CSE 454

Security III
“Outbreak!”

Internet Outbreaks: Epidemiology and Defenses

Stefan Savage

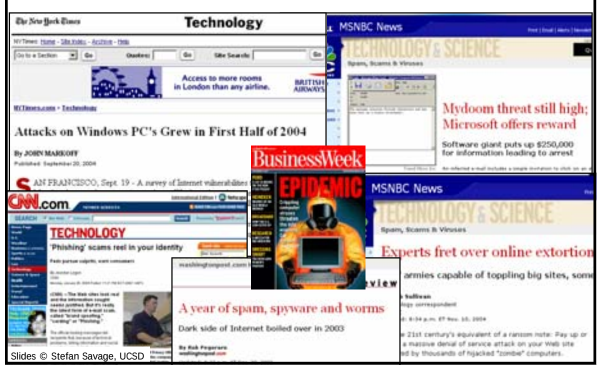
Cooperative Center for Internet Epidemiology and Defenses
Department of Computer Science & Engineering
University of California at San Diego

In collaboration with Cristian Estan, Justin Ma, David Moore, Vern Paxson (ICSI), Colleen Shannon, Sumeet Singh, Alex Snoeren, Stuart Staniford (Nevis), Amin Vahdat, George Varghese, Geoff Voelker, Michael Vrable, Nick Weaver (ICSI)

Collaborative Center for Internet Epidemiology and Defenses (CCIED)

- **Joint project (UCSD/ICSI)**
  - Other PIs: Vern Paxson, Nick Weaver, Geoff Voelker, George Varghese
  - ~15 staff and students in addition
  - Funded by NSF with additional support from Microsoft, Intel, HP, and UCSD’s CNS
- **Three key areas of interest**
  - Infrastructure and analysis for understanding large-scale Internet threads
  - Automated defensive technologies
  - Forensic and legal requirements

How Chicken Little sees the Internet…



Why Chicken Little is a naïve optimist

- Imagine the following species:
  - Poor genetic diversity; heavily inbred
  - Lives in “hot zone”; thriving ecosystem of infectious pathogens
  - Instantaneous transmission of disease
  - Immune response 10-1M times slower
  - Poor hygiene practices
- **What would its long-term prognosis be?**
- What if diseases were designed…
  - Trivial to create a *new* disease
  - Highly profitable to do so

Threat transformation

- **Traditional threats**
  - Attacker manually targets high-value system/resource
  - Defender increases cost to compromise high-value systems
  - Biggest threat: insider attacker

- **Modern threats**
  - Attacker uses automation to target **all** systems at once (can filter later)
  - Defender must defend **all** systems at once
  - Biggest threats: software vulnerabilities & naïve users

Large-scale technical enablers

- ***Unrestricted connectivity***
  - Large-scale adoption of IP model for networks & apps
- ***Software homogeneity & user naiveté***
  - Single bug = mass vulnerability in millions of hosts
  - Trusting users ("ok") = mass vulnerability in millions of hosts
- **Few meaningful defenses**
- **Effective anonymity (minimal risk)**

Driving Economic Forces

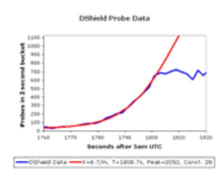
- No longer just for fun, but for profit
  - SPAM forwarding (MyDoom.A backdoor, SoBig), Credit Card theft (Korgo), DDoS extortion, etc…
  - Symbiotic relationship: worms, bots, SPAM, etc
  - Fluid third-party exchange market (**millions** of hosts for sale)
    - Going rate for SPAM proxying 3 -10 cents/host/week
    - Seems small, but 25k botnet gets you $40k-130k/yr
  - Generalized search capabilities are next
- "Virtuous" economic cycle
  - The bad guys have large incentive to get better

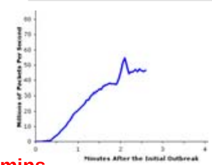
Today's focus: Outbreaks

- Outbreaks?
  - Acute epidemics of infectious malcode designed to actively *spread* from host to host over the network
  - E.g. Worms, viruses (ignore pedantic distinctions)
- Why epidemics?
  - Epidemic spreading is the fastest method for large-scale network compromise
- Why fast?
  - Slow infections allow much more time for detection, analysis, etc (traditional methods may cope)

A pretty fast outbreak: 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)

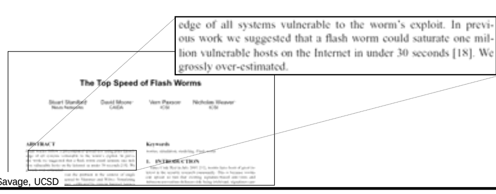




See: Moore et al, IEEE Security & Privacy, 1(4), 2003 for more details

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!

