15-744: Computer Networking

L-23 Worms

Overview

- Worm propagation
- Worm signatures

Threat Model

Traditional

- High-value targets
- Insider threats

Worms & Botnets

- Automated attack of millions of targets
- Value in aggregate, not individual systems
- Threats: Software vulnerabilities; naïve users

... and it's profitable

- Botnets used for
  - Spam (and more spam)?
  - Credit card theft
  - DDoS extortion
- Flourishing Exchange market
  - Spam proxying: 3-10 cents/host/week
  - 25k botnets: $40k - $130k/year
  - Also for stolen account compromised machines, credit cards, identities, etc. (be worried)?

Why is this problem hard?

- Monoculture: little “genetic diversity” in hosts
- Instantaneous transmission: Almost entire network within 500ms
- Slow immune response: human scales (10x-1Mx slower!)?
- Poor hygiene: Out of date / misconfigured systems; naïve users
- Intelligent designer ... of pathogens
- Near-Anonymitity

Code Red I v1

- July 12th, 2001
- Exploited a known vulnerability in Microsoft’s Internet Information Server (IIS)
  - Buffer overflow in a rarely used URL decoding routine – published June 18th
- 1st – 19th of each month: attempts to spread
  - Random scanning of IP address space
  - 99 propagation threads, 100th defaced pages on server
  - Static random number generator seed
    - Every worm copy scans the same set of addresses

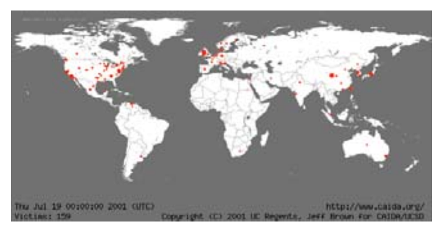
    → Linear growth

Code Red I v1

- 20th – 28th of each month: attacks
  - DDOS attack against 198.137.240.91 (www.whitehouse.gov)
- Memory resident – rebooting the system removes the worm
  - However, could quickly be reinfected

Code Red I v2

- July 19th, 2001
- Largely same codebase – same author?
- Ends website defacements
- Fixes random number generator seeding bug
  - Scanned address space grew exponentially
  - 359,000 hosts infected in 14 hours
  - Compromised almost all vulnerable IIS servers on internet



Analysis of Code Red I v2

- Random Constant Spread model
- Constants
  - N = total number of vulnerable machines
  - K = initial compromise rate, per hour
  - T = Time at which incident happens
- Variables
  - a = proportion of vulnerable machines compromised
  - t = time in hours

Analysis of Code Red I v2

$$Nda = (Na)K(1-a)dt.$$

$$\frac{da}{dt} = Ka(1-a)$$

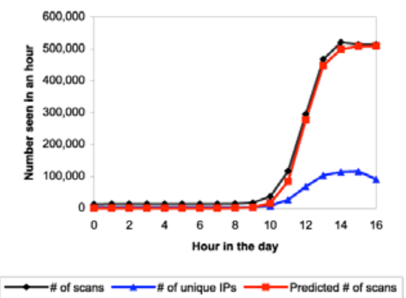
N = total number of vulnerable machines
K = initial compromise rate, per hour
T = Time at which incident happens

$$a = \frac{e^{K(t-T)}}{1+e^{K(t-T)}},$$

Variables
a = proportion of vulnerable machines compromised
t = time in hours

"Logistic equation"
Rate of growth of epidemic in finite systems when all entities have an equal likelihood of infecting any other entity

Code Red I v2 – Plot



- K = 1.8
- T = 11.9

Hourly probe rate data for inbound port 80 at the Chemical Abstracts Service during the initial outbreak of Code Red I on July 19th, 2001.

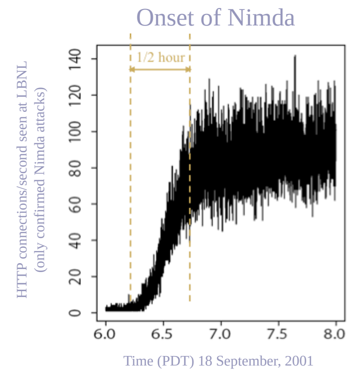
Improvements: Localized scanning

- Observation: Density of vulnerable hosts in IP address space is not uniform
- Idea: Bias scanning towards local network
- Used in CodeRed II
  - P=0.50: Choose address from local class-A network (/8)
  - P=0.38: Choose address from local class-B network (/16)
  - P=0.12: Choose random address
- Allows worm to spread more quickly

Code Red II (August 2001)

- Began : August 4th, 2001
- Exploit : Microsoft IIS webservers (buffer overflow)
- Named "Code Red II" because :
  - It contained a comment stating so. However the codebase was new.
- Infected IIS on windows 2000 successfully but caused system crash on windows NT.
- Installed a root backdoor on the infected machine.

