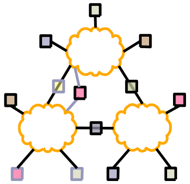
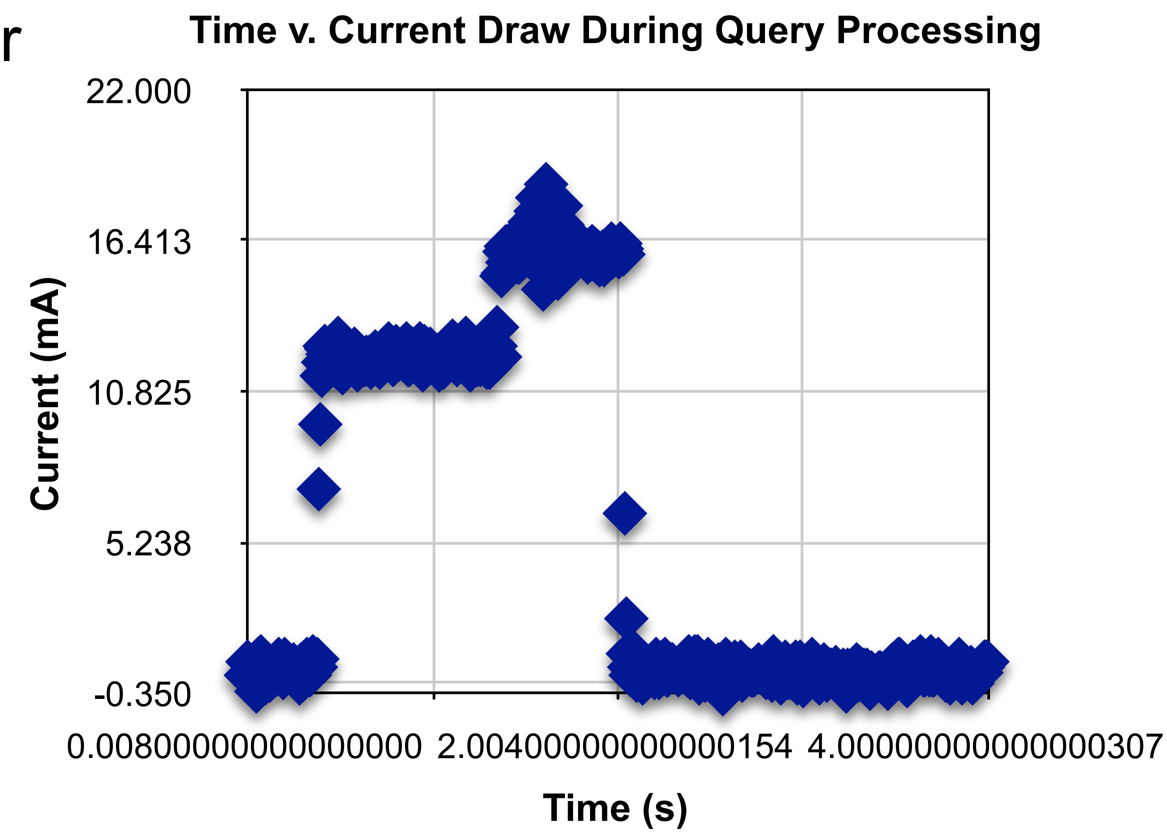
15-744: Computer Networking

Review



Sensor Nets Metric: Communication

- Lifetime from one pair of AA batteries
  - 2-3 days at full power
  - 6 months at 2% duty cycle
- Communication dominates cost
  - < few mS to compute
  - 30mS to send message



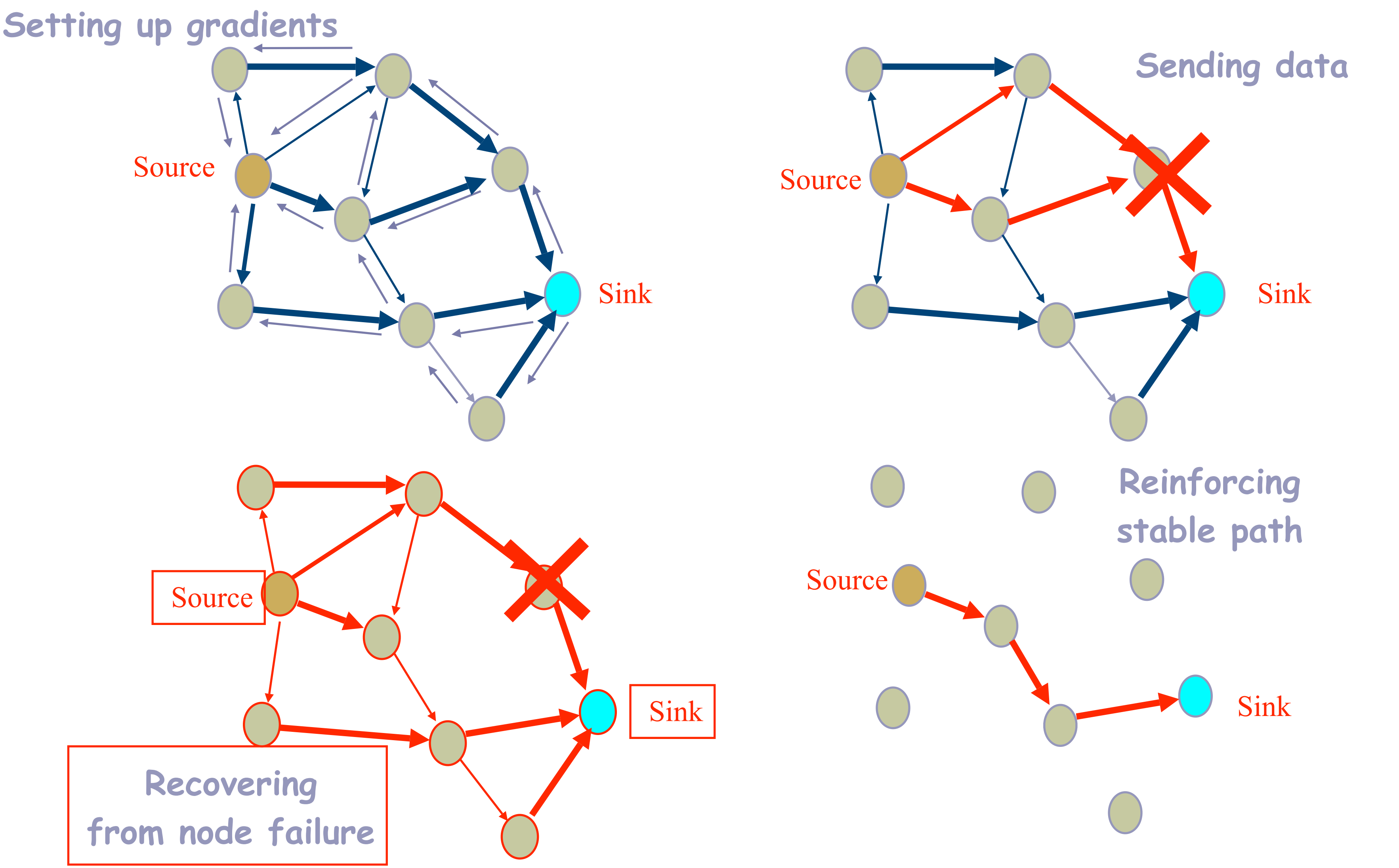
Directed Diffusion

- Data centric – nodes are unimportant
- Request driven:
  - Sinks place requests as interests
  - Sources are eventually found and satisfy interests
  - Intermediate nodes route data toward sinks
- Localized repair and reinforcement
- Multi-path delivery for multiple sources, sinks, and queries

Diffusion (High Level)

- Sinks broadcast interest to neighbors
- Interests are cached by neighbors
- Gradients are set up pointing back to where interests came from at low data rate
- Once a sensor receives an interest, it routes measurements along gradients

Illustrating Directed Diffusion



TAG Introduction

- Programming sensor nets is hard!
- Declarative queries are easy
  - Tiny Aggregation (TAG): In-network processing via declarative queries
- In-network processing of aggregates
  - Common data analysis operation
  - Communication reducing
    - Operator dependent benefit
  - Across nodes during same epoch
- Exploit semantics improve efficiency!

- Example:
    - Vehicle tracking application: 2 weeks for 2 students
    - Vehicle tracking query: took 2 minutes to write, worked just as well!

```
SELECT MAX(mag)
FROM sensors
WHERE mag > thresh
EPOCH DURATION 64ms
```

Basic Aggregation

- In each epoch:
  - Each node samples local sensors once
  - Generates **partial state record (PSR)**
    - local readings
    - readings from children
  - Outputs PSR during its comm. slot.

- At end of epoch, PSR for whole network output at root
- (In paper: pipelining, grouping)

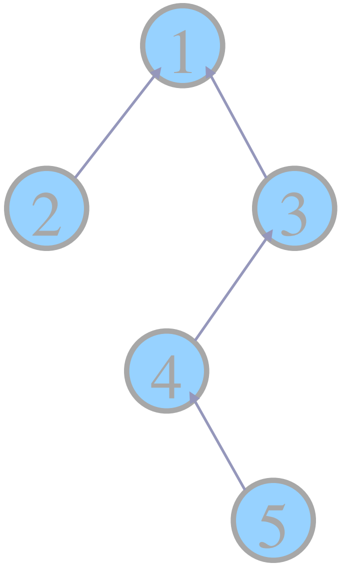


Illustration: Aggregation

```
SELECT COUNT(*)
FROM sensors
```

Sensor #

Slot #

	1	2	3	4	5
1					1
2					
3					
4					
1					

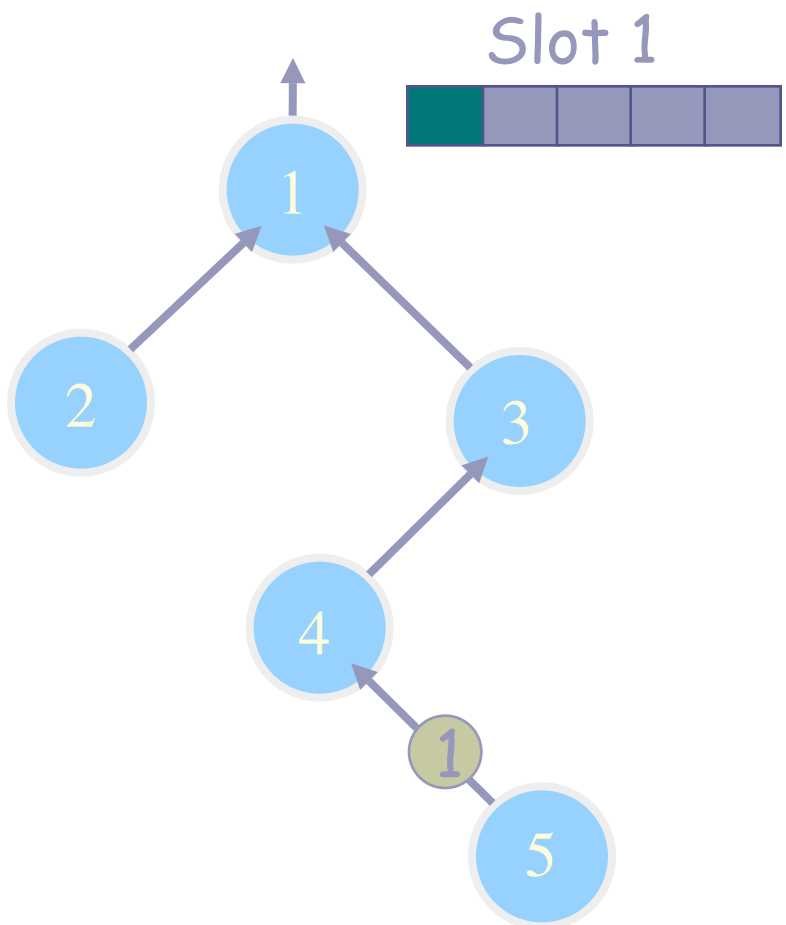


Illustration: Aggregation

```
SELECT COUNT(*)
FROM sensors
```

Sensor #

Slot #

	1	2	3	4	5
1					1
2				2	
3					
4					
1					

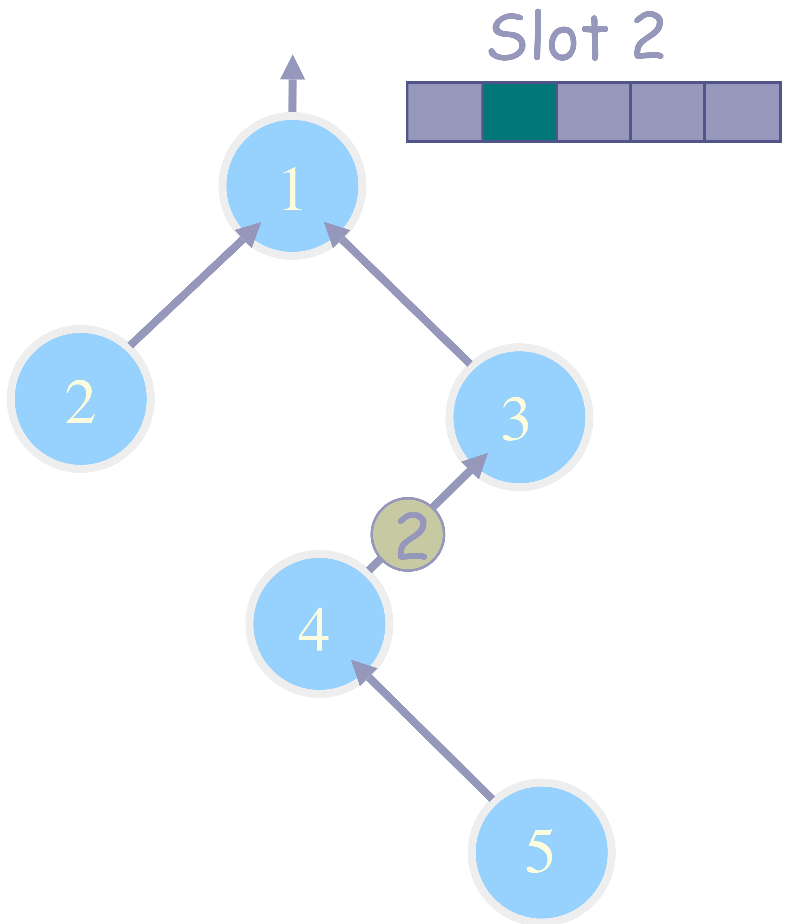
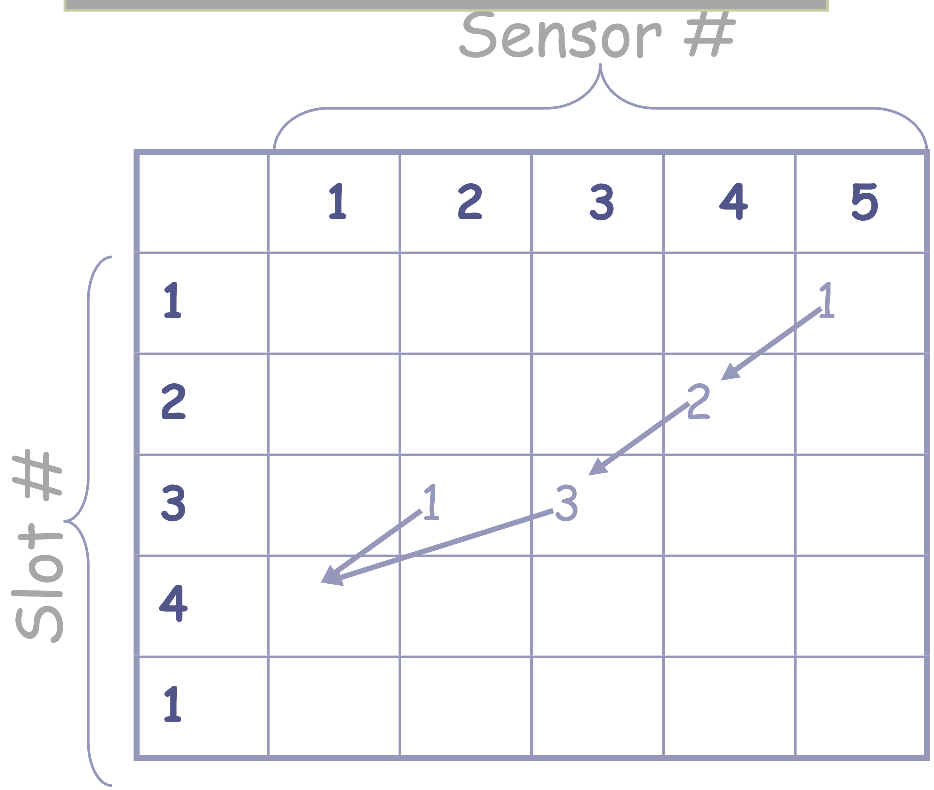


