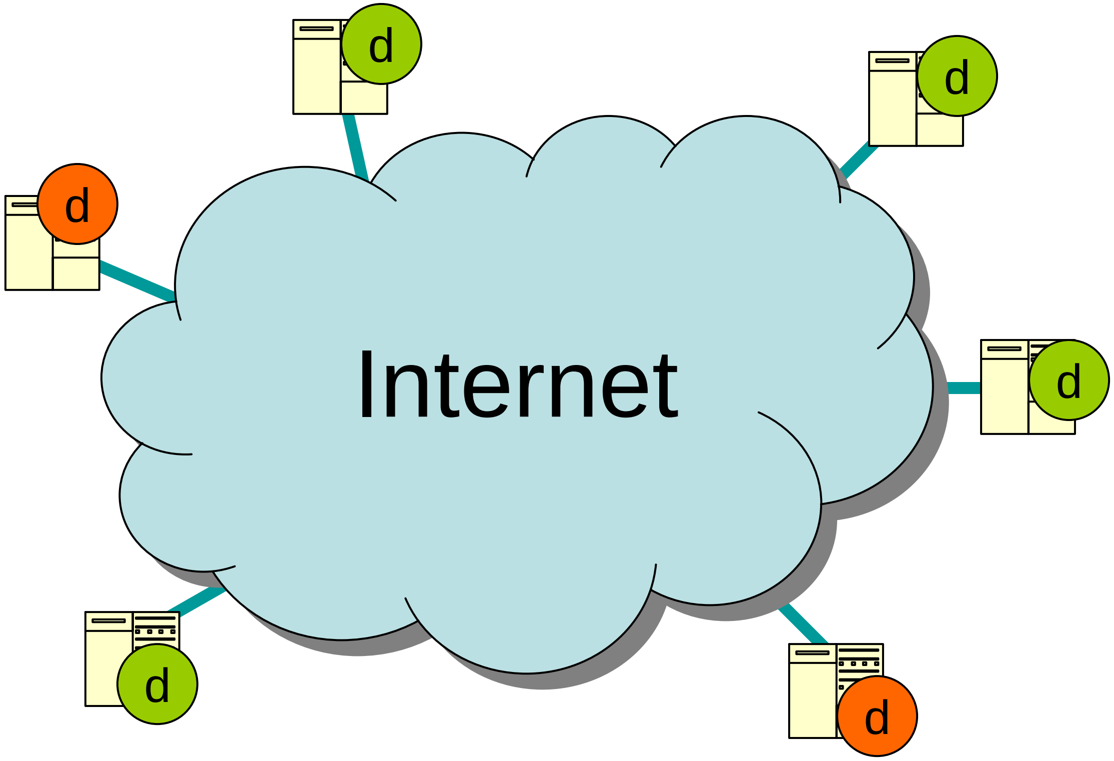
A Solution to the Past-Insider Attack

Many slides by Christian Scheideler: Thanks!

Matthias Baumgart
Christian Scheideler
Stefan Schmid

Motivation

In 2007, a major DoS attack was launched against the root servers of the DNS system



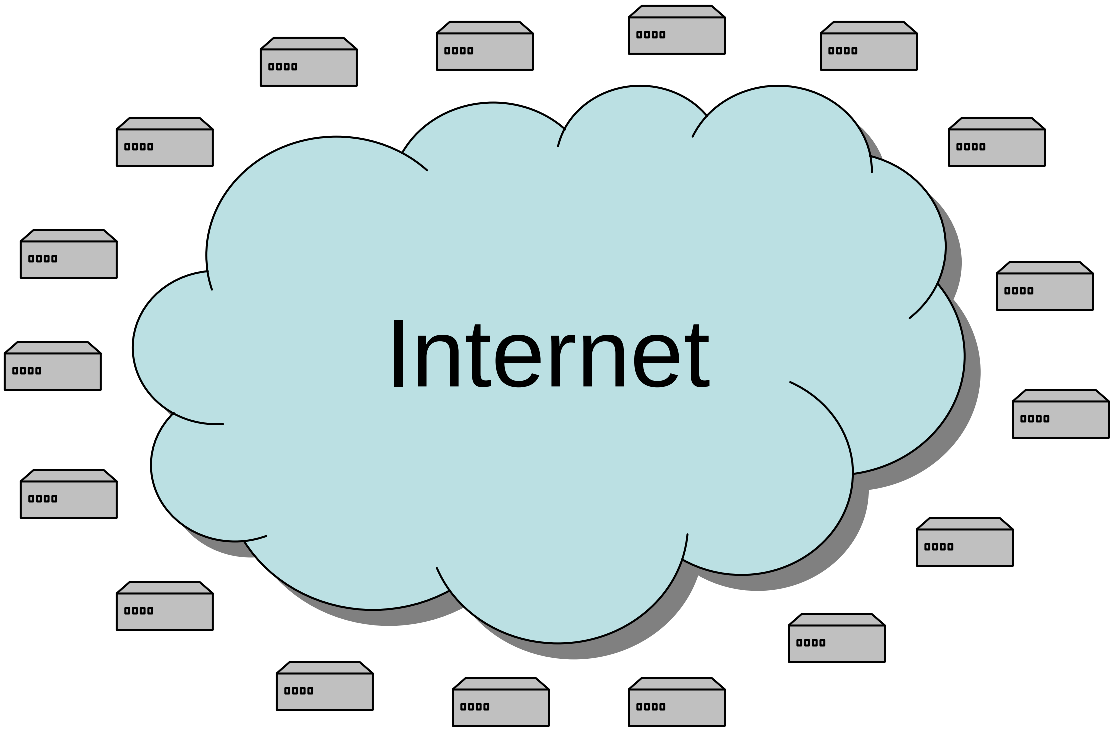
Data loss...
Solution?

