scan packets $a|h$ and $a|i$ originating from a have escaped, and so have worm packets $b|k$ and $c|l$ originating from newly infected hosts b and c . After $c|l$ is encountered, the source address dispersion counter reaches threshold $\theta_S = 3$, and the filter is updated. Subsequent worm packets are discarded. (e) Steady-state configuration.

Additional scans $a|e$, $a|f$, and $a|g$ originating from a are on their way to infecting e , f , and g . In Figure 36(d), packets $b|k$ and $c|l$ have passed through, and so have $a|h$ and $a|i$ (packets $a|i$, $c|l$, and the one following $b|k$ arrive at r “simultaneously”; we assume that $a|i$ is serviced first, followed by $c|l$). The source address dispersion counter reaches θ_S when r sees $c|l$, at which point detection occurs and the filter at r is updated. All subsequent worm packets are discarded by r . In Figure 36(e) shows the steady-state configuration after the “dust has settled.”

The preceding analysis reveals that infection damage before detection, filter update, and containment can take effect increases by $\lambda\tau \propto \mathcal{L} \times \mathcal{B}$ due to the necessity to suffer $\theta_S - 1$ infection before source address dispersion has a chance of being observed. In today’s broadband WANs, reactive cost is dominated by the effective delay-bandwidth product faced by typical attack hosts. An x86 PC with a GigE interface can readily generate traffic at 60000 pps (1500 B packet size). In today’s Internet, most hosts connect via access speeds less than FastEthernet (a typical Internet access link by a large enterprise network is 1 Gbps for the entire community). Broadband home connections (cable modem and DSL) give around 1–2 Mbps uplink speeds. Thus the actual maximum scan rate of a typical host is 2 orders of magnitude smaller, in the ballpark of 600–1000 pps. Coast-to-coast delay (including queueing delay) in the U.S. is well below 100 msec, which gives a single-host effective delay-bandwidth product that limits scan rate at around 60–100 pps. Since in most enterprise and residential networks individual Internet access bandwidth is limited, it is the delay component that dictates reactive infection damage in the zero-day worm attack defense based on cooperative filtering. Thus ignoring the local signature upload latency δ , infection damage increases proportionally to scan rate and network delay, where scan rate is bounded by access bandwidth of typical hosts on the Internet. Of course, if an attacker can compromise a host that has 100 Mbps (or even 1 Gbps) access speed to the global Internet, infection damage can be correspondingly higher by admitting higher scan rates.

11.1.5 Source Address Dispersion Detection: “Less Is More Paradox”

Reactive infection damage as a function of the number of initial attacker k has a subtle feature in that it is non-monotonic in k . For $k \leq \theta_S - 1$, reactive infection damage is given by (11) which is proportional to the delay-bandwidth product. However, at $k = \theta_S$, reactive infection damage (assuming FIFO scheduling at r) is $\theta_S/(1 - \rho)$. Thus for $k \leq \theta_S - 1$ we have the infection damage where scan rate λ in (11) is replaced by the amplified packet rate $k\lambda$ under k -source attack. For $k \geq \theta_S$, reactive infection damage is bounded by

$$\frac{\theta_S}{1 - \rho} + k\lambda\delta \quad (12)$$

Since (11) \gg (12) with λ replaced by $k\lambda$ in (11), we arrive at a “paradox” where even though more attackers are initially recruited for zero-day worm attack, the resultant infection damage is (significantly) less than that achieved with fewer initial attackers. Viewing initial attack sites as “resources” employed by an attacker, an analogy can be made to the Braess paradox [9] where, in certain networks, adding additional network resources in terms of a high bandwidth link can, under selfish routing, result in worse system performance (the paradox is “how can things get worse when resources are more plentiful?”).

The preceding analysis can be extended to more complex networks such as a group of connected star topologies which bear closer resemblance to Internet AS topologies. These are relatively straightforward extensions, and the results are omitted here. Rigorous performance analysis with power-law graphs is in the initial stages of research [4, 28].

11.2 Large-scale Network Simulation Environment

11.2.1 Hardware and Software Set-up

The star topology based analysis of the interplay between scanning, detection, and containment on infection damage is useful to establish basic properties that are relevant for Internet AS topologies where star subgraphs occur as prominent graph embeddings. Nonetheless, Internet AS topologies are complex networks that cannot be accurately approximated by dynamics within a star graph, and we use large-scale network simulation to capture the subtle interactions and dynamics underlying zero-day worm propagation, attack detection, and containment.

We use an extension of DaSSF [73], a parallel/distributed network simulation environment following the SSF framework, which facilitates distributed simulation over PC clusters running MPI. The physical computing platform consists of 24 x86 PCs running Linux 2.4.21 connected by two gigabit Ethernet switches. 5 machines have 4 GB memory each, 5 have 2 GB memory, and 14 machines are configured with 1 GB memories. The (extended) DaSSF protocol stack supports TCP/IP with FIFO output buffered queueing, dynamic routing (BGP based), and application layer programming through a “system call” interface. DaSSF follows a process-centric simulation model (in contrast, *ns* is an event-driven simulator) which makes it well-suited for dynamic performance evaluation. Another distributed simulation platform based on *ns* is *pdns* [56].

11.2.2 Dynamic Network Simulation: Parameter Settings

Table 5 summarizes the parameter settings used in dynamic network simulation, specifying their range and default values, divided into three parts: network system, attack model, and protection system. Other parameters such as infection round-trip delay τ , detection time t^* , filter update time t^{**} , and eventual infection damage ξ are dependent variables of the above parameters that act as independent variables of the system. Detection is done by a detector on an individual basis: no sharing of information among detectors is assumed nor instituted. Signatures are generated by a detector and remote filter updates are carried over the network via signaling packets. A detector multicasts new worm signatures to all filters in the detector/filter group. A single simulation run takes about 20 minutes of wall clock time on the 24 PCs running the distributed network simulator.

11.3 Content Prevalence and Destination Address Dispersion Detection

We first study the interplay and collective time dynamics of worm propagation, detection, containment, and infection damage under content prevalence and destination address dispersion zero-day worm detection.

