Distributed Quota Enforcement for Spam Control

Michael Walfish, J.D. Zamfirescu,
Hari Balakrishnan, David Karger, and Scott Shenker*

MIT Computer Science and AI Lab
** UC Berkeley and ICSI*

09 May 2006

"...and you spent 5.73 years of your life deleting spam from your e-mail."

Email Has Gone from Unusable to Unreliable

- Culprit: spam (unsolicited bulk email)
  - Inboxes flooded → *email unusable*
  - 50-70% of email today is spam [MessageLabs]
- Common solution: filters (examine email text)
  - Reject, e.g., mortg@ges, Viagra, v!@gr@, Nigeria

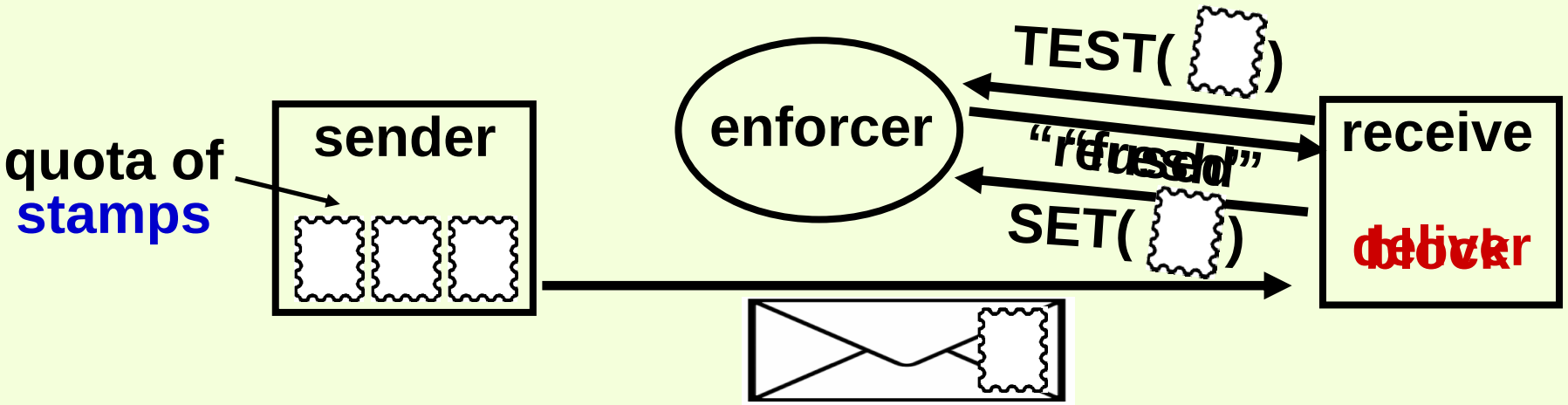
"Our anti-spam software deleted your report because the flow chart was shaped sort of like a Nigerian prince."

Email Has Gone from Unusable to Unreliable

- Culprit: spam (unsolicited bulk email)
  - Inboxes flooded → *email unusable*
  - 50-70% of email today is spam [MessageLabs]
- Common solution: filters (examine email text)
  - Reject, e.g., mortg@ges, Viagra, v!@gr@, Nigeria
  - But valid email blocked → *email unreliable*
- Whitelisting (Re, Goodmail)
- Many other solutions… [FUSSP]

Our Solution: Restore Reliability w/ Quotas

- Quotas on the # of mails a sender can send
  - Limit volumes w/out semantic discrimination
  - Set to make level of spam negligible

