Anonymity

Christian Grothoff
christian@grothoff.org
http://grothoff.org/christian/

"A society that gets rid of all its troublemakers goes downhill."
–Robert A. Heinlein

Agenda

- Definitions and Metrics
- Techniques, Research Proposals and Systems
  - Dining Cryptographers, Mixes, Mixminion, PipeNet, Busses, Mute, Ants, StealthNet, Freenet, P5, APFS, Crowds, Hordes
  - **GNUnet**, Economics and Anonymity, Excess-based Economics

GAP

K. Bennett and C. Grothoff introduced GAP: *practical anonymous networking*:

- based on link-to-link encryted network with only symmetric key operations after links are established
- implemented in GNUnet, supporting GNUnet's integrity and accounting requirements

GAP: features

- a new perspective how to determine anonymity
- search integrated: initiator and responder anonymity
- nodes can individually trade anonymity for efficiency
- nodes can not gain anonymity at the expense of other nodes

⇒ "correct" economic incentives

GAP: query — reply

GAP only supports a very simple query-reply scheme:

- sender basically asks using 512-bit hash code
- responder sends back up go 32k encrypted data
- intermediaries can cryptographically check that encrypted response matches query — without decrypting either!

GAP: key idea

Source rewriting was traditionally used to hide the identity of the source. GAP uses it in a different way:

- Anonymity is achieved by making the initiator look like a router that acts on behalf of somebody else
- It is important to make traffic originating from the router look identical to traffic that the router indirects
- It is **not** necessary to avoid a direct network connection between the responder and the initiator

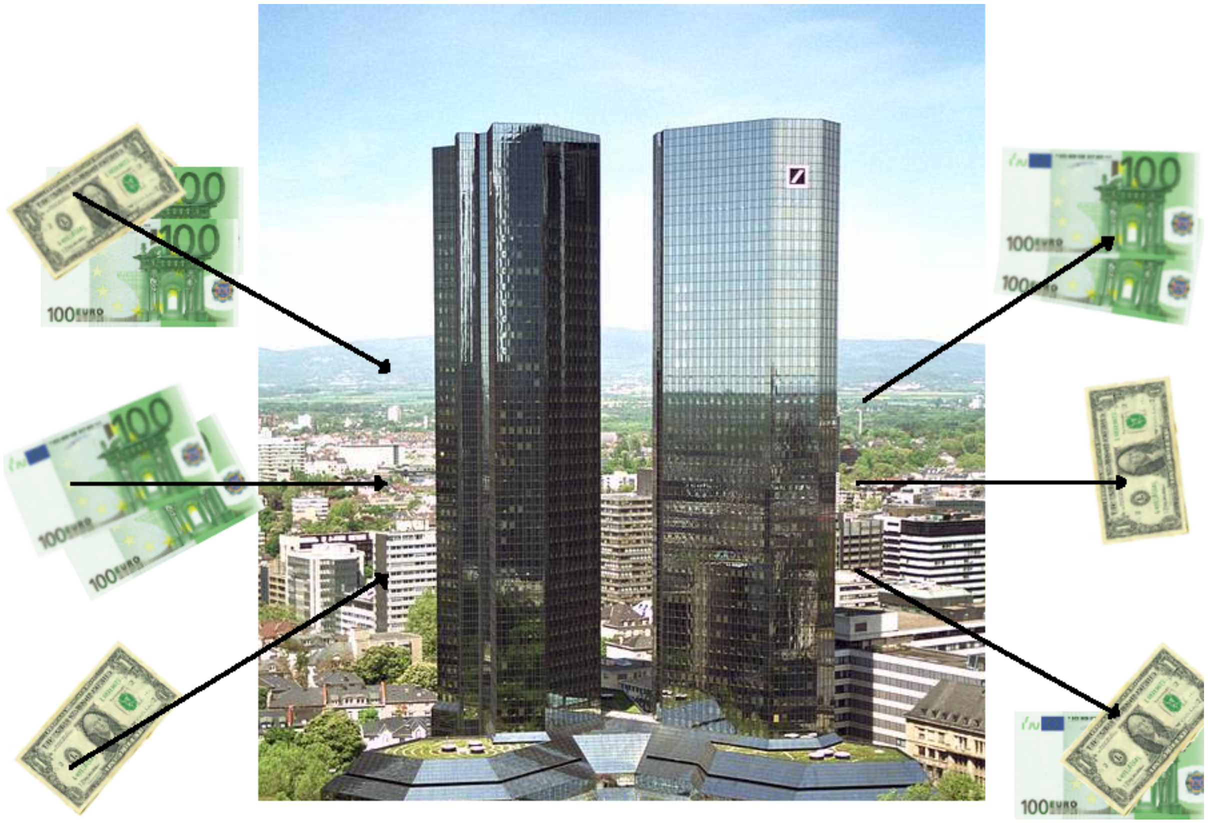
GAP: Money Laundering

Lets illustrate our new perspective with the example of money laundering. If you wanted to hide your financial traces, would you:

- Give the money to your neighbor,
- expect that your neighbor gives it to me,
- and then hope that I give it to the intended recipient?

Worse: trust everybody involved, not only that we do not steal the money but also do not tell the FBI?

GAP: Banks!



GAP: Why indirect?

- Indirections do not protect the sender or receiver
- Indirections can help the indirector to hide its own traffic
- If the indirector cheats (e.g. by keeping the sender address when forwarding) it only exposes its own action and does not change the anonymity of the original participants

GAP: Key Realization

We can restate the key idea behind GAP:

Anonymity can be measured in terms of

- how much traffic from non-malicious hosts is indirected compared to the self-generated traffic
- in a time-interval small enough such that timing analysis can not disambiguate the sources.

GAP: basic protocol

- HELLO: introduce nodes
- SET KEY, PING, PONG: exchang session key
- QUERY: question is $H(E_{H(c)}(C))$
- CONTENT: answer is $E_{H(C)}(C)$

Routig in the Mesh Network

- GNUnet is an **unstructured** peer-to-peer network
- applications can impose a structure on GNUnet
- peers can have different configurations
- peers do **not** communicate their configuration
- GAP routing is based on "smart" flooding

Routing: Local Heuristics

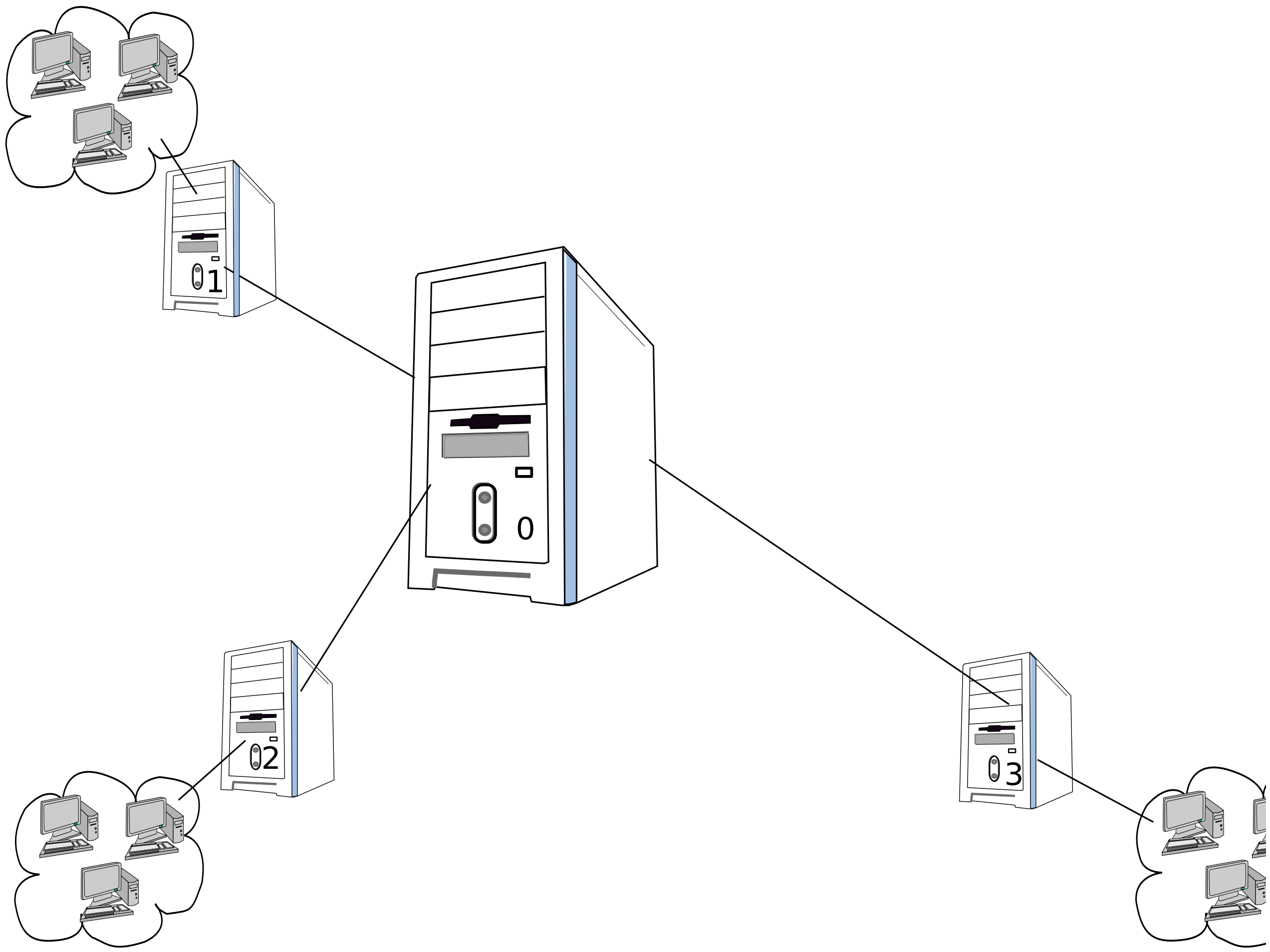
- structured routing is **predictable** and **analyzable**
- GAP keeps routing hard to predict
- **proximity**-based routing is **efficient** for **migrated** content
- **hot-path** routing is **efficient** if queries are **correlated**
- **flodding** is **efficient** if merely **noise** is **substituted**
- **How long should a peer keep track of which queries?**

Time-to-Live

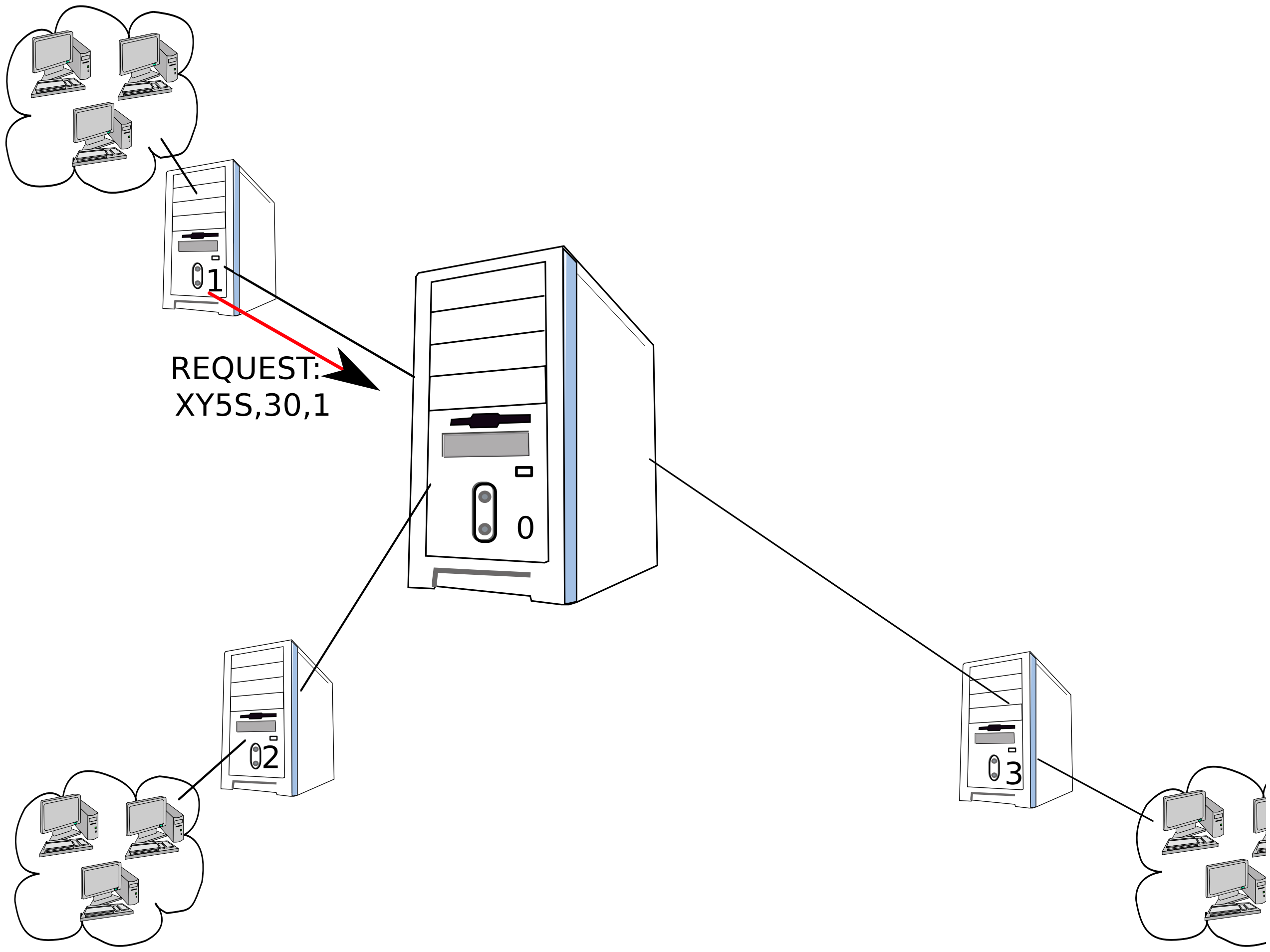
- TTL field in queries is **relative time** and can be **negative**.
- Absolute TTL = NOW + relative TTL
- Absolute TTL and decies which query to **drop**.
- TTL is decremented at each hop.
- peers can still route "expired" queries indefinitely

