

Scalable Internet Threat Monitoring

Stefan Savage

Collaborative Center for Internet Epidemiology and Defenses
Department of Computer Science & Engineering
University of California at San Diego

In collaboration with Jay Chen, Cristian Estan, Ranjit Jhala, Chris Kanich, Erin Kenneally, Justin Ma, David Moore, Vern Paxson (ICSI), Colleen Shannon, Sumeet Singh, Alex Snoeren, Amin Vahdat, Erik Vandekeift, George Varghese, Geoff Voelker, Michael Vrable, Nick Weaver (ICSI)

Context

- Large scale exploits have established owned hosts as a commodity service **platform**
- Most bad things on the Internet are **applications** of this platform (DDoS, SPAM, Phishing, id theft)
- The establishment of a “virtuous” economic cycles means this relationship is robust

The collage features several overlapping elements:

- The New York Times Technology** header with a search bar and navigation links.
- A **BusinessWeek** magazine cover with the headline **EPIDEMIC** and a person in a biohazard suit holding a laptop. Other headlines include "Crippling computer viruses threaten the info economy. Can they be stopped?", "HEINEKEN WAKING UP AN OLD WORLD BREWER", "BROADBAND HOW THE U.S. CAN CATCH UP", "RESEARCH A MECCA FOR BIO-MEDICINE", and "DRESSING SMART THE NEW LOOK IN MEN'S FASHION".
- A **washingtonpost.com Highlights** banner with the text **the year in review** and the headline **A year of spam, spyware and worms**.
- An **MSNBC News** banner with the text **TECHNOLOGY & SCIENCE** and the headline **Spam, Scams & Viruses**.
- A **CNN.com** search bar and navigation menu with the headline **'Phishing' scams reel**.
- A news article snippet from **Attacks on Windows PC's Grew in First Half of 2004** by **JOHN MARKOFF**, published on **September 20, 2004**. The text begins: **S**AN FRANCISCO, Sept. 19 - A survey of Internet vulner Monday shows a sharp jump in attacks on Windows-base the first six months of 2004, along with a marked increase in con

Challenges

- Defense and deterrence is predicated on *intelligence*
 - Need to detect, characterize and analyze new malware threats quickly
- At Internet scale this present problems
 - Automated malware can spread quickly
 - Slammer covered the Internet in minutes (Moore et al, 2003)
 - 1M hosts in <1sec (Staniford et al, 2004)
 - **How to act in this time frame?**
 - Malware is increasingly sophisticated
 - How do we know if something is malware?
 - **How do we know what its doing?**

Threat Detection/Monitoring

- Classes of monitors
 - Network-based
 - Ease of deployment, significant coverage
 - Inter-host correlation
 - Scalability challenges (performance)
 - Endpoint-based
 - Host offers high-fidelity vantage point (execution vs lexical domain)
 - Scalability challenges (deployment)
- Monitoring environments
 - In-situ: real activity as it happens
 - Network/host IDS
 - Ex-situ: “canary in the coal mine”
 - HoneyNets/Honeypots

Today

- Two examples of *scalable* threat detection approaches (complementary)
 - **Earlybird**: network-based worm detection and signature inference system (content sifting)
 - **Potemkin**: large scale virtual machine-based honeyfarm environment

For more details see:

Singh, Estan, Varghese and Savage, *Automated Worm Fingerprinting*, OSDI 2004

Vrable et al, *Scalability, Fidelity and Containment in the Potemkin Virtual Honeyfarm*, SOSP 2005

Worm Signature Inference

- For some worms needs sub minute and even *sub second* automated defense
[Moore et al, 2003]
- Challenge: need to automatically *learn* a content “signature” for each new worm – in less than a second!

Approach

- Monitor network and look for strings common to traffic with worm-like behavior
- Signatures can then be used for content filtering

PACKET HEADER	
SRC: 11.12.13.14.3920 DST: 132.239.13.24.5000 PROT: TCP	
PACKET PAYLOAD (CONTENT)	
00F0	90 90 90 90
0100	90 90 90 90M?.w
0110	90 90 90 90cd.....
0120	90 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 B9ZJ3.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 by Earlybird on May 14th, 2004

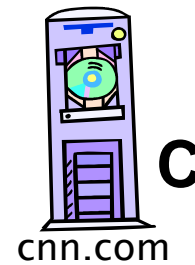
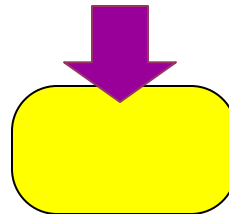
Content sifting