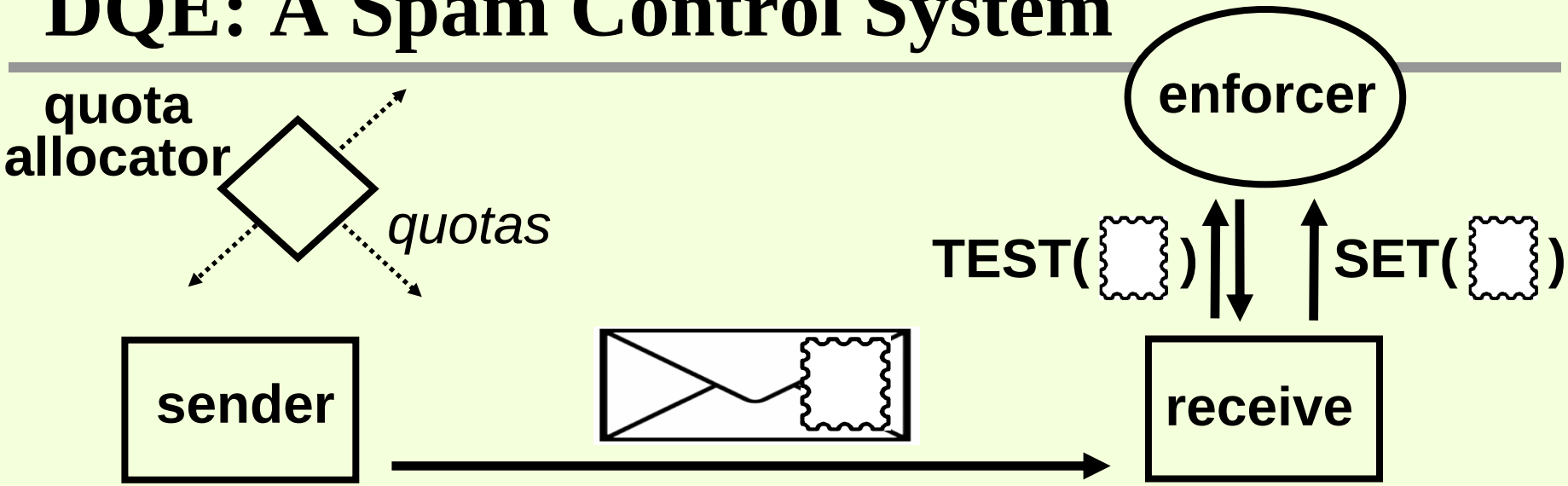


- Implement quotas with stamps
  - If stamp reused: receiver **blocks** mail (assumes spam)
  - If stamp fresh: receiver **delivers** mail and SETs stamp
- Valid email ⇔ fresh stamp → *delivered*

What this Talk Is and Is Not About

- Not about quota allocation (social/econ. problem)
  - In our system, trusted *allocators* decide policy (e.g., payment, proof of human identity, …)
  - Our system works with any policy
- Not about detailed justification for quotas
- Not about disrespecting your favorite spam solution
  - Or about the adoption paths for ours
- About the technical problems in quota enforcement
  - Minimal and fault-tolerant distributed system

DQE: A Spam Control System



Guiding Principles:

- Never label valid email as spam
- Separate allocation and enforcement
- Handle world email volume (100 B msgs/day)
- Don't trust the enforcer
  (make it distrust*ed* and distrust*ing*)

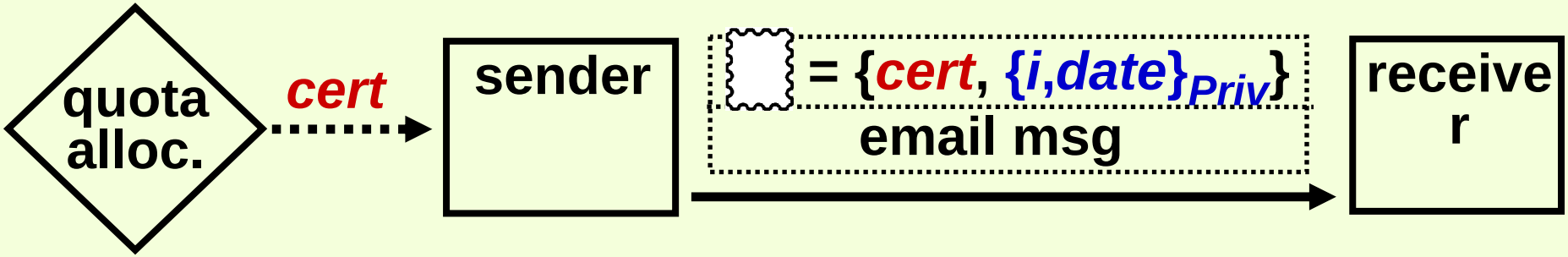
Rest of the Talk:

