Advanced Network Security

Worms and Viruses

Amir Mahdi Sadeghzadeh, Ph.D.

Most slides have been adapted from Mehdi Kharrazi, CE40-817, Sharif University of Technology, 2015

Worms and Viruses

- What are they?
- How do they spread?
- What can be done about them?

Viruses

- "Infected" program (or floppy)
- When program is executed, it performs its normal function
- It also infects some other programs
- It may carry an extra "payload" that performs other functions

Worms

- Similar to viruses, but they spread between machines
- Some are fully automatic; some require manual intervention to spread
- Some exploit bugs; others use social engineering

Classic Worms

Early Worms

- IBM Christmas Card "Virus", December 1987
- Morris Internet Worm, November 1988
- Most worms since then have emulated one or both of those

Modern Worms

- Most resemble either the Christmas card worm or the Internet worm
- Today's email worms try to trick the user with tempting Subject: lines
  - million dollar award, software "updates", etc.
  - A notable one: "Osama bin Laden Captured", with an attached "video"
  - Some pose as anti-virus software updates. . .
- Can get through many firewalls

Stealthiness

- Deceptive filenames for the attachments
- Add a phony extension before the real one: Saddam_Capture.jpg.exe
- Hide in an encrypted .zip file, with the password in the body of the email
- Many strategies for hiding on hosts, including strange filenames, etc.

Trust Patterns

- Preferentially attack within the same network — may be on the inside of a firewall
- Exploit shared disks
- Mass-mailing worms rely on apparent trustworthy source

Spreading Via Buggy Code

- Exploit many different (Windows) bugs
- Can spread much more quickly
- Slammer spread about as far is it could in just 15 minutes, and clogged much of the Internet

The Slammer Worm

- Exploited a bug in Microsoft's SQL server
- Used UDP, not TCP — a single 376-byte packet to UDP port 1434 could infect a machine!
- Use of UDP instead of TCP let it spread much faster — one packet, from a forged source address, instead of a three-way handshake, payload transmission, and close() sequence
- No direct damage, but it clogged network links very quickly

Worm Effects

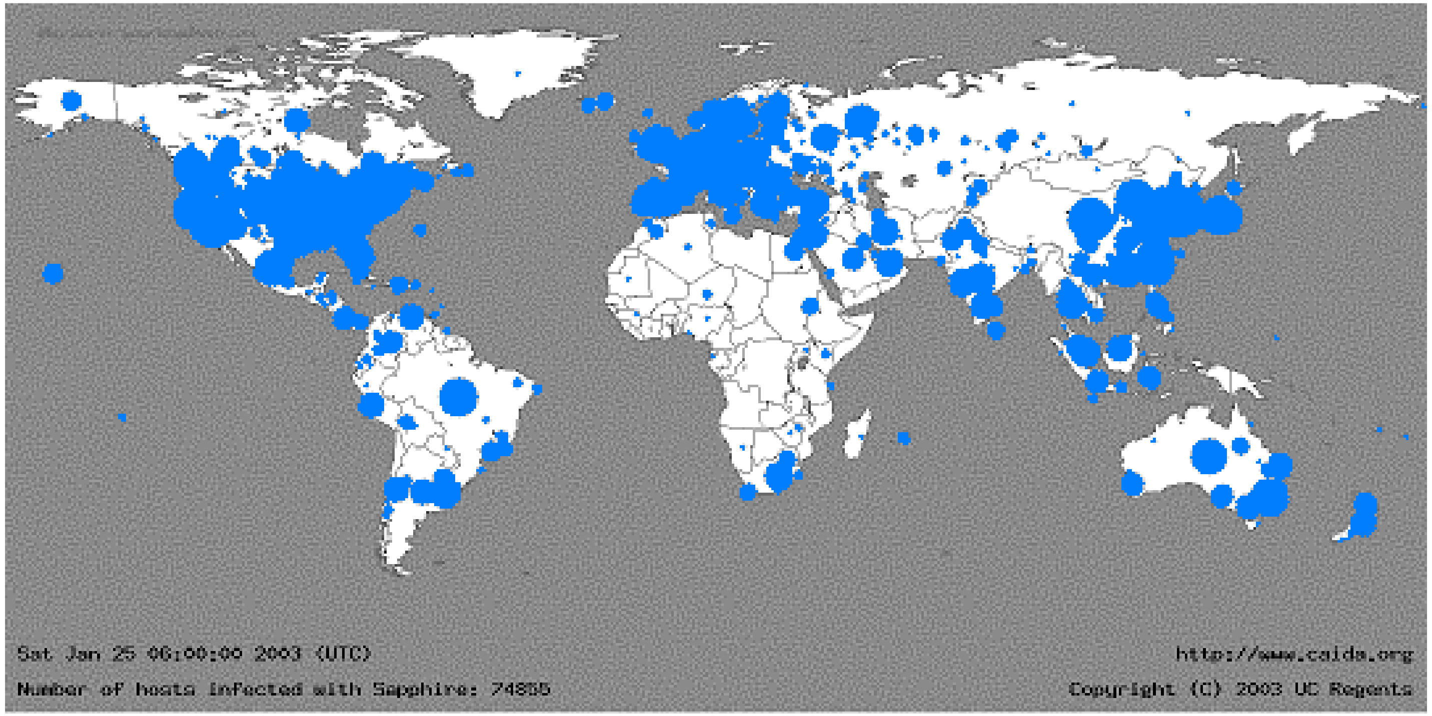
- Seriously clogged networks
- Slammer affected some ATM and air traffic control networks
- CSX Railroad's signaling network was affected

Problem

- how to react quickly to worms?
- CodeRed 2001
  - Infected ~360,000 hosts within 11 hours
- Sapphire/Slammer (376 bytes) 2002
  - Infected ~75,000 hosts within 10 minutes

The SQL Slammer Worm: 30 Minutes After “Release”

- Infections doubled every 8.5 seconds
- Spread 100X faster than Code Red
- At peak, scanned 55 million hosts per second.



Network Effects Of The SQL Slammer Worm

- At the height of infections
  - Several ISPs noted significant bandwidth consumption at peering points
  - Average packet loss approached 20%
  - South Korea lost almost all Internet service for period of time
  - Financial ATMs were affected
  - Some airline ticketing systems overwhelmed

Current Detection Methods

- Typically an IDS helps the administrators
- Isolation of the worm
- Security experts create the worms signature
- Updates to antivirus and network filtering software
- Correct but expensive, slow and manual procedure.
- Reaction time should be max 60 sec to contain a worm

Background

- CodeRed in 2001
  - Repair rate: 2% per day - With media attention
  - Automatic Intervention is necessary
- Signature-based models can halt all matching network activity, when the worm's signature is created

Worm Detection

- Three classes of methods
  - Scan detection
  - Honeypots
  - Behavioral techniques

Scan Detection

- Look for unusual frequency and distribution of address scanning
  - Here is where a telescope would be useful
- Limitations
  - Not suited to worms that spread in a non-random fashion (i.e. emails, IM, P2P apps)
    - Based on a target list
    - Spread topologically

Scan Detection

- More limitations
  - Detects infected sites
  - Does not produce a signature

Honeypots

- Monitored idle hosts with untreated vulnerabilities
  - Used to isolate worms
- Limitations
  - Manual extraction of signatures
  - Depend on quick infections

Behavioral Detection

- Looks for unusual system call patterns
  - Sending a packet from the same buffer containing a received packet
  - Can detect slow moving worms
- Limitations
  - Needs application-specific knowledge
  - Cannot infer a large-scale outbreak

Characterization

- Process of analyzing and identifying a new worm
- Current approaches
  - Use a priori vulnerability signatures
  - Automated signature extraction

Next Step: Vulnerability Signatures

- Example
  - Slammer Worm
    - UDP traffic on port 1434 that is longer than 100 bytes (buffer overflow)
- Can be deployed before the outbreak
  - Can only be applied to well-known vulnerabilities

Some Automated Signature Extraction Techniques

- Allows worm to infect decoy programs
  - Extracts the modified regions of the decoy
  - Uses heuristics to identify invariant code strings across infected instances

Some Automated Signature Extraction Techniques

- Limitation
    - Assumes the presence of a worm in a controlled environment

Next Step: Containment

- Mechanism used to deter the spread of an active worm
  - Host quarantine
    - Via IP ACLs on routers or firewalls
- String-matching
- Connection throttling
  - On all outgoing connections

Automated Worm Fingerprinting, Sumeet Singh, Cristian Estan, George Varghese and Stefan Savage, Proceedings of the ACM/USENIX Symposium on Operating System Design and Implementation, San Francisco, CA, December 2004.