$\Rightarrow$ better solution than traditional hop-count

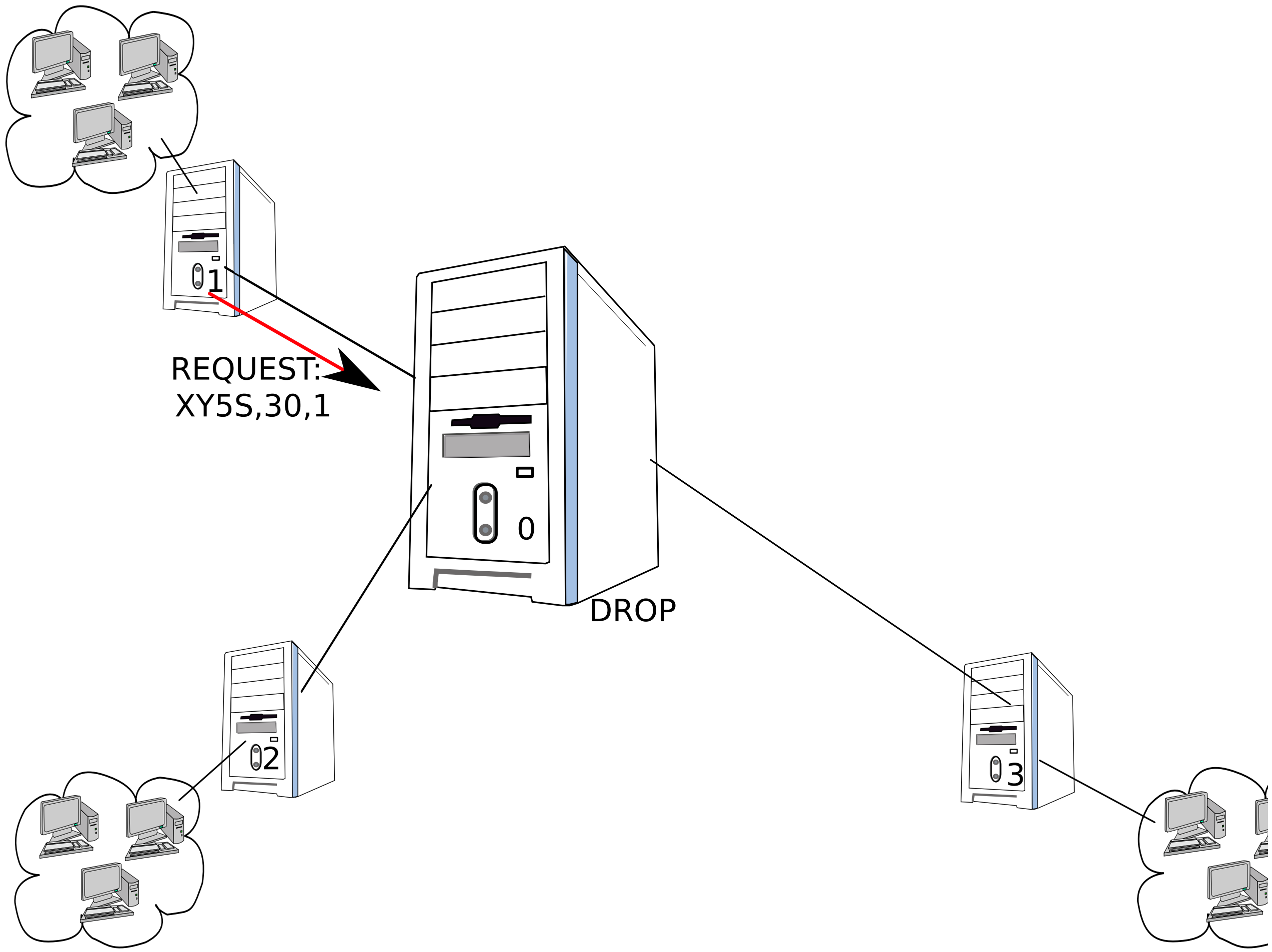
GAP illustrated (1/9)



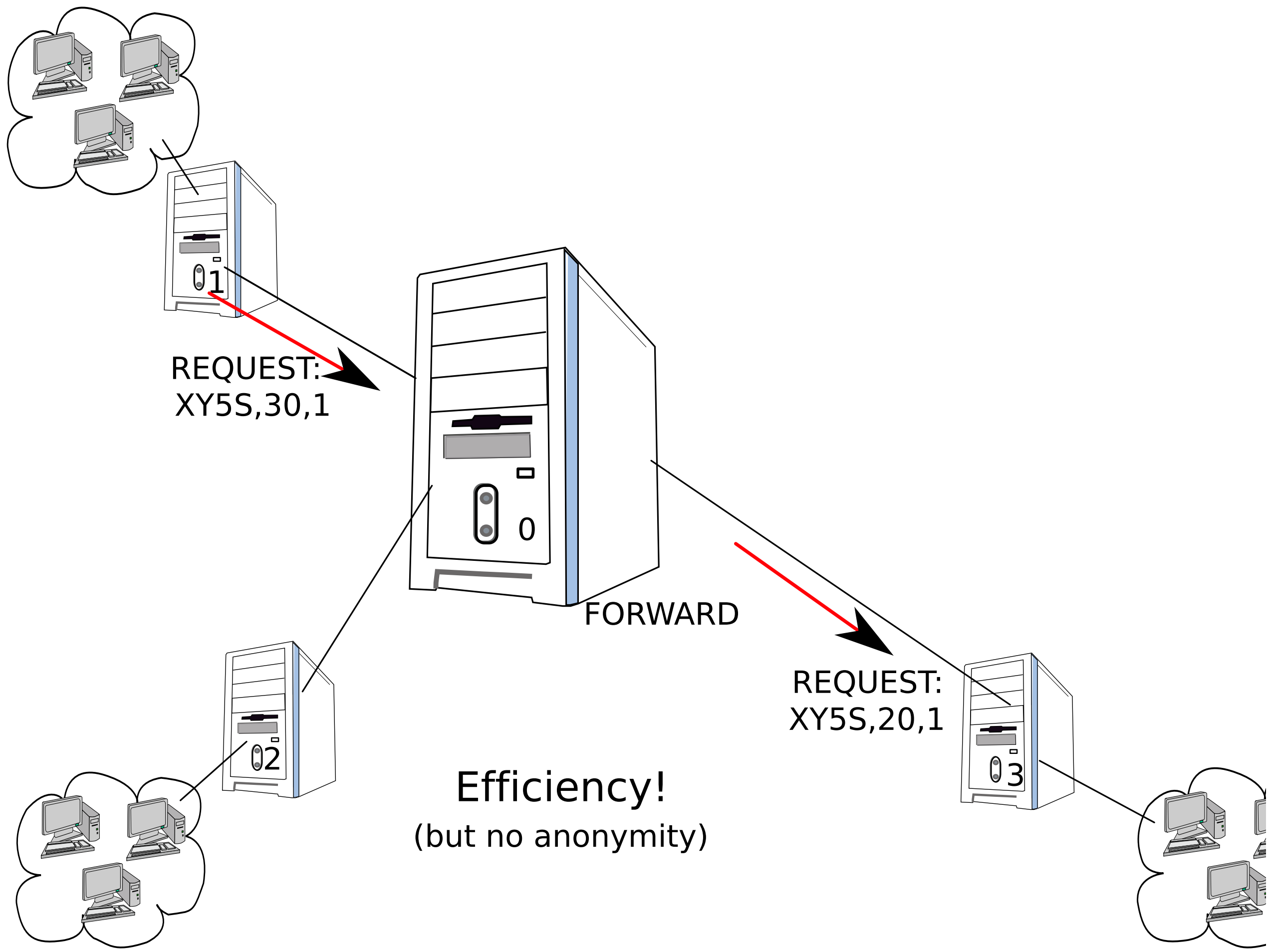
GAP illustrated (2/9)



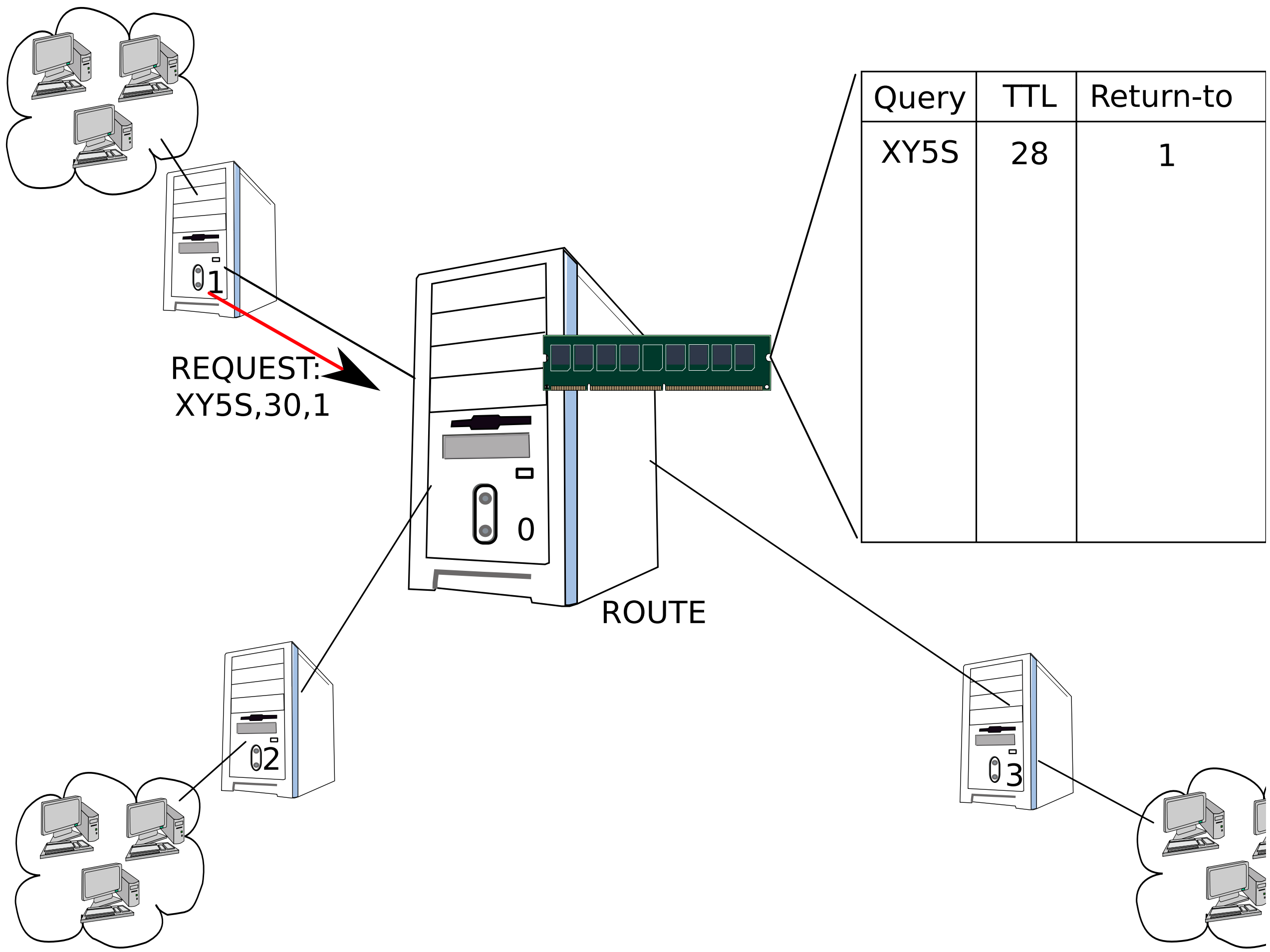
GAP illustrated (3/9)



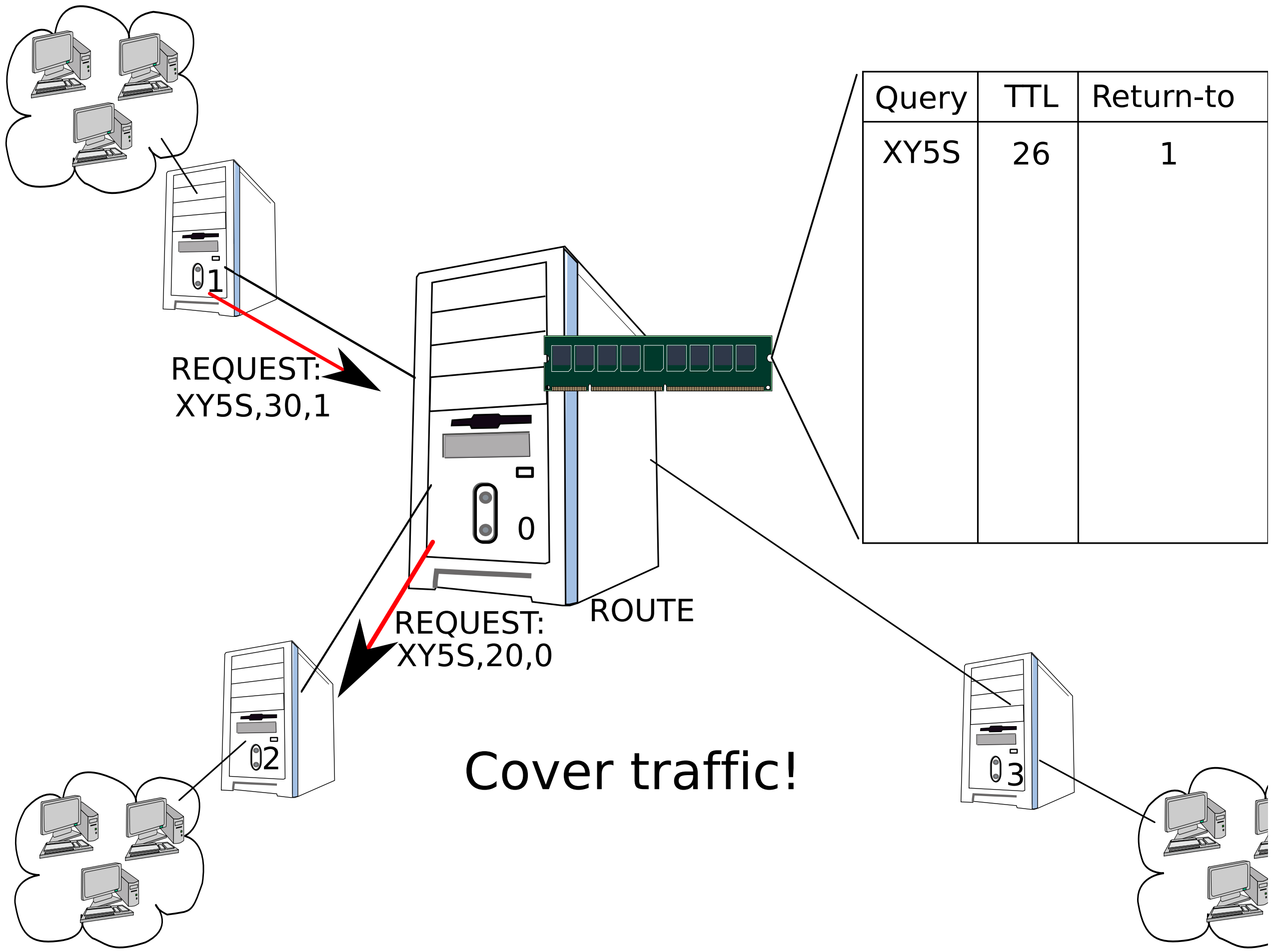
GAP illustrated (4/9)



