

Game Theory

Presented by Hakim Weatherspoon

Game Theory

- Main Question: Can we cheat (and get away with it)?
- BitTorrent
 - P2P file distribution tool designed with *incentives for contribution*
 - Users contribute resources to get good performance
- Do Incentives Build Robustness in BitTorrent?
 - Can BitTorrent be gamed?
- BAR Gossip
 - Can content distribution be tamed?

BitTorrent

- Written by Bram Cohen (in Python) in 2001
- “Pull-based” “swarming” approach
 - Each file split into smaller **pieces**
 - Nodes request desired pieces from neighbors
 - As opposed to parents pushing data that they receive
 - Pieces not downloaded in sequential order
 - Previous multicast schemes aimed to support “streaming”; BitTorrent does not
- Encourages contribution by all nodes

BitTorrent

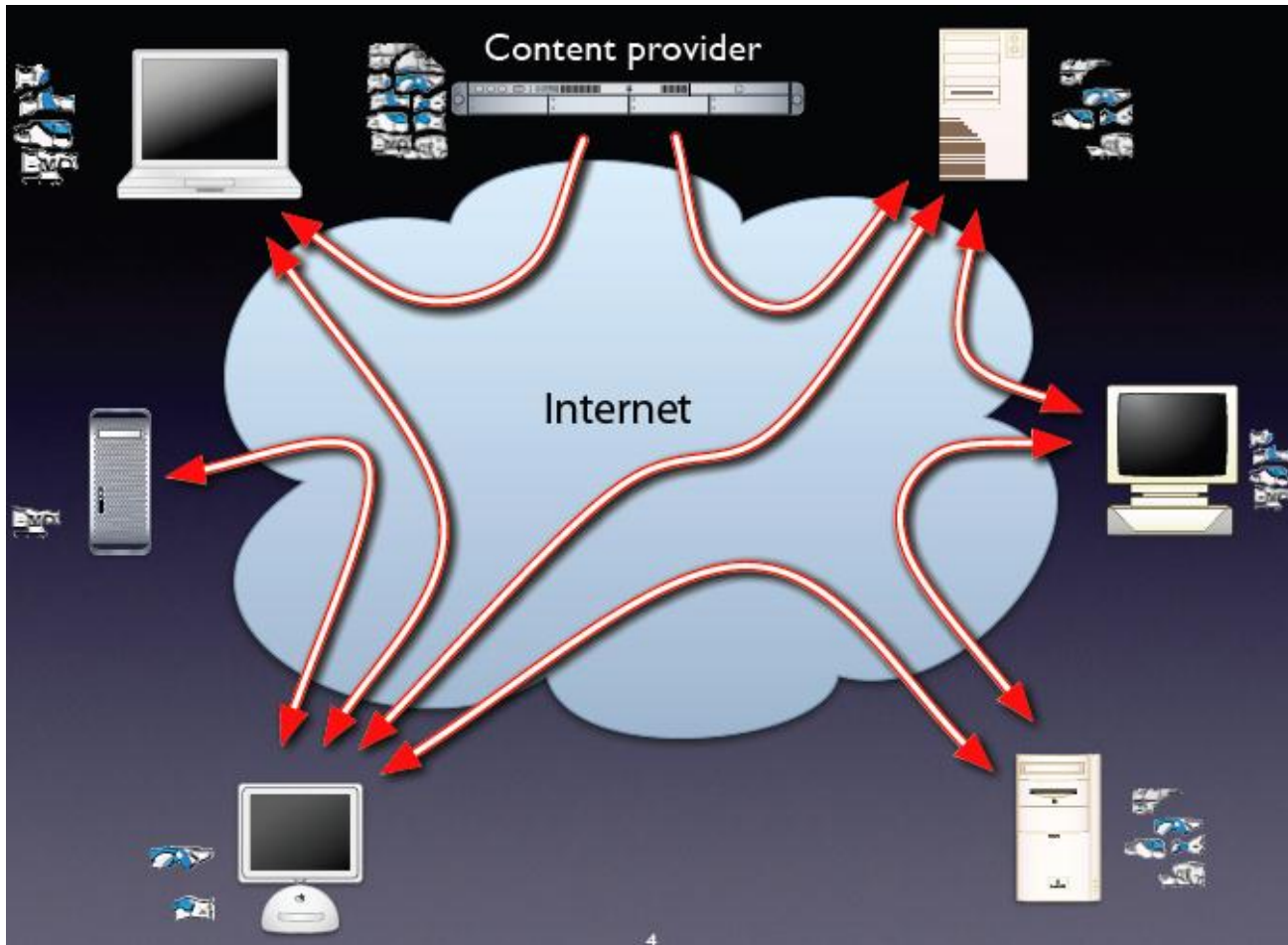


A big file



Broken
into pieces

BitTorrent



BitTorrent Swarm

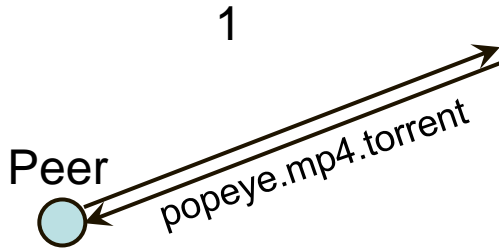
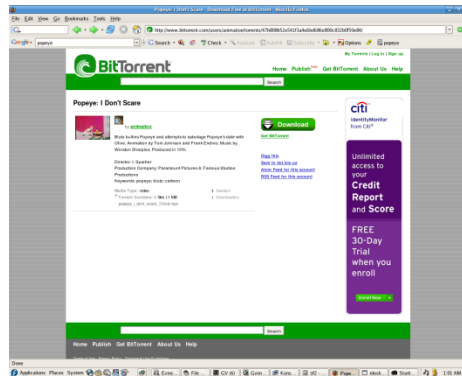
- **Swarm**
 - Set of peers all downloading the same file
 - Organized as a random mesh