- 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

The basic algorithm

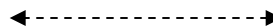


Detector in network

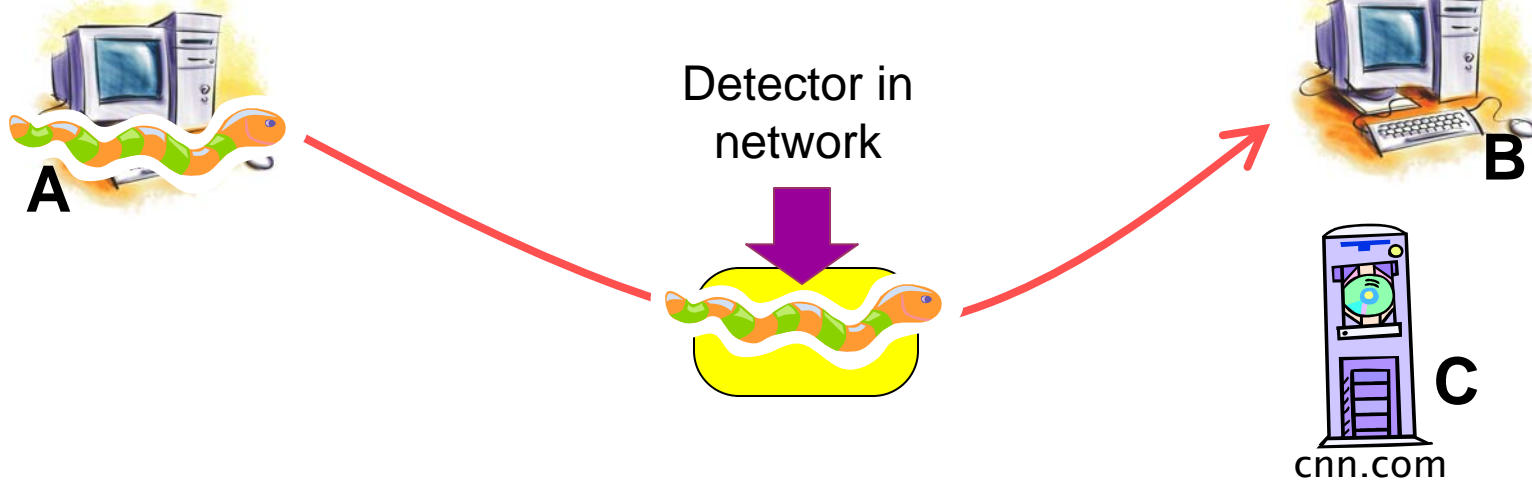


Prevalence Table


Address Dispersion Table
Sources Destinations



The basic algorithm

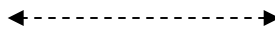


Prevalence Table

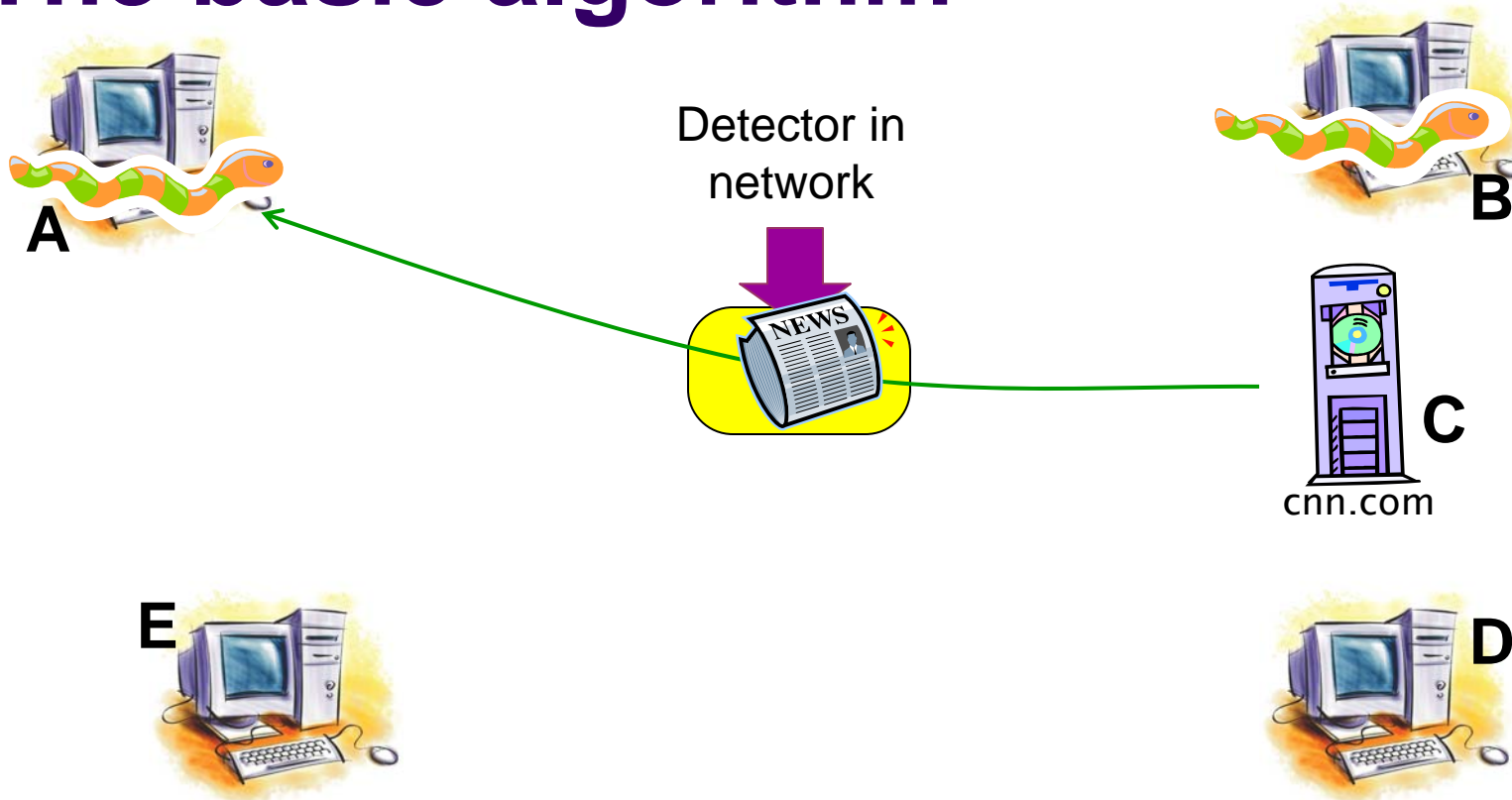
	1

Address Dispersion Table
Sources Destinations



1 (A)	1 (B)