DoS-resistant Information System

Solution: Replication

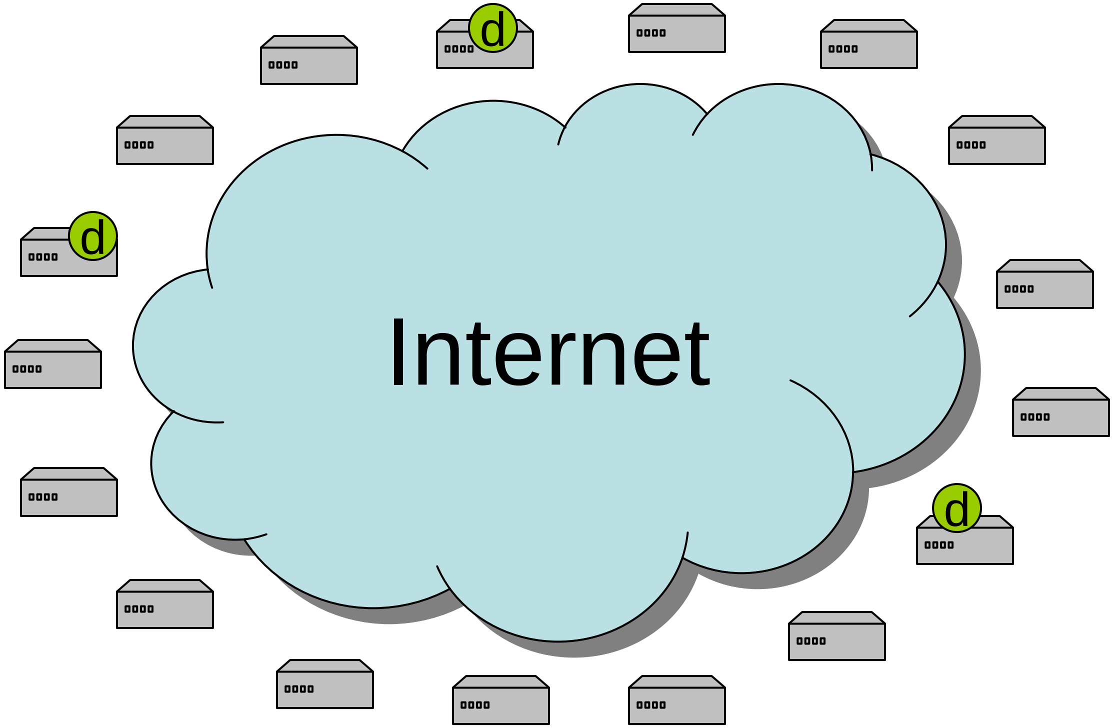
Problem: DNS-approach of full replication not feasible in large information systems

off-the-shelf servers



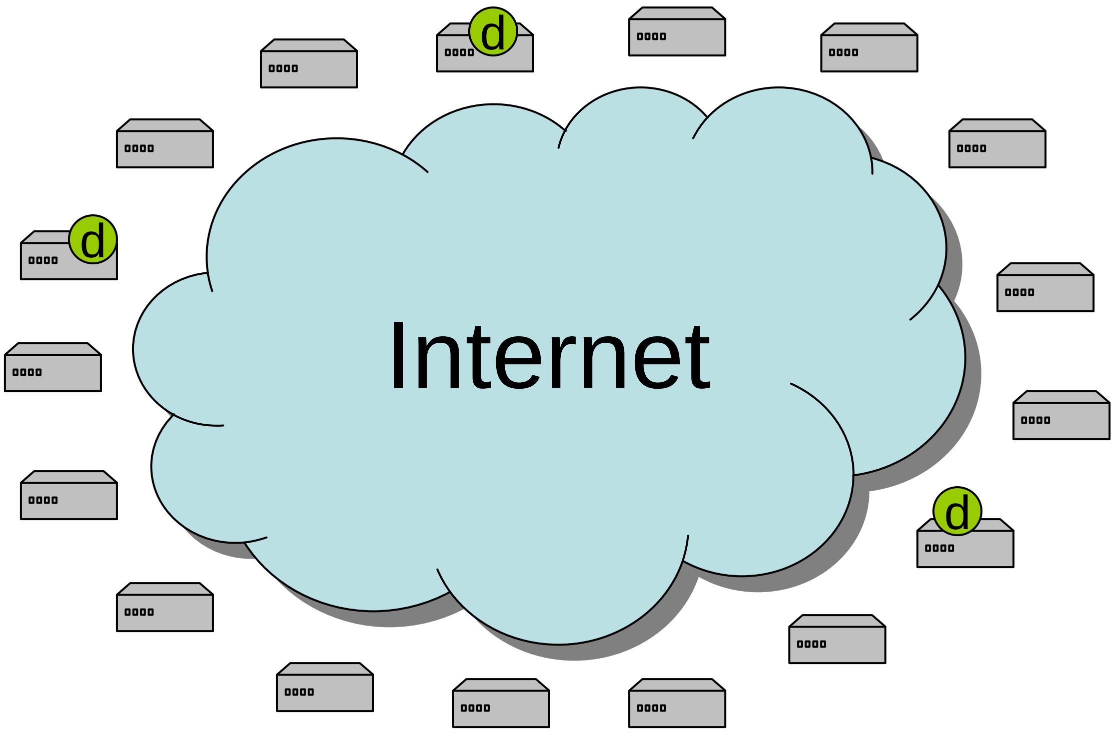
DoS-resistant Information System

Scalable information system: storage over-head limited to logarithmic factor



DoS-resistant Information System

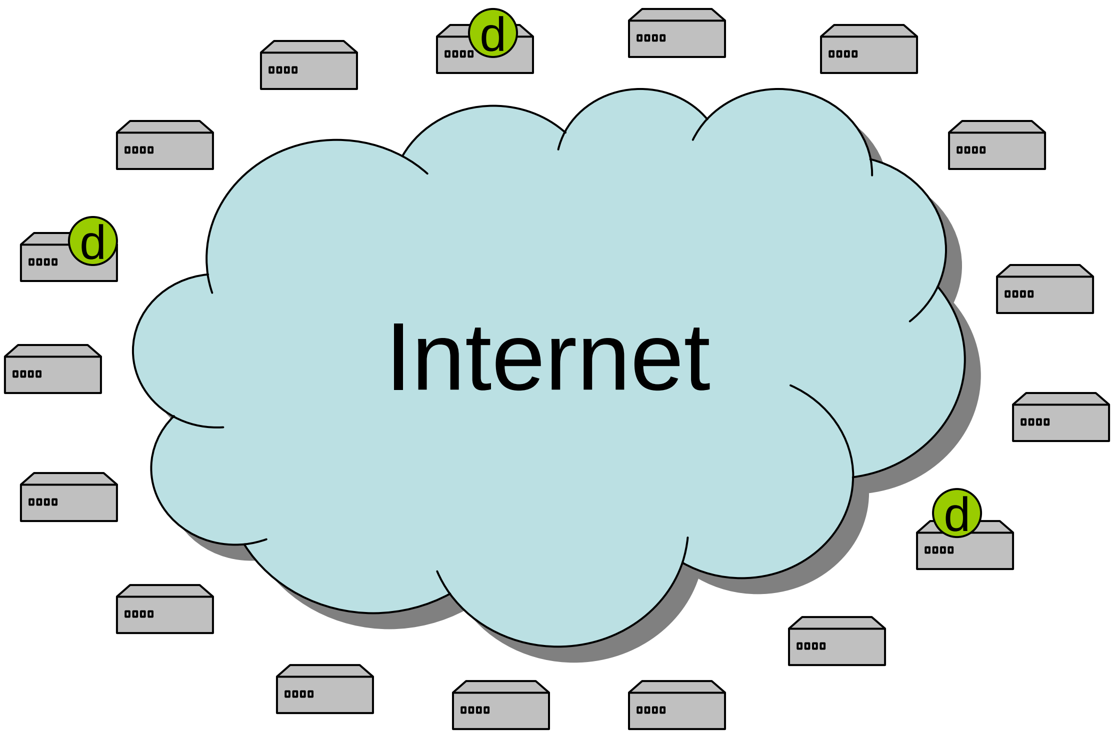
storage overhead limited to log factor: scalable put und get operations possible



