Research Topics in Networks and Distributed Systems

Arvind Krishnamurthy

University of Washington

Research Interests

- Peer-to-peer systems
- Network security
- Privacy systems & Censorship resistance
- Data center networks
- Distributed systems

P2P Systems

- *Decentralized* distribution model
- Hugely popular, *dozens* of file-sharing and streaming applications
- About *20%* of Internet users use P2P systems
- Responsible for significant Internet traffic

Napster

- Centralized database of which nodes has what files
  - Join: on startup, client contacts central server
  - Publish: client reports list of files to server
  - Search: query the server for which peers have a file
  - Fetch: get the file directly from the peer
- Pros: simple, search is O(1)
- What are the weaknesses?

Gnutella

- Basic idea: query flooding, no central state
  - Join: client contacts a few other nodes; these become its neighbors
  - Publish: N/A
  - Search: ask neighbors, who ask their neighbors, and so on; reply to sender when found
    - TTL (time-to-live) limits propagation
  - Fetch: get the file directly from peer
- What are the pros/cons? How can it be optimized?

Kazaa

- Supernode based query flooding
  - Join: on startup, client contacts a “supernode” ... may at some point become one itself
  - Publish: send list of files to supernode
  - Search: send query to supernode, supernodes flood query amongst themselves.
  - Fetch: get the file directly from peer(s); can fetch simultaneously from multiple peers

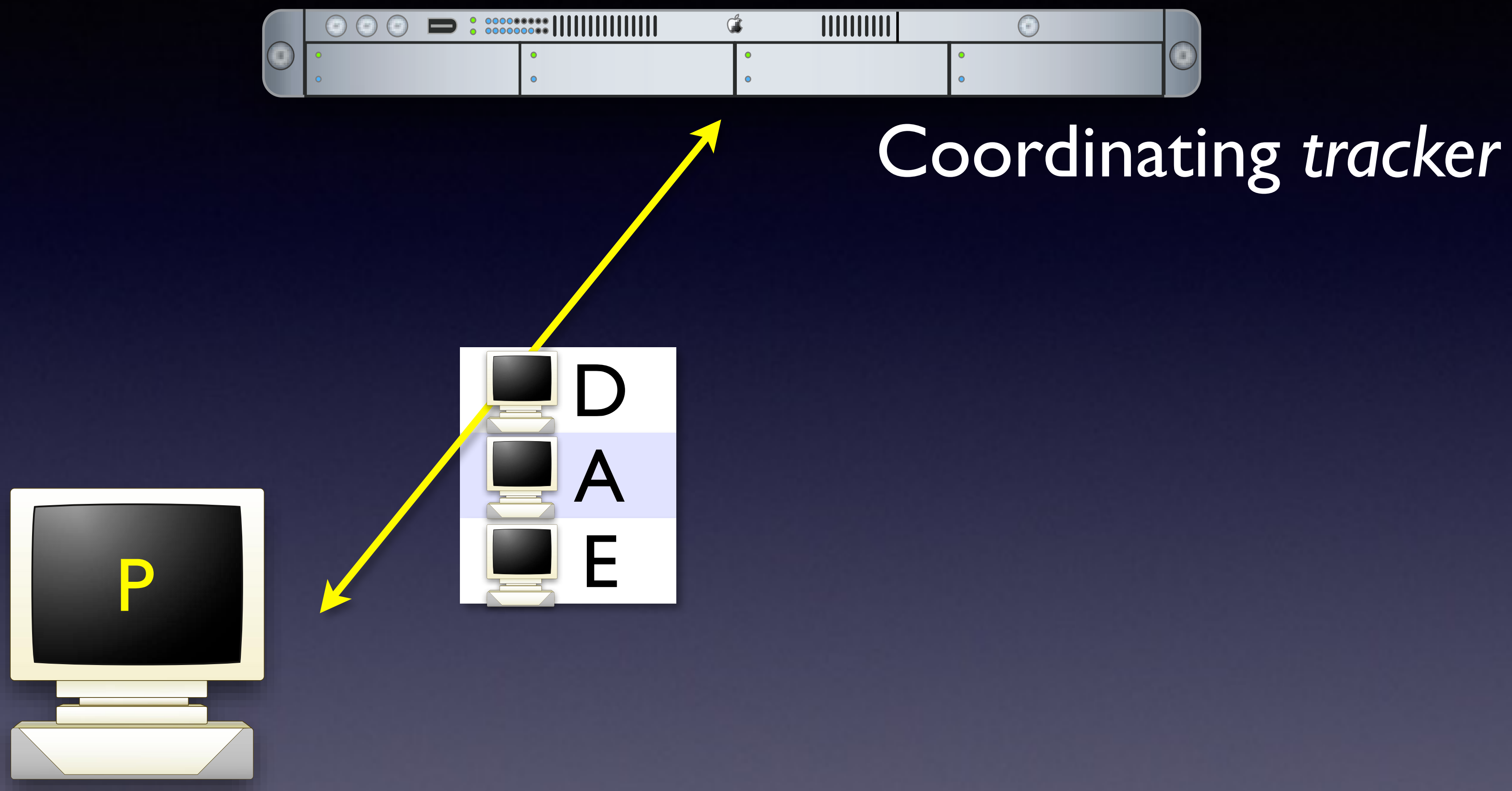
Evolution of P2P incentives

- Early P2P systems did not provide contribution incentives
  - 70% of Gnutella users *didn't share*
  - 50% of queries answered by 1% of hosts
- Subsequent designs:
  - "Incentive priorities" in Kazaa were spoofed
  - Centralized accounting (MojoNation) not adopted
- BitTorrent: explicit, decentralized contribution incentives

Incentives in BitTorrent

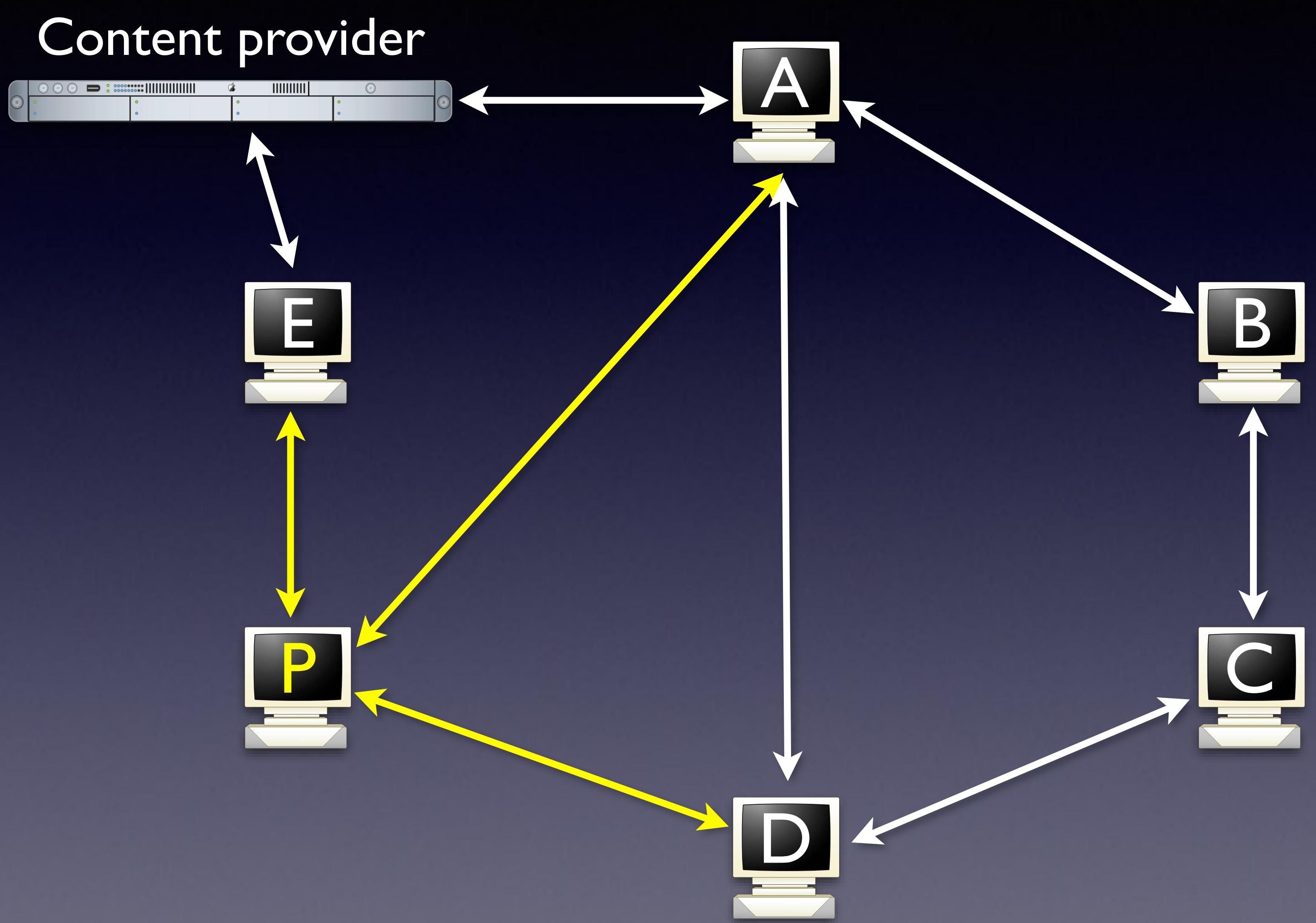
- A case study: *did BitTorrent get it right?*
- Can a strategic user game the system? – *Yes*
- Are BitTorrent's incentives strong? – *No*
- Can we design a system with persistent and strong incentives? – *Possible, but requires careful engineering*