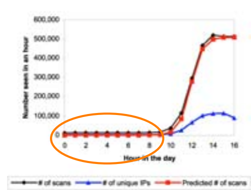
Improvements: Multi-vector



- Idea: Use multiple propagation methods simultaneously
- Example: Nimda
  - IIS vulnerability
  - Bulk e-mails
  - Open network shares
  - Defaced web pages
  - Code Red II backdoor

Better Worms: Hit-list Scanning

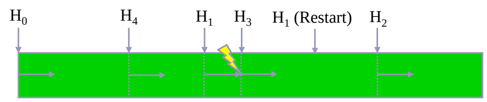
- Worm takes a long time to "get off the ground"
- Worm author collects a list of, say, 10,00 vulnerable machines
- Worm initially attempts to infect these hosts



How to build Hit-List

- Stealthy randomized scan over number of months
- Distributed scanning via botnet
- DNS searches – e.g. assemble domain list, search for IP address of mail server in MX records
- Web crawling spider similar to search engines
- Public surveys – e.g. Netcraft
- Listening for announcements – e.g. vulnerable IIS servers during Code Red I

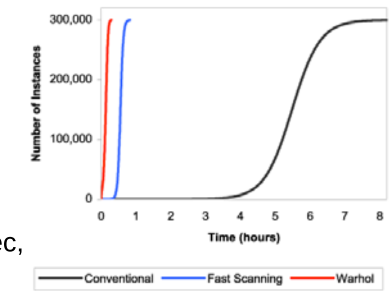
Better Worms: Permutation scanning



- Problem: Many addresses are scanned multiple times
- Idea: Generate random permutation of all IP addresses, scan in order
  - Hit-list hosts start at their own position in the permutation
  - When an infected host is found, restart at a random point
  - Can be combined with divide-and-conquer approach

Warhol Worm

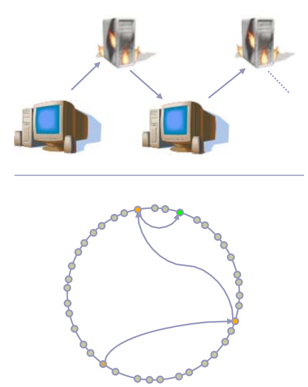
- Simulation shows that employing the two previous techniques, can attack 300,000 hosts in less than 15 minutes
- Conventional = 10 scans/sec
- Fast Scanning = 100 scans/sec
- Warhol = 100 scans/sec,
- Permutation scanning and 10,000 entry hit list



Flash worms

- A flash worm would start with a hit list that contains most/all vulnerable hosts
- Realistic scenario:
  - Complete scan takes 2h with an OC-12
  - Internet warfare?
- Problem: Size of the hit list
  - 9 million hosts ⇒ 36 MB
  - Compression works: 7.5MB
  - Can be sent over a 256kbps DSL link in 3 seconds
- Extremely fast:
  - Full infection in tens of seconds!

Surreptitious worms



- Idea: Hide worms in inconspicuous traffic to avoid detection
- Leverage P2P systems?
  - High node degree
  - Lots of traffic to hide in
  - Proprietary protocols
  - Homogeneous software
  - Immense size (30,000,000 Kazaa downloads!)

Example Outbreak: SQL Slammer (2003)

- Single, small UDP packet exploit (376 b)
- First ~1min: classic random scanning
  - Doubles # of infected hosts every ~8.5sec
  - (In comparison: Code Red doubled in 40min)
- After 1min, starts to saturate access b/w
  - Interferes with itself, so it slows down
  - By this point, was sending 20M pps
  - Peak of 55 million IP scans/sec @ 3min
- 90% of Internet scanned in < 10mins
- Infected ~100k or more hosts

Prevention

- Get rid of the or permute vulnerabilities
  - (e.g., address space randomization)
  - makes it harder to compromise
- Block traffic (firewalls)
  - only takes one vulnerable computer wandering between in & out or multi-homed, etc.
- Keep vulnerable hosts off network
  - incomplete vuln. databases & 0-day worms
- Slow down scan rate
  - Allow hosts limited # of new contacts/sec.
  - Can slow worms down, but they do still spread
- Quarantine
  - Detect worm, block it

23

Overview

- Worm propagation
- Worm signatures

24

Context

- Worm Detection
  - Scan detection
  - Honeypots
  - Host based behavioral detection
  - Payload-based ???

Worm behavior

- Content Invariance
  - Limited polymorphism e.g. encryption
  - key portions are invariant e.g. decryption routine
- Content Prevalence
  - invariant portion appear frequently
- Address Dispersion
  - # of infected distinct hosts grow overtime
  - reflecting different source and dest. addresses

Signature Inference

- Content prevalence: Autograph, EarlyBird, etc.
  - Assumes some content invariance
  - Pretty reasonable for starters.
  - Goal: Identify "attack" substrings
    - Maximize detection rate
    - Minimize false positive rate

Content Sifting

- For each string ***w***, maintain
  - prevalence(***w***): Number of times it is found in the network traffic
  - sources(***w***): Number of unique sources corresponding to it
  - destinations(***w***): Number of unique destinations corresponding to it
- If thresholds exceeded, then block(***w***)

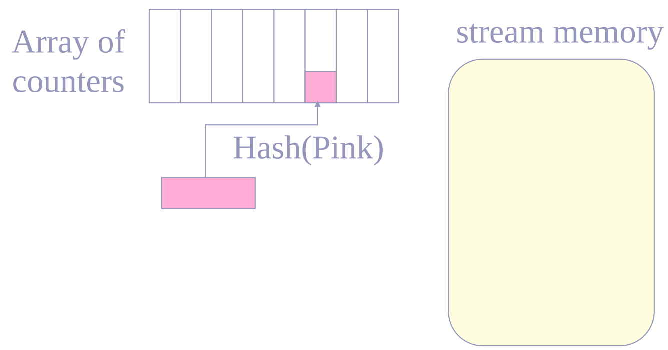
Issues

- How to compute prevalence(***w***), sources(***w***) and destinations(***w***) *efficiently*?
- Scalable
- Low memory and CPU requirements
- Real time deployment over a Gigabit link

