Internet-based
Collaborative Decentralized Systems
Collaboration for performance

Maarten van Steen

vrije Universiteit amsterdam

2 of 47

Introduction

So far:

- Large-scale Internet-based systems can benefit a lot from (sometimes massive) replication
- Assumption: there will be trusted servers to host replicated content

Question: What are the issues when dealing with **collaborative decentralized systems**, such as, for example, collaborative content distribution networks.

Note: Many peer-to-peer systems fall into this category.

3 of 47

Collaboration fundamentals

Issue: How can we enforce several parties (unknown to each other) to help run a distributed system?

- **Traditional:** Be able to continuously monitor behavior, and take measures in the case of misbehaving participants:
  - Withdraw from collaboration (i.e., defeat)
  - Punish misbehaving participant (i.e., enforce collaboration)
- **Alternative:** Provide incentives so that participants want to collaborate.

Note: Sometimes it is sufficient to rely on altruistic behavior by participants (e.g., Wikipedia)

4 of 47

Evolution of cooperation

Prisoner's dilemma: Two players, two options (cooperate or defect). Each player must take a decision without knowing what the other will do. There are four payoffs:

T: Temptation to defect
R: Reward for co-operation
P: Punishment for mutual defect
S: Sucker's payoff when only one defects

Constraint: $T > R > P > S$. We can get something like:

		Player 2	
		Cooperate	Defect
Player 1	Cooperate	$R_1 = 3, R_2 = 3$	$S_1 = 0, T_2 = 5$
	Defect	$T_1 = 5, S_2 = 0$	$P_1 = 1, P_2 = 1$

Note: Total payoff is highest when co-operating

5 of 47

Iterative Prisoner's Dilemma

Situation: We let two parties repeat the game, taking the result from the previous game into account. For co-operation to emerge, we also demand $2R > T + S$.

Strategy: Somewhat surprisingly, the simplest strategy turns out to be the best: **tit-for-tat**:

- Cooperate on the first interaction (i.e., be optimistic)
- Subsequently do the same as what your partner did (i.e., reciprocate)

6 of 47

Cooperation in practice

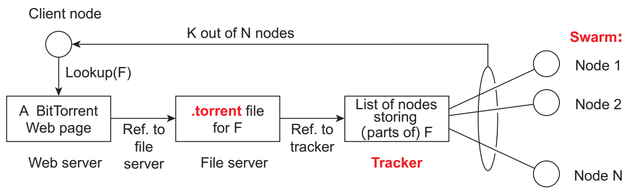
For cooperation to emerge in practice, the following conditions need to be met:

1. Parties should be able to recognize each other
2. There should be repeated interaction between parties
3. Interactions should be durable or frequent
4. The strategy followed by the other party should be transparent

Note: Often, almost each of these conditions are violated in "collaborative" (peer-to-peer) systems! We often need small, persistent groups for cooperation to emerge.

7 of 47

Example: BitTorrent



Pieces, blocks	Files are split into pieces, in turn split into blocks
Peer set	List of peers to which a node has open TCP connections
Leecher, seeder	Leechers are providing blocks, but also need blocks to complete. A seeder has all the necessary blocks and continues to provide them (altruistic behavior).
Potential set	Peers in peer set that have at least one block to trade.

8 of 47

BitTorrent at work

- A peer P locates a tracker T for file F. T sends randomly selected set of peers $PS[P]$. P request blocks from $PS[P]$.
- A peer Q regularly computes best uploading peers in $PS[Q]$ and chokes the ones not uploading to Q.
- A peer Q regularly unchokes a randomly selected peer P from $S(F)$: Q will altruistically send blocks to P when requested for.
- A peer P requests blocks from the rarest available pieces in $PS[P]$. Note: this requires exchange of information on available pieces.
- Peer sets are regularly updated by letting P contact its tracker T.

Note: BitTorrent essentially follows a tit-for-tat strategy; there are many details to consider to make the protocol lead to collaboration.