BitTorrent overview

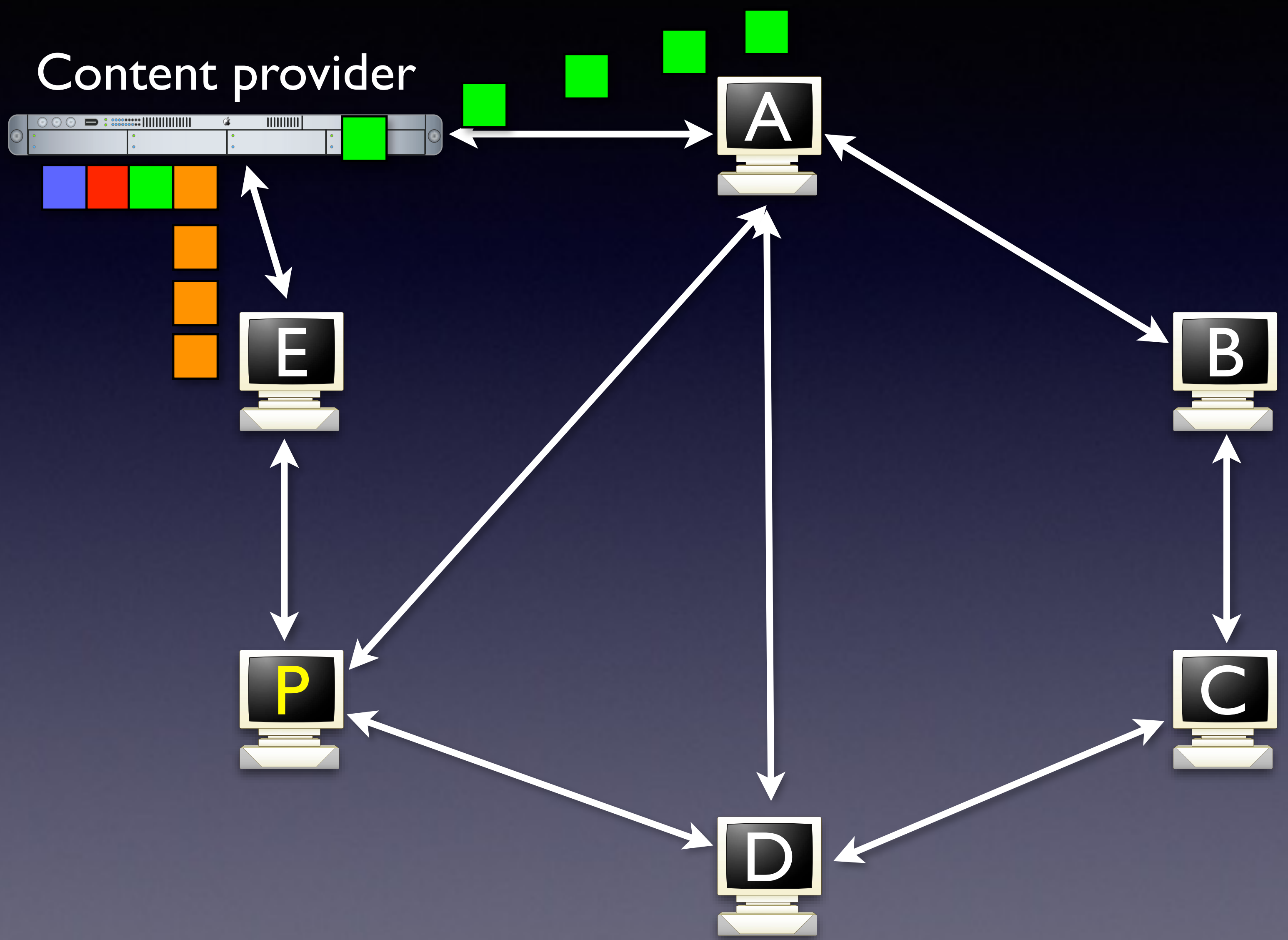


P joins the system by obtaining a random subset of current peers from a centralized coordinator