Estimating Content Prevalence

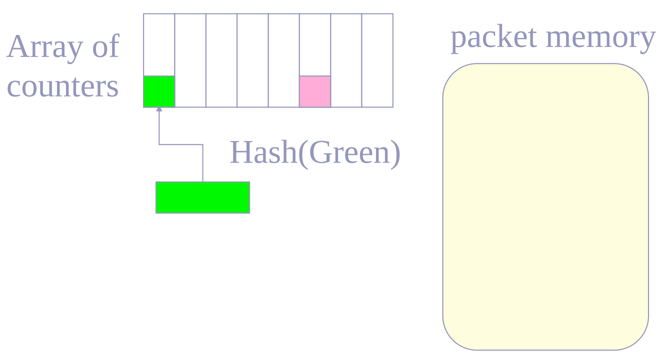
- Table[payload]
  - 1 GB table filled in 10 seconds
- Table[hash[payload]]
  - 1 GB table filled in 4 minutes
  - Tracking millions of ants to track a few elephants
  - Collisions...false positives

Multistage Filters

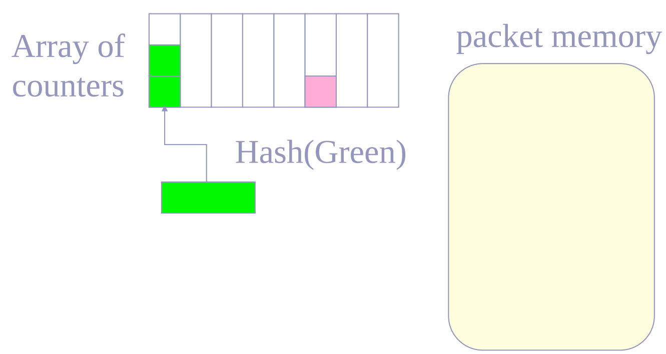


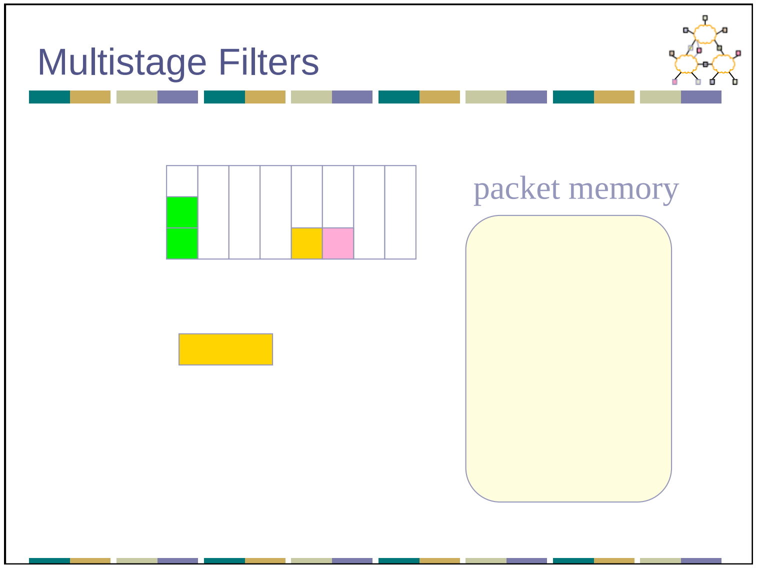
[Singh et al. 2002]