I. Stamps and Protocols

II. Design of the Enforcer

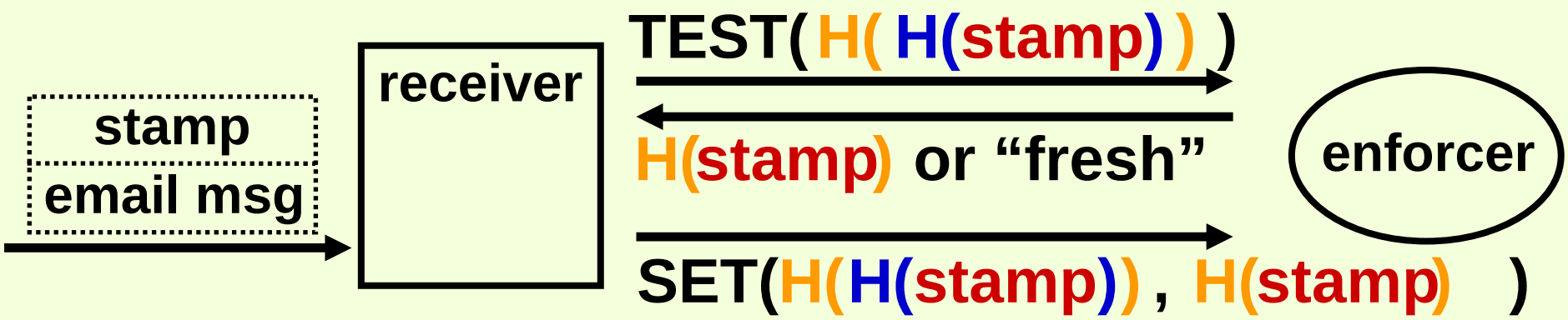
Allocation and Enforcement Protocols

- *Qpub* : globally known public key of quota allocator
- *(Priv, Pub):* sender's public-private key pair



- *cert* = {*Pub, daily_quota*}*Qpriv*
  - Tells world how many stamps sender can "mint"
- *stamp* = {*cert*, {*i,date*}*Priv*}
  - *i* must be unused; 1 ≤ *i* ≤ *daily_quota*
- Receiver must check: *msg under quota?*
  - First test: *Is stamp authentic?* Requires only *Qpub*
  - Second test: *Has stamp been used before?*

Enforcement Protocol, Continued



(H is a hard-to-invert hash function)

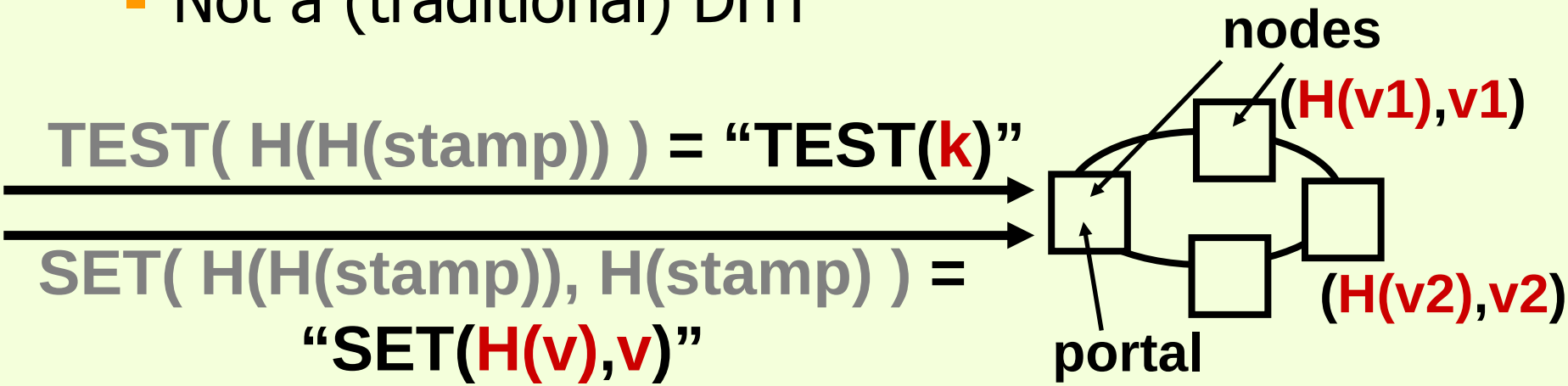
- No email with fresh stamps flagged as spam
- Protocol distrusts enforcer
- Privacy: replace stamp with H(stamp)

I. Stamps and Protocols

II. Design of the Enforcer

30,000-Foot View of the Enforcer

- Purpose: prevent too much stamp reuse
- Best-effort storage of key-value pairs
  - Not a (traditional) DHT



- Design is agnostic about where nodes come from
  - Can be one org or many, LAN or wide-area
- It is practical to build an enforcer to handle the world's e-mail