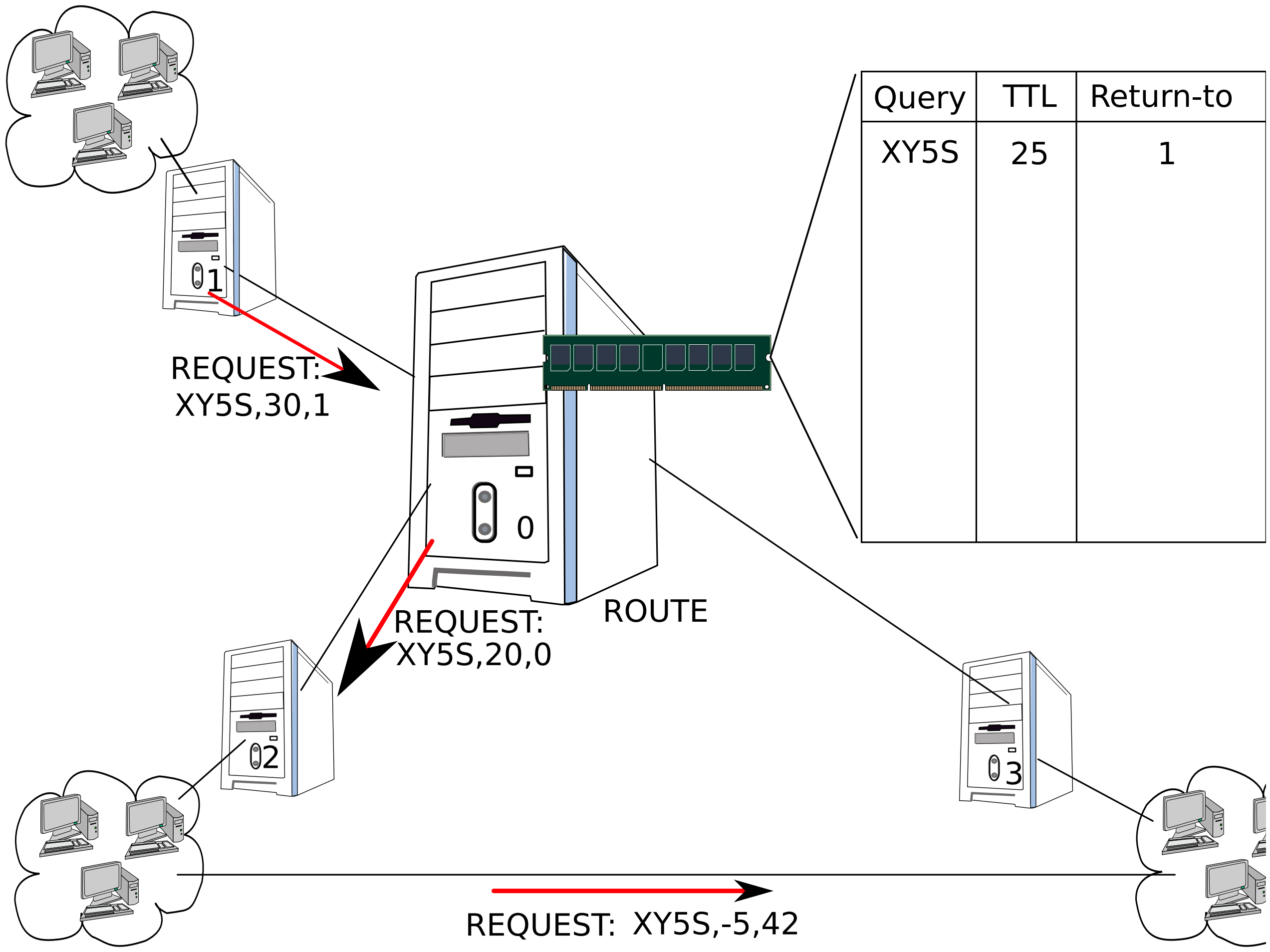
GAP illustrated (5/9)



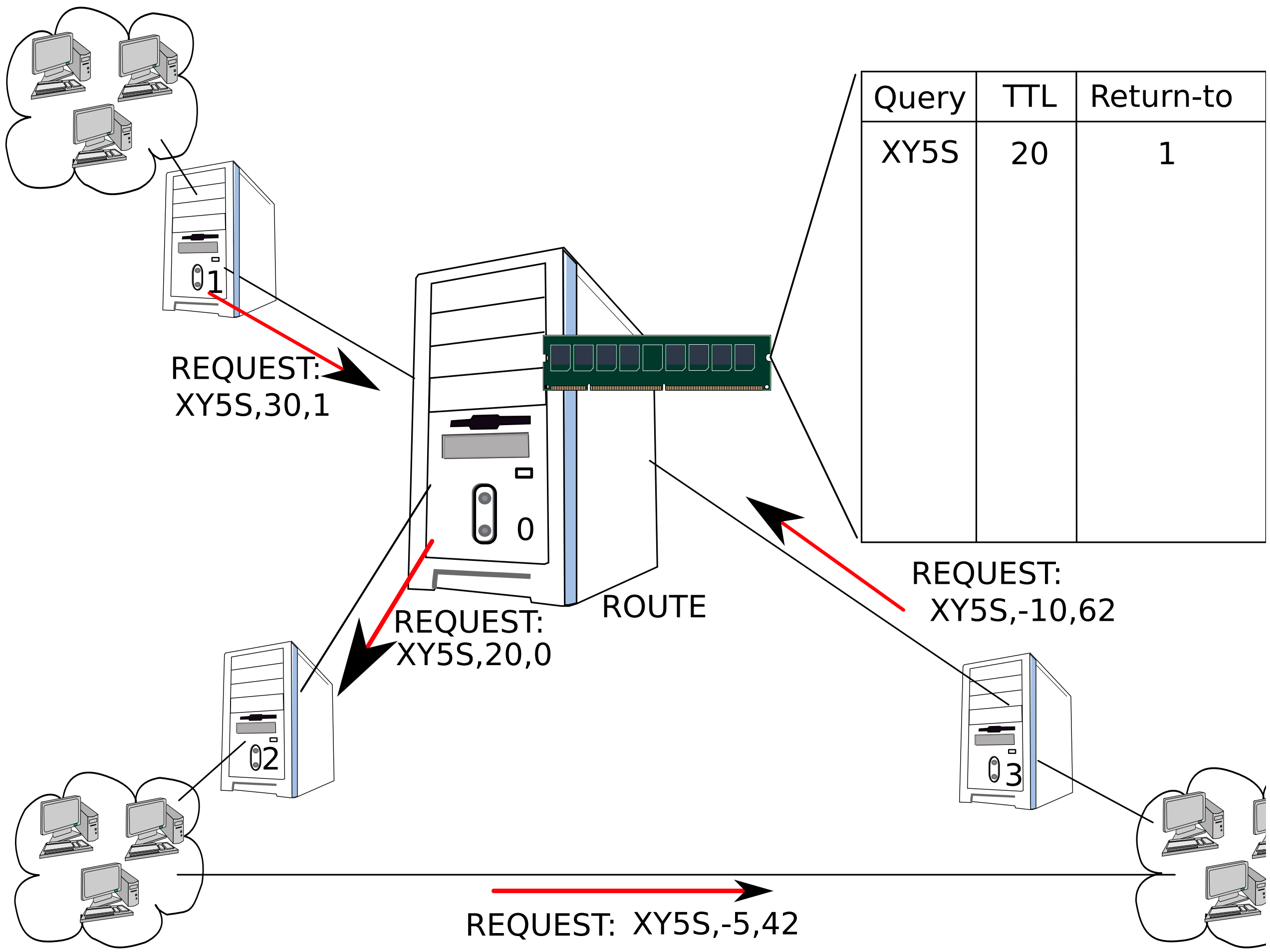
GAP illustrated (6/9)



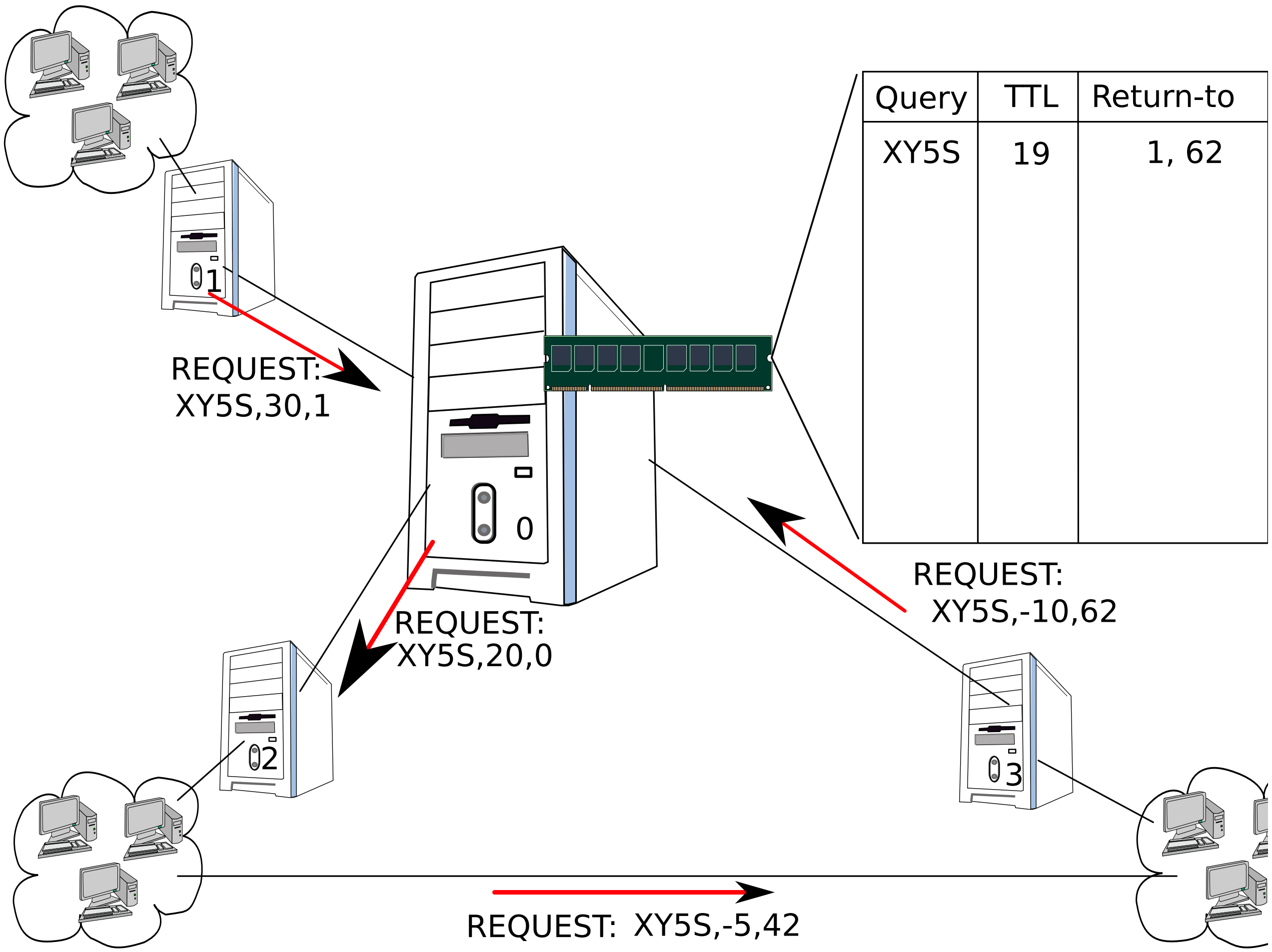
GAP illustrated (7/9)



GAP illustrated (8/9)



GAP illustrated (9/9)



GAP: Searching

Searching in GNUnet comes naturally from GNUnet's *best effort* paradigm:

- receive query, drop if busy
- indirect query if not too busy
- forward query if not very busy
- perform local lookup, send reply if not too busy
- introduce random delays

GAP: efficient or anonymous

When a node M processes a query from A, it can choose:

- to how many other nodes C_i should receive the query
- to tell C_i to send the reply directly to A
- to send a reply if content is available

GAP can take short cuts

If a node forwards a query preserving the identity of the originator, it may *expose* the actual initiator to the responder. This is ok:

- Next hop has still no certainty that the exposed predecessor is not routing for somebody else
- Same argument holds for the other direction

Costs and benefits of short-cuts

By preserving the previous sender of the query when the short-cutting peer forwarded the query:

- the peer has exposed its own routing behavior for this message, reducing the set of messages it can use to hide its own traffic
- the peer has gained performance (bandwidth) since it does not have to route the reply

GAP: Making a good call!