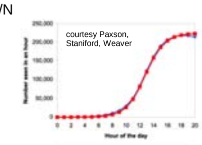


How to think about worms

- Reasonably 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

$$\frac{dI}{dt} = \beta \frac{IS}{N} \qquad \frac{dS}{dt} = -\beta \frac{IS}{N} \quad \rightarrow \quad \frac{di}{dt} = \beta i (1-i)$$

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



What's important?

- There are lots of improvements to the model…
  - Chen et al, *Modeling the Spread of Active Worms*, Infocom 2003 (discrete time)
  - Wang et al, *Modeling Timing Parameters for Virus Propagation on the Internet* , ACM WORM '04 (delay)
  - Ganesh et al, *The Effect of Network Topology on the Spread of Epidemics*, Infocom 2005 (topology)
- … but the bottom line is the same. We 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

What can be done?

- Reduce the number of susceptible hosts
  - **Prevention**, reduce S(t) while I(t) is still small (ideally reduce S(0))
- Reduce the contact rate
  - **Containment**, reduce ß while I(t) is still small

Prevention: Software Quality

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

Prevention: Software Heterogeneity

- **Goal:** reduce impact of vulnerability
- Use software diversity to tolerate attack
  - Exploit *existing* heterogeneity
    - Junqueria et al, *Surviving Internet Catastrophes*, USENIX '05
  - Create *Artificial* heterogeneity (hot topic)
    - Forrest et al, *Building Diverse Computer Systems*, HotOS '97
    - Large contemporary literature
- Open questions: class of vulnerabilities that can be masked, strength of protection, cost of support

Prevention: Software Updating

- **Goal:** reduce window of vulnerability
- Most worms exploit known vulnerability (1 day -> 3 months)
  - Window shrinking: automated patch->exploit
  - Patch deployment challenges, downtime, Q/A, etc
  - Rescorla, *Is finding security holes a good idea?*, WEIS '04
- Network-based filtering: 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
  - Symantec: Generic Exploit Blocking

proactive

- Automated patch creation: fix the vulnerability on-line
  - Sidiroglou et al, *Building a Reactive Immune System for Software Services*, USENIX '05
- Anti-worms: block the vulnerability and propagate
  - Castaneda et al, *Worm vs WORM: Preliminary Study of an Active counter-Attack Mechanism*, WORM '04

reactive

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…
  - Automated version of what they do by hand at NSF
- Cisco Network Admission Control (NAC)

What can be done?

- Reduce the number of susceptible hosts
  - **Prevention**, reduce S(t) while I(t) is still small (ideally reduce S(0))
- Reduce the contact rate
  - **Containment**, reduce ß while I(t) is still small

Containment

- Reduce contact rate
- **Slow down**
  - Throttle connection rate to slow spread
    - Twycross & Williamson, *Implementing and Testing a Virus Throttle*, USENIX Sec '03
  - Important capability, but worm still spreads…
- **Quarantine**
  - Detect and block worm

Defense requirements

- 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?

Methodology

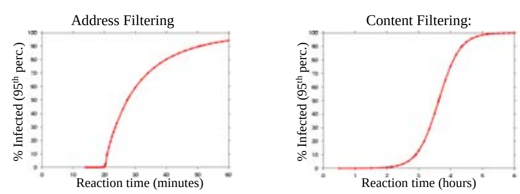
- **Simulate spread of worm across Internet topology**
  - Infected hosts *attempt* to spread at a fixed rate (probes/sec)
  - Target selection is uniformly random over IPv4 space
- **Source data**
  - Vulnerable hosts: 359,000 IP addresses of CodeRed v2 *victims*
  - Internet topology: AS routing topology 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

See: Moore et al, *Internet Quarantine: Requirements for Containing Self-Propagating Code*, Infocom 2003 for more details

Naïve model: Universal deployment

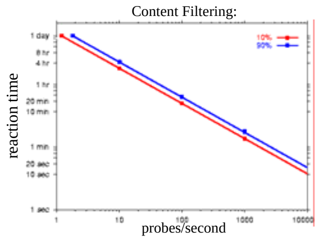
- Assume **every host** employs the containment strategy
- Two containment strategies :
  - ***Address filtering:***
    - Block traffic from malicious source IP addresses
    - Reaction time is relative to each infected host
    - **MUCH** easier to implement
  - ***Content filtering:***
    - Block 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 filtering**: reaction time must be < 25 minutes.
  - **Content filtering**: reaction time must be < 3 hours

How sensitive is reaction time to worm probe rate?