Network System	Parameter Range [Default]
network topology	1998–2004 Internet AS topologies [2004]
total number of ASes n	3222–16921 [16921]
total number of hosts N	6444000–33842000 [33842000]
host assignment	2000 hosts per AS, degree-based [2000]
link bandwidth	10–1000 Mbps [1000]
inter-AS latency	1–10 msec [1]
intra-AS latency	1–100 msec [0]
routing protocol	shortest-path, other [shortest-path]
buffer size	64 KB [unbounded]
Attack Model	Parameter Range [Default]
type of scanning	random, local, hybrid [random]
scan rate λ	100–10000 scans per sec per host [1000]
infection latency ω	0–100 msec [0]
number of initial infections	1–1000 [10]
placement of initial infections	random, worst-case [random]
Protection System	Parameter Range [Default]
placement of detectors and filters	load-, VC-, detection-based [load-based]
deployment level	1–10% [3]
content prevalence threshold θ	1–100 [3]
destination address dispersion threshold θ_D	1–100 [30]
source address dispersion threshold θ_S	1–100 [30]
packet sampling rate ρ	50–100% [100]
detection and signature generation overhead δ	0–500 msec [0]

Table 5: Parameter settings for dynamic network simulation of zero-day worm attack and protection divided into three parts: network system, attack model, and protection system.

11.3.1 Scale-invariant Containment: Scan Rate

Figure 37(a) shows AS-level infection under varying detector/filter deployment for recurrent vs. zero-day worm attack. We observe a slight upward shift implying that infection damage has increased. To achieve containment at the level of 1.5% filter deployment in recurrent attack, a deployment of 2% is needed under zero-day attack which represents the reactive penalty.

Figure 37(b) shows AS-level infection damage as a function of detector/filter deployment for scan rates 100, 1000, and 10000. Even though scan rate has increased 100-fold, we find that infection damage has remained nearly invariant, consistent with the analysis carried out for the star topology. Thus even though the Internet AS topology is far more complex than a single star topology, the qualitative features gleaned in the latter remain useful in understanding dynamic infection, detection, and containment behavior in the Internet AS topology.

Figure 38(a) shows time evolution of the number of AS infections, detecting detector nodes, and updated filter nodes for 3% detector/filter deployment for scan rate 1000. We observe

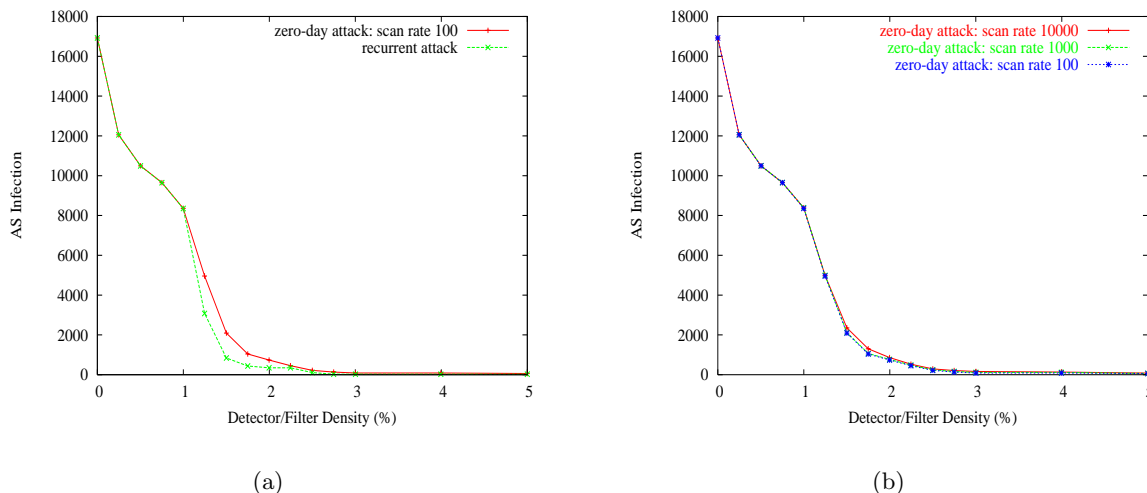


Figure 37: (a) Comparison of infection damage under varying detector/filter deployment for recurrent vs. zero-day worm attack. (b) Comparison of infection damage under varying detector/filter deployment for scan rates 100, 1000, and 10000.

new worm detection at 0.001 sec followed by filter updates that is completed at 0.005 msec. Figure 38(b) shows infection, detection, and containment dynamics for the same system under 1% detector/filter deployment. Detection and filter update is as rapid as under 3% deployment, yielding only 20 AS infections at time 0.02 sec, which is the same as under 3% deployment. If we inspect Figure 37(a), however, we find that infection at 1% deployment is 8358 whereas infection at 3% deployment is only 86. The discrepancy between the two figures is due to fact that we take a conservative approach to estimating infection damage in the following sense: when a single host within an AS is infected, we treat the entire AS as being infected; more importantly, if a single AS within a pocket is infected, we treat the entire pocket as being infected when counting AS infection damage. Hence, in Figure 38(b), although the number of actual ASes infected at time 0.02 sec is only 86, 1% filter deployment has the consequence that the pocket sizes to which the 86 ASes belong to is now significantly bigger than that of 3% filter deployment. In the absence of intra-pocket filtering, we assume that all ASes within the infected pockets eventually get infected when tallying final infection damage.

This is further explicated in Figure 39(a) which shows the number of pockets infected under 1% vs. 3% filter deployment. We observe that the 1% deployment case has fewer infected pockets, consistent with the fact that there are fewer pockets under 1% deployment. Figure 39(b) shows that the opposite is true of the total size of the pockets hereto infected (even if only a single AS within a pocket is infected), in stark contrast to 3% deployment where pocket sizes are significantly smaller.

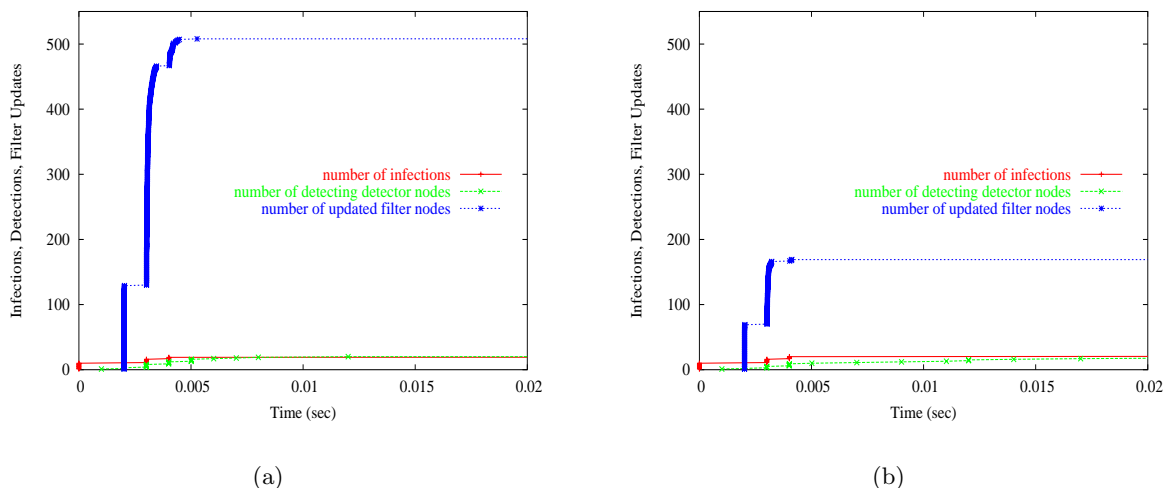


Figure 38: (a) Time evolution of number of AS infections, detecting detector nodes, and updated filter nodes for 3% detector/filter deployment under scan rate 1000. (b) Same for 1% deployment.

