Outline

- Morning session (understanding)
  - The 10,000 foot issues
  - Overview and taxonomy
  - Worm history
  - Epidemiological modeling
- Afternoon session (defenses)
  - **Overview**
  - Detection
    - Signature-based
    - Behavioral
  - Mitigation

UCSDCSE Computer Science and Engineering

ICSI INTERNATIONAL COMPUTER SCIENCE INSTITUTE

Recap: how to think about outbreaks

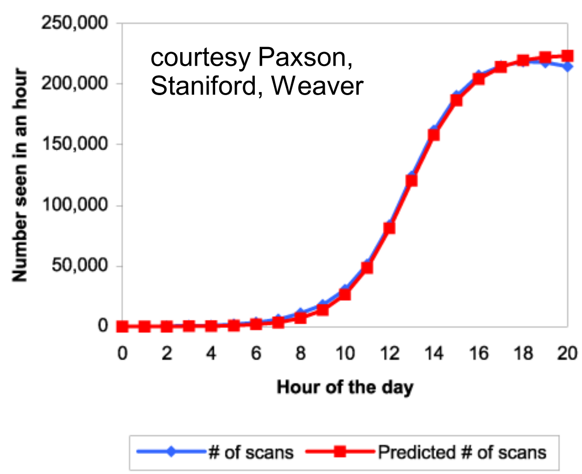
- Worms well described as infectious epidemics
  - Simplest model: Homogeneous random contacts
- Classic SI model
  - N: population size
  - S(t): susceptible hosts at time t
  - I(t): infected hosts at time t
  - ß: contact rate
  - i(t): I(t)/N, s(t): S(t)/N

$$\frac{dI}{dt} = \beta \frac{IS}{N}$$

$$\frac{dS}{dt} = -\beta \frac{IS}{N}$$

$$\Rightarrow \frac{di}{dt} = \beta i(1-i)$$

$$i(t) = \frac{e^{\beta(t-T)}}{1+e^{\beta(t-T)}}$$



What's important?

- We primarily care about two things
- How **likely** is it that a given infection attempt is successful?
  - Target selection (random, biased, hitlist, topological,…)
  - Vulnerability distribution (e.g. density – S(0)/N)
- How **frequently** are infections attempted?
  - ß: Contact rate

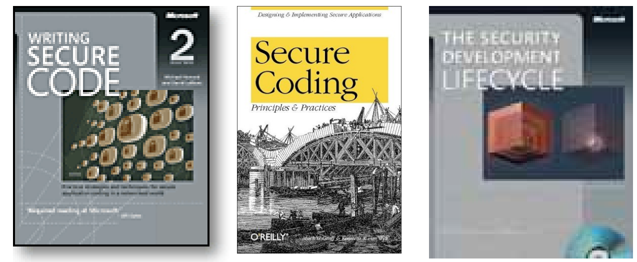
UCSD CSE Computer Science and Engineering INTERNATIONAL COMPUTER SCIENCE INSTITUTE

What can be done?

- Reduce the number of susceptible hosts
  - **Prevention**, reduce S(t) while I(t) is still small (ideally reduce S(0))
    - Software quality, wrappers, artificial heterogeneity, patching, known exploit blocking, hygiene enforcement
- Reduce the number of infected hosts
  - **Recovery**, reduce I(t) after the fact
    - Clean up
- Reduce the contact rate
  - **Containment**, reduce ß while I(t) is still small

Prevention: Software Quality

- **Goal:** eliminate vulnerability



- Software process, code review, etc.
  - Taken seriously in industry
  - Security code review *alone* for Windows Server 2003 ~ $200M
- Static/dynamic testing (e.g. Cowan, Wagner, Engler, etc)
  - **Active** research and industrial development
  - Traditional problems: soundness, completeness, usability

coverity

FORTIFY
_security->starts here

- Practical problems: scale and cost

Prevention: Mitigations

- **Goal**: make it harder to exploit vulnerability

- Exploit detection
  - Stack overflow: NX, Stackguard, /GS, ProPolice, etc
  - Heap overflows: heap cookies, robust link/unlink

- Artificial software heterogeneity
  - PaX, ALSR, code/data polymorphism, pointer encryption

- System call Sandboxing
  - BSD Jail, GreenBorders

Prevention: Software Updating

- **Goal:** reduce window of vulnerability

- Many (most?) exploits target known vulnerability
  - Window shrinking: automated patch ⇒ exploit
  - Patch deployment challenges, downtime, **Q/A**, etc
  - Rescorla*, Is finding security holes a good idea?*, WEIS '04

- Known vulnerability filtering: address Q/A issue
  - Decouple "patch" from code
    - E.g. TCP packet to port 1434 and > 60 bytes
  - Wang et al, *Shield: Vulnerability-Driven Network Filters for Preventing Known Vulnerability Exploits*, SIGCOMM '04
  - TippingPoint, Symantec, etc…

Prevention: Known Exploit Blocking

- Get early samples of new exploit
  - Network sensors/honeypots
  - "Zoo" samples
- Anti-virus/IPS company distills "signature"
  - Labor intensive process
- Signature pushed out to all customers
- Host recognizer checks files/memory before execution
  - Much more than grep… polymorphism/metamorphism

needs short reaction window

- Example: Symantec
  - Gets early intelligence via managed service side of business and DeepSight sensor system
  - >60TB of signature updates per day

Prevention: Hygiene Enforcement

- **Goal**: keep susceptible hosts off network
- Only let hosts connect to network if they are "well cared for"
  - Recently patched, up-to-date anti-virus, etc…
  - Manual version in place at some organizations (e.g. NSF)
- Cisco: Network Admission Control (NAC)
  - Lots of other vendors now in the space

UCSDCSE Computer Science and Engineering
ICSI INTERNATIONAL COMPUTER SCIENCE INSTITUTE

Recovery

- Reduce I(t) after the outbreak is done
  - Practically speaking, this is where much happens because our defenses are so bad
- **Two issues**
  - How to detect infected hosts post hoc?
    - They still spew traffic (commonly true, but poor assumption)
      - Ma et al, "*Self-stopping Worms*", WORM '05
    - Look for known signature (malware detector)
      - Problems with rootkits, etc…
  - What to do with infected hosts?
    - Wipe whole machine
    - Custom disinfector (need to be sure you get it all…backdoors)
    - Opportunities for virtualization (checkpoint/rollback)
    - Aside: interaction with SB1386 in California

Containment

- **Goal**: Reduce infection rate
- **Slow down**
  - Throttle connection rate to slow spread
    - Twycross & Williamson, *Implementing and Testing a Virus Throttle*, USENIX Sec ’03
    - Version used in some HP switches
  - Important capability, but worm still spreads…
- **Quarantine**
  - Detect and characterize worm
    - Network-level vs. host level
  - Block future spreading
    - Behavior or signature blocking in network or on host
    - Automated patch creation: Sidiroglou et al, *Building a Reactive Immune System for Software Services*, USENIX ‘05
    - Anti-worms: Castaneda et al, *Worm vs WORM: Preliminary Study of an Active counter-Attack Mechanism*, WORM ‘04

Containment requirements to protect the Internet [MSV+03]

- We can define reactive defenses in terms of:
  - ***Reaction time*** – **how long** to detect, propagate information, and activate response
  - ***Containment strategy*** – **how** malicious behavior is identified and stopped
  - ***Deployment scenario*** - **who** participates in the system

- Given these, what are the engineering requirements for **any** effective defense?

[MSV+03] Moore, Shannon, Voelker & Savage, **Internet Quarantine: Requirements for Containing Self-Propagating Code**, Infocom 2003

Methodology

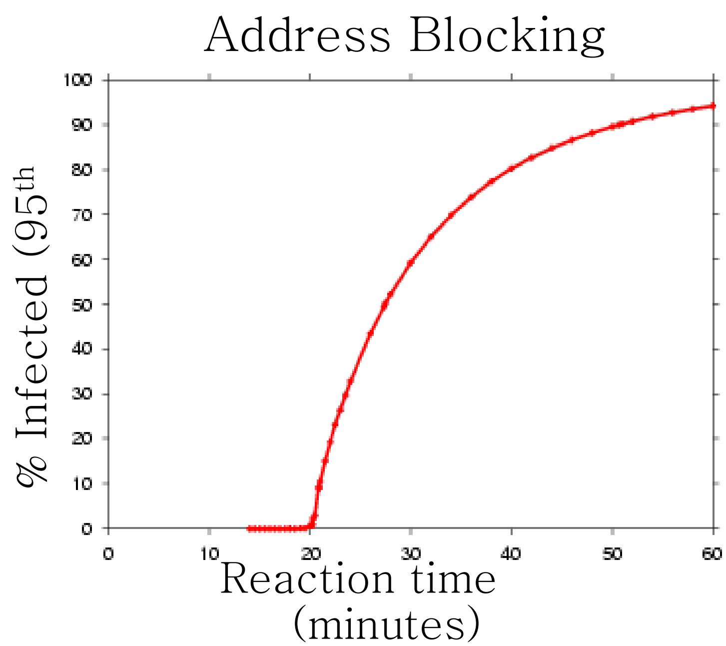
- **Simulate spread of worm across Internet topology**
  - Infected hosts *attempt* to spread at a fixed rate (probes/sec)
  - Target selection is uniformly random over IPv4 space
- **Source data**
  - Vulnerable hosts: 359,000 IP addresses of CodeRed v2 *victims*
  - Internet topology: AS routing topology derived from RouteViews
- **Simulation of defense**
  - System detects infection within reaction time
  - Subset of network nodes employ a containment strategy
- **Evaluation metric**
  - % of vulnerable hosts infected in 24 hours
  - 100 runs of each set of parameters (95th percentile taken)
    - Systems must plan for reasonable situations, **not** the average case

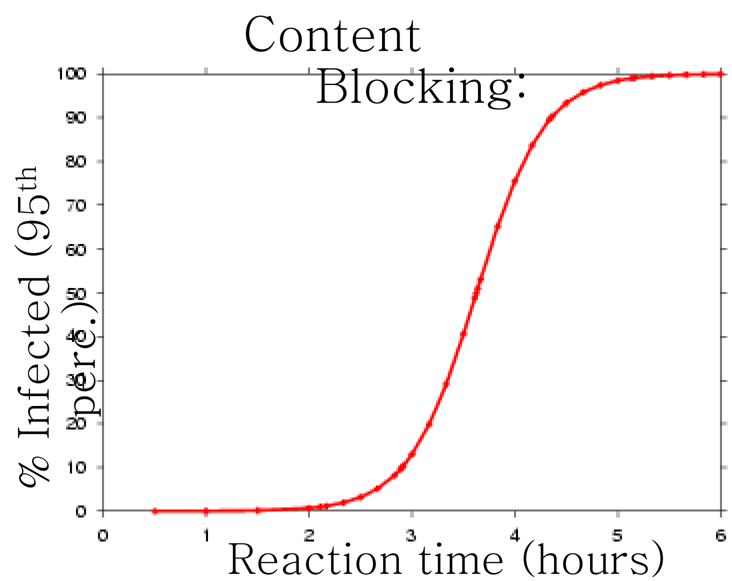
Naïve model: Universal deployment

- Assume **every host** employs the containment strategy

- Two containment strategies :
  - ***Address blocking:***
    - Block traffic from malicious source IP addresses
    - Reaction time is relative to each infected host
    - **MUCH** easier to implement
  - ***Content blocking:***
    - Block traffic based on signature of content
    - Reaction time is from first infection
- How quickly does each strategy need to react?
- How sensitive is reaction time to worm probe rate?