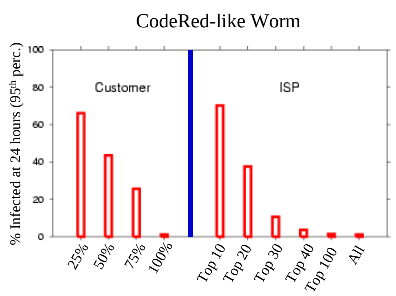


- Reaction times must be fast when probe rates get high:
  - **10 probes/sec: reaction time must be < 3 hours**
  - **1000 probes/sec: reaction time must be < 2 minutes**

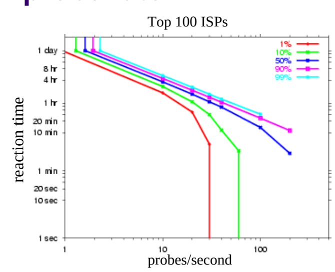
Limited network deployment

- 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?



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.

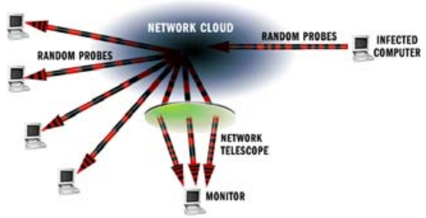
Defense requirements summary

- **Reaction time**
  - Required reaction times are a couple minutes or less for CR-style worms (seconds for worms like Slammer)
- **Containment strategy**
  - Content filtering is far more effective than address blacklisting for a given reaction speed
- **Deployment scenarios**
  - Need nearly all customer networks to provide containment
  - Need at least top 40 ISPs provide containment; top 100 ideal
- Is this possible? Lets see…

Outbreak Detection/Monitoring

- Two classes of detection
  - **Scan detection**: detect that host is infected by infection attempts
  - **Signature inference**: automatically identify content signature for exploit (sharable)
- Two classes of monitors
  - Ex-situ: "canary in the coal mine"
    - Network Telescopes
    - HoneyNets/Honeypots
  - In-situ: real activity as it happens

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

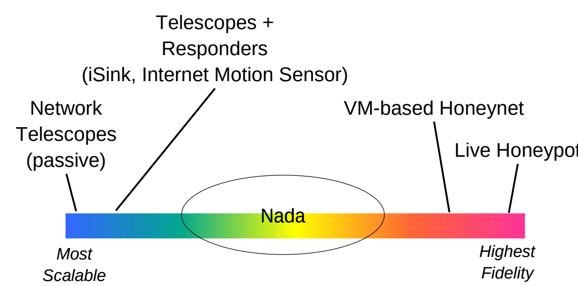
Telescopes + Active Responders

- 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 SYNACK (Internet Motion Sensor), per-protocol responders (iSink)
  - Stateful: Honeyd
  - Can differentiate and fingerprint payload
    - False positives generally low since no regular traffic

HoneyNets

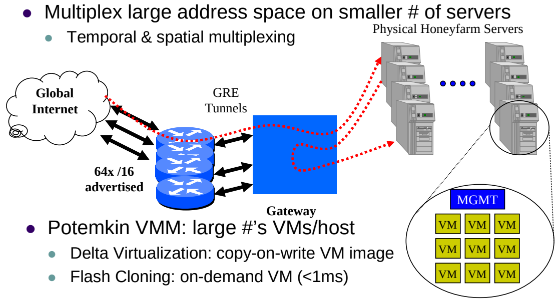
- Problem: don't know what worm/virus would do? No code ever executes after all.
- Solution: redirect scans to real "infectable" hosts (honeypots)
  - Individual hosts or VM-based: Collapsar, HoneyStat, Symantec
  - Can reduce false positives/negatives with host-analysis (e.g. TaintCheck, Vigilante, 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



New CCIED project: large scale high-fidelity honeyfarm

- Goal: emulate significant fraction of Internet hosts (1M)
- Multiplex large address space on smaller # of servers
  - Temporal & spatial multiplexing



- Potemkin VMM: large #'s VMs/host
  - Delta Virtualization: copy-on-write VM image
  - Flash Cloning: on-demand VM (<1ms)