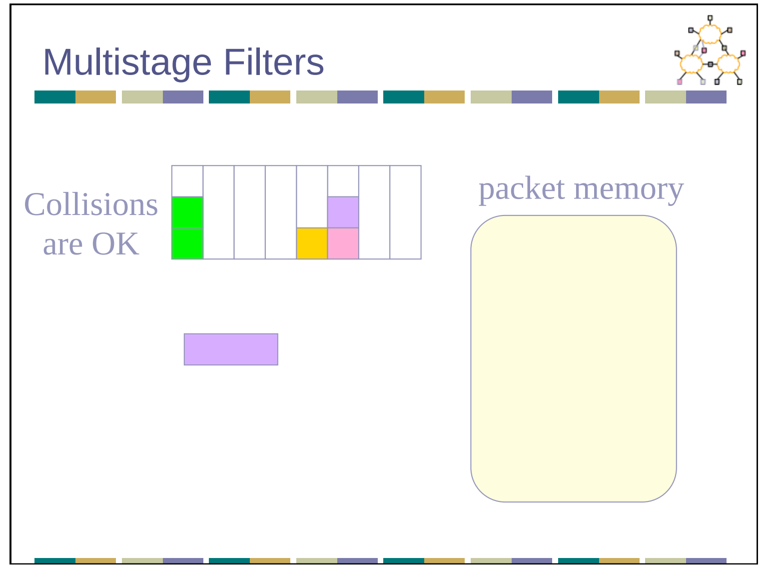
Multistage Filters

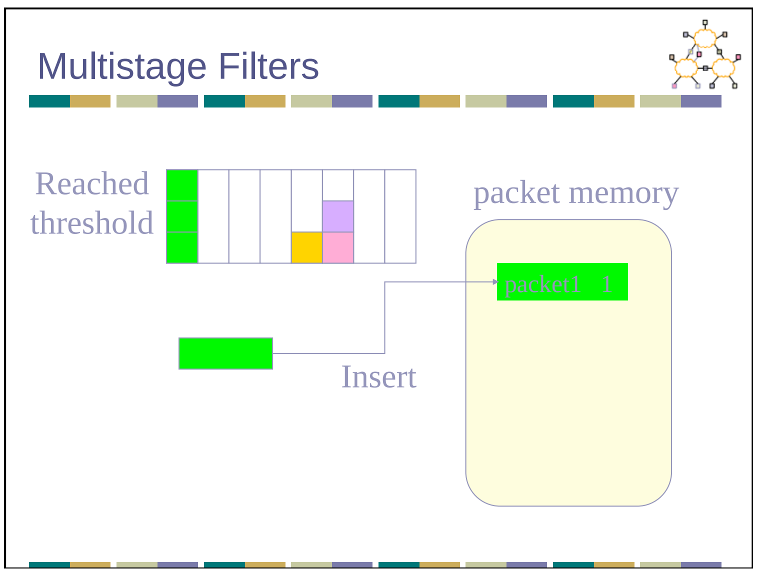


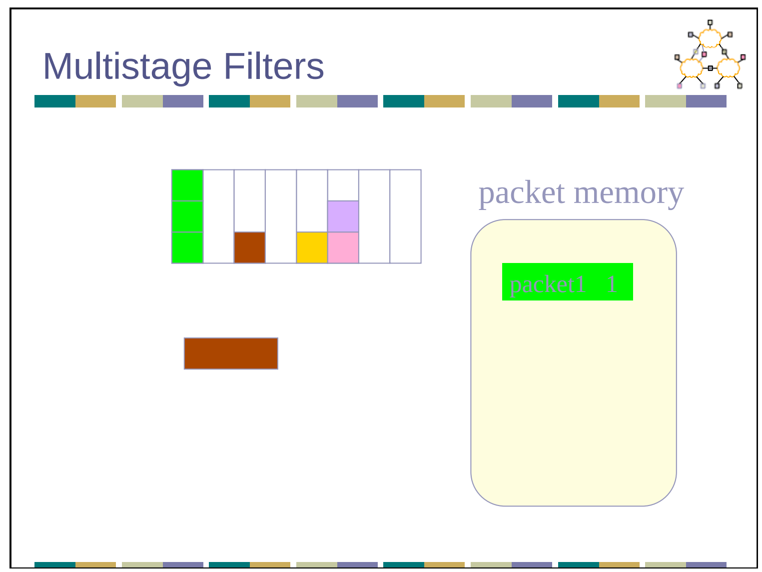
Multistage Filters

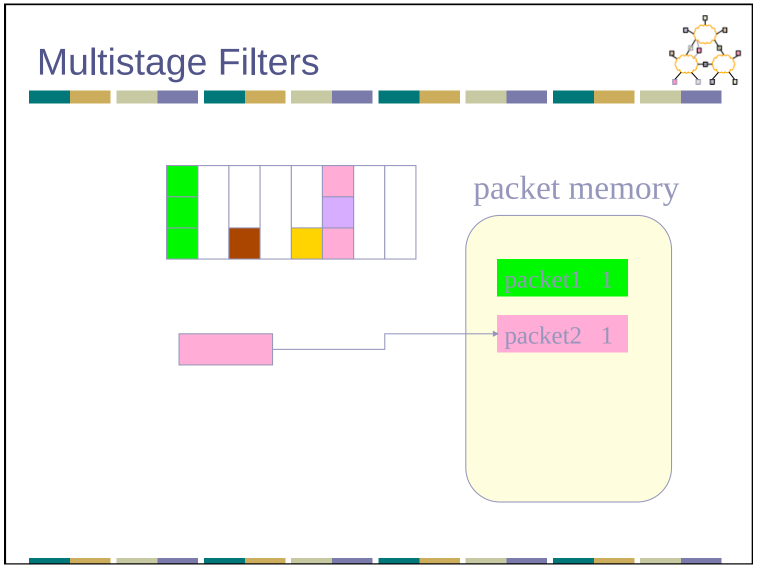


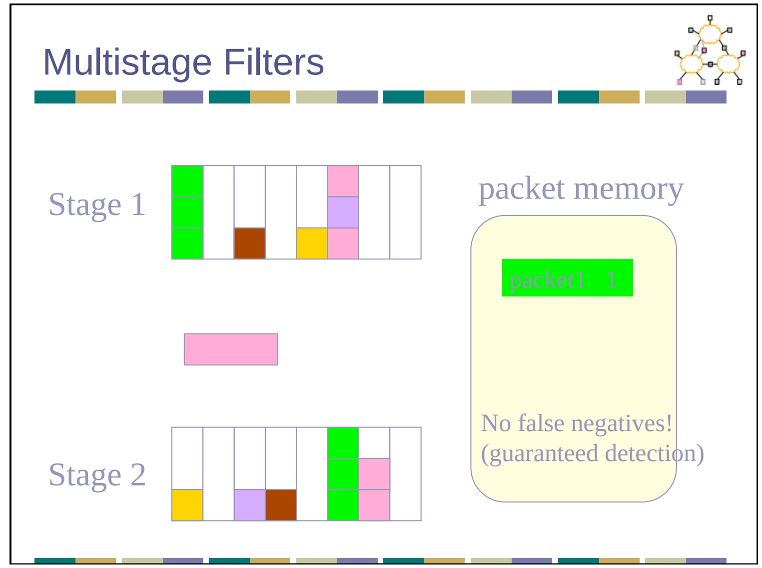

Multistage Filters
packet memory

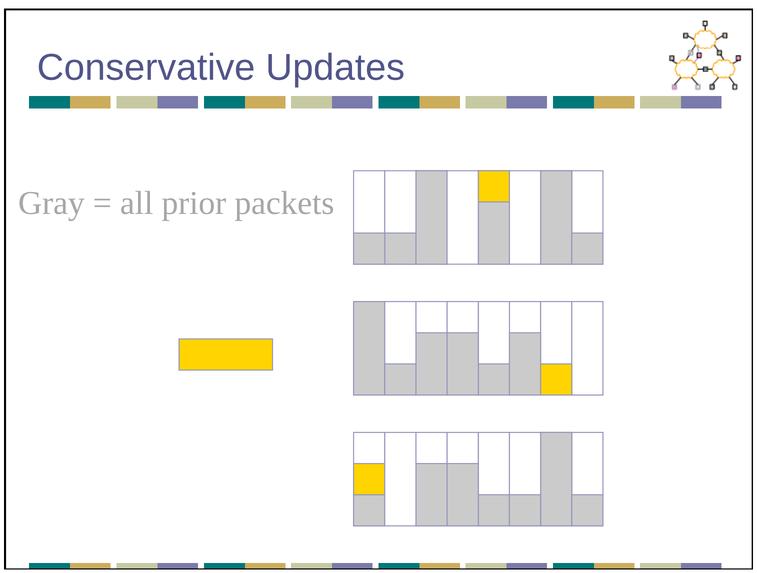

Multistage Filters
Collisions are OK
packet memory

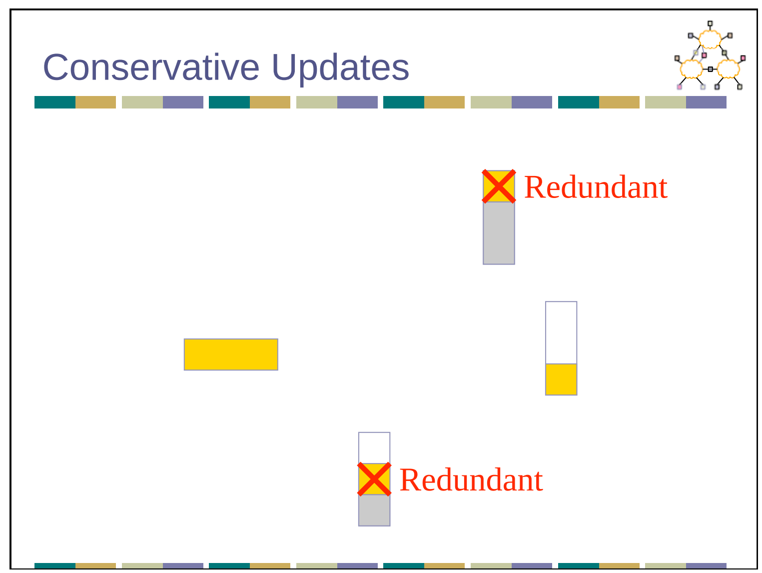

Multistage Filters
Reached threshold
packet memory
packet1 1
Insert


Multistage Filters
packet memory
packet1 1

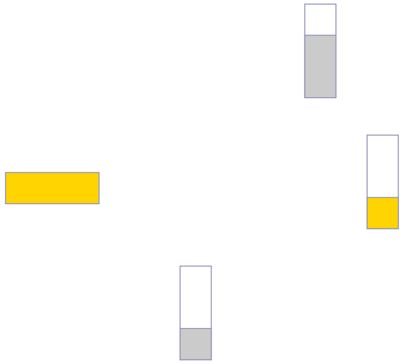