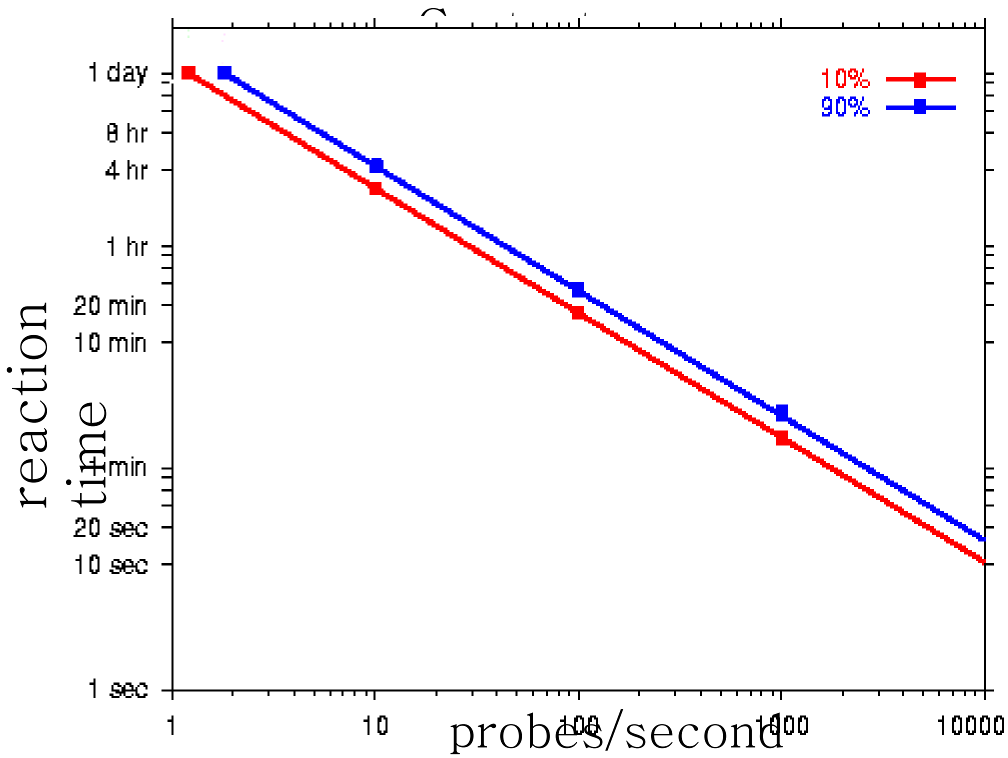
How quickly does each strategy need to react?





- To contain worms to 10% of vulnerable hosts after 24 hours of spreading at 10 probes/sec (CodeRed-like):
  - **Address blocking**: reaction time must be < 25 minutes.
  - **Content blocking**: reaction time must be < 3 hours

How sensitive is reaction time to worm probe rate?



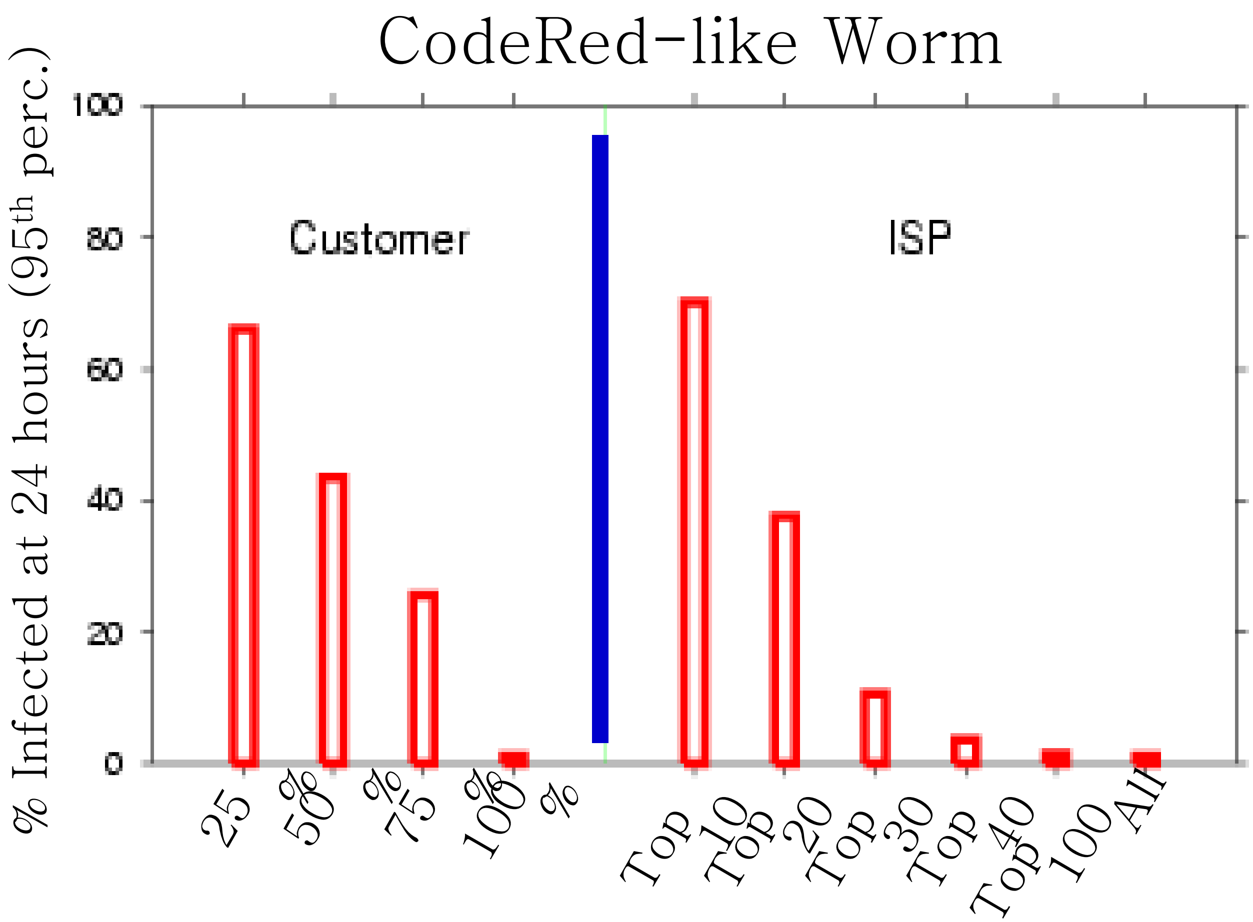
- Reaction times must be fast when probe rates get high:
  - **10 probes/sec: reaction time must be < 3 hours**
  - **1000 probes/sec: reaction time must be < 2 minutes**

UCSDCSE Computer Science and Engineering
INTERNATIONAL COMPUTER SCIENCE INSTITUTE

Limited network deployment

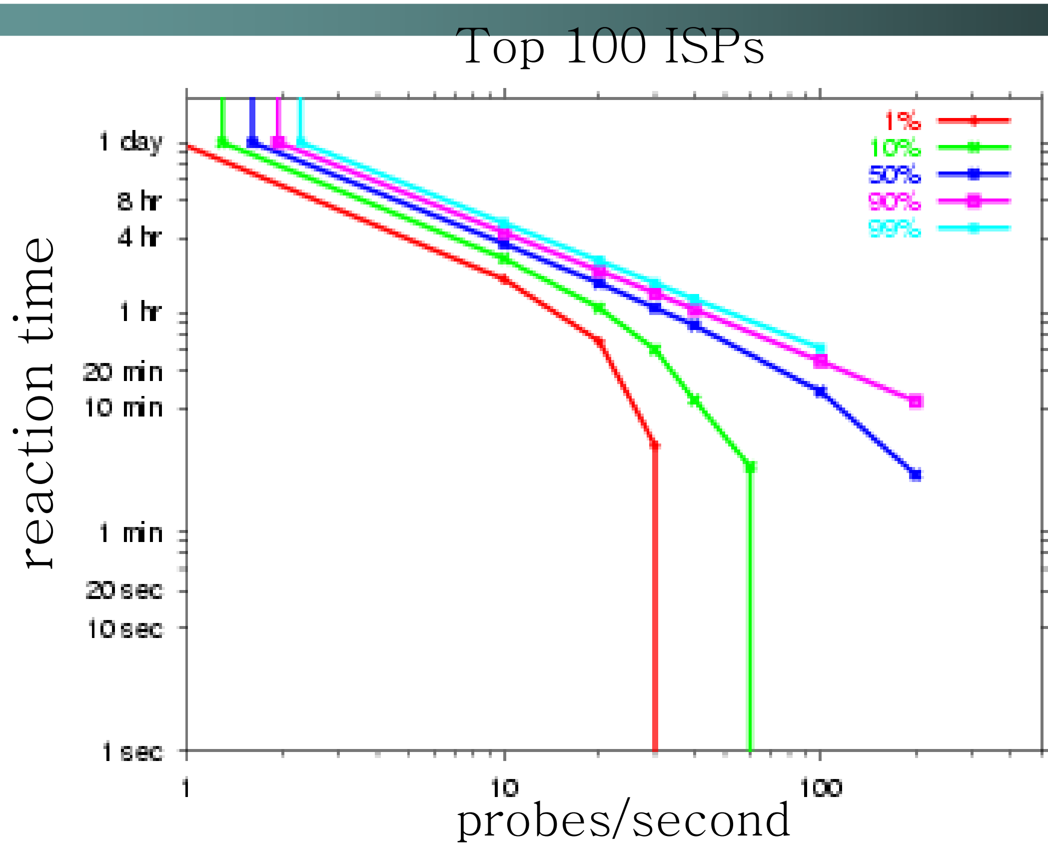
- Depending on **every** host to implement containment is probably a bit optimistic:
  - Installation and administration costs
  - System communication overhead
- A more realistic scenario is **limited** deployment in the **network**:
  - Customer Network: firewall-like inbound filtering of traffic
  - ISP Network: traffic through border routers of large transit ISPs
- How effective are the deployment scenarios?
- How sensitive is reaction time to worm probe rate under limited network deployment?