Illustration: Aggregation

```
SELECT COUNT(*)
FROM sensors
```



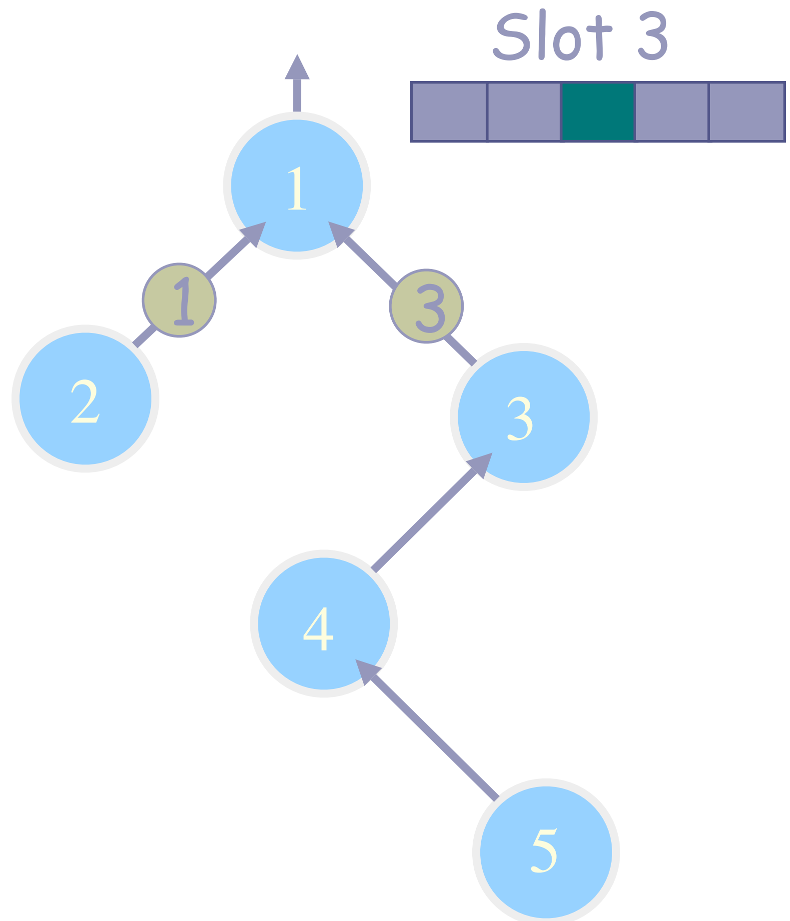
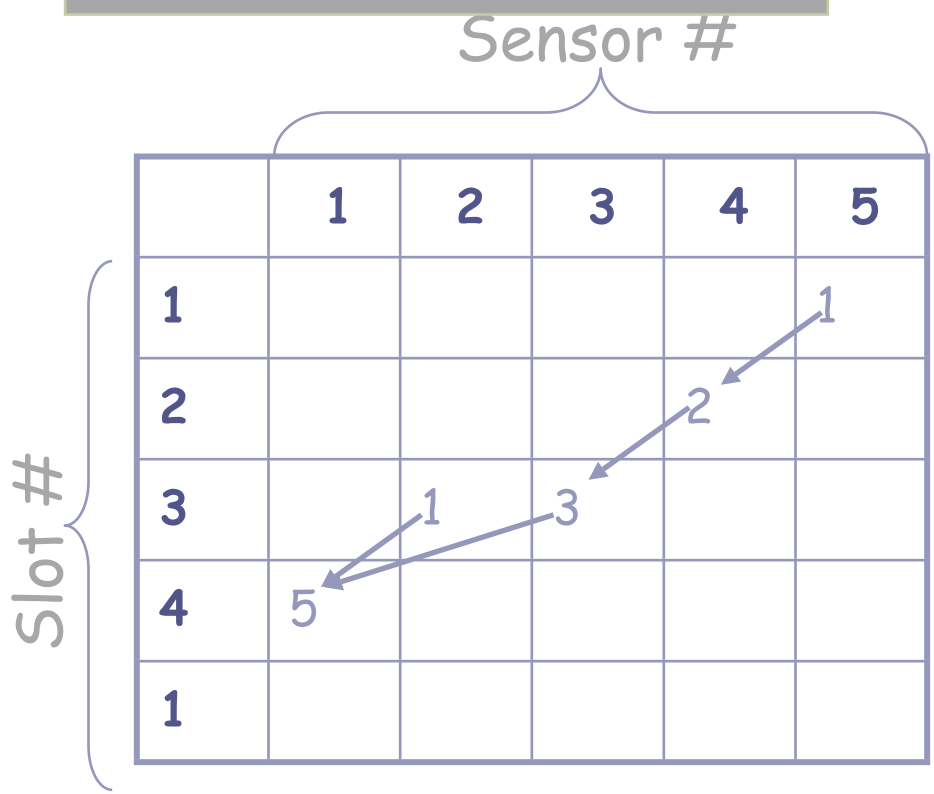


Illustration: Aggregation

```
SELECT COUNT(*)
FROM sensors
```



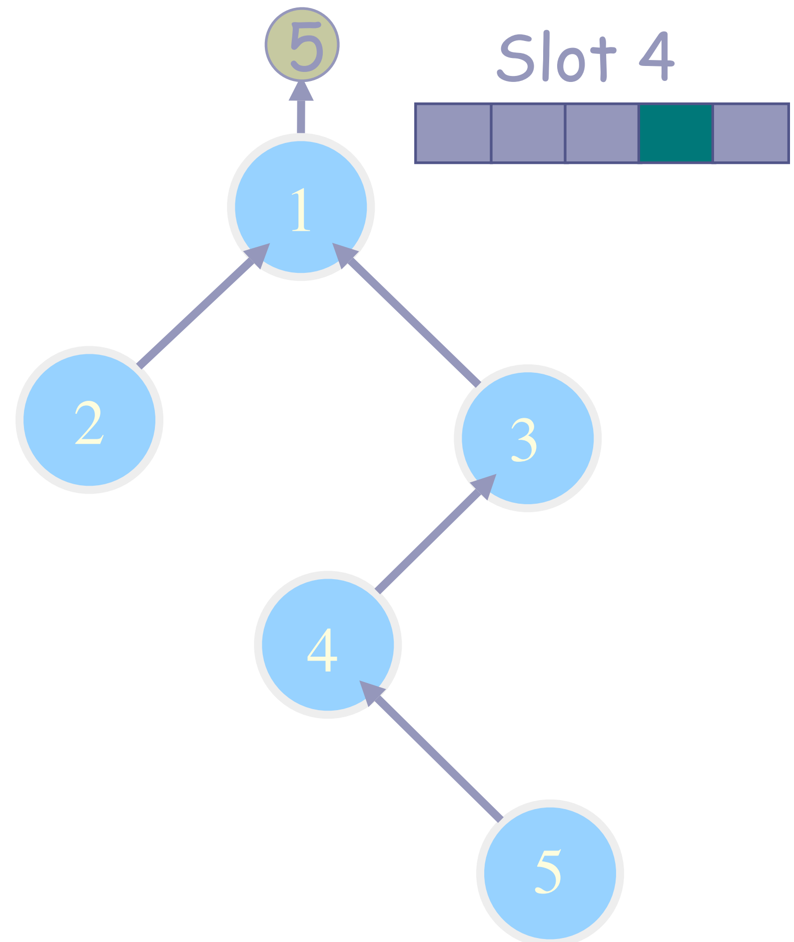
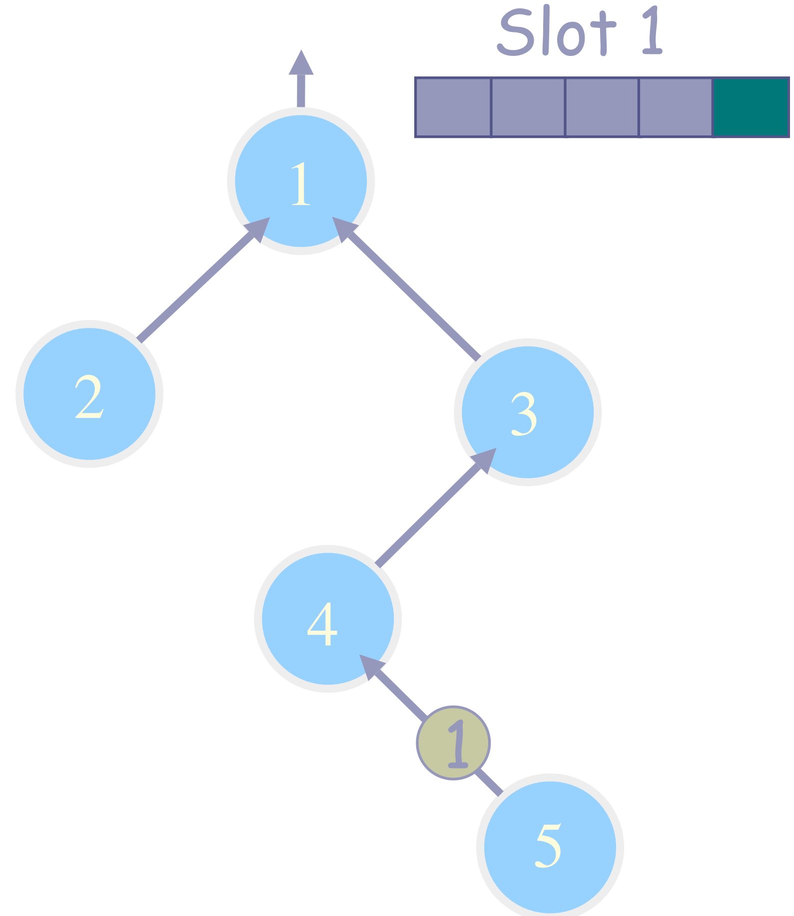


Illustration: Aggregation

```
SELECT COUNT(*)
FROM sensors
```

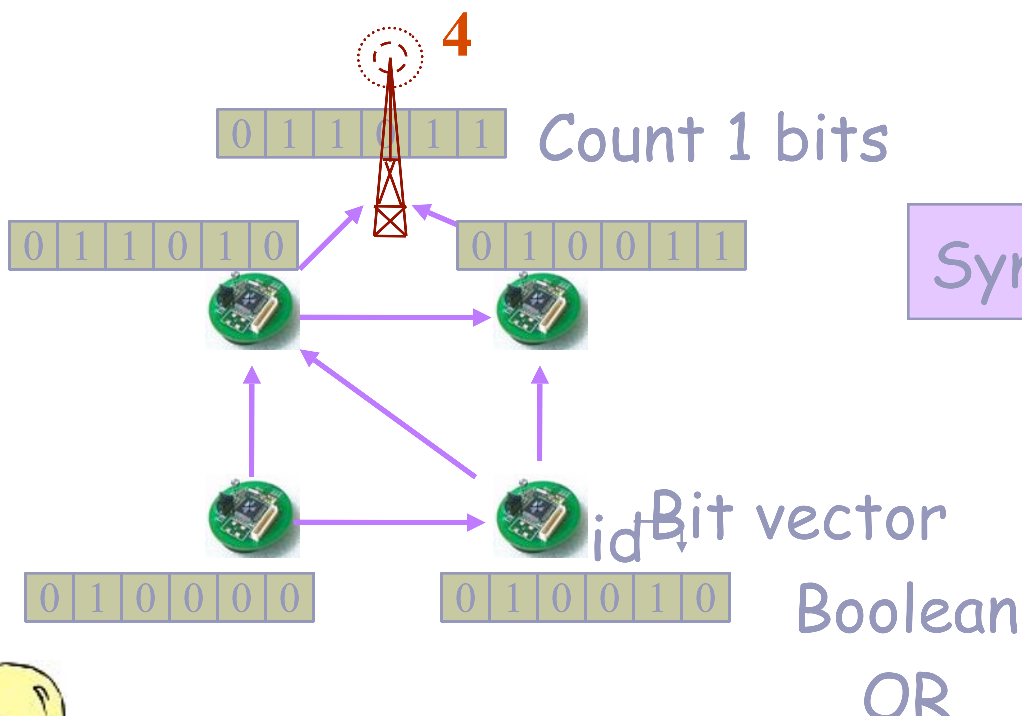
Sensor #

Slot #	1	2	3	4	5
1					1
2				2	
3		1	3		
4	5				
1					1



Synopsis Diffusion (SenSys'04)

- Goal: <u>count</u> the live sensors in the network



Challenge

Synopsis should be small

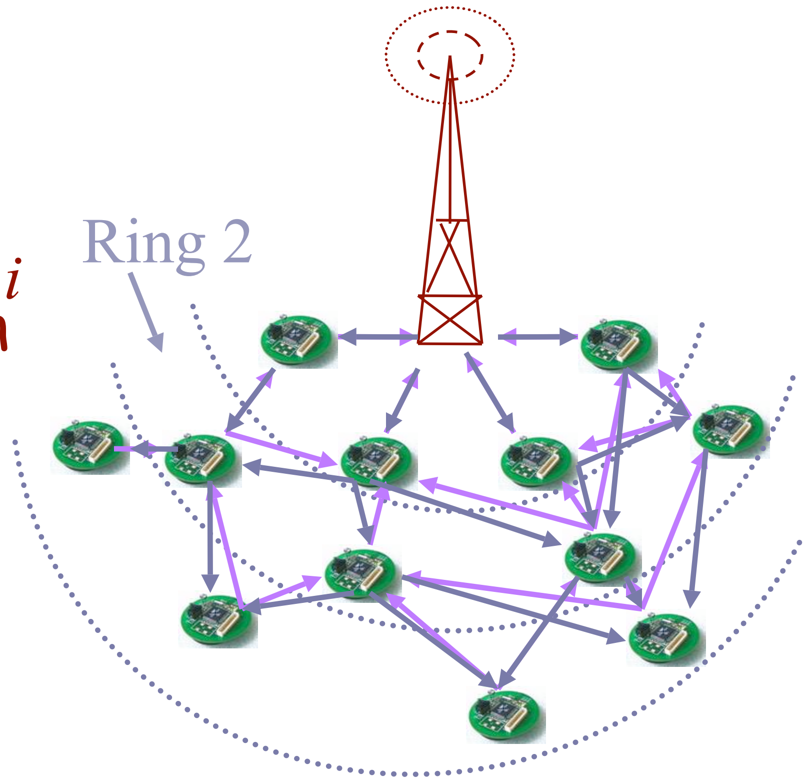
Approximate COUNT algorithm: logarithmic size bit vector

Synopsis Diffusion over Rings

- A node is in ring i if it is i hops away from the base-station

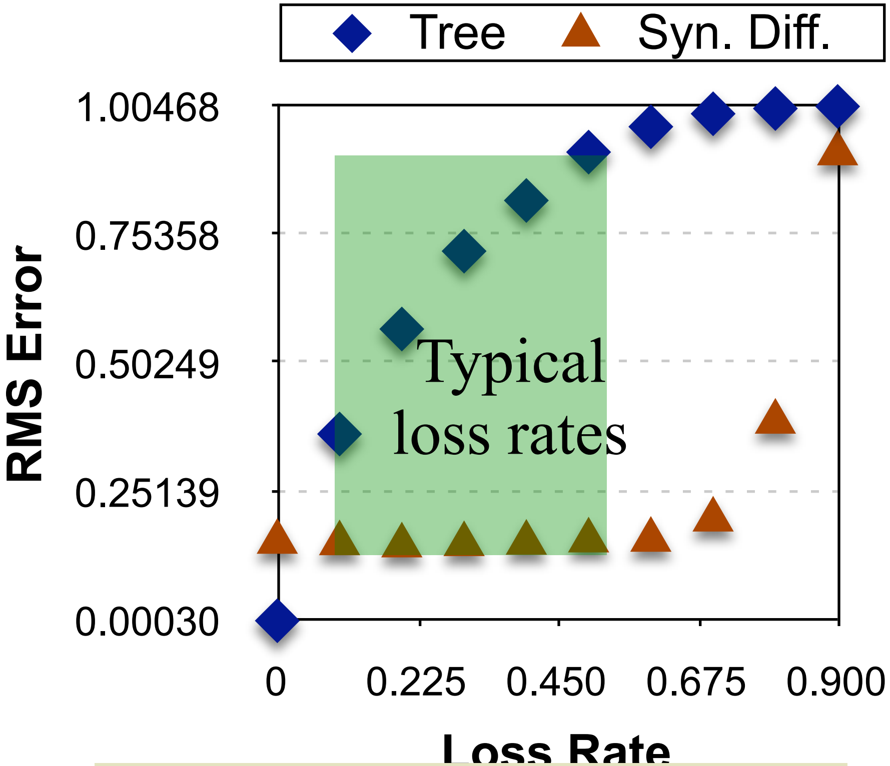
- <u>Broadcasts</u> by nodes in ring i are received by neighbors in ring i-1