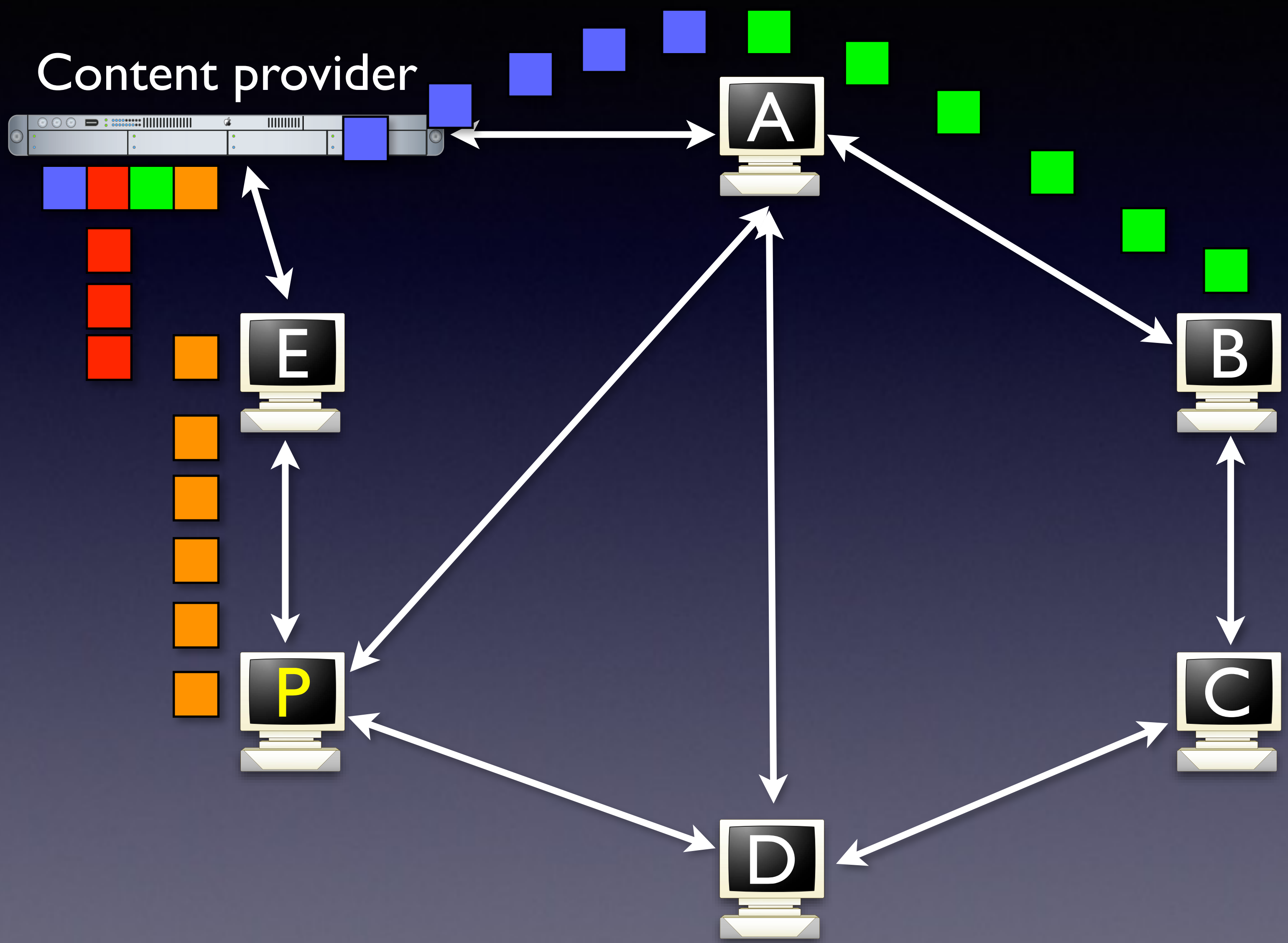
BitTorrent overview



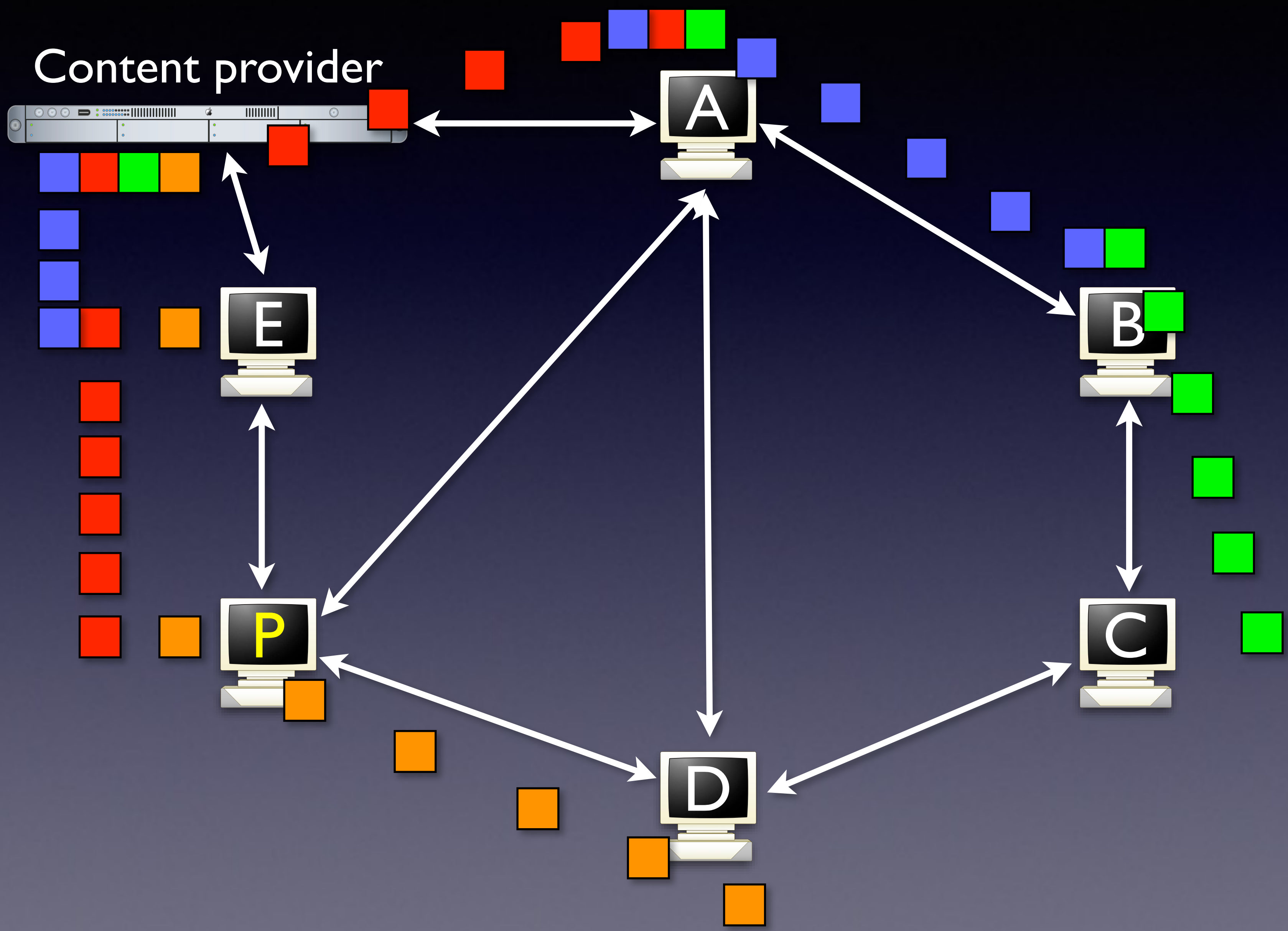
BitTorrent overview



BitTorrent overview



BitTorrent overview



Tit-for-tat in BitTorrent

Choosing *peers* and *rates*:

1. Sort peers by incoming data rate
2. Reciprocate with top *k*
   e.g., $k \propto \sqrt{\text{rate}}$
3. *Optimistically unchoke* one other peer
4. Send each peer selected an *equal split* of capacity

Peer	*Rate*	*Split*
A	17	15
C	13	15
D	8	
E	0	
F	0	15

If *k*=2, P reciprocates with A and C

45
Equal split = 45/3 = 15

Building *BitTyrant*

- *Key idea:* maximize return on investment (RoI)
  - strategic *peer* selection
  - strategic *upload rate* allocation

- *Cost:* upload rate to peer p, u_p
  Benefit: download rate from peer p, d_p

- BitTyrant dynamically estimates these rates each tit-for-tat round

Selecting peers & rates

Each TFT round, order and reciprocate with peers:

$$\underbrace{\frac{d_0}{u_0}, \frac{d_1}{u_1}, \frac{d_2}{u_2}, \frac{d_3}{u_3}, \frac{d_4}{u_4}}_{\text{choose } k \mid \sum_{i=0}^{k} u_i \leq cap}, \ldots$$

After each round, for each peer:

If peer reciprocates:

$d_p \leftarrow$ direct observation

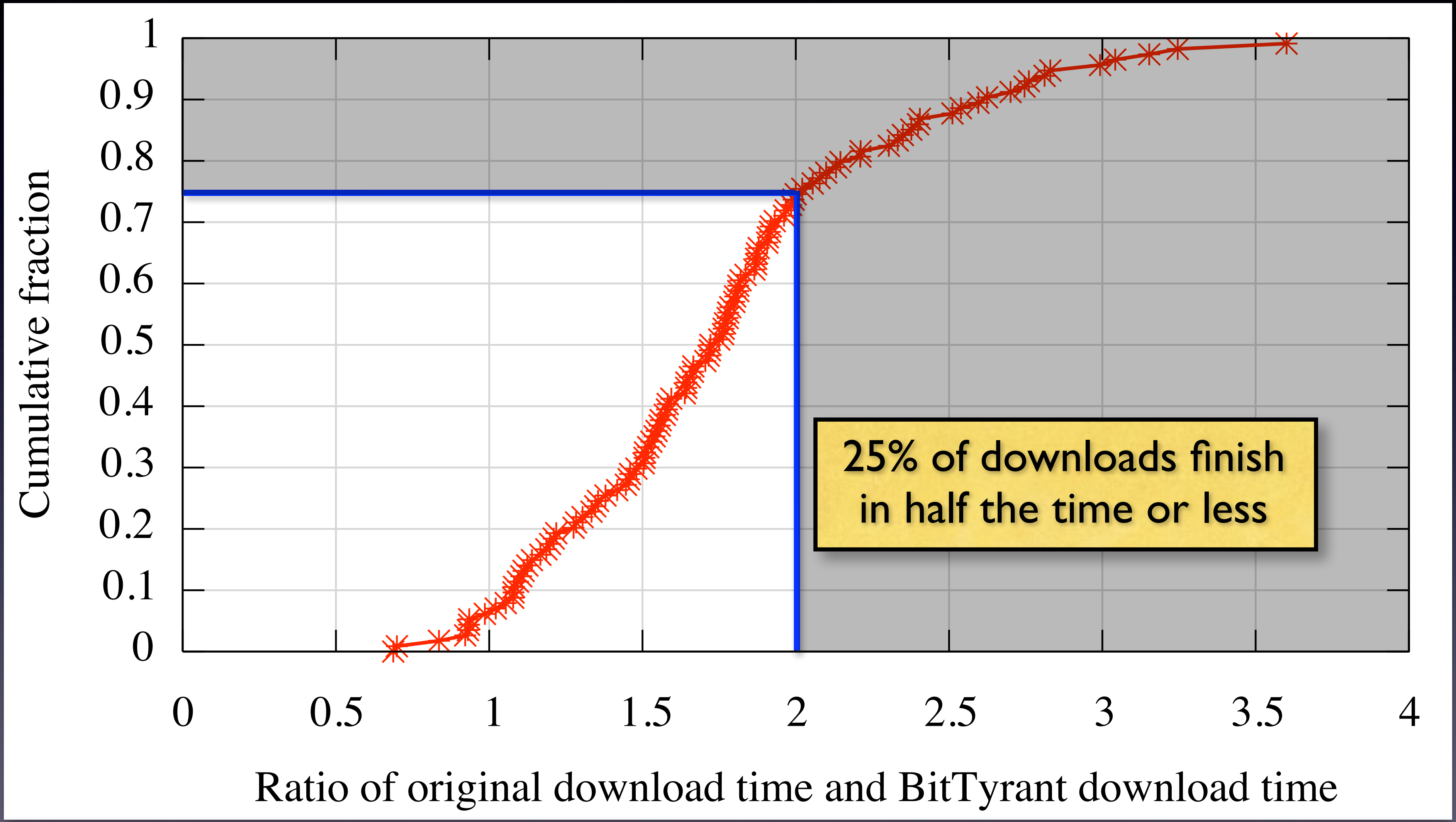
...and continues to do so:

Reduce u_p

No reciprocation:

Increase u_p

Swarms in the wild



BitTyrant improves performance in current swarms

Research Interests

- Peer-to-peer systems
- Network security
- Privacy systems & Censorship resistance
- Data center networks
- Distributed storage systems

The Internet is unsafe

- Problems in the Internet today:
  - Spam: 100 billion emails/day
  - DDoS attacks:
  - Click fraud: 20% of clicks are fraudulent
  - Phishing, identity theft, etc.