How effective are the deployment scenarios?



UCSDCSE Computer Science and Engineering
ICSI INTERNATIONAL COMPUTER SCIENCE INSTITUTE

How sensitive is reaction time to worm probe rate?



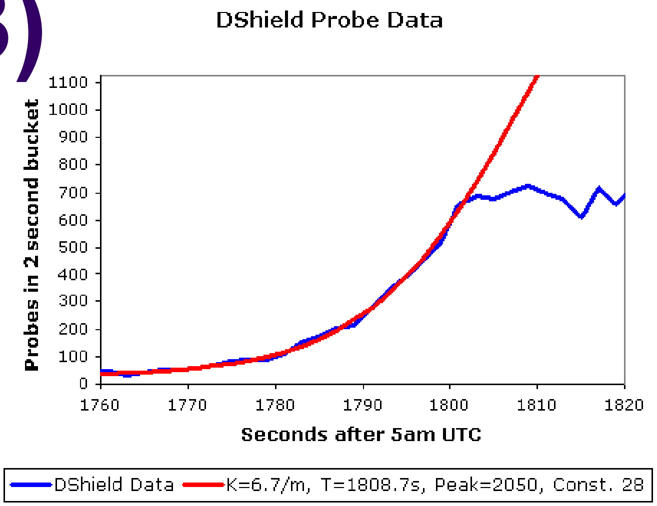
- Above 60 probes/sec, containment to 10% hosts within 24 hours is ***impossible*** for top 100 ISPs even with ***instantaneous*** reaction.

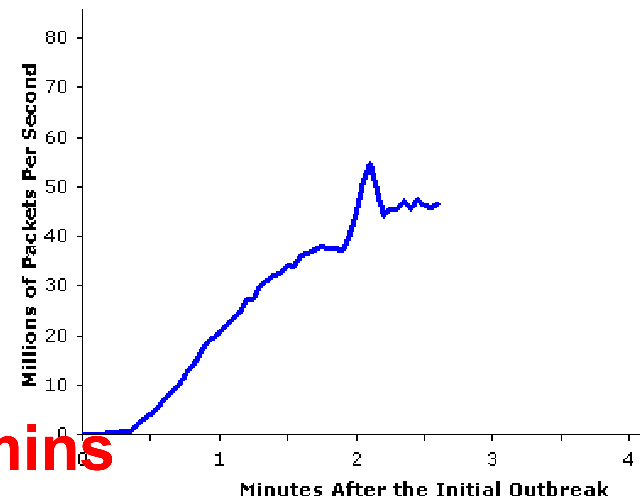
Bottom line: its difficult…

- Even with universal defense deployment, containing a CodeRed-style worm (10 pps) (<10% in 24 hours) is tough
  - Address filtering (blacklists), must respond < 25mins
  - Content filtering (signatures), must respond < 3hrs
- Gets proportionally worse as worms get faster
- For non-universal deployment, life is still worse
- Containing a fast worm seems to require responding in seconds or less!
  - Bottom line: way faster than people
- Chicken Little or real threat?

Recap: Slammer (2003)

- First ~1min behaves like classic random scanning worm
  - Doubling time of ~8.5 seconds
  - CodeRed doubled every 40mins
- >1min worm starts to saturate access bandwidth
  - Some hosts issue >20,000 scans per second
  - Self-interfering (no congestion control)
- Peaks at ~3min
  - **>55million IP scans/sec**
- **90% of Internet scanned in <10mins**
  - Infected ~100k hosts (conservative)





[MPS+03] Moore, Paxson, Savage, Shannon, Staniford &Weaver, **The spread of the sapphire/slammer worm**, IEEE Security & Privacy, 1(4), 2003

UCSDCSE Computer Science and Engineering
ICSI INTERNATIONAL COMPUTER SCIENCE INSTITUTE

Was Slammer really fast?

- **Yes**, it was orders of magnitude faster than CR
- **No**, it was poorly written and unsophisticated
- **Who cares?** It is *literally* an academic point
  - The current debate is whether one can get < 500ms
  - **Bottom line**: way faster than people!

edge of all systems vulnerable to the worm's exploit. In previous work we suggested that a flash worm could saturate one million vulnerable hosts on the Internet in under 30 seconds [18]. We grossly over-estimated.

The Top Speed of Flash Worms

Stuart Staniford, Nevis Networks · David Moore, CAIDA · Vern Paxson, ICSI · Nicholas Weaver, ICSI

ABSTRACT

Flash worms follow a precomputed spread tree using prior knowledge of all systems vulnerable to the worm's exploit. In previous work we suggested that a flash worm could saturate one million vulnerable hosts on the Internet in under 30 seconds [18]. We grossly over-estimated.

In this paper, we revisit the problem in the context of single packet UDP worms (inspired by Slammer and Witty). Simulating a flash version of Slammer, calibrated by current Internet latency

Keywords

worms, simulation, modeling, Flash worm

1. INTRODUCTION

Since Code Red in July 2001 [11], worms have been of great interest in the security research community. This is because worms can spread so fast that existing signature-based anti-virus and intrusion-prevention defenses risk being irrelevant; signatures can-

[SMP+04] Staniford, Moore, Paxson & Weaver, **The Top Speed of Flash Worms**, ACM WORM, 2004

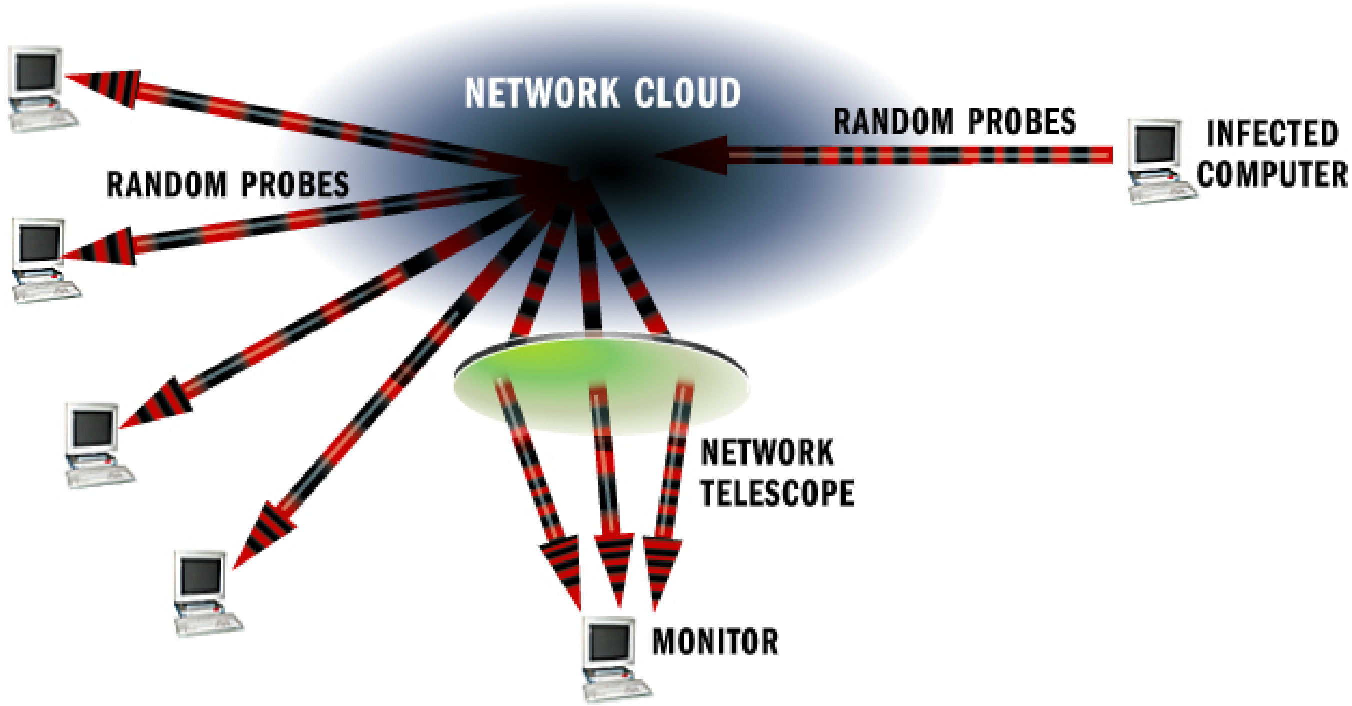
Outline

- Morning session (understanding)
  - The 10,000 foot issues
  - Overview and taxonomy
  - Worm history
  - Epidemiological modeling
- Afternoon session (defenses)
  - Overview
  - **Detection**
    - **Signature-based**
    - Behavioral
  - Mitigation