The basic algorithm

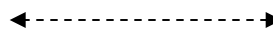


Prevalence Table

	1
	1

Address Dispersion Table
Sources Destinations

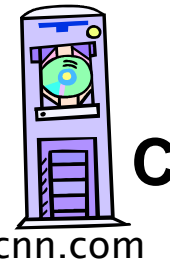
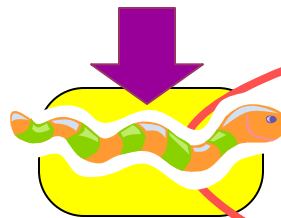
1 (A)	1 (B)
1 (C)	1 (A)





The basic algorithm



Detector in network

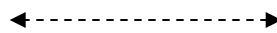


Prevalence Table

	2
	1

Address Dispersion Table
Sources Destinations

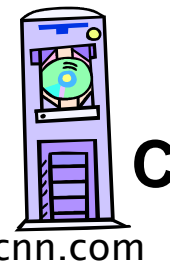
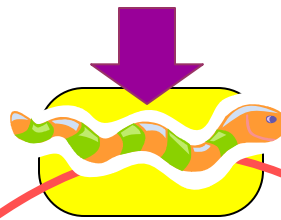
2 (A,B)	2 (B,D)
1 (C)	1 (A)



The basic algorithm

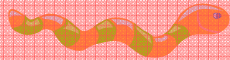



Detector in network



Prevalence Table

Address Dispersion Table
Sources Destinations

	3	↔	3 (A,B,D)	3 (B,D,E)
	1		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

Which substrings to index?

- **Approach 1: Index all substrings**
 - Way too many substrings → too much computation → too much state
- **Approach 2: Index whole packet**
 - Very fast but trivially evadable (e.g., Witty, Email Viruses)
- **Approach 3: Index all contiguous substrings of a fixed length 'S'**
 - Can capture all signatures of length 'S' and larger

A B C D E F G H I J K

How to represent substrings?

- Store **hash** instead of literal to reduce state
- **Incremental hash** to reduce computation
- **Rabin fingerprint** is one such efficient incremental hash function [Rabin81,Manber94]
 - One multiplication, addition and mask per byte

P1 R A N D A B C D O M

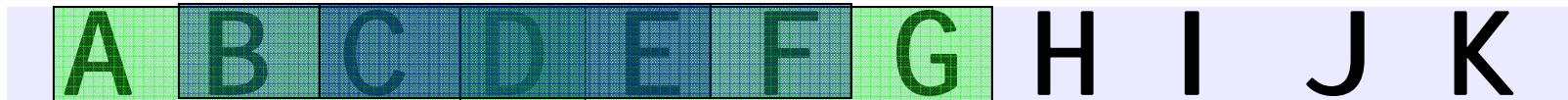
Fingerprint = 11000000

P2 R A B C D A N D O M

Fingerprint = 11000000

Value sampling [Manber '94]

- Sample hash if last 'N' bits of the hash are equal to the value 'V'
 - The number of bits 'N' can be dynamically set
 - The value 'V' can be randomized for resiliency