Botnets

- Botnets are often the underlying infrastructure
  - Network of compromised hosts
  - Controlled by attacker using state-of-the-art fault tolerant distributed mechanisms

Botnets not well understood

- Limited information on how they operate
- Most analysis is post-hoc
- Inconsistent information

> "25% of all Internet-connected computers are part of a botnet."
>
> Vint Cerf

> "Storm botnet has 50 million nodes."
>
> Sept 2007

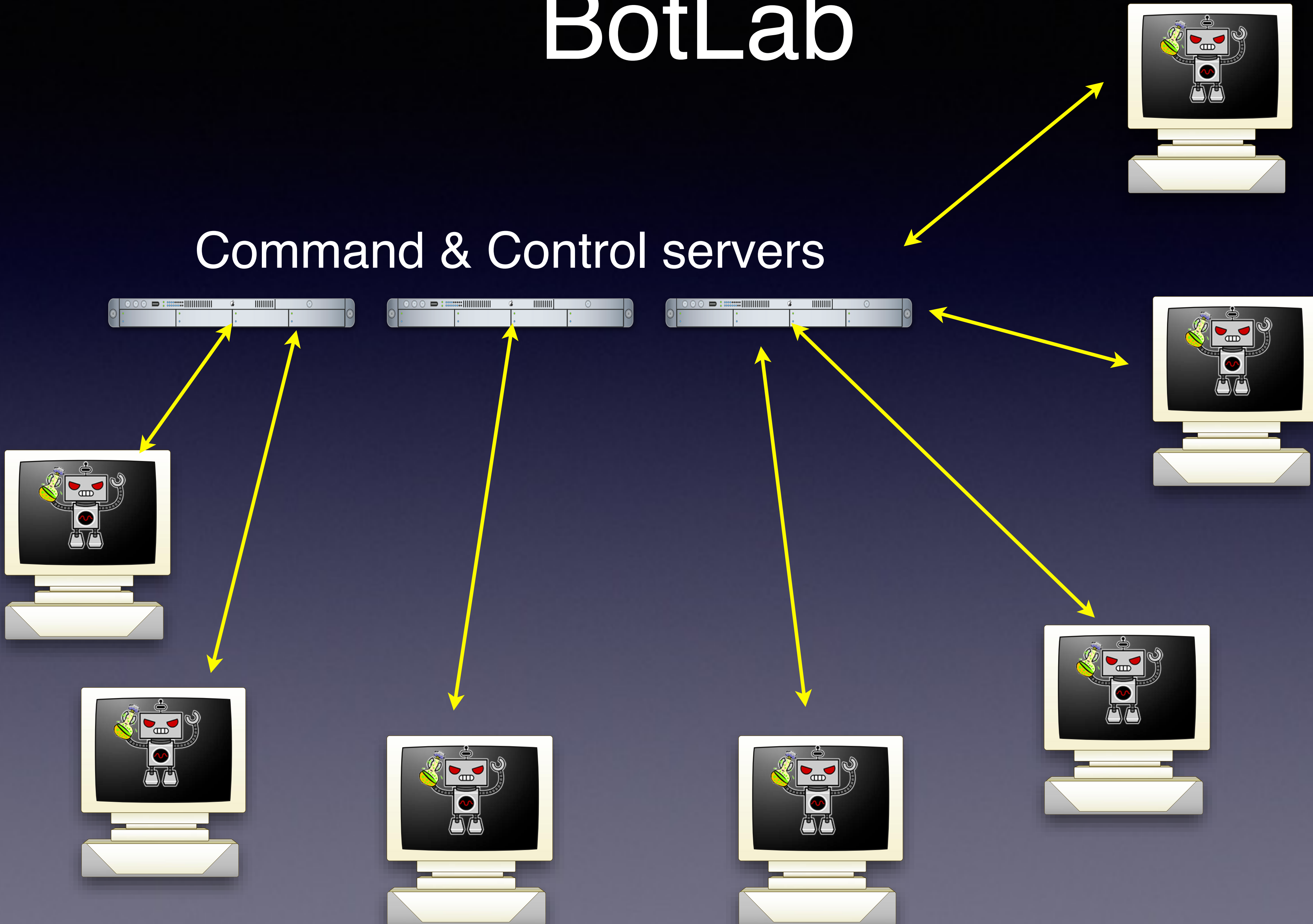
> "Storm botnet has 20 thousand nodes."
>
> Oct 2007

> "Most nodes in Storm botnet are from security researchers."
>
> Apr 2008

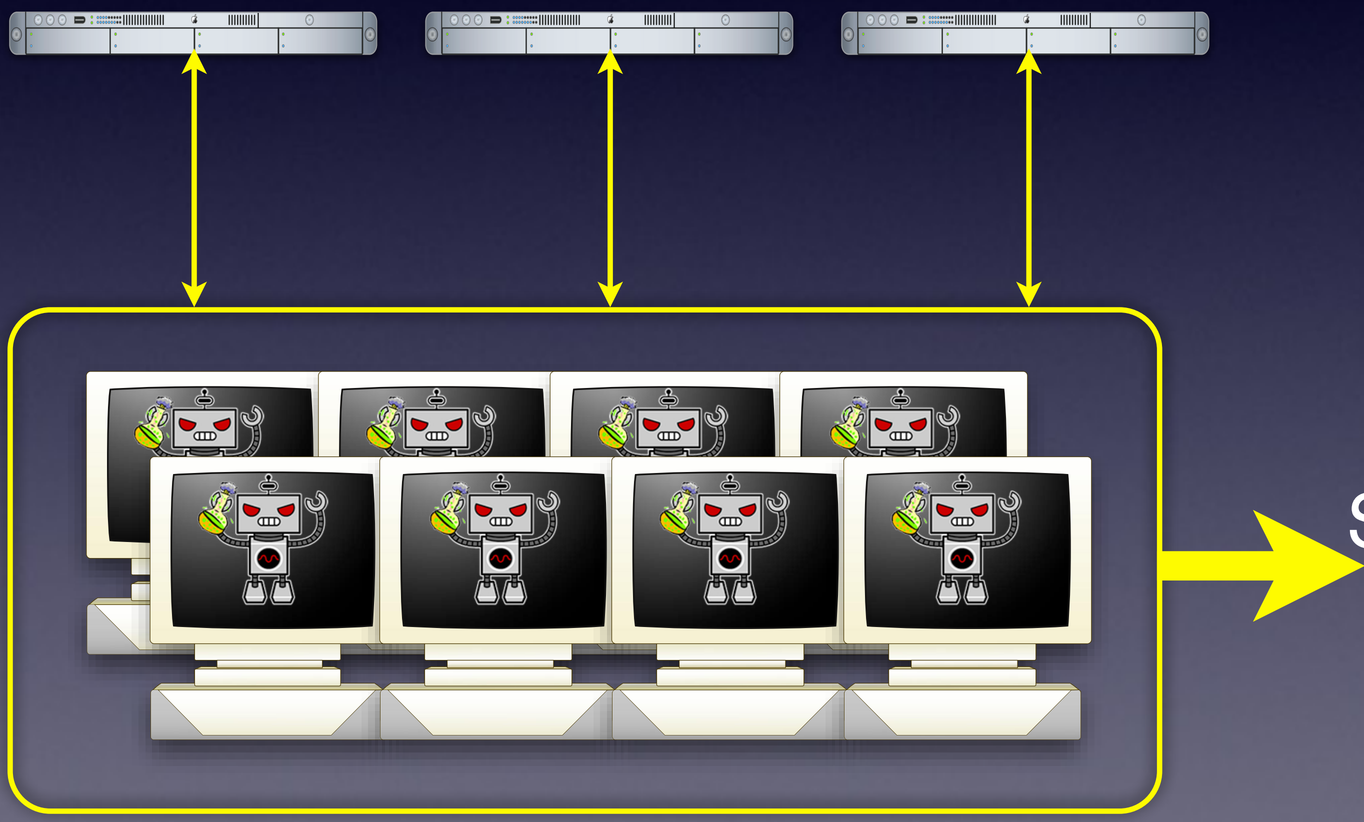
Goal: Build BotLab

To build a system, which can,
in a timely fashion,
with minimum human interaction,
monitor botnets and their propagation.