Earlybird

- Automatic detection and containment of new worms
- Content Sifting:
  - Content of worm traffic is invariant
  - Worm spread dynamics atypical of Internet Applications
- Frequently repeated and widely dispersed content strings -> new worm

Defining Worm Behavior

- Content invariance
  - Portions of a worm are invariant (e.g. the decryption routine)
- Content prevalence
  - Appears frequently on the network
- Address dispersion
  - Distribution of destination addresses more uniform to spread fast

Finding Worm Signatures

- Traffic pattern is sufficient for detecting worms
  - Relatively straightforward
  - Extract all possible substrings
  - Raise an alarm when
    - FrequencyCounter[substring] > threshold1
    - SourceCounter[substring] > threshold2
    - DestCounter[substring] > threshold3

Practical Content Sifting

- Characteristics
  - Small processing requirements
  - Small memory requirements
  - Allows arbitrary deployment strategies

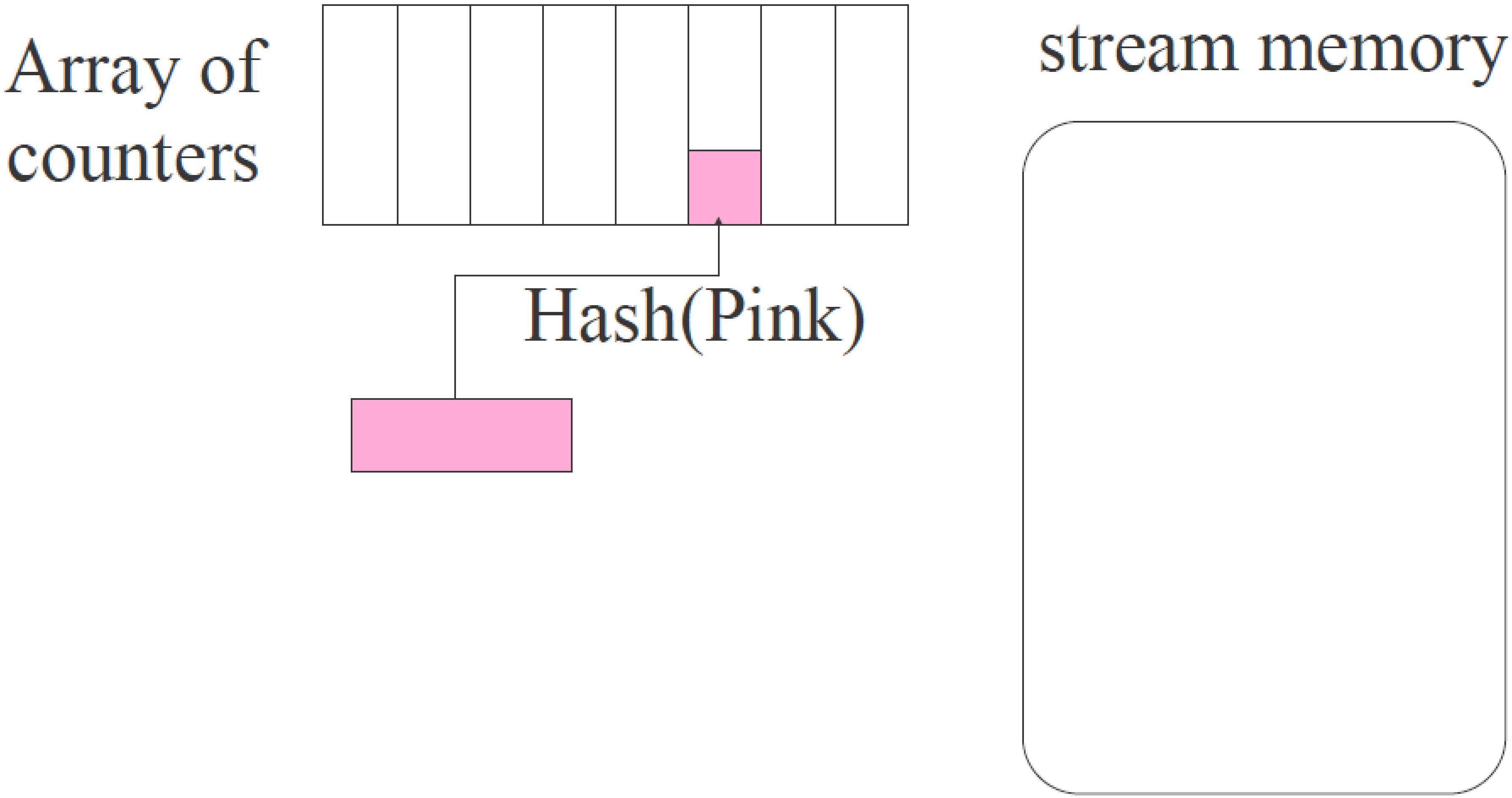
Estimating Content Prevalence

- Finding the packet payloads that appear at least x times among the N packets sent
  - During a given interval