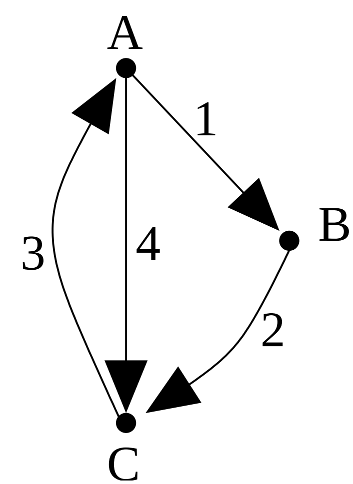
In GAP, a node decides to forward a query based on the current load. Thus:

- if the load is low, the node maximizes the indirected traffic and thus its anonymity
- if the load is high, the node is already covered in terms of anonymity and it reduces its load (does not have to route the replies) by forwarding
- if the load is far too high, the node just drops packets.

GAP: individual trade-offs

From this realization, we can motivate GNUnet's anonymity policy:

- indirect when idle,
- forward when busy,
- drop when very busy.



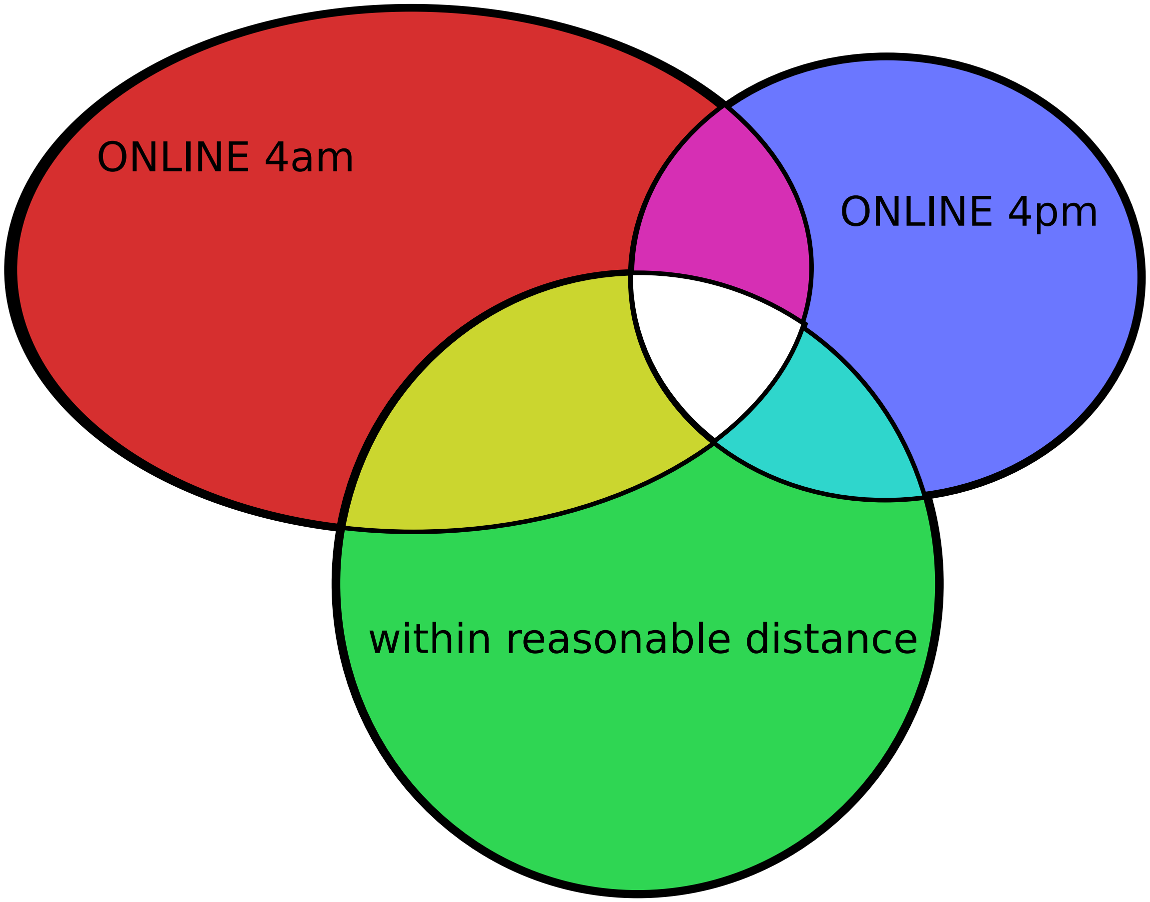
If we are indirecting lots of traffic, we don't need more to hide ourselves and can be *more efficient*!

GAP is unreliable

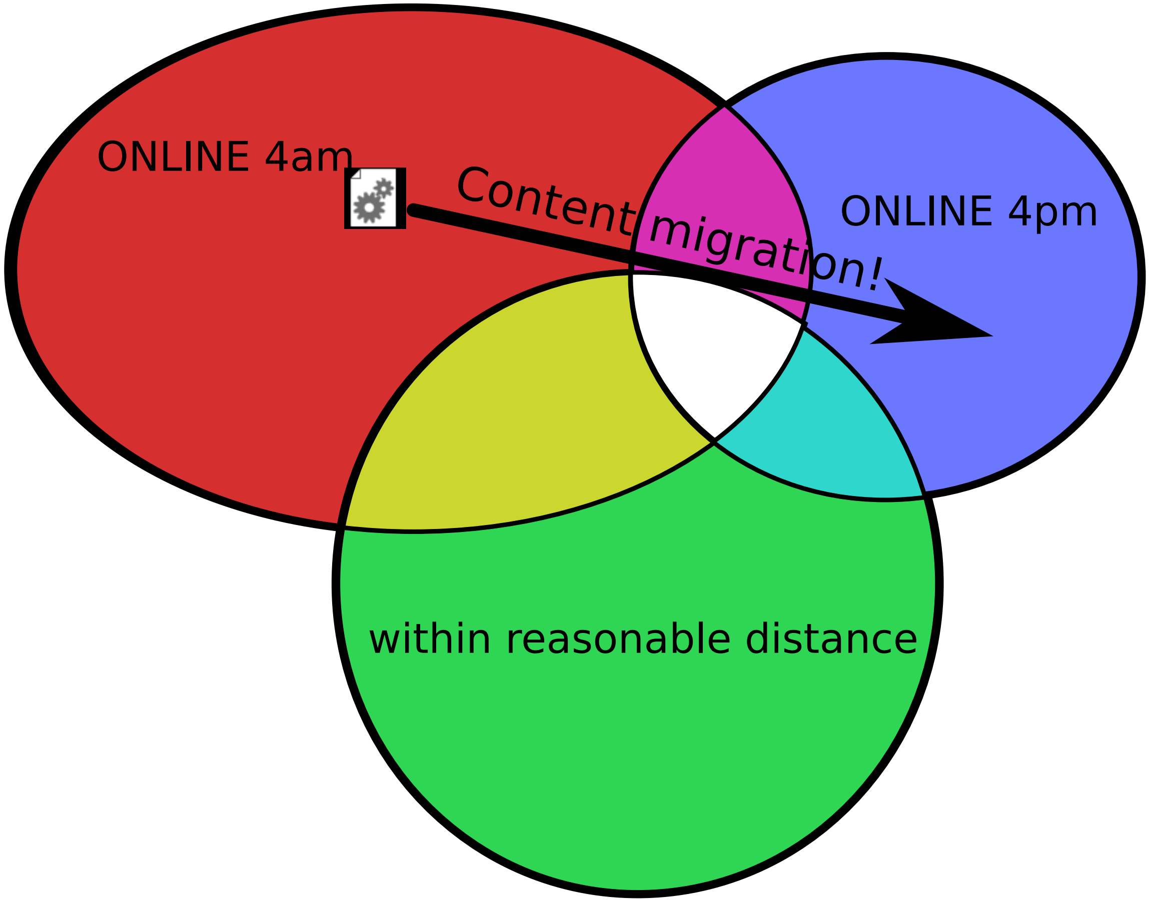
Unlike all other anonymous protocols, GAP is unreliable and has best-effort semantics:

- packets can be lost, duplicated or arrive out-of-order
- nodes can act more randomly and adjust to load
- application layer is responsible for adding reliability

Attacks: Partitioning (1/2)



Attacks: Partitioning (2/2)



GAP: Traffic Analysis?

A powerful adversary doing traffic analysis sees:

- encrypted packets
- unlinkable queries or replies at collaborating nodes
- random delays, unpredictable packet drops
- unpredictable packet duplication (send query to multiple hosts, send reply (!) to multiple hosts)
- only a small part of the network's topology since no routing information is exchanged

GAP: Attack?

So how would you attack GAP?

GAP: Conclusion

GAP is an efficient scheme that can achieve:

- any degree of anonymity based on the bandwidth available to the user compared to the adversary
- scalability because busy nodes can increase thoughput without compromising anonymity (of the node itself or other nodes)

Economics

R. Dingledine and P. Syverson wrote about *Open Issues in the Economics of Anonymity*:

- Anonymity requires introducing inefficiencies, who pays for that?
- The anonymizing server that has the best reputation (performance, most traffic) is presumably compromised.
- Providing anonymity services has economic disincentives (DoS, legal liability)
- One person may create and control several distinct online identities.

HashCash

Adam Back proposed *HashCash* as a solution to stop unsolicited mass E-mailing (also known as spam). Key idea:

- the sender pays per E-mail
- instead of money, use CPU time

HashCash: protocol

- In order to send an E-mail, the sender must find a collision in a hashcode.
- The hashcode can be provided by the receiver (challenge) or be derived from the E-mail with the receiver address and time for a non-interactive version.
- The number of bits that must match in the two hashcodes can be used to make it more or less expensive for the sender.

HashCash: problems

- Cost applies also for legitimate mass-mailings (aka mailinglists)
- CPU time is wasted
- Cost must be adjusted to match current CPUs, thus the protocol never benefits as better hardware becomes available.

HashCash

Why did it not get adopted?

Reputation

R. Dingledine, N. Mathewson and P. Syverson wrote about *Reputation in Privacy Enhancing Technologies*:

- Reputation is a way to track past performance and reward (Freehaven: you stored 1k for a week, I store 7k for a day).
- If reputation is global, claims must be verified, which can be very hard.
- If reputation is local, servers must *risk* resources to new nodes to keep the network open; vulnerability: "screw every server once" attack

Reputation: Musings

R. Dingledine, N. Mathewson and P. Syverson dream on:

- Reputation as Currency? Transitivity?
- Does reputation expire?
- Multiple currencies and convertability?
- Where does currency come from?

Trust *yourself*

C. Grothoff proposed an *Excess Based Economy*:

- use trust instead of money
- but trust no one except your resource allocation algorithm

Common Problems

- No accounting: easy to mount DoS attack
- Centralization
- Lack of acceptance for micropayments
- Patents

Excess Based Economy: Goals

- Reward contributing nodes with better service
- Detect attacks:
  - detect flooding,
  - detect abuse,
  - detect excessive free-loading, but
  - allow *harmless* amounts of free-loading

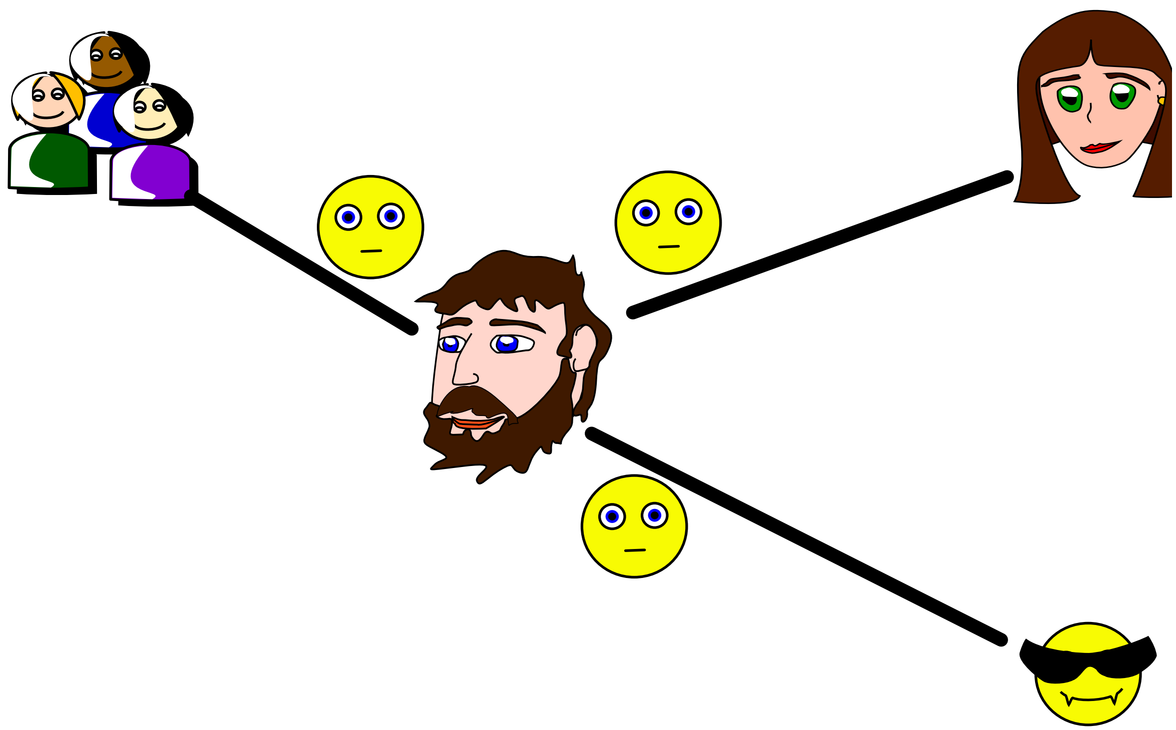
Excess Based Economy: Requirements

- No central server.
- No trusted authority.
- Everybody else is malicious and violates the protocols.
- Everybody can make-up a new identity at any time.
- New nodes should be able to join the network.