9 of 47

Amortized Tit-for-Tat

Observation: BitTorrent plays tit-for-tat (1) on the basis of content exchange and (2) cannot preserve information between sessions.

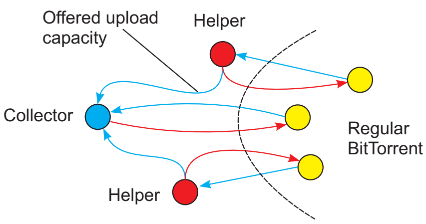
Also note: In environments with asymmetric links (e.g., ADSL), download speed is often limited by upload capacity ⇒ slow downloads.

Alternative: amortized tit-for-tat:

- Exchange bandwidth instead of content
- Provide bandwidth now, and try to get help the next time you need it

10 of 47

Amortized TFT at work



Note: Helpers use their upload capacity to (1) trade blocks with regular BT nodes and (2) pass blocks to collector.

11 of 47

Performance of amortized TFT

N	# leechers
S	# seeders
K	# blocks file has been split into (1 piece = 1 block)
L_b	# peers having block b ($L_{b_1} \approx L_{b_2}$)
n_i	# blocks currently held by P_i
$m_{i,b}$	1 if P_i has block b, otherwise 0
μ	Upload capacity of single peer. Assume the same for all peers
c	Download capacity, assume $c \geq \mu$

12 of 47

Effective download of collector

- S seeders equally divide μ over N leechers
- Collector can use full μ capacity to barter with BT peers.
- Helper i provides fraction f_i of μ for helping download; h helpers

$$d = \frac{S}{N} \cdot \mu + \mu + \sum_{i=1}^{h} f_i \cdot \mu$$

Note: For helpers to be useful, we assume that $c > S\mu/N + \mu$

N	# leechers	K	# file blocks	n_i	# blocks at P_i	μ	upload cap.
S	# seeders	L	# peers /w block b	$m_{i,b}$	1 iff b @ P_i	c	download cap.

13 of 47

Effective download of collector

- Helper h_i has effective download of $S\mu/N + (1-f_i)\mu$.
- Helper cannot transfer data to collector faster than it is getting data, so that $f_i \cdot \mu \leq S\mu/N + (1-f_i)\mu$.
- Conclusion: maximum is reached when $f_i = (S \cdot N + 1)/2$.

$$d_{max} = \left(\frac{S}{N}+1\right)\left(1+\frac{h}{2}\right)\mu \leq c \Rightarrow h_{opt} = 2\left(\frac{cN}{(S+N)\mu}-1\right)$$

N	# leechers	K	# file blocks	n_i	# blocks at P_i	μ	upload cap.
S	# seeders	L	# peers /w block b	$m_{i,b}$	1 iff b @ P_i	c	download cap.

14 of 47

Speedup with helpers

Observation: We can now easily determine how much a collector can gain by using helpers. For the **speedup** u we find:

$$u = \frac{d}{S/N \cdot \mu + \mu} = \begin{cases} 1 + \frac{h}{2} & \text{if } h < h_{opt} \\ \frac{cN}{(S+N)\mu} & \text{otherwise} \end{cases}$$

Note: More helpers also introduces overhead:

- In BitTorrent, downloads are slow in the beginning at the end.
- With helpers, we are effectively partitioning what needs to be downloaded $\Rightarrow$ more helpers, smaller subfiles, relatively longer start and end phases.

N	# leechers	K	# file blocks	n_i	# blocks at P_i	μ	upload cap.
S	# seeders	L	# peers /w block b	$m_{i,b}$	1 iff b @ P_i	c	download cap.

15 of 47

Speedup with helpers

Observation: Peer i can barter block k_1 for block k_2 at peer j iff:

$$m_{i,k_1}(1-m_{j,k_2})m_{j,k_2}(1-m_{i,k_2}) = 1$$

Let B_i be total number of block exchanges:

$$B_i = \sum_{j,k_1,k_2} m_{i,k_1}(1-m_{j,k_2})m_{j,k_2}(1-m_{i,k_2})$$

N	# leechers	K	# file blocks	n_i	# blocks at P_i	μ	upload cap.
S	# seeders	L	# peers /w block b	$m_{i,b}$	1 iff b @ P_i	c	download cap.