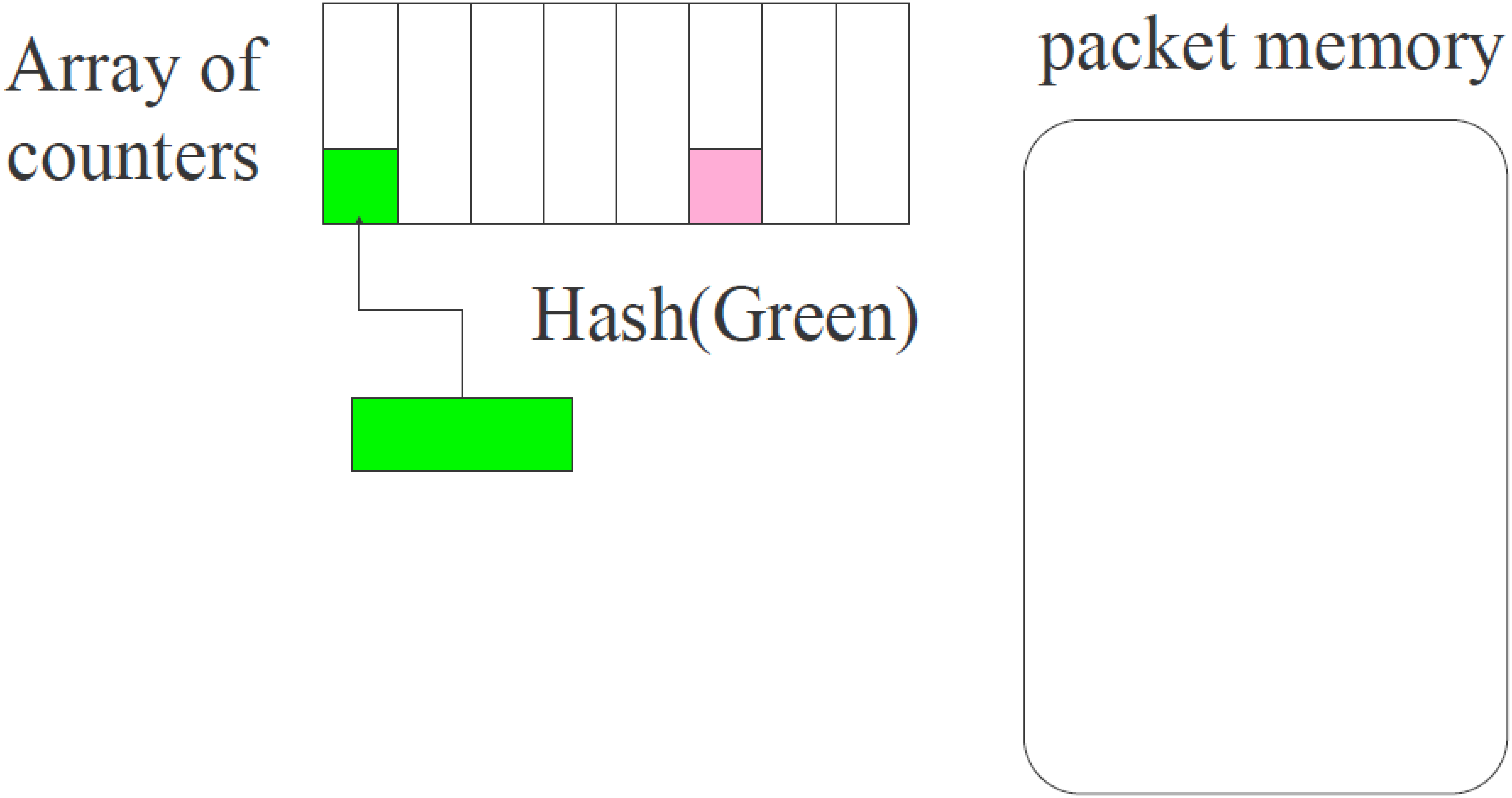
Estimating Content Prevalence

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

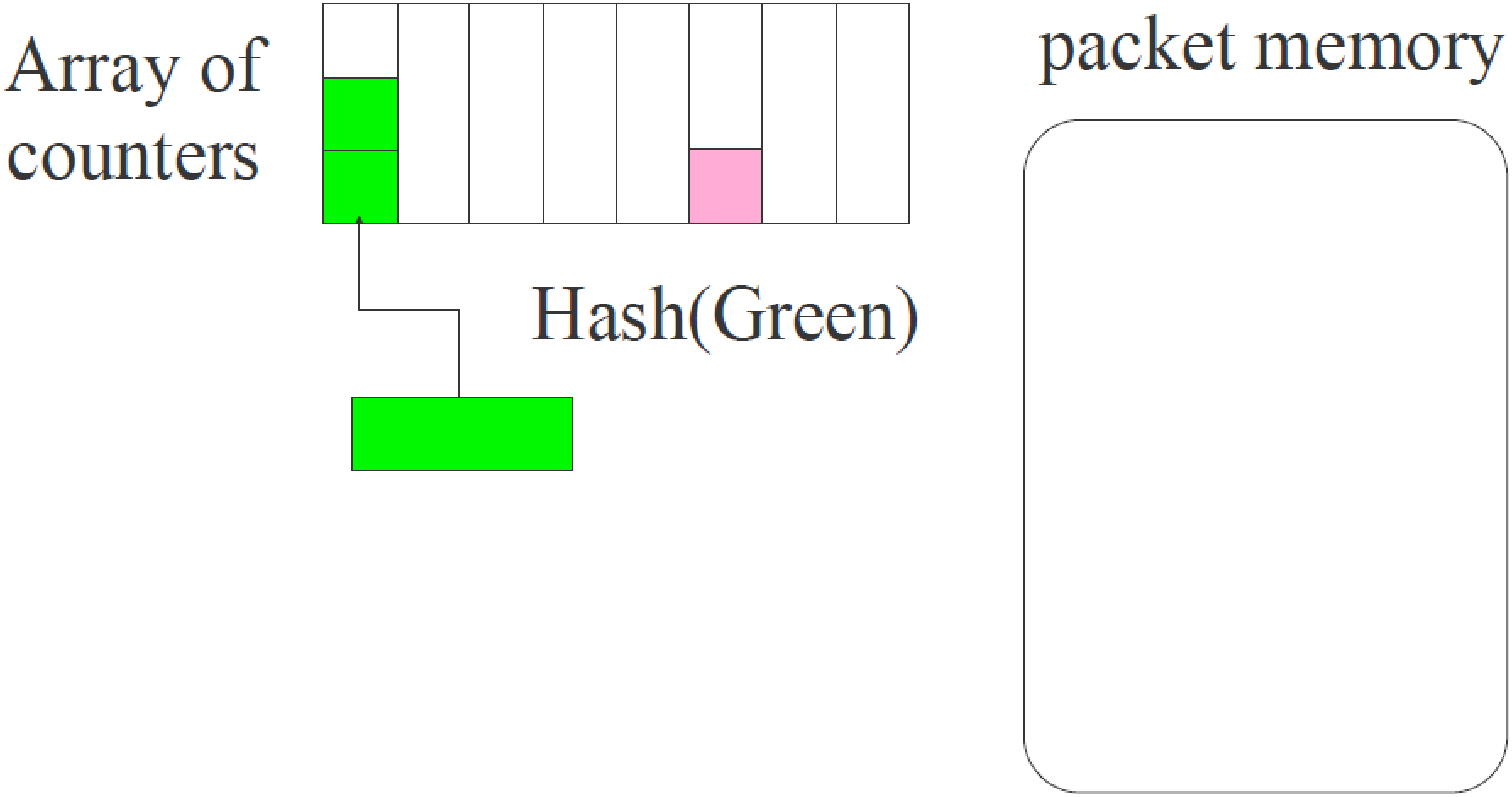
Multistage Filters



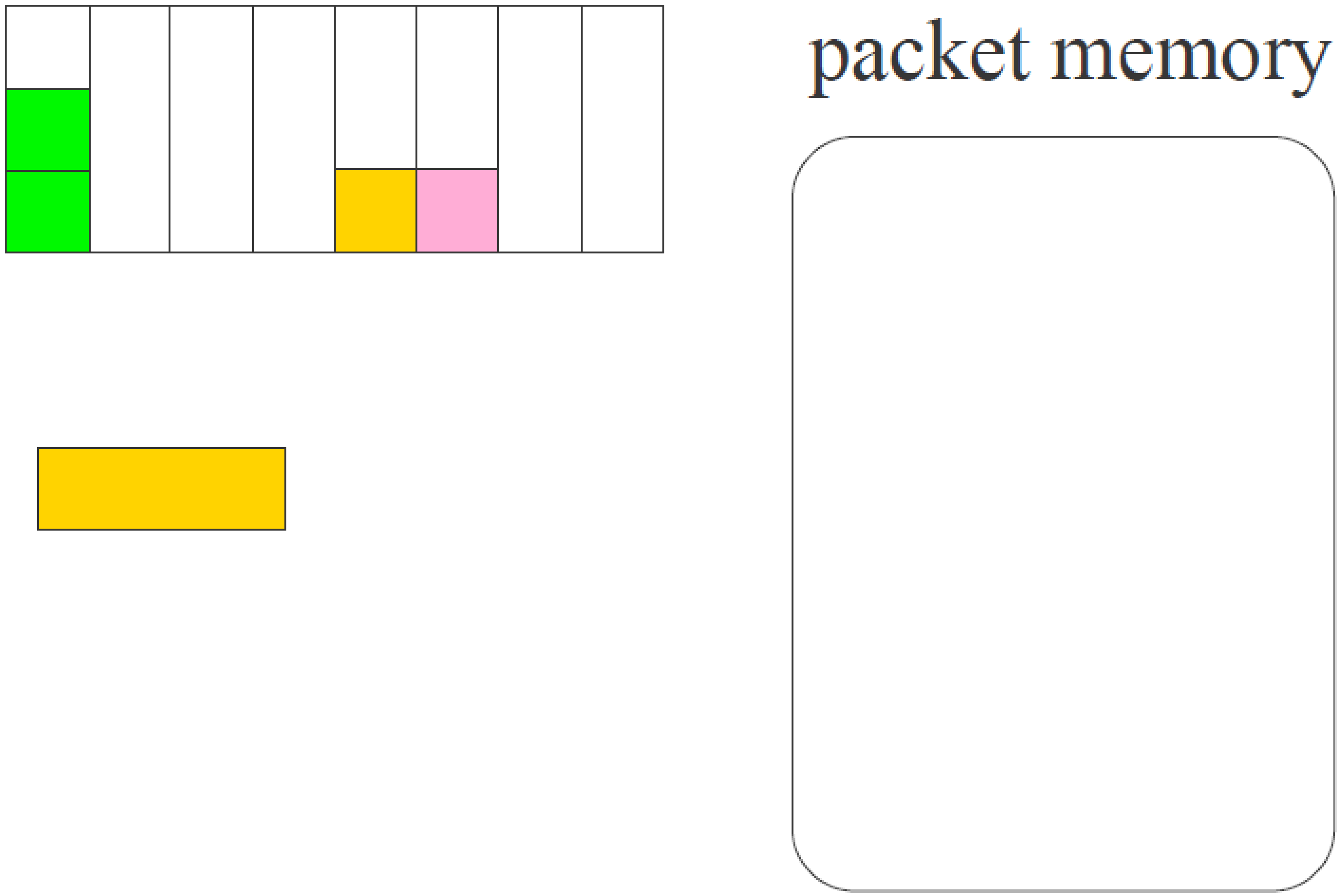
Multistage Filters



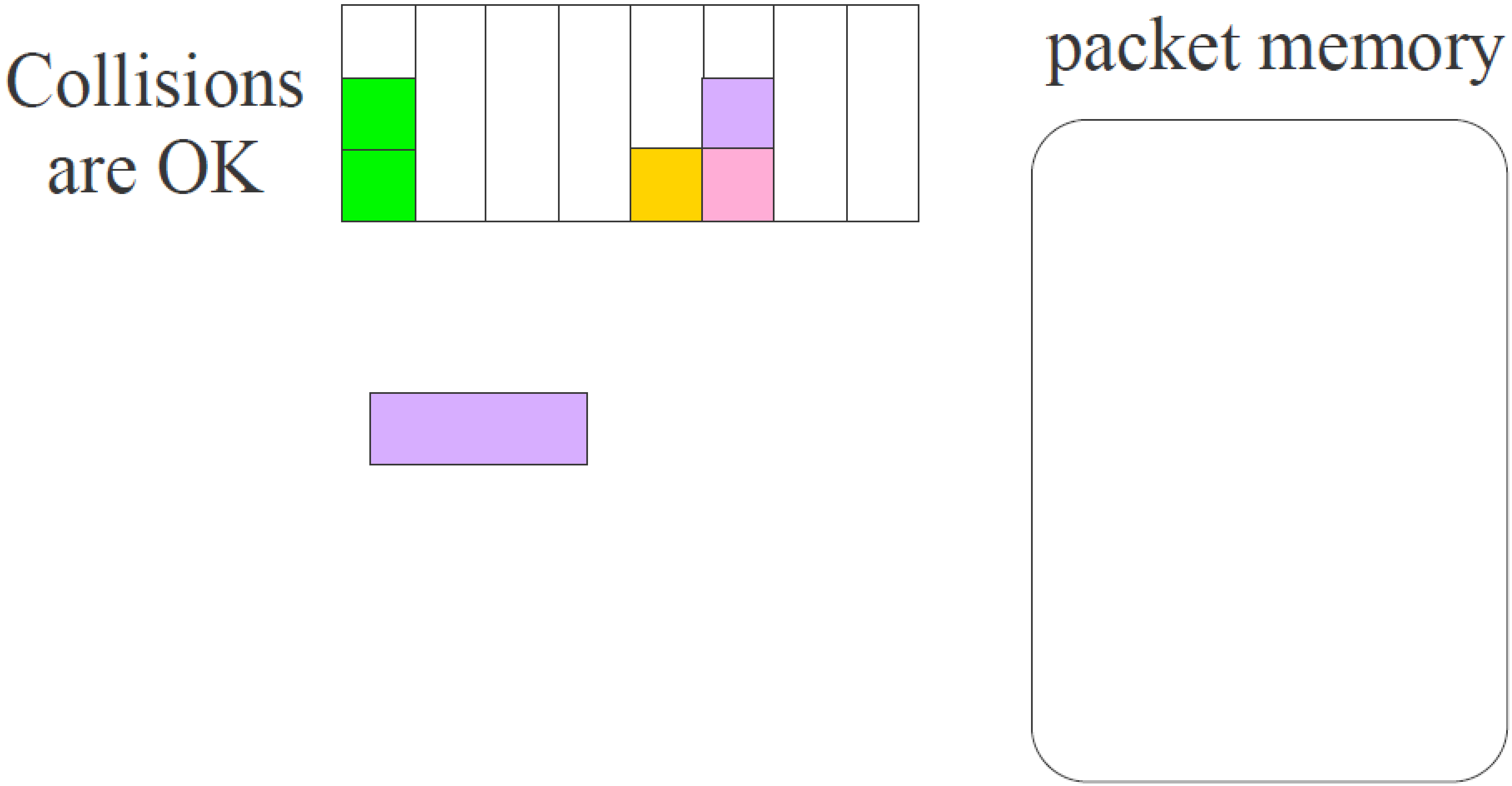
Multistage Filters



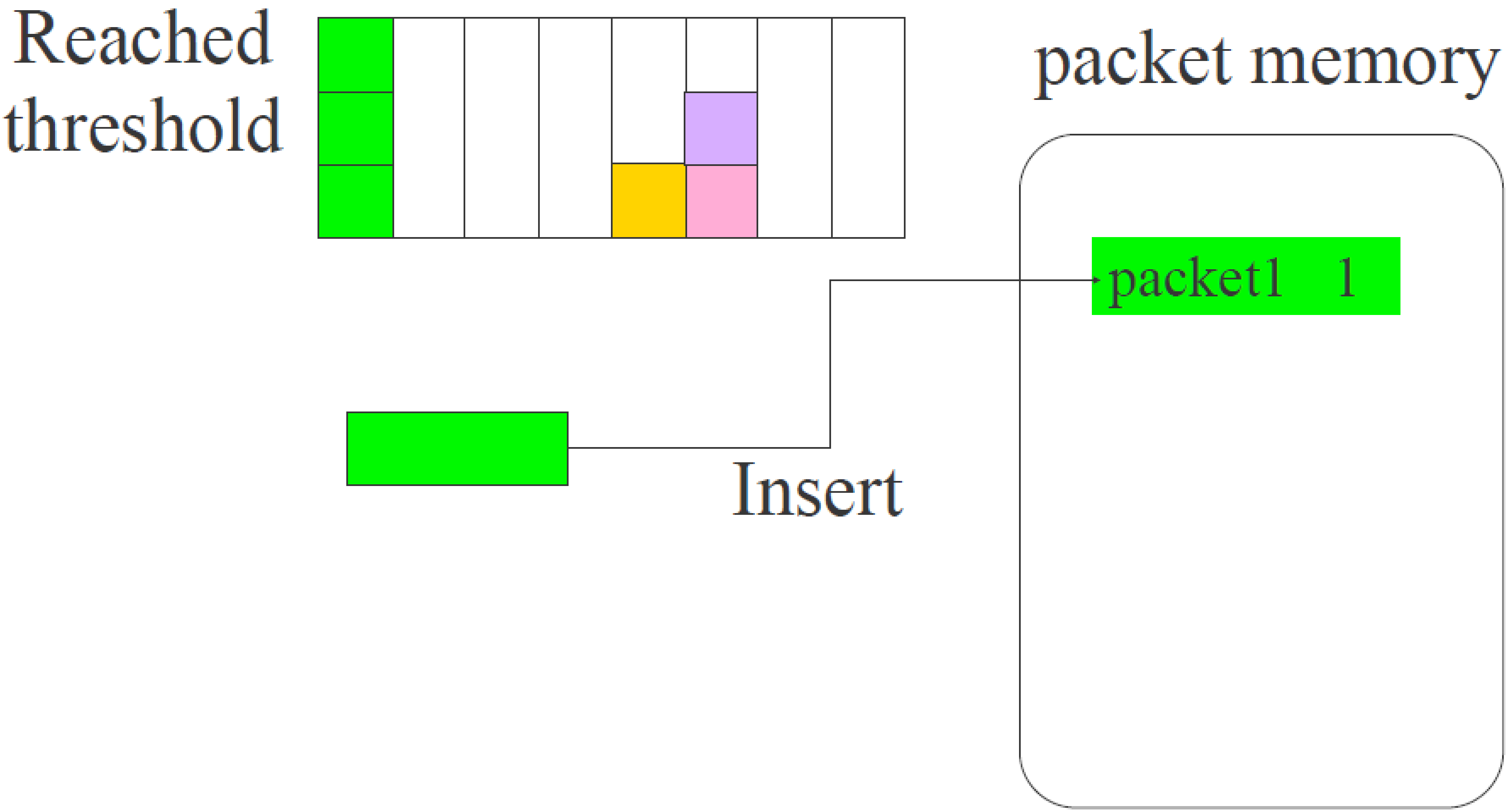