11.3.2 Scale-invariant Containment: Delay and Bandwidth

Our star topology analysis indicates that infection damage is invariant with respect to link latency and bandwidth. Figure 40(a) shows that this holds in the 2004 Internet AS topology with respect to link latency as it is varied from 0.1 msec up to 100 msec. Figure 40(b) shows that scale-invariant containment also holds with respect to bandwidth where it is increased from 10 Mbps to 10 Gbps, a 1000-fold increase.

11.3.3 Content Prevalence and Destination Address Dispersion Thresholds

The preceding results confirm that even in the 2004 Internet AS topology that is much more complex than a single star topology, dynamic infection, detection, and containment obey scale-invariance with respect to scan rate, delay and bandwidth, rendering reactive protection against zero-day worm attack a viable option.

We now turn to the impact of content prevalence and destination address dispersion thresholds on containment performance. We note that during the initial infection phase, the destination IP addresses on scan packets are distinct with high probability so we only show containment performance for varying content prevalence threshold. Figure 41(a) shows AS infection under varying detector/filter deployment for content prevalence threshold θ in the range 3–100. As the threshold is increased, we observe an upward shift in infection damage implying higher deployment to achieve a given containment performance. Figure 41(b) shows AS infection as a function of content prevalence threshold under 3% detector/filter deployment. We observe an approximately linear curve, consistent with the containment performance analysis for the star topology. However, there is an important difference in that the slope of the line is about 20 (i.e., infection damage \approx

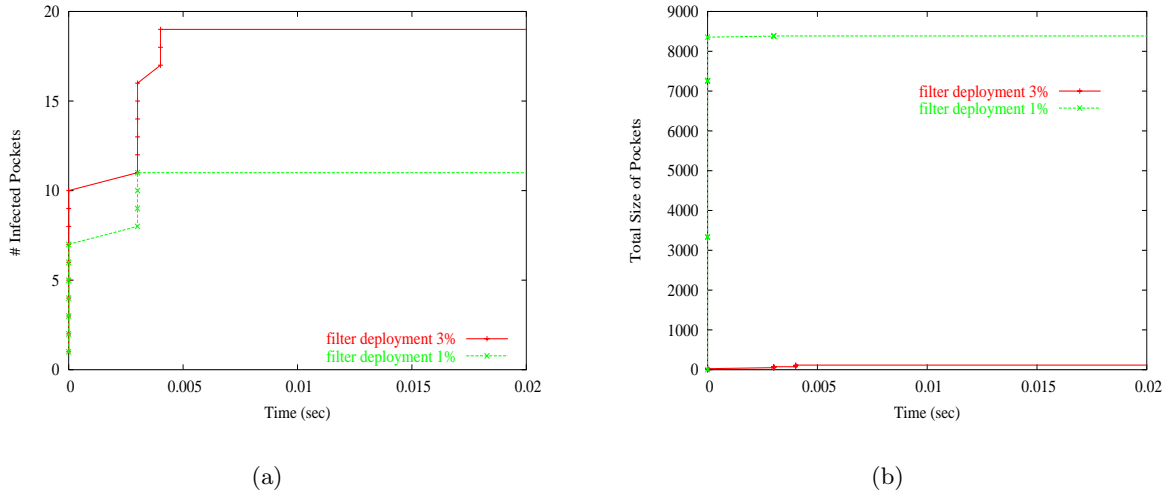


Figure 39: (a) Time dynamics of number of pockets infected under 1% and 3% detector/filter deployment. (b) Time dynamics of total size of infected pockets with respect to number of ASes under 1% and 3% detector/filter deployment.

20θ) which is by a factor of 20 higher than θ . The discrepancy stems from the fact that the Internet AS topology is not a star topology, which introduces additional infection damage stemming from imperfect detection and remote filter upload during which additional worm packets elude the filter net before they are fully updated.

Containment performance is scalable in the sense that infection damage remains linear in θ , irrespective of scan rate and delay-bandwidth product. The same is not true under source address dispersion.

11.3.4 Scan Type: Random, Local, and Hybrid

Our analysis with the star topology shows that containment performance is independent of the type of scanning employed: random, local (or topological), or hybrid. This property, however, is very specific to the star topology since all traffic must go through the center node.

Figure 42 shows that containment performance in the 2004 Internet AS topology remains approximately invariant under random, local, and hybrid scanning. In hybrid scanning, for each worm packet, a coin toss with probability p determines whether the destination address is chosen locally or globally. In Figure 42 hybrid scanning is carried out with $p = 0.5$. Invariance under different types of scanning is due to the powerful detection capability of load-based detectors in power-law networks which admits early detection of global scanning activity as well as local scanning activity. At the granularity of pockets where detectors form a vertex cover of the super-node graph, detection with respect to pocket-level resolution is optimal as discussed above.

