CS 552
Computer Security

Richard Martin

Aspects of Security

- **Confidentiality**: can a 3rd party see it?
- **Authentication**: Who am I talking to?
- **Non-repudiation**: can you claim you didn't send it even if you really did?
- **Integrity**: was it altered before I got it?
- **Authorization**: Are you allowed to perform the action (method)?
- **Auditing**: what happened, when, by who?

Outline

- Basics:
  - Stack smashing
  - worms, viruses
- Papers:
  - Basic Security
  - Portscan detection
  - Denial of Service traceback

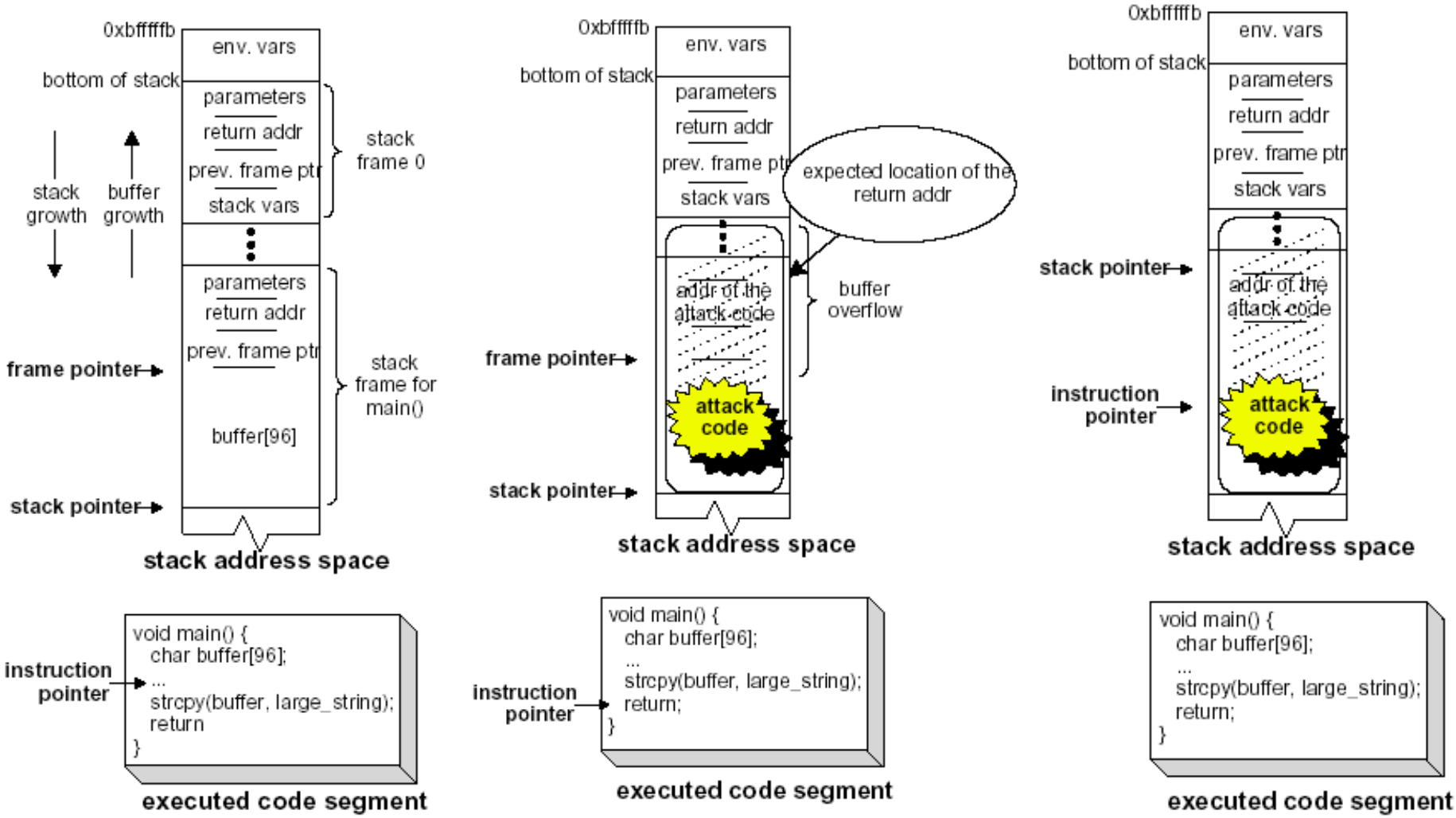
Stack Vulnerability

- Any program written without memory protection (i.e., pointers) + writable stacks
- Recall from architecture/compilers:
  - local variables in a function/procedure/method are kept on a stack
  - So is the return address of the program counter.
- “Stack smashing” Give bad data to a program that:
  - Writes executable code into the program somewhere (most often the stack)
  - corrupts return address on stack to jump to code

Stack Smashing Code Example

```
#include <stdio.h>
char shellcode[] =
   "\xeb\x1f\x5e\x89\x76\x08\x31\xc0\x88\x46\x07\x89\x46\x0c\x
   b0\x0b\x89\xf3\x8d\x4e\x08\x8d\x56\x0c\xcd\x80\x31\xdb\x89\
   xd8\x40\xcd\x80\xe8\xdc\xff\xff\xff/bin/sh";
char large string[128];
int i;
long *long_ptr;
int main() {
     char buffer[96];
     long ptr = (long *)large_string;
     for (i=0; i<32; i++)
        (long_ptr+i) = (int)buffer;
     for (i=0; i<(int)strlen(shellcode); i++)
        large_string[i] = shellcode[i];
strcpy(buffer, large string);
return 0; }
```

Stack Smashing Attack



Vulnerable code example

```
int doSomething(int variable1);
   int arg1, arg2;
   char nextCommand[MAX_COMMAND];
   char inputFromWorld[MAX_INPUT];
...
   newsock_fd = accept(sockfd);
...
   read(newsock_fd,inputFromWorld,MAX_INPUT];
   sscanf(inputFromWorld,"%d %s %d ",&arg1,
  nextCommand, &arg2);
```

Overview of Protocol Security

- TCP syn impersonation
- Routing attacks
  - Source routing
  - Protocols:
    - RIP/EGP
  - ICMP
- Services and Server programs
  - Finger
  - Email (SMTP, POP)
  - Name lookup (DNS)
  - File transfer (FTP)
  - Network management (SNMP)
  - Remote Boot (TFTP, BTOOP)

TCP syn impersionation

- Assume attacker has compromised a machine, X
- Goal:
  - X can impersonate machine T to server S
    - Even if X has not compromised T
    - Even if X can't observe traffic of S->T
- Strategy:
  - Send S a fake message
  - Guess Sequence number of S->T
  - Send fake data, get S to believe packet is from S

TCP Syn attack

Normal Syn/Ack sequence

C->S:SYN(ISN_c)

S->C:SYN(ISNs),ACK(ISNc)