Multistage Filters



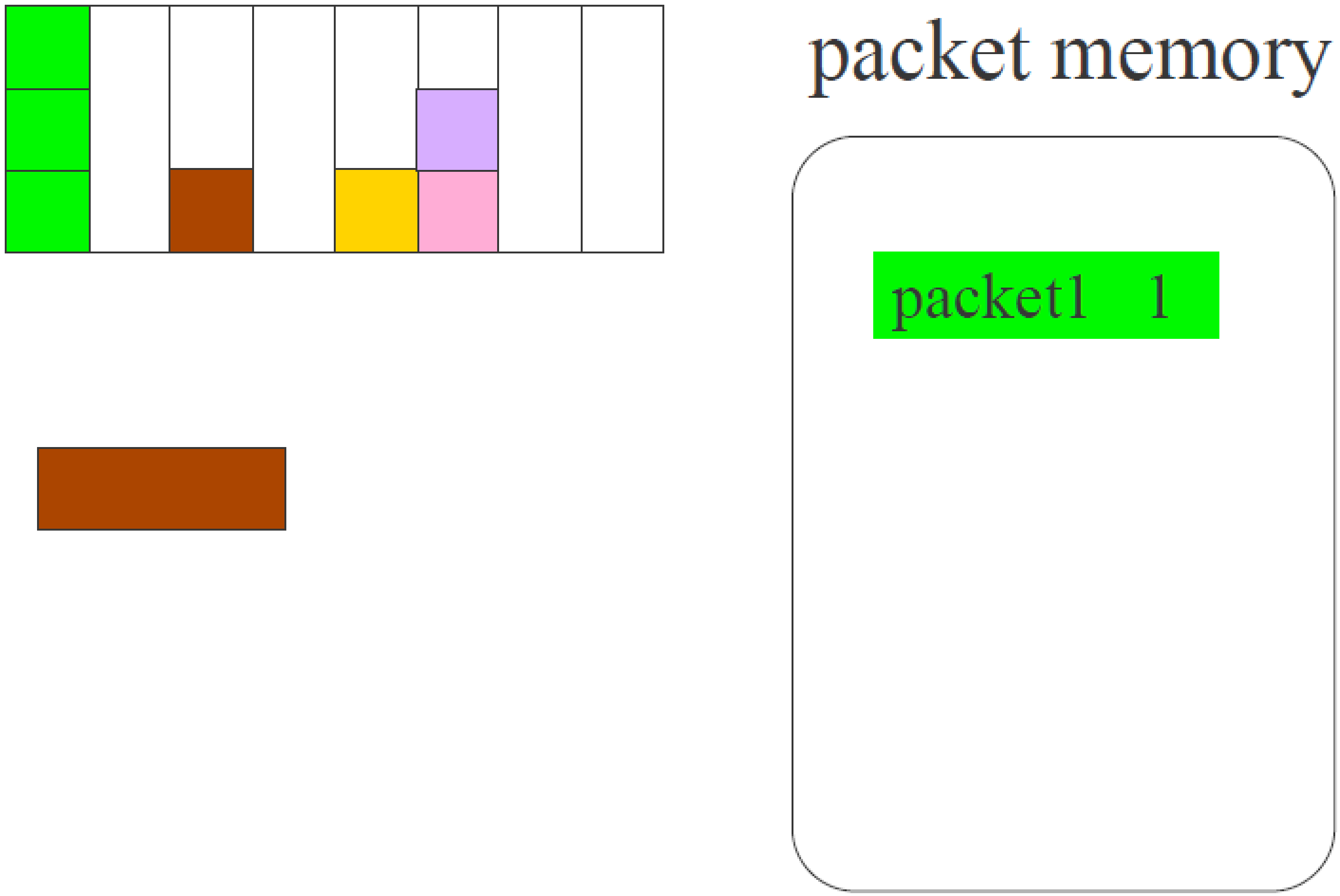
Multistage Filters



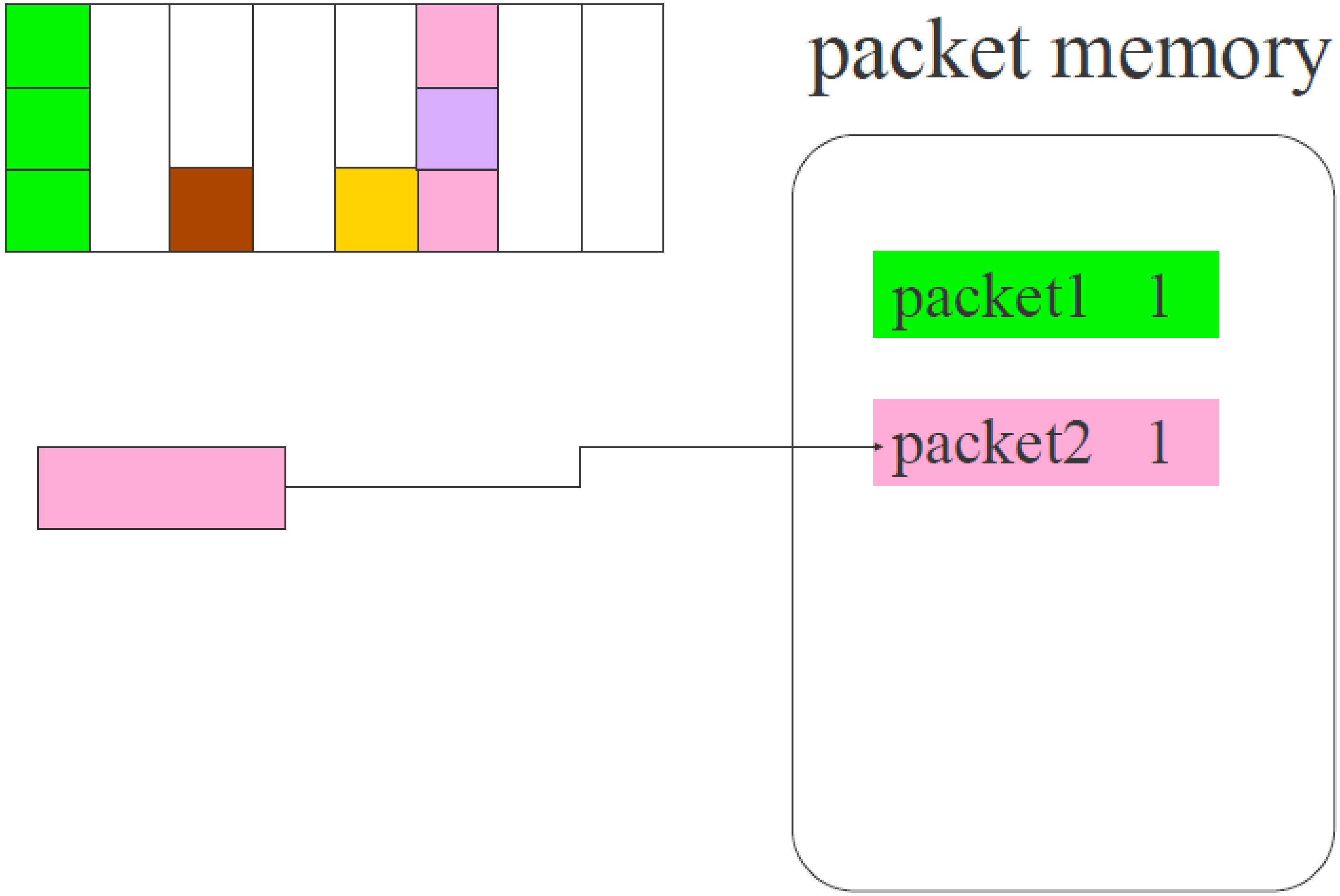
Multistage Filters



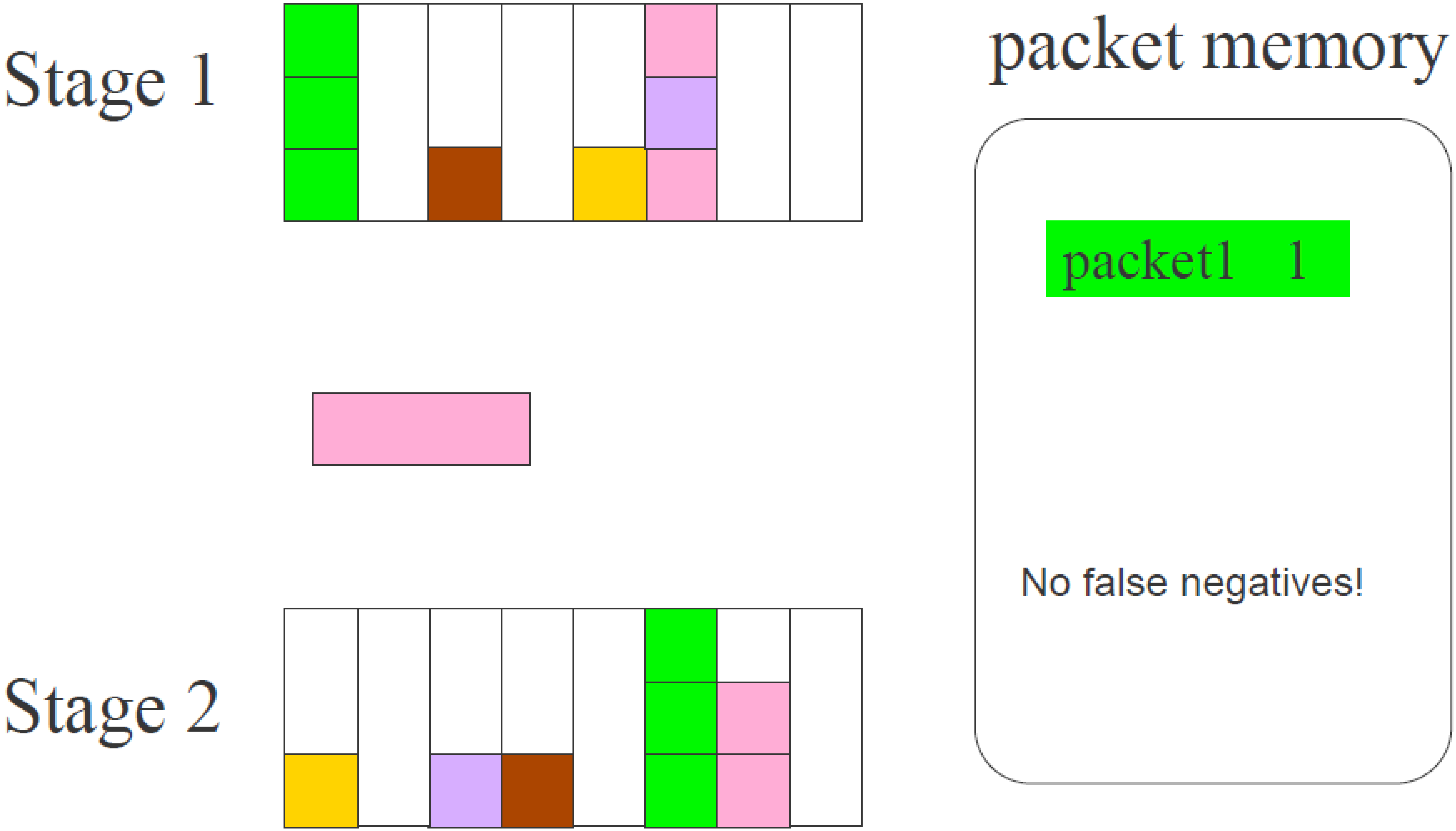
Multistage Filters



Multistage Filters



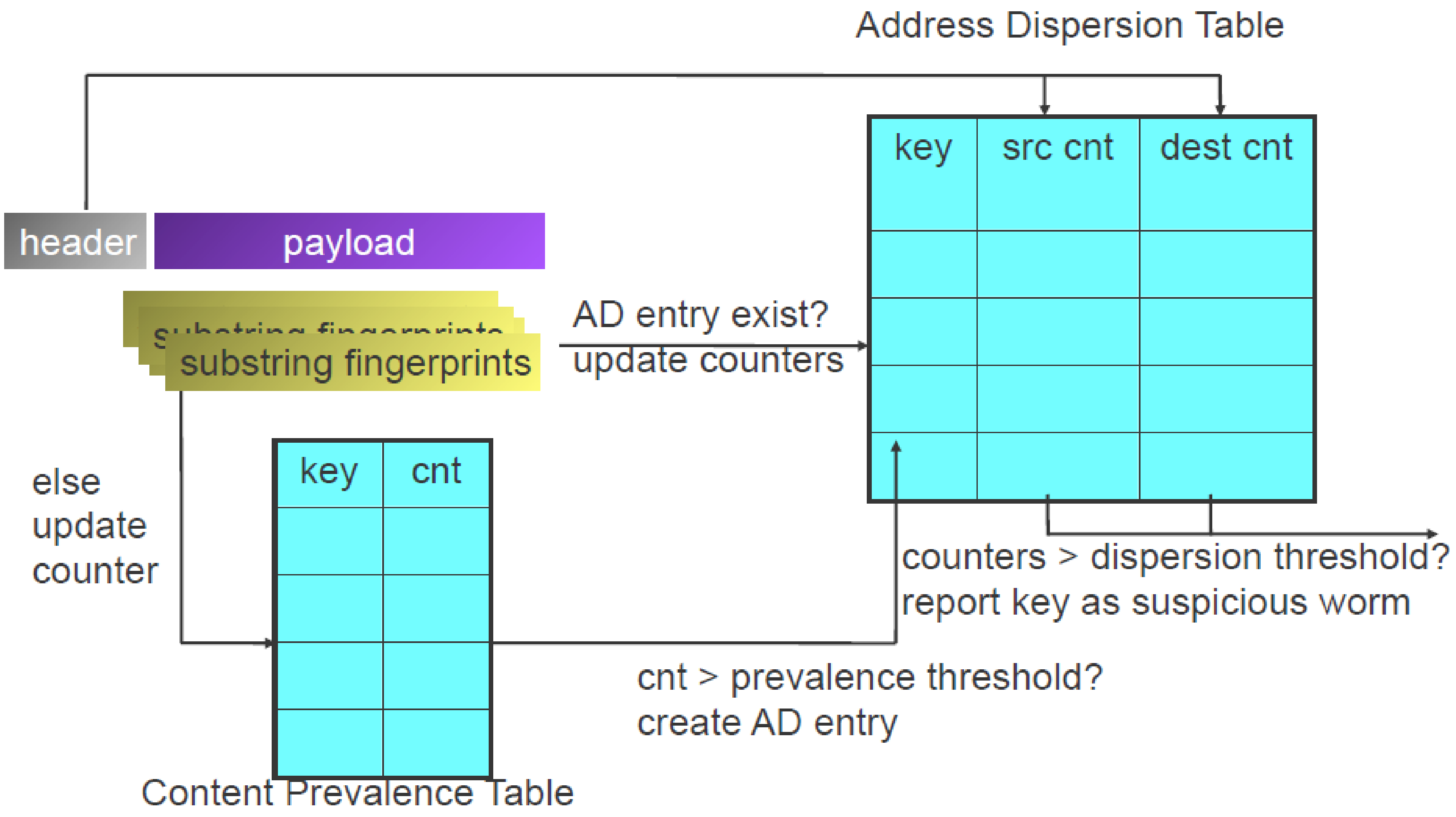
Multistage Filters



Estimating Address Dispersion

- Not sufficient to count the number of source and destination pairs
  - e.g. send a mail to a mailing list
    - Two sources—mail server and the sender
    - Many destinations
- Need to track the distinct source and destination IP addresses
  - For each substring
- Simple list or hash table is too expensive
  - Use Bitmap data structure