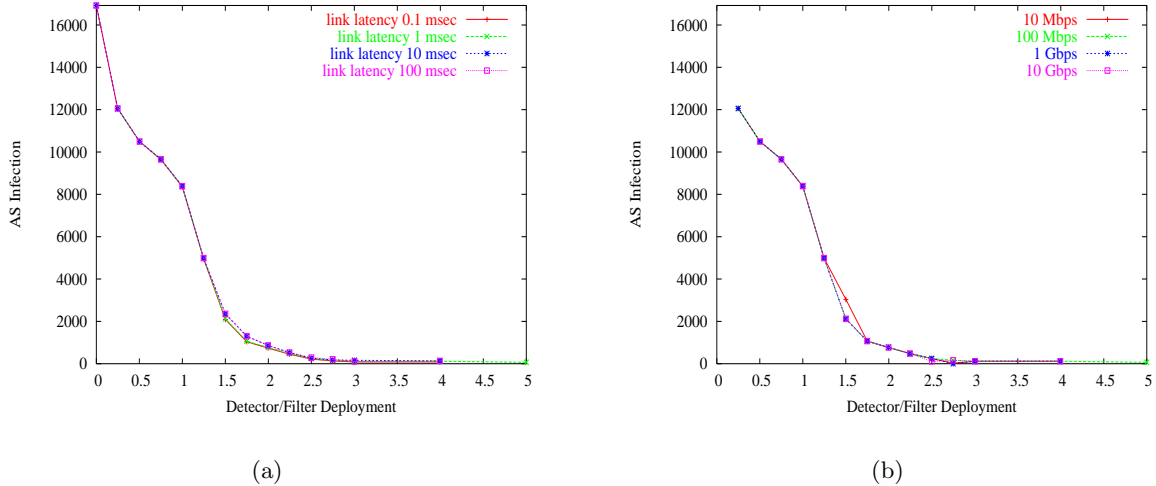


Figure 40: (a) Infection damage under varying detector/filter deployment for link latencies 0.1 msec, 1 msec, 10 msec, and 100 msec. (b) Infection damage under varying detector/filter deployment for link bandwidth 10 Mbps, 100 Mbps, 1 Gbps, and 10 Gbps.

11.3.5 Number of Initial Attackers

Figure 43(a) shows infection damage—total, initial, and net—as a function of the number of initial attackers. Total infection means the total number of ASes infected for a given number of initial attackers. Initial infection tallies the number of AS infections arising from the initially infected ASes (compromised out-of-band), not those arising from dynamic infection. Net infection captures the number of ASes contaminated by dynamic infection, the effect targeted by an attacker. Figure 43(a) shows that the bulk of infection damage is due to initial infection, which is outside the scope of the zero-day worm attack protection system. We observe that net infection damage, after an initial rise up to 100 initial attackers, levels off, staying approximately invariant as the number of initial attackers is increased to 1000.

Figure 43(b) shows infection damage at 3% detector/filter placement with 50 initial attack hosts placed at ASes (one host per AS) in random (ASes are chosen uniformly randomly), “good” (ASes are chosen in the left of 8000 range in Figure 27), and “bad” (ASes are chosen right of 8000) with respect to detectability. We observe that dynamic infection damage is detrimentally impacted by the location of attackers with respect to detectability.

The worst-case scenario for total infection damage arises when initial attackers are positioned at the top pockets, as shown in Figure 34. They can get the most “bang for the buck” since within a pocket there is no containment mechanism that can prevent eventual infection of all ASes belonging to a pocket. Figure 34 shows that for pockets ranked 1–1000 their collective mass increases rapidly, leveling off thereafter.

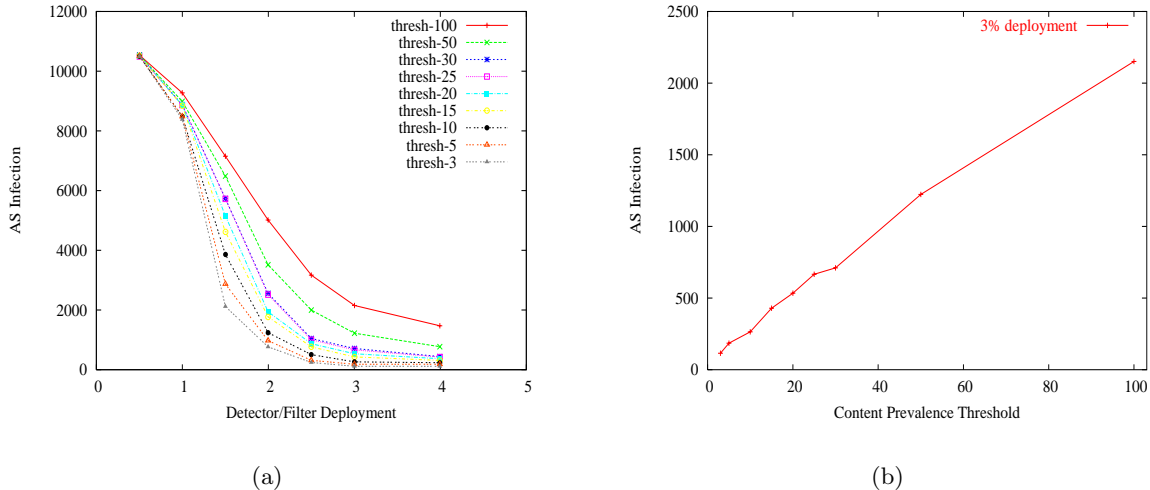


Figure 41: (a) Comparison of infection damage under varying detector/filter deployment for content prevalence thresholds 3, 5, 10, 15, 20, 25, 30, 50, and 100. (b) Infection damage under 3% filter deployment as a function of content prevalence threshold.

11.3.6 Degree-based IP Address to AS Assignment

As indicated in the star topology infection, detection, and containment analysis, the worst-case IP address to AS mapping for dynamic infection is the uniform distribution where each AS has the same number of IP addresses assigned to it. This was the default case studied thus far. Given that IP prefix allocation to ASes is far from uniform, we may consider the additional impact non-uniform address assignment may have on infection, detection, and containment dynamics. As a case in point, we consider an IP address to AS assignment method that apportions IP addresses proportional to the degree of an AS. That is,

$$\alpha_i = \frac{\deg(v_i)}{\sum_{j=1}^n \deg(v_j)} N$$

where α_i is the size of the i 'th AS's IP address space. Although there is evidence that high-degree ASes possess more routers [75], the degree-based address assignment is meant to be used as a comparative reference point, not as an accurate representation of the Internet's IP address allocation.

Figure 44 compares infection damage under uniform vs. degree-based IP address to AS assignment as detector/filter deployment is varied. We observe that there is only marginal difference between the two cases, which reflects the fact that during the initial infection stage where detection and containment have to kick in, the likelihood that an AS is visited twice, even under degree-based assignment is small.

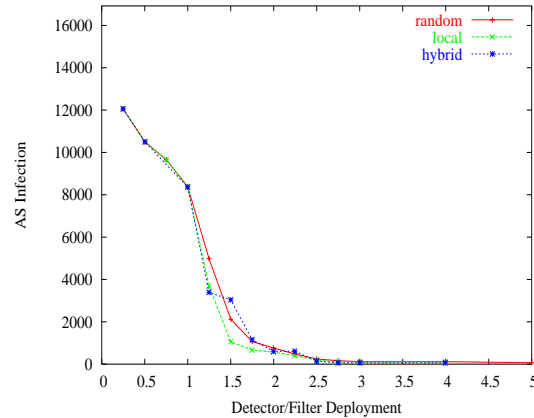


Figure 42: Infection damage under varying detector/filter deployment for random, local, and hybrid scanning.