Multistage Filters
packet memory
packet1 1
packet2 1


Multistage Filters
Stage 1
packet memory
packet1 1
No false negatives!
(guaranteed detection)
Stage 2


Conservative Updates
Gray = all prior packets


Conservative Updates
Redundant
Redundant

Conservative Updates



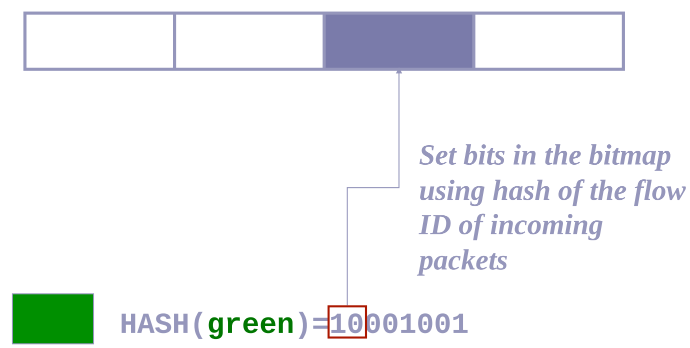
Value Sampling

- The problem: ***s-b+1*** *sub*strings
- Solution: Sample
- But: Random sampling is not good enough
- Trick: Sample only those substrings for which the fingerprint matches a certain pattern

sources(*w*) & destinations(*w*)