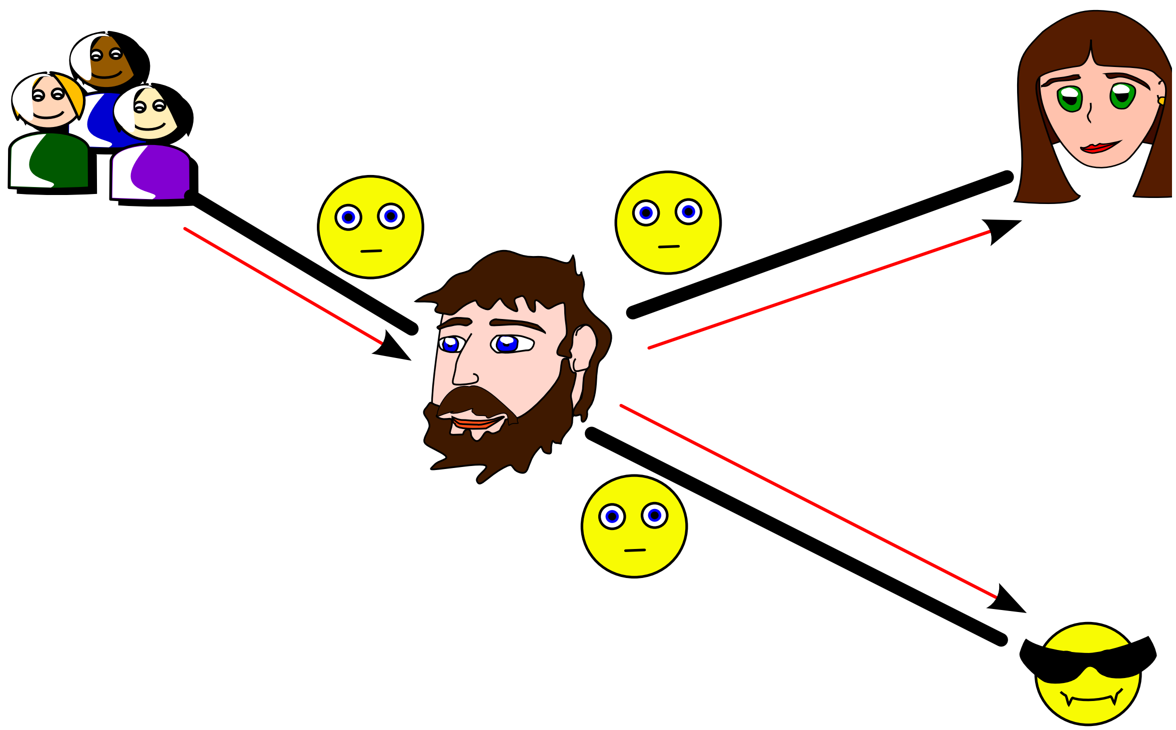
Excess Based Economy: Human Relationships

- We do not have to *trust* anybody to form an opinion.
- Opinions are formed on a one-on-one basis, and
- may not be perceived equally by both parties.
- We do *not* charge for every little favour.
- We *are* grateful for every favour.
- There is no guarantee in life, in particular Alice does not have to be kind to Bob because he was kind to her.

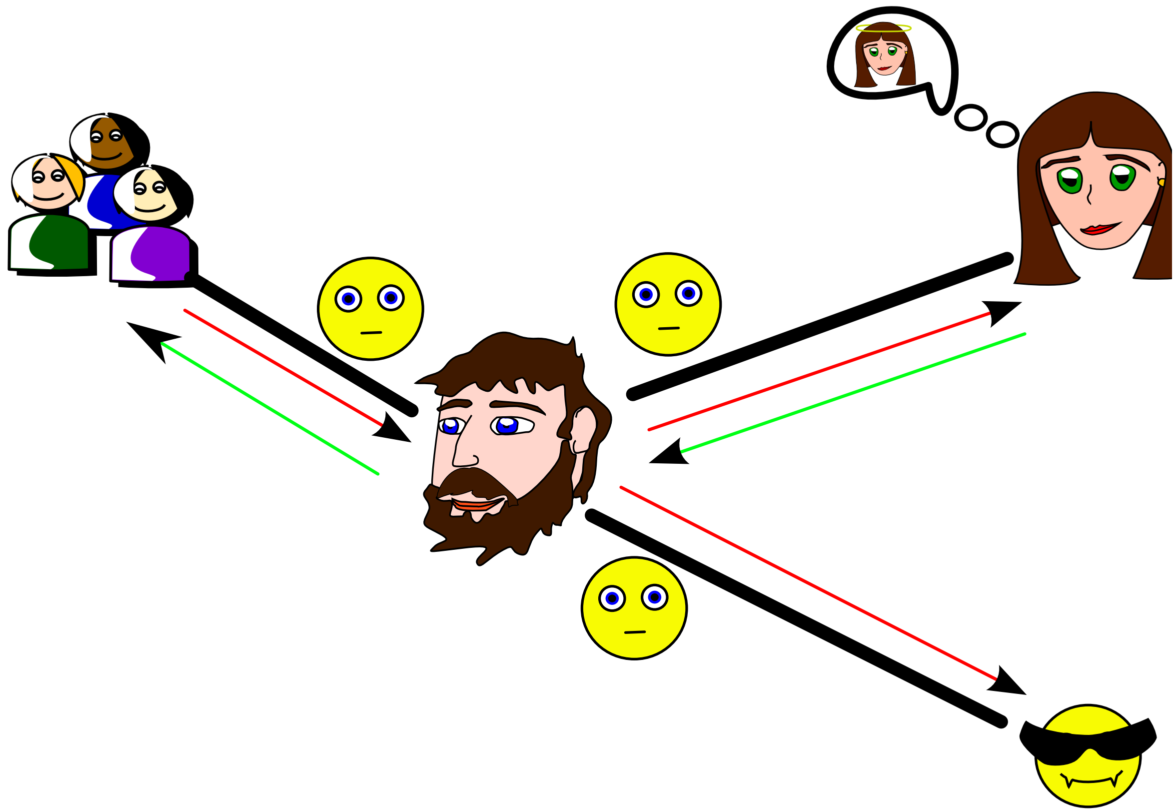
Excess-based Economy Illustrated (1/8)



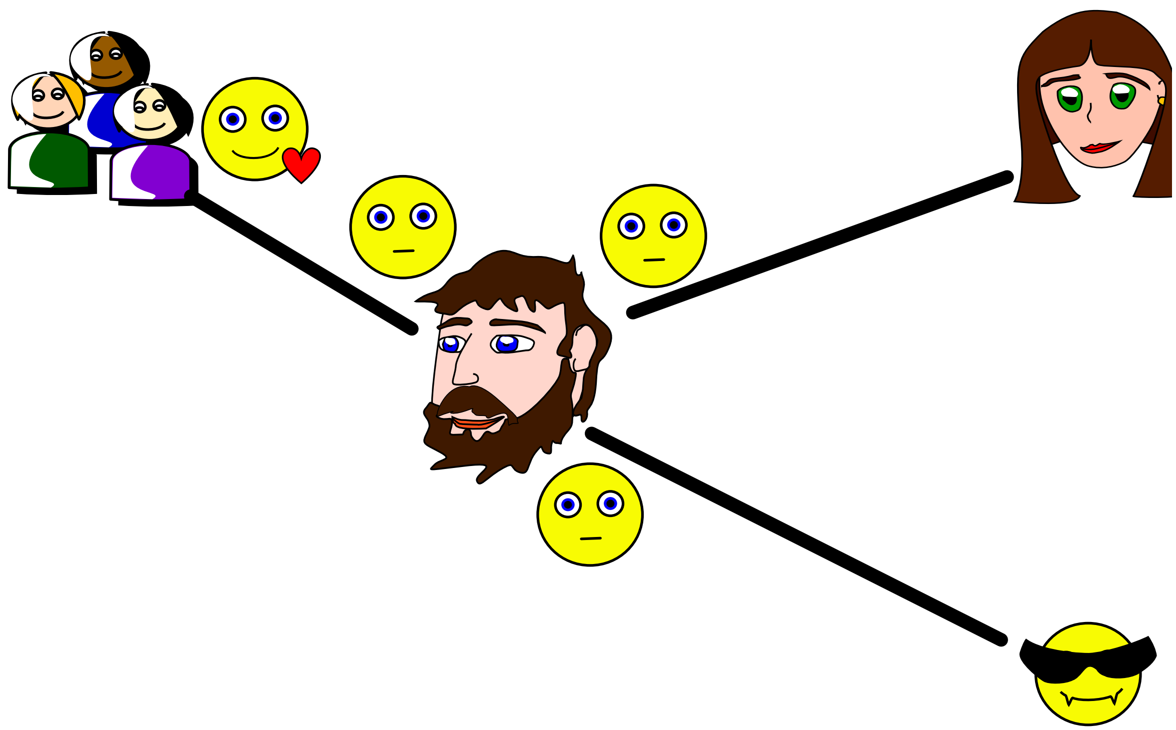
Excess-based Economy Illustrated (2/8)



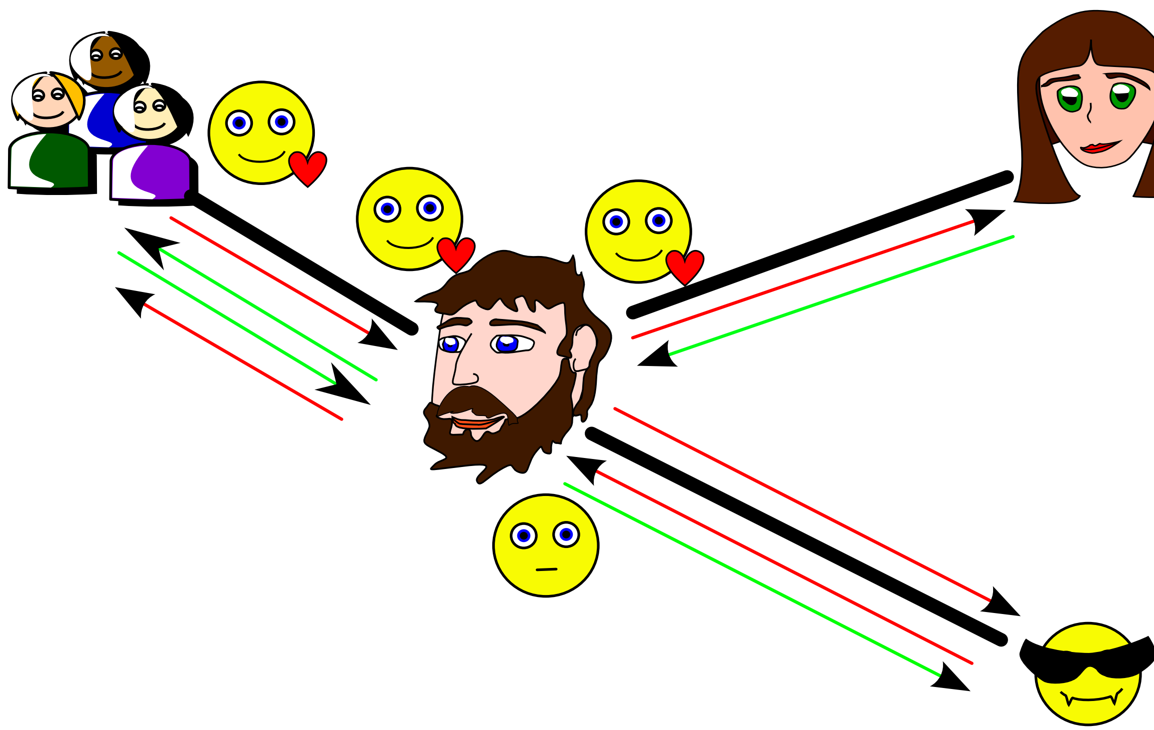
Excess-based Economy Illustrated (3/8)



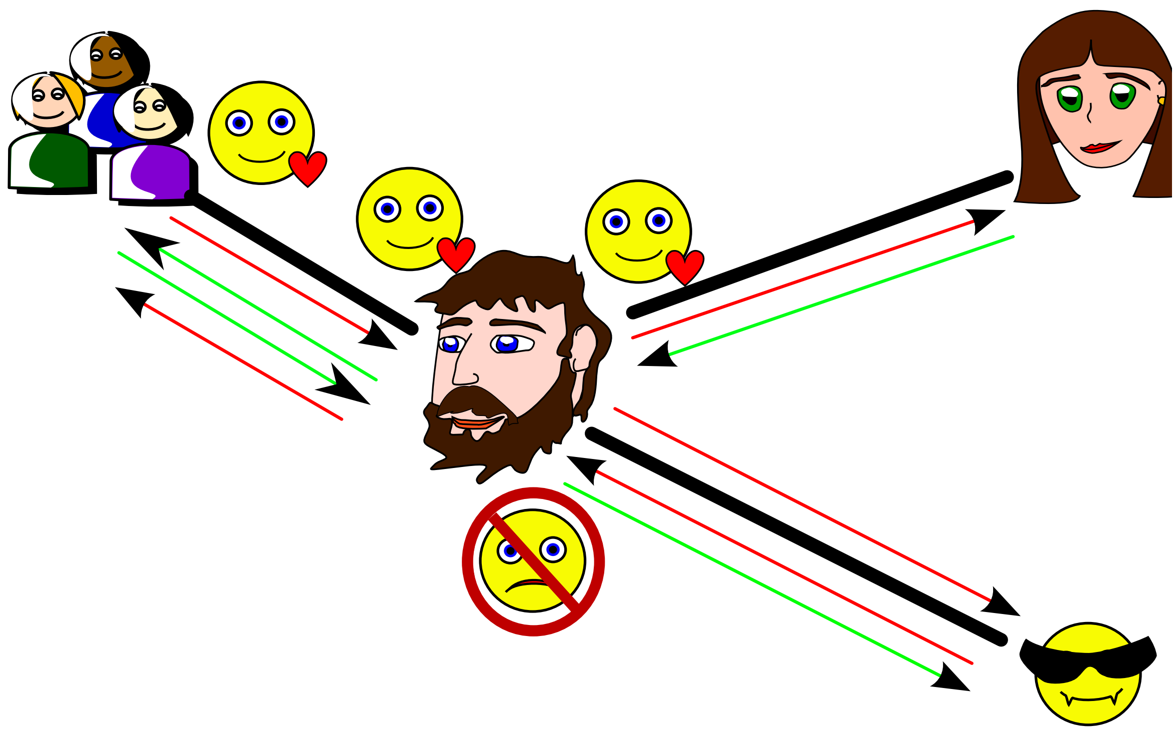
Excess-based Economy Illustrated (4/8)