- Each node transmits once = optimal energy cost (same as Tree)



Evaluation

Approximate COUNT with Synopsis Diffusion



Scheme	Energy
Tree	41.8 mJ
Syn. Diff.	42.1 mJ

Per node energy

More robust than Tree

Almost as energy efficient as Tree

15-744: Computer Networking

L-14 Network Topology

Trends in Topology Modeling

Observation

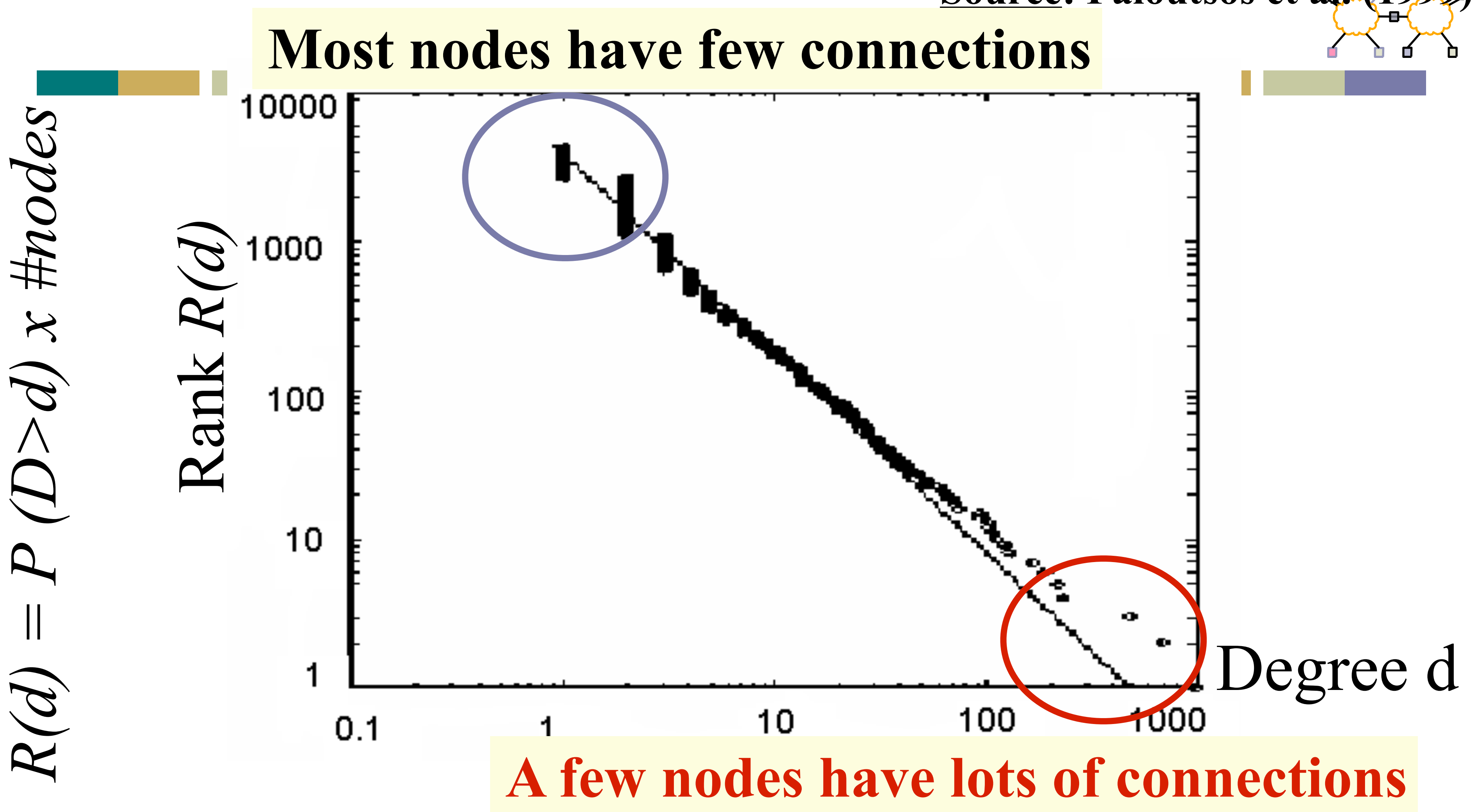
- Long-range links are expensive
- Real networks are not random, but have obvious hierarchy
- Internet topologies exhibit power law degree distributions (Faloutsos et al., 1999)
- Physical networks have hard technological (and economic) constraints.

Modeling Approach

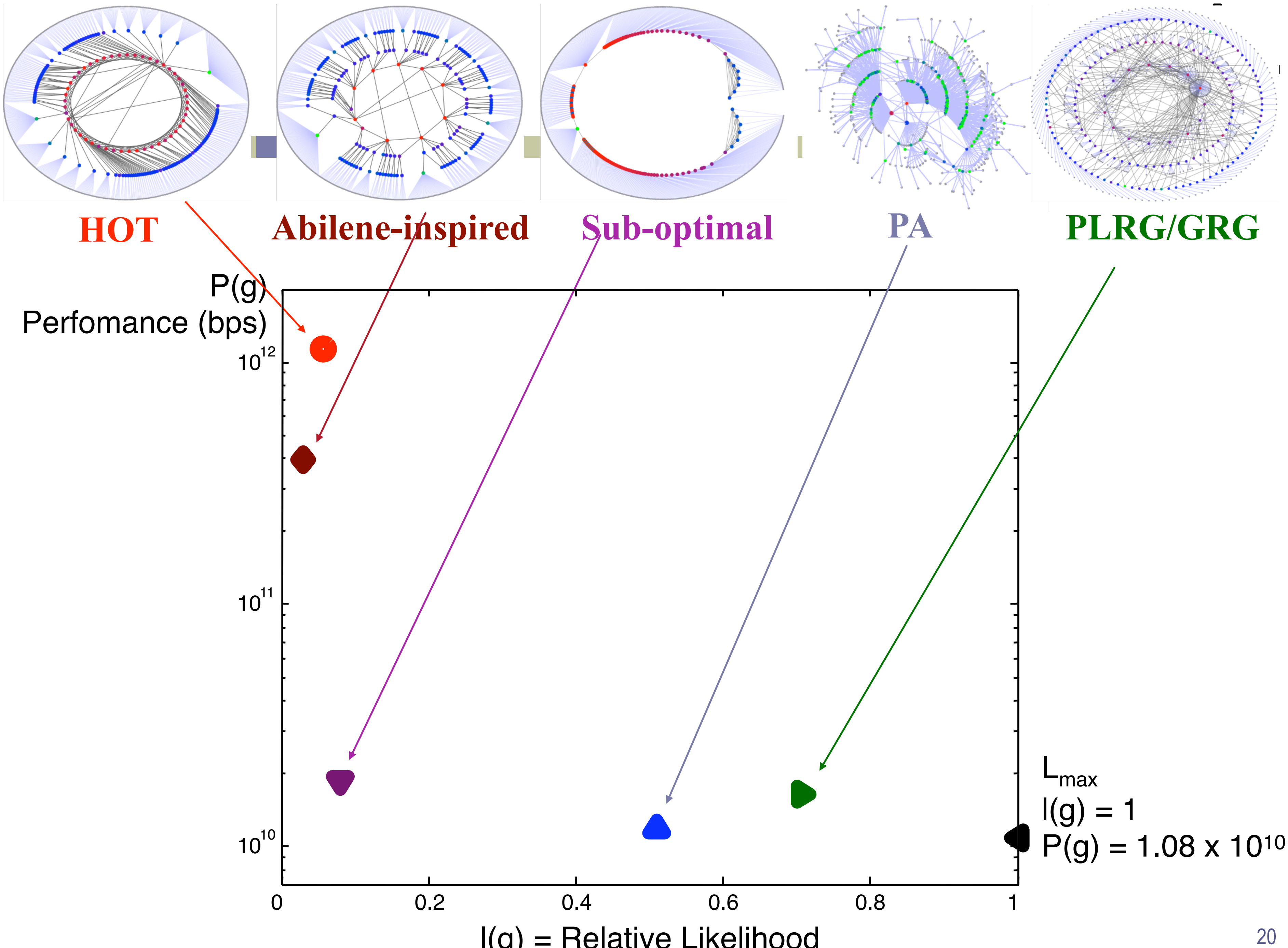
- Random graph (Waxman88)
- Structural models (GT-ITM Calvert/Zegura, 1996)
- Degree-based models replicate power-law degree sequences
- Optimization-driven models topologies consistent with design tradeoffs of network engineers

Power Laws and Internet Topology

<u>Source</u>: **Faloutsos et al. (1999)**

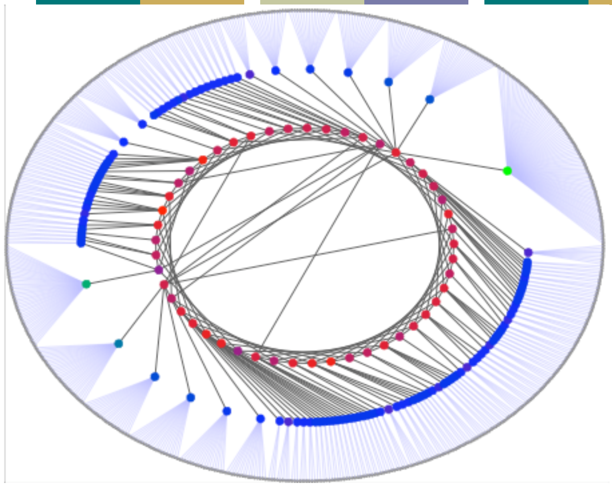
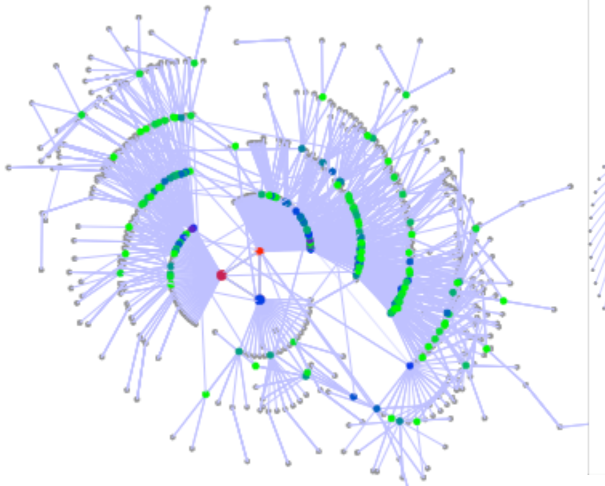
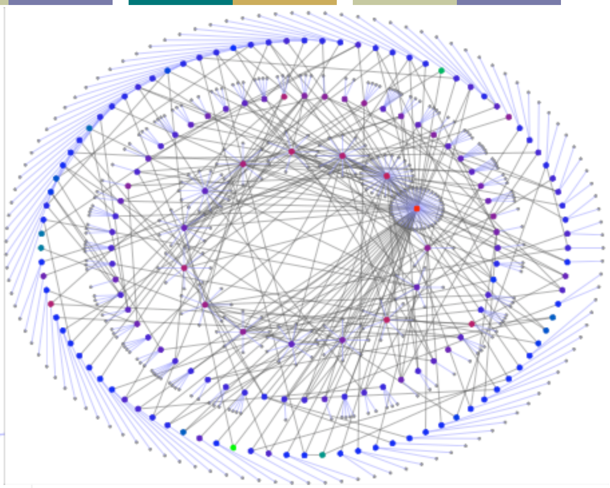


- Router-level graph & Autonomous System (AS) graph
- Led to active research in *degree-based* network models

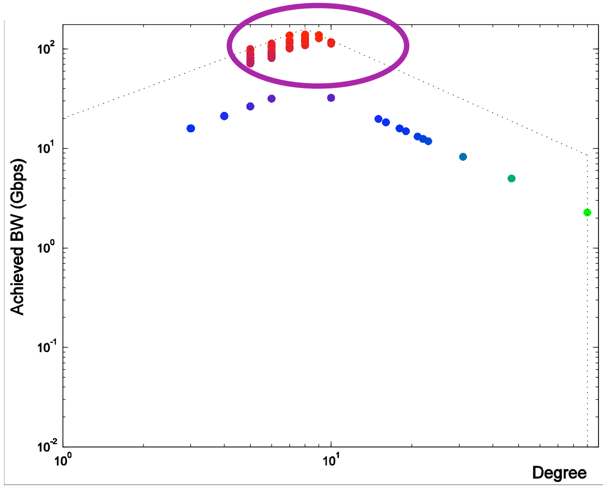

HOT
Abilene-inspired
Sub-optimal
PA
PLRG/GRG
P(g)
Perfomance (bps)
10^{12}
10^{11}
10^{10}
0
0.2
0.4
0.6
0.8
1
l(g) = Relative Likelihood
L_{max}
l(g) = 1
P(g) = 1.08 x 10^{10}