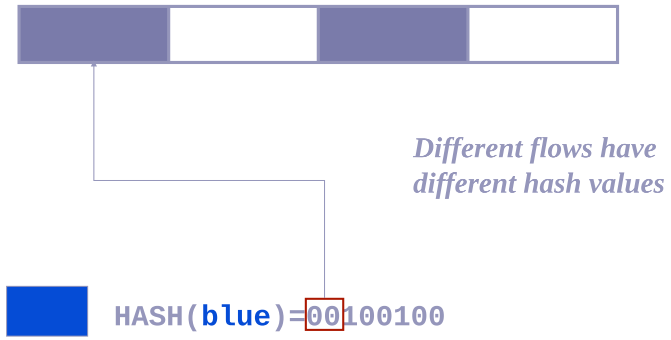
- Address Dispersion
- Counting distinct elements vs. repeating elements
- Simple list or hash table is too expensive
- Key Idea: Bitmaps
- Trick : Scaled Bitmaps

Bitmap counting – direct bitmap

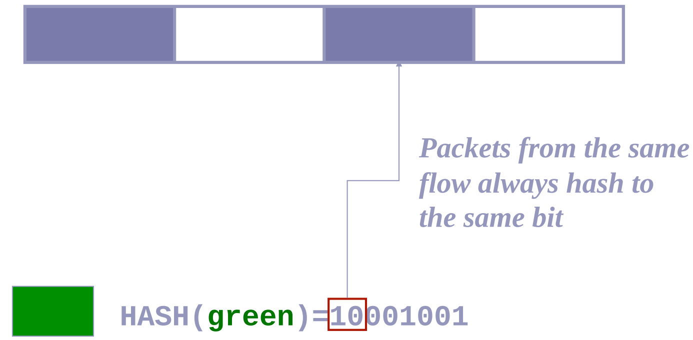


[Estan et al. 2003]