DoS-resistant Information System

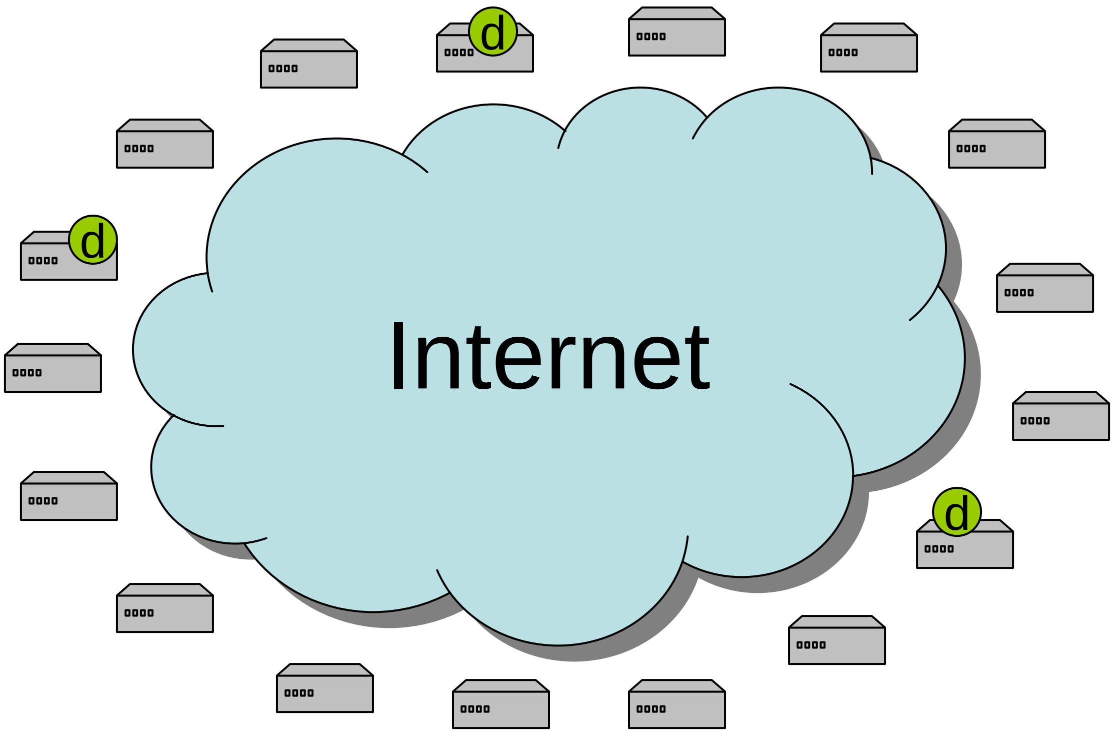
storage overhead limited to log factor:

but how to be robust against DoS attacks?



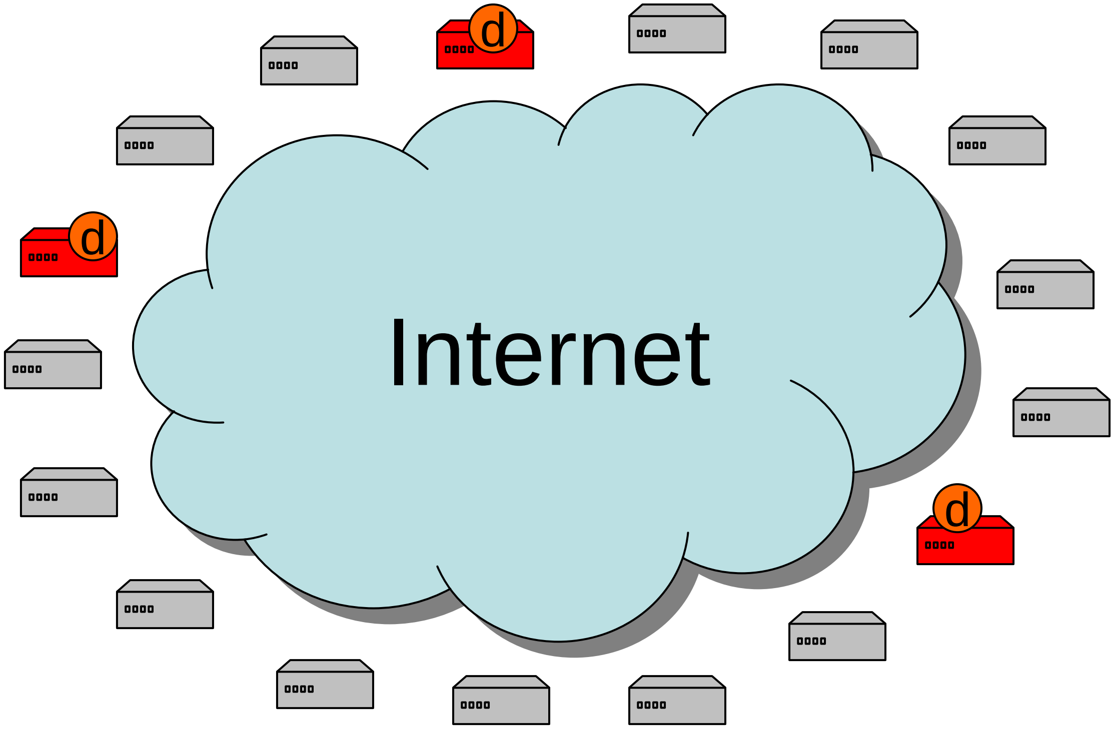
Fundamental Dilemma

- Scalability: minimize replication of information
- Robustness: maximize resources needed by attacker