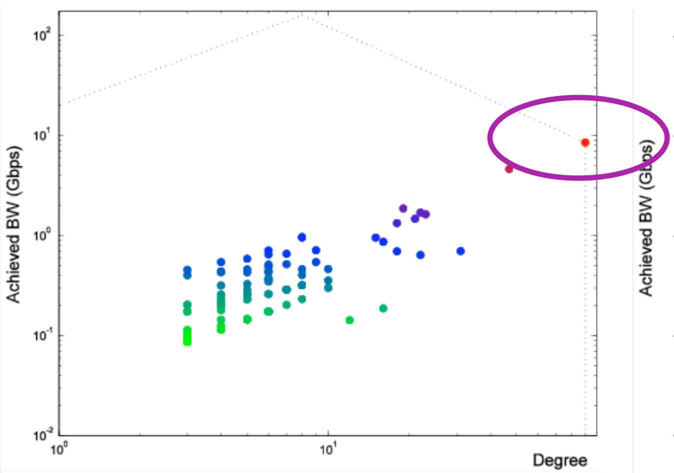
Structure Determines Performance

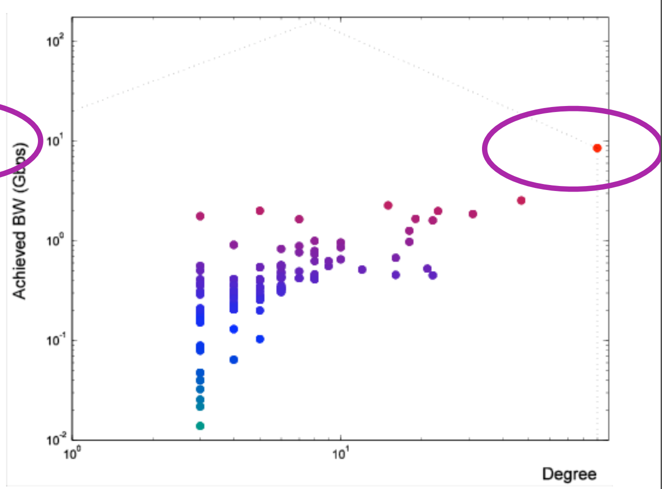
HOT | **PA** | **PLRG/GRG**

P(g) = 1.13 x 10^{12} | **P(g) = 1.19 x 10^{10}** | **P(g) = 1.64 x 10^{10}**



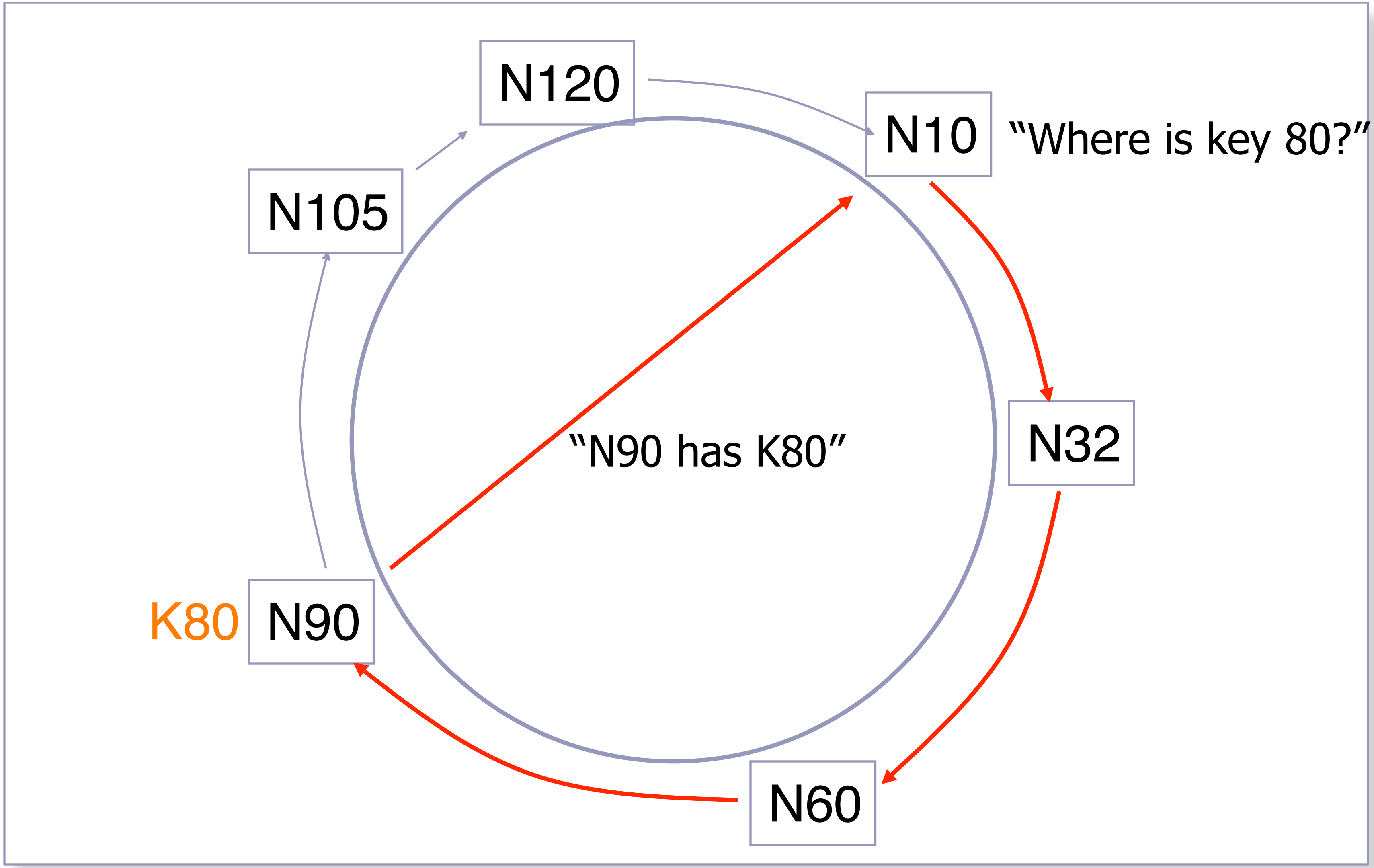




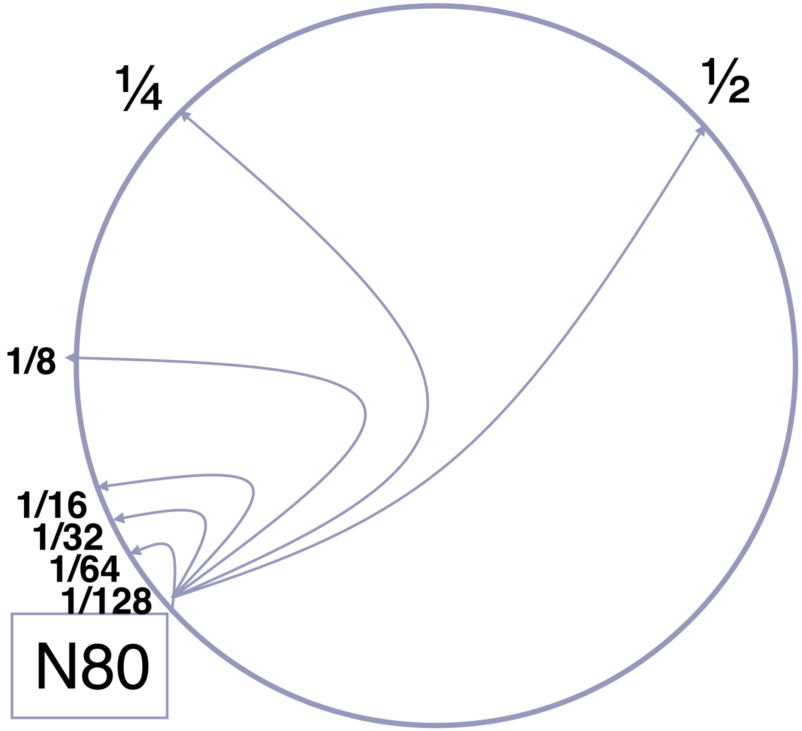
Routing: Chord

- Associate to each node and item a unique *id* in an *uni*-dimensional space
- Properties
  - Routing table size O(log(*N*)) , where *N* is the total number of nodes
  - Guarantees that a file is found in O(log(*N*)) steps

Routing: Chord Basic Lookup



Routing: Finger table - Faster Lookups



Aside: Hashing

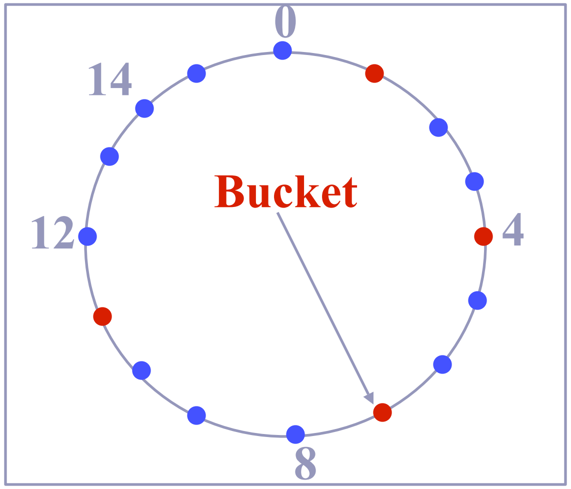
- Advantages
  - Let nodes be numbered 1..m
  - Client uses a ***good*** hash function to map a URL to 1..m
  - Say hash (url) = *x*, so, client fetches content from node *x*
  - No duplication – not being fault tolerant.
  - One hop access
  - Any problems?
    - What happens if a node goes down?
    - What happens if a node comes back up?
    - What if different nodes have different views?

Consistent Hash

- "view" = subset of all hash buckets that are visible
- Desired features
  - Balanced – in any one view, load is equal across buckets
  - Smoothness – little impact on hash bucket contents when buckets are added/removed
  - Spread – small set of hash buckets that may hold an object regardless of views
  - Load – across all views # of objects assigned to hash bucket is small

Consistent Hash – Example

- Construction
  - Assign each of C hash buckets to random points on mod 2^n circle, where, hash key size = *n.*
  - Map object to random position on circle
  - Hash of object = closest clockwise bucket



- Smoothness → addition of bucket does not cause much movement between existing buckets
- Spread & Load → small set of buckets that lie near object
- Balance → no bucket is responsible for large number of objects

Geometry’s Impact on Routing

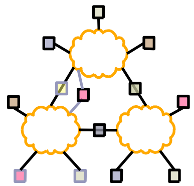
- Routing
  - Neighbor selection: how a node picks its routing entries
  - Route selection: how a node picks the next hop

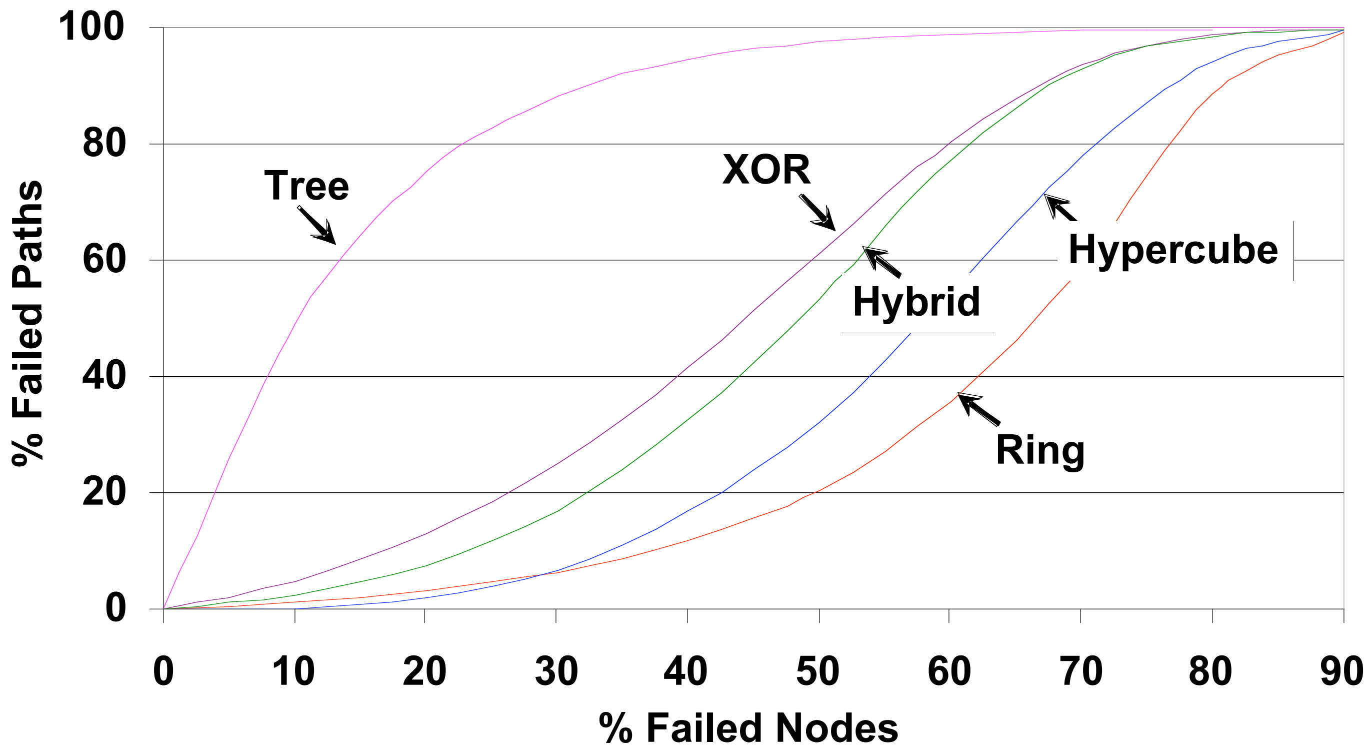
- Proposed metric: flexibility
  - amount of freedom to choose neighbors and next-hop paths
    - FNS: flexibility in neighbor selection
    - FRS: flexibility in route selection

  - intuition: captures ability to “tune” DHT performance

  - single predictor metric dependent only on routing issues

Does flexibility affect static resilience?

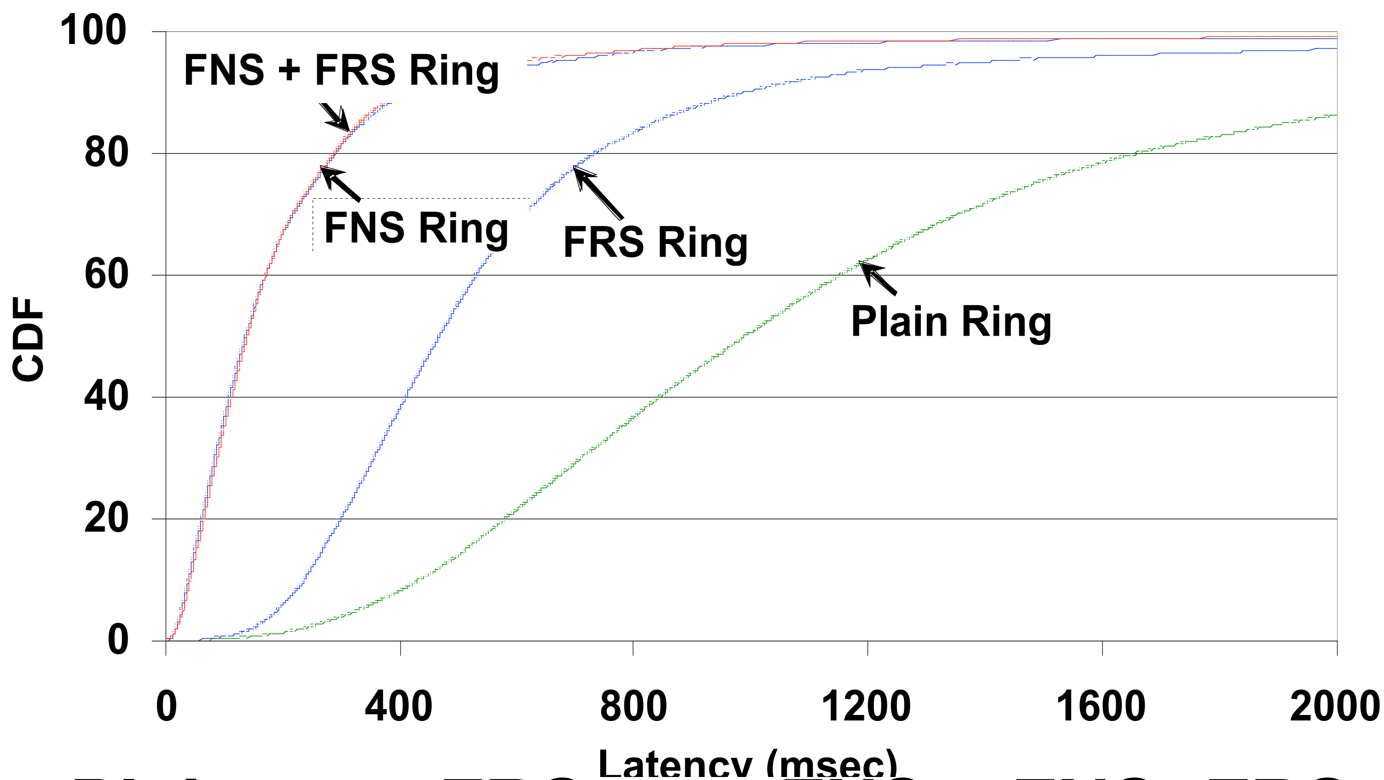




Tree << XOR ≈ Hybrid < Hypercube < Ring

Flexibility in Route Selection matters for Static Resilience

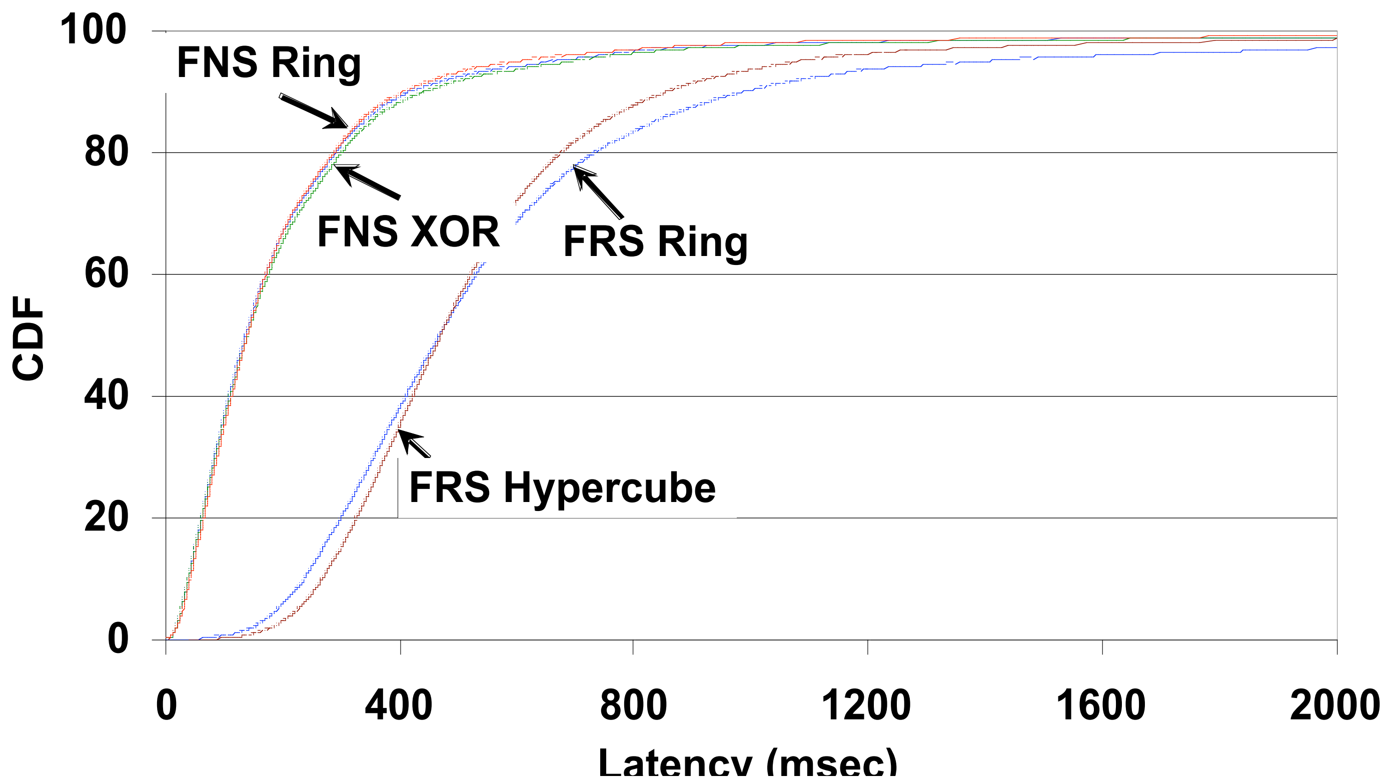
Which is more effective, FNS or FRS?