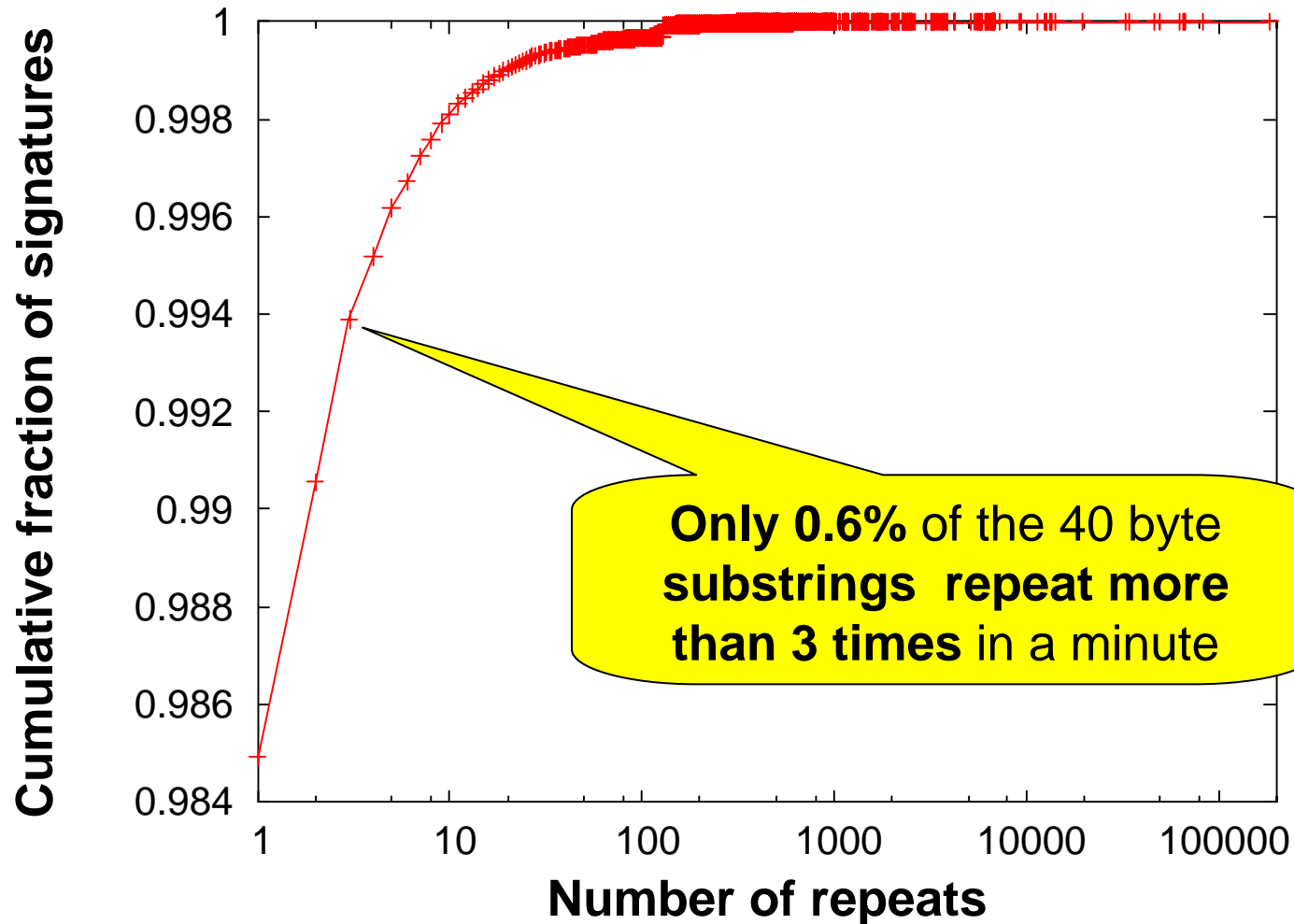


Fingerprint: 00000000000000000000000000000000

SAMPLE OR SAMPLE

- $P_{\text{track}} \rightarrow$ Probability of selecting **at least one** substring of length S in a L byte invariant
 - For 1/64 sampling (last 6 bits equal to 0), and 40 byte substrings
 $P_{\text{track}} = 99.64\%$ for a 400 byte invariant

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 [Estan02]
 - Instead of using flow id, use **content hash**
 - Rabin Fingerprints with Mandber’s Value sampling
 - Three orders of magnitude memory savings

Observation:

High *address dispersion* is rare too

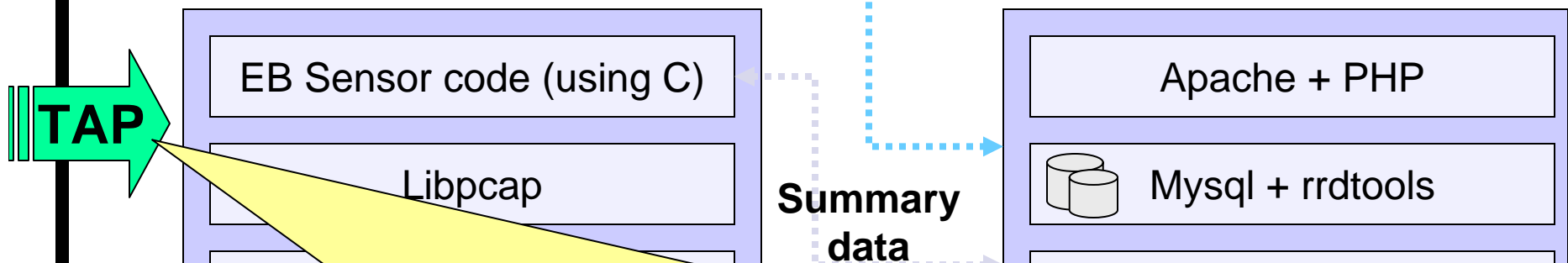
- Naïve implementation might maintain a list of sources (or destinations) for each string hash
- But dispersion **only** matters if its *over* threshold
 - Approximate counting may suffice
 - **Trades accuracy for state in data structure**
- **Scalable Bitmap Counters**
 - Similar to multi-resolution bitmaps [Estan03]
 - Reduce memory by 5x for modest accuracy error

Content sifting summary

- Index fixed-length substrings using incremental hashes
- Subsample hashes as function of hash value
- Multi-stage filters to filter out uncommon strings
- Scalable bitmaps to tell if number of distinct addresses per hash crosses threshold
- **Now** its fast enough to implement

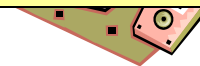
Software prototype: Earlybird

To other sensors and
blocking devices



Setup 1: Large fraction of the UCSD campus traffic,
Traffic mix: approximately 5000 end-hosts, dedicated servers for campus wide services (DNS, Email, NFS etc.)
Line-rate of traffic varies between 100 & 500Mbps.

Setup 2: Fraction of local ISP Traffic,
Traffic mix: dialup customers, leased-line customers
Line-rate of traffic is roughly 100Mbps.



Experience

- Generally... ahem... *good*.
 - Detected and automatically generated signatures for **every** known worm outbreak over eight months
 - **Can** produce a precise signature for a new worm in a *fraction* of a second
 - Software implementation keeps up with 200Mbps
- **Known worms detected:**
 - Code Red, Nimda, WebDav, Slammer, Opaserv, ...
- **Unknown worms (with no public signatures) detected:**
 - MsBlaster, Bagle, Sasser, Kibvu, ...