Fundamental Dilemma

- Limitation to „legal“ attacks / information hiding
- Information hiding difficult under insider attacks



DoS-resistant Information System

Past-Insider-Attack: Attacker knows everything about system till (unknown) time t_0

Goal: scalable information system so that everything that was inserted after t_0 is safe (w.h.p.) against any past-insider DoS attack that can shut down any ε-fraction of the servers, for some $\varepsilon>0$, and create any legal set of put and get requests

Formal Model

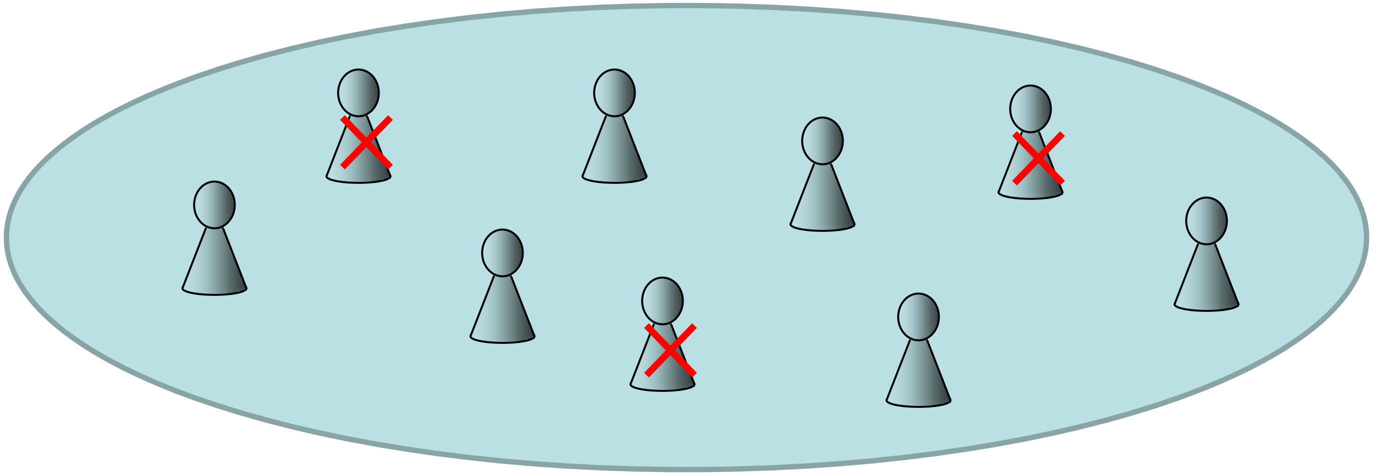
We are given a static set of n reliable servers.

ε-bounded attacker:

- knows entire system till time t_0 (unknown to system)
- can block any ε-fraction of servers
- can generate any set of put/get requests, one per server

Goals:

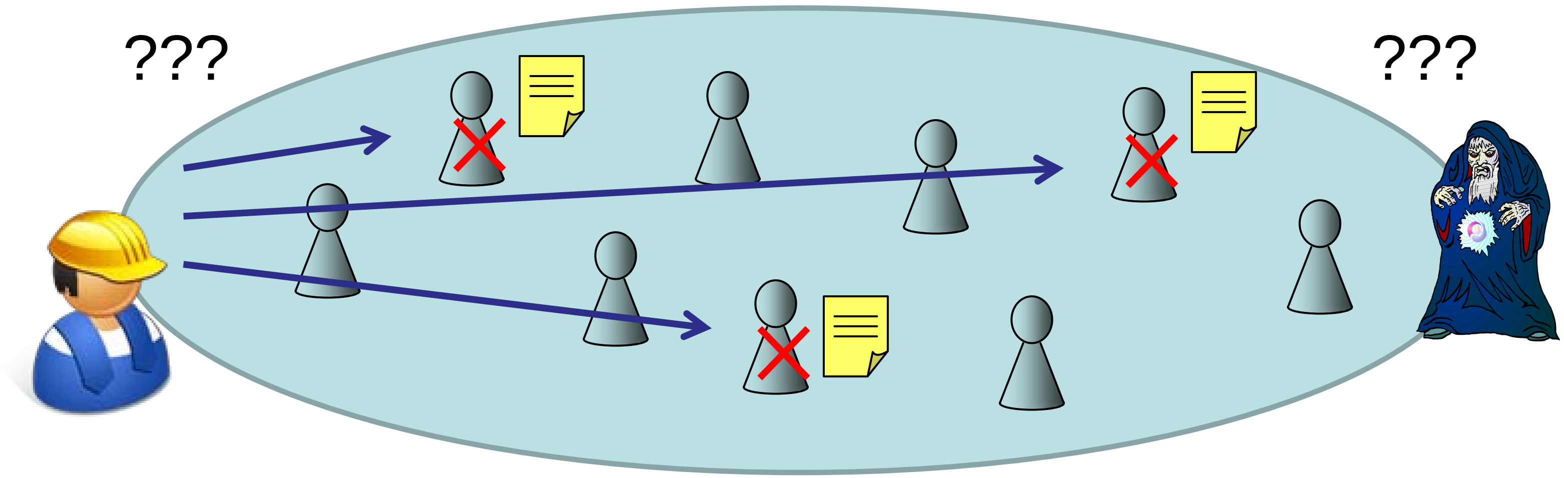
- Scalability: every server spends at most polylog time and work on put and get requests
- Robustness: every get request to a data item inserted or updated after t_0 is served correctly
- Correctness: every get request to a data item is served correctly if the system is not under DoS-attack



DoS-resilient Information System

Dilemma:

- just polylog copies allowed per data item to be scalable



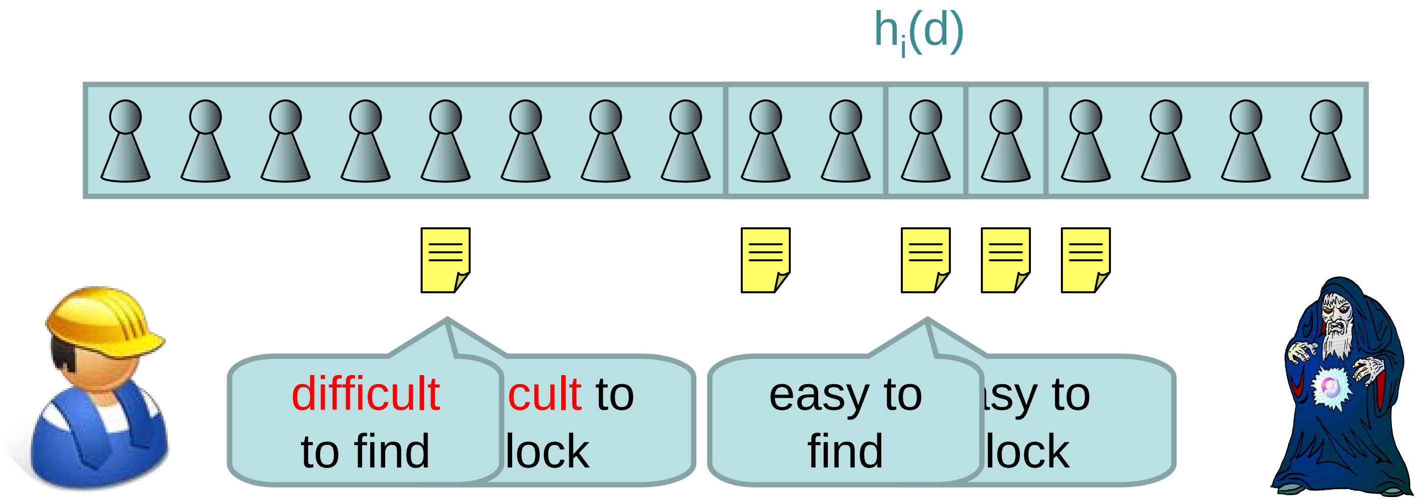
Don‘t know where
to attack – and search!

randomized placement
deterministic placement

DoS-resilient Information System

Basic strategy:

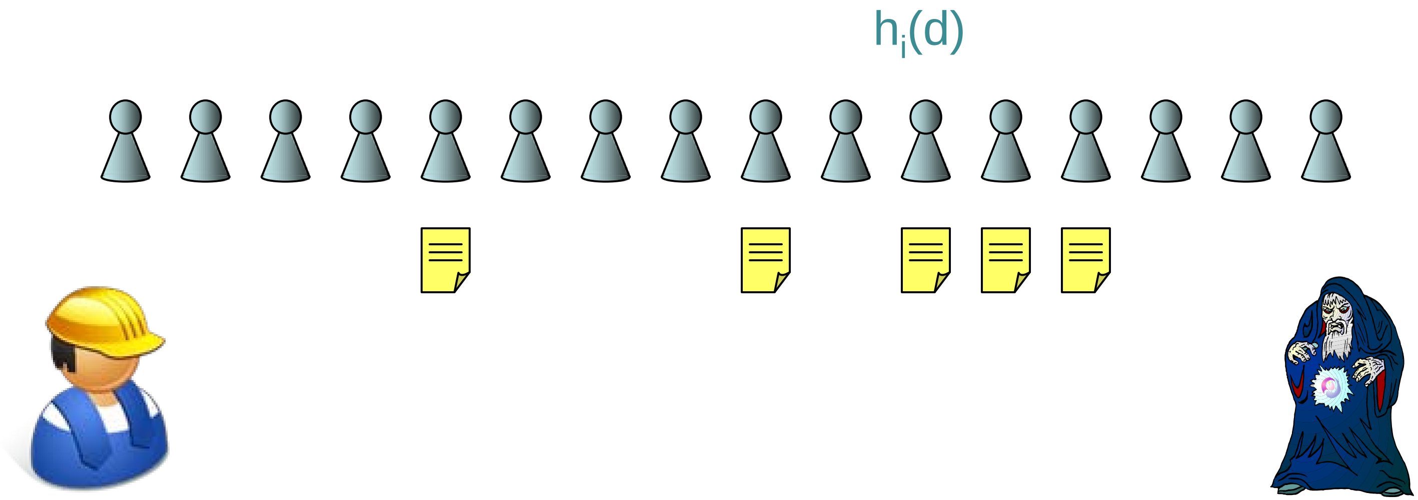
- choose suitable hash functions $h_1,..,h_c$:D→V
  (D: name space of data, V: set of servers)
- Store copy of item d for every i and j randomly in a set of servers of size 2^j that contains $h_i(d)$



DoS-resilient Information System

„Tie“ sufficient for get requests [DISC 07]:

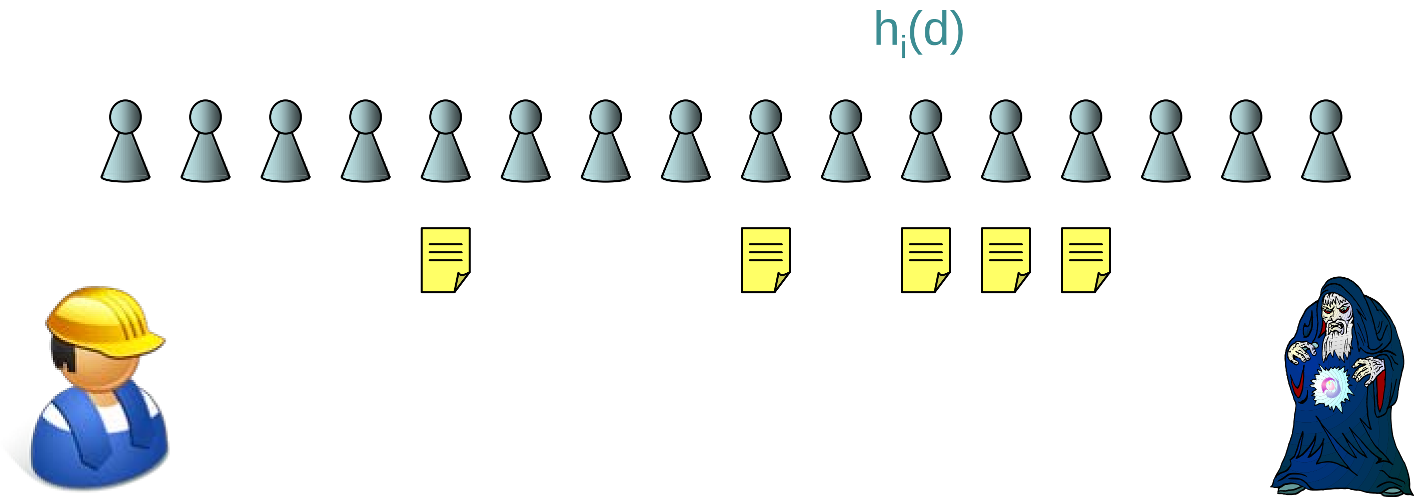
- Most get requests can access close-by copies, only a few get requests have to find distant copies
- Work for each server altogether just polylog(n) for any set of n get requests, one per server