Enforcer's Distribution and Trust Model

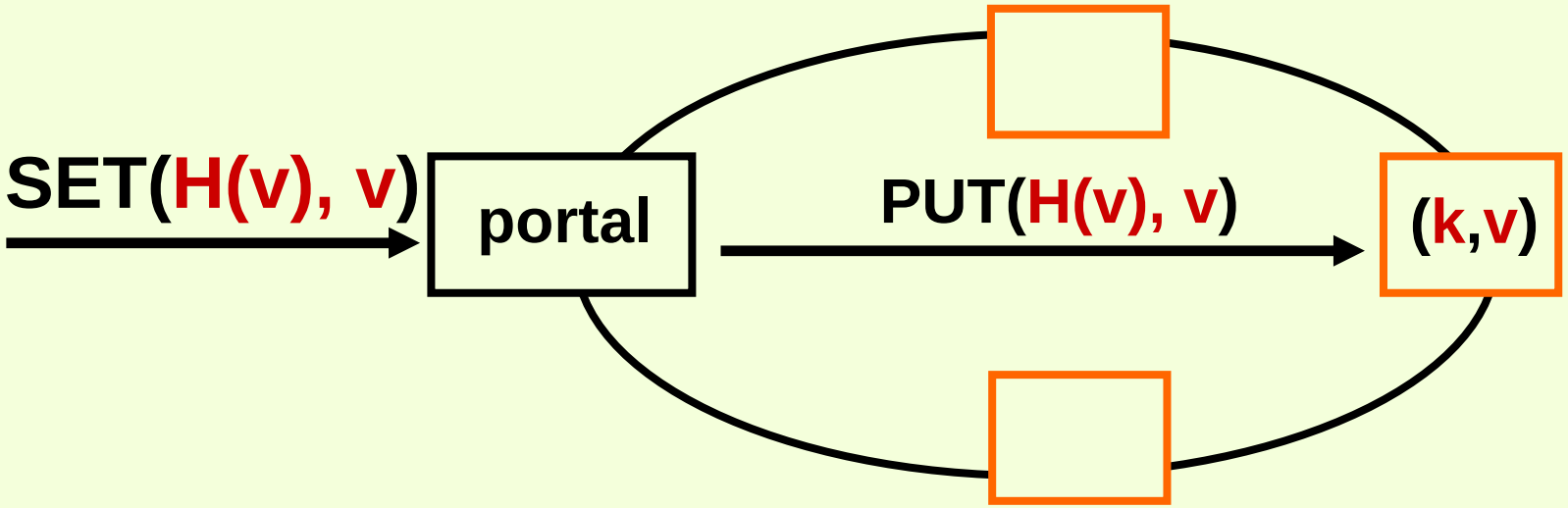
Why must the enforcer be multiple machines?

- 100 B emails/day → 1-2 M stamp checks per sec.
- 40 bytes/stamp → 4 Tbytes of storage per day

Setup: enforcer is *n* machines

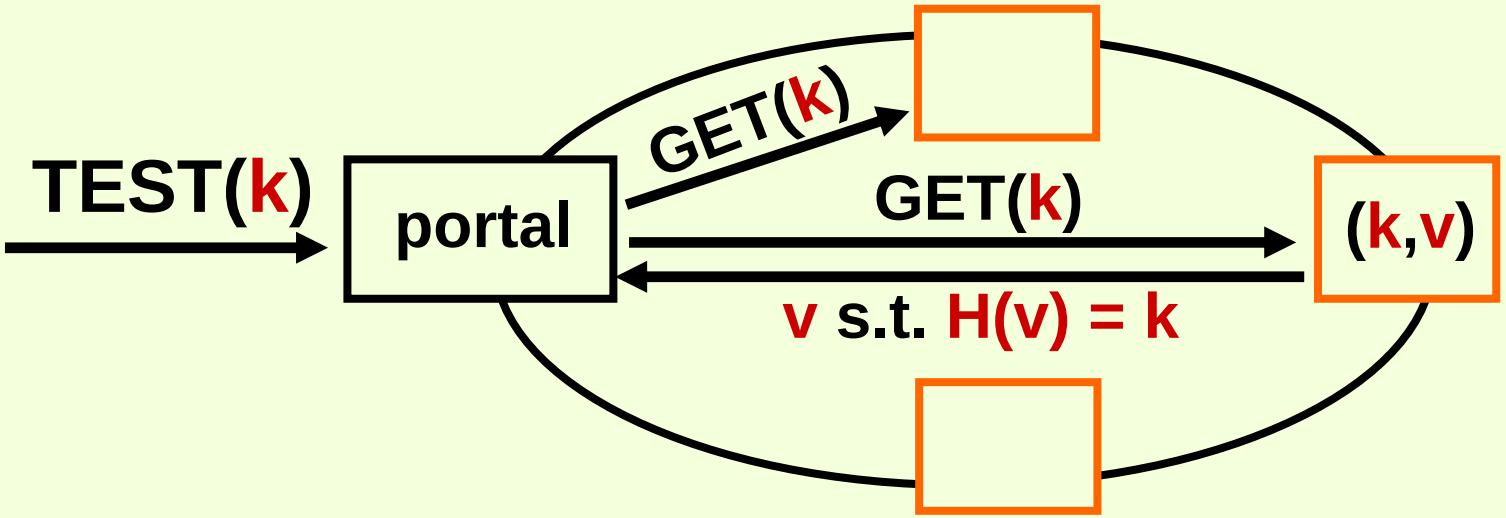
- Assume: they know each other (justified later)
  - For now, we seek to minimize required # of machines
- They do not trust each other
- They can have crash or Byzantine faults

