Table 3: Geographic Distribution of Results

Continent	Spoofing Successes	Spoofing Rate
N. America	498	18.2%
S. America	44	19.4%
Europe	389	19.1%
Asia	289	32.6%
Oceania	40	25.6%
Africa	15	17.4%

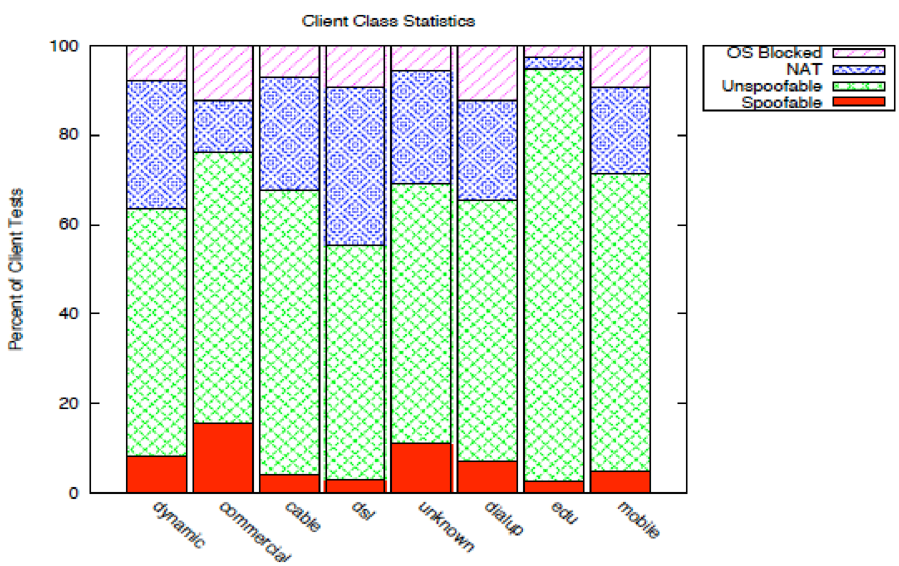


Table 5: Longitudinal comparison between three-month periods in 2005 and 2009 with 1,100 and 400 distinct clients respectively.

	Proportion Spoofable		
Metric	2005 (dest. MIT)	2009 (dest. MIT)	2009 (all dests.)
Sessions	18.8±3.2%	29.9±6.0%	31.2±6.0
Netblocks	20.0±3.5%	30.2±6.4%	31.7±6.5
Addresses	5.0±1.8%	11.0±4.1%	11.1±4.1
ASes	23.4±5.0%	31.8±7.6%	34.1±7.6