16 of 47

Speedup with helpers

Assume $\mathbb{P}[m_{i,k}=1]=n_i/K$ (and thus $\mathbb{P}[m_{i,k}=0]=(1-n_i/K)$):

$$\begin{aligned}
\mathbb{E}(B_i) &= \mathbb{E}\big(\textstyle\sum_{j,k_1,k_2} m_{i,k_1}(1-m_{j,k_2})m_{j,k_2}(1-m_{i,k_2})\big) \\
&= \textstyle\sum_{j,k_1,k_2}\big(\mathbb{P}[m_{i,k_1}=1]\mathbb{P}[m_{j,k_1}=0]\mathbb{P}[m_{j,k_2}=1]\mathbb{P}[m_{i,k_2}=0]\big) \\
&= \textstyle\sum_{j,k_1,k_2}\big(\frac{n_i}{K}(1-\frac{n_j}{K})\frac{n_j}{K}(1-\frac{n_i}{K})\big) = \frac{n_i(K-n_i)}{K^2}\sum_j\sum_{k_1,k_2}\frac{n_j(K-n_j)}{K^2} \\
&= \frac{n_i(K-n_i)}{K^2}\textstyle\sum_j(n_jK-n_j^2) = \frac{n_i(K-n_i)}{K^2}(K^2L-\sum_j n_j^2) \\
&= n_i(K-n_i)(L-\textstyle\sum_j\big[\frac{n_j}{K}\big]^2)
\end{aligned}$$

$\Rightarrow$ maximal when $n_i=(K/2)$

N	# leechers	K	# file blocks	n_i	# blocks at P_i	μ	upload cap.
S	# seeders	L	# peers /w block b	$m_{i,b}$	1 iff b @ P_i	c	download cap.

17 of 47

Improvements

Redundant block download: If download at helper H starts to drop, H increases its attractiveness by downloading blocks the collector already has $\Rightarrow$ increases attractiveness for bartering.

Sharing swarm information: Pass information on new peers and the blocks held by the collector and other peers among the helping nodes $\Rightarrow$ considerable speedup during start phase.

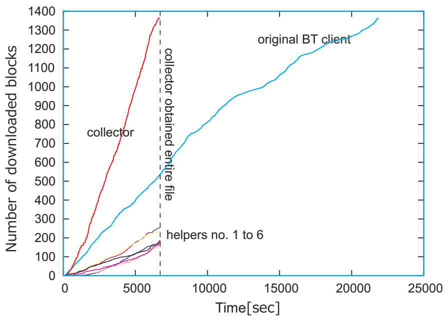
18 of 47

Performance

Experiment: Simply set up a combination of a collector with helpers and attach to existing BitTorrent swarms.

Upload/download bandwidth [kbps]	Speedup		Optimal number of helpers	
	Theoretical	Measured	Theoretical	Measured
682/1024	1.36	1.27	1	1
512/1024	1.82	1.72	2	2
256/1024	3.64	3.25	6	7
128/1024	7.27	6.4	14	17

Performance



19 of 47

Getting helpers

Issue: Why would any peer want to help: during a session a helper is not getting anything in return $\Rightarrow$ we need to look at **mechanism design**.

- Each peer p maintains a **local view** $V[p]$ of randomly selected peers from the network (e.g., by means of a peer sampling service).
- Each peer p maintains a set $C[p]$ of **contributors** (i.e., helpers it once made use of)
- Each peer p maintains a set $B[p]$ of **borrowers** (i.e., collectors to which it contributed bandwidth)

20 of 47

Algorithm: explore

Basics: We consider an explore and a select phase:
$\{n_p$: maximal # borrowers from peer $p\}$

---

explore:
while $(B[p] < n_p) \wedge (V[p] \backslash C[p] \neq \emptyset)$ **do**
select random peer $q \in V[p] \backslash C[p]$
$C[q] \leftarrow C[q] \cup \{p\}$ $\{p$ *offers services to* $q\}$
$B[p] \leftarrow B[p] \cup \{q\}$ $\{q$ *may become borrower of* p*'s bandwidth*$\}$
end while

Essence: explore potential new peers to offer bandwidth to, and later select the best ones.