How the Enforcer Stores Key-Value Pairs



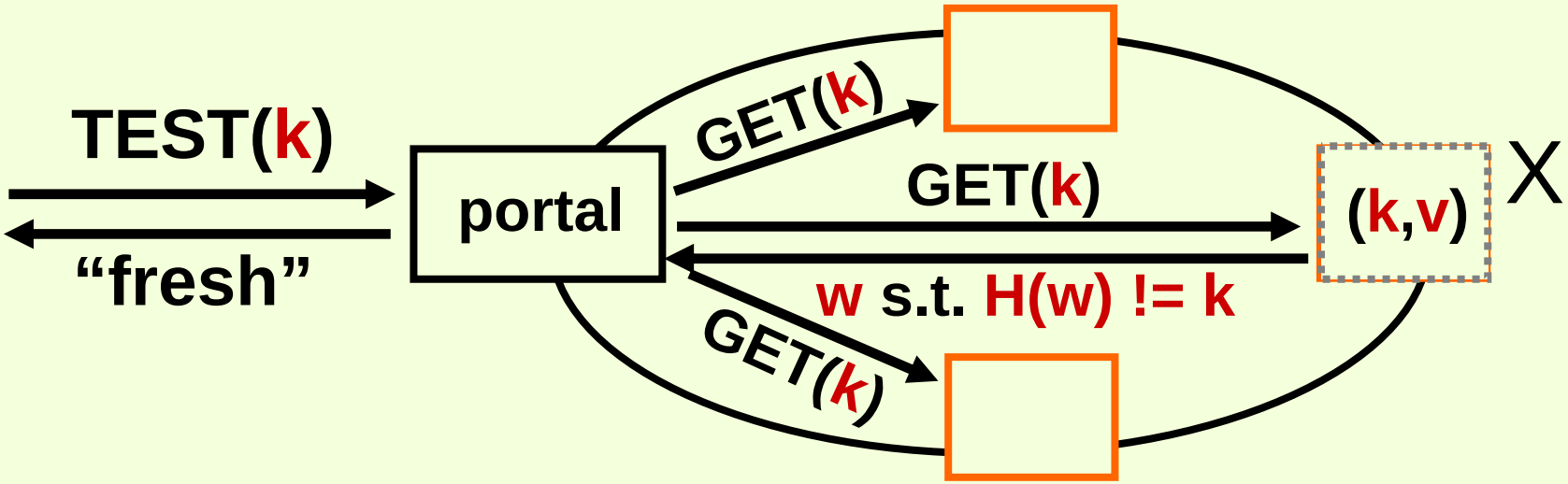
- Each key, k, maps to r potential nodes. r « n.
  - r derived in paper; based on expected faults and n
- Internal enforcer interface: PUT(H(v),v), GET(k)
- SET() → portal PUTs at a random potential node