Detecting new worms

- Where to detect
  - *In situ* (production hosts or bump-on-wire)
    - Pro: see attacks on *your* network/systems
    - Con: noise in data stream, performance impact on hosts
  - *Ex situ* (honeypots/telescopes/darknets)
    - Pro: clean environment (no one should talk to you)
    - Cons: someone has to talk to you
- How to detect
  - Signature-oriented vs behavior-oriented (fuzzy distinction at times)

Recap: Network Telescopes



- Infected host scans for other vulnerable hosts by randomly generating IP addresses
- Network Telescope: monitor large range of unused IP addresses – will receive scans from infected host
- Very scalable. CCIED monitors 17M+ addresses

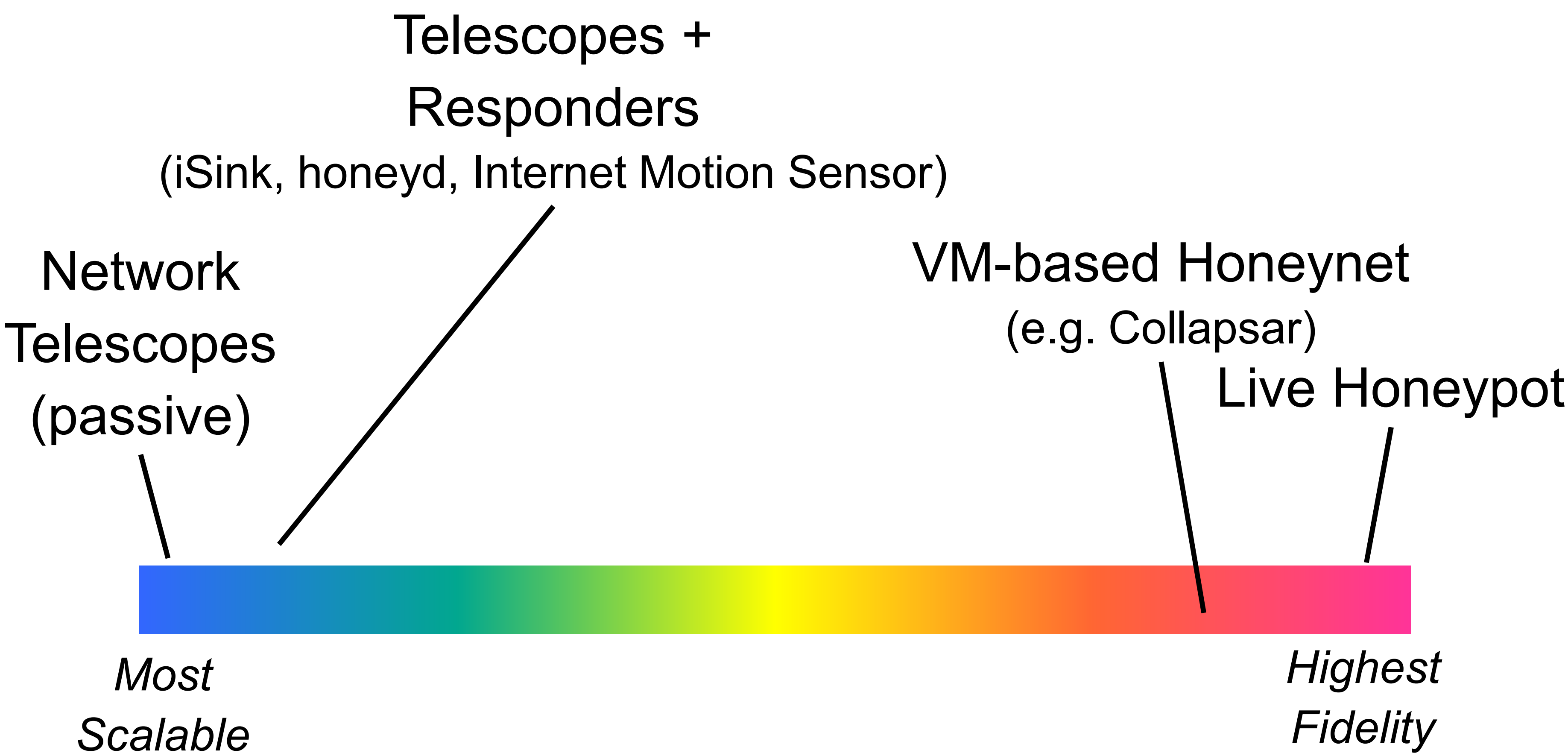
Telescopes + Active Responders

- **Problem**: Telescopes are **passive**, can't respond to TCP handshake
  - Is a SYN from a host infected by CodeRed or Welchia? Dunno.
  - What does the worm payload look like? Dunno.
- **Solution**: proxy responder
  - Stateless: TCP SYN/ACK (Internet Motion Sensor), per-protocol responders (iSink)
  - Stateful: Honeyd (still can scale quite well)
  - Can differentiate and fingerprint initial payload
    - Assumes this is enough to identify/differentiate malcode
    - W32.Femot counter example (90 pairs of exchanges needed!)

Honeypots

- **Problem**: responders offer poor **fidelity**.
  - Don't know what worm/virus would do? No code ever executes after all… What bot code would be downloaded? Where from? What control channels?
- **Solution**: redirect to real "infectable" hosts (**honeypots**)
  - Individual hosts or VM-based: Collapsar, HoneyStat, Symatec Deepsight
- Challenges
  - Scalability ($$$)
  - Liability (grey legal areas)
  - Isolation (control for inter-malware competition)
  - Detection (VMWare detection code in the wild)

The Scalability/Fidelity tradeoff



Opportunity #1: Network-level multiplexing

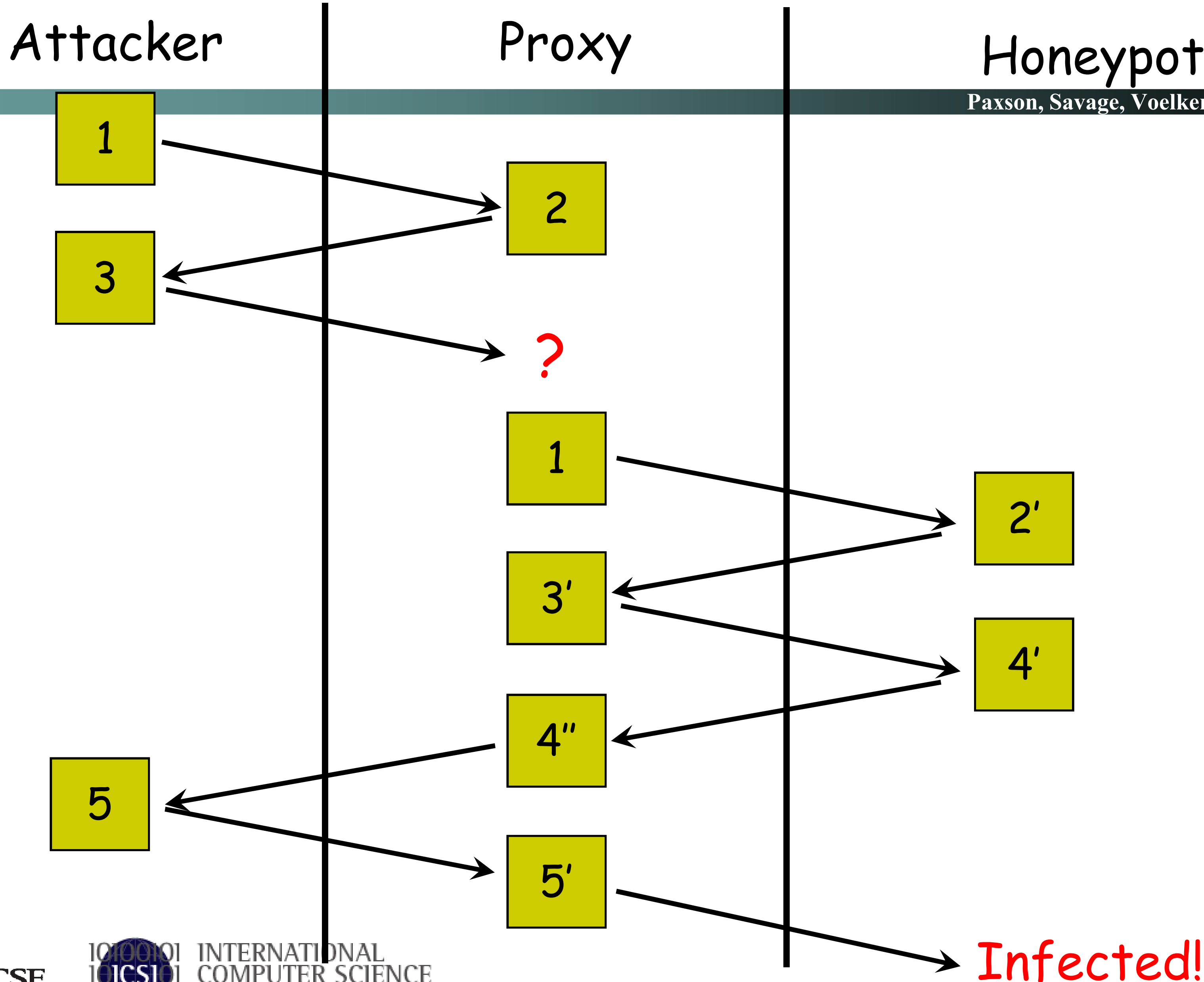
- Most addresses are ***idle*** at any given time
  - Late bind honeypots to IP addresses
- Most traffic **does not** cause an infection
  - Recycle honeypots if can't detect anything interesting
  - Only maintain honeypots of interest for extended periods
  - Can easily get 200:1 improvement here
    (IMS, GQ, Potemkin)

UCSDCSE Computer Science and Engineering
ICSI INTERNATIONAL COMPUTER SCIENCE INSTITUTE

Network-level Filtering

- Scan filtering
  - A given remote source is allowed to probe at most N addresses in a given period of time
  - Don't need to see the same thing again and again
- First-packet filtering
  - Filter probes based on the first data packet
    - Again, don't care about details of known threats
    - Can block worms such as Code Red and Slammer
  - Cannot filter multi-stage attacks
- **Replay proxy**
  - Use responder-side replay to filter multi-stage attacks
  - Use initiator-side replay to bring honeypots "up to speed"

Replay Proxy



Roleplayer Replay Proxy [CPW+06]

- Challenge: how to replay the conversation if you don't know the protocol?
  - Need to normalize addresses, lengths, etc…
- Crazy idea:
  - Watch instances of the protocol and "learn" its dynamic features
  - Use to automatically create protocol-specific replay scripts
- Amazingly this actually works!