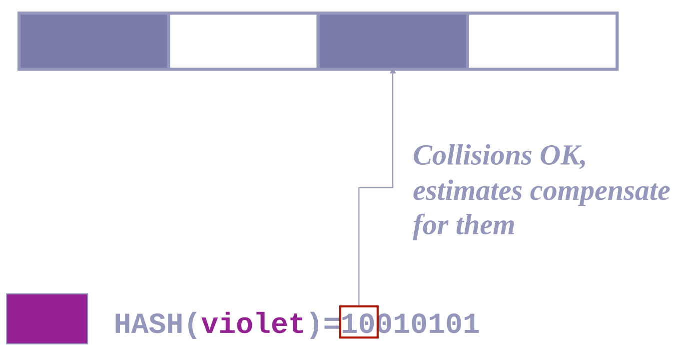
Bitmap counting – direct bitmap



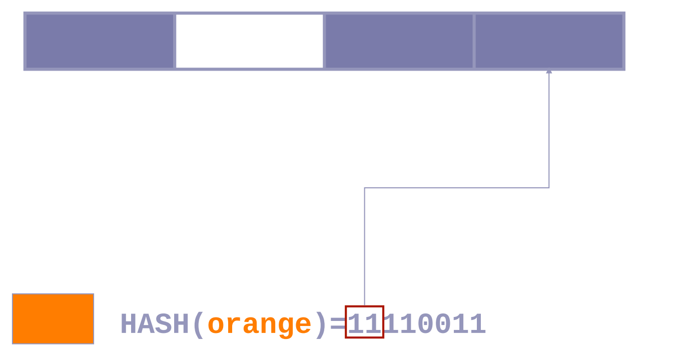
Bitmap counting – direct bitmap



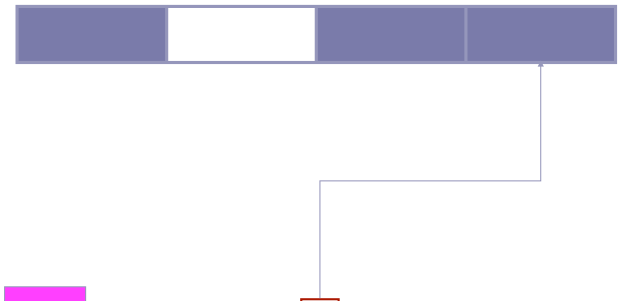
Bitmap counting – direct bitmap



Bitmap counting – direct bitmap

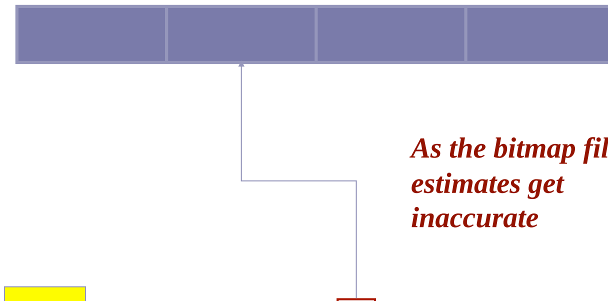


Bitmap counting – direct bitmap



HASH(pink)=11100000

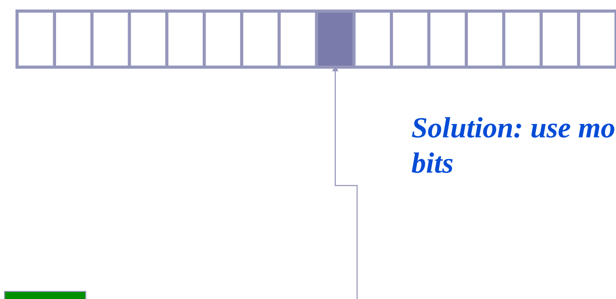
Bitmap counting – direct bitmap



As the bitmap fills up, estimates get inaccurate

HASH(yellow)=01100011

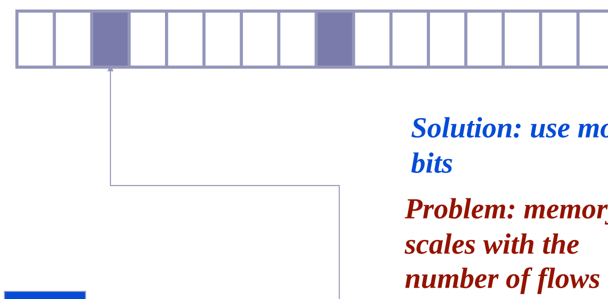
Bitmap counting – direct bitmap



Solution: use more bits

HASH(green)=10001001

Bitmap counting – direct bitmap



Solution: use more bits

Problem: memory scales with the number of flows

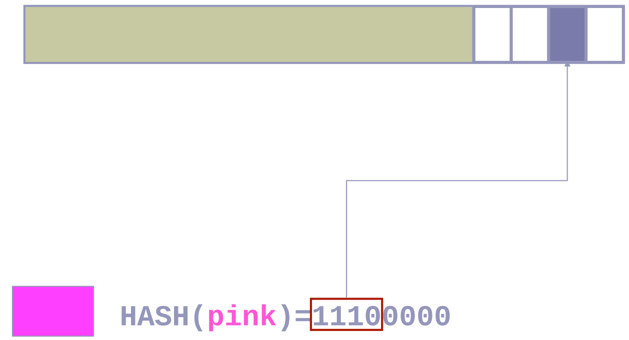
HASH(blue)=00100100

Bitmap counting – virtual bitmap

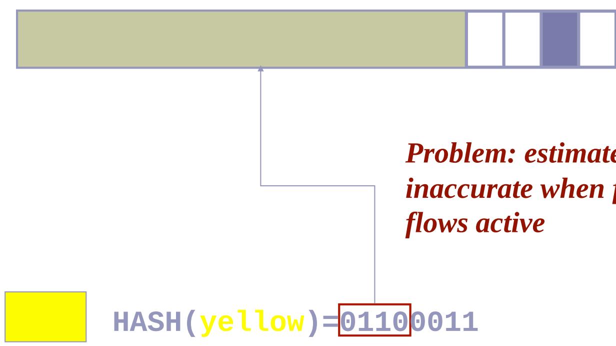
Solution: a) store only a portion of the bitmap

b) multiply estimate by scaling factor

Bitmap counting – virtual bitmap

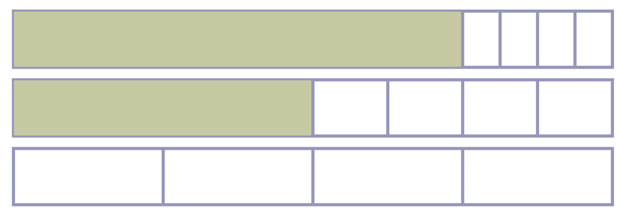


Bitmap counting – virtual bitmap



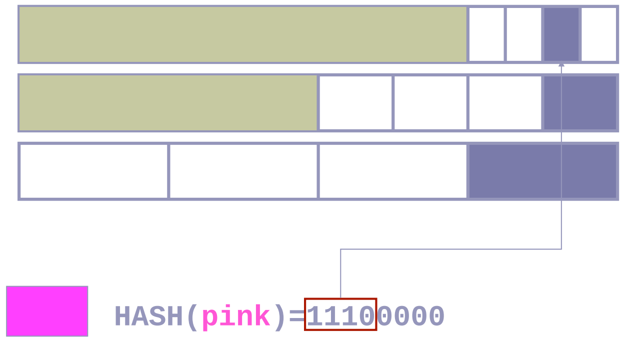
Problem: estimate inaccurate when few flows active