21 of 47

22 of 47

Algorithm: select

select:
$\{r_p$: maximal # randomly selected borrowers from $p\}$
$B[p] \leftarrow \{q \in C[p] \mid q$ offered nonzero contribution$\}$
Sort $B[p]$ according to descending bandwidth
while $|B[p]| > n_p - r_p$ **do**
  Remove lowest-ranked peer from $B[p]$
end while

Essence: Give preference to helping peers that have helped you in the past.

Note: When a peer p wants download help, it can select peers from $C[p]$ according to some specific strategy (notably: peers to which p has provided help before).

23 of 47

Analysis

Input: Contribution c_i of peer i: average amount of bandwidth to borrowers. Gain g_i: obtained bandwidth from contributors.

Without proof: It can be shown that (1) $c_i > c_j \Rightarrow g_i \geq g_j$, and (2) maximize gain by contributing entire upload bandwidth.

Borrower set size: Intuitively, we would like to keep the values n_p small, in order to minimize risk of free-riding.

n	: size of borrower set (same for all peers)
$1/\lambda$	: average length of idle period (same for all peers)
$1/\mu$	: average length requesting period (same for all peers)

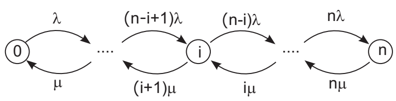
Assumption: helpers divide their bandwidth evenly among borrowers.

24 of 47

Analysis

Bandwidth utilization u_n: fraction of idle time when bandwidth is completely used by borrowers.

Observation: Evolution of # borrowers can be modeled as a birth-death process (with state representing # borrowers):



Required: $(n-i)\lambda\pi_i = i\mu\pi_{i+1} \Rightarrow \pi_i = \binom{n}{i}\left(\frac{\lambda}{\mu}\right)^i \pi_0$

25 of 47

Analysis

Condition: $\sum_{i=0}^{n} \pi_i = 1 \Rightarrow$

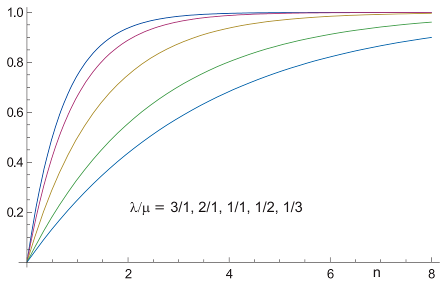
$$\pi_0 \sum_{i=0}^{n} \binom{n}{i} \left(\frac{\lambda}{\mu}\right)^i = \pi_0 \left(1 + \frac{\lambda}{\mu}\right)^n = 1$$

Assumption: Bandwidth is fully utilized when there is at least one borrower $\Rightarrow$

$$u_n = 1 - \pi_0 = 1 - \left(1 + \frac{\lambda}{\mu}\right)^{-n}$$

26 of 47

Analysis



Conclusion: We can keep n small while still attaining high utilization.

27 of 47

Collaborative CDNs

Issue: Replication for performance requires content to be placed at remote servers. **Question:** How can we enforce collaboration between servers that form a **CCDN**?

Specific case: Rather than considering all kinds of replication strategies, focus on simple case:

Peers need each other's help with bursty traffic in order to guarantee specific level of QoS.

Note: Matters are complicated by the fact that we demand that clients are left unmodified.

Enforcing collaboration

- Servers need to recognize each other and have repeated interactions ⇒ Fix the topology of the overlay.
- Deploy tit-for-tat ⇒ set up accounting
- Trust by check: enable verification of reported service provisioning by collaborative servers.