Putting It Together



System Design

- Two major components
  - Sensors
    - Sift through traffic for a given address space
    - Report signatures
  - An aggregator
    - Coordinates real-time updates
    - Distributes signatures

Implementation and Environment

- Implementation
  - Written in C and MySQL (5,000 lines)
  - rrd-tools library for graphical reporting
  - PHP scripting for administrative control
  - Prototype executes on a 1.6Ghz AMD Opteron 242 1U Server
    - Linux 2.6 kernel
- Processes 1TB of traffic per day
- Can keep up with 200Mbps of continuous traffic

Content prevalence threshold

- Using a 60 second measurement interval and a whole packet CRC
  - over 97 percent of all signatures repeat two or fewer times and 94.5 percent are only observed once
- Using a finer grained content hash or a longer measurement interval increases these numbers even further
- Default: 3 repetitions

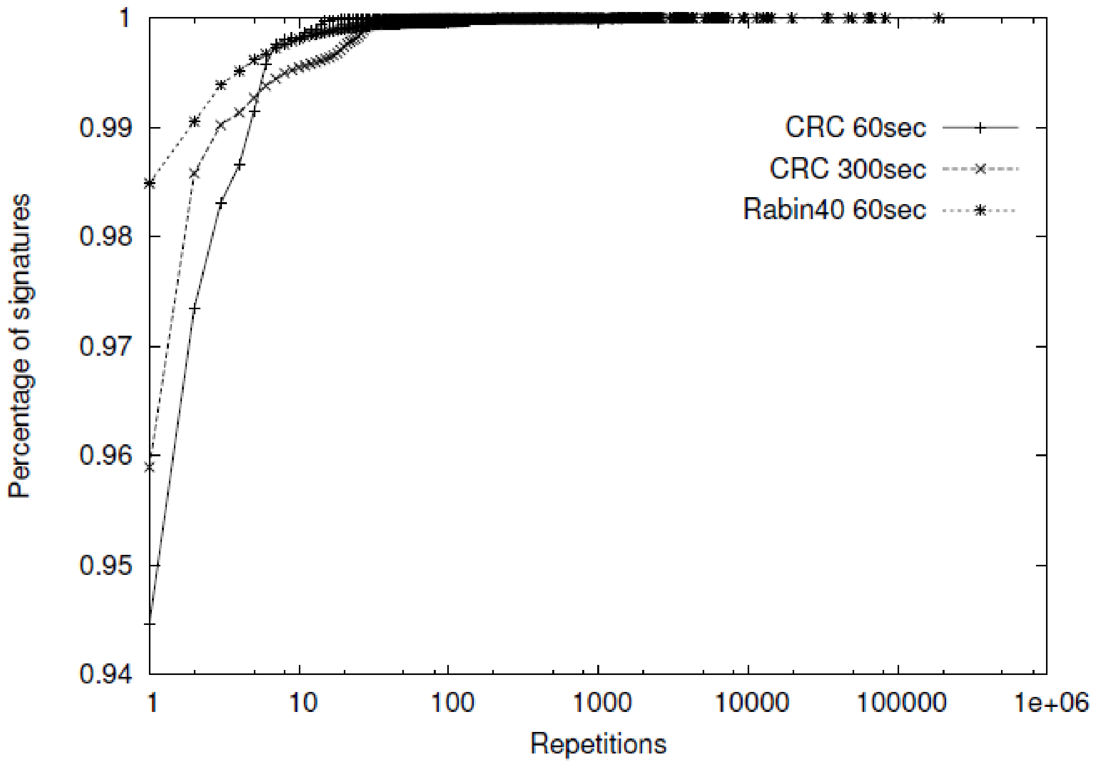


Figure 8: Cumulative distribution function of content signatures for different hash functions. This CDF is computed from the set of repetitions found in each measurement interval over a period of 10 minutes. Note that the y axis is artificially truncated to show more detail.

Address dispersion threshold

- After 10 minutes there are over 1000 signatures with a low dispersion threshold of 2
- Using a threshold of 30, there are only 5 or 6 prevalent strings meeting the dispersion criteria
- Default: 30 sources and 30 destinations

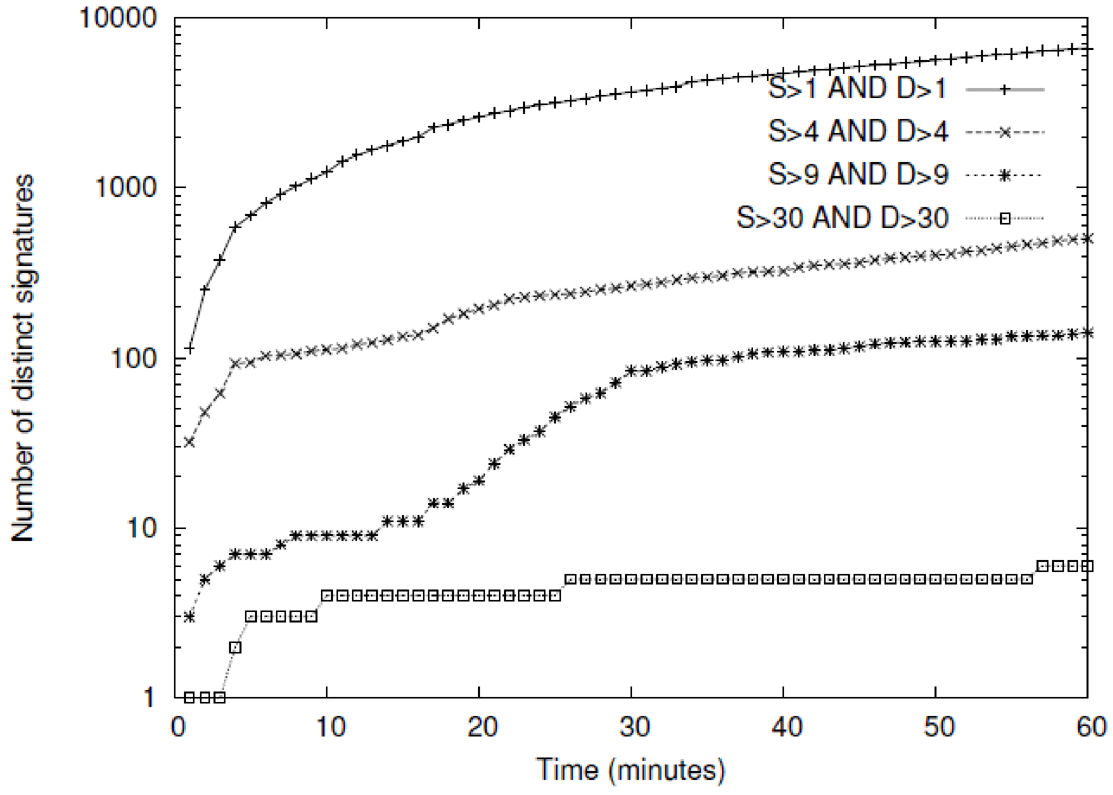


Figure 9: Number of distinct signatures detected over time for different address dispersion thresholds.