False Negatives

- Easy to prove presence, impossible to prove absence
- **Live evaluation:** over 8 months detected every worm outbreak reported on popular security mailing lists
- **Offline evaluation:** several traffic traces run against both Earlybird and Snort IDS (w/all worm-related signatures)
 - Worms not detected by Snort, but detected by Earlybird
 - The converse never true

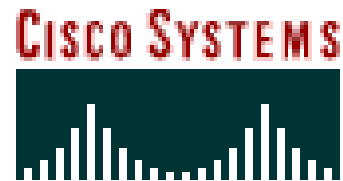
False Positives

- **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:
```

Since paper published...

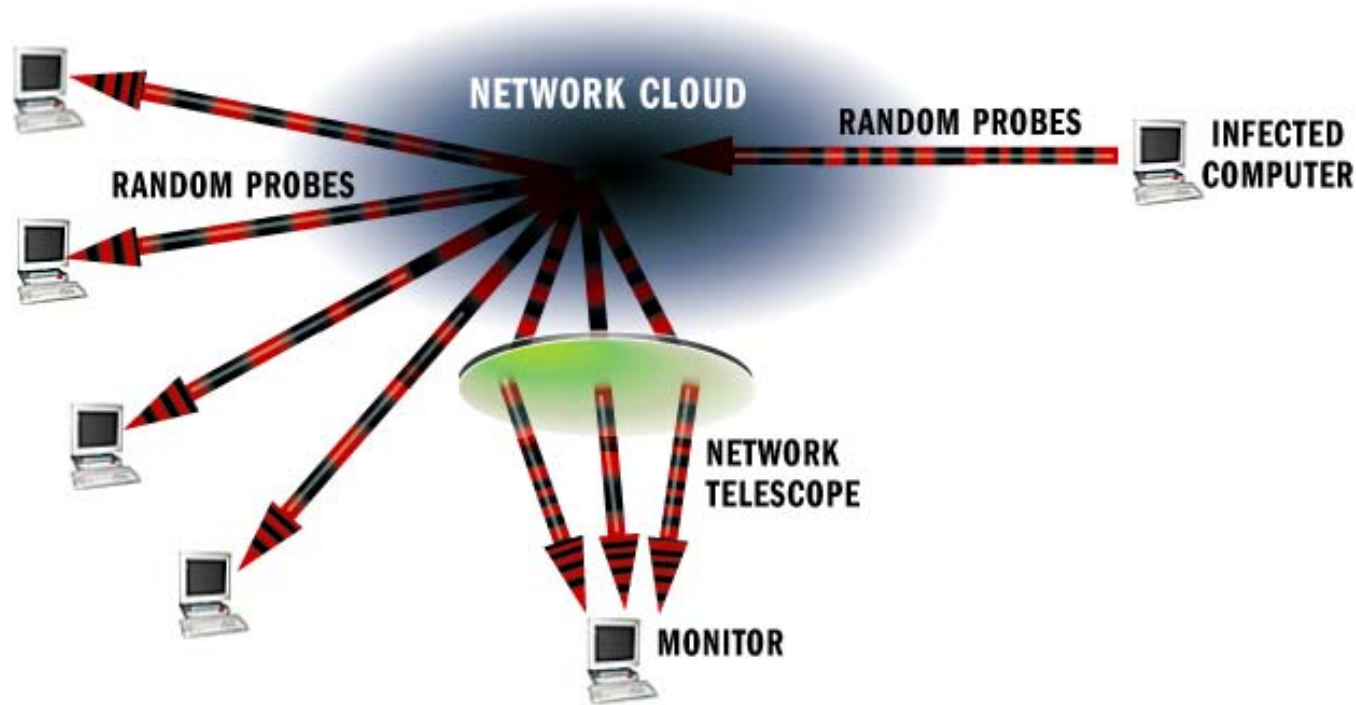
- *Content sifting* technologies patented by UC and licensed to startup, Netsift Inc.
- Netsift significantly improved performance (esp hardware impl) and capabilities
- In June, Netsift was acquired by Cisco



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

- Evasion
 - Polymorphism/metamorphism
 - Newsom et al, *Polygraph: Automatically Generating Signatures for Polymorphic Worms*, Oakland '05
 - But always favors bad guy; fuzzy match + prevalence = noise
 - 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
 - Only covers one link
- Analysis & Forensics
 - What does the worm/virus/bot do?
 - Who is controlling it?
- Alternative: endpoint monitoring

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. UCSD monitors 17M+ addresses

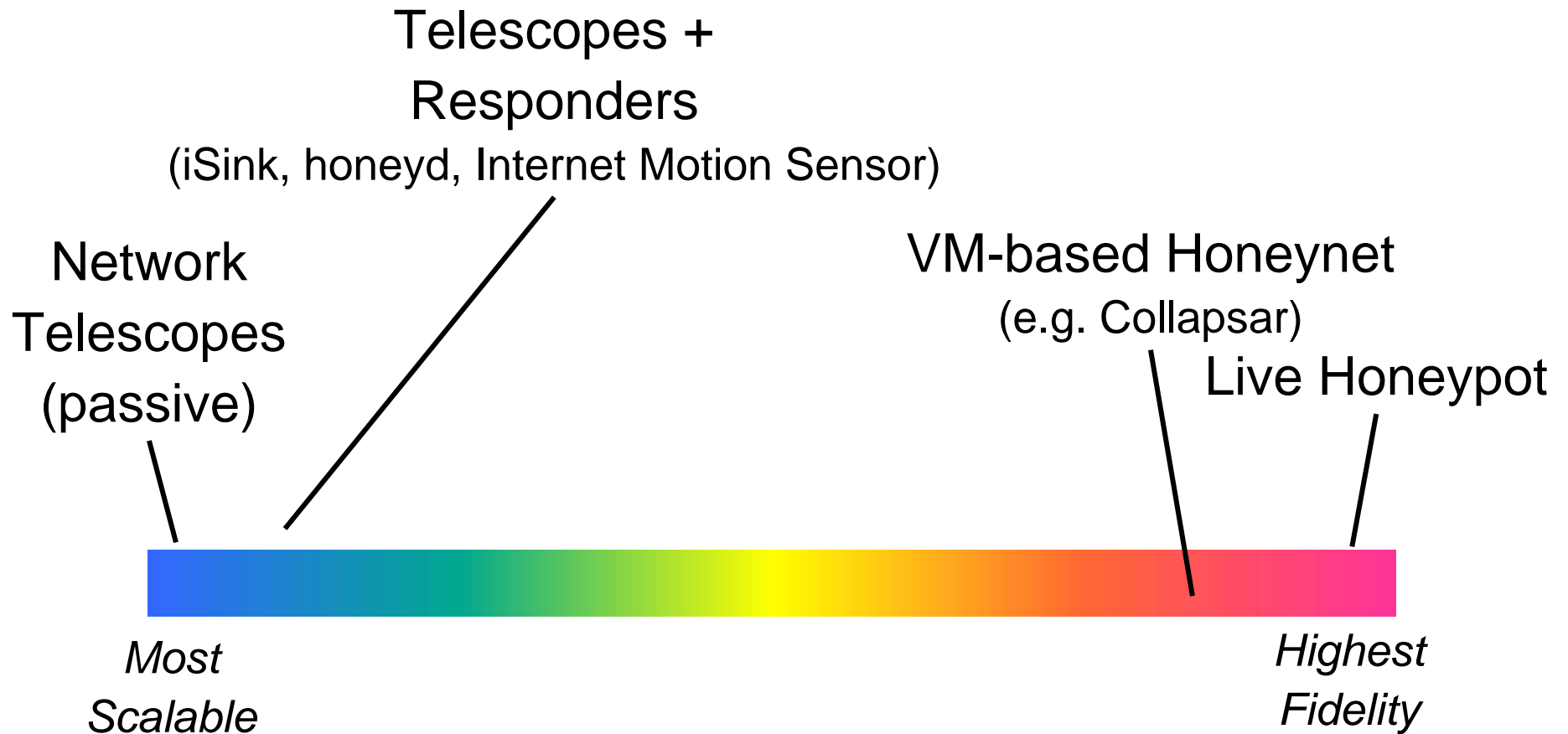
Telescopes + Active Responders

- 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
 - Can differentiate and fingerprint payload
 - False positives generally low since no regular traffic

HoneyNets