C->S:ACK(ISNs)

C->S:data

Attacker based

X->S:SYN(ISNx),SRC=T

S->T:SYN(ISNs),ACK(ISNx)

X->S:ACK(ISNs),SRC=T

X->S:ACK(ISNs),SRC=T, bad_data

TCP syn attacks

- Success of attack depends on 2 weaknesses:
  - Initial guess of sequence number ISNs
  - S doing something with the bad data
    - Higher level of authentication

Routing Attacks

- Insert bogus routing entries for
  - Entire network
  - Single host
- Re-route packets for target system
  - Allows packet-interposition

Practical Security

- What does it mean to be secure?
  - Definitions from slide

Retrospective 1989-2005

- Authentication at all levels
  - “end-to-end” principal applies?
- DDoS
  - Never considered
- Auditing importance
- End-to-end confidentiality
  - Ssh, HTTPS
- Worry about bad input
  - Buffer overruns, SQL attacks
- Human impact on security
  - Too hard/complicated-> no security
  - Phishing
- Economic factors
  - Spam

Portscanning Intro

- Port Scanning: Reconnaissance
  - Hackers will scan host/hosts for vulnerable ports as potential avenues of attack
- Not clearly defined
  - Scan sweeps
    - Connection to a few addresses, some fail?
  - Granularity
    - Separate sources as one scan?
  - Temporal
    - Over what timeframe should activity be tracked
  - Intent
    - Hard to differentiate between benign scans and scans with malicious intent

Prior Detection Techniques

- Malformed Packets
  - Packets used for “stealth scanning”
- Connections to ports/hosts per unit time
  - Checks whether a source hits more than X ports on Y hosts in Z time
- Failed connections
  - Malicious connections will have a higher ratio of failed connection attempts

Bro NIDS

- Current algorithm in use for years
- High efficiency
- Counts local connections from remote host
- Differentiates connections by service
- Sets threshold
- Blocks suspected malicious hosts

Flaws in Bro

- Skewed for little-used servers
  - Example: a private host that one worker remotely logs into from home
- Difficult to choose probabilities
- Difficult to determine never-accessed hosts
  - Needs data to determine appropriate parameters

Threshold Random Walk (TRW)

- Objectives for the new algorithm:
  - Require performance near Bro
  - High speed
  - Flag as scanner if no useful connection
  - Detect single remote hosts

Data Analysis

- Data analyzed from two sites, LBL and ICSI
  - Research laboratories with minimal firewalling
  - LBL: 6000 hosts, sparse host density
  - ICSI: 200 hosts, dense host density

		LBL	ICSI
1	Total inbound connections	15,614,500	161,122
2	Size of local address space	131,836	512
3	Active hosts	5,906	217
4	Total unique remote hosts	190,928	29,528
5	Scanners detected by Bro	122	7
6	HTTP worms	37	69
7	`other_bad`	74,383	15
8	*remainder*	116,386	29,437

Separating Possible Scanners

- Which of remainder are likely, but undetected scanners?
  - Argument nearly circular
  - Show that there are properties plausibly used to distinguish likely scanners in the remainder
  - Use that as a ground truth to develop an algorithm against

Data Analysis (cont.)

- First model
  - Look at remainder hosts making failed connections
  - Compare all of remainder to known bad
  - Hope for two modes, where the failed connection mode resembles the known bad
  - No such modality exists

Data Analysis (cont.)

- Second model
  - Examine ratio of hosts with failed connections made to successful connections made
  - Known bad have a high percentage of failed connections
  - Conclusion: remainder hosts with <80% failure are potentially benign
  - Rest are suspect

Variables

Y_i Trial i; 0=connection succeeded, 1=failed

θ_0 Probability connection succeed given source is begin

θ_1 Probability connection succeed given source is a scanner

α Ideal false positive upper bound

β Ideal detector lower bound

TRW – continued

- Detect failed/succeeded connections
- Sequential Hypothesis Testing
  - Two hypotheses: benign (H_0) and scanner (H_1)
  - Probabilities determined by the equations
  - Theta_0 > theta_1 (benign has higher chance of succeeding connection)
  - Four outcomes: detection, false positive, false negative, nominal

$$\Pr[Y_i = 0 | H_0] = \theta_0, \quad \Pr[Y_i = 1 | H_0] = 1 - \theta_0$$
$$\Pr[Y_i = 0 | H_1] = \theta_1, \quad \Pr[Y_i = 1 | H_1] = 1 - \theta_1$$

Thresholds

- Choose Thresholds
  - Set upper and lower thresholds, n_0 and n_1
  - Calculate likelihood ratio
  - Compare to thresholds

$$\Lambda(Y) \equiv \frac{\Pr[Y|H_1]}{\Pr[Y|H_0]} = \Pi_{i=1}^{n} \frac{\Pr[Y_i|H_1]}{\Pr[Y_i|H_0]}$$