[CPW+06] Cui, Paxson, Weaver & Katz, **Protocol-Independent Adaptive Replay of Application Dialog**, NDSS 2006

Opportunity #2: Host-level multiplexing

- CPU utilization in each honeypot is quite low (<<1%)
  - Use VMM to multiplex honeypots on single machine
  - Done in practice, but limited by memory bottleneck
- **Memory coherence property**
  - Few memory pages are actually modified in input
  - Share unmodified pages between VMs
- Scalability relates to *unique* memory per VM

Host-level multiplexing

- CPU utilization in each honeypot is quite low (<<1%)
  - Use VMM to multiplex honeypots on single machine
  - Done in practice, but limited by memory bottleneck
- Memory coherence property
  - Few memory pages are actually modified in input
  - Share unmodified pages between VMs
- Scalability relates to *unique* memory per VM

Potemkin VMM [VMC+06]

- Xen-based, using new shadow translate mode
  - Integrated into VT support
- Clone manager instantiates frozen VM image and keeps it resident in physical memory
  - **Flash cloning:** memory instantiated via eager copy of PTE pages and lazy faulting of data pages (no software startup)
  - **Delta virtualization**: copy implemented as copy-on-write (no memory overhead for shared code/data)
- Creating new VM is a lightweight operation
- Supports **hundreds** of simultaneous VMs per host

[VMC+06] Vrable, Ma, Chen, Moore, Vandekieft, Snoeren, Voelker & Savage, **Scalability, Fidelity and Containment in the Potemkin Virtual Honeyfarm**, SOSP 2005

UCSDCSE Computer Science and Engineering
INTERNATIONAL COMPUTER SCIENCE INSTITUTE

Containment

- Key issue: 3rd party liability and contributory damages
  - Honeyfarm = **worm accelerator**
  - Worse I knowingly allowed my hosts to be infected (premeditated negligence and outside best-practice safe harbor)
- Export policy tradeoffs between risk and fidelity
  - Block all outbound packets: no TCP connections
  - Only allow outbound packets to host that previously send packet: no outbound DNS, no botnet updates
  - Allow outbound, but “scrub”: is this a best practice?
  - Redirect outbound packets back into honeyfarm (i.e. to other honeypot)
  - In the end, need fairly flexible policy capabilities
    - Complex interaction between technical & legal drivers
    - This is one reason CCIED has a lawyer on staff

Overall challenges for honeypots

- **Depends** on asynchronous input
  - What if they don't scan that range (smart bias)
  - What if they propagate via e-mail, IM? (doable, but privacy issues)
- Inherent tradeoff between liability exposure and detectability
  - Honeypot detection software exists… perfect virtualization tough
  - Resource exhaustion (from outbreak or DoS)
- It doesn't necessary reflect what's happening on **your** network (can't count on it for local protection)

Signature-oriented detection

- Power of signatures
- Lessons from the anti-virus world
- How to learn signatures
  - Network-based learning
  - Host-based systems
- How to distribute signatures

Why we love signatures?

- They are precise (hopefully)
  - Allows *least restrictive* defense

- **You can share them!**
  - This is the big win…
  - Reactive on a large-scale
    - Leverage detection of many parties
    - You can be defended without ever being attacked

Tangent: anti-virus industry (what they learned about signatures)

- Historically, focused on malware signatures
  - Precise description of **particular** malicious code
- Basic Procedure
  - Gather samples of known bad stuff
  - Generate unique *malcode signature* for each one (also filter against known good corpus)
  - Distribute signatures; repeat
- Works great for a while… then the adversary adapts

Tangent: anti-virus industry Virus/anti-virus co-evolution

- Early virus scanners only check head/tail of files
  - Virus authors insert branch at beginning
- Scalpel scanning: follow control flow and then scan
  - Encrypted viruses: encrypted payload
- Signature on decryptor
  - Polymorphic viruses: encrypted payload and random decryptor (xor-based)
- X-ray scanning: infer key by xoring against know signature
  - Don't use XOR
- Generic decryption: emulate program in VM until memory decrypted
  - Entry-point obscuring viruses (anti-emulation)
- Custom per-engine detectors
  - Etc…
- Two big observations: antivirus is hard (not just grep) and all of your assumptions will become incorrect iff you are successful

[N97] C. Nachenberg, **Computer Virus-Antivirus Coevolution**, CACM, January 1997.

Automated learning

- **Network-based**
  - Correlate traffic between many hosts
  - Signature -> lexical similarity between anomalous payloads
- **Host-based**
  - Identify signature from single host infection
  - Signature -> textual input involved in control-flow violation

Network-based learning

- Challenge: need to automatically ***learn*** a content "signature" for each new worm – quickly (<1 sec?)
- Approach: Monitor network and look for strings common to traffic with *worm-like* behavior
- Signatures can then be used for content blocking

PACKET HEADER

```
SRC: 11.12.13.14.3920 DST: 132.239.13.24.5000 PROT: TCP
```

PACKET PAYLOAD (CONTENT)

```
00F0   90 90 90 9
0100   90 90 90 9
0110   90 90 90 9
0120   90 90 90 90 90 90 90               90 90 90 90 90 ................
0130   90 90 90 90 90 90 90 90 EB 10 5A 4A 33 C9 66 B9 ..........ZJ3.f.
0140   66 01 80 34 0A 99 E2 FA EB 05 E8 EB FF FF FF 70 f..4...........p
. . .
```

Kibvu.B signature captured on May 14^{th}, 2004

Content sifting [SEV+04]

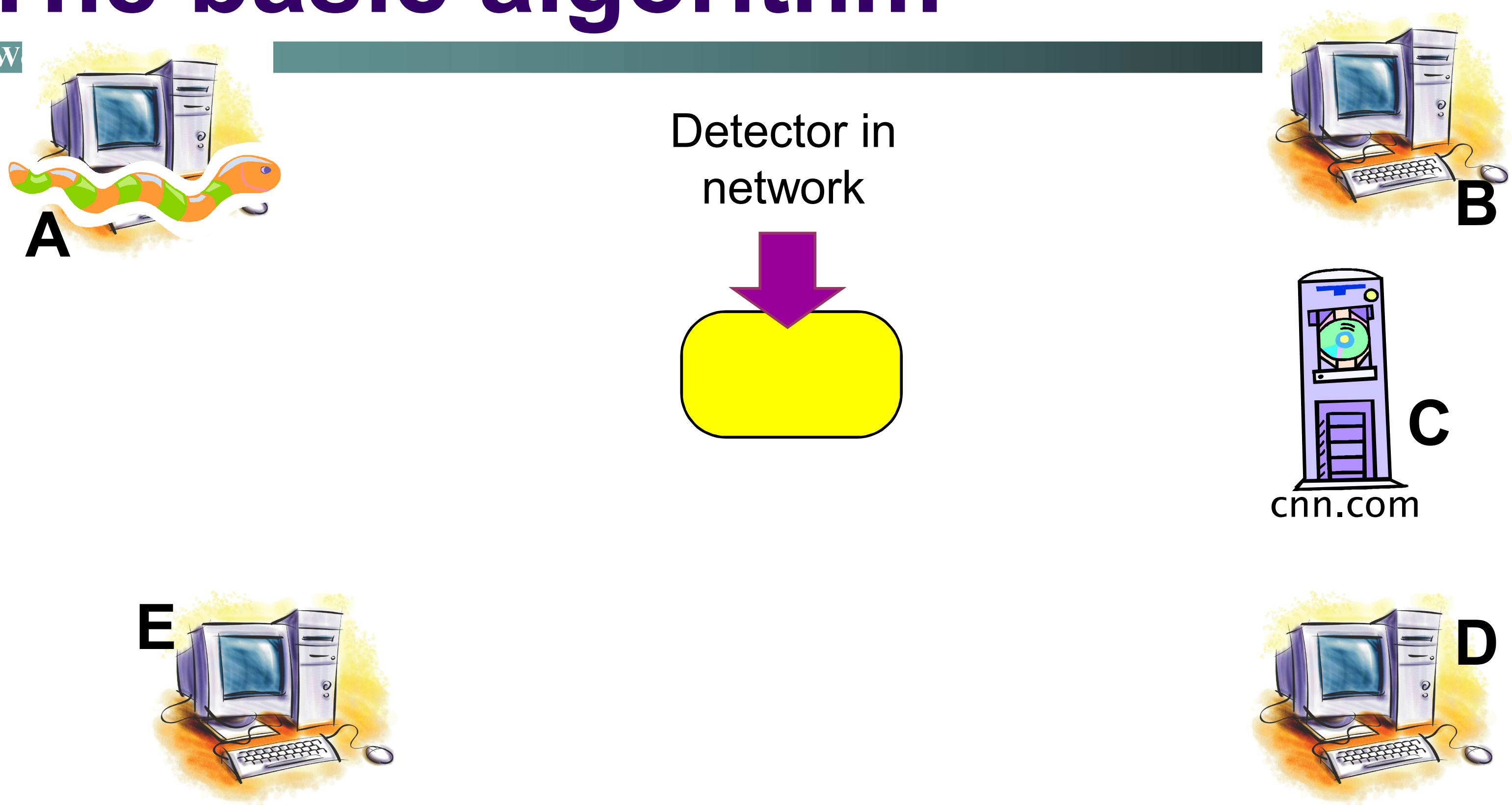
- Assume there exists some (relatively) unique invariant bitstring *W* across all instances of a particular worm

- Two consequences
  - **Content Prevalence**: *W* will be more common in traffic than other bitstrings of the same length
  - **Address Dispersion**: the set of packets containing *W* will address a disproportionate number of distinct sources and destinations