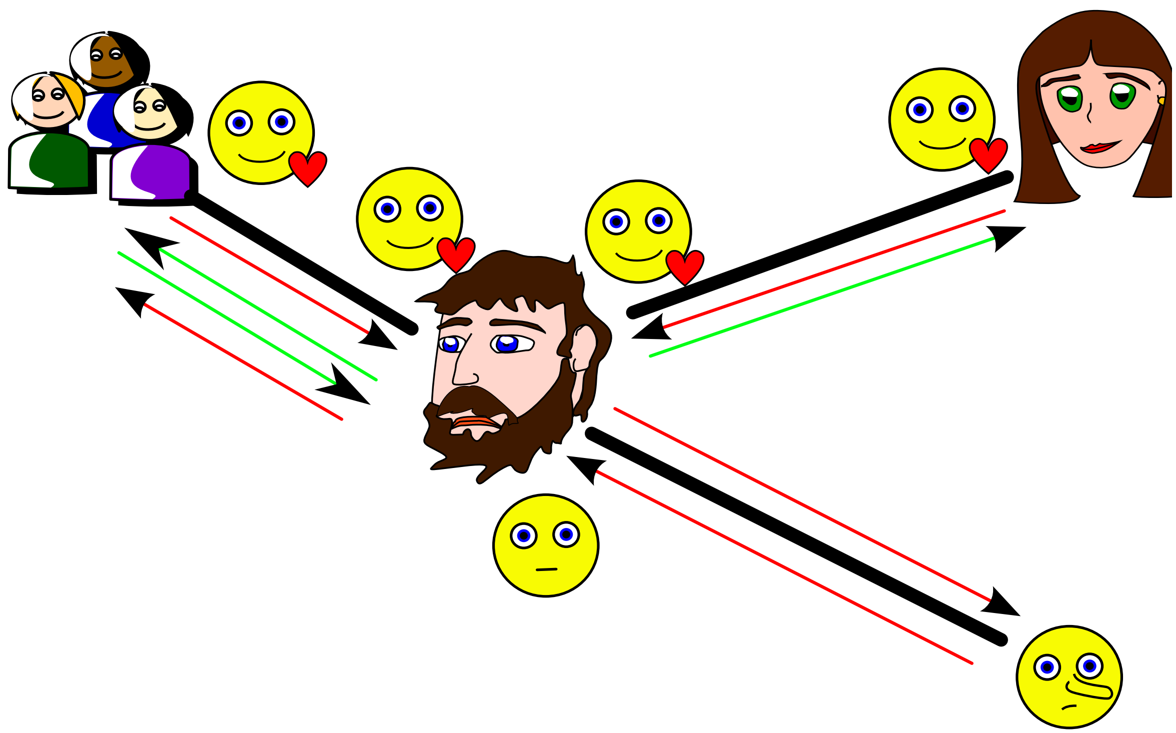
Excess-based Economy Illustrated (5/8)



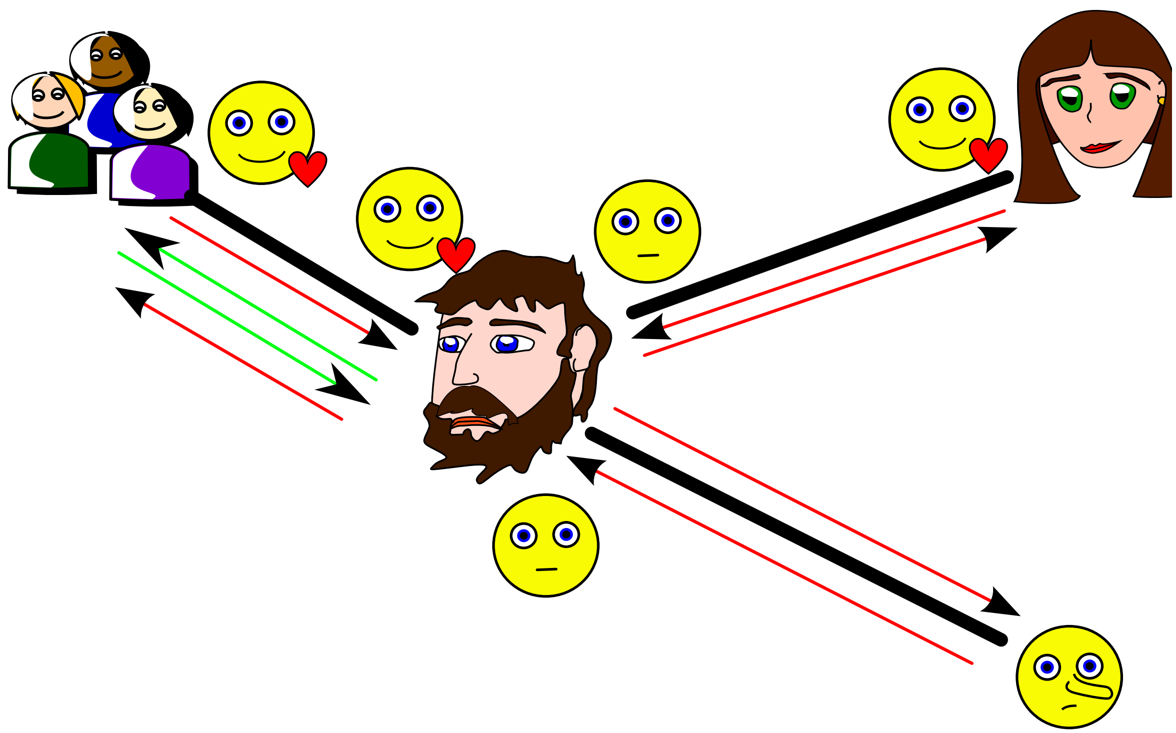
Excess-based Economy Illustrated (6/8)



Excess-based Economy Illustrated (7/8)



Excess-based Economy Illustrated (8/8)



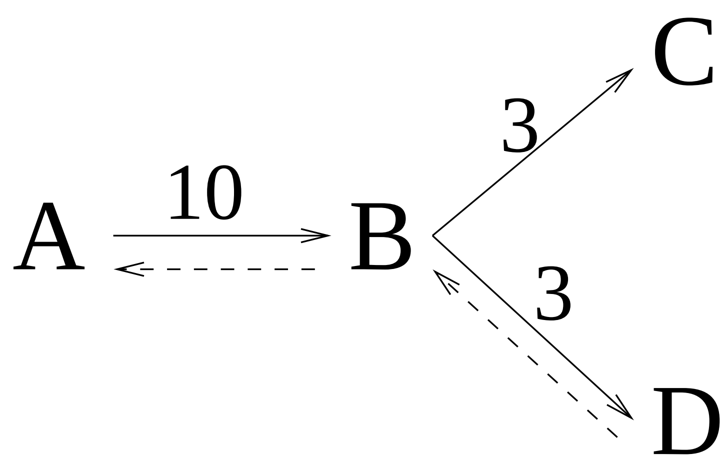
Excess-based Economy

GNUnet's economy is based on the following principals:

- if you are *idle*, doing a favour for free does not cost anything;
- if somebody does you a favour, remember it;
- if you are *busy*, work for whoever you like most, but remember that you paid the favour back;
- have a *neutral* attitude towards new entities;
- never dislike anybody (they could create a new identity anytime).

Excess Based Economy: Transitivity

If a node acts on behalf on another, it must ensure that the sum of the charges it may suffer from other nodes is lower than the amount it charged the sender:



Excess Based Economy: Open Issues

- If a node is idle, it will not charge the sender; if a node delegates (indirects), it will use a lower priority than the amount it charged itself; if an idle node delegates, it will always give priority 0. A receiver can not benefit from answering a query with priority 0.
- If the priority is 0, content will not be marked as valuable.
- under heavy use and long attacks, all trust may disappear

Excess Based Economy: Achievements

We have presented an economic model, that:

- solves the problem of initial accumulation
- does not rely on trusted entities
- can be used for resource allocation
- requires link-to-link authenticated messages, but no other cryptographic operations
- does not require a global view of the transaction and can thus be used with GAP

Economy: Requirements for Encoding

- Need content encoding that makes cheating not viable!

Encoding Data for File-Sharing

- Requirements
- Content encoding
- Support for searching

Problems with Other Encoding Mechanisms

- Content distributed in plaintext (e.g. gnutella) facilitates censorship and may void deniability
- Content must be inserted into the network and is then stored twice, in plaintext (by the originator) and encrypted (by the network – e.g. Freenet)
- Independent insertions of the same file result in different copies in the network (e.g. Publius)
- Verification of content integrity can only occur after download is complete (most systems)

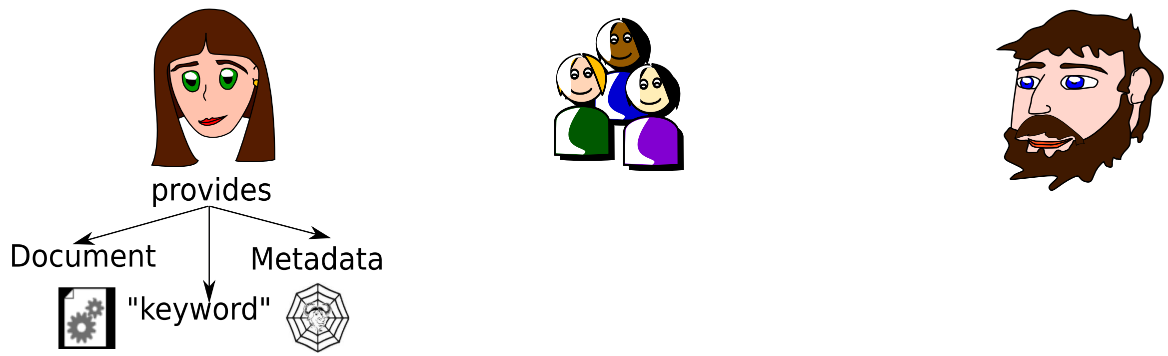
Properties of ECRS Encoding

- Breaks large files into small, uniform blocks
- Keeps storage (and bandwidth) overhead small
- Intermediaries *cannot* view content or queries
  $\Rightarrow$ Peers can send replies to queries and plausibly deny having knowledge of their contents
- Intermediaries *are* able to verify validity of responses
  $\Rightarrow$ Enables swarming, even in the presence of malicious peers trying to corrupt files

Properties of ECRS Implementation

- All operations performed by routers have expected runtime and memory use of $O(1)$
- All operations performed by responders have expected runtime $O(\log n)$ where n is the size of the datastore (under the assumption that a modern database with index can do lookups in $O(\log n)$)
- All receiver operations have (amortized) runtime $O(n)$ where n is the size of the result set or the size of the file; memory use for files of size n is $O(\log n)$ with a tiny constant

ECRS Illustrated (1/9)



How to get the Keywords?