11.4 Dynamics under Source Address Dispersion

The preceding results focused on infection, detection, and containment dynamics when detection is assumed to be effected by content prevalence and/or destination address dispersion. If effective content prevalence and destination address dispersion detection methods are advanced, then our results show where detectors need to be placed to achieve early detection at the global Internet scale, where to place filters to achieve reactive containment, and how the interplay between infection, detection, and containment evolves yielding scale-invariant protection against zero-day worm attacks.

Our analysis shows that under source address dispersion detection, a delay-bandwidth product reactive penalty is incurred that can significantly increase infection damage. Figure 45(a) compares infection damage for varying detector/filter density under recurrent attack, content prevalence detection ($\theta = 3$), and combined—i.e., content prevalence ($\theta = 3$), destination address dispersion ($\theta_D = 30$), and source address dispersion ($\theta_S = 30$)—detection. Content prevalence detection with threshold $\theta = 3$ exacts a small reactive penalty vis-à-vis the recurrent attack case. The intermediate case comprised of content prevalence ($\theta = 3$) and destination address dispersion ($\theta_D = 30$) only adds another penalty. The most significant shift results under combined detection with $\theta = 3$, $\theta_D = 30$, and $\theta_S = 30$. Containment is still effected but a significantly higher reactive penalty is incurred due to infection damage increasing proportionally with the delay-bandwidth product. Figure 45(b) compares infection damage under combined detection for scan rates 100, 1000, and 10000. Unlike in the content prevalence and destination address dispersion cases where infection remains invariant with respect to scan rate, in source address dispersion increase in scan rate results in increased infection damage. We note that the infection metrics are conservative in the sense that we classify an AS as infected even if a single host within the AS is infected, and we treat all ASes within a pocket as infected even if only one AS is infected.

To deal with the source address dispersion problem which injects a reactive penalty pro-

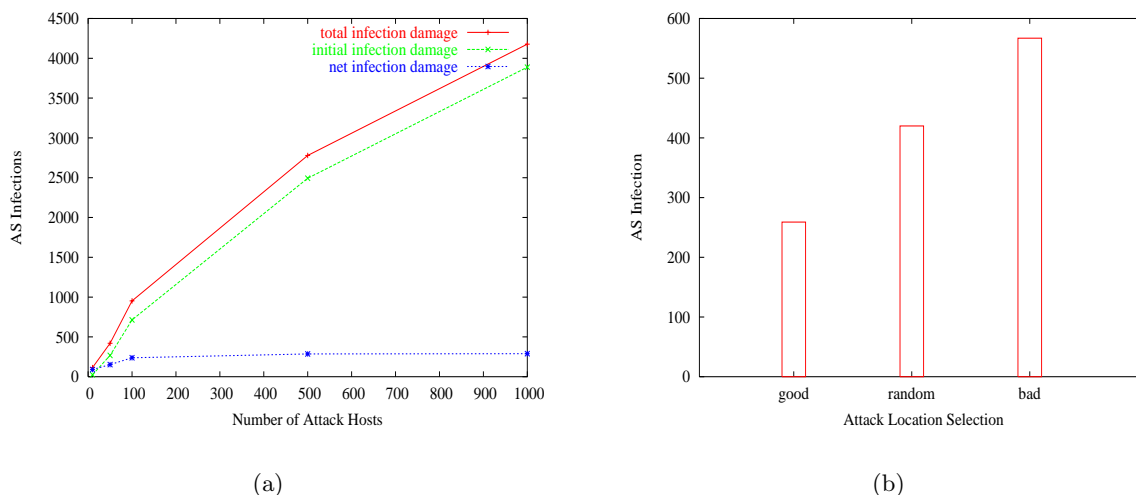


Figure 43: (a) Infection damage—total, initial, and net—under varying number of initial attack hosts, each located in a separate AS. (b) Infection damage at 3% detector/filter deployment with 50 initial attack hosts placed (at the granularity of ASes) in random, “good,” or “bad” locations.

portional to the delay-bandwidth product of a network (i.e., translating to maximum scan rate achievable in such a network), we have advanced an improved protection architecture that uses detector honeypots co-located with content prevalence, destination and source address dispersion detectors which yields significantly improved performance over the combined case without honeypot detection. These results are part of on-going effort.

12 Related Work

A range of methods and tools exist for protecting against worm attacks, including software patching [23, 64], firewalls [17, 83], intrusion prevention systems [66, 68, 72], content-based worm filtering at end systems [79], static program checking [18, 46, 67], run-time monitoring/management of worm exploits [20, 27], and anti-worms [14, 29]. With the exception of anti-worms, these solutions protect a local system—individual host or enterprise network—from outside malware inclusive worms. An overview of recent trends and taxonomy can be found in [40, 80]. End system filtering of known worms has been studied in [79], which addresses the slow deployment of software patches, in particular, those stemming from technically motivated reasons. By implementing worm filtering using WinSock2 LSP, deployment issues and costs resulting from disruption, unreliability, and irreversibility associated with kernel-level DLL updates are mitigated, which alleviates many of the technically grounded concerns. However, non-technical barriers such as awareness and laxity remain, leading to partial deployment that curtail the effectiveness of host-based protection. Anti-worms, which find their spiritual root in Robert Morris Sr.’s Darwin computer game from the late 1960s [29], act analogously to anti-bodies in biological immune systems. They

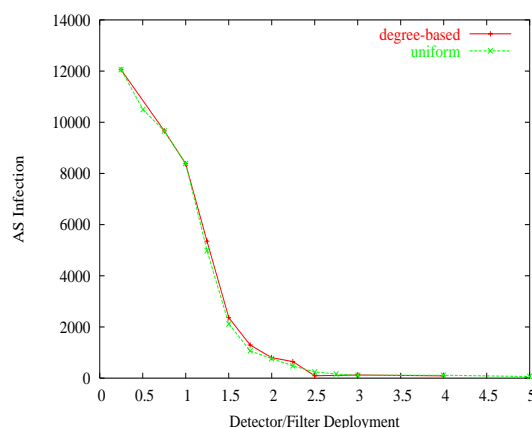


Figure 44: Infection damage under varying detector/filter deployment for uniform vs. degree-based IP address to AS assignment.

represent an instance of non-local protection, engaging in a global competition whose dynamics can be analyzed via multi-type extensions of the SIR epidemiology model [6, 36]. CodeGreen and CRclean were devised as anti-worms to CodeRed I & II in September 2001, but were not actually released. Welchia immunized a host system against Blaster by downloading its patch, however, the resultant bandwidth and server load—factors not considered in classical epidemiology models—overwhelmed many network systems, causing more harm than good. A study of anti-worm generation and their effectiveness is provided in [14].