- *Content sifting*: find *W*'s with high content prevalence and high address dispersion and drop that traffic

[SEV+04] Singh, Estan, Varghese & Savage, **Automated Worm Fingerprinting**, OSDI 2004

The basic algorithm

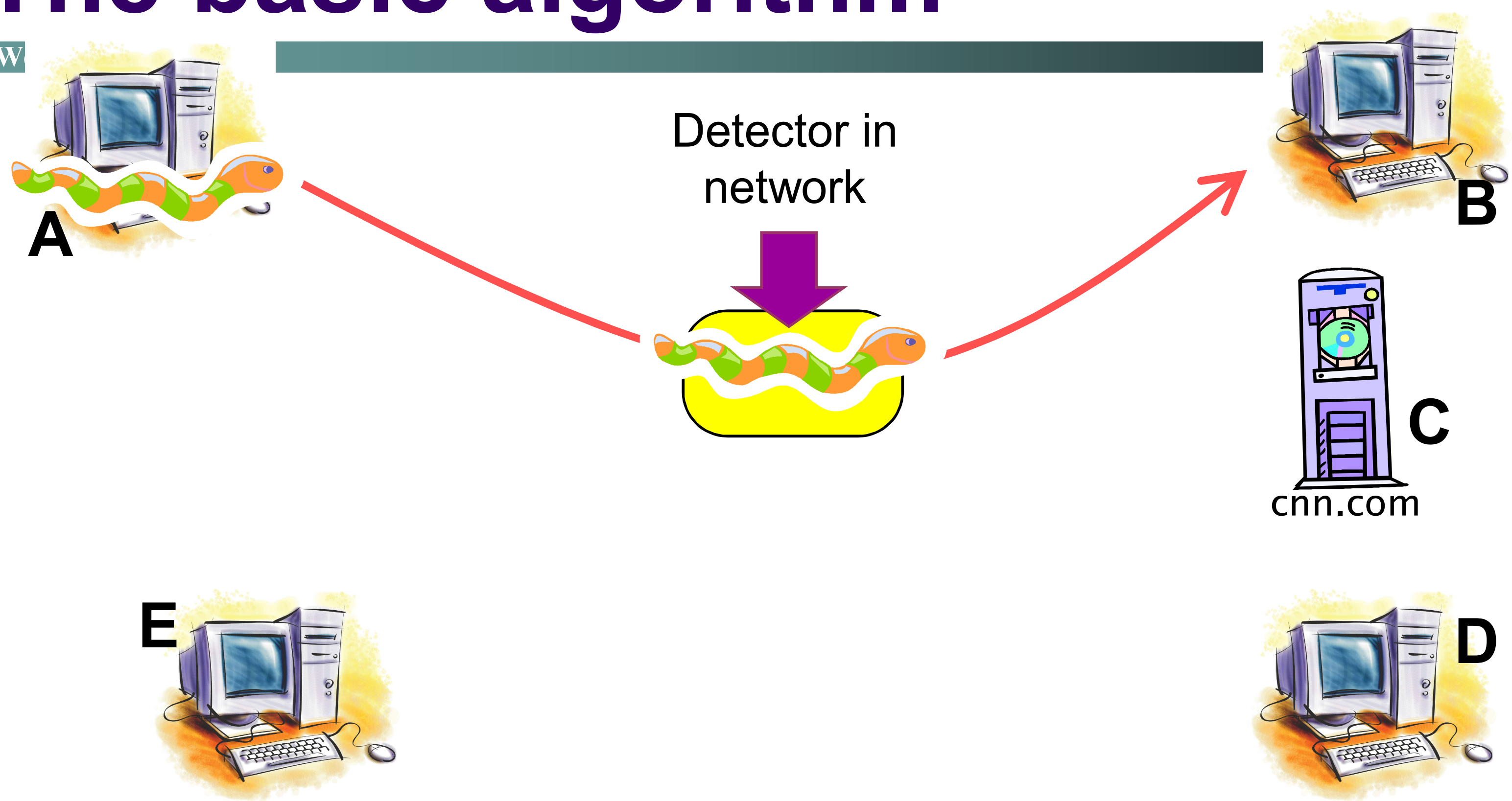


Prevalence Table

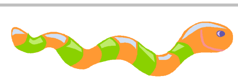
Address **Dispersion** Table

Sources	Destinations

The basic algorithm



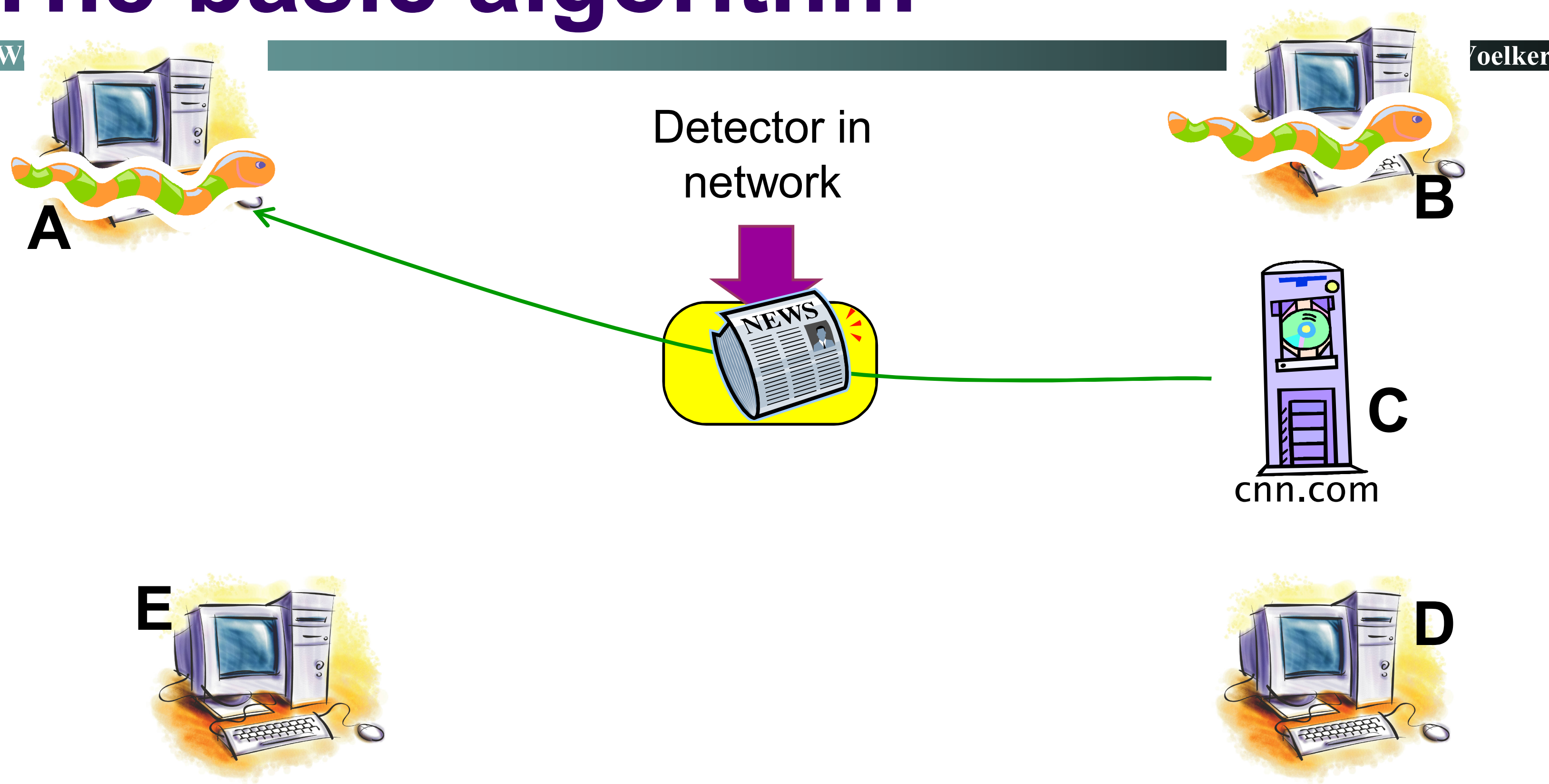
Prevalence Table

	1

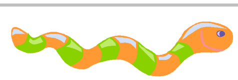
Address **Dispersion** Table

Sources	Destinations
1 (A)	1 (B)