- Each node knows list of pieces downloaded by neighbors
- Node requests pieces it does not own from neighbors
 - Exact method explained later

How a node enters a swarm for file “popeye.mp4”

- File popeye.mp4.torrent hosted at a (well-known) webserver
- The .torrent has address of **tracker** for file
- The tracker, which runs on a webserver as well, keeps track of all peers downloading file

How a node enters a swarm for file “popeye.mp4”

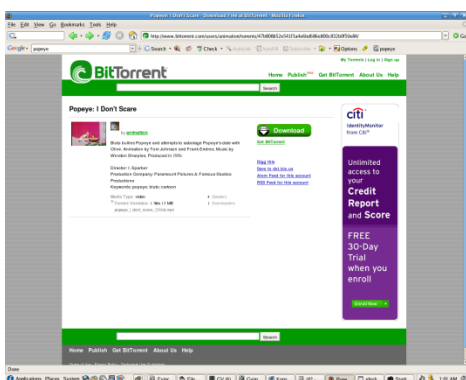
www.bittorrent.com



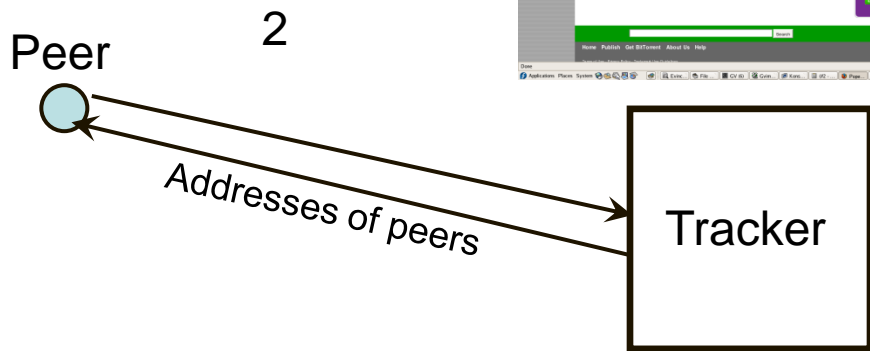
- File popeye.mp4.torrent hosted at a (well-known) webserver
- The .torrent has address of **tracker** for file
- The tracker, which runs on a webserver as well, keeps track of all peers downloading file

How a node enters a swarm for file “popeye.mp4”

www.bittorrent.com

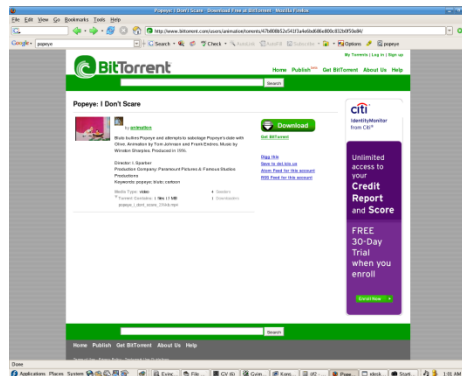


- File popeye.mp4.torrent hosted at a (well-known) webserver
- The .torrent has address of **tracker** for file
- The tracker, which runs on a webserver as well, keeps track of all peers downloading file