Garbage Collection

- When the timeout is set to 100 seconds, then almost 60 percent of all signatures are garbage collected before a subsequent update
- Using a timeout of 1000 seconds, this number is reduced to roughly 20 percent of signatures
- Default: several hours

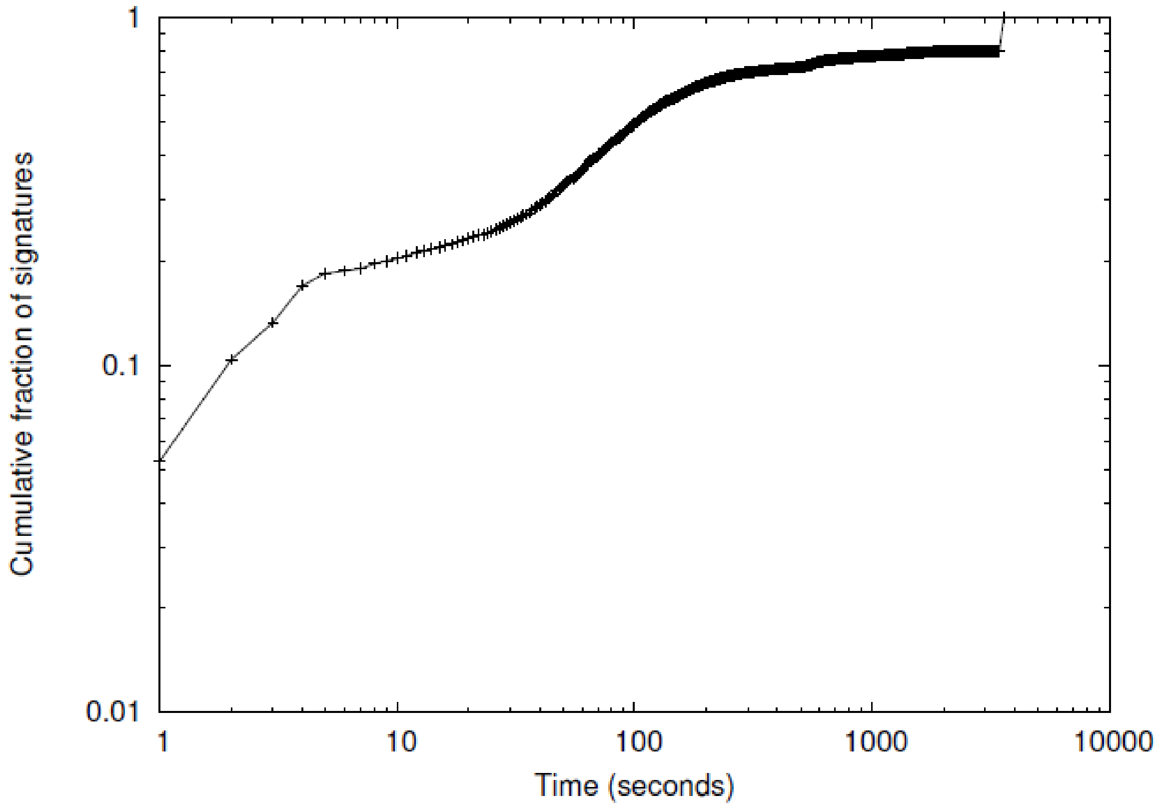


Figure 10: Log-scale cumulative distribution function of the maximum time period between updates for entries in the address dispersion table.

Performance Processing Time

	Mean	Std. Dev.
Component wise breakdown		
Rabin Fingerprint		
First Fingerprint (40 bytes)	0.349	0.472
Increment (each byte)	0.037	0.004
Multi Stage Filter		
Test & Increment	0.146	0.049
AD Table Entry		
Lookup	0.021	0.032
Update	0.027	0.013
Create	0.252	0.306
Insert	0.113	0.075
Overall Packet		
Header Parsing & First Fingerprint	0.444	0.522
Per-byte processing	0.409	0.148
Overall Packet with Flow-Reassembly		
Header Parsing & Flow maintenance	0.671	0.923
Per-byte processing	0.451	0.186

Table 1: This table shows overhead (in microseconds) incurred by each of the individual operations performed on a packet. The mean and standard deviation are computed over a 10 minute interval (25 million Packets). This table represents raw overheads before sampling. Using 1 in 64 value sampling, the effective mean per byte processing time reduces to 0.042 microseconds.

Performance Memory Consumption

- Prevalence table
  - Totals to 2 MB
- Address Dispersion Table
  - under 1MB
- Total less than 4MB

Trace-Based Verification

- Two main sources of false positives
  - 2,000 common protocol headers
    - e.g. HTTP, SMTP
    - Whitelisted
  - SPAM e-mails
  - BitTorrent
    - Many-to-many download

False Negatives

- So far none
- Detected every worm outbreak

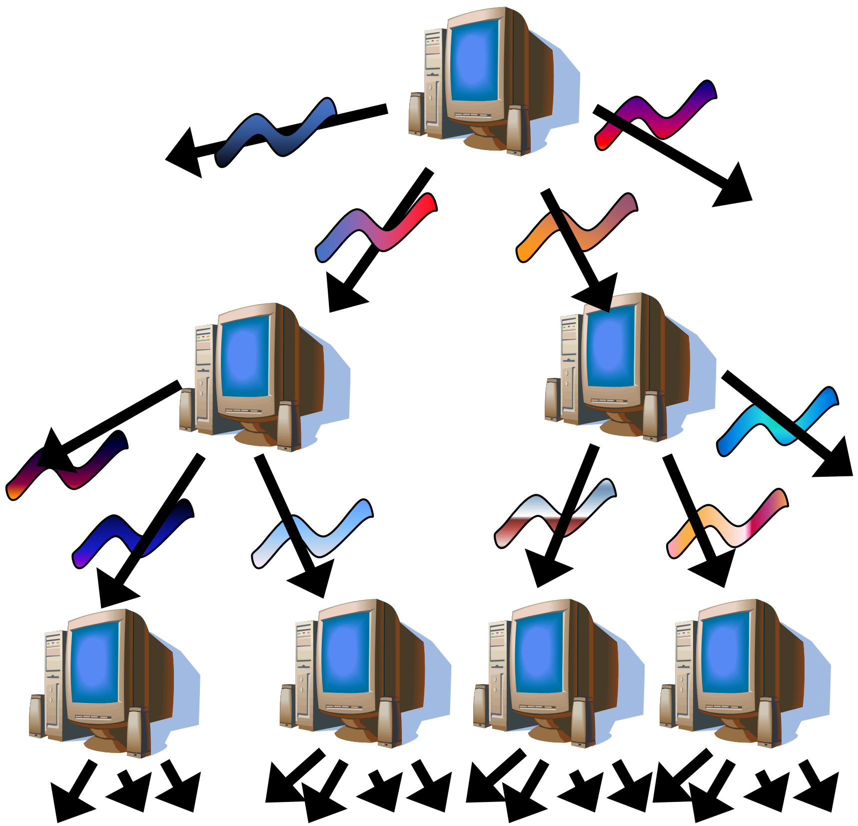
Evasions

- An attacker might evade detection by splitting an invariant string across packets
  - Have fingerprints across packets
- Traffic normalization
  - remember attacks on IDS
- Polymorphic viruses
  - Semantically equivalent but textually distinct code
  - Invariant decoding routine

POLYGRAPH: Automatically Generating Signatures for Polymorphic Worms, James Newsome, Brad Karp, Dawn Song, IEEE Security and Privacy Symposium, May 2005.

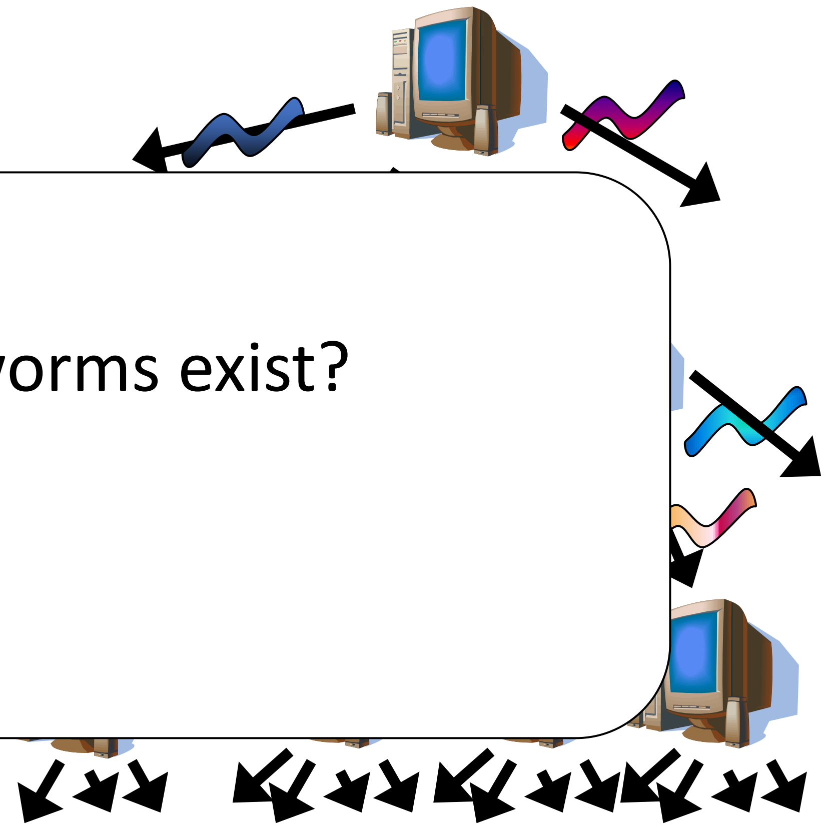
Challenge: Polymorphic Worms

- Polymorphic worms minimize invariant content
  - Encrypted payload
  - Obfuscated decryption routine
- Polymorphic tools are already available
  - Clet, ADMmutate



Challenge: Polymorphic Worms

- Polymorphic worms
  minimize invariant content
- Po
  alr



Do good signatures for polymorphic worms exist?