BotLab

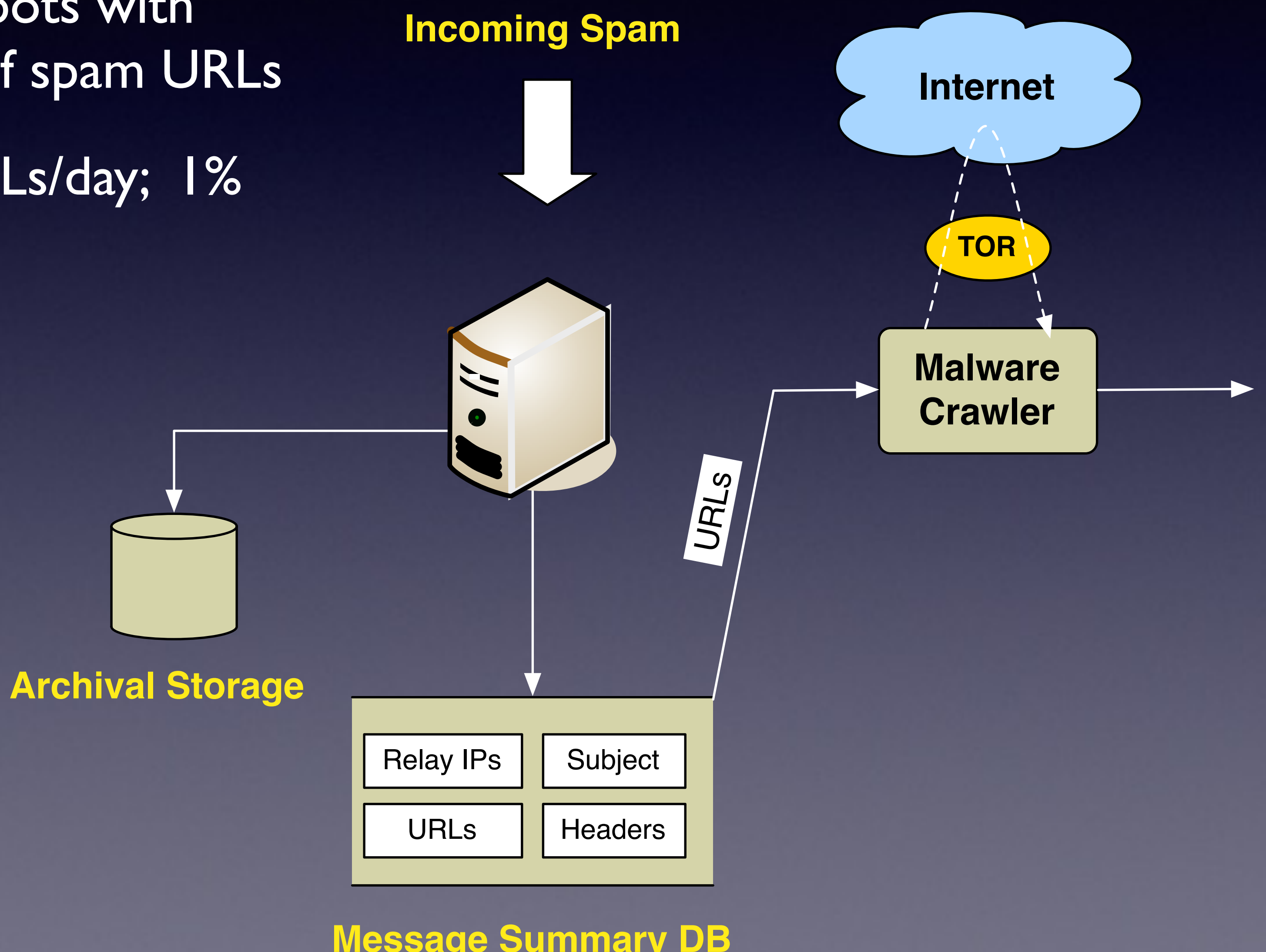


BotLab

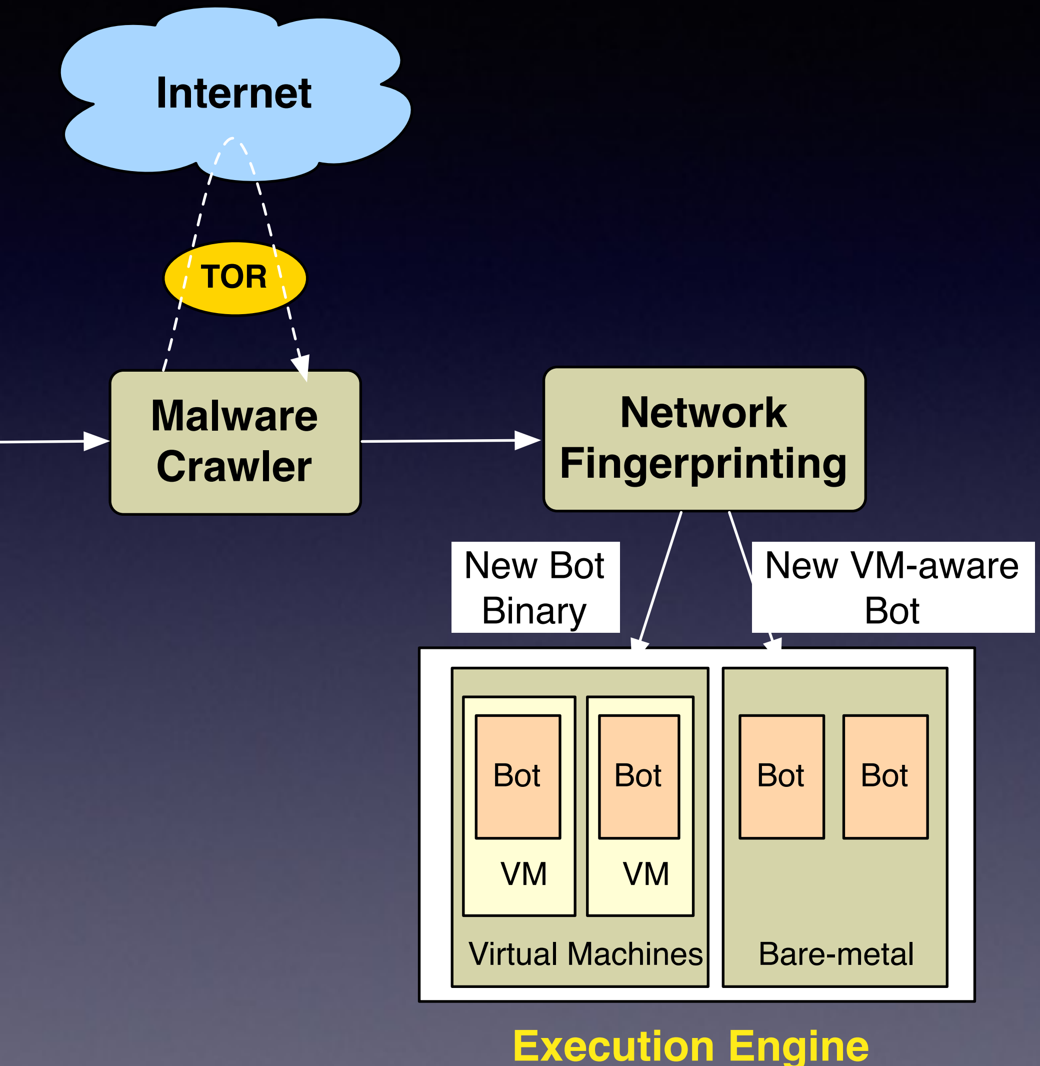


Malware Collection

- Augment honeypots with active crawling of spam URLs
- 100K unique URLs/day; 1% malicious