How the Enforcer Stores Key-Value Pairs



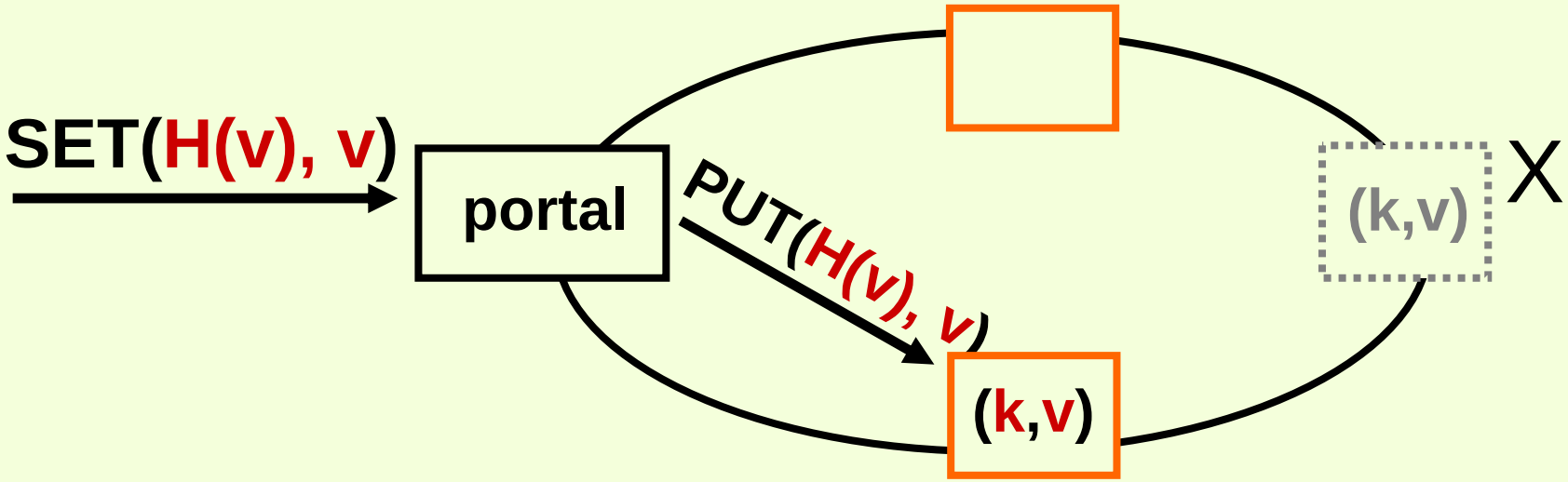
- Each key, k, maps to r potential nodes. r « n.
  - r derived in paper; based on expected faults and n
- Internal enforcer interface: PUT(H(v),v), GET(k)
- SET() → portal PUTs at a random potential node
- TEST() → portal GETs at up to r potential nodes
- PUT once (instead of r) to minimize resource use

Why is this Design Fault-Tolerant?



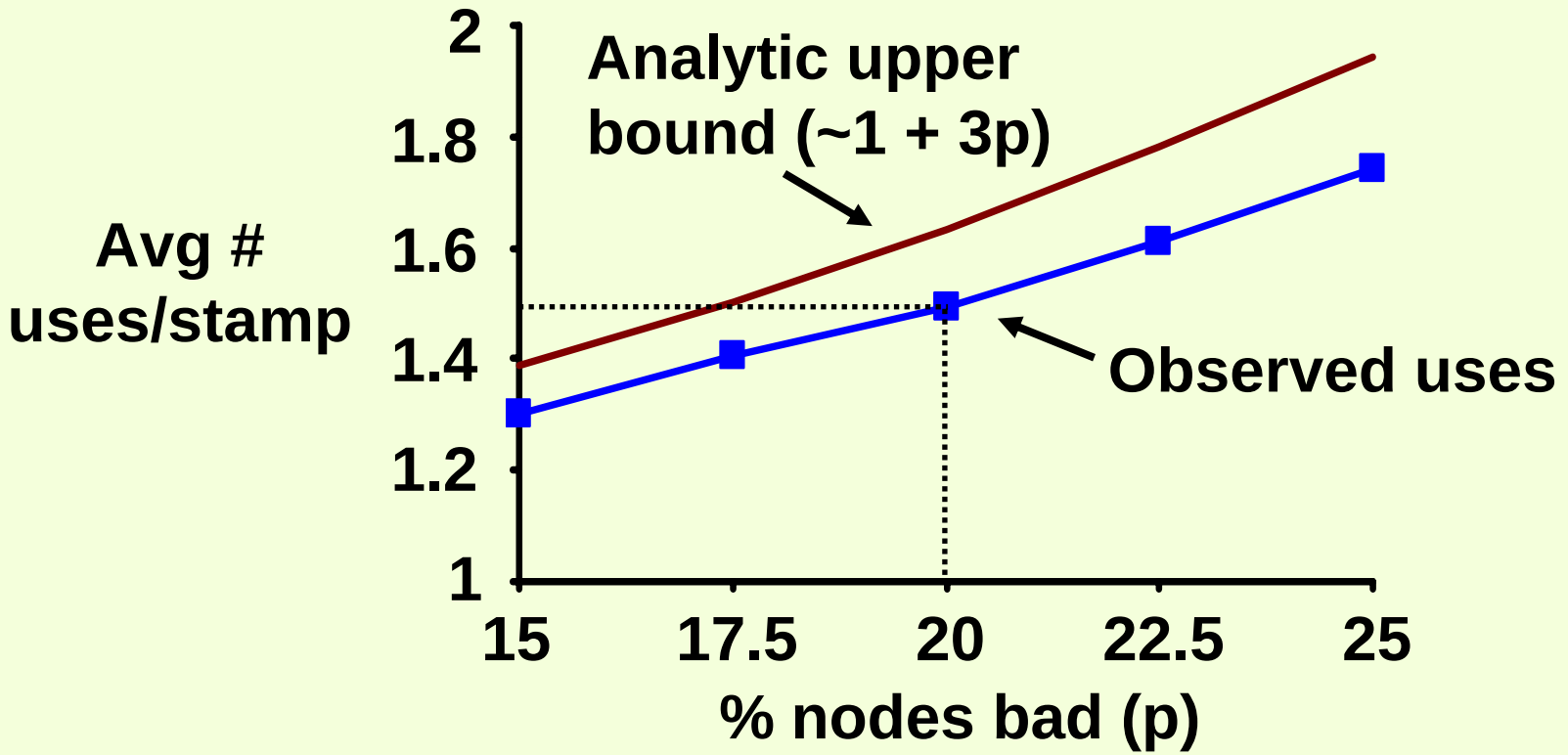
- Let's investigate a Byzantine failure
  - Assume portal good; paper relaxes that assumption
- Failures do not cause fresh stamp to look reused
- Failures do cause:
  - Reused stamp to look fresh (so, more spam)