The basic algorithm



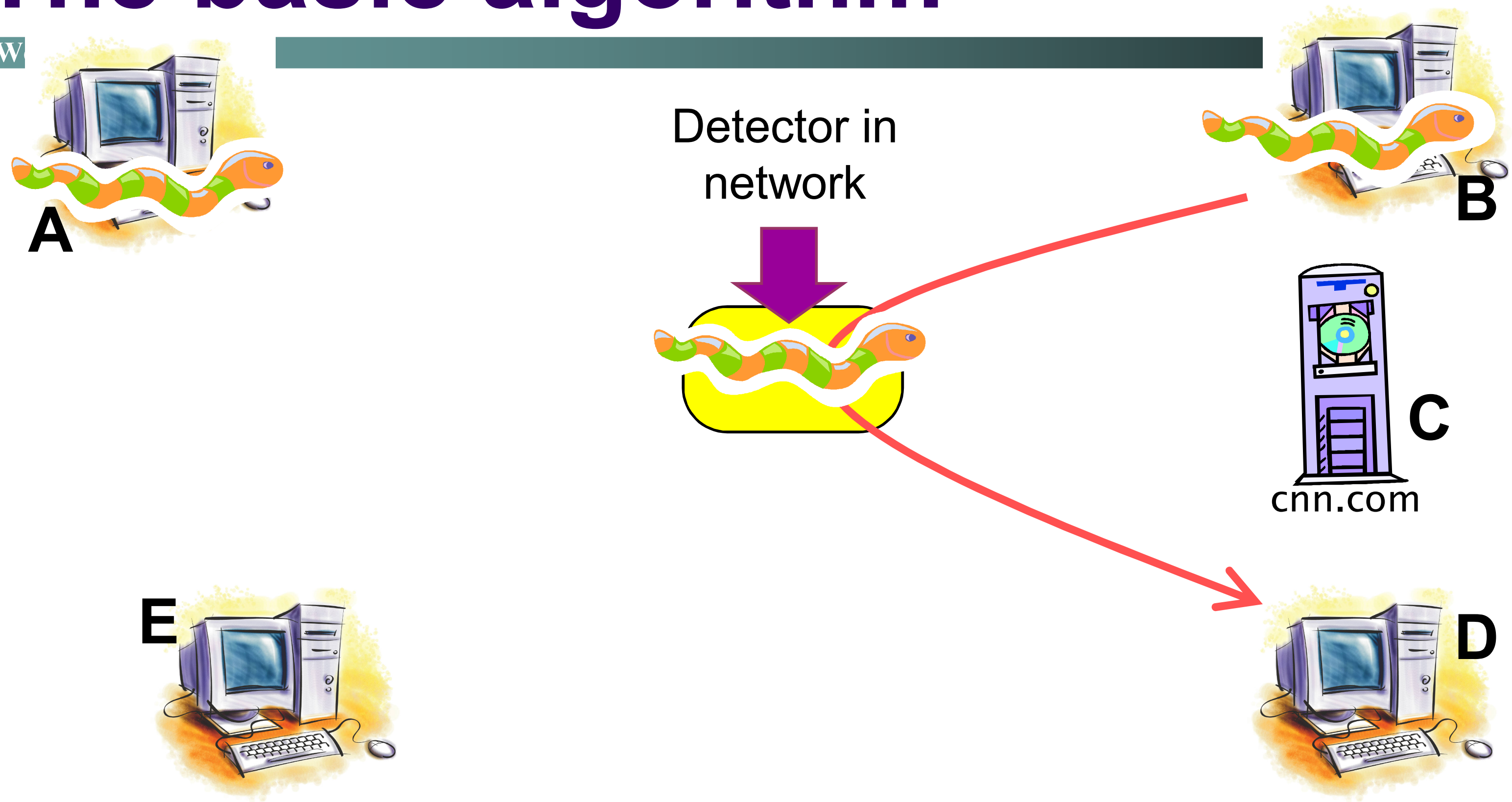
Prevalence Table

	1
	1

Address **Dispersion** Table

Sources	Destinations
1 (A)	1 (B)
1 (C)	1 (A)

The basic algorithm



Prevalence Table

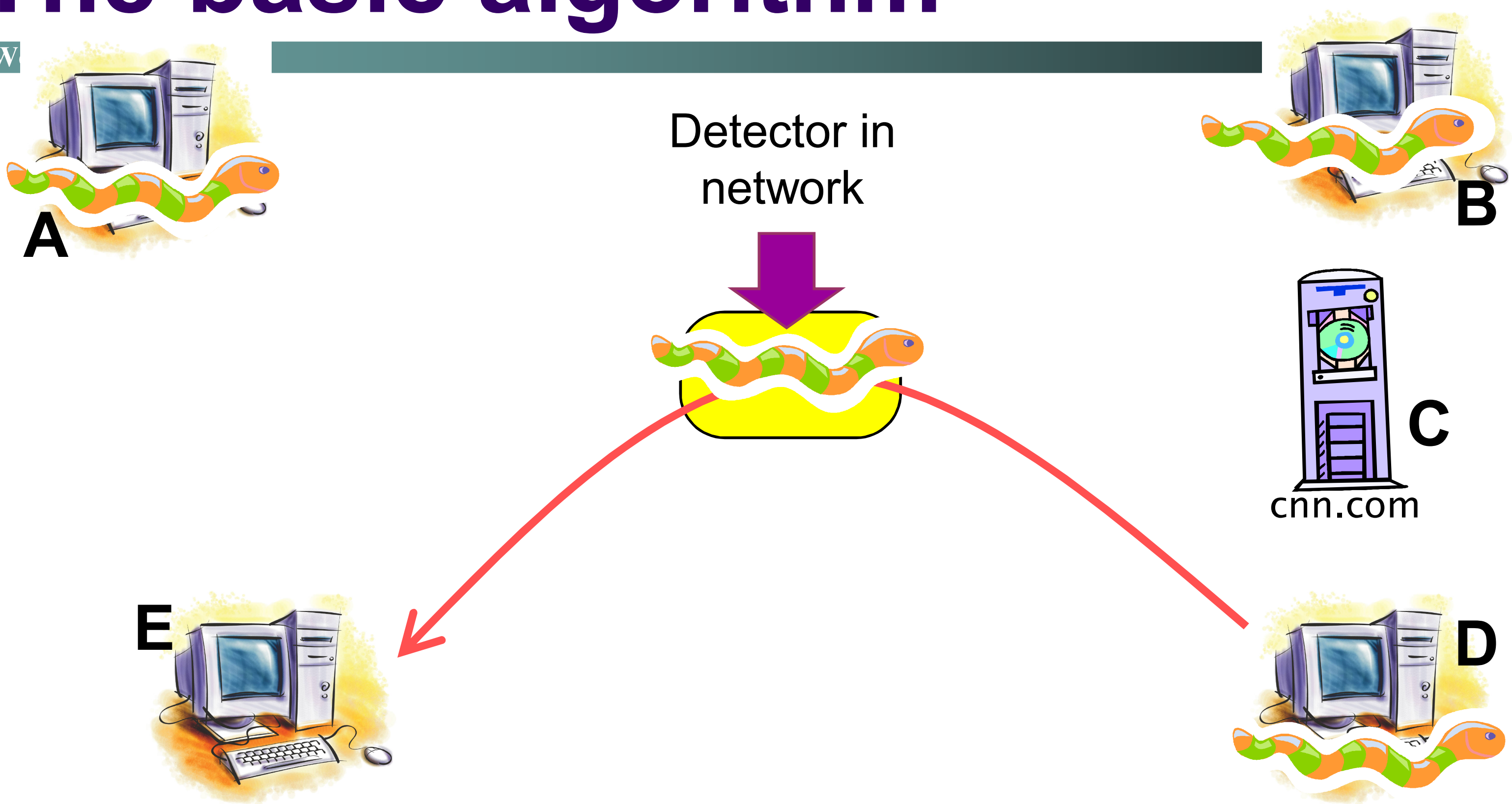
	2
	1

Address **Dispersion** Table

Sources	Destinations
2 (A,B)	2 (B,D)
1 (C)	1 (A)

UCSDCSE Computer Science and Engineering
ICSI INTERNATIONAL COMPUTER SCIENCE INSTITUTE

The basic algorithm



Prevalence Table

(worm)	3
(news)	1

Address **Dispersion** Table

Sources	Destinations
3 (A,B,D)	3 (B,D,E)
1 (C)	1 (A)

Challenges

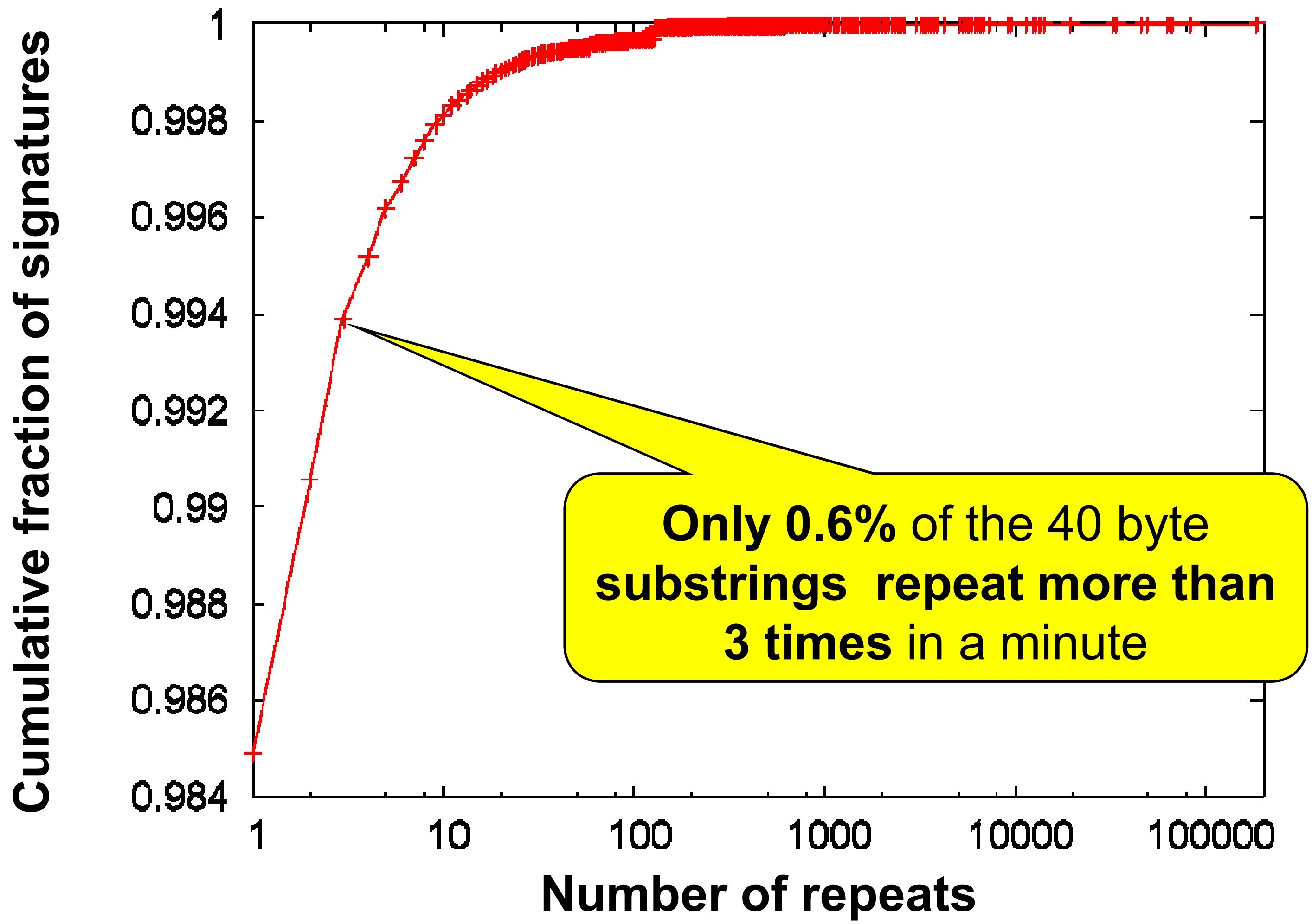
- **Computation**
  - To support a 1Gbps line rate we have 12us to process each packet, at 10Gbps 1.2us, at 40Gbps…
    - Dominated by memory references; state expensive
  - Content sifting requires looking at ***every*** byte in a packet
- **State**
  - On a fully-loaded 1Gbps link a naïve implementation can easily consume 100MB/sec for table
  - Computation/memory duality: on high-speed (ASIC) implementation, latency requirements may limit state to on-chip SRAM

Another approach: fast content sifting algorithms

- Reduce substring representation
  - Index fixed-length substrings
  - Represent with incremental hashes
  - Value sampling in hash space
- Reduce prevalence/dispersion state
  - **High-pass filter to only store frequent substrings**
  - Approximation algorithm to tell if number of unique src/dst pairs is large

NetSift → Cisco Systems

Observation: High-prevalence strings are rare



Efficient high-pass filters for content

- Only want to keep state for prevalent substrings
- Chicken vs egg: how to count strings without maintaining state for them?