Overall limitations of telescope, honeynet, etc monitoring

- **Depends** on worms scanning it
  - What if they don't scan that range (smart bias)
  - What if they propagate via e-mail, IM?
- Inherent tradeoff between liability exposure and detectability
  - Honeypot detection software exists
- It doesn't necessary reflect what's happening on **your** network (can't count on it for local protection)
- Hence, we're always interested in native detection as well

Scan Detection

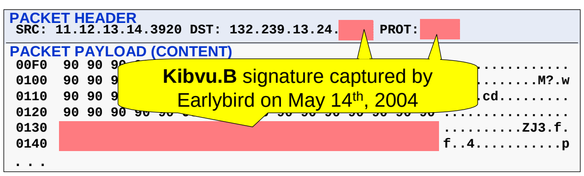
- Idea: detect worm's infection attempts
  - In the small: ZoneAlarm, but how to do in the network?
- Indirect scan detection
  - Wong et al, *A Study of Mass-mailing Worms*, WORM '04
  - Whyte et al. *DNS-based Detection of Scanning Worms in an Enterprise Network,* NDSS '05
- Direct scan detection
  - Weaver et al. *Very Fast Containment of Scanning Worms*, USENIX Sec '04
    - Threshold Random Walk – bias source based on connection success rate (Jung et al); use approximate state for fast hardware implementation
    - Can support multi-Gigabit implementation, detect scan within 10 attempts
    - Few false positives: Gnutella (finding accessing), Windows File Sharing (benign scanning)
  - Venkataraman et al, *New Streaming Algorithms for Fast Detection of Superspreaders*, just recently

Signature inference

- Challenge: need to automatically ***learn*** a content "signature" for each new worm – potentially in less than a second!
- **Singh et al, *Automated Worm Fingerprinting*, OSDI '04**
- Kim et al, *Autograph: Toward Automated, Distributed Worm Signature Detection*, USENIX Sec '04

Approach

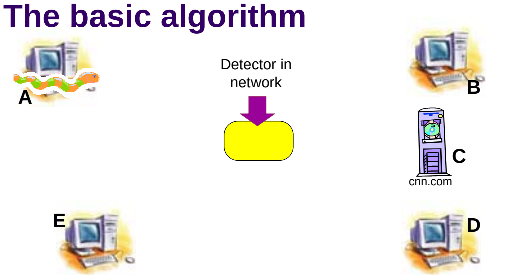
- Monitor network and look for strings common to traffic with worm-like behavior
- Signatures can then be used for content filtering



Content sifting

- Assume there exists some (relatively) unique invariant bitstring *W* across all instances of a particular worm (*true today, not tomorrow...*)
- 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

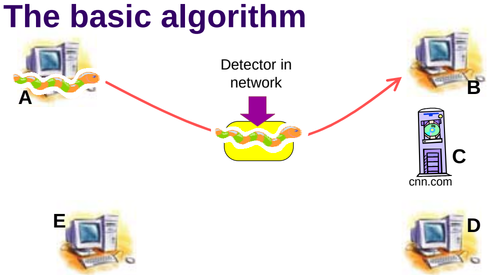


Prevalence Table

Address **Dispersion** Table

Sources	Destinations

The basic algorithm



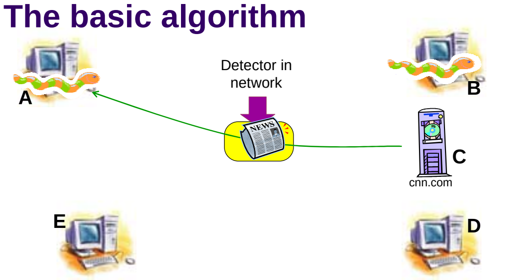
Prevalence Table

(worm)	**1**

Address **Dispersion** Table

Sources	Destinations
1 (A)	**1 (B)**

The basic algorithm



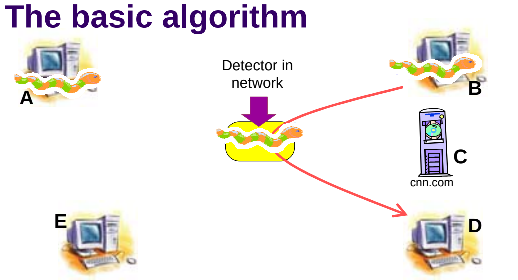
Prevalence Table

	1
	1

Address **Dispersion** Table

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

The basic algorithm



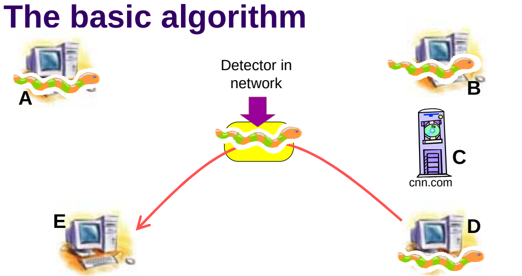
Prevalence Table

	2
	1

Address **Dispersion** Table

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

The basic algorithm



Prevalence Table

	3
	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
    - 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 tables