28 of 47

Fixing the topology

Approach: Simply have a centralized server hand out (1) IDs and (2) list of neighbors.

- When joining the system, a server places itself in the topology, and notifies its neighbors by handing over a certificate
- Is also a solution to **Sybil** attacks, whereby nodes can generate IDs at will.

29 of 47

Accounting

Observation: With a fixed set of neighbors, accounting has become much easier. Maintain two sets of data:

- **Volume of data exchanged:** Each peer p keeps track of two variables, for each of its neighbors q:
  - $Cons_p[q]$: total amount of data served by q on behalf of p
  - $Prov_p[q]$: total amount of data served by p on behalf of q

  Requirement: $Cons_p[q] - Prov_p[q] < M_p^{data}[q]$
- **Data rate exchanged:** Keep track of rate at which clients where served. Note: we need to motivate servers to provide good quality of service.

30 of 47

Accounting: data rate served

Issue: Do we keep an accurate account of rates and reciprocate exactly those? Not a good strategy, as it is far too rigid.

- $RateCons_p[q]$: trend for rate served by q on behalf of p
- $RateProv_p[q]$: trend for rate served by p on behalf of q
- Both values are regularly updated:

$$\begin{aligned} RateCons_p[q] &= \alpha \cdot RateConsNow_p[q] + (1-\alpha) \cdot RateCons_p[q] \\ RateProv_p[q] &= \alpha \cdot RateProvNow_p[q] + (1-\alpha) \cdot RateProv_p[q] \end{aligned}$$

- $RateAssigned_x[y]$: Rate assigned by x to serve clients of y

$$\begin{aligned} RateCons_p[q]/RateProv_p[q] &< M_p^{rate}[q] \Rightarrow \text{punish } q \\ RateCons_p[q]/RateProv_p[q] &> M_p^{rate}[q] \Rightarrow \text{reward } q \end{aligned}$$

Question: What are reasonable values for $M_p^{rate}[q]$?

31 of 47

Building trust

Observation: We can make the various limits $M_p[q]$ dependent on the observed behavior of a peer:

- $Cons_p[q]$ increases $\Rightarrow M_p^{data}[q] \leftarrow M_p^{data}[q] + \Delta Cons_p[q] \cdot \gamma_{incr}$
- $Cons_p[q]$ decreases $\Rightarrow M_p^{data}[q] \leftarrow M_p^{data}[q] \cdot \gamma_{decr}$

Important: decrease of $M_p^{data}[q]$ should be larger than increase with same value for $|\Delta Cons_p[q]|$. Other adaptations of $M_p^{data}[q]$ are also possible.

Big question: How do we verify claims from remote peers

32 of 47

Checking peers

Basic idea: Redirect clients to a peer, but break in into the connection to monitor progress and verify that progress against reports from peer:

- Client C contacts origin server O; O redirects C to peer P.
- After some time, O requests handover of (C,P) TCP connection.
- C will send data request with associated sequence number to $O \Rightarrow O$ can verify progress against reports from P.
- If all is well, connection can be handed back to P.

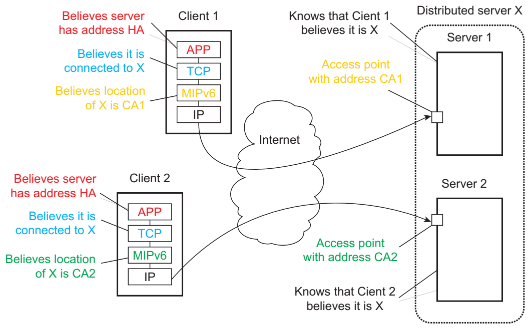
Issue: How to implement client-transparent TCP handoffs?

33 of 47

34 of 47

Solution: MIPv6

Basic idea: We can develop distributed servers with stable IPv6 addresses.