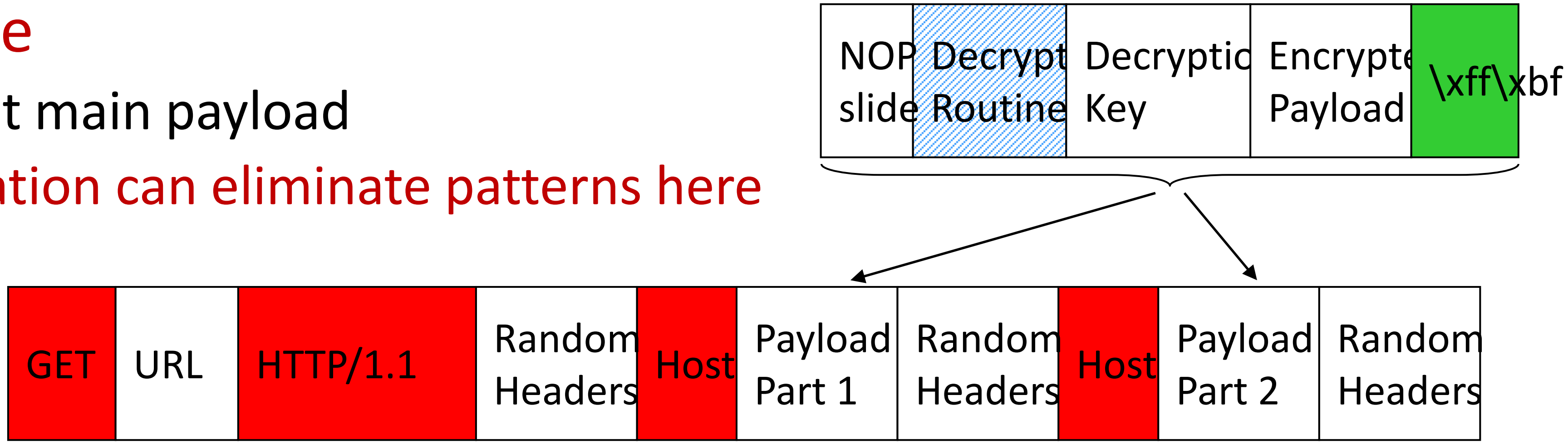
Can we generate them *automatically*?

Goals

- Identify classes of signatures that can
  - Accurately describe polymorphic worms
  - Be used to filter a high speed network line
  - Be generated automatically and efficiently

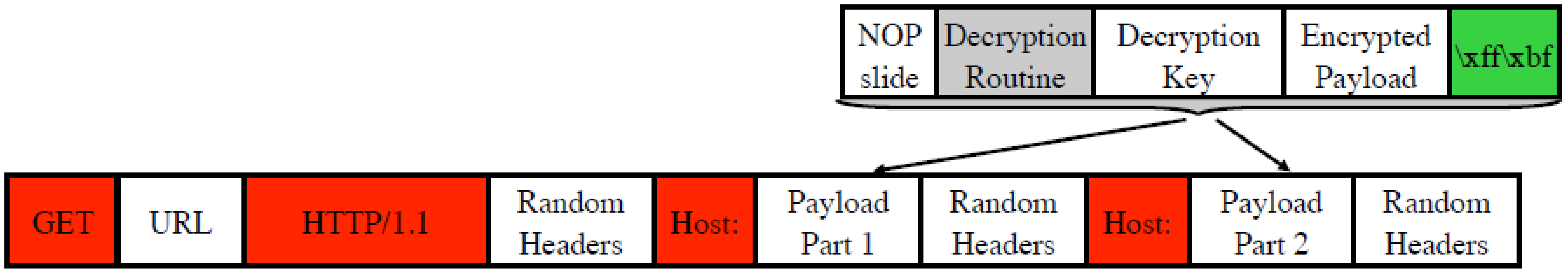
Good News: Still some invariant content

- Protocol framing
  - Needed to make server go down vulnerable code path
- Overwritten Return Address
  - Needed to redirect execution to worm code
- Decryption routine
  - Needed to decrypt main payload
  - BUT, code obfuscation can eliminate patterns here



Bad News: Previous Approaches Insufficient

- Previous approaches use a common substring
- Longest substring
  - “HTTP/1.1”
  - 93% false positive rate
- Most specific substring
  - “\xff\xbf”
  - .008% false positive rate (10 / 125,301)

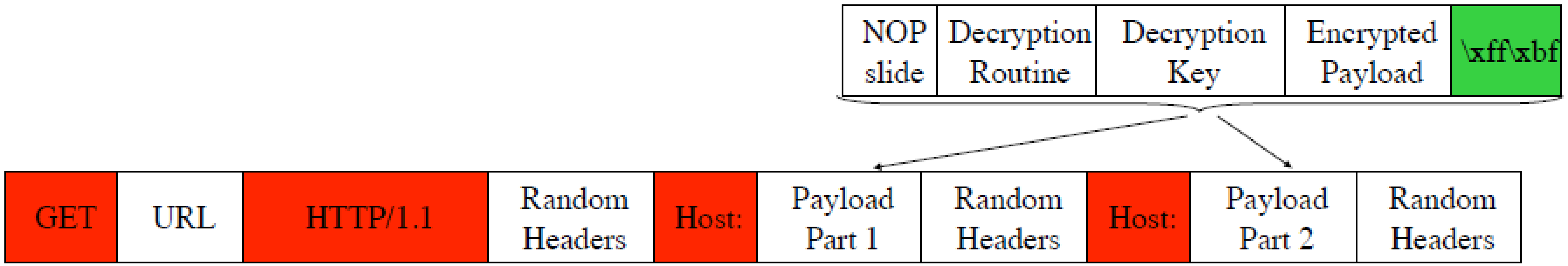


What to do?

- No one substring is specific enough
- BUT, there are multiple substrings
  - Protocol framing
  - Value used to overwrite return address
  - (Parts of poorly obfuscated code)
- Our approach: combine the substrings

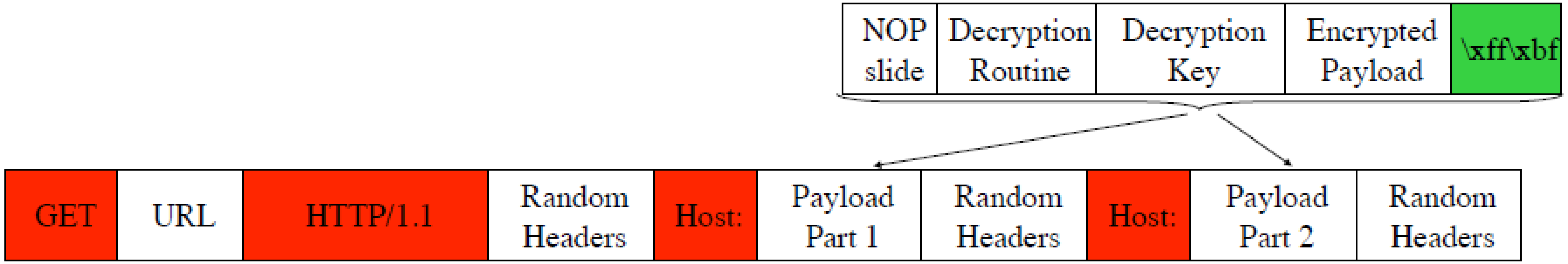
Signature Class (I): Conjunction

- Signature is a set of strings (tokens)
- Flow matches signature iff it contains all tokens in the signature
- O(n) time to match (n is flow length)
- Generated signature:
  - "GET" and "HTTP/1.1" and "\r\nHost:" and "\r\nHost:" and "\xff\xbf"
  - .0024% false positive rate (3 / 125,301)



Signature Class (II): Token Subsequence

- Signature is an ordered set of tokens
- Flow matches iff it contains all the tokens in signature, in the given order
- O(n) time to match (n is flow length)
- Generated signature:
  - GET.*HTTP/1.1.*\r\nHost:.*\r\nHost:.*\xff\xbf
  - .0008% false positive rate (1 / 125,301)



Experiment: Signature Generation

- How many worm samples do we need?
  - Too few samples --> signature is too specific --> false negatives
- Experimental setup
  - Using a 15 day port 80 trace from lab perimeter
  - Innocuous pool: First 5 days (45,111 streams)
  - Suspicious Pool:
    - Using Apache exploit described in paper
    - Non-invariant portions filled with random bytes
- Signature evaluation:
  - False positives: Last 10 days (125,301 streams)
  - False negatives: 1000 generated worm samples

Signature Generation Results

# Worm Samples	Conjunction	Subseq
2	100% FN	100% FN
3 to 100	0% FN .0024% FP	0% FN .0008% FP

GET .* HTTP/1.1\r\n.*\r\nHost: .*\xee\xb7.*\xb2\x1e.*\r\nHost: .*\xef\xa3.*\x8b\xf4.*\x89\x8b.*E\xeb.*\xff\xbf

GET .* HTTP/1.1\r\n.*\r\nHost: .*\r\nHost:.*\xff\xbf

Acknowledgments/References

- [Singh04] Automated Worm Fingerprinting, Sumeet Singh, Cristian Estan, George Varghese and Stefan Savage, Proceedings of the ACM/USENIX Symposium on Operating System Design and Implementation, San Francisco, CA, December 2004.
- [Wang05] FSU COP 5611 (Spring 2005) Advanced Operating Systems by Andy Wang.
- [kuzma] ww.cs.northwestern.edu/~akuzma/classes/CS495-s05/doc/awf.ppt
- [hy558] ww.csd.uoc.gr/~hy558/reports/itsomp_fingerprinting.ppt
- [zou07] Research in Computer and Network Security, CDA6938, Cliff Zou, 2007
- [Newsome05] POLYGRAPH: Automatically Generating Signatures for Polymorphic Worms, James Newsome, Brad Karp, Dawn Song, presentation at IEEE Security and Privacy Symposium, May 2005.