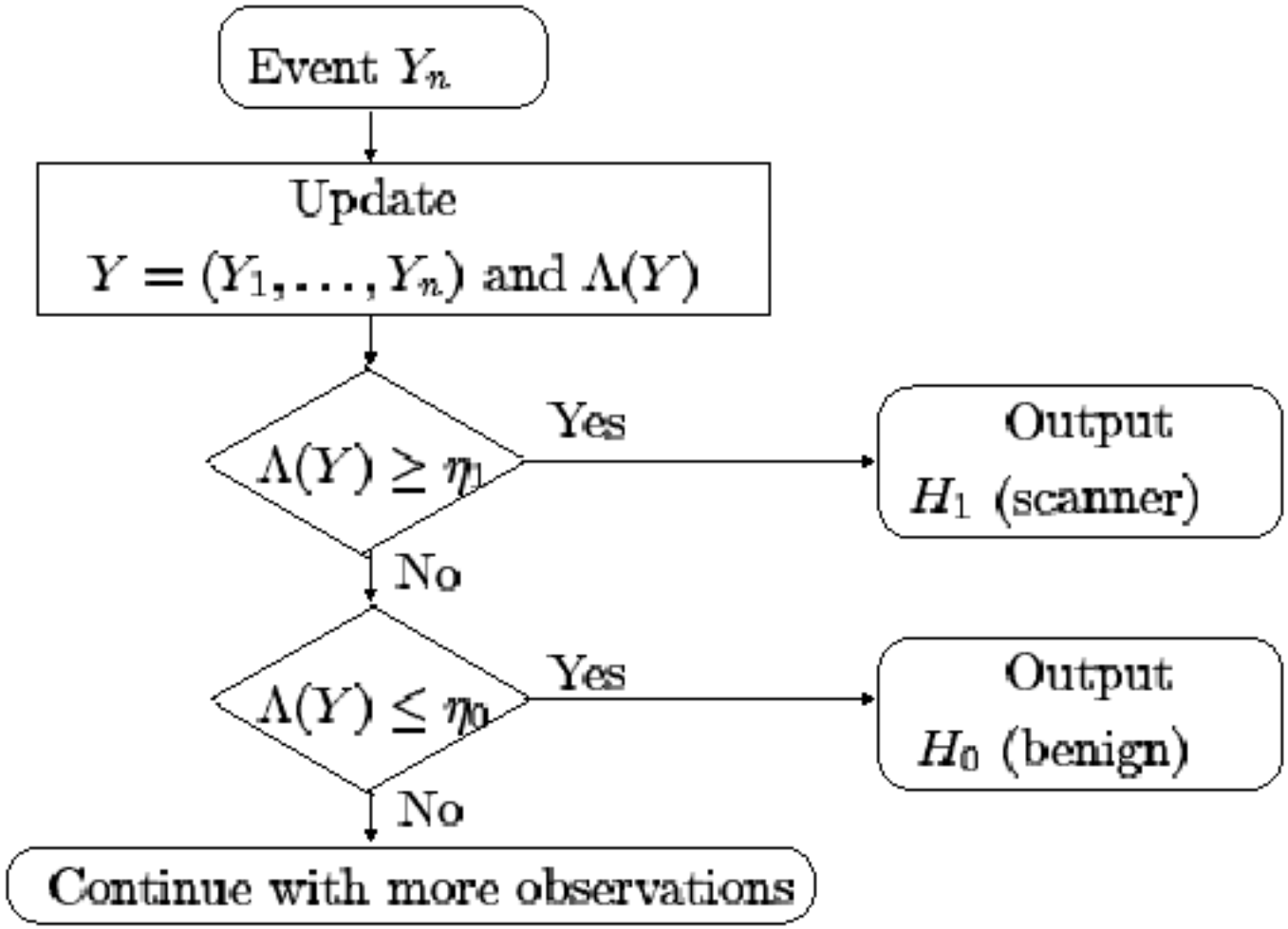


Figure 3. Flow diagram of the real-time detection algorithm

Choosing Thresholds

- Choose two constants, alpha and beta
  - Probability of false positive (P_f) <= alpha
  - Detection probability (P_d) >= beta
  - Typical values: alpha = 0.01, beta = 0.99
- Thresholds can be defined in terms of P_f and P_d or alpha and beta
  - n_1 <= P_d/P_f
  - n_0 >= (1-P_d)/(1-P_f)
  - Can be approximated using alpha and beta
  - n_1 = beta/alpha
  - n_0 = (1-beta)/(1-alpha)

Evaluation Methodology

- Used the data from the two labs
- Knowledge of whether each connection is established, rejected, or unanswered
- Maintains 3 variables for each remote host
  - D_s, the set of distinct hosts previously connected to
  - S_s, the decision state (pending, H_0, or H_1)
  - L_s, the likelihood ratio

Evaluation Methodology (cont.)

- For each line in dataset
  - Skip if not pending
  - Determine if connection is successful
  - Check whether is already in connection set; if so, proceed to next line
  - Update D_s and L_s
  - If L_s goes beyond either threshold, update state accordingly

Results

Type		LBL Count	LBL P_D	LBL $\overline{N}$	LBL Max N	ICSI Count	ICSI P_D	ICSI $\overline{N}$	ICSI Max N
scan	Total	122	-	-	-	7	-	-	-
	H_1	122	1.000	4.0	6	7	1.000	4.3	6
worm	Total	32	-	-	-	51	-	-	-
	H_1	27	0.844	4.5	6	45	0.882	5.1	6
	PENDING	5	-	-	5	6	-	-	5
other_bad	Total	13257	-	-	-	0	-	-	-
	H_1	13059	0.985	4.0	10	0	-	-	-
	H_0	15	-	5.1	10	0	-	-	-
	PENDING	183	-	-	11	0	-	-	-
benign	Total	2811	-	-	-	96	-	-	-
	H_1	33	-	8.1	24	0	-	-	-
	H_0	2343	-	4.1	16	72	-	4.0	4
	PENDING	435	-	-	14	24	-	-	9
suspect	Total	692	-	-	-	236	-	-	-
	H_1	659	0.952	4.1	16	234	0.992	4.0	8
	PENDING	33	-	-	7	2	-	-	7

TRW Evaluation

- Efficiency – true positives to rate of H1
- Effectiveness – true positives to all scanners
- N – Average number of hosts probed before detection

Trace	Measures	TRW	Bro	Snort
LBL	Efficiency	0.963	1.000	0.615
	Effectiveness	0.960	0.150	0.126
	$\overline{N}$	4.08	21.40	14.06
ICSI	Efficiency	1.000	1.000	1.000
	Effectiveness	0.992	0.029	0.029
	$\overline{N}$	4.06	36.91	6.00

Table 8. Comparison of the efficiency and effectiveness across TRW, Bro, and Snort

TRW Evaluation (cont.)

- TRW is far more effective than the other two
- TRW is almost as efficient as Bro
- TRW detects scanners in far less time

Potential Improvements

- Leverage Additional Information
  - Factor for specific services (e.g. HTTP)
  - Distinguish between unanswered and rejected connections
  - Consider time local host has been inactive
  - Consider rate
  - Introduce correlations (e.g. 2 failed in a row worse than 1 fail, 1 success, 1 fail)
  - Devise a model on history of the hosts

Improvements (cont.)

- Managing State
  - Requires large amount of maintained states for tracking
  - However, capping the state is vulnerable to state overflow attacks
- How to Respond
  - What to do when a scanner is detected?
  - Is it worth blocking?
- Evasion and Gaming
  - Spoofed IPs
    - Institute “whitelists”
    - Use a honeypot to try to connect
  - Evasion (inserting legitimate connections in scan)
    - Incorporating other information, such as a model of what is normal for legitimate users and give less weight to connections not fitting the pattern
- Distributed Scans
  - Scans originating from more than one source
  - Difficult to fix in this framework