Kim et al’s solution: Autograph

- Pre-filter flows for those that exhibit scanning behavior (i.e. low TCP connection ratio)
  - HUGE reduction in input, fewer prevalent substrings
  - Don’t need to track dispersion at all
  - Fewer possibilities of false positives
- However, only works with TCP scanning worms
  - Not UDP (Slammer), e-mail viruses (MyDoom), IM-based worms (Bizex), P2P (Benjamin)
- Alternatives? More efficient algorithms.

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

How to subsample?

- **Approach 1: sample packets**
  - If we chose 1 in N, detection will be slowed by N
- **Approach 2: sample at particular byte offsets**
  - Susceptible to simple evasion attacks
  - No guarantee that we will sample same sub-string in every packet
- **Approach 3: sample based on the hash of the substring**

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

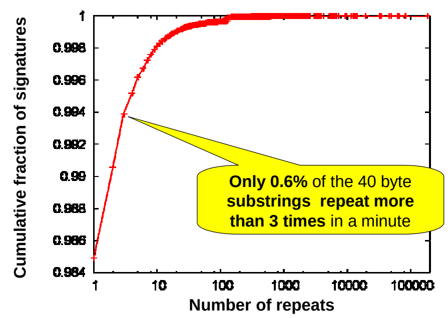
A B C D E F G H I J K

Fingerprint = ...000000

SAMPLE IGNORE IGNORE SAMPLE

- P_{track} → 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_{track} = 99.64% for a 400 byte invariant**

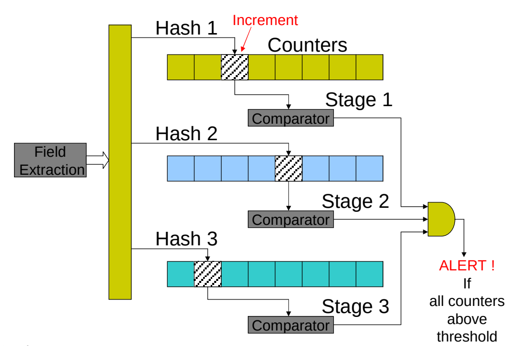
Observation: High-prevalence strings are rare



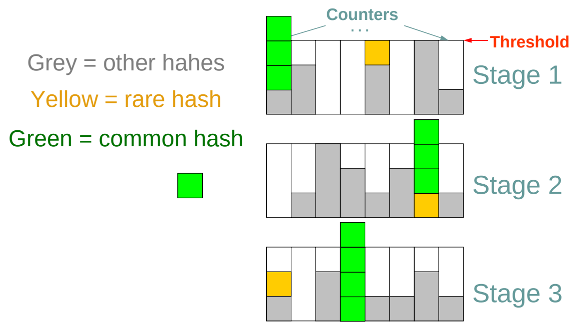
Efficient high-pass filters for content

- 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**
  - Three orders of magnitude memory savings

Finding "heavy hitters" via Multistage Filters



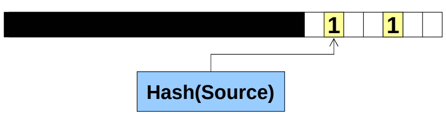
Multistage filters in action



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

Scalable Bitmap Counters



- Hash : based on Source (or Destination)
- Sample : keep only a sample of the bitmap
- Estimate : scale up sampled count
- Adapt : periodically increase scaling factor

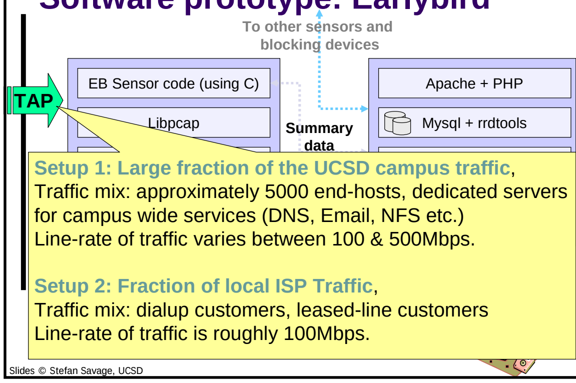
$$\text{Error Factor} = 2/(2^{numBitmaps}-1)$$