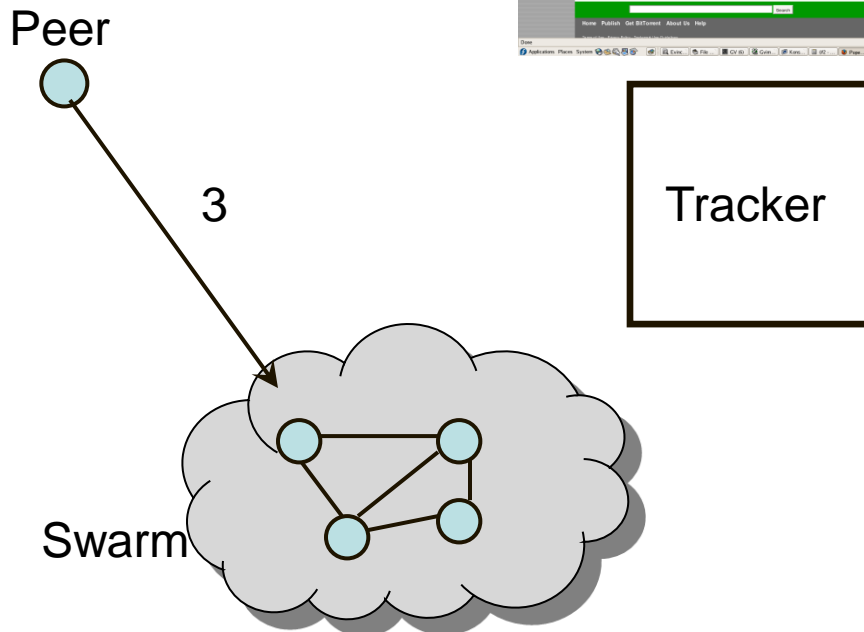


How a node enters a swarm for file “popeye.mp4”

www.bittorrent.com



- File popeye.mp4.torrent hosted at a (well-known) webserver
- The .torrent has address of **tracker** for file
- The tracker, which runs on a webserver as well, keeps track of all peers downloading file



Contents of .torrent file

- URL of tracker
- Piece length – Usually 256 KB
- SHA-1 hashes of each piece in file
 - For reliability
- “files” – allows download of multiple files

Terminology

- **Seed**: peer with the entire file
 - Original Seed: The first seed
- **Leech**: peer that's downloading the file
 - Fairer term might have been “downloader”
- **Sub-piece**: Further subdivision of a piece
 - The “unit for requests” is a subpiece
 - But a peer uploads only after assembling complete piece

Peer-peer transactions:

Choosing pieces to request

- **Rarest-first:** Look at all pieces at all peers, and request piece that's owned by fewest peers
 - Increases diversity in the pieces downloaded
 - avoids case where a node and each of its peers have exactly the same pieces; increases throughput
 - Increases likelihood all pieces still available even if original seed leaves before any one node has downloaded entire file

Choosing pieces to request

- **Random First Piece:**
 - When peer starts to download, request random piece.
 - So as to assemble first complete piece quickly
 - Then participate in uploads
 - When first complete piece assembled, switch to rarest-first

Choosing pieces to request

- **End-game mode:**
 - When requests sent for all sub-pieces, (re)send requests to all peers.
 - To speed up completion of download
 - Cancel request for downloaded sub-pieces

Tit-for-tat as incentive to upload

- Want to encourage all peers to contribute
- Peer A said to **choke** peer B if it (A) decides *not* to upload to B
- Each peer (say A) **unchoke** at most 4 *interested* peers at any time (Azureus client)
 - The three with the largest upload rates to A
 - Where the tit-for-tat comes in
 - Another randomly chosen (**Optimistic Unchoke**)
 - To periodically look for better choices
 - 4 is size of **active set**, but can be more dynamic

Anti-snubbing

- A peer is said to be snubbed if each of its peers chokes it
- To handle this, snubbed peer stops uploading to its peers
- Optimistic unchoking done more often
 - Hope is that will discover a new peer that will upload to us

Why BitTorrent took off?