35 of 47

Distributed servers

Essence: Clients having MIPv6 can transparently set up a connection to any peer:

- Client C sets up connection to IPv6 home address HA
- HA is maintained by a (network-level) home agent, which hands off the connection to a registered care-of address CA.
- C can then apply route optimization by directly forwarding packets to address CA (i.e., without the handoff through the home agent).

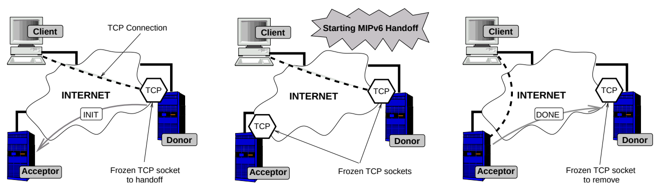
Collaborative CDNs: Origin server maintains a home address, but hands off connections to address of collaborating peer.

Effect: Origin server and peer appear as one server.

36 of 47

Side note: handoff details

Issue: efficiently handing off an entire socket:

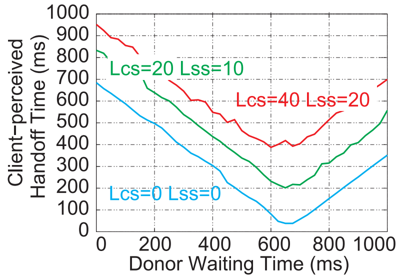


1. Send init message and freeze socket: drop incoming messages (will be retransmitted later).
2. Start handing off connection by transferring state.
3. Send done message; take over care-of address, and continue where TCP connection was frozen.

37 of 47

Handoff optimization

Issue: If donor hands off connection immediately after a send() call, the socket buffer will be full ⇒ wait until buffered data has been sent to the client (i.e., empty TCP buffers).

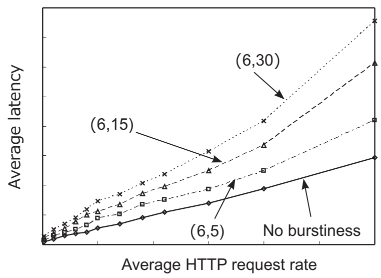


L_{CS}: client-to-server latency
L_{SS}: server-to-server latency

38 of 47

Evaluation

Question: Do we actually need help in the case of bursty traffic?



(B,F): F% of the time, we have a burst B times the average rate.

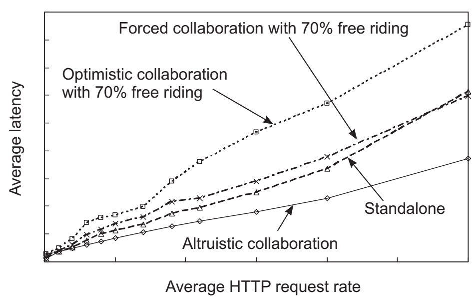
Collaborators are altruistic

Observation: With bursts, the server's performance degrades worse than linear.

39 of 47

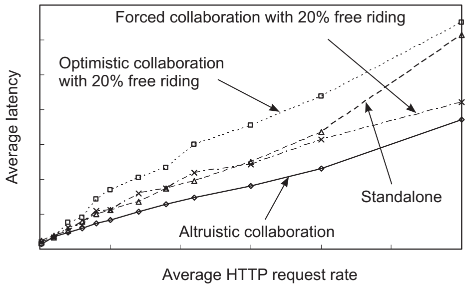
Enforcing collaboration

Comparison: Consider the $(6,15)$ burstiness case:



Enforcing collaboration

Comparison: The $(6, 15)$ burstiness case with 20% free riding:



40 of 47

Food for thought

So far: We've been discussing various mechanisms to enforce or promote collaboration among peers. A realistic question is: **do we really need such mechanisms**?

- Wikipedia relies entirely on altruistic collaboration (and some centralized control)
- BitTorrent networks often exhibit altruistic seeding behavior: we may need only simple mechanisms to enforce collaboration
- Social and economic factors appear to influence seeding behavior

Observation: It appears there is no need to be extremely pessimistic.