- Problem: poor **fidelity**. Don't know what worm/virus would do? No code ever executes after all...
- Solution: redirect scans to real “infectable” hosts (honeypots)
 - Individual hosts or VM-based: Collapsar, HoneyStat, Symantec
 - Can reduce false positives/negatives with host-analysis (e.g. Vigilante, TaintCheck, Minos) and behavioral/procedural signatures
- Challenges
 - **Scalability**
 - Liability (honeywall)
 - Isolation (2000 IP addrs -> 40 physical machines)
 - Detection (VMWare detection code in the wild)

The Scalability/Fidelity tradeoff



Why can't we have both?

- Naïve approach: one machine per IP address
 - 1M addresses = 1M hosts = \$2B+ investment
 - Overkill... most resources will be wasted
- In truth, **only** necessary to maintain the *illusion* of continuously life honeypot systems

The historical inspiration...



“If a tree falls in the forest and no one hears it,
does it make a sound?”

Bishop Berkeley, early 1700s

... and the modern bastardization

Claim: aggressive multiplexing can reduce honeypot resource demands by **5-6 orders of magnitude**

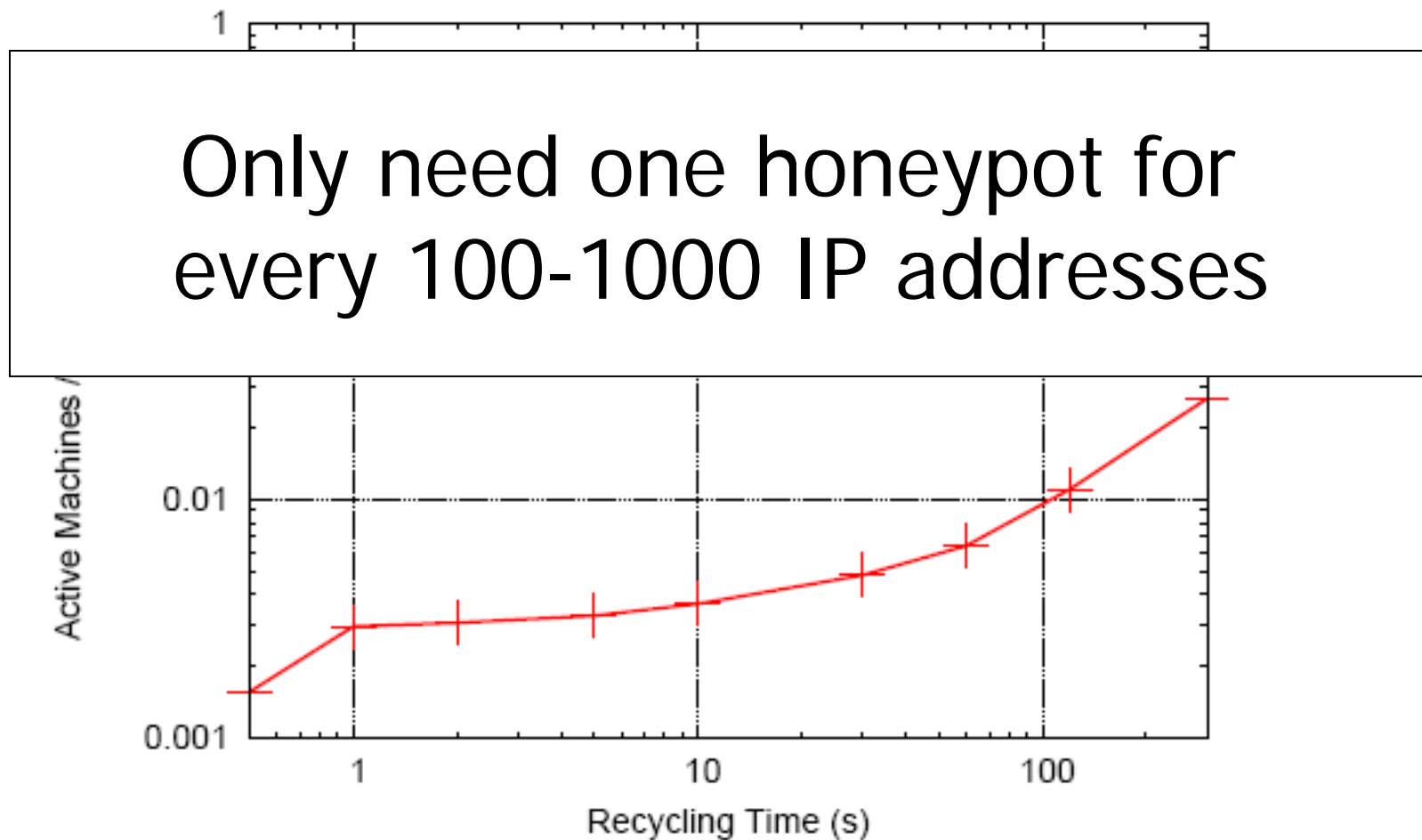
“If a honeypot is vulnerable, but it isn’t exploited, does anyone care?”

Me, group meeting

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
- Scalability is related to both the workload and the recycling rate

Network multiplexing efficiency

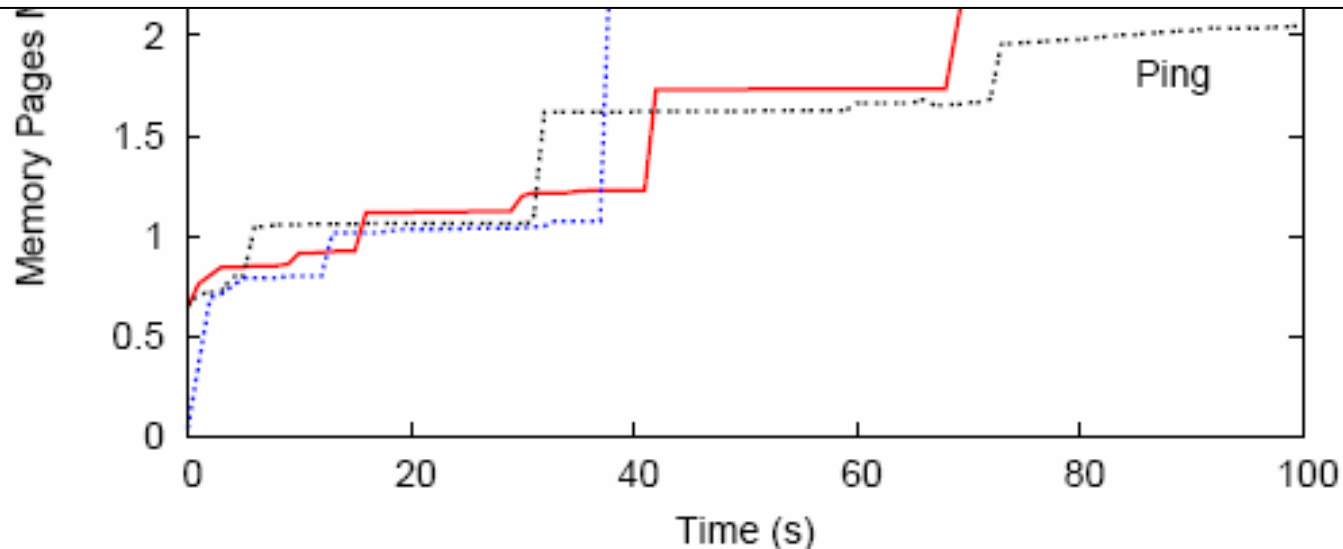


Host-level multiplexing

- CPU utilization in each honeypot is quite low ($\ll 1\%$)
 - 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

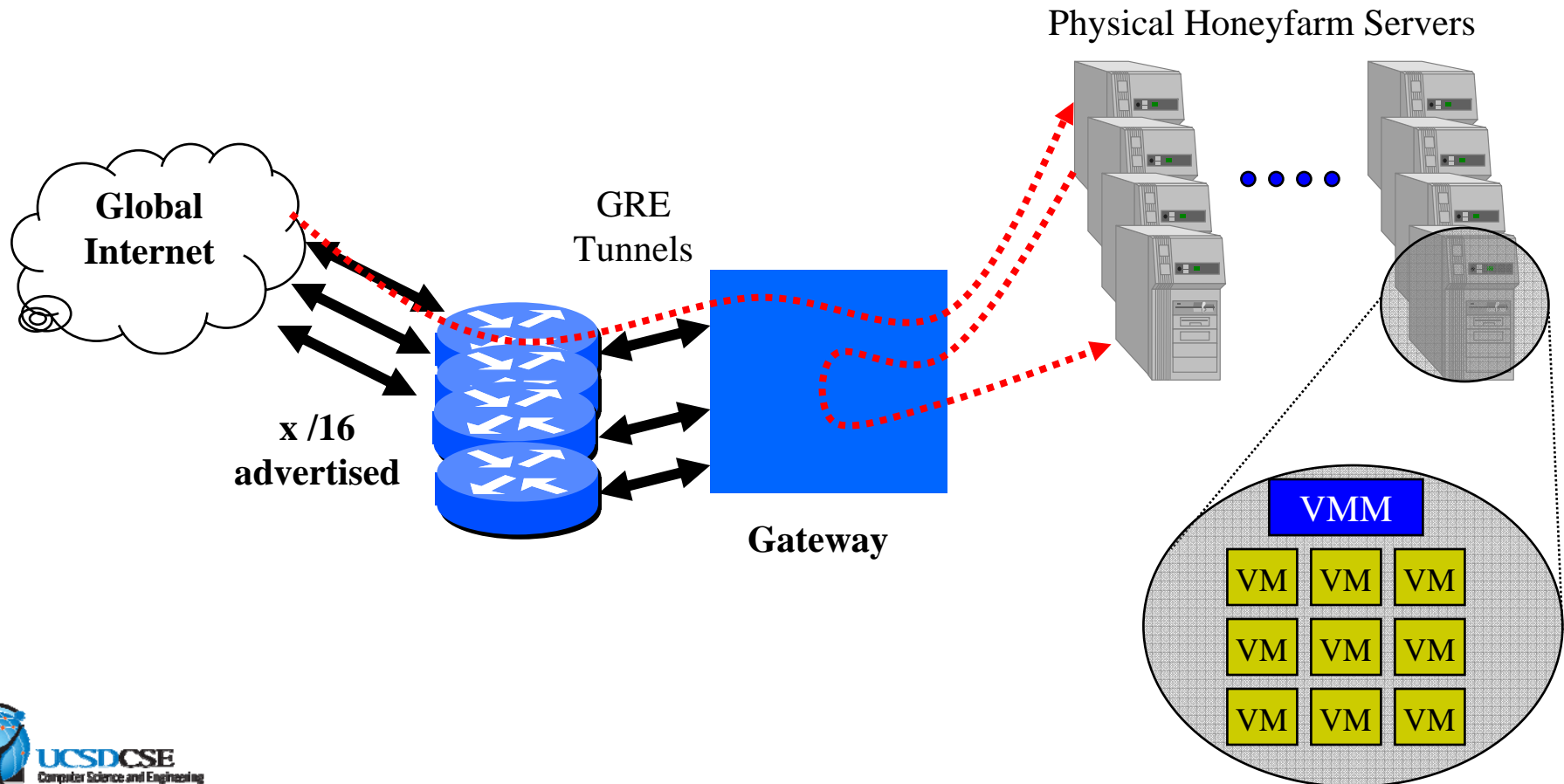