- Better performance through “pull-based” transfer
 - Slow nodes don’t bog down other nodes
- Allows uploading from hosts that have downloaded parts of a file
 - In common with other end-host based multicast schemes

Why BitTorrent took off?

- Practical Reasons (perhaps more important!)
 - Working implementation (Bram Cohen) with simple well-defined interfaces for plugging in new content
 - Many recent competitors got sued / shut down
 - Napster, Kazaa
 - Doesn't do "search" per se. Users use well-known, trusted sources to locate content
 - Avoids the pollution problem, where garbage is passed off as authentic content

Pros and cons of BitTorrent

- Pros
 - Proficient in utilizing partially downloaded files
 - Discourages “freeloading”
 - By rewarding fastest uploaders
 - Encourages diversity through “rarest-first”
 - Extends lifetime of swarm
- Works well for “hot content”

Pros and cons of BitTorrent

- Cons
 - Assumes all interested peers active at same time; performance deteriorates if swarm “cools off”
 - Even worse: no trackers for obscure content
- Recent studies by team at U. Washington found that many swarms “fail” because there are few changes for repeated interaction with the same peer
 - They suggest fixes, such as “one hop reputation” idea presented at NSDI 2008

Pros and cons of BitTorrent

- Dependence on centralized tracker: pro/con?
 - ☹ Single point of failure: New nodes can't enter swarm if tracker goes down
 - Lack of a search feature
 - 😊 Prevents pollution attacks
 - ☹ Users need to resort to out-of-band search: well known torrent-hosting sites / plain old web-search

Do Incentives Build Robustness in BitTorrent?



- Michael Piatek
 - Grad Student @ UW



- Tomas Isdal
 - Grad Student @ UW



- Tom Anderson
 - Prof @ UW. Author of Nacho's OS



- Arvind Krishnamurthy
 - Prof @ UW



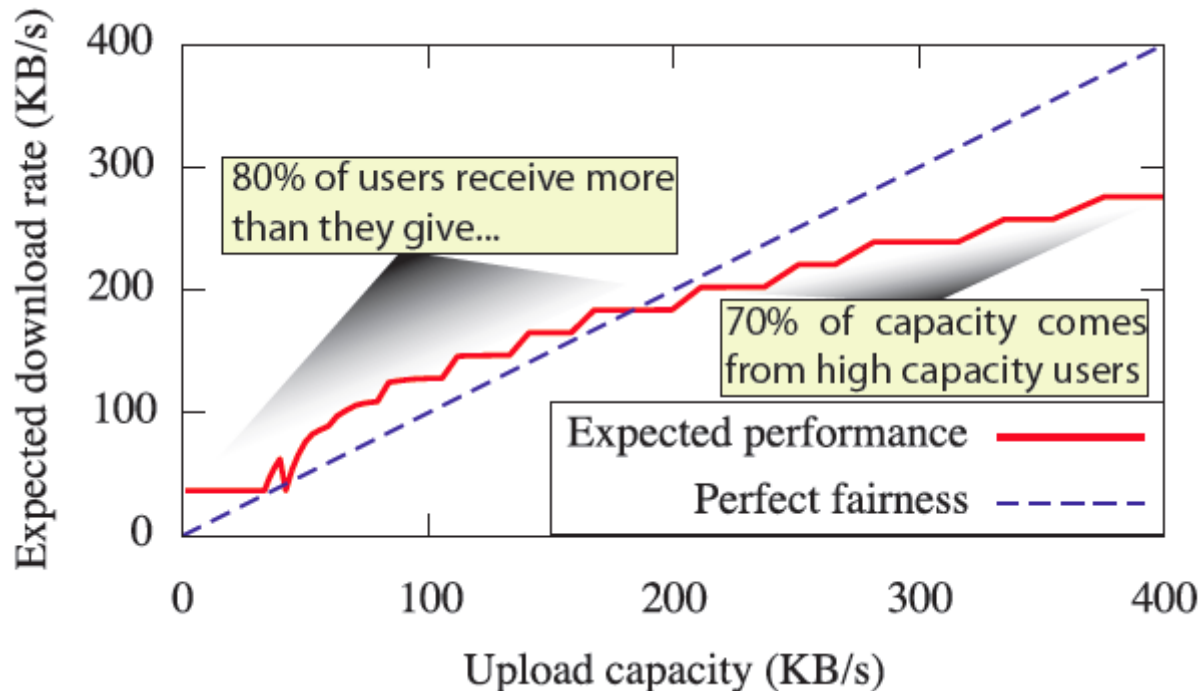
- Arun Venkataramani
 - Prof @ Umass Amerst.