41 of 47

Artificial Social Networks

Question: Can we let computers form a network in which collaboration prevails?

Experiment: Consider a large collection of nodes organized in an unstructured peer-to-peer network:

- Each node i has an observable utility u_i.
- Each node i follows a published strategy s_i, which may change over time.

Basics: Copy the strategy from nodes that are apparently doing better than you (i.e., their utility is higher).

42 of 47

43 of 47

The SLACER algorithm

Select a random node j {*e.g., using a peer sampling service*}
if $u_{me} \leq u_j$ **then**
  $s_{me} \leftarrow s_j$ {*copy j's strategy*}
  for all $k \in N(me)$ **do**
    With probability $\mathbb{P}(W)$: remove k from $N(me)$
  end for
  $N(me) \leftarrow N(me) \cup N(j) \cup \{j\}$
  Remove random elements from $N(me)$ until $|N(me)| = c$
  With probability $\mathbb{P}(S)$: change strategy s_{me}
  With probability $\mathbb{P}(L)$:
  for all $k \in N_{me}$ **do**
    With probability $\mathbb{P}(W)$: remove k from $N(me)$
    $N(me) \leftarrow N(me) \cup \{$random node $x\}$
  end for
end if $u_{me} \leftarrow 0$

Note: $N(i)$ is the current set of neighbors of i

44 of 47

SLACER effects

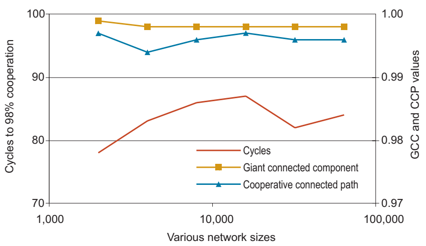
Application: Play the prisoner's dilemma game, with two strategies: collaborate or defect.

- Utility is expressed in terms of payoffs $T > R > P > S$ (reward single defect; reward collaboration; punishment mutual defect; sucker's payoff).
- Performance metrics: average path length, cluster coefficient
- Additional metric: cooperatively connected path:
  - i and j are CCP-connected if there is a (i, j) path with only cooperative intermediate nodes.
  - Measure the fraction of CCP-connected node pairs.

45 of 47

Evaluation

Initially: Random network and all strategies set to defect. $\mathbb{P}(S) = 0.001$; $\mathbb{P}(L) = 0.01$; View list size $c = 20$.



Observation: With the right payoffs, collaboration results

Conclusions

- There is a lot to gain from collaboration in distributed systems
- Tit-for-tat mechanisms appear to be workable in practice
- Issue: designing mechanisms by which strategies can be checked
- Issue: unclear to what extent only technical solutions are needed

Overall: There is a lot of room for research

46 of 47

Reading material

[1] R. Axelrod. *The Evolution of Cooperation*. Basic Books, 1984.

[2] P. Garbacki, D. Epema, and M. van Steen. "An Amortized Tit-For-Tat Protocol for Exchanging Bandwidth instead of Content in P2P Networks." In *Proc. First Int'l Conf. Self-Adaptive & Self-Organizing Syst.*, June 2007. IEEE Computer Society Press, Los Alamitos, CA.

[3] P. Garbacki, A. Iosup, D. Epema, and M. van Steen. "2Fast: Collaborative Downloads in P2P Networks." In *Proc. Sixth Int'l Conf. Peer-to-Peer Comput.*, Sept. 2006. IEEE Computer Society Press, Los Alamitos, CA.

[4] D. Hales and S. Arteconi. "SLACER: A Self-Organizing Protocol for Coordination in Peer-to-Peer Networks." *IEEE Intelligent Systems*, 21(2):29–35, Mar. 2006.

[5] M. Szymaniak, G. Pierre, M. Simons-Nikolova, and M. van Steen. "Enabling Service Adaptability with Versatile Anycast." *Conc. & Comput.: Prac. & Exp.*, 19(13):1837–1863, Sept. 2007.

47 of 47