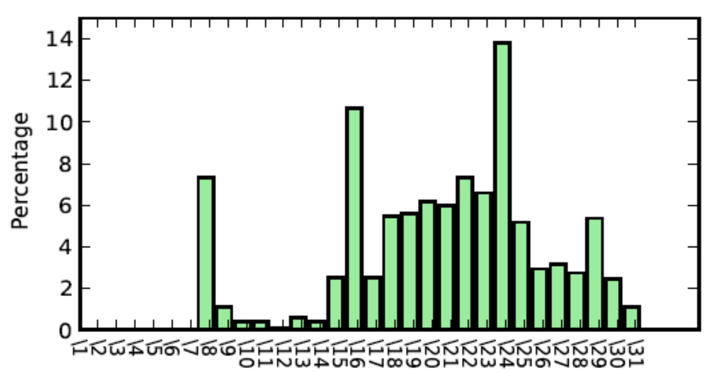
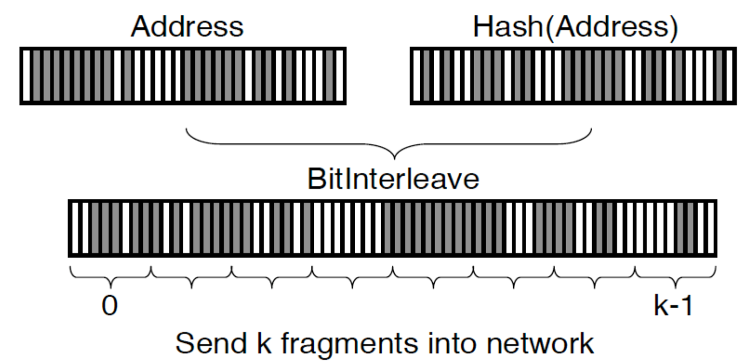
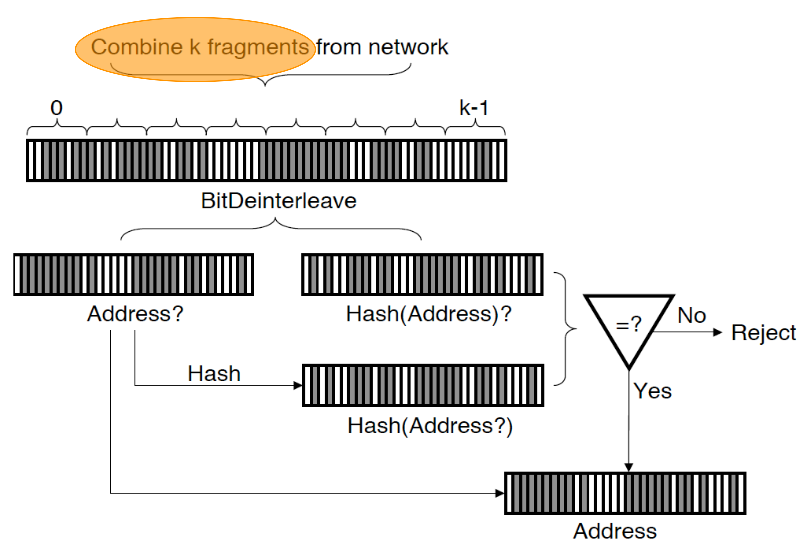
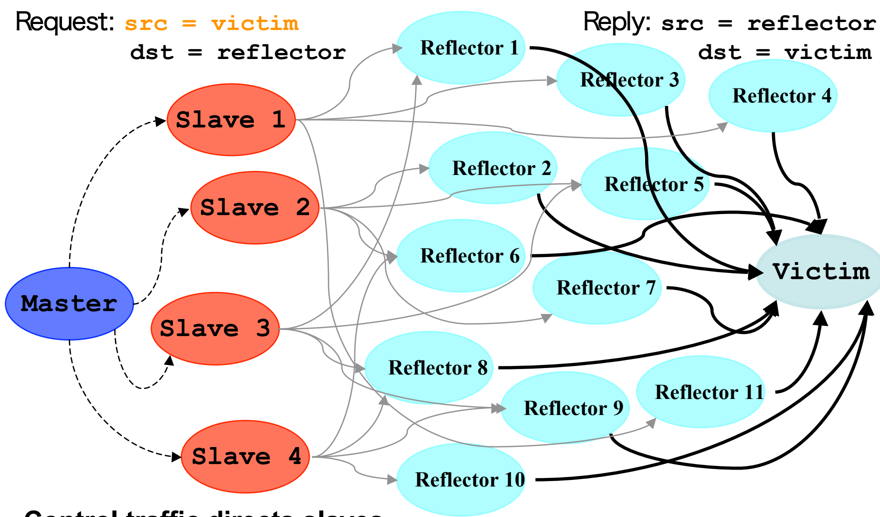


Figure 8: Spoofing neighboring addresses: Probability mass of filtering policy granularity


Address
Hash(Address)
BitInterleave
0
k-1
Send k fragments into network


Combine k fragments from network
0
k-1
BitDeinterleave
Address?
Hash(Address)?
Hash
Hash(Address?)
=?
No
Reject
Yes
Address

Diffuse DDoS: Reflector Attack



Control traffic directs slaves at victim & reflectors

Reflectors send streams of non-spoofed but unsolicited traffic to victim

Marking procedure at router R:

for each packet w
    let x be a random number from [0..1)
    if $x < p$ then
        write R into w.start and 0 into w.distance
    else
        if w.distance = 0 then
            write R into w.end
        increment w.distance

Path reconstruction procedure at victim v:

```
let G be a tree with root v
let edges in G be tuples (start,end,distance)
for each packet w from attacker
   if w.distance = 0 then
      insert edge (w.start,v,0) into G
   else
      insert edge (w.start,w.end,w.distance) into G
remove any edge (x,y,d) with d ≠ distance from x to v in G
extract path (R_i..R_j) by enumerating acyclic paths in G
```

Marking procedure at router R:

```
let R' = BitIntereave(R, Hash(R))
let k be the number of non-overlapping fragments in R'
for each packet w
   let x be a random number from [0..1)
   if x < p then
      let o be a random integer from [0..k − 1]
      let f be the fragment of R' at offset o
      write f into w.frag
      write 0 into w.distance
      write o into w.offset
   else
      if w.distance = 0 then
         let f be the fragment of R' at offset w.offset
         write f ⊕ w.frag into w.frag
      increment w.distance
```

Path reconstruction procedure at victim v:

```
let FragTbl be a table of tuples (frag,offset,distance)
let G be a tree with root v
let edges in G be tuples (start,end,distance)
let maxd := 0
let last := v
for each packet w from attacker
    FragTbl.Insert(w.frag,w.offset,w.distance)
    if w.distance > maxd then
        maxd := w.distance
for d := 0 to maxd
    for all ordered combinations of fragments at distance d
        construct edge z
        if d ≠ 0 then
            z := z ⊕ last
        if Hash(EvenBits(z)) = OddBits(z) then
            insert edge (z,EvenBits(z),d) into G
            last := EvenBits(z);
remove any edge (x,y,d) with d ≠ distance from x to v in G
extract path (R_i..R_j) by enumerating acyclic paths in G
```