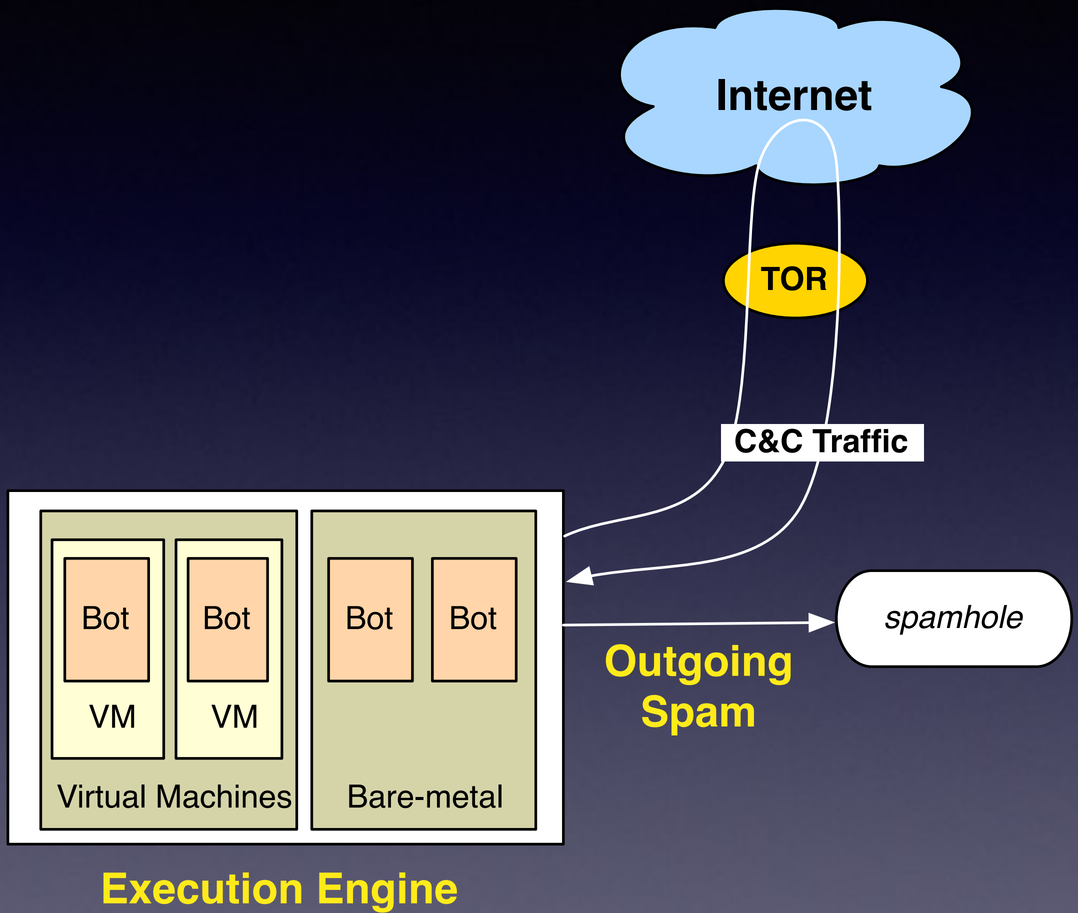


Network Fingerprinting



- Goal: find new bots while discarding old ones
- Execute binaries and generate a fingerprint, which is a sequence of flow records
- Execute both inside and outside of VM to check for VM detection

Coaxing Bots to Run



- Bots send "verification" emails before they start sending regular spam
- Some other bots spam using webservices (such as HotMail)
- Bots with 100% email delivery rate are considered suspicious
- Fortunately only O(10) botnets; so manual tweaking possible

Findings

- Botnet monitoring problem seems tractable:
  - Small number of botnets (< 10) account for most of the spam
  - Most spam botnets have fewer than 100K bots
  - Scam hosts and C&C servers don't change often

- Other related projects:
  - How to foil malware distribution?
    - How to prevent webservers from being compromised?
    - How to prevent search engine pollution?
  - How to integrate information from BotLab to safeguard end-hosts?

Research Interests

- Peer-to-peer systems
- Network security
- Privacy systems & Censorship resistance
- Data center networks
- Distributed storage systems

P2P Monitoring

- Open protocols, open access ⇒ self-scaling
- *Easy* to monitor
- We performed a month long study of BitTorrent:
  - Tracked membership in *55,523 swarms*, observed more than *14 million peers*