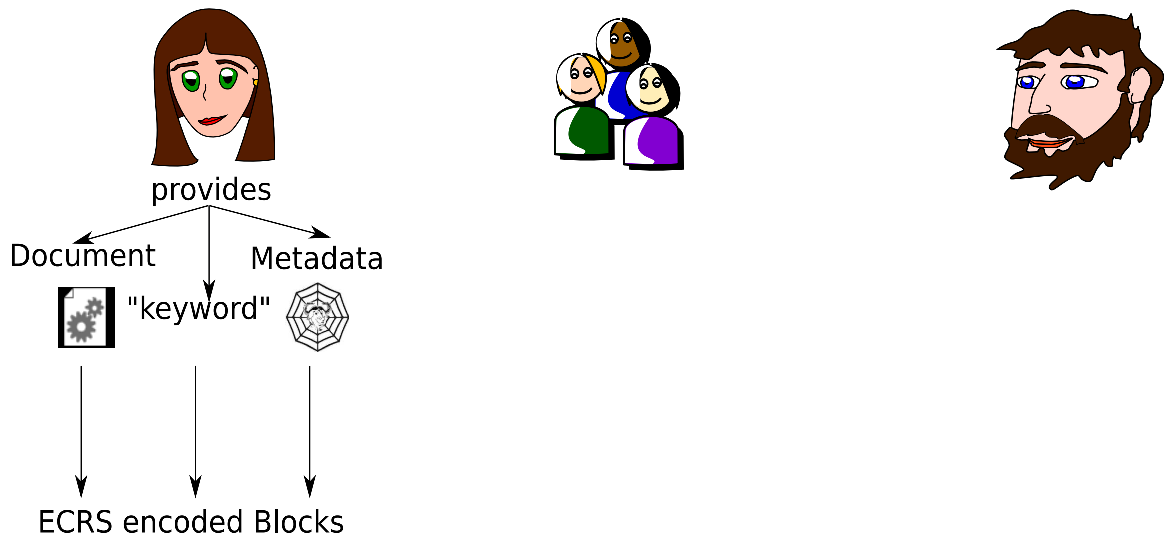
- Automatically extract metadata!
- Many file formats
  $\Rightarrow$ pluggable architecture

Developed as seperate library:

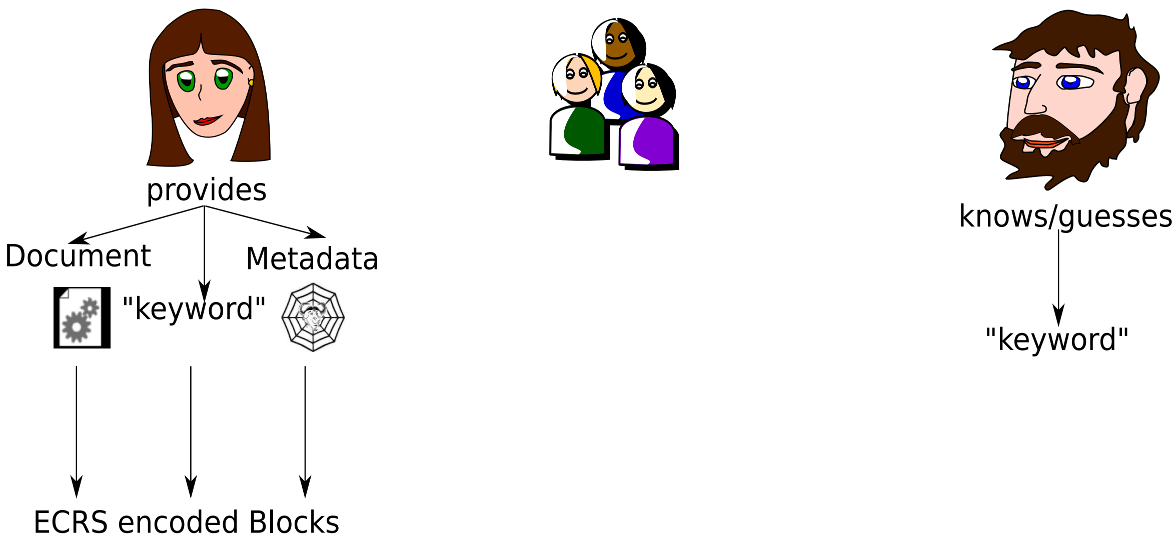
`http://gnunet.org/libextractor/`

`http://gnunet.org/libextractor/demo.php3`

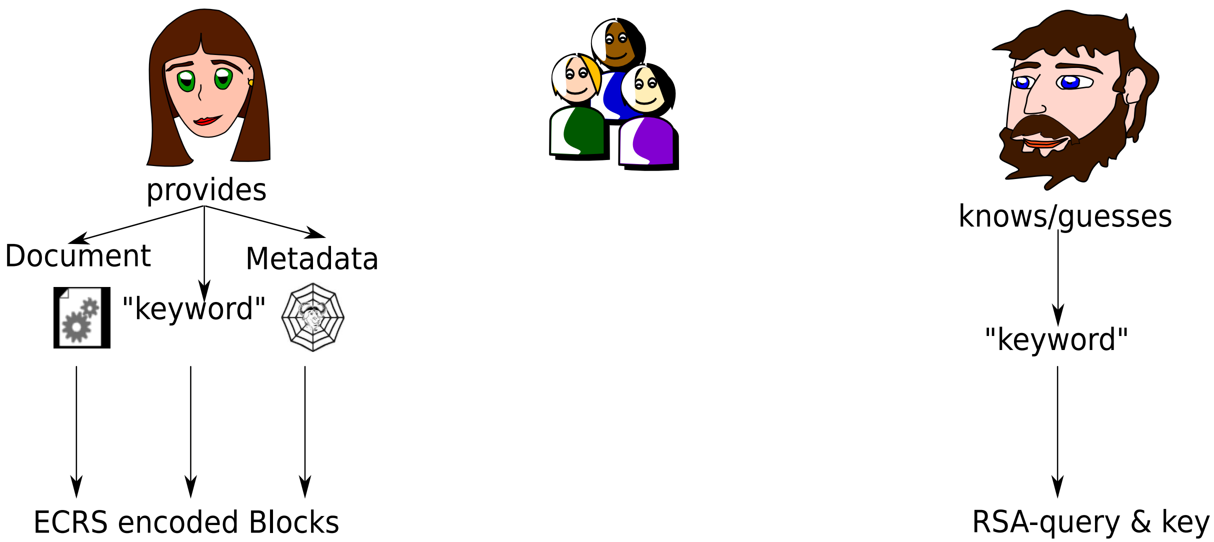
ECRS Illustrated (2/9)



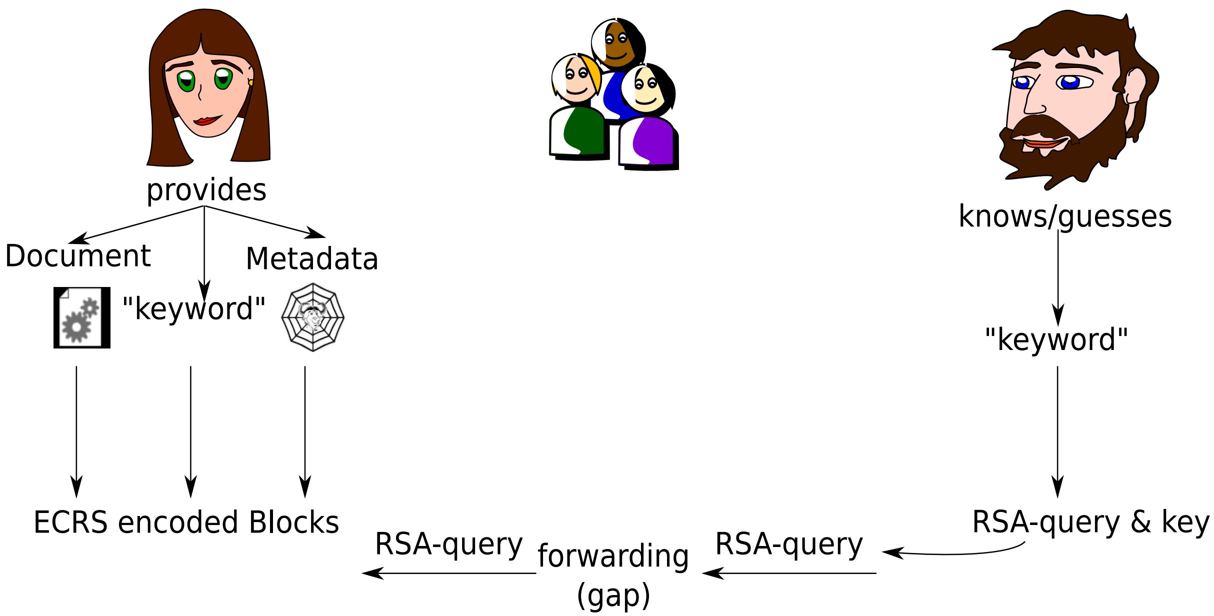
ECRS Illustrated (3/9)



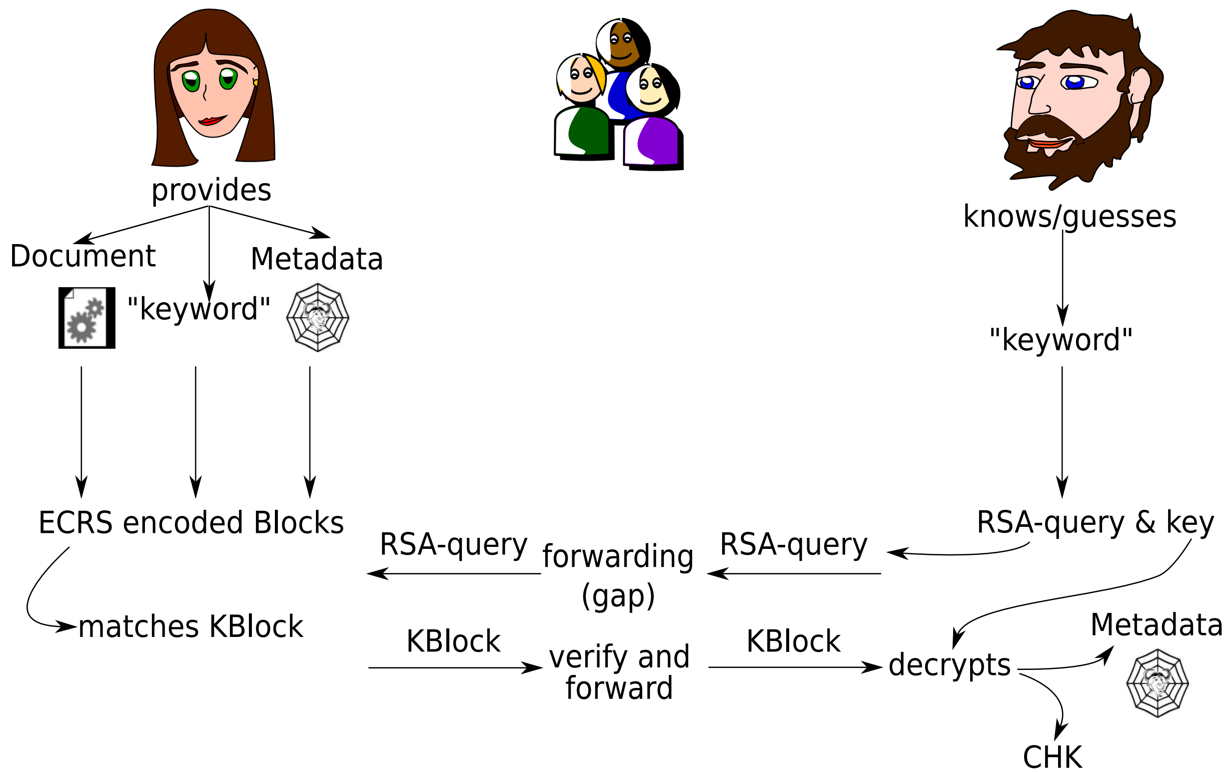
ECRS Illustrated (4/9)



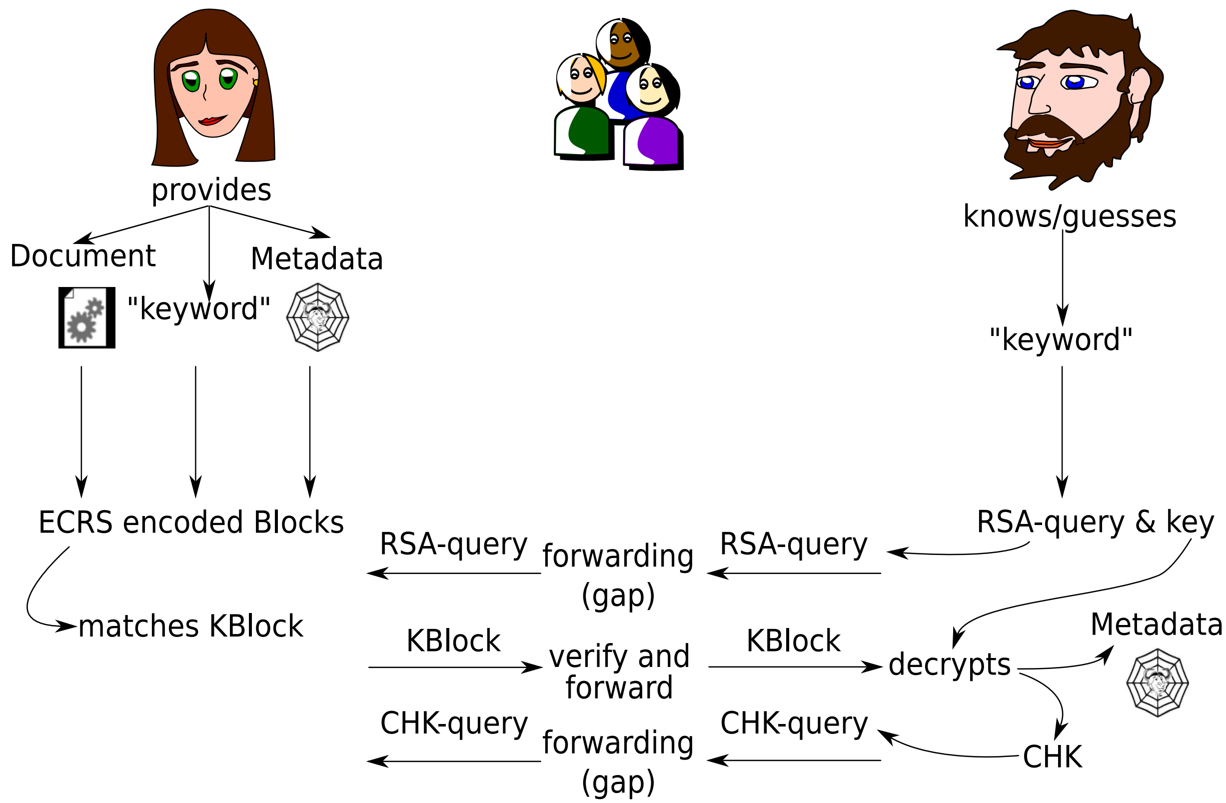
ECRS Illustrated (5/9)