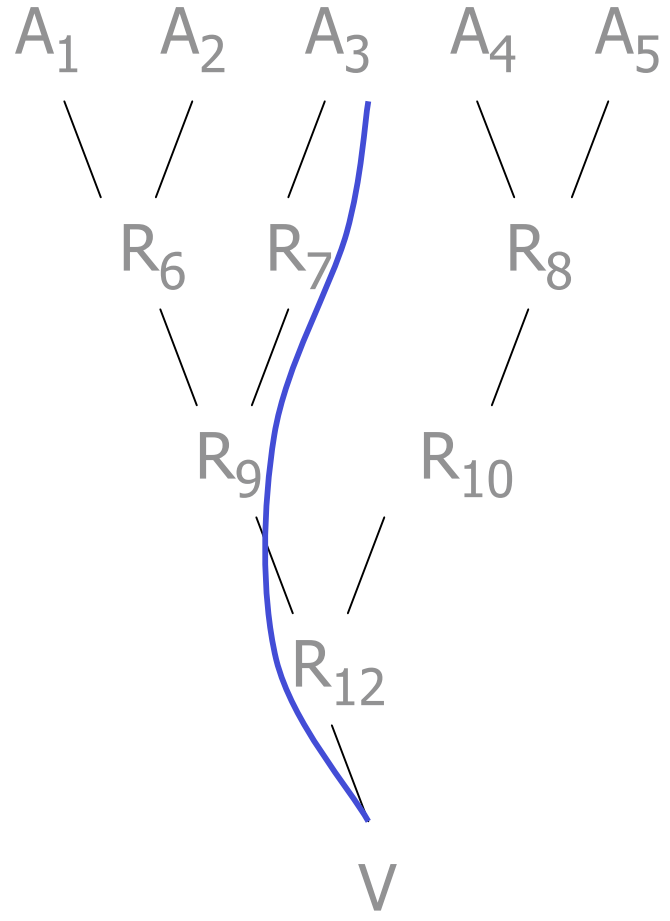
Summary

- TRW- based on ratio of failed/succeeded connections
- Sequential Hypothesis Testing
- Highly accurate
  - 4-5 vs 20 attempts on average. Meaningful?
- Quick Response

Modify routers to allow IP traceback

Traceback problem

- Goal
  - Given set of packets
  - Determine path
- Assumptions
  - Most routers remain uncompromised
  - Attacker sends many packets
  - Route from attacker to victim remains relatively stable

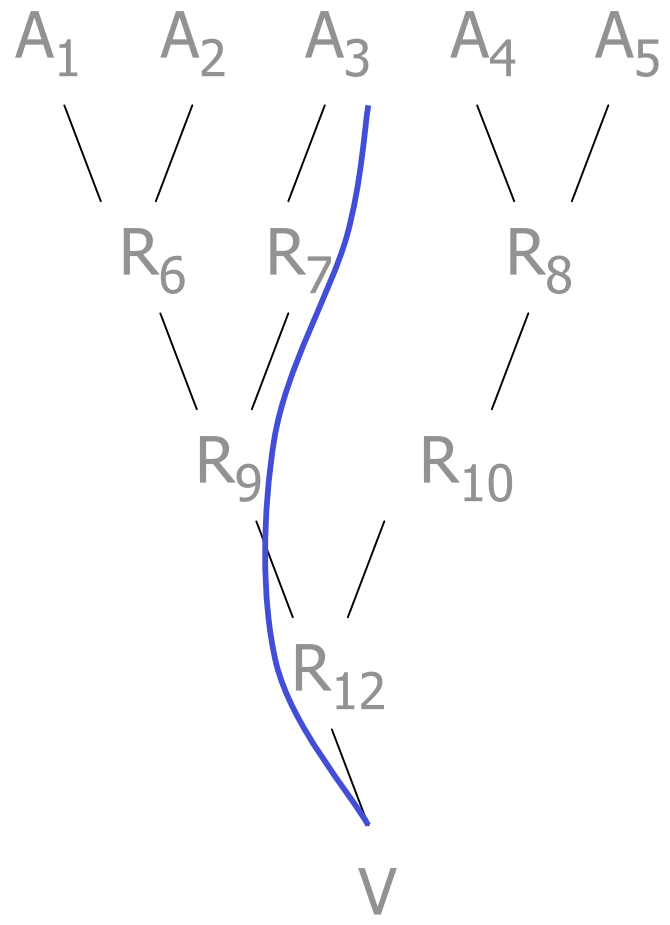


Simple method

- Record path
  - Each router adds IP address to packet
  - Victim reads path from packet
- Problem
  - Requires space in packet
    - Path can be long
    - No extra fields in current IP format
      - Changes to packet format are not practical

Better idea

- Many packets
  - DDoS involves many packets on same path
- Store one link in each packet
  - Each router probabilistically stores own address
  - Fixed space regardless of path length



Edge Sampling

- Data fields
  - Edge: *start* and *end* IP addresses
  - Distance: number of hops since edge stored
- Marking procedure for router R

  with probability p

  write R into start address

  write 0 into distance field

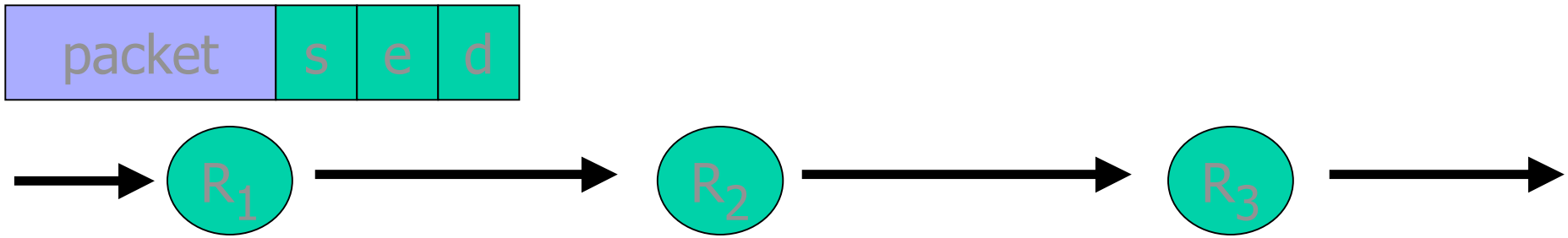
  else

  if distance ==0 write R into end field

  increment distance field

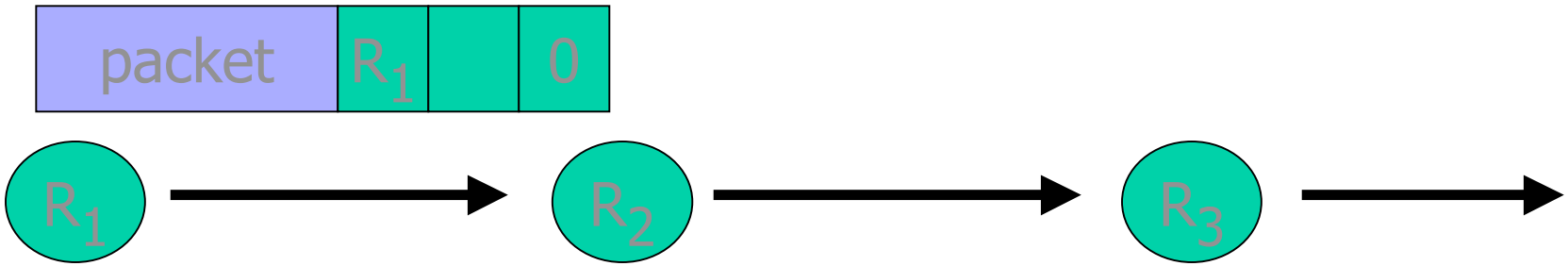
Edge Sampling: picture

- Packet received
  - R_1 receives packet from source or another router
  - Packet contains space for start, end, distance