Why is this Design Fault-Tolerant?



- Let's investigate a Byzantine failure
  - Assume portal good; paper relaxes that assumption
- Failures do not cause fresh stamp to look reused
- Failures do cause:
  - Reused stamp to look fresh (so, more spam)
  - Which causes another SET and another PUT
- If most nodes good, stamp quickly PUT to good one

Experimental Evidence for Fault-Tolerance



- 40 nodes connected to LAN; models cluster
- Each stamp queried 32 times at random portals
- The reuse is acceptable because quotas already set spam to a negligible level

The Required Scale is Manageable

- Understanding the system bottleneck:
  - TEST of fresh stamp: response is fast
  - TEST of reused stamp: may require a disk seek
  - Worst case: every spam generates one disk seek
- To calculate the required number of machines:
  - 100B emails/day; 65% spam → 65 B spams/day
  - One disk: 400 seeks/sec. → 35 M seeks/day
  - So, ~2000 disks needed: ~700 high-end servers

The Enforcer is Practical and Plausible

- Its one trust assumption is realistic, we believe
  - Human-scale job to distribute daily list of *n* nodes
- Minimal design
  - No request routing
  - No keeping track of other nodes
  - No replica maintenance
  - But some engineering required; see paper
- We discussed fault-tolerance, mutual distrust, scale
  - Paper discusses attack resistance

Summary

- Quotas: economic mechanism to control volume
- Enforcer: technical mechanism to enforce quotas

Our focus: a practical enforcer that

- Can handle workload from world's email volume *without much mechanism*
  - By exploiting weak application semantics
- Almost always blocks spam
- Always lets valid email through

http://nms.csail.mit.edu/dqe

Appendix Slides

Resource Attacks

- Internal attacks (by adversarial nodes)
  - Spurious PUTs and GETs to exhaust storage
  - Defense: nodes have "PUT/GET quotas" for each other
- External attacks (by adversarial "receivers")
  - Spurious TESTs and SETs to waste enforcer's resources
  - Defense 1: profile-and-block anomalous requesters
  - Defense 2: "make requests cost bandwidth"
    - Assumes attackers have *some* bandwidth constraint
    - Then, enforcer can limit volume of TESTs and SETs

Can Stamps be Stolen?

- Need to hack outbound mail server to steal
  - Because that's where stamps are stored
  - But this vulnerability is not introduced by DQE
- You might think, "What about botted hosts?" But:
  - Most bots not running *legitimate* mail servers
  - Attack must be bot impersonating sender to outbound mail server. Can thwart by:
    - Authenticating the sender
    - Provider contacting customer out-of-band

What if Portal is Adversarial?