Plain << FRS << FNS ≈ FNS+FRS

Neighbor Selection is much better than Route Selection

Does Geometry affect performance of FNS or FRS?



No, performance of FNS/FRS is independent of Geometry

A Geometry's support for neighbor selection is crucial

Lookup Methods

Recursive query:

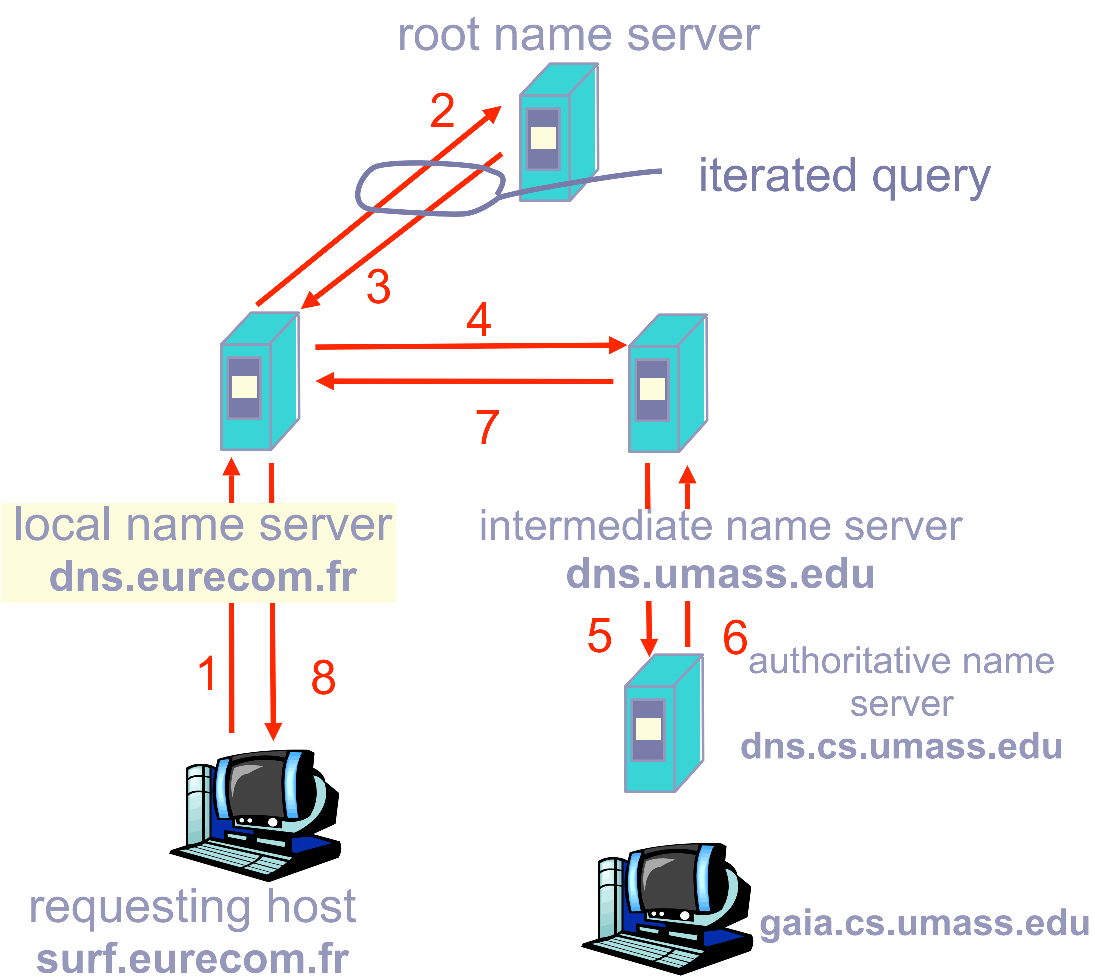
- Server goes out and searches for more info (recursive)
- Only returns final answer or "not found"

Iterative query:

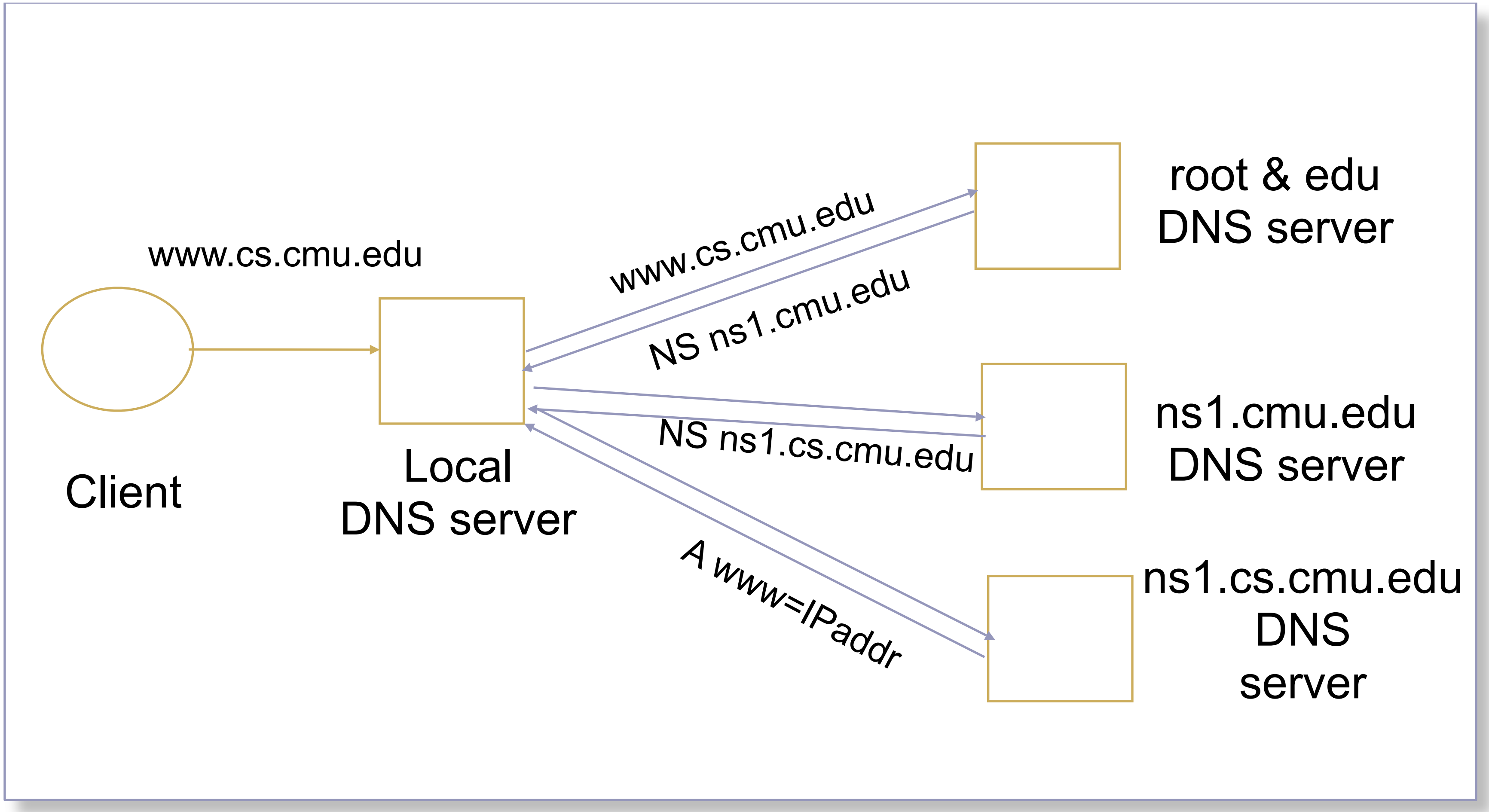
- Server responds with as much as it knows (iterative)
- "I don't know this name, but ask this server"

Workload impact on choice?

- Local server typically does recursive
- Root/distant server does iterative



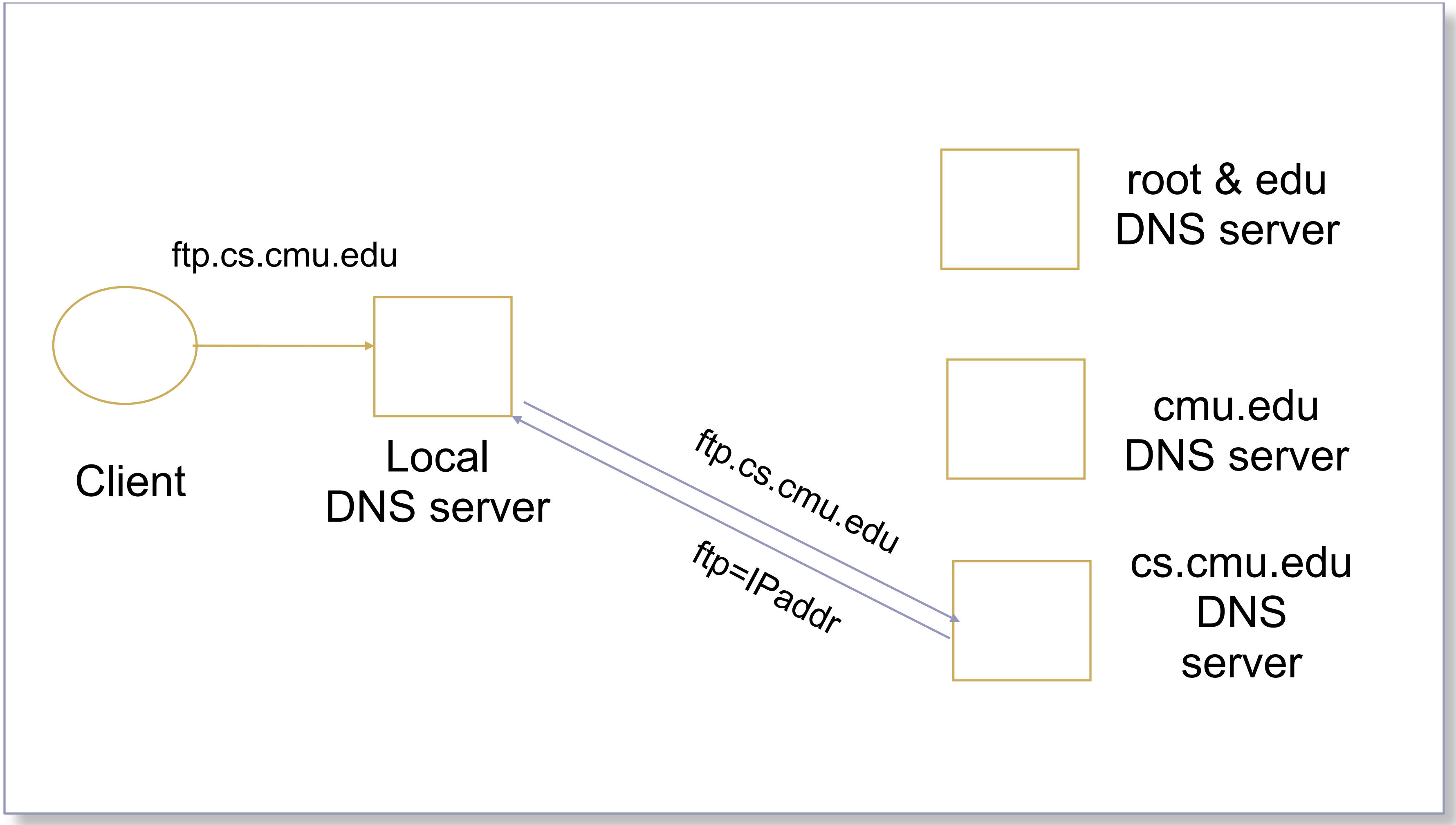
Typical Resolution



Workload and Caching

- What workload do you expect for different servers/names?
  - Why might this be a problem? How can we solve this problem?
- DNS responses are cached
  - Quick response for repeated translations
  - Other queries may reuse some parts of lookup
    - NS records for domains
- DNS negative queries are cached
  - Don’t have to repeat past mistakes
  - E.g. misspellings, search strings in resolv.conf
- Cached data periodically times out
  - Lifetime (TTL) of data controlled by owner of data
  - TTL passed with every record

Subsequent Lookup Example



DNS Experience

- 23% of lookups with no answer
  - Retransmit aggressively → most packets in trace for unanswered lookups!
  - Correct answers tend to come back quickly/with few retries
- 10 - 42% negative answers → most = no name exists
  - Inverse lookups and bogus NS records
- Worst 10% lookup latency got much worse
  - Median 85→97, 90th percentile 447→1176
- Increasing share of low TTL records → what is happening to caching?

DNS Experience

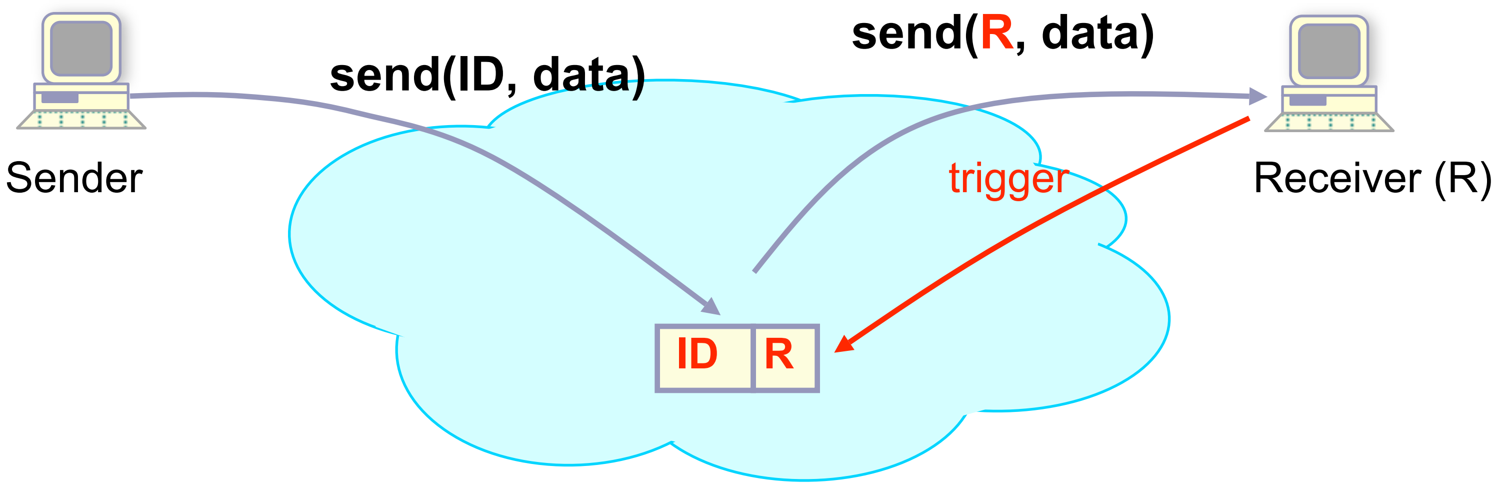
- Hit rate for DNS = 80% → 1-(#DNS/#connections)
  - Most Internet traffic is Web
  - What does a typical page look like? → average of 4-5 imbedded objects → needs 4-5 transfers → accounts for 80% hit rate!
- 70% hit rate for NS records → i.e. don't go to root/gTLD servers
  - NS TTLs are much longer than A TTLs
  - NS record caching is much more important to scalability
- Name distribution = Zipf-like = $1/x^a$
- A records → TTLs = 10 minutes similar to TTLs = infinite
- 10 client hit rate = 1000+ client hit rate