DoS-resilient Information System

„Tie“ sufficient for get requests [DISC 07]:

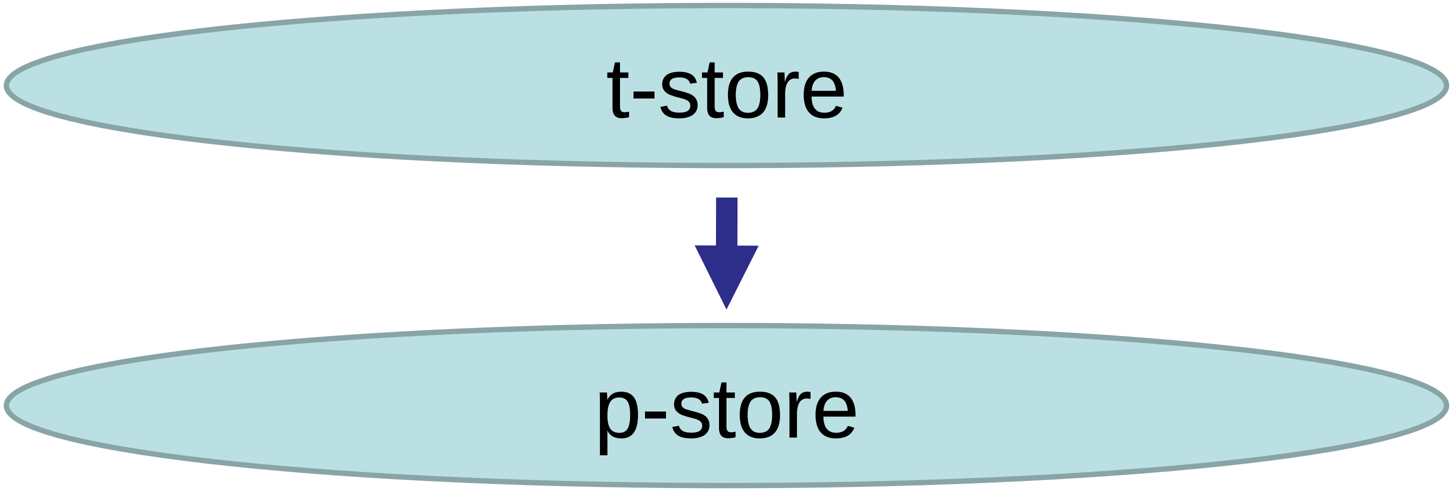
BUT for get requests to work, all areas must have up-to-date copies, so put requests may fail under DoS attack



DoS-resilient Information System

Chameleon system: two stores (DHTs)

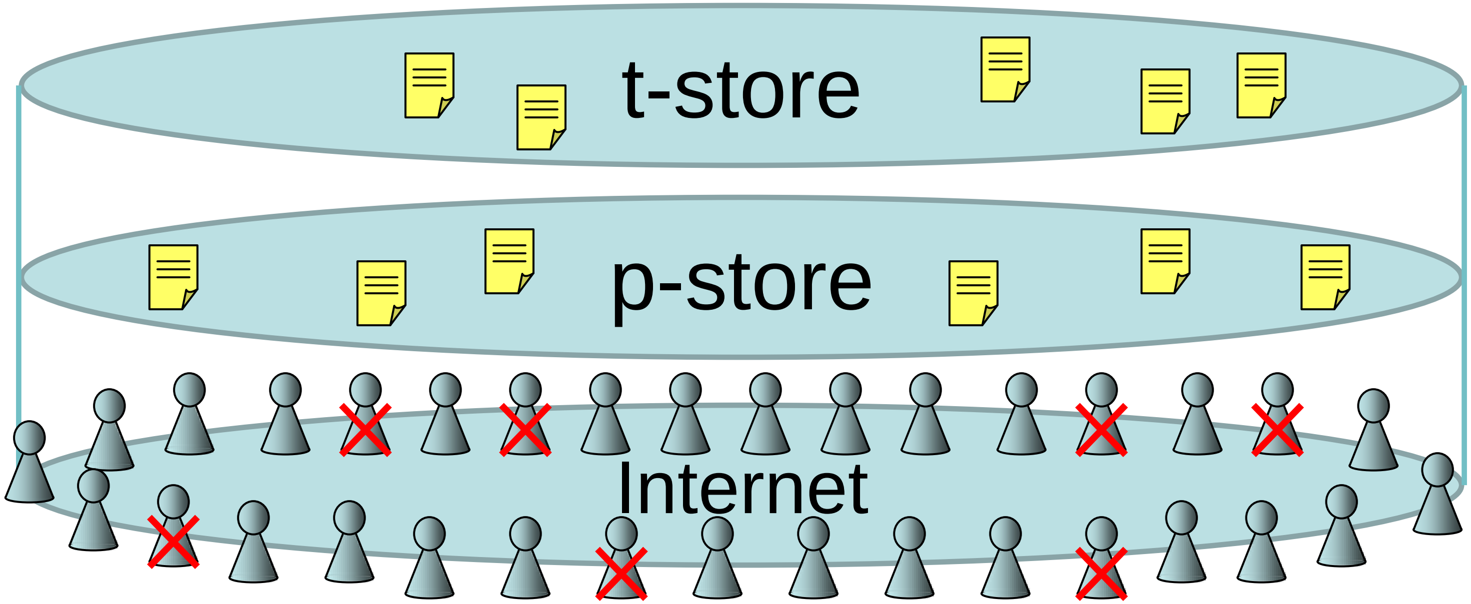
- Permanent distributed hash table (p-store)
  $h_1,..,h_c$ fixed
- Temporary distributed hash table (t-store)
  hash function h continuously changes
  - a „buffer“, at most O(n) items wait
  - not known by past insider!



DoS-resilient Information System

Phase of Chameleon system:

1. Adversary blocks servers and initiates put & get requests
2. build new t-store, transfer data from old to new t-store
3. process all put requests in t-store
4. process all get requests in t-store and p-store
5. try to transfer data items from t-store to p-store



Stage 2: Build new t-store

t-store: distributed hash table (DHT)
(de Bruijn network + consistent hashing)

New t-store:

- Join protocol: Every node chooses new random location in de Bruijn network, searches for neighbors in p-store
- Insert protocol: Data items in old t-store are stored in new t-store (just O(n) items w.h.p.)