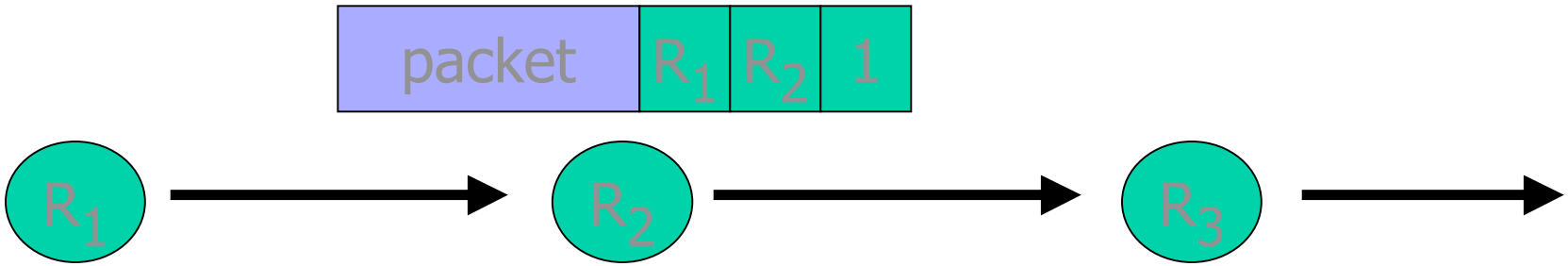
Edge Sampling: picture

- Begin writing edge
  - R_1 chooses to write start of edge
  - Sets distance to 0



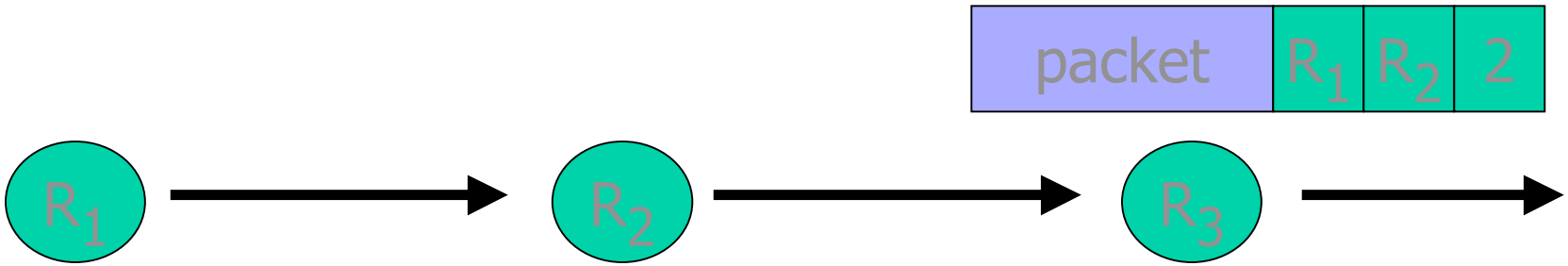
Edge Sampling

- Finish writing edge
  - R_2 chooses not to overwrite edge
  - Distance is 0
    - Write end of edge, increment distance to 1



Edge Sampling

- Increment distance
  - R_3 chooses not to overwrite edge
  - Distance >0
    - Increment distance to 2



Path reconstruction

- Extract identifiers from attack packets
- Build graph rooted at victim
  - Each (start,end,distance) tuple provides an edge
  - Eliminate edges with inconsistent distance
  - Traverse edges from root to find attack paths
- # packets needed to reconstruct path

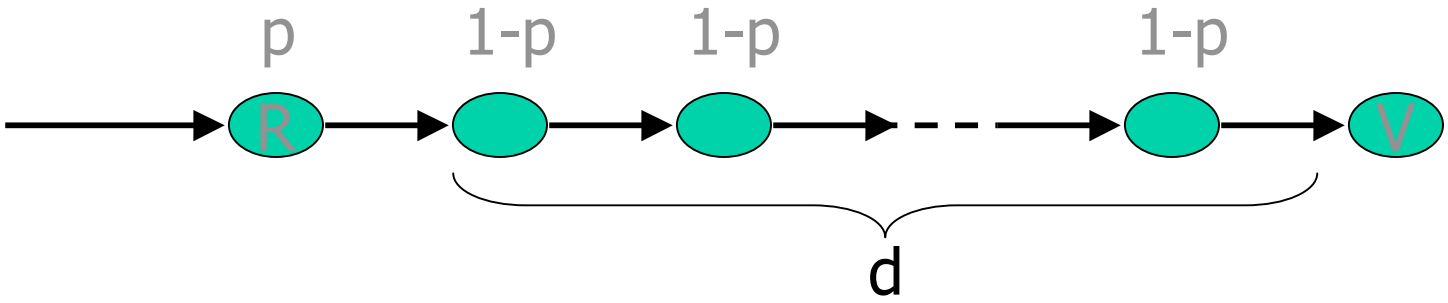
$$E(X) < \frac{\ln(d)}{p(1-p)^{d-1}}$$

where p is marking probability, d is length of path

Optimal p is 1/d ... can vary probability by distance

Node Sampling?

- Less data than edge sampling
  - Each router writes own address with probability p
- Infer order by number of packets
  - Router at distance d has probability $p(1-p)^d$ of showing up in marked packet

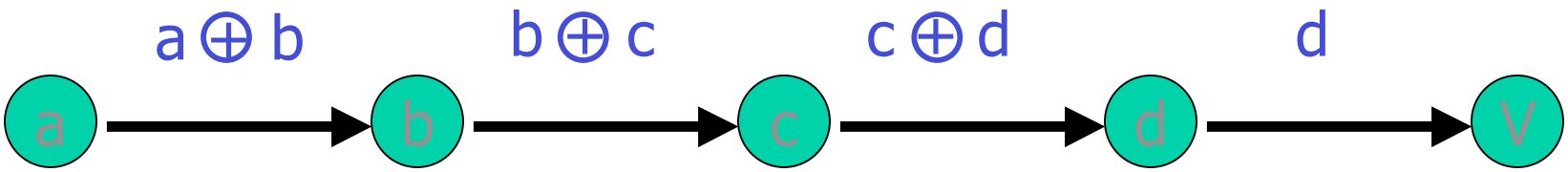


- Problems
  - Need many packets to infer path order
  - Does not work well if many paths

Reduce Space Requirement

- XOR edge IP addresses
  - Store edge as start ⊕ end
  - Work backwards to get path:

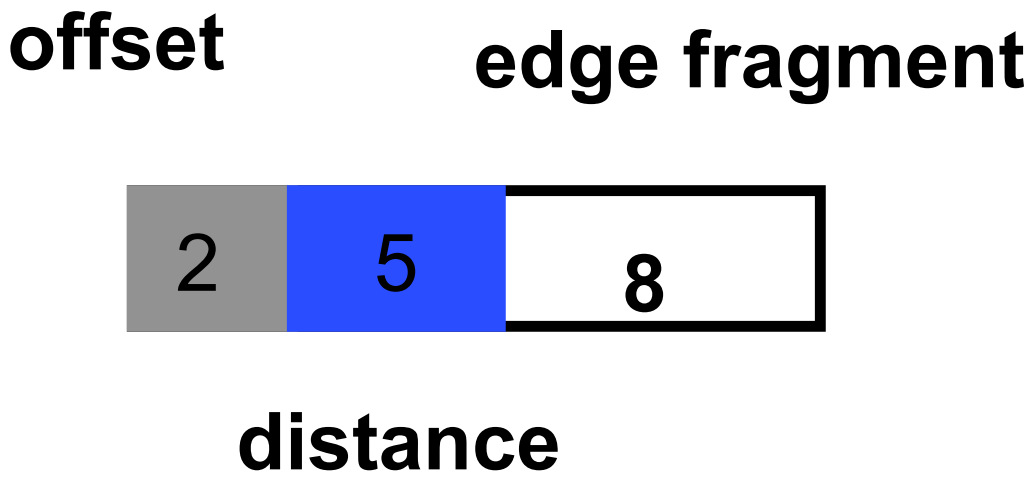
    (start ⊕ end) ⊕ end = start
- Sample attack path



Creating Unique Edge-ids

- Edge-id fragments are not unique
  - with multiple attackers, multiple edge fragments with the same offset and distance
- Bit-interleave has code with IP address

Encoding Edge Fragments



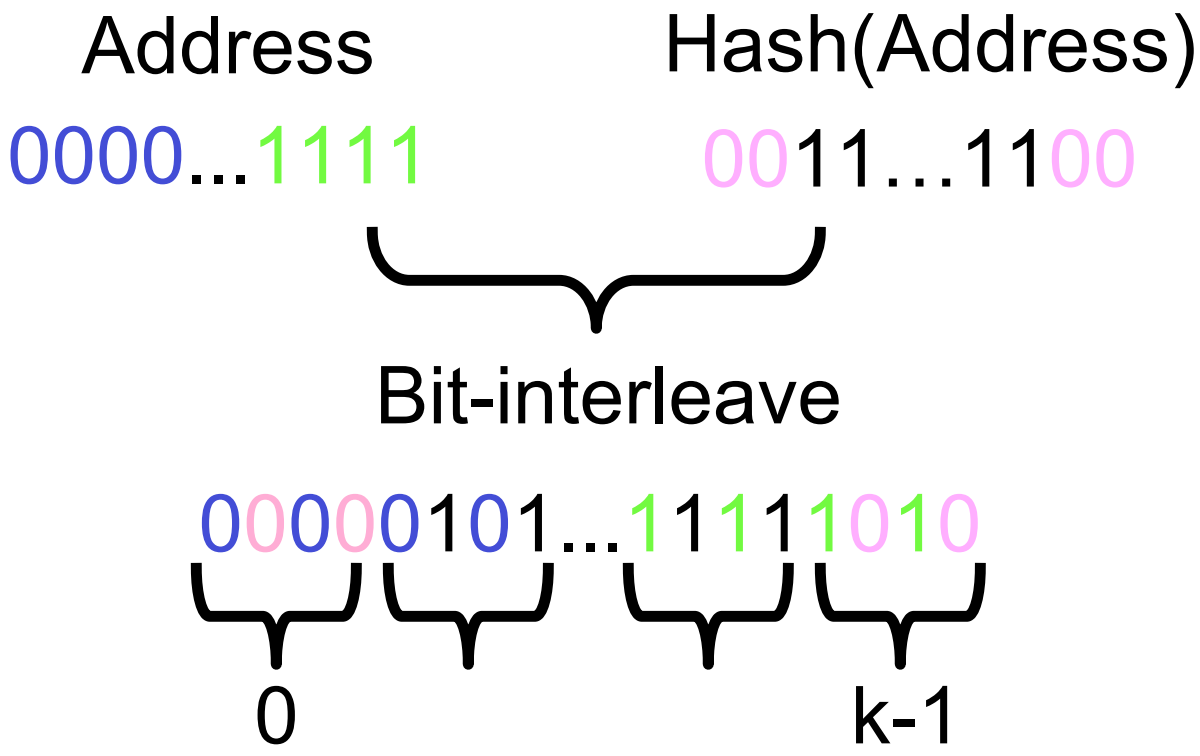
IP Header Encoding

- Backwards compatibility
- Two problems
  - Writing same values into id fields of frags from different datagrams
  - Writing different values into id fields of frags of same datagrams