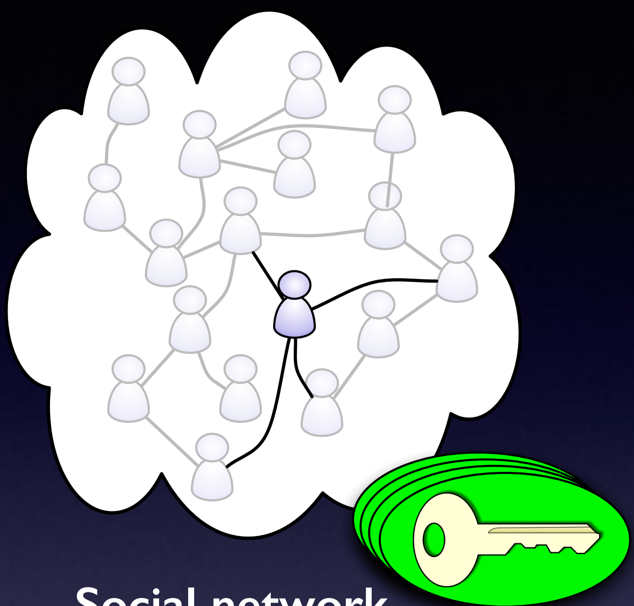
Goal

Can we build a P2P system that is both *efficient* and *privacy preserving?*

Data sharing

1. Private
2. Public (not sensitive)
3. Public but *without attribution*

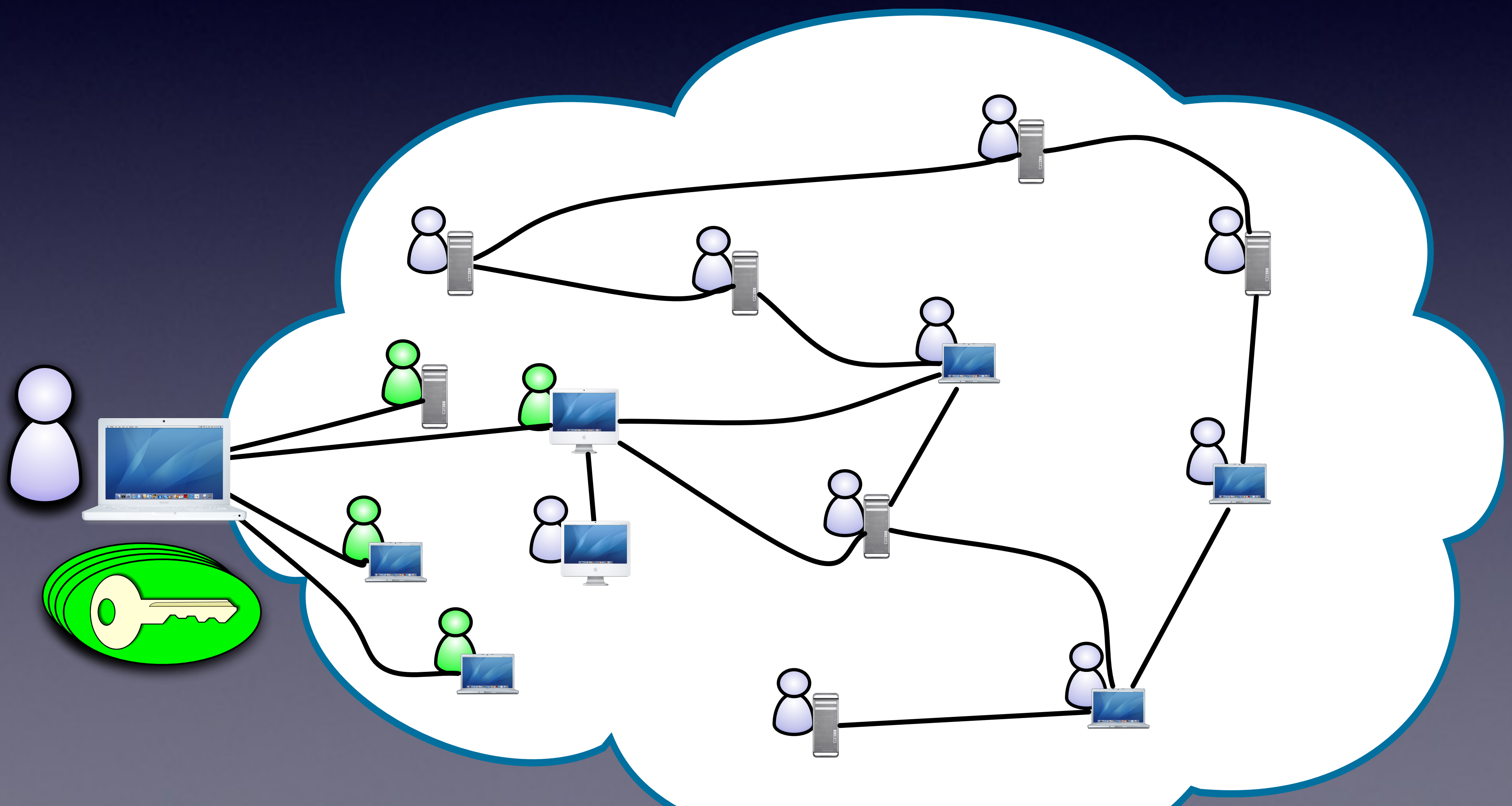
OneSwarm sketch



Social network

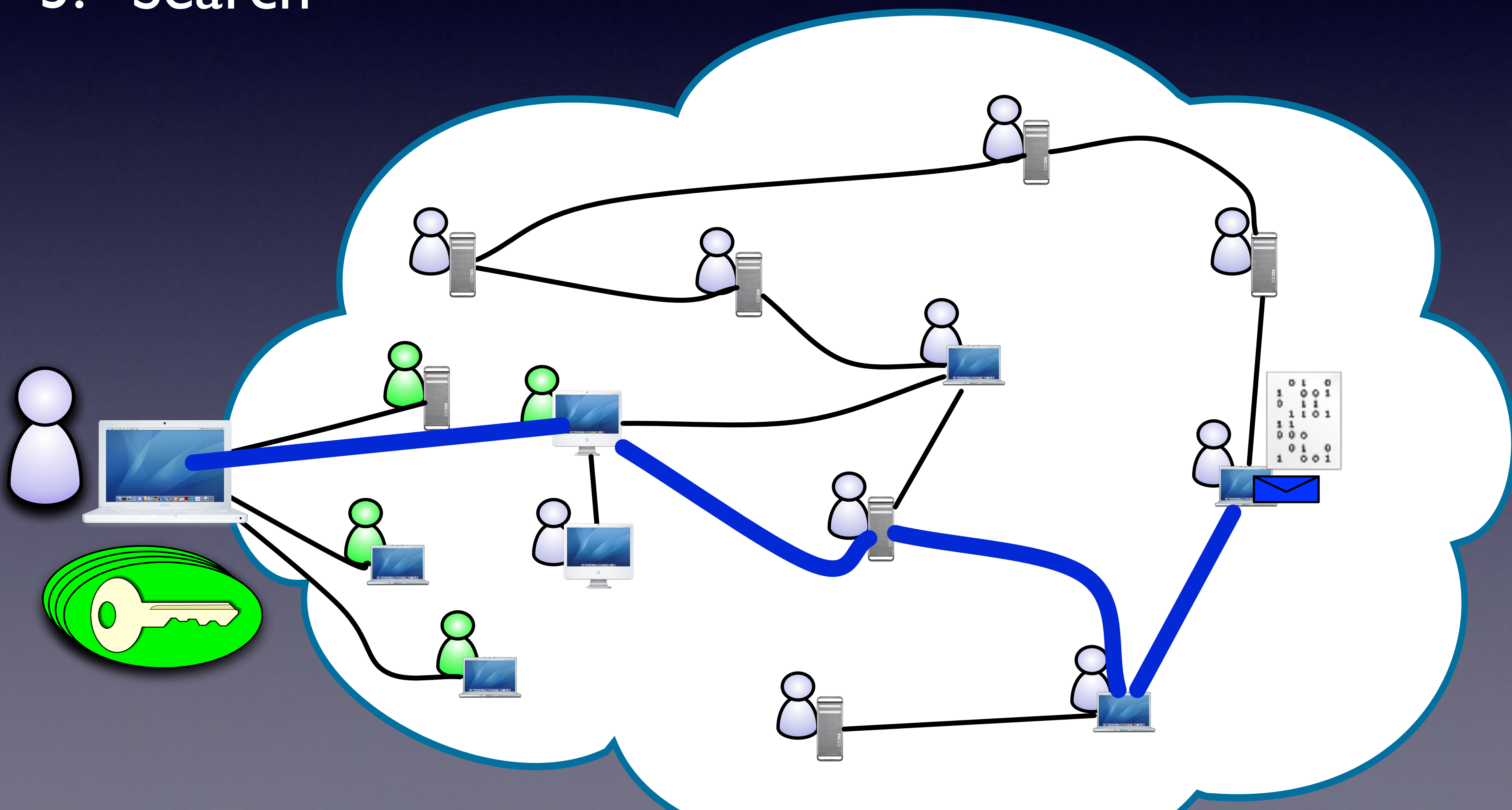
OneSwarm sketch

1. Import keys
2. Connect



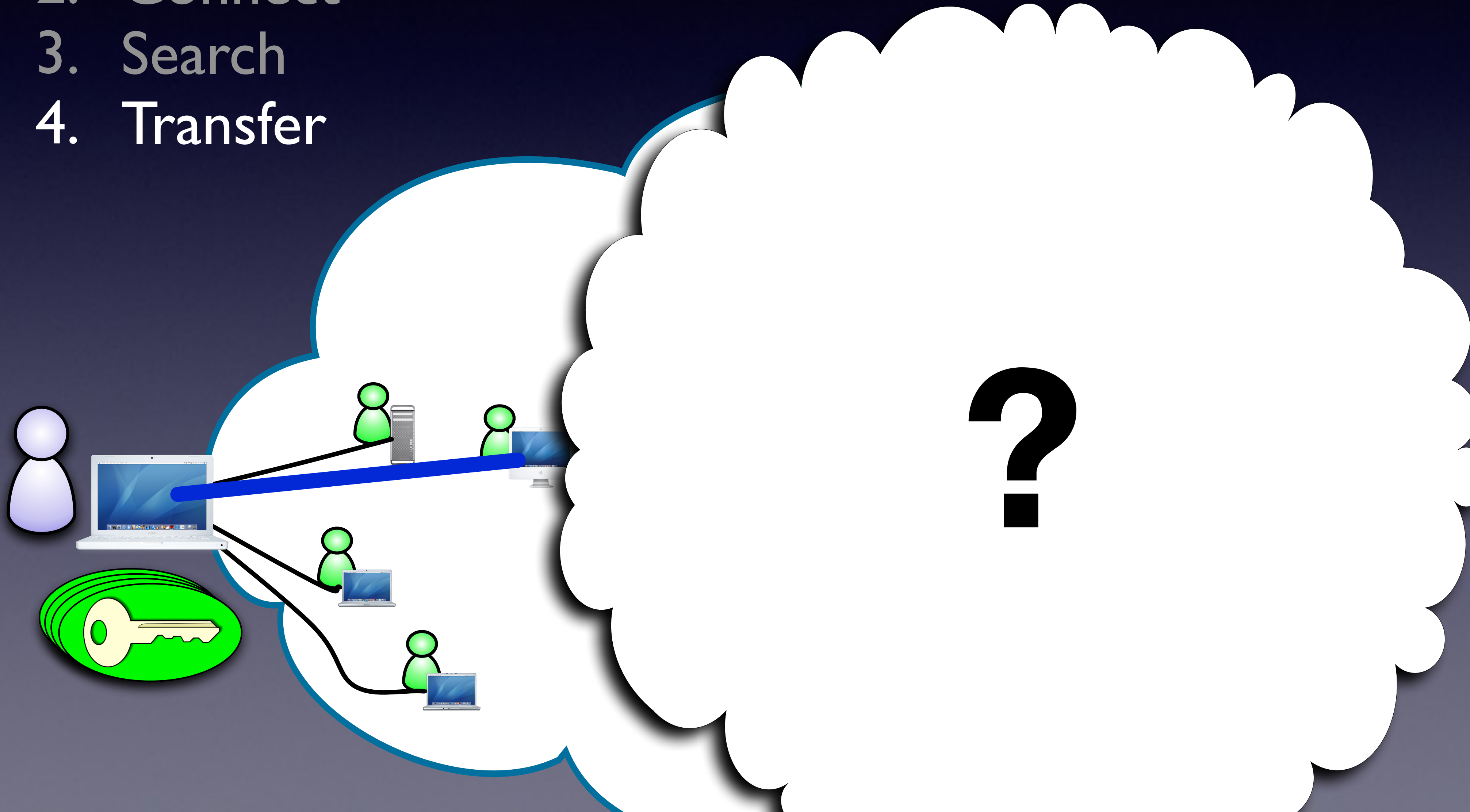
OneSwarm sketch

1. Import keys
2. Connect
3. Search



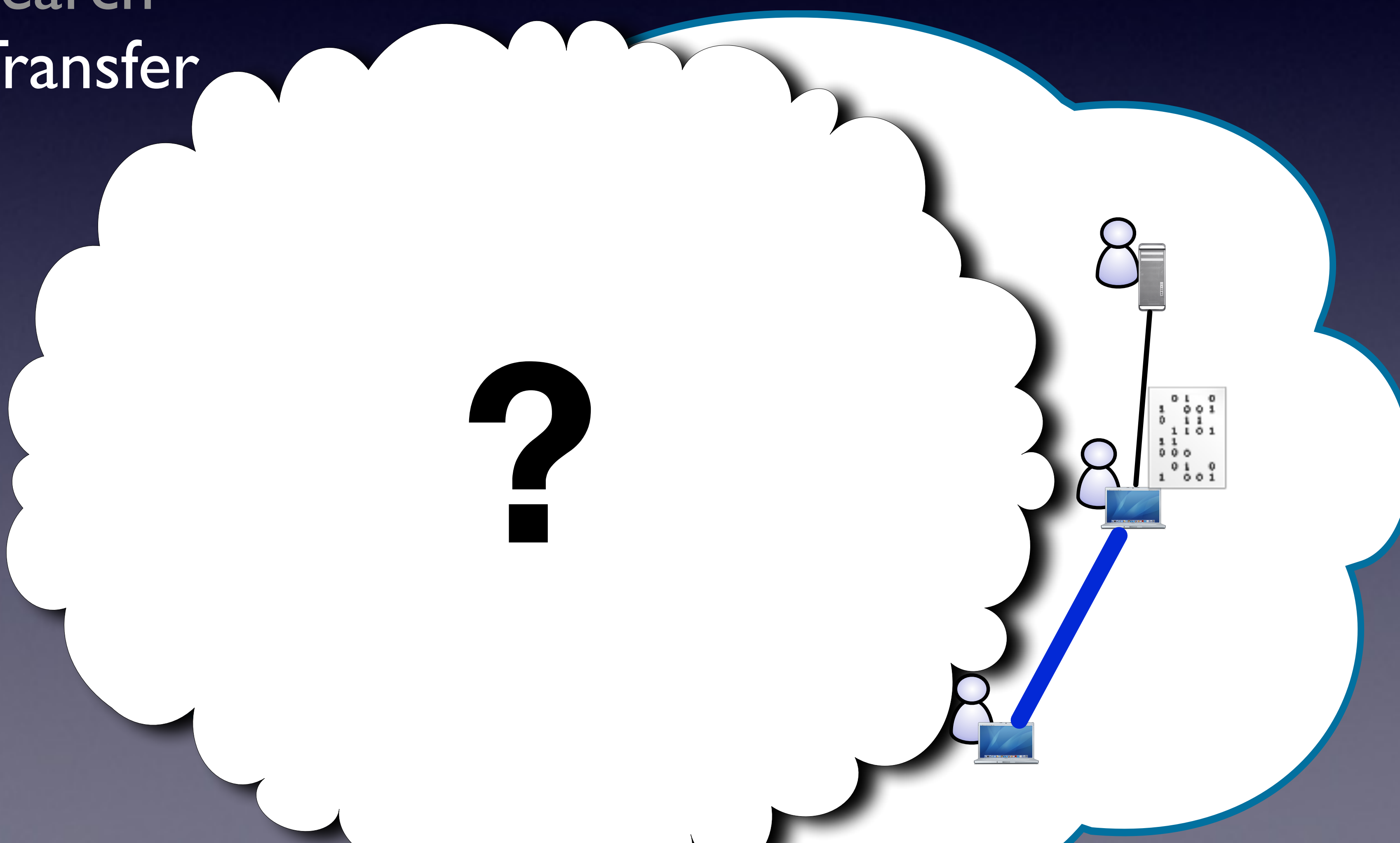
Receiver view

1. Import keys
2. Connect
3. Search
4. Transfer



Sender view

1. Import keys
2. Connect
3. Search
4. Transfer



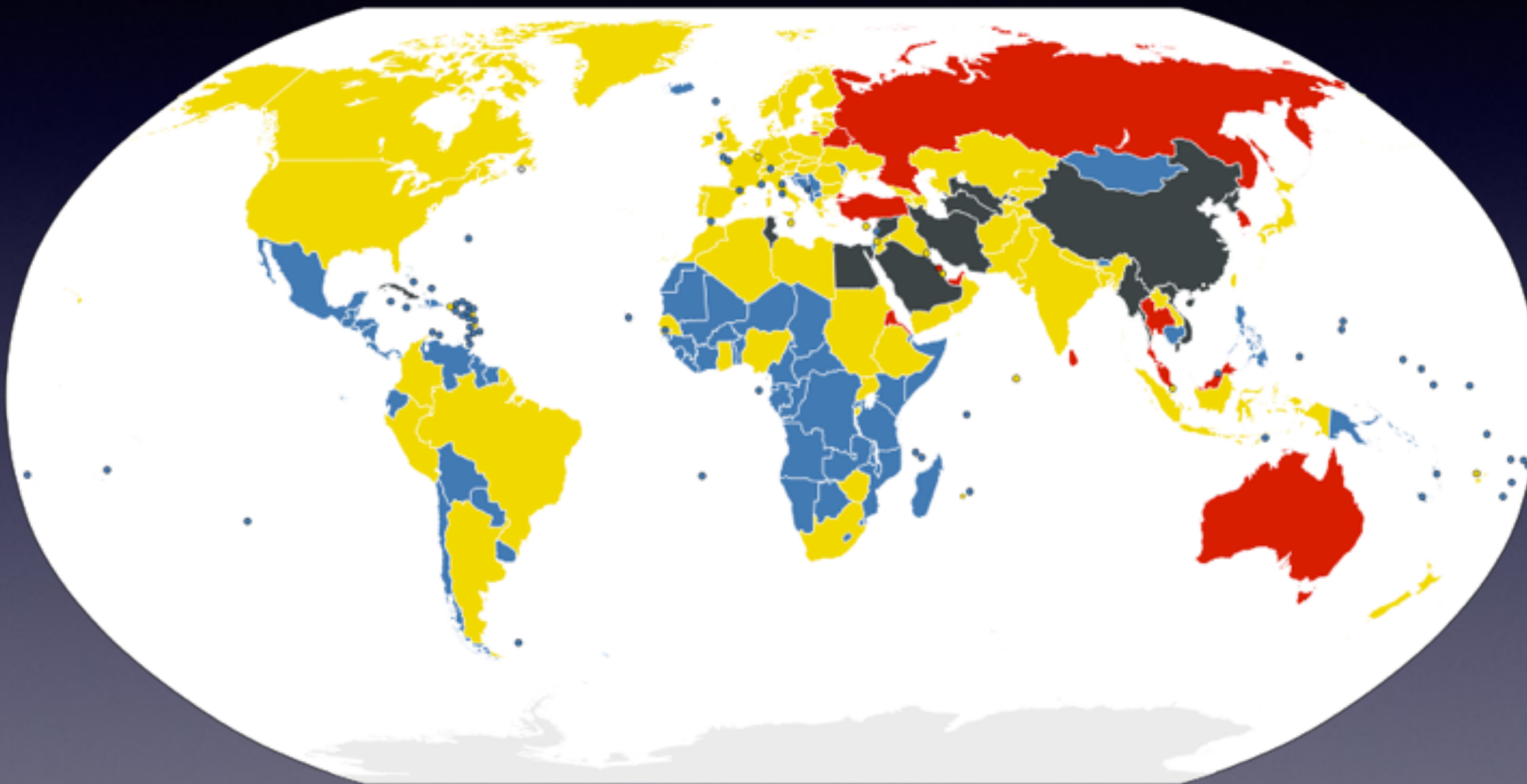
Design challenges

- Controlled flooding based on workload
- Locating mobile peers
- Robustness despite sparse social networks
- Maintain performance despite long paths

OneSwarm

- Client publicly *released*
- *Hundreds of thousands of unique users*
- Flexible protocol gives users control of *privacy/performance* tradeoff

Censorship



No censoring
Some censorship
Surveillance
Heavy censors

Adversary Resilient Network Services

- To achieve censorship resistance, we can leverage prior work on:
  - Social networks ⇒ basic level of trust
  - P2P systems ⇒ hide with legitimate traffic, exploit dynamics of IP and churn
  - Botnet design!

Challenges

- How to achieve good performance?
  - Multi-hop overlay communication incurs high latency
  - Churn could reduce availability
  - Bandwidth bottlenecks at censorship boundary crossings
- How to cope with a powerful adversary?
  - Can reverse engineer protocols, block bootstrapping, spoof DNS results, and so on.
- How to provide Sybil resistance?
  - Adversary will attempt to crawl the system; existing defenses might not inspire confidence in users
- And many more...