An integral part of worm attack prevention is worm attack detection. Recent research has focused on detecting various forms of malware activity, including worm and DDoS attacks, in the broader scope of assessing the composition and impact of network anomalies spanning malware attacks, flash crowds, route instability, and network outages [7, 43, 47, 58, 62]. A simple, yet elegant and innovative idea in the detection arena is spoofed DDoS attack detection using backscatter [53] which aims to discern malware attack from traffic activity in unused but routable address spaces. Since then, this idea, in the extended form of network telescopes [52], has been applied in a variety of contexts including detecting Internet worm activity through active honeypots [58]. Other detection methods include passive honeypots [42, 63] and execution analyses, from system call tracing to run-time stack behavior analysis [20, 27]. A promising advent in worm detection is automated signature generation for unknown worms [41, 69] that exploit content invariance and address dispersion to identify new worms. Although the idea of detecting and automatically generating signatures for malware is not new [38, 78], the speed and effectiveness of automated worm detection and signature generation by EarlyBird [69] in UCSD’s production network is promising.

Packet filters, proposed in [50] in the context of a stack-based virtual machine abstraction, have varied uses including application-specific demultiplexing in operating systems [24, 50], traffic engineering [33, 44], network monitoring [3, 61], among others. Although research varies along different dimensions—e.g., interpreter-based filtering [48, 50], pattern-based matching [5], rule op-

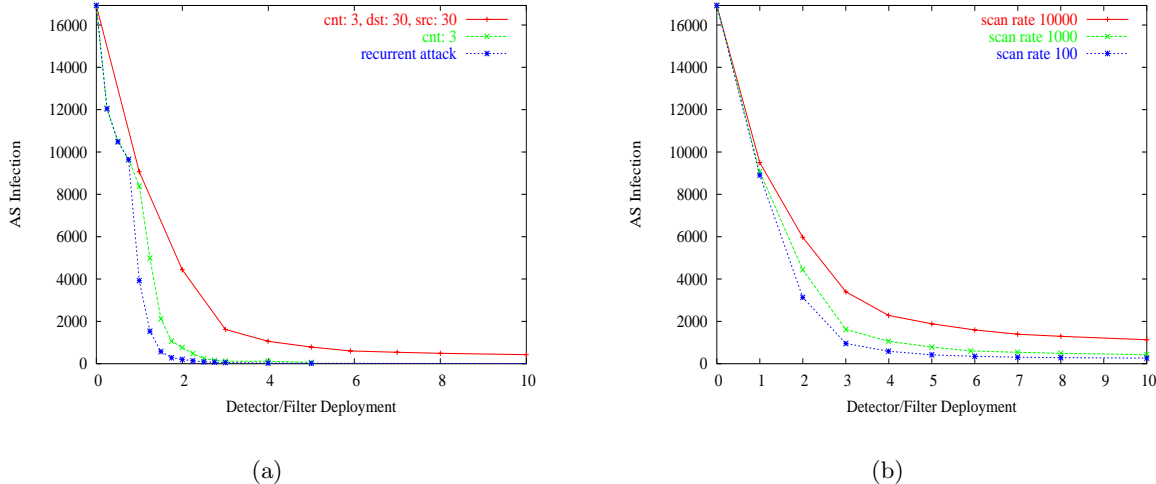


Figure 45: (a) Comparison of infection damage for varying detector/filter density under recurrent attack, content prevalence detection ($\theta = 3$), and combined—content prevalence ($\theta = 3$), destination address dispersion ($\theta_D = 30$), source address dispersion ($\theta_S = 30$)—detection. (b) Infection damage under combined detection for scan rates 100, 1000, and 10000.

timization [8, 33], and dynamic code generation [24]—all have tried to maintain general-purpose filtering capability with degrees of expressiveness that stand in trade-off with performance. Domain specificity provides some, but not deciding, restraints on the input space, requiring system support for thousands of filter rules. Thus whereas recursive flow classification [33] aims to optimize a rule tree using structure in ISP provided classifiers and their rules, in worm filtering the goal is to achieve a compact rule tree by exploiting protocol embedding and length invariance of polymorphic worms that result in a small signature set. That is, one that is of sufficiently small cardinality such that scalable implementation in network processor platforms [19] is feasible.

13 Conclusion

Our research has yielded several significant results that advance the science and engineering implication of protection inter-domain and intra-domain IP networks from recurrent (i.e., known) and zero-day (i.e., unknown) worm attacks. The two principal results are: (i) advancing an architecture for scalable polymorphic worm filtering, implementing it in a gigabit network processor environment, and using accurate and comprehensive benchmarking to establish that gigabit wire speed worm filtering is feasible; (ii) advancing a cooperative filter architecture for protecting against zero-day worm attacks which tackles the optimal zero-day worm attack detection problem when content-based worm detection is based on content prevalence, destination and source address dispersion, and using large-scale network simulation to establish the effectiveness of reactive protection against zero-day worm attacks using cooperative filtering.

There is significant on-going work, some of which are mentioned in the report and others have been excluded due to time constraints. Principal among them is combating source address dispersion in zero-day attack detection and protection, and Internet workload-based performance evaluation of polymorphic worm filtering. Technical papers from this research is in submission and under preparation.