Fragmentation Issues

- Copy data into ICMP packet
- Check the checksum at higher level
- etc

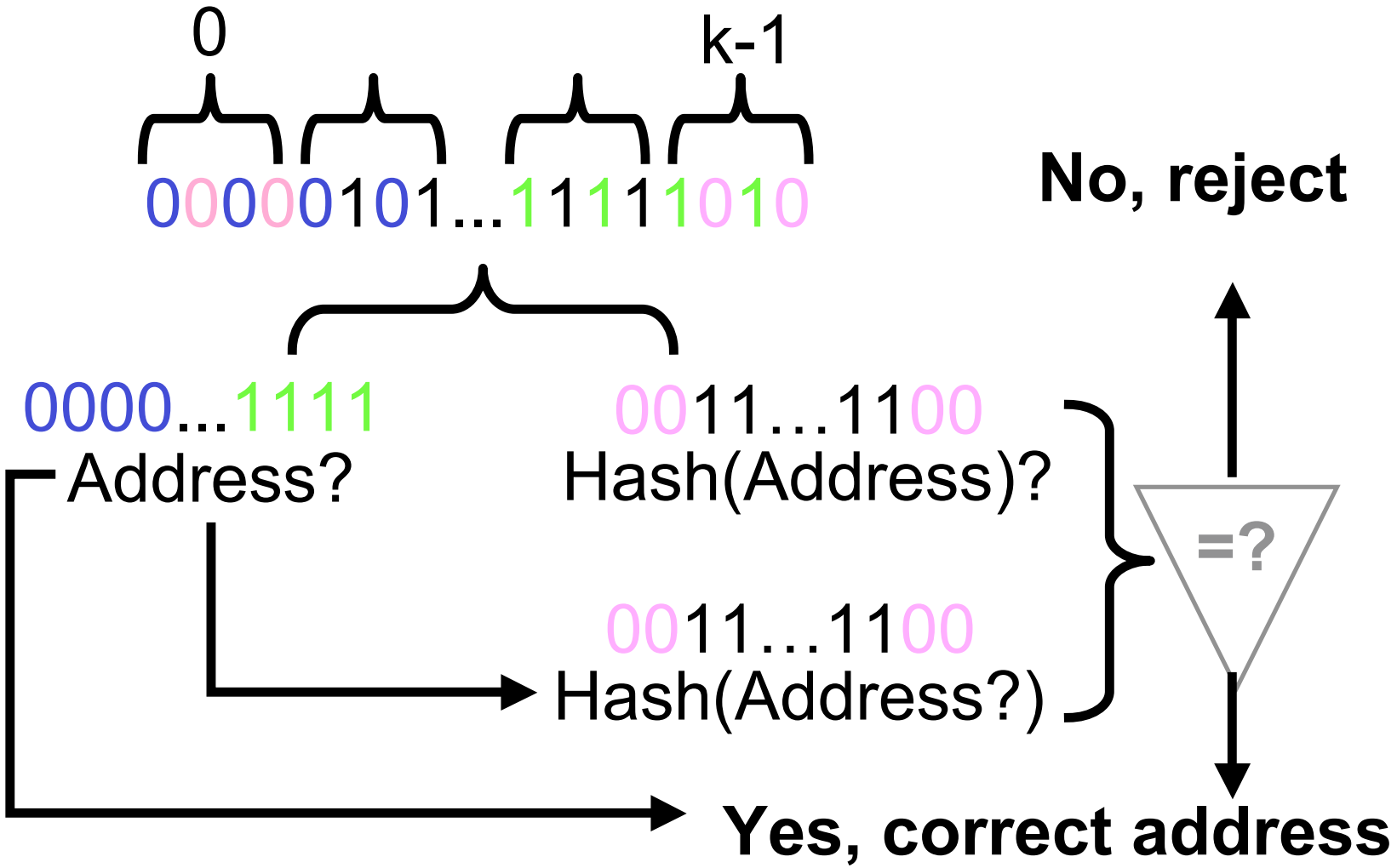
Creating Unique Edge-ids



Candidate Edge-ids

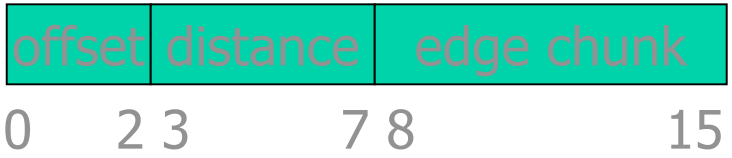
- Combine all permutations of fragments at each distance with disjoint offset values
- Check that the hash matches hash of the address

Construction Candidate Edges



Details: where to store edge

- Identification field
  - Used for fragmentation
  - Fragmentation is rare
  - 16 bits
- Store edge in 16 bits?



  - Break into chunks
  - Store start + end

Version	Header Length
Type of Service	
Total Length	
Identification	
Flags	Fragment Offset
Time to Live	
Protocol	
Header Checksum	
Source Address of Originating Host	
Destination Address of Target Host	
Options	
Padding	
IP Data	

Edge Sampling (Cont)

- The expected number of packets needed for the victim to reconstruct the entire path is at most $\ln(d)/p(1-p)^{d-1}$
  - Example: p=0.1, d=10, reconstruction requires about 75 packets
  - This is related to the coupon-collection problem
- Edge sampling allows reconstruction of the whole attack tree
- Encoding start, end, and distance is a problem
  - Not backward compatible if we change the IP header!
  - There are ways around this

Digression: Coupon Collection

- Suppose you have t types of coupons, C_1, C_2, … , C_t
  - Each time you open baseball cards, you get a coupon of type i with probability 1/t
  - How many coupons do you need before you have a complete set?
    - Note that in real competitions, all types are not 1/t
    - Call total number you need N (a random variable)
  - Define a random variable N_i indicating the number of draws you need to use when you hold i-1 coupon types and you want a new type
    - Then $N = N_1 + N_2 + \ldots + N_t$

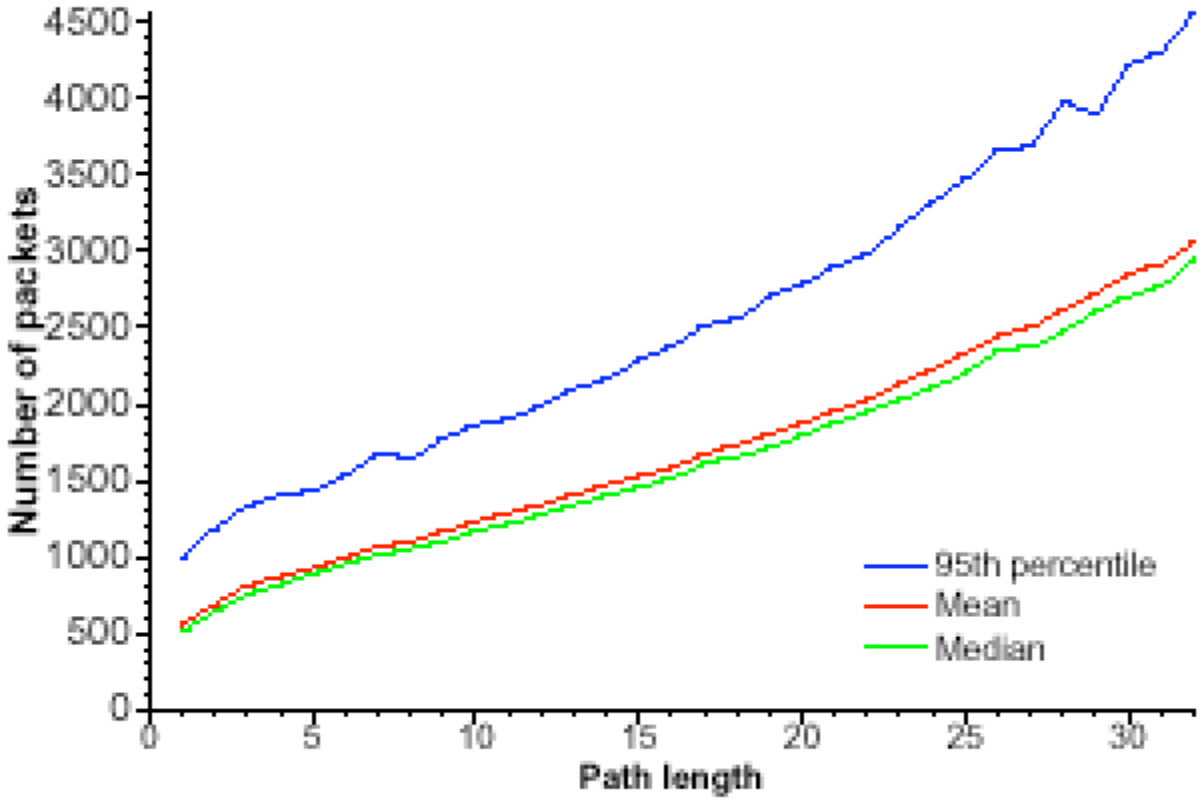
Coupon Collection (cont)

- Then $E(N) = E(N_1) + \ldots E(N_t)$
  - Linearity of expectation
- What is $E(N_i)$?
  - Probability of getting a new type if you have i-1 types is (N-i+1)/N, so expectation is N/(N-i+1)
    - For geometric random variables, expectation is the inverse of the parameter
    - If you have a fair die, it takes an expected 6 rolls to get a 4 (for example)
  - So $E(N) = 1 + N/(N-2) + N/(N-3) + \ldots + N$ or $N\ H_N$
    - Here H_N is the Nth harmonic number
    - This is approximately N ln N.