O(log n) time and congestion w.h.p.

Stage 3: Process puts in t-store

- t-put protocol: de-Bruijn routing with combining to store data in new t-store

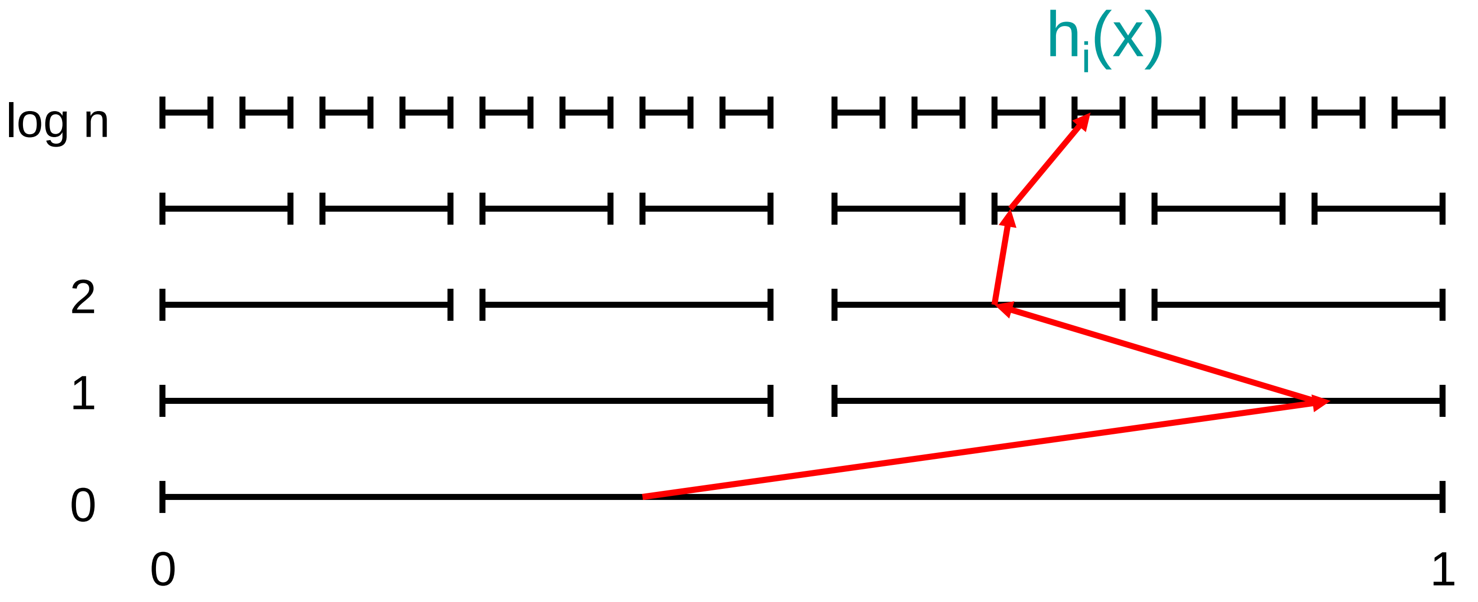
$O(\log n)$ time and $O(\log^2 n)$ congestion

Stage 4: Process get Requests

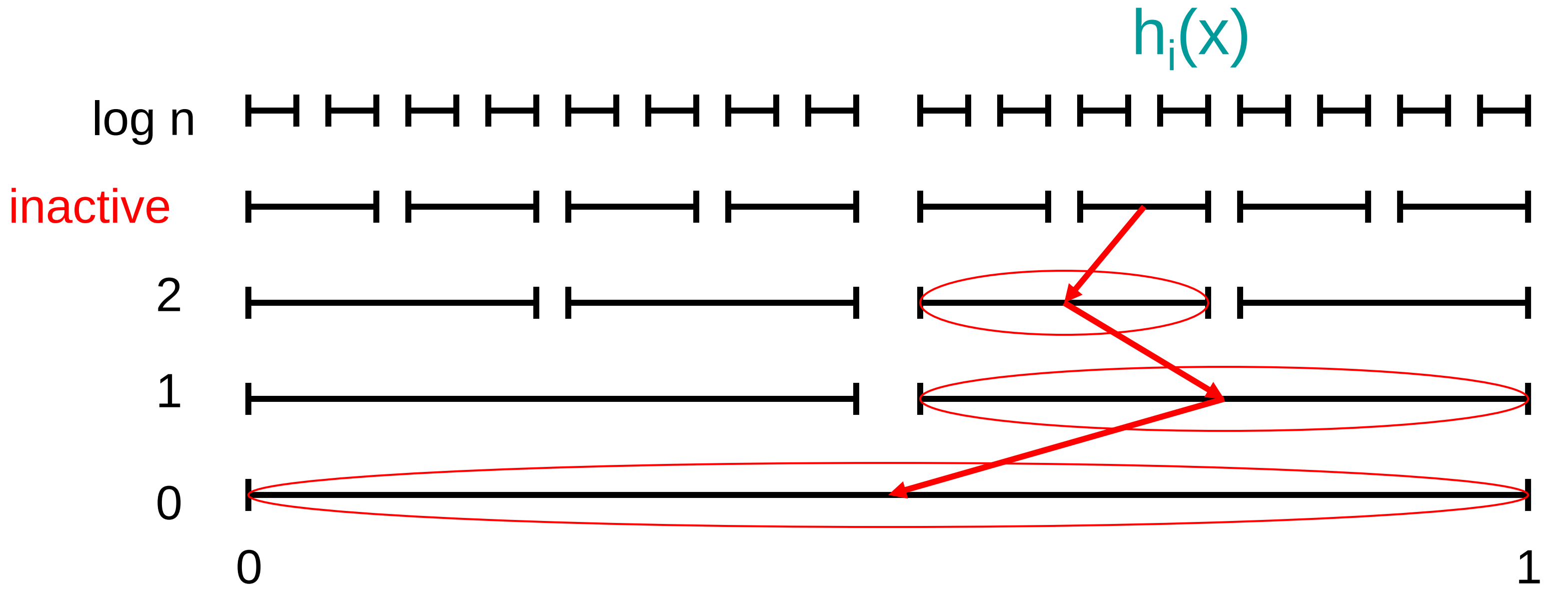
- t-get protocol: de Bruijn routing with combining to lookup data in t-store ($O(\log n)$ time and $O(\log^2 n)$ congestion)