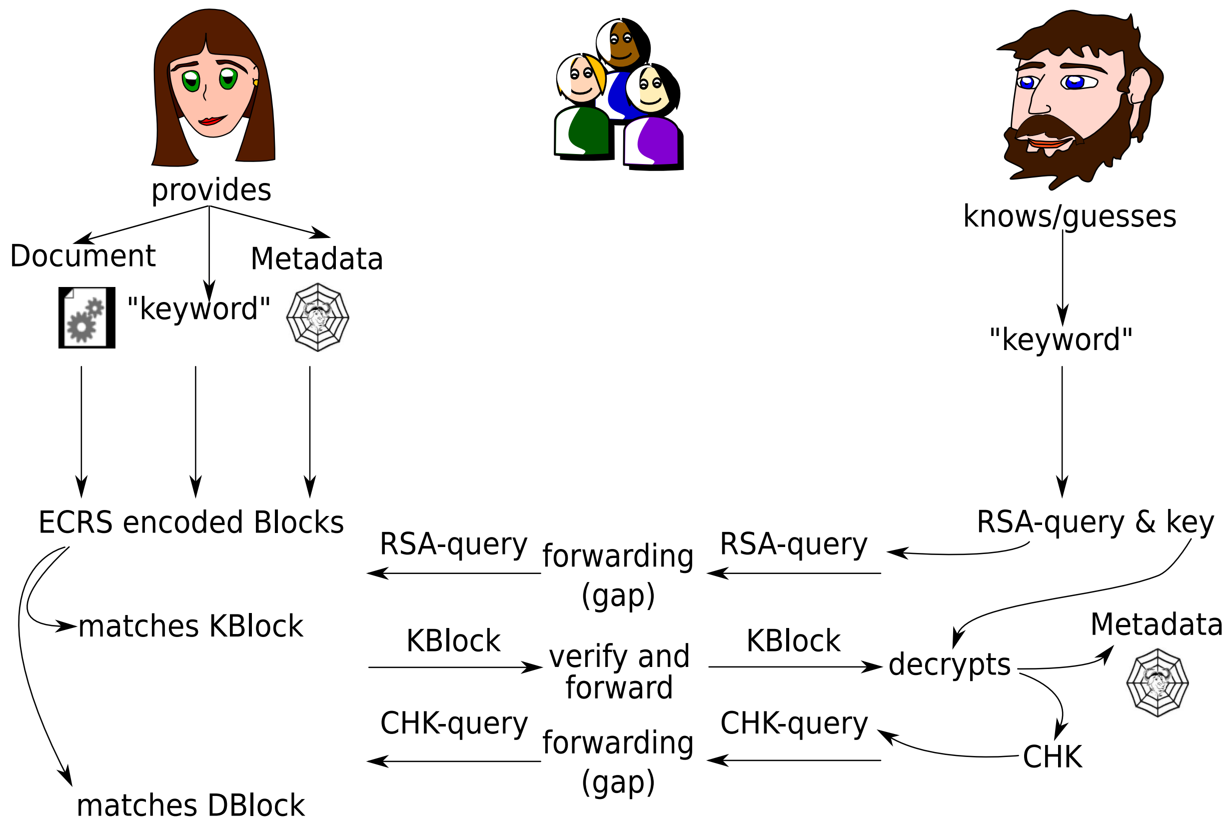
ECRS Illustrated (6/9)



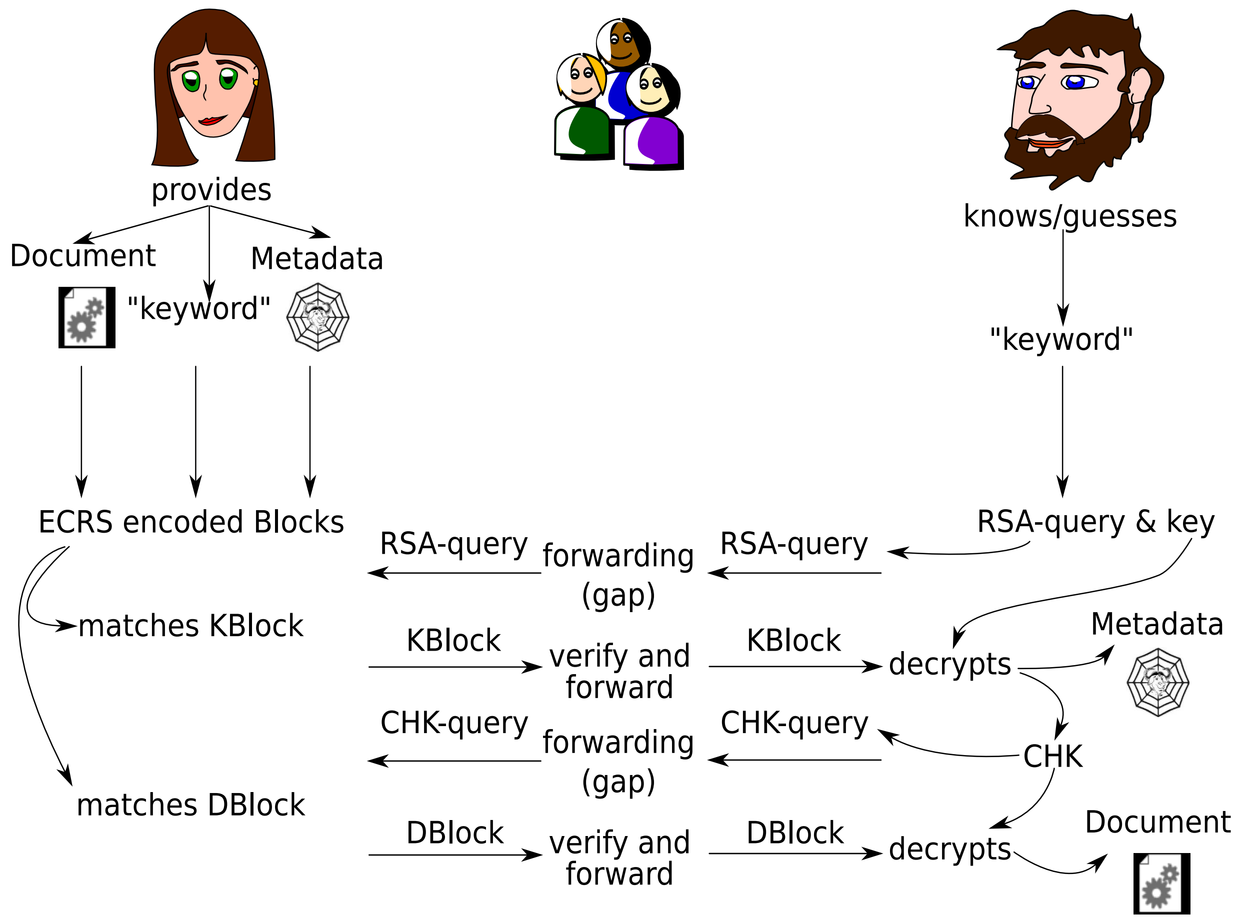
ECRS Illustrated (7/9)



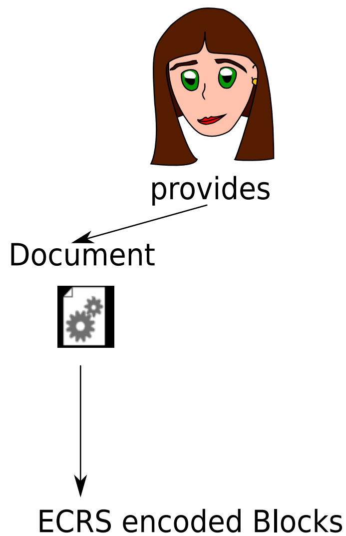
ECRS Illustrated (8/9)



ECRS Illustrated (9/9)

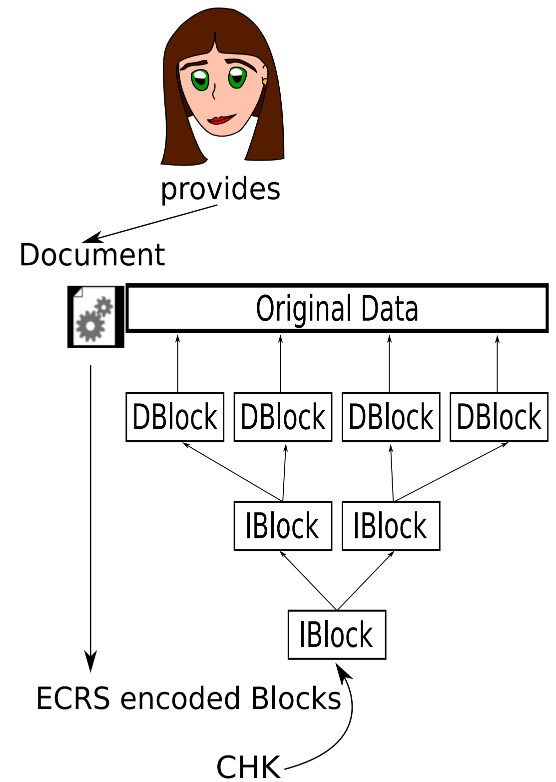


ECRS Details: Document Encoding



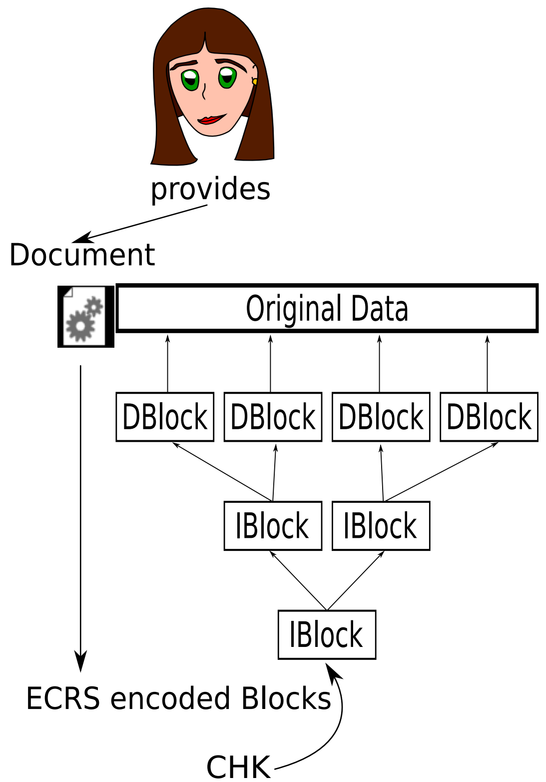
- Split content into 32k blocks B
- AES-256 encrypt B with $H(B)$
- Store $E_{H(B)}(B)$ under $H(E_{H(B)}(B))$
- Build tree containing up to 256 CHK pairs: $H(B), H(E_{H(B)}(B))$

ECRS Details: Document Encoding



- Split content into 32k blocks B
- AES-256 encrypt B with $H(B)$
- Store $E_{H(B)}(B)$ under $H(E_{H(B)}(B))$
- Build tree containing up to 256 CHK pairs: $H(B), H(E_{H(B)}(B))$

ECRS Details: Document Encoding

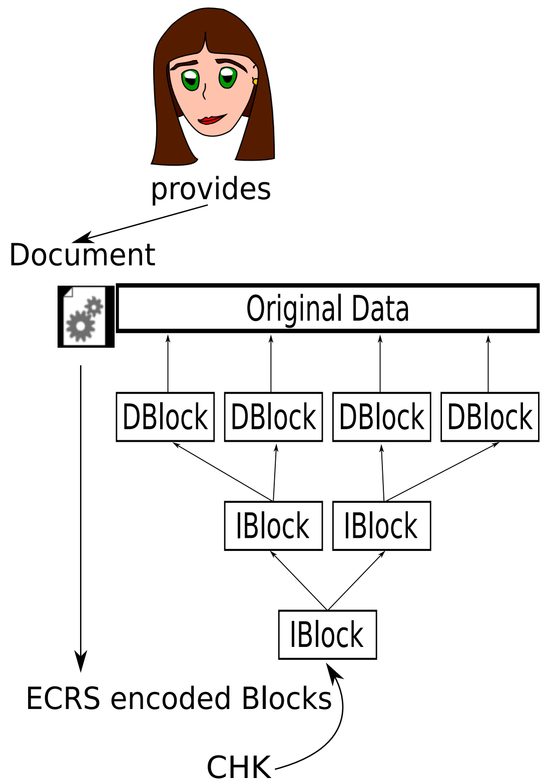


- Encryption of blocks independent of each other
- Inherent integrity checks
- Multiple (independent) insertions result in identical blocks
- Small blocksize makes traffic more uniform
  $\Rightarrow$ traffic analysis is harder

ECRS Details: Document Encoding Limitations

- If the exact data can be guessed... participating hosts can match the content.
  Intended to reduce storage costs!

ECRS Search Design Requirements



- Retrieve content with simple, natural-language keyword
- Guard against malicious hosts: prevent attackers from providing useless replies!
- Do not expose actual keyword used
- Do not expose CHK or metadata: encrypt CHK and metadata as well!

ECRS Searching: KBlocks

Let R be the (plaintext) metadata and CHK.

- For each keyword K use K to generate RSA key pair $(PRIV_K, PUB_K)$, store $E_{H(K)}(R), PUB_K$ signed with $PRIV_K$
- User searching also computes RSA key pair and sends query: $H(PUB_K)$
- Intermediates match PUB_K against $H(PUB_K)$ and verify signature

Benefits and Limitations of KBlocks

+ Malicious peer cannot learn R without guessing the keyword
+ Malicious peer must guess keyword to generate valid reply
+ Malicious peer cannot modify reply without being detected
- Cryptographic operations are quite expensive

Open Issues

- Multiple Search Results
- Content updates
- Approximate queries

The Multiple Search Result Problem

- Responder can not send “fake” response (ECRS)
- Responder can send same response again and again

⇒ No incentive to look for alternative responses!

⇒ First (few) responses to keyword spread far and wide, others will never be displayed!

⇒ Need to use creative keywords (but in that case, caching is much less effective!)

Solution (1/2)

- As part of the query, communicate what replies are not acceptable
- Can not include full replies (too big)

$\Rightarrow$ Use bloomfilter of hash codes of encrypted replies

Solution (2/2)

- Bloomfilter is probabilistic
- Even relatively generous bloomfilters would filter approximately $1{:}2^{10}$ valid replies
- Solution: add random 32-bit nounce to hash function, change nounce (sometimes) when repeating requests

$\Rightarrow$ False-positives less than $1{:}2^{42}$

Other Solutions to Search

- Directories
- Namespaces / Pseudonyms
- `SBlocks` are essentially cryptographically signed `KBlocks` (with possibly some additional information, such as pointers to updates)

Copyright