Host multiplexing efficiency

Only need one physical machine for every 100-1000 honeypots



Potemkin: a large scale high-fidelity honeyfarm

- Two components
 - Network Gateway: multiplexes traffic on physical servers
 - Potemkin VMM: multiplexes VM on servers



Potemkin Gateway

- Gateway (Click-based) terminates inbound GRE tunnels
- Maintains external IP address->type mapping
 - i.e. 132.239.4.8 should be a Windows XP box w/IIS version 5, etc
- Mapping made concrete when packet arrives
 - Flow entry created and pkt dispatched to *type-compatible* physical host
 - VMM on host creates new VM with target IP address
 - VM and flow mapping GC'd after system determines that an interaction is uninteresting (detectors)

Potemkin VMM

- Xen-based, using new shadow translate mode
- 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)
- Supports hundreds of simultaneous VMs per host
- Overhead: currently takes 0.2-0.5s to create new VM
 - Imperceptible to human user and under TCP handshake timeout
 - Wildly unoptimized (e.g. includes multiple Python invocations)
 - Pre-allocated VM's can be invoked in ~5ms

Containment

- Key issue: 3rd party liability and contributory damages
 - Honeyfarm = **worm accelerator**
 - Worse I knowingly allowed my hosts to be infected (premeditated negligence)
- 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?
 - In the end, need fairly flexible policy capabilities
 - Could do whole talk on interaction between technical & legal drivers
- But it gets more complex...

Internal reflection

- If outbound packet not permitted to real internet, it can be sent back through gateway