Bitmap counting – multiple bmps

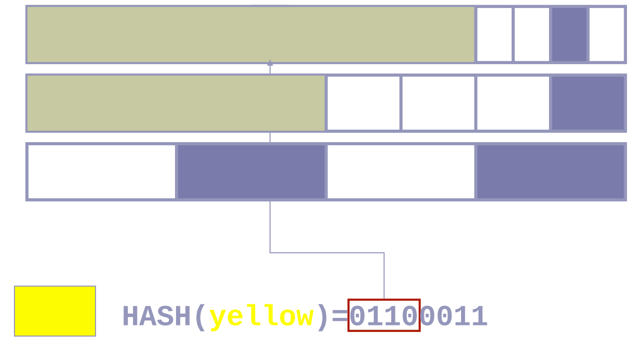


Solution: use many bitmaps, each accurate for a different range

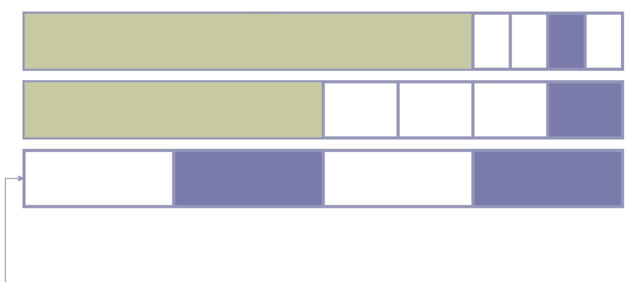
Bitmap counting – multiple bmps



Bitmap counting – multiple bmps

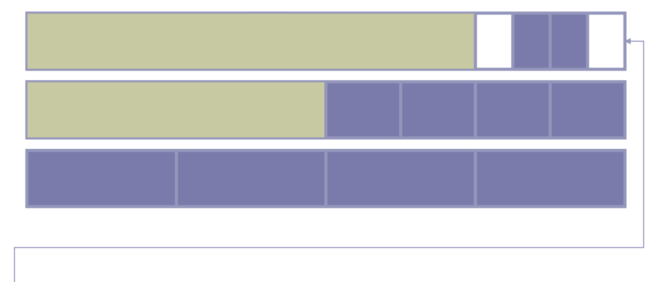


Bitmap counting – multiple bmps



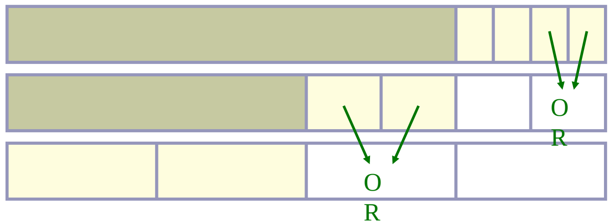
Use this bitmap to estimate number of flows

Bitmap counting – multiple bmps



Use this bitmap to estimate number of flows

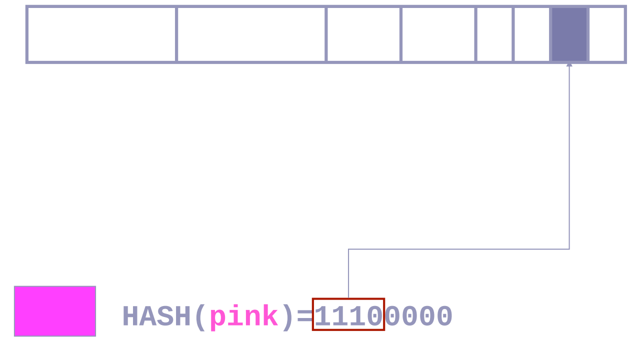
Bitmap counting – multires. bmp



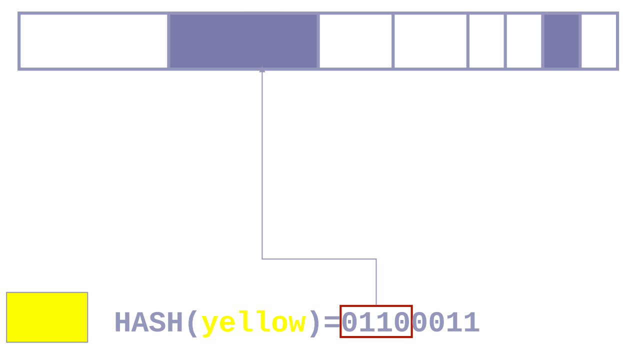
Problem: must update up to three bitmaps per packet

Solution: combine bitmaps into one

Bitmap counting – multires. bmp



Bitmap counting – multires. bmp

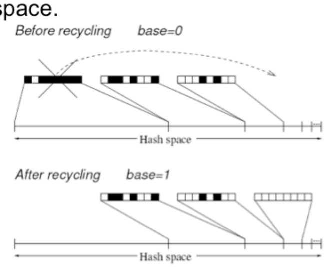


Multiresolution Bitmaps

- Still too expensive to scale
- Scaled bitmap
  - Recycles the hash space with too many bits set
  - Adjusts the scaling factor according

Scaled Bitmap

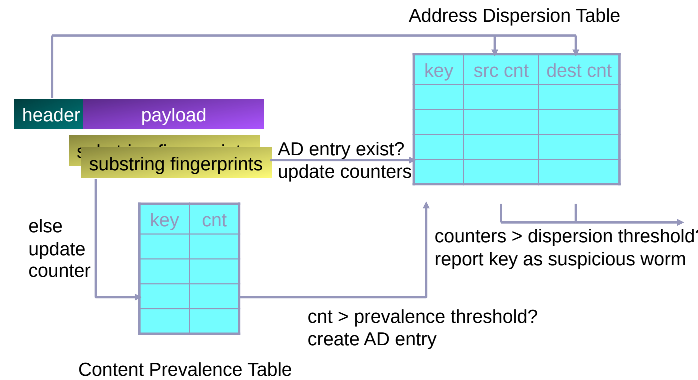
- Idea: Subsample the range of hash space
- How it works?
  - multiple bitmaps each mapped to progressively smaller and smaller portions of the hash space.
  - bitmap recycled if necessary.



Result

Roughly 5 time less memory + actual estimation of address dispersion

Putting It Together



Putting It Together

- Sample frequency: 1/64
- String length: 40
- Use 4 hash functions to update prevalence table
  - Multistage filter reset every 60 seconds

Parameter Tuning

- Prevalence threshold: 3
  - Very few signatures repeat
- Address dispersion threshold
  - 30 sources and 30 destinations
  - Reset every few hours
  - Reduces the number of reported signatures down to ~25,000

Parameter Tuning

- Tradeoff between and speed and accuracy
  - Can detect Slammer in 1 second as opposed to 5 seconds
    - With 100x more reported signatures

False Negatives in EB

- False Negatives
  - Very hard to prove...
- Earlybird detected all worm outbreaks reported on security lists over 8 months
- EB detected all worms detected by Snort (signature-based IDS)?
- And some that weren't

False Positives in EB

- Common protocol headers
  - HTTP, SMTP headers
  - p2p protocol headers
- Non-worm epidemic activity
  - Spam
  - BitTorrent (!)
- Solution:
  - Small whitelist...