14 Bibliography

- [1] W. Aiello, F. Chung, and L. Lu. A random graph model for massive graphs. In *Proc. ACM STOC '00*, pages 171–180, 2000.
- [2] Gene Amdahl. Validity of the single-processor approach to achieving large scale computing capabilities. In *AFIPS Conference Proc.*, pages 483–485, 1967.
- [3] K. Anagnostakis, S. Ioannidis, S. Miltchev, J. Ioannidis, M. Greenwald, and J. Smith. Efficient packet monitoring for network management. In *Proc. 8th IFIP/IEEE NOMS*, pages 423–436, 2002.
- [4] B. Armbruster, C. Smith, and K. Park. A packet filter placement problem with application to defense against spoofed denial of service attacks. *To appear in European Journal of Operational Research*, 2005.
- [5] M. Bailey, B. Gopal, M. Pagels, L. Peterson, and P. Sarkar. PathFinder: A pattern-based packet classifier. In *Proc. USENIX OSDI '94*, pages 115–123, 1994.
- [6] Norman Bailey. *The Mathematical Theory of Infectious Diseases*. Oxford University Press, New York, 1987.
- [7] P. Barford and D. Plonka. Characteristics of network traffic flow anomalies. In *Proc. ACM Internet Measurement Workshop*, pages 69–73, 2001.
- [8] A. Begel, S. McCanne, and S. Graham. BPF+: Exploiting global data-flow optimization in a generalized packet filter architecture. In *Proc. ACM SIGCOMM '99*, pages 123–134, 1999.
- [9] Dietrich Braess. Über ein paradoxon aus der verkehrsplanung. *Unternehmensforschung*, pages 258–268, 1969.
- [10] A. Broder and M. Mitzenmacher. Network applications of Bloom filters: A survey. In *Proc. 40th Annual Allerton Conference*, pages 636–646, 2002.
- [11] A. Broido and kc claffy. Internet topology: connectivity of IP graphs. In *Proc. SPIE Scalability and Traffic Control in IP Networks*, pages 172–187, 2001.
- [12] CAIDA. Analysis of fragmented packet traffic, 2000. <http://www.caida.org/analysis/workload/fragments>.

- [13] CAIDA. skitter, 2002. <http://www.caida.org/tools/measurement/skitter>.
- [14] F. Castaneda, E. Sezer, and J. Xuy. WORM vs. WORM: Preliminary study on active counter-attack mechanism. In *Proc. ACM Workshop on Rapid Malcode*, pages 83–93, 2004.
- [15] H. Chang, S. Jamin, and W. Willinger. Inferring AS-level Internet topology from router-level path traces. In *Proc. SPIE Scalability and Traffic Control in IP Networks*, pages 196–207, 2001.
- [16] D. Chess and S. White. An undetectable computer virus. In *Virus Bulletin Conference*, 2000.
- [17] W. Cheswick and S. Bellovin. *Firewalls and Internet Security*. Addison-Wesley Pub. Co., 1994.
- [18] M. Christodorescu and S. Jha. Static analysis of executables to detect malicious patterns. In *Proc. USENIX Security Symposium*, pages 169–186, 2003.
- [19] Douglas Comer. *Network Systems Design using Network Processors*. Pearson Prentice Hall, 2004.
- [20] C. Cowan, C. Pu, D. Maier, H. Hinton, J. Walpole, P. Bakke, S. Beattie, A. Grier, P. Wagle, and Q. Zhang. StackGuard: automatic adaptive detection and prevention of buffer-overflow attacks. In *Proc. USENIX Security Symposium*, 1998.
- [21] C. Cowan, P. Wagle, C. Pu, S. Beattie, and J. Walpole. Buffer overflows: attacks and defenses for the vulnerability of the decade. In *Proc. DARPA Information Survivability Conference and Exposition (DISCEX)*, pages 119–129, 2000.
- [22] N. Desi. Increasing performance in high speed nids : A look at snort’s internals, 2002. <http://www.snort.org>.
- [23] J. Dunagan, R. Roussev, B. Daniels, A. Johnson, C. Verbowski, and Y. Wang. Towards a self-managing software patching process using black-box persistent-state manifests. In *Proc. International Conference on Autonomic Computing*, pages 106–113, 2004.
- [24] D. Engler and F. Kaashoek. DPF: Fast, flexible message demultiplexing using dynamic code generation. In *Proc. ACM SIGCOMM ’96*, pages 53–59, 1996.
- [25] M. Faloutsos, P. Faloutsos, and C. Faloutsos. On power-law relationships of the Internet topology. In *Proc. ACM SIGCOMM ’99*, pages 251–262, 1999.
- [26] William Feller. *An Introduction to Probability Theory and Its Applications*, volume I. John Wiley & Sons, third edition, 1968.
- [27] H. Feng, O. Kolesnikov, O. Fogla, and W. Lee. Anomaly detection using call stack information. In *Proc. IEEE Symp. on Security and Privacy*, 2003.

- [28] A. Ferrante, G. Pandurangan, and K. Park. Complexity of combinatorial optimization in power law graphs. Preprint, 2005.
- [29] P. Ferrie, F. Perriot, and P. Ször. Worm wars. *Virus Bulletin*, October 2003.
- [30] M. Garey and D. Johnson. *Computers and Intractability*. W. H. Freeman and Company, 1979.
- [31] R. Govindan and H. Tangmunarunkit. Heuristics for Internet map discovery. In *Proc. IEEE INFOCOM '00*, pages 1371–1380, 2000.
- [32] Geoffrey Grimmett. *Percolation*. Springer-Verlag, 2nd edition, 1999.
- [33] P. Gupta and N. McKeown. Packet classification on multiple fields. In *Proc. ACM SIGCOMM '99*, pages 147–160, 1999.
- [34] J. Hale and S. Verduyn Lunel. *Functional Differential Equations*. Springer-Verlag, 1993.
- [35] Garrett Hardin. The tragedy of the commons. *Science*, 162:1243–1248, 1968.
- [36] Herbert Hethcote. The mathematics of infectious diseases. *SIAM Review*, 42(4):599–653, 2000.
- [37] K2. Admmutate, 2004. <http://www.ktwo.ca/security.html>.
- [38] J. Kephart and W. Arnold. Automatic extraction of computer virus signatures. In *Proc. Virus Bulletin International Conference*, pages 178–184, 1994.
- [39] W. Kermack and A. McKendrick. A contribution to the mathematical theory of epidemics. *Proc. Roy. Soc. Lond. A*, 115:700–721, 1927.
- [40] D. Kienzle and M. Elder. Recent worms: A survey and trends. In *Proc. ACM Workshop in Rapid Malcode*, pages 1–10, 2003.
- [41] H. Kim and B. Karp. Autograph: Toward automated, distributed worm signature detection. In *Proc. USENIX Security Symposium*, pages 271–286, 2004.
- [42] C. Kreibich and J. Crowcroft. Honeycomb—creating intrusion detection signatures using honeypots. In *Proc. HotNets-II*, 2003.
- [43] A. Lakhina, M. Crovella, and C. Diot. Diagnosing network-wide traffic anomalies in traffic flows. In *Proc. ACM SIGCOMM '04*, pages 219–230, 2004.
- [44] T.V. Lakshman, , and D. Stiliadis. High-speed policy-based packet forwarding using efficient multi-dimensional range matching. In *Proc. ACM SIGCOMM '98*, pages 203–214, 1998.
- [45] L. Li, D. Alderson, W. Willinger, and J. Doyle. A first-principles approach to understanding the Internet's router-level topology. In *Proc. ACM SIGCOMM '04*, pages 3–14, 2004.