How Akamai Works

- Root server gives NS record for akamai.net
- Akamai.net name server returns NS record for g.akamaitech.net
  - Name server chosen to be in region of client's name server
  - TTL is large
- G.akamaitech.net nameserver choses server in region
  - Should try to chose server that has file in cache - How to choose?
  - Uses aXYZ name and consistent hash
  - TTL is small

i3: Rendezvous Communication

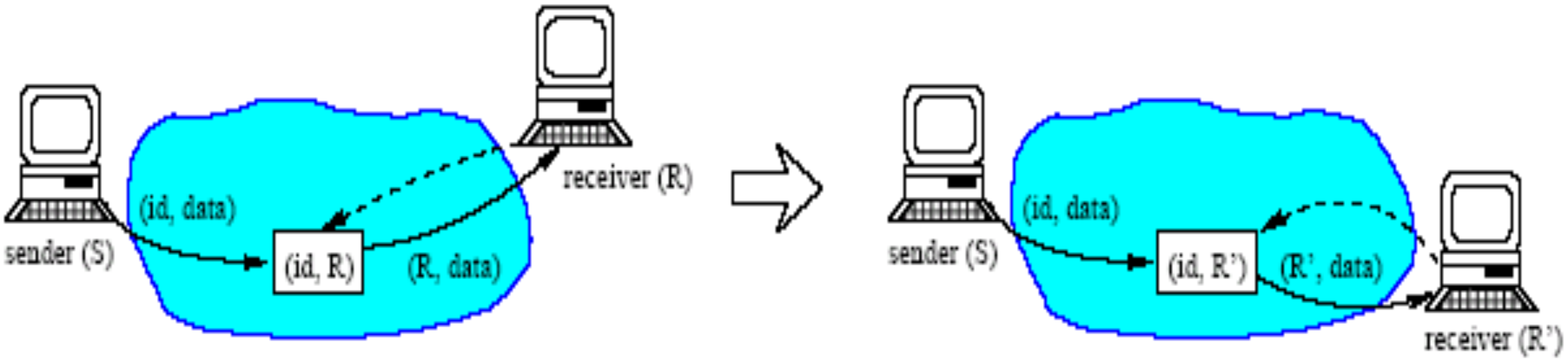
- Packets addressed to identifiers (“names”)
- Trigger=(Identifier, IP address): inserted by receiver



Senders *decoupled* from receivers

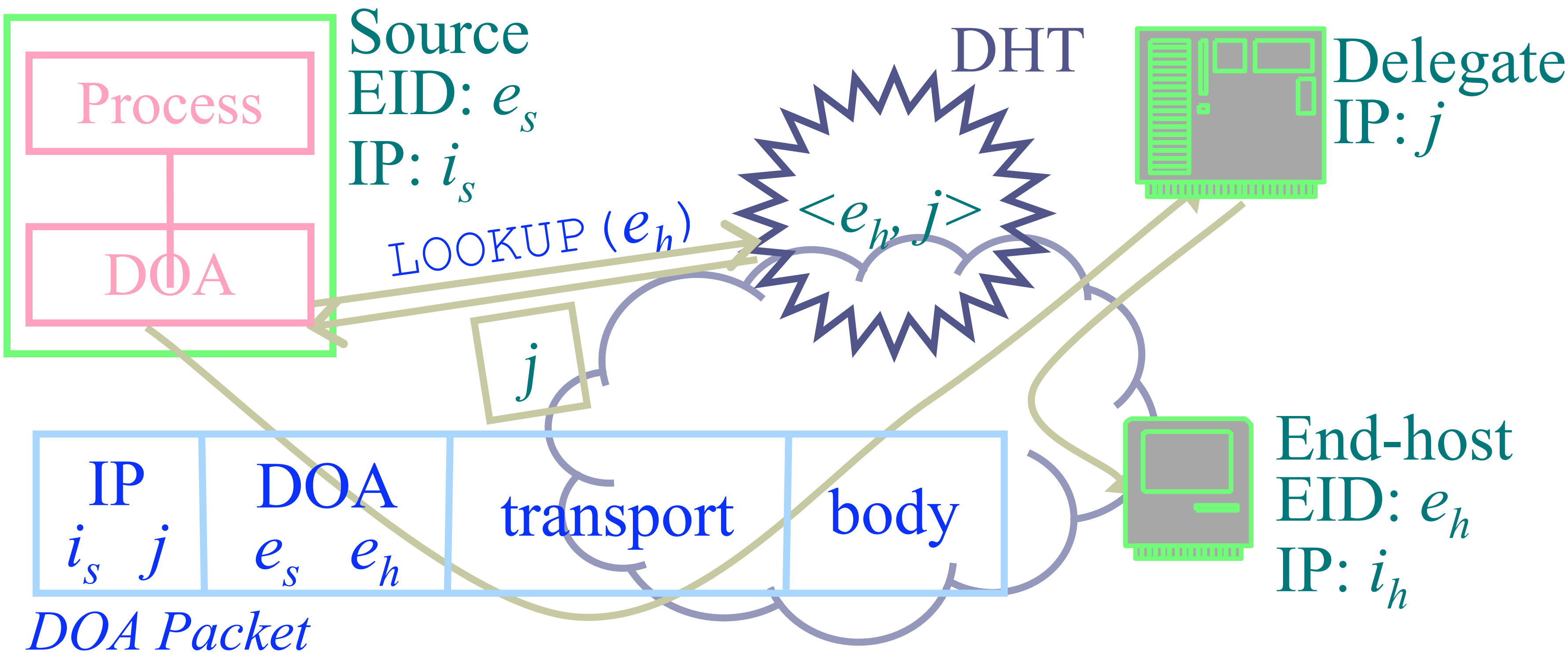
Mobility

- The change of the receiver's address
- from R to R' is transparent to the sender



(a) Mobility

DOA in a Nutshell



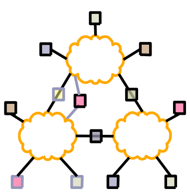
- End-host replies to source by resolving e_s
- Authenticity, performance: discussed in the paper

A Bit More About DOA

- Incrementally deployable. Requires:
  - Changes to hosts and middleboxes
  - No changes to IP routers (design requirement)
  - Global resolution infrastructure for flat IDs
- Recall core properties:
  - Topology-independent, globally unique identifiers
  - Let end-hosts invoke and revoke middleboxes
- Recall goals: reduce harmful effects, permit new functions

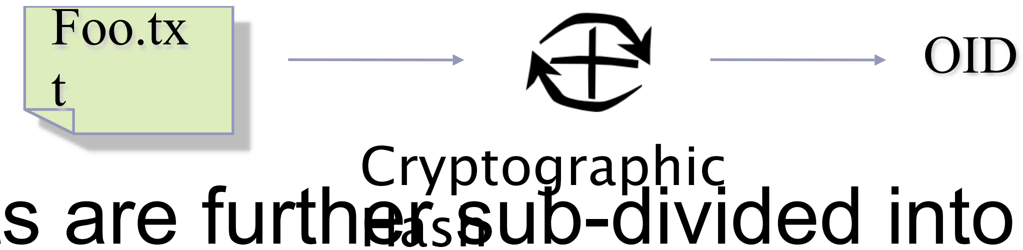
15-744: Computer Networking

L-20 Data-Oriented Networking

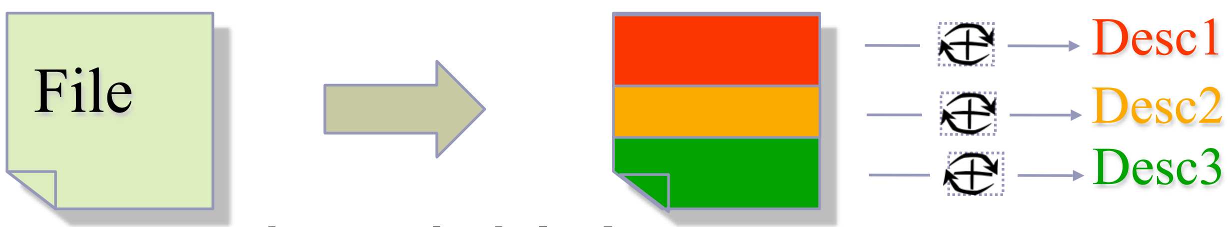


Naming Data (DOT)

- Application defined names are not portable
- Use content-naming for globally unique names
- Objects represented by an OID



- Objects are further sub-divided into “chunks”

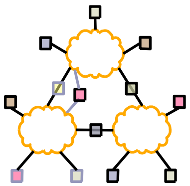


- Secure and scalable!

Naming Data (DONA)

- Names organized around principals.
- Names are of the form P : L.
  - P is cryptographic hash of principal's public key, and
  - L is a unique label chosen by the principal.
- Granularity of naming left up to principals.
- Names are "flat".

Self-certifying Names



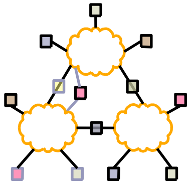
- A piece of data comes with a public key and a signature.
- Client can verify the data did come from the principal by
  - Checking the public key hashes into P, and
  - Validating that the signature corresponds to the public key.
- Challenge is to resolve the flat names into a location.

Naming Data (DTN)

- Endpoint IDs are processed as names
  - refer to one or more DTN nodes
  - expressed as Internet URI, matched as strings
- URIs
  - Internet standard naming scheme [RFC3986]
  - Format: <scheme> : <SSP>
- SSP can be arbitrary, based on (various) *schemes*
- More flexible than DOT/DONA design but less secure/scalable