- With 3, 32-bit bitmaps, error factor = 28.5%

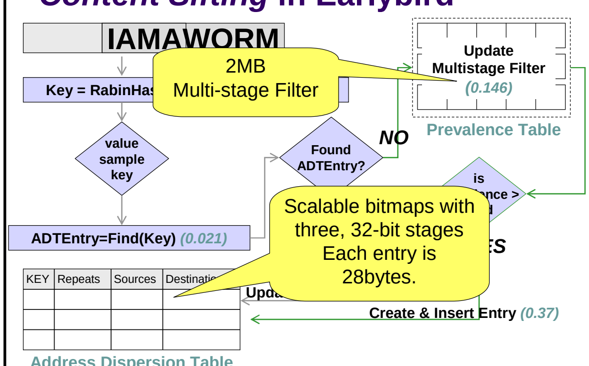
Content sifting summary

- 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



Content Sifting in Earlybird



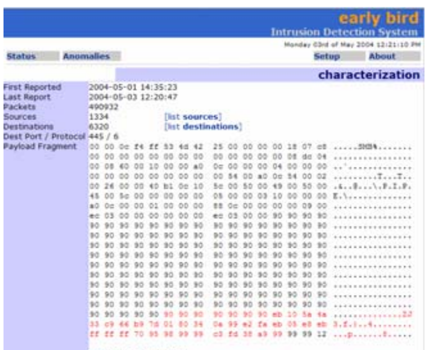
Content sifting overhead

- Mean per-byte processing cost
  - **0.409 microseconds**, without value sampling
  - **0.042 microseconds**, with 1/64 value sampling (~60 microseconds for a 1500 byte packet, can keep up with 200Mbps)
- Additional overhead in per-byte processing cost for flow-state maintenance (if enabled):
  - **0.042 microseconds**

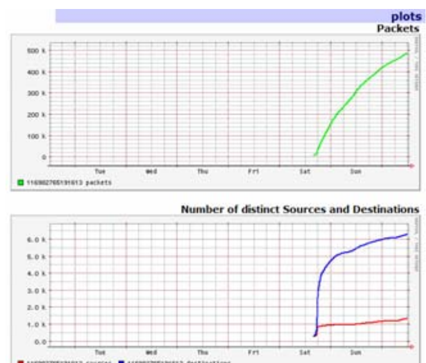
Experience

- 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
- **Known worms detected**:
  - Code Red, Nimda, WebDav, Slammer, Opaserv, …
- **Unknown worms (with no public signatures) detected**:
  - MsBlaster, Bagle, Sasser, Kibvu, …

Sasser

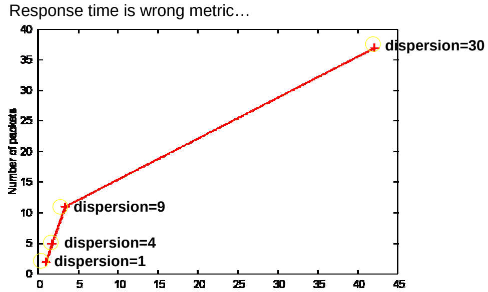


Sasser



Kibvu

- Slower spread (1.5 packets/minute inbound)
- Consequently, slower detection (42mins to dispersion of 30)
- Response time is wrong metric…



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:
```


Limitations/ongoing work

- Variant content
  - Polymorphism, metamorphism
  - Newsom et al, *Polygraph: Automatically Generating Signatures for Polymorphic Worms*, Oakland '05
- Network evasion
  - Normalization at high-speed tricky
- End-to-end encryption vs content-based security
  - Privacy vs security policy
- Self-tuning thresholds
- Slow/stealthy worms
- DoS via manipulation

Summary

- Internet-connected hosts are highly vulnerable to worm outbreaks
  - Millions of hosts can be "taken" before anyone realizes
  - If only 10,000 hosts are targeted, no one may notice
- Prevention is a critical element, but there will always be outbreaks
- Containment requires fully automated response
- Scaling issues favor network-based defenses
- Different detection strategies, monitoring approaches
  - Very active research community
- Content sifting: automatically sift bad traffic from good