Testing the Algorithm

- Uses Simulation
  - Create random paths
  - Originate attacks
- Mark P() = 1/25
- 1,000 random test runs
- Vary path lengths

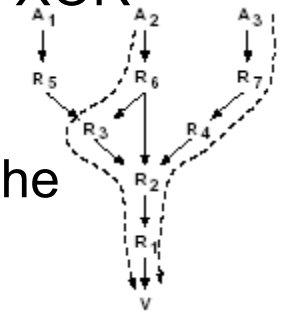
Results


Number of packets
4500
4000
3500
3000
2500
2000
1500
1000
500
0
0
5
10
15
20
25
30
Path length
95th percentile
Mean
Median

Advanced Marking Schemes for IP Traceback

- Approach by Savage et. al. is impractical when > 25 *attackers*
- Use Internet maps and alternative encoding techniques to reconstruct multiple attack paths with high probability

- Victims can either maintain a current map of the routers leading to them or they can create such maps after the attack for analysis.
- The edges are represented as the current router information XOR the information of an upstream router.
- To minimize the number of collisions and hence reduce the number of false positives due to many routers ending up at the same distance, ***a set of hashing functions is used***.



Solution continued…

- A set of hash functions of size 2^w is used.
- A router with probability q inserts its own information hashed with random hash function h, whose index x is w bits. Also distance is set to 0. Otherwise the router increments the distance and XORs the result of $h(x,$ current IP) with the edge information in the packet.
- When the victim performs reconstruction it will perform a breadth first search and then match the decoded packets with the routers it is aware of in the tree. ***To reduce the number of false positives a m- threshold scheme can be used which requires a router to be identified as an attacker m times.***
- The reconstruction scheme can identify more than 2000 attacker sites with very few false positives and runs in $O(\sum_d |S_d| * |E_{d+1}|)$ vs. Savage's $O(\sum_d |S_d| * |E_{d+1}|^8)$ (S_d set of routers on the attack path at distance d, E_d unique edges at distance d)
- Not all the packets have to be collected before the reconstruction can begin.

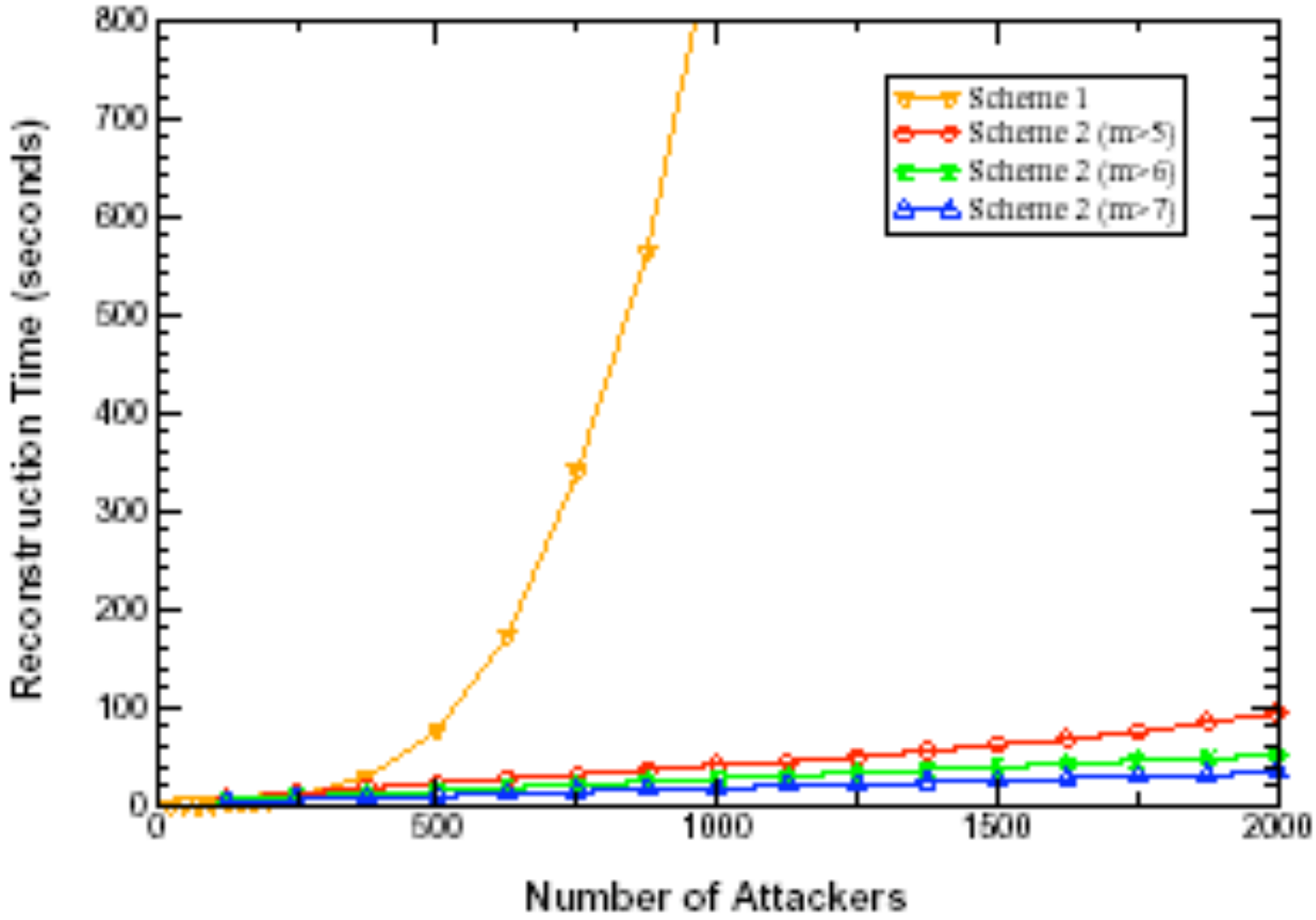
Solution continued…

- Routers can use time varying MACs (Message Authenticating Codes) to sign the packets. This will allow the victim to reconstruct the map by referencing the timestamp of the packet and the public site that contains the MAC keys for routers at different time periods.

Analysis/Impact

- The scheme does not require all of the packets to be received before the reconstruction can begin and a victim does not need the entire Internet map as long as most of the offending paths are on the map.

Results


Reconstruction Time (seconds)
800
700
600
500
400
300
200
100
0
0
500
1000
1500
2000
Number of Attackers
Scheme 1
Scheme 2 (m=5)
Scheme 2 (m=6)
Scheme 2 (m=7)