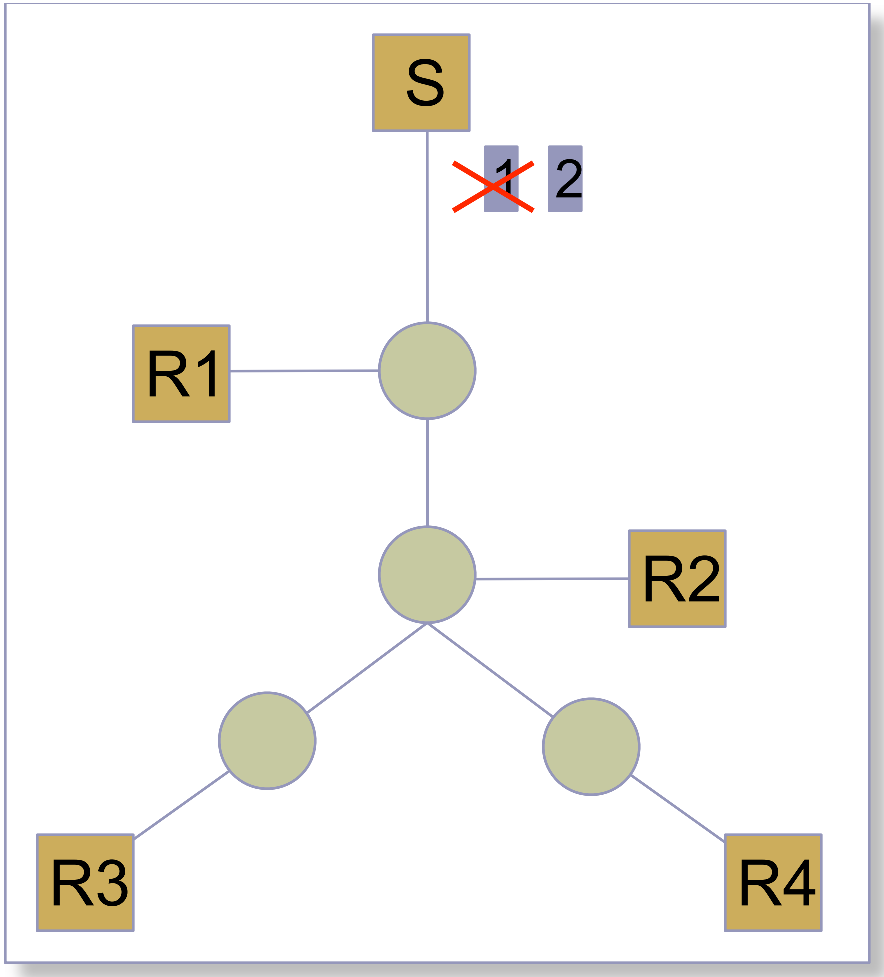
15-744: Computer Networking

L-20 Multicast

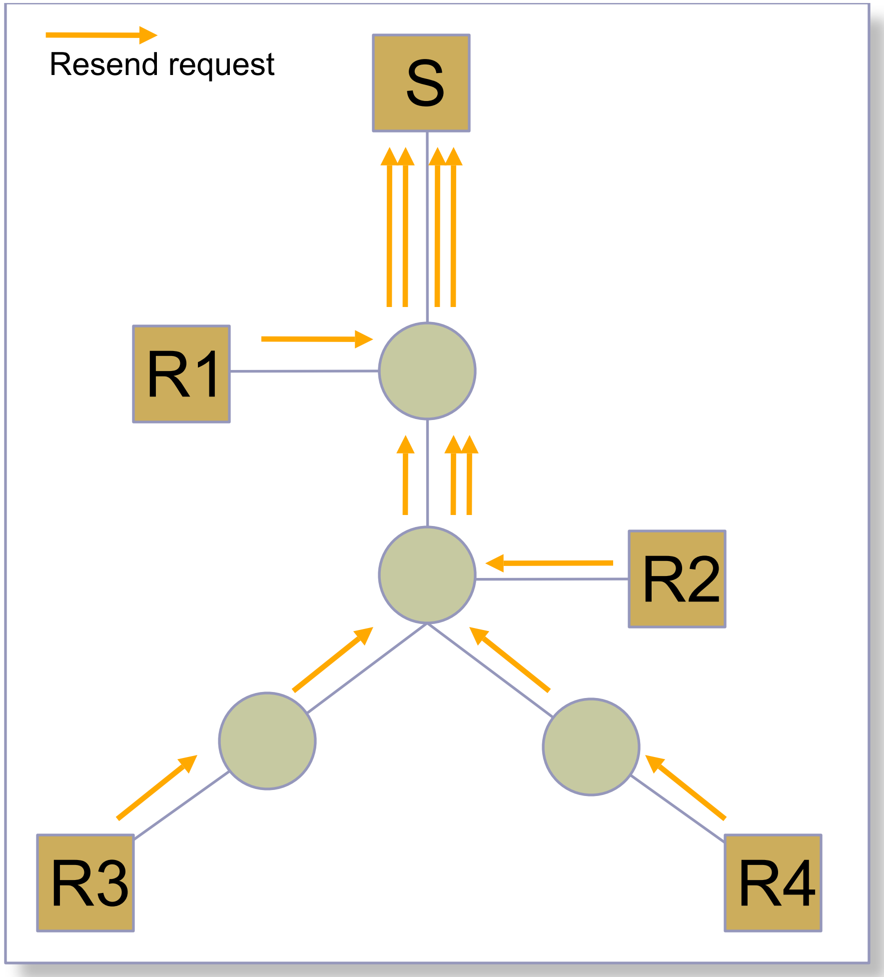


Implosion

Packet 1 is lost

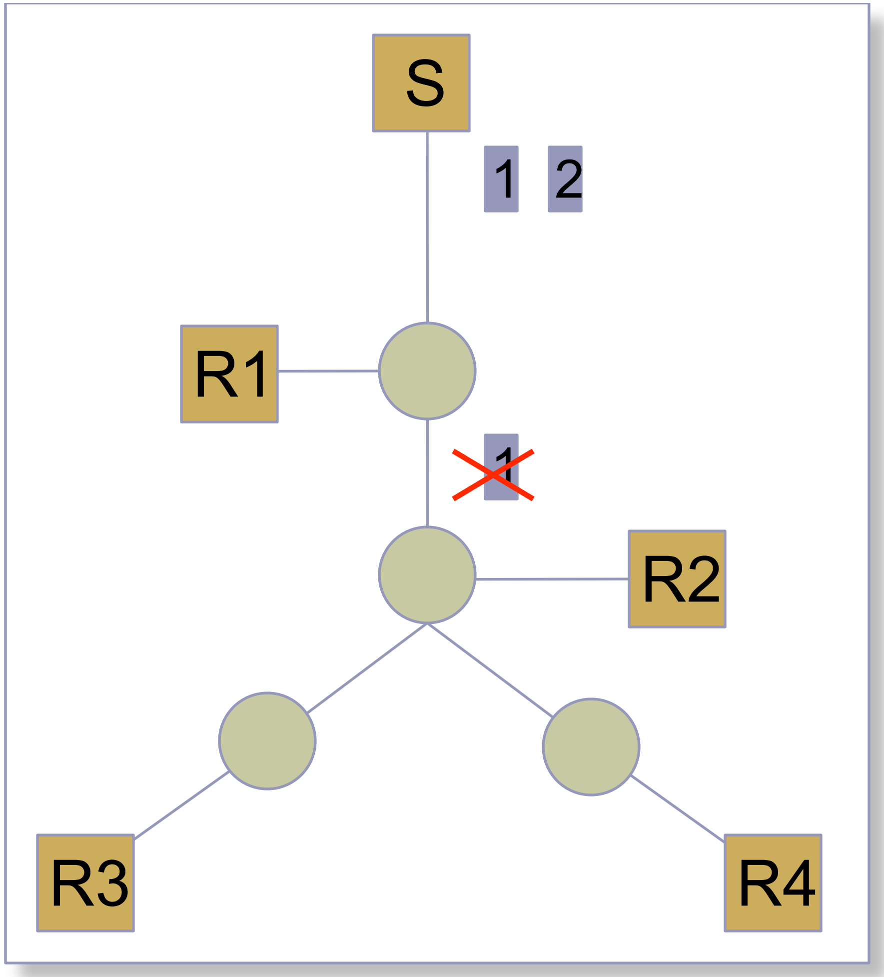


All 4 receivers request a resend

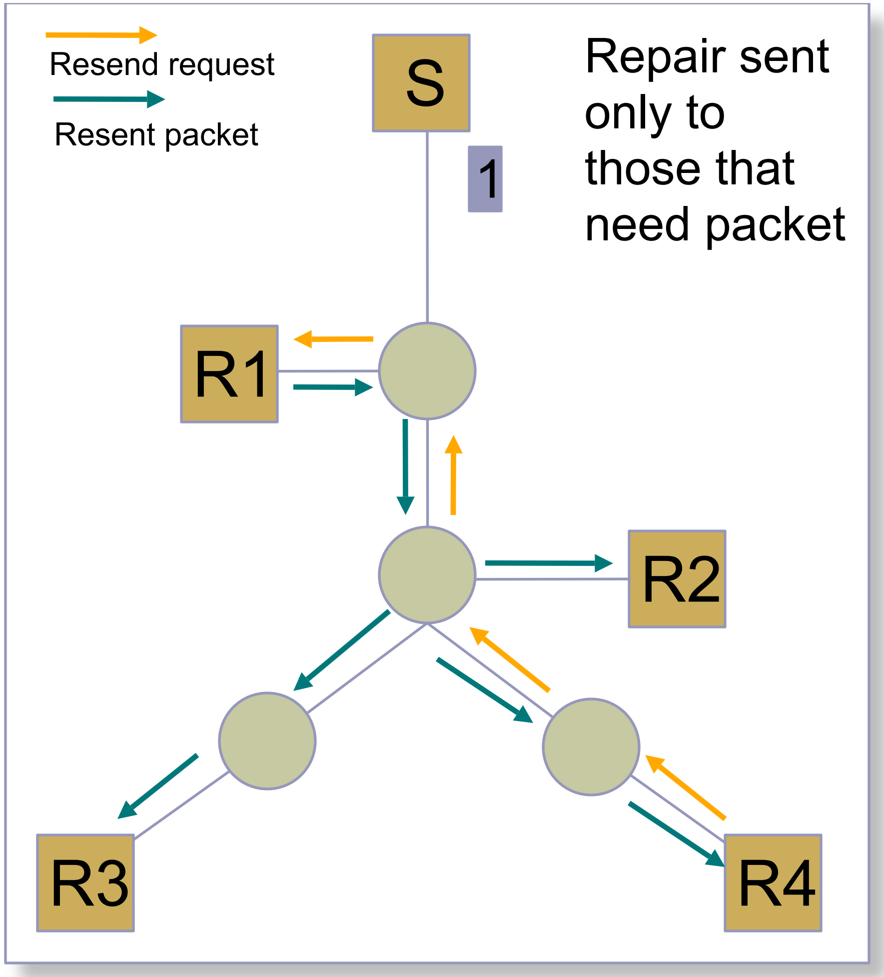


Ideal Recovery Model

Packet 1 reaches R1 but is lost before reaching other Receivers

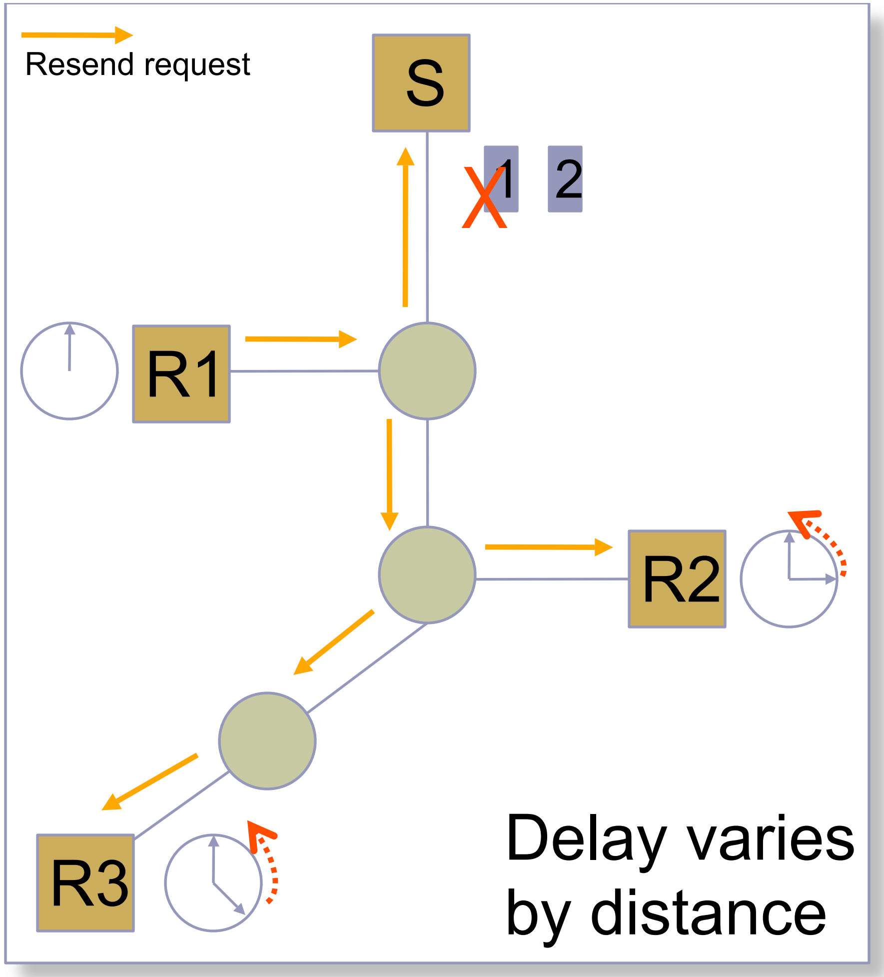


Only one receiver sends NACK to the nearest S or R with packet

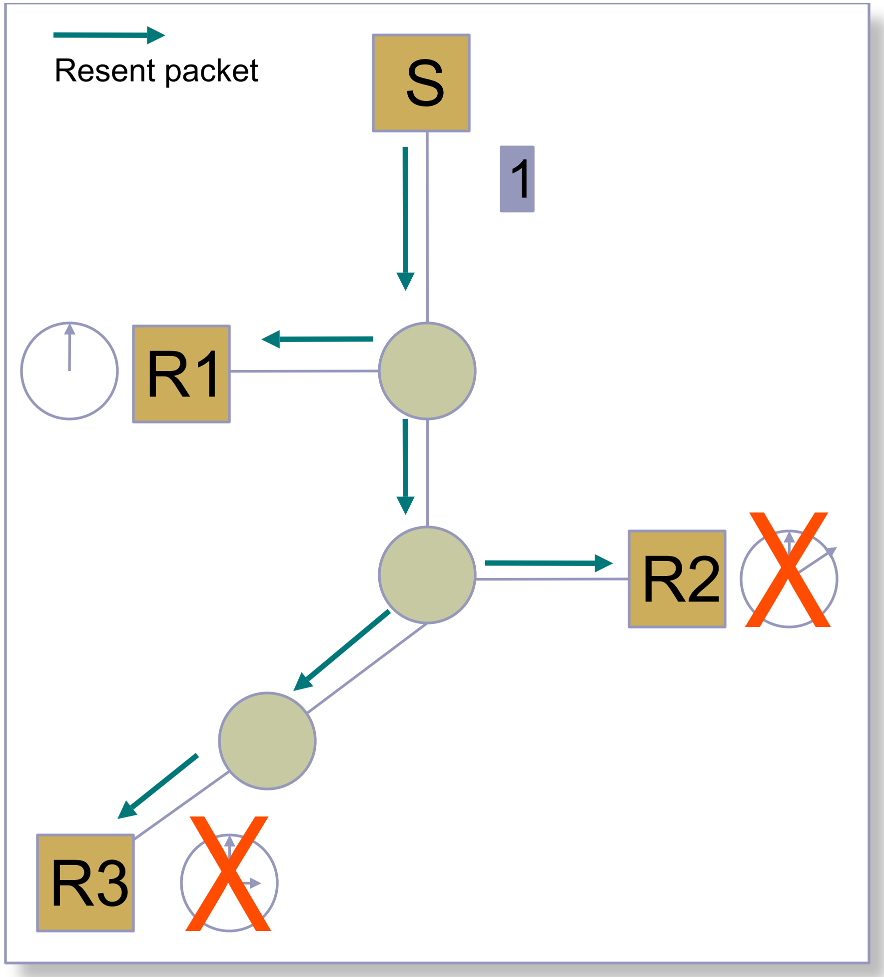


SRM Request Suppression

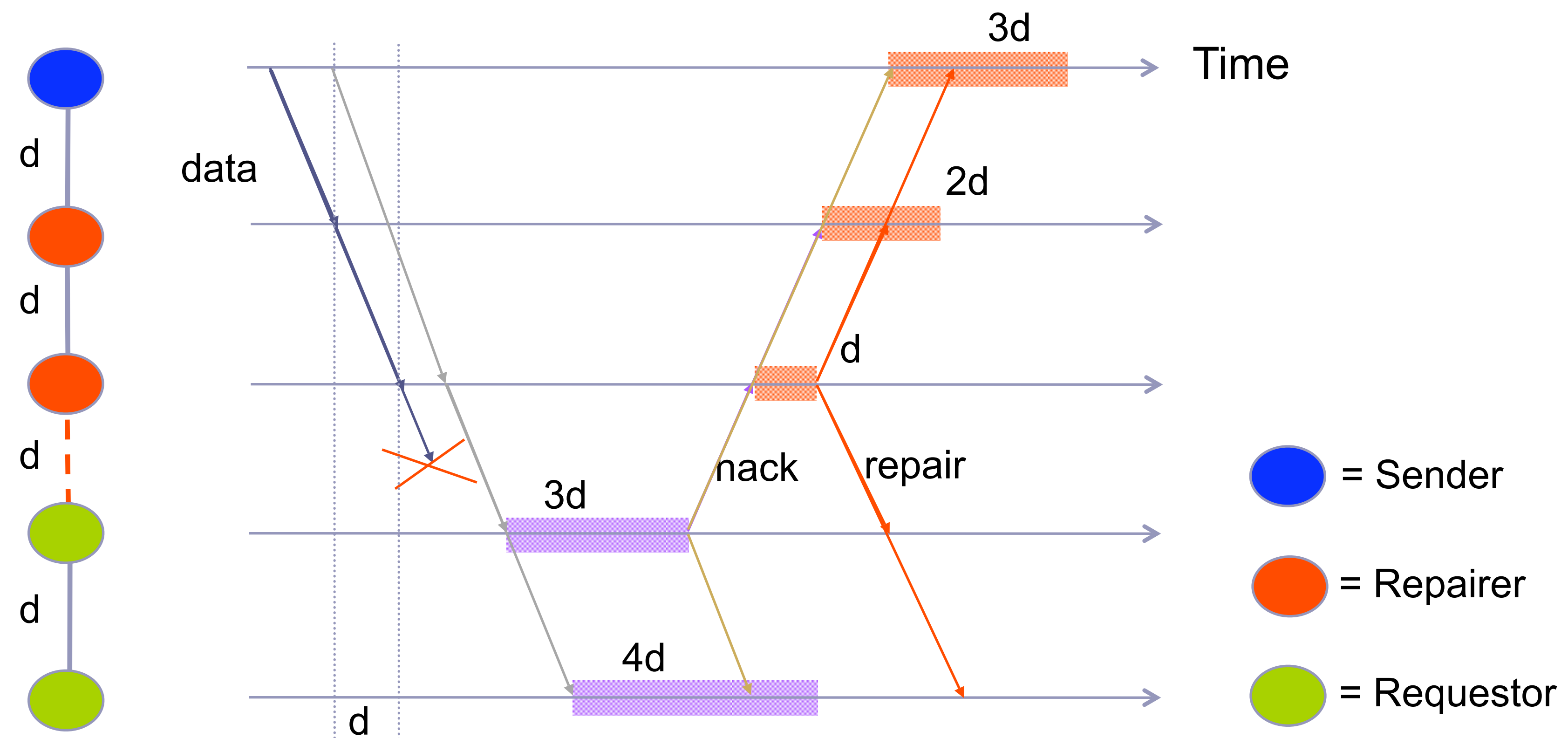
Packet 1 is lost; R1 requests resend to Source and Receivers