- **Multi Stage Filters:** randomized technique for counting "heavy hitter" network flows with low state and few false positives [Estan SIGCOMM02]

- **Three orders of magnitude** memory savings

Earlybird

- Software implementation (200Mbps)
- Over 6mos at UCSD
  - Detected and automatically generated signatures for **every** known worm outbreak over eight months
  - ***Can*** produce a precise signature for a new worm/virus in a *fraction* of a second
- **Known worms detected**:
  - Code Red, Nimda, WebDav, Slammer, Opaserv, …
- **Unknown worms (with no public signatures) detected**:
  - MsBlaster, Bagle, Sasser, Kibvu, …

Sample report: Sasser

early bird
Intrusion Detection System

Monday 03rd of May 2004 12:21:10 PM

Status **Anomalies** **Setup** **About**

characterization

First Reported	2004-05-01 14:35:23	
Last Report	2004-05-03 12:20:47	
Packets	490932	
Sources	1334	[list **sources**]
Destinations	6320	[list **destinations**]
Dest Port / Protocol	445 / 6	

Payload Fragment

```
00 00 0c f4 ff 53 4d 42  25 00 00 00 00 18 07 c8  .....SMB%.......
00 00 00 00 00 00 00 00  00 00 00 00 00 08 dc 04  ................
00 08 60 00 10 00 00 a0  0c 00 00 00 04 00 00 00  ..`.............
00 00 00 00 00 00 00 00  00 54 00 a0 0c 54 00 02  .........T...T..
00 26 00 00 40 b1 0c 10  5c 00 50 00 49 00 50 00  .&..@...\.P.I.P.
45 00 5c 00 00 00 00 00  05 00 00 03 10 00 00 00  E.\.............
a0 0c 00 00 01 00 00 00  88 0c 00 00 00 00 09 00  ................
ec 03 00 00 00 00 00 00  ec 03 00 00 90 90 90 90  ................
90 90 90 90 90 90 90 90  90 90 90 90 90 90 90 90  ................
90 90 90 90 90 90 90 90  90 90 90 90 90 90 90 90  ................
90 90 90 90 90 90 90 90  90 90 90 90 90 90 90 90  ................
90 90 90 90 90 90 90 90  90 90 90 90 90 90 90 90  ................
90 90 90 90 90 90 90 90  90 90 90 90 90 90 90 90  ................
90 90 90 90 90 90 90 90  90 90 90 90 90 90 90 90  ................
90 90 90 90 90 90 90 90  90 90 90 90 90 90 90 90  ................
90 90 90 90 90 90 90 90  90 90 90 90 90 90 90 90  ................
90 90 90 90 90 90 90 90  90 90 90 90 90 90 90 90  ................
90 90 90 90 90 90 90 90  90 90 90 90 eb 10 5a 4a  ..............ZJ
33 c9 66 b9 7d 01 80 34  0a 99 e2 fa eb 05 e8 eb  3.f.}..4........
ff ff ff 70 95 98 99 99  c3 fd 38 a9 99 99 99 12  ...p......8.....
```

[display **entire** payload]

UCSDCSE Computer Science and Engineering
ICSI COMPUTER SCIENCE INSTITUTE

False Positives

- **Common protocol headers**
  - Mainly HTTP and SMTP headers
  - Distributed (P2P) system protocol headers
  - **Procedural whitelist**
    - Small number of popular protocols
- **Non-worm epidemic Activity**
  - **SPAM**
  - BitTorrent

```
GNUTELLA.CONNECT
/0.6..X-Max-TTL:
.3..X-Dynamic-Qu
erying:.0.1..X-V
ersion:.4.0.4..X
-Query-Routing:.
0.1..User-Agent:
.LimeWire/4.0.6.
.Vendor-Message:
.0.1..X-Ultrapee
r-Query-Routing:
```

False Negatives

- Compared w/Snort and new vulns on Bugtraq
  - Found non, but that tells you nothing…the real question is: Could I **cause** a false negative?
  - Answer: Yes
- Contiguous invariant bitstring assumption
  - **What about polymorphic or metamorphic shellcode?**
  - **Hey, this problem sounds familiar…**

Learning polymorphic signatures

- Premise: while payload may be random, there are invariants in exploit (or at least vulnerability)
  - Protocol framing, target return address (e.g., return-to-libc exploit), etc.
- Approach: [NSK05,LSC+06]
  - Oracle groups suspicious vs innocuous flows
  - Extract subsequences from suspicious flows
    - Similar to sequences in other suspicious flows and not found in innocuous flows
    - Use conjunction or ordered list of sequences as signature
    - E.g., GET.*HTTP/1.1.*\r\nHost:.*\r\nHost:.*\xff\xbf

[NSK05] Newsome "Polygraph: Automatically Generating Sigantures for Polymorphic Worms", Oakland 2005
[LSC+06] Li, Sanghi, Chavez, Chen & Kao, Hamsa: Fast Signature Generation for Zero-day Polymorphic Worms with Provable Attack Resilience, Oakland 2006.

Key limitations of network approach: *lexical* point of view is limited

- Evasion
  - Training/mimicry/polymorphism/metamorphism
    - Ultimately favors bad guy; fundamental limitation of vantage point
  - Network evasion
    - Hard to normalize traffic at speed
    - Dharmapurikar et al, *Robust TCP Stream Reassembly in the Presence of Adversaries, USENIX Sec '05*
  - End-to-end encryption
- Denial-of-service via controlled false-positives
  - Worm-like traffic with string "Republican" or "Democrat" in it?
- Analysis & Forensics
  - What does the worm/virus/bot do?
  - Who is controlling it?
- Alternative: host-level detection

Host-based signature learning

- Idea:
  - End system has ideal vantage point
    - Can observe attack in execution domain
  - Carefully instrument host and monitor infections
  - Use run-time analysis of infection to create signature
- Two parts:
  - Exploit detection
  - Signature generation

Host-based exploit detection

- Most exploits redirect control flow via some form of memory overwrite
- Taint checking
  - Tag all input data with a "taint" bit
  - Tag all targets of stores dependent on tainted data as tainted
  - Trap on control flow transfer through tainted data
- Range of implementation options (mostly all slow)
  - Binary rewriting [CCC+04, NS05]
  - Whole system emulation/hardware [KBA02,CC04]
  - Hybrid VM/emulation [HFC+06]

[CCC+04] Costa, Crowcroft, Castro, and Rowstron. **Can we contain Internet worms?** Hotnets 2004
[NS05] Newsome & Song. **Dynamic Taint Analysis: Automatic Detection, Analysis, and Signature Generation of Exploit Attacks on Commodity Software**. NDSS 2005.
[CC04] Crandall & Chong. **Minos: Architectural support for software security through control data integrity**. *International Symposium on Microarchitecture*, 2004.
[KBA02] **Secure execution via program shepherding**. *USENIX Security* 2002.
[HFC+06], Ho, Fetterman, Clark, Warfield & Hand, **Practical Taint-based Protection using Demand Emulation,** *Eurosys* 2006.

Host-based signature generation

- Syntactic signatures
  - Heuristic dataflow analysis [CCC+05]
    - Identify input conditions on which control flow is dependent
    - E.g., input corresponding to target branch
  - Model checking [BNS+06]
    - Derive set of paths that allow reaching this particular bad state
    - Related back to input (not precise)
- Execution signatures
  - Filter on control flow (don't worry about input)
  - More expensive at run-time

[CCC+05] Costa, Crowcroft, Castro, Rowstron, Zhou, Zhang and Barham. **Vigilante: End-to-End Containment of Internet Worms** SOSP 2005
[BNS+06] Brumley, Newsome, Song, Wang & Jha, **Towards Automatic Generation of Vulnerability-Based Signatures**, Oakdland 2006.

Distributing signatures

- Why should you trust my signatures?
  - Self Certifying Alerts:
    - Send enough info to allow recipient to prove that vulnerability exists (perhaps send the exploit itself)
    - Recipient tests alert in sandboxed environment (VM)
    - Note that this is really best suited to the host context
- How do I get my signatures out there quickly?
  - Large-scale push infrastructure (e.g. use Akamai)
  - Peer-to-peer transmission
  - *White* worm

Overall challenges for honeypots

- **Depends** on asynchronous input
  - What if they don't scan that range (smart bias)
  - What if they propagate via e-mail, IM? (doable, but privacy issues)
- Inherent tradeoff between liability exposure and detectability
  - Honeypot detection software exists… perfect virtualization tough (although we're working hard on it)
- Resource exhaustion (from outbreak or DoS)
- It doesn't necessary reflect what's happening on **your** network (can't count on it for local protection)

- Hence, there is a need for both honeyfarm and in-situ approaches