- p-get protocol (related to [DISC 07]):
  - Preprocessing stage: determine blocked areas in p-store via sampling ($O(1)$ time and $O(\log^2 n)$ congestion)
  - Contraction stage: try to get as close as possible to hash-based positions ($O(\log n)$ time and $O(\log^3 n)$ congestion)
  - Expansion stage: look for copies at successively wider areas ($O(\log^2 n)$ time and $O(\log^3 n)$ congestion)

Contraction Stage



Expansion Stage



Stage 5: data from t-store to p-store

- p-put protocol:
  - Preprocessing stage: determine blocked areas and average load in p-store via sampling
    ($O(1)$ time and $O(\log^2 n)$ congestion)
  - Contraction stage: try to get to sufficiently many hash-based positions in p-store
    ($O(\log n)$ time and $O(\log^3 n)$ congestion)
  - Permanent storage stage: for each successful data item, store new copies and delete as many old ones as possible
    ($O(\log n)$ time and $O(\log^2 n)$ congestion)

DoS-resistant Information System

Theorem: Under any ε-bounded past-insider attack (for some constant $\varepsilon>0$), the Chameleon system can serve any set of requests (one per server) in $O(\log^2 n)$ time s.t. every get request to a data item inserted or updated after t_0 is served correctly, w.h.p.

No degradation over time:

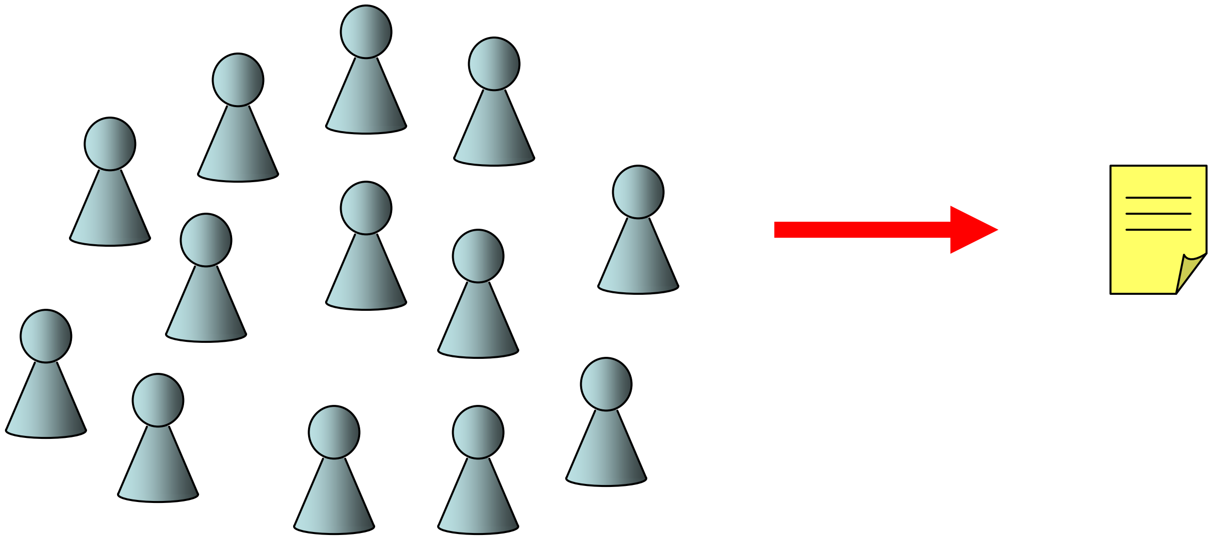
- $O(\log^2 n)$ copies per data item
- fair distribution of data among servers

Related Work

Many scalable information systems:

- Chord, CAN, Pastry, Tapestry,…

But many of these designs not even robust against flash crowds

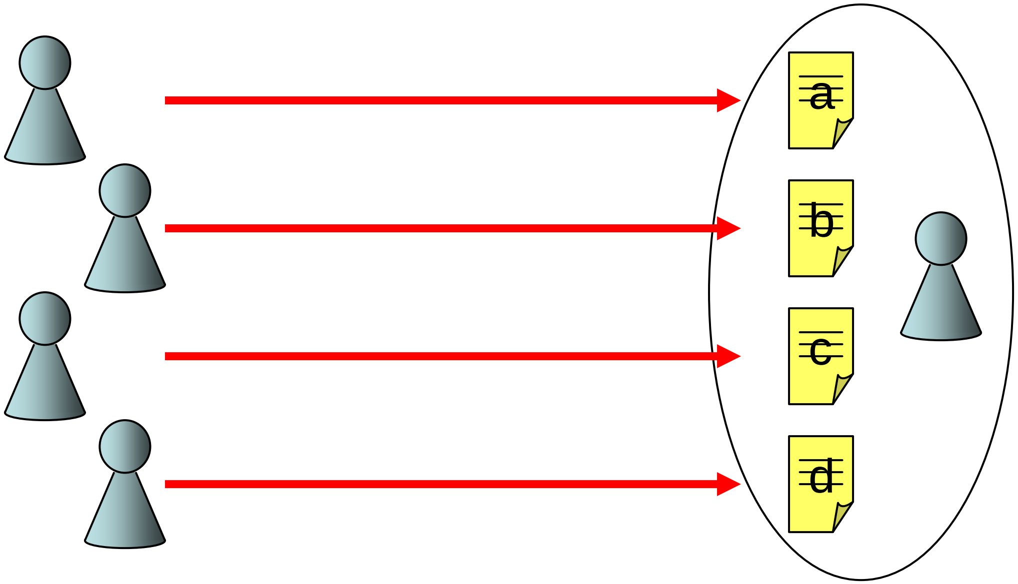


Related Work

Caching strategies against flash crowds:

- CoopNet, Backlash, PROOFS,…
- Naor&Wieder 03

But not robust against adaptive lookup attacks



Related Work

Systems robust against DoS-attacks:

- SOS, WebSOS, Mayday, III,…
- Basic strategy: indirection infrastructure to hide original location of data

Does not work against past insiders

Related Work

Awerbuch & Sch. (DISC 07):

DoS-resistent information system that can only handle get requests under DoS attack

Conclusion

Applications: DoS-resistant platform for e-commerce or critical information services

Open problems:

- More light-weight solution
- DoS-resistant system with bounded degree

Any Questions?