Packet 1 is resent; R2 and R3 no longer have to request a resend



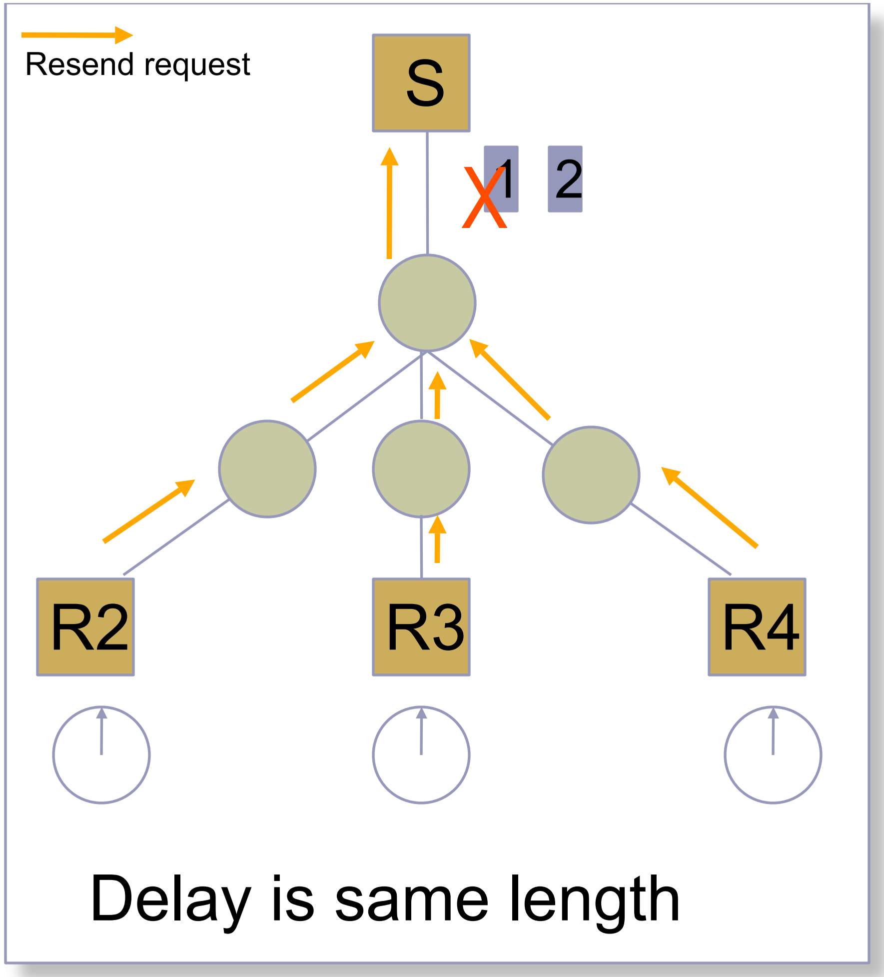
Deterministic Suppression



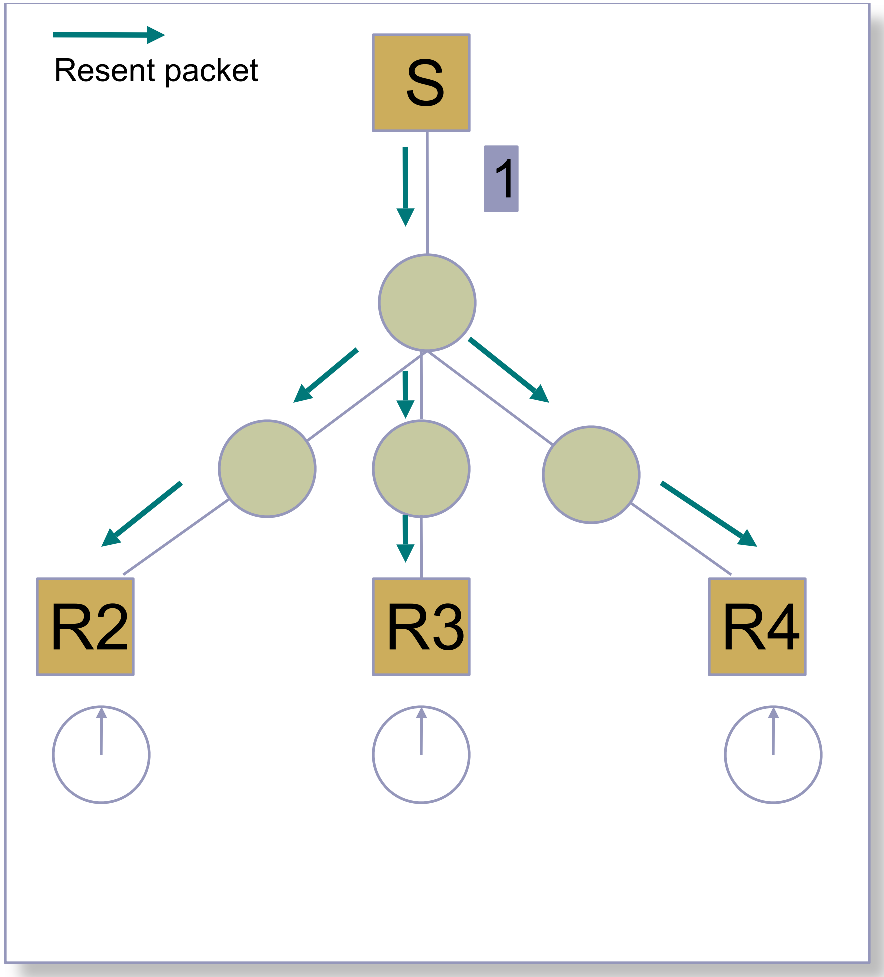
$$\text{Delay} = C_1 \times d_{S,R}$$

SRM Star Topology

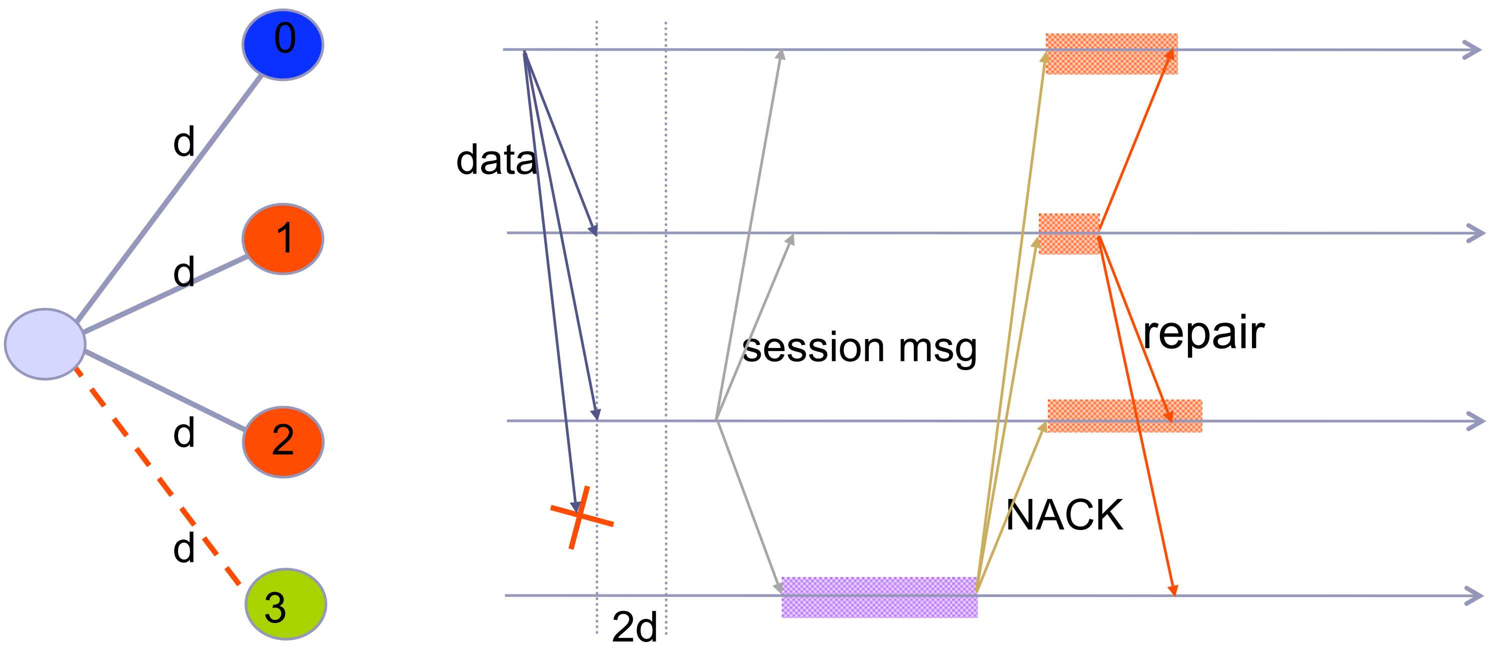
Packet 1 is lost; All Receivers request resends



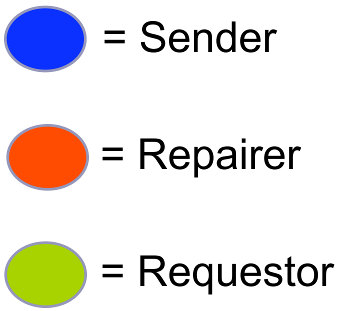
Packet 1 is resent to all Receivers



SRM: Stochastic Suppression



$$\text{Delay} = U[0,D_2] \times d_{S,R}$$



SRM (Summary)

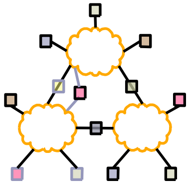
- NACK/Retransmission suppression
  - Delay before sending
  - Delay based on RTT estimation
  - Deterministic + Stochastic components
- Periodic session messages
  - Full reliability
  - Estimation of distance matrix among members

Routing Techniques

- Flood and prune
  - Begin by flooding traffic to entire network
  - Prune branches with no receivers
  - Examples: DVMRP, PIM-DM
  - *Unwanted state where there are no receivers*
- Link-state multicast protocols
  - Routers advertise groups for which they have receivers to entire network
  - Compute trees on demand
  - Example: MOSPF
  - *Unwanted state where there are no senders*

15-744: Computer Networking

L-22 Security and DoS

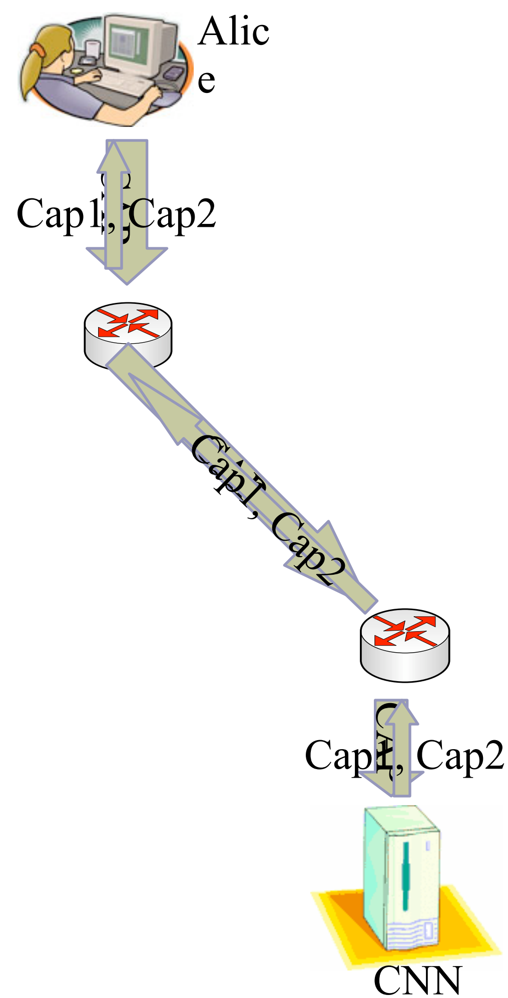


TVA (Capability)

Capability =

timestamp || Hash (N, T, PreCap)

- *N bytes, T seconds*
- Stateless receiver
  - Does not store *N, T*



Balancing Authorized Traffic

- It is quite possible for a compromised insider to allow packet floods from outside
- A fair-queuing policy is implemented and the bandwidth is decreased as the network becomes busier
- To limit the number of queues, a bounded policy is used which only queues those flows that send faster than N/T
- Other senders are limited by FIFO service

The Need for Traceback

- Internet hosts are vulnerable
  - Many attacks consist of *very few packets*
  - Fraggle, Teardrop, ping-of-death, etc.
- Internet Protocol permits anonymity
  - Attackers can “spoof” source address
  - IP forwarding maintains no audit trails
- Need a separate *traceback* facility
  - For a given packet, find the path to *source*

Approaches to Traceback

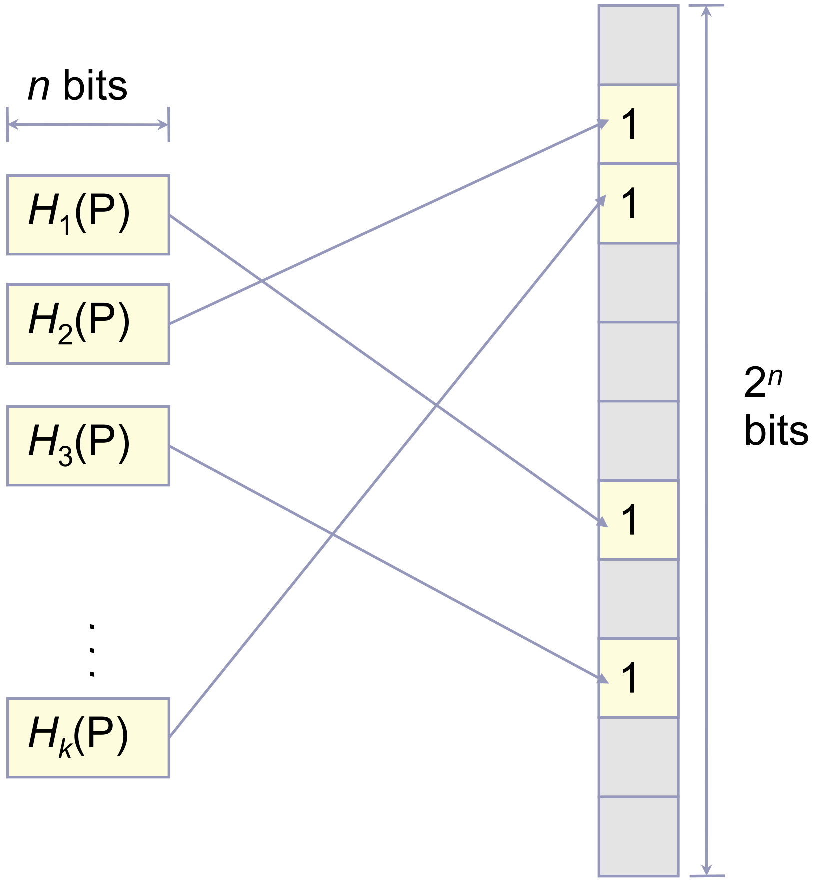
- Path data can be noted in several places
  - In the packet itself [Savage et al.],
  - At the destination [I-Trace], or
  - In the network infrastructure
- Logging: a naïve in-network approach
  - Record each packet forwarding event
  - Can trace a single packet to a source router, ingress point, or subverted router(s)

Solution: Packet Digesting

- Record only invariant packet content
  - Mask dynamic fields (TTL, checksum, etc.)
  - Store information required to invert packet transformations at performing router
- Compute *packet digests* instead
  - Use hash function to compute small digest
  - Store probabilistically in Bloom filters
- Impossible to retrieve stored packets

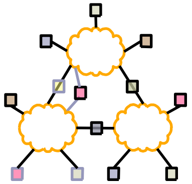
Bloom Filters

- Fixed structure size
  - Uses 2n bit array
  - Initialized to zeros
- Insertion is easy
  - Use n-bit digest as indices into bit array
  - Mitigate collisions by using multiple digests
- Variable capacity
  - Easy to adjust
  - Page when full



15-744: Computer Networking

L-23 Worms



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

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

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

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

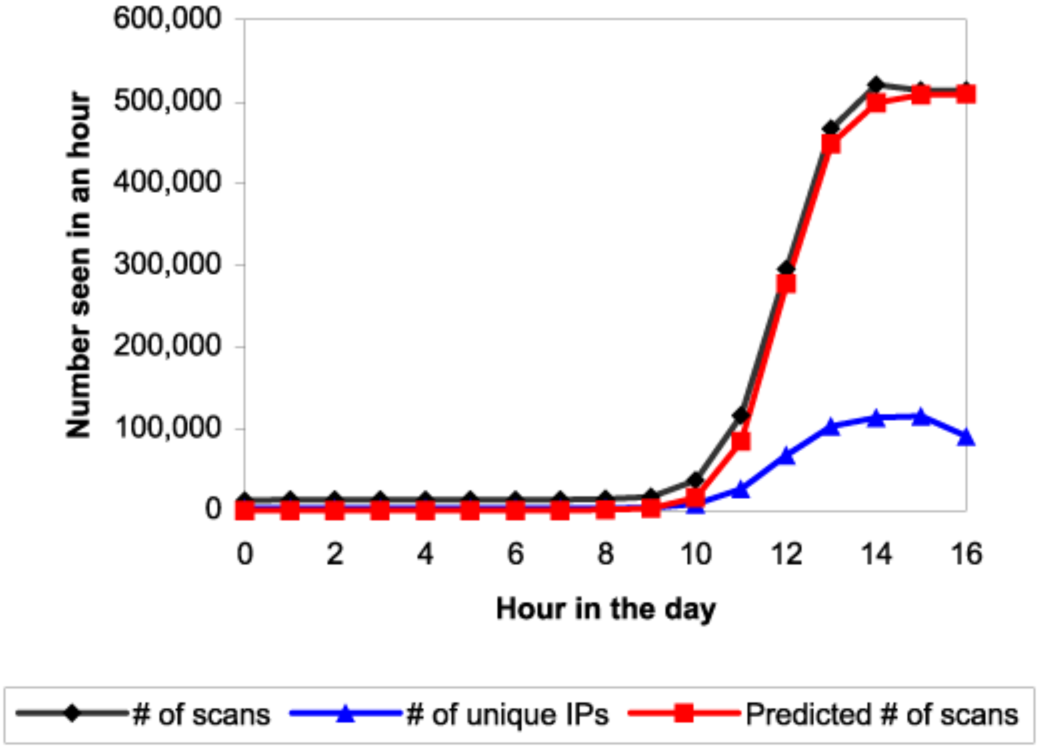
$$a = \frac{e^{K(t-T)}}{1+e^{K(t-T)}},$$

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

"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

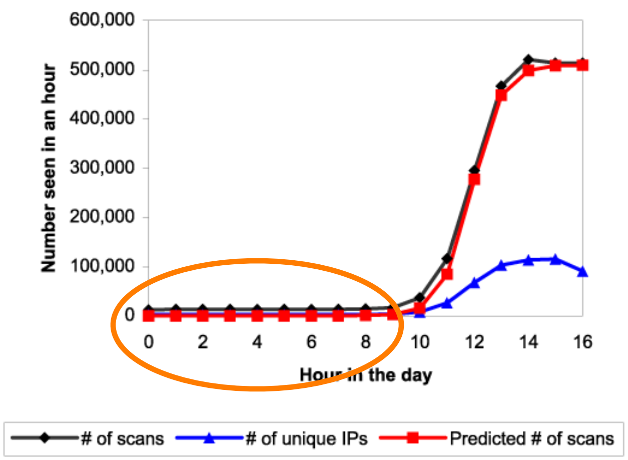


- K = 1.8
- T = 11.9

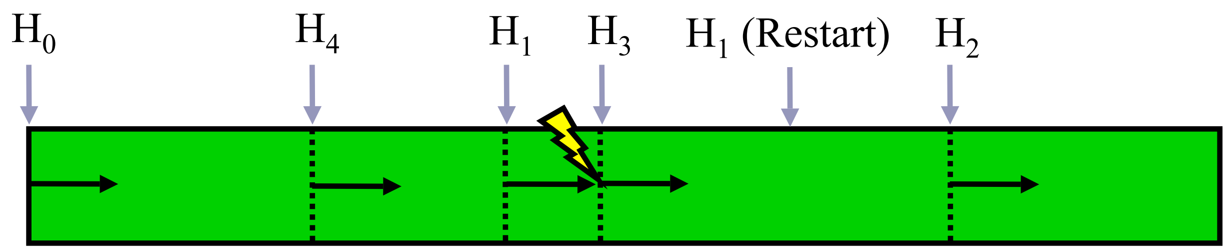
Hourly probe rate data for inbound port 80 at the Chemical Abstracts Service during the initial outbreak of Code Red I on July 19th, 2001.

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



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

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

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

Comparison

<table>
<tr><th>Earlybird</th><th>Autograph</th></tr>
<tr><td colspan="2">Infect the system with Network Data (real traces)</td></tr>
<tr><td colspan="2">Rabin fingerprint</td></tr>
<tr><td colspan="2">White-list/blacklist</td></tr>
<tr><td>No-prefiltering</td><td>Flow-reassembly</td></tr>
<tr><td>Single sensor algorithmics + centralized aggregators</td><td>Distributed Deployment + active cooperation between multiple sensors</td></tr>
<tr><td>On-line</td><td>Off-line</td></tr>
<tr><td>Overlapping, fixed-length chunks</td><td>Non-overlapping, variable-length chunks</td></tr>
</table>