- Client can choose portal
  - Random choice likely to find good one
- If much spam appears to have fresh stamps:
  - Client can switch portals
  - Client can contact multiple portals

Other Spam Solutions (Incomplete List)

Postage:

- Sender pays receiver in $$; if sender good, receiver refunds
- Sender pays receiver in computation [hashcash, camram]

Bankable Postage (closest to quotas):

- Sender gets stamps offline. [Penny Black, SHRED, Goodmail]
- Existing proposals don't meet our "guiding principles"

Other:

- Sender-address validation (RBLs, DomainKeys, SPF)
- Throttle untrusted heavy senders [Templeton]
- Bounce suspected spam (Mail Avenger)

Possible Deployment Paths for DQE

- Note: only mail servers need to work with stamps

Deployment possibilities:

(1) Large email providers drive

- They could federate, agree on stamp format, and allocate quotas to their users
- Or each could run its own *separate* enforcer

(2) Organization-by-organization adoption

- Treat in-quota stamps as whitelisting tool
- Stamp identifies "guaranteed valid" mail

Mailing Lists and DQE

- Moderated lists: sender spends one stamp and
  - List owner can sign the message or
  - List owner can spend stamps for each receiver
- Unmoderated lists: problematic.
- Partially moderated lists?
  - Monitor messages from new contributors, only?

How to Set Quotas?

- Can reduce spam by factor f with per-mail price:
  - Assume spammers are profit-maximizing …
  - … and make $P by sending m messages
  - Per-stamp cost of $c → # msgs limited to P/c
  - So set c = f * (P/m)
  - But this was a very pessimistic calculation
- How to accommodate legitimate heavy senders?
  - Whitelists or refunds
- What about people in poorer parts of the world?

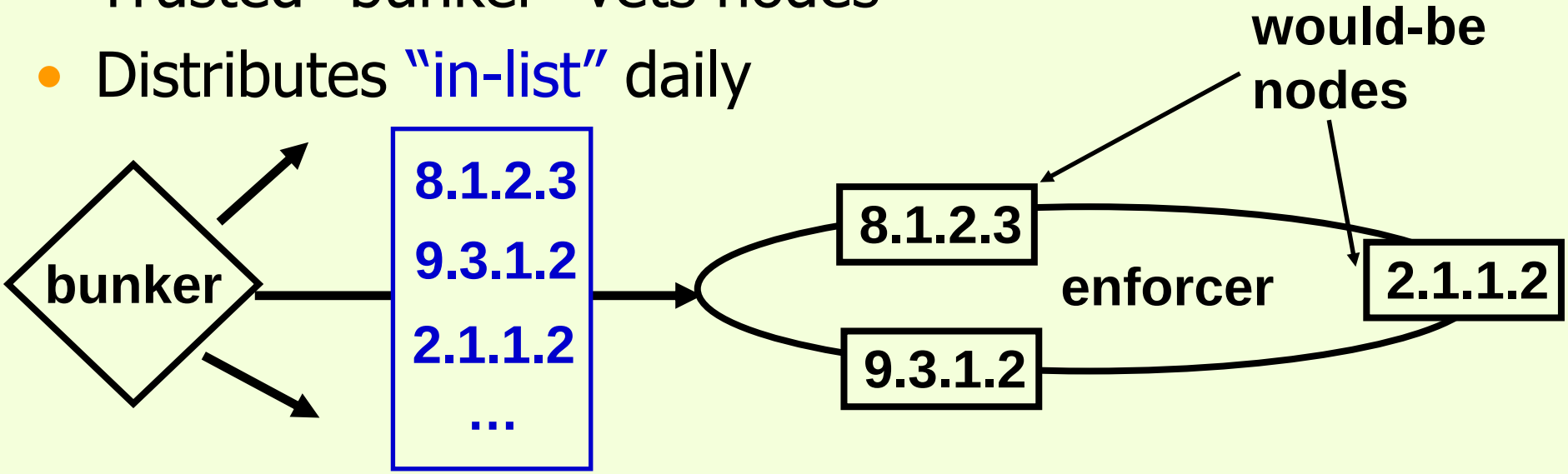
Bounding Stamp Reuse Analytically

- Model:
  - Define "good": node remains up and follows protocol
  - Let p = prob. a node isn't "good" while a stamp is "live"
- Analysis:
  - Once PUT to a "good" node, stamp s no longer reused …
  - … and if most nodes "good", this event happens soon
  - Let U = E[uses of s]. Paper shows: $U < 1/(1-2p) + p^r n$
  - Choose $r = 1 + \log_{1/p} n$. Then $U < 1 + 3p$
  - For example, if $p = .1$, $U < 1.3$ uses

This reuse is acceptable because the quotas already set spam to a negligible level.

The Required Scale Permits Static Config.

- Trusted "bunker" vets nodes
- Distributes "in-list" daily



- Does not track whether nodes are "up"
- We believe bunker is realistic assumption
  - Vetting can be light
  - Human-scale job for 100s or 1000s of PCs