- [46] R. Lo, K. Levitt, and R. Olsson. MCF: a malicious code filter. *Computers & Security*, 14(6):541–566, 1995.
- [47] B. Madhusudan and J. Lockwood. Design of a system for real-time worm detection. In *Proc. HotI-12*, pages 77–83, 2004.
- [48] S. McCanne and V. Jacobson. The BSD packet filter: A new architecture for user-level packet capture. In *Proc. Winter USENIX Technical Conference '93*, pages 259–269, 1993.
- [49] Mercator Internet AS Map. Courtesy of Ramesh Govindan, USC/ISI, 2002.
- [50] J. Mogul, R. Rashid, and M. Accetta. The packet filter: An efficient mechanism for user-level network code. In *Proc. ACM SOSP '87*, pages 39–51, 1987.
- [51] D. Moore, C. Shannon, G. Voelker, and S. Savage. Internet quarantine: Requirements for containing self-propagating code. In *Proc. IEEE INFOCOM '03*, 2003.
- [52] D. Moore, C. Shannon, G. Voelker, and S. Savage. Network telescopes. Technical Report CS2004-0795, CSE Department, UCSD, 2004.
- [53] D. Moore, G. Voelker, and S. Savage. Inferring Internet denial-of-service activity. In *Proc. USENIX Security Symposium*, pages 9–22, 2001.
- [54] National Laboratory for Applied Network Research. Routing data, 2000. Supported by NSF, <http://moat.nlanr.net/Routing/rawdata/>.
- [55] William Norton. Co-Founder and Chief Technical Liaison, Equinix. Personal communication.
- [56] Georgia Tech Parallel/Distributed NS. pdns 2.1b7a. Available in <http://www.cc.gatech.edu/computing/compass/pdns/>, February 2001.
- [57] D. Dagon O. Kolesnikov and W. Lee. Advanced polymorphic worms: Evading ids by blending in with normal traffic, 2004. http://www.cc.gatech.edu/ok/w/ok_pw.pdf.
- [58] R. Pang, V. Yegneswaran, P. Barford, V. Paxson, and L. Peterson. Characteristics of Internet background radiation. In *Proc. ACM Internet Measurement Conference*, pages 27–40, 2004.
- [59] K. Park and H. Lee. On the effectiveness of route-based packet filtering for distributed DoS attack prevention in power-law internets. In *Proc. ACM SIGCOMM '01*, pages 15–26, 2001.
- [60] K. Park and W. Willinger, editors. *Self-Similar Network Traffic and Performance Evaluation*. Wiley-Interscience, 2000.
- [61] V. Paxson. Bro: A system for detecting network intruders in real-time. In *In 7th USENIX Security Symposium, San Antonio, TX, USA*, volume 25, pages 172–180, 1982.

- [62] Vern Paxson. Bro: a system for detecting network intruders in real-time. In *Proc. USENIX Security Symposium*, 1999.
- [63] Niels Provos. Honeyd Virtual Honeyd, 2003. <http://www.honeyd.org/>.
- [64] Eric Rescorla. Optimal time to patch revisited, 2004. Working Paper.
- [65] RIPE. Routing Information Service Raw Data, 2002. <http://data.ris.ripe.net>.
- [66] Christopher Rouland. Defining the rules of preemptive protection: the ISS intrusion prevention system. White Paper.
- [67] M. Schultz, E. Eskin, E. Zadok, and S. Stolfo. Data mining methods for detection of new malicious executables. In *Proc. IEEE Symposium on Security and Privacy*, pages 178–184, 2001.
- [68] R. Sekar and V. Uppuluri. Synthesizing fast intrusion prevention/detection systems from high-level specifications. In *Proc. USENIX Security Symposium*, 1999.
- [69] S. Singh, C. Estan, G. Varghese, and S. Savage. Automated worm fingerprinting. In *Proc. ACM/USENIX OSDI '04*, pages 45–60, 2004.
- [70] Skitter Project. Caida, 2001. <http://www.caida.org/tools/measurement/skitter/>.
- [71] N. Spring, R. Mahajan, and D. Wetherall. Measuring ISP topologies with Rocketfuel. In *Proc. ACM SIGCOMM '02*, pages 133–145, 2002.
- [72] T. Sproull and J. Lockwood. Wide-area hardware-accelerated intrusion prevention systems (WHIPS). In *Proc. IWAN '04*, 2004.
- [73] Dartmouth SSF. Dassf 3.1.5. Available in <http://www.cs.dartmouth.edu/~jasonliu/projects/ssf/>, August 2001.
- [74] Y. Malcom T. Detristan, T. Ulenspiegel and M. v. Underduk. Polymorphic shellcode engine using spectrum analysis. <http://www.phrack.org/show.php?p=61&a=9>.
- [75] H. Tangmunarunkit, J. Doyle, R. Govindan, S. Jamin, W. Willinger, and S. Shenker. Does AS size determine AS degree? *Computer Communication Review*, 31(5), 2001.
- [76] University of Michigan. AS Graph Data Sets, 2002. <http://topology.eecs.umich.edu/data.html>.
- [77] University of Oregon. Oregon Route Views, 2000. <http://www.routeviews.org/> and <http://archive.routeviews.org/>.
- [78] J. Voas and J. Payne. A model for detecting the existence of unknown computer viruses in real-time. In *Proc. 5th International Computer Virus & Security Conference*, 1992.

- [79] H. Wang, C. Guo, D. Simon, and A. Zugenmaier. Shield: vulnerability-driven network filters for preventing known vulnerability exploits. In *Proc. ACM SIGCOMM '04*, pages 194–204, 2004.
- [80] N. Weaver, V. Paxson, S. Staniford, and R. Cunningham. A taxonomy of computer worms. In *Proc. ACM Workshop in Rapid Malcode*, pages 11–18, 2003.
- [81] J. Winick and S. Jamin. Inet-3.0: Internet Topology Generator. Technical Report CSE-TR-456-02, Department of EECS, University of Michigan, 2002.
- [82] Mihalis Yannakakis. Node- and edge-deletion NP-complete problems. In *Proc. ACM STOC '78*, pages 253–264, 1978.
- [83] E. Zwicky, S. Cooper, D. Chapman, and D. Ru. *Building Internet Firewalls*. O'Reilly & Associates, Inc., 2nd edition, 2000.