BitTorrent

- BitTorrent is a protocol for bulk data transfer
- The more you give, the more you get
- Tit-for-tat
 - Not really

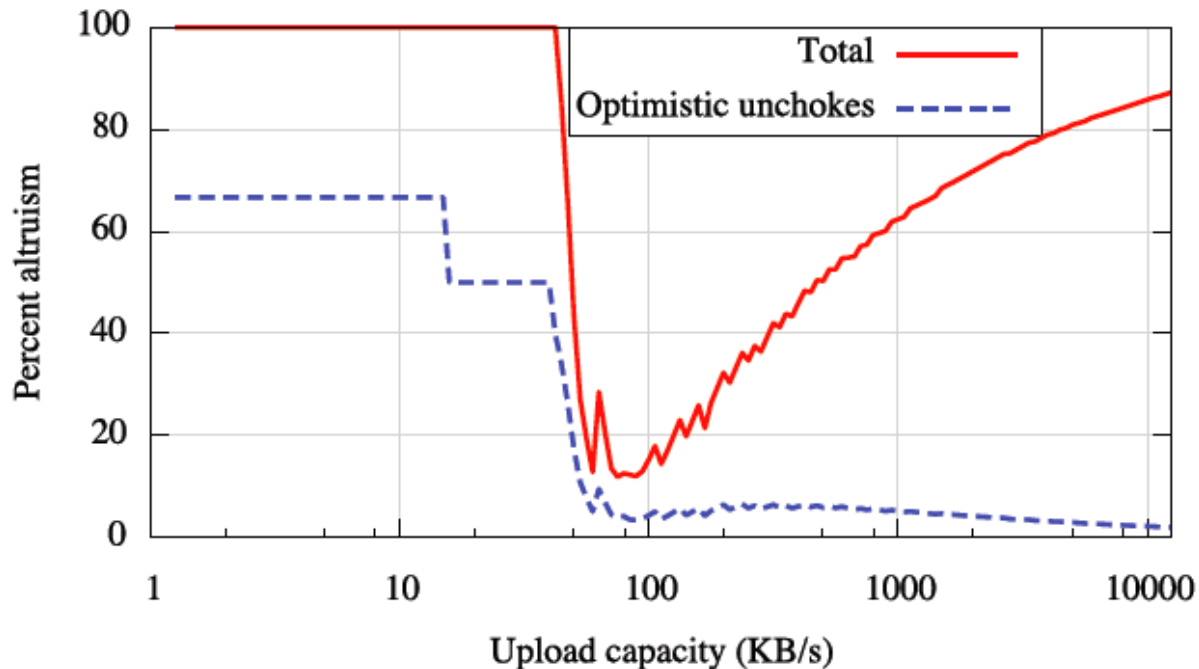
Altruism

- Not really TFT
 - 80% of users get more than they give
 - 70% of capacity comes from high capacity users



Altruism

- Not really TFT
 - 80% of users get more than they give
 - 70% of capacity comes from high capacity users
 - Not really a fair protocol
 - Operates based on altruism of high capacity users



Exploiting Altruism

- Selfish users
 - Rational
 - Want to cheat (as long as they won't get caught!)
 - Can download with many low-speed connections rather than 1 high-speed

Building BitTyrant

- Maximize reciprocation bandwidth per connection
 - Find peers that give the most for the least
- Maximize number of reciprocating peers
 - Get as many peers as possible
 - Until benefit of new peer is outweighed by cost of reduced reciprocation from other peers
- Deviate from equal
 - Decrease uploading on each connection
 - Until peer stops reciprocating

Results

- BitTyrant improves average download performance by 70%
- Regardless of capacity, using BitTyrant is in the selfish interest of every peer individually
- When all peers behave selfishly, average performance degrades for all peers, even those with high capacity

Take-away

- BitTorrent works because people use the default client
 - No cheating
- BitTyrant is now available in the wild
 - This is a test – Do incentives build robustness?
 - Maybe users will continue to donate excess bandwidth
 - Maybe users will be selfish
 - Proven to reduce overall capacity