 - New VM generated to assume target address (honeyfarm emulates external Internet)
 - Allows causal detection (A->B->C->D) and can *dramatically* reduce false positives
- However, creates new problem:
 - Is there only one version of IP address A?
 - Yes, single “universe” inside honeyfarm
 - No isolation between infections
 - Also allows cross contamination (liability rears its head again)
 - No, how are packets routed internally?

Causal address space aliasing

- A new packet i destined for address t , creates a new *universe* U_{it}
- Each VM created by actions rooted at t is said to exist in the same universe and a single export policy is shared
 - In essence, the 32-bit IP address space is augmented with a universe-id that provides aliasing
 - Universes are closed; no leaking
- What about symbiotic infections? (e.g. Nimda)
 - When a universe is created it can be made open it to multiple outside influences
 - Common use: a fraction of all traffic is directed to a shared universe with draconian export rules

Overall challenges for honeyfarms

- **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 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

Summary

- Reacting to outbreaks over Internet-connected hosts
 - Requires high speed and high coverage
- Different detection strategies, monitoring approaches
 - Line-rate network-based content sifting (sub-ms signatures)
 - Large scale high-fidelity honeyfarm (scales to >100k IPs per host)
- Smart bad guys can still avoid both kinds of approaches

Questions



<http://www.ccied.org/>