BAR Gossip

Henry Li, Allen Clement, Edmund
Wong, Jeff Napper, Indrajit Roy,
Lorenzo Alvisi, and Michael Dahlin

BAR Model

- **Byzantine (Arbitrary) Nodes**
- **Altruistic (Generous) Nodes**
- **Rational (Selfish) Nodes**

BAR Gossip Vision

- In presence of:
 - Selfish nodes
 - Byzantine nodes
- We want:
 - Predictable throughput
 - Low latency

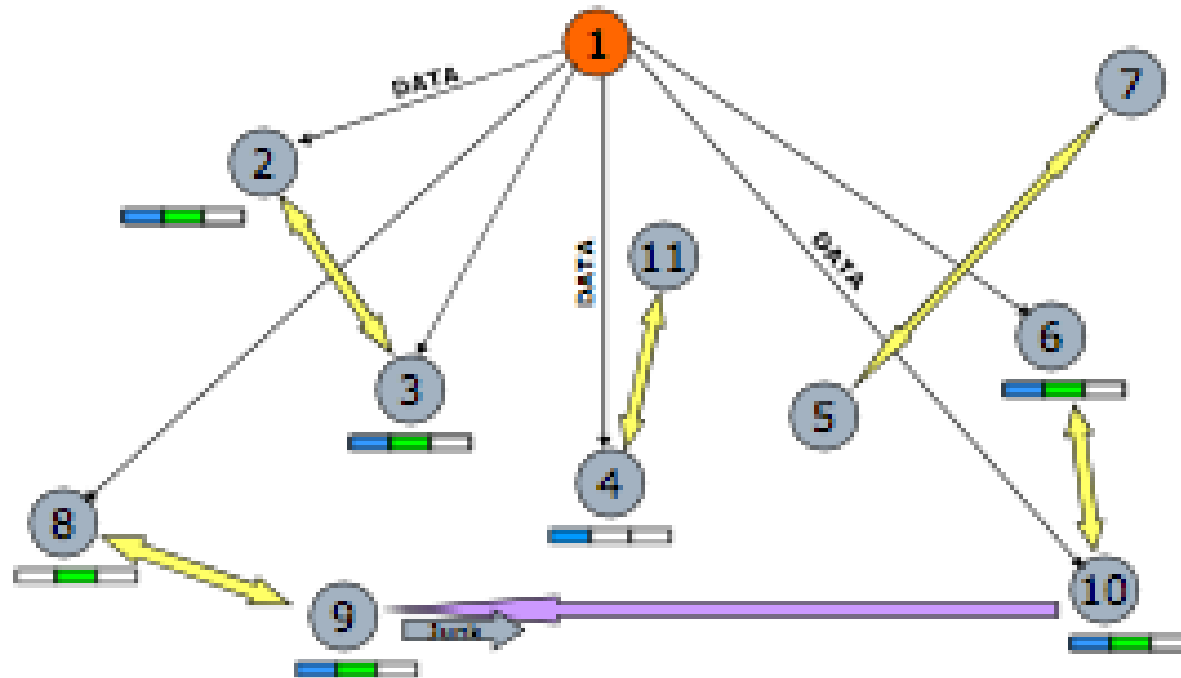
BAR Gossip Differences

- Data exchange in short periods
- No long-term reputation
- Exchanges small blocks of data
- Robust to both *Selfish* and *Byzantine* behavior

BAR Gossip Assumptions

- One data broadcaster
- Static Membership
- Reliable Cryptographic primitives (SHA1,RSA)
- Unique keys and signatures for nodes

Overview



Core Idea

- **Balanced Exchange**
 - When a peer gives some data in exchange of some data. “Trade data for data.”
- **Optimistic Push**
 - Every peer will willingly help others by giving them data for free. Be a good person and give data for free.
- What about when I don't have anything to trade off with, am I out of the game then ?

Take-away

- Current gossip protocols are ill-suited for selfish environments.
- Bar Gossip
 - Verifiable pseudo-randomness
 - Signatures
 - Balanced Exchange, achieves 98% reliability.
 - With Optimistic Push, increases to almost 99.9%

Next Time

- Final Presentations
 - **Room 5126** (from 9am-3pm)
 - 9am – 3:00pm, Thursday, December 1st
 - 15 - 20 minute presentations, 5 -10 minute questions
 - Signup for presentation slot via CMS
- Final paper
 - Due next Thursday, December 8th
 - Complete paper and comprehensive evaluation
- Thank you!