Rescuing Tit-for-Tat with Source Coding

Thomas Locher, ETH Zurich

Stefan Schmid, ETH Zurich

Roger Wattenhofer, ETH Zurich

Distributed Computing Group

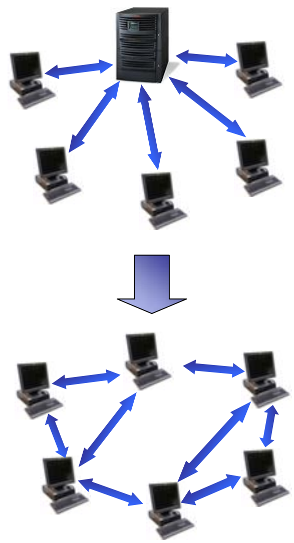
7th IEEE Int. Conference on Peer-to-Peer Computing (P2P)

Galway, Ireland, September 2007

Motivation: Collaboration is mandatory!

P2P computing has many advantages over the traditional client server model:

- Increased scalability
- Better use of bandwidth
- Fault tolerance
- ...



However, it only works if peers cooperate → All p2p systems crucially depend on collaboration!

How can collaboration be guaranteed?

Motivation: Solutions in practice

Do popular file sharing networks guarantee a fair sharing of resources?

Examples:

- FastTrack: No.

„Participation level" can be manipulated.

Kazaa Lite sets it to the maximum (1000)

- eDonkey: No.

Local credits can improve the peer's position in the queue, but otherwise no incentives to upload.

- Gnutella: No.

There are many studies about free riding on Gnutella. Most users do not share anything!

The free riding client BitThief never uploads anything!

- BitTorrent: No.

Its weak incentive mechanism encourages users to upload, but uploading is not enforced.

Motivation: Incentive Mechanisms

1. BitTorrent uses a tit-for-tat-like mechanism where uploading peers are favored. All peers repeatedly get a chance to reciprocate ("optimistic unchoking").

Weaknesses of BitTorrent:

- The seeders can be exploited.
- The "optimistic unchoking" can be exploited.

Seeders do not only "seed" the file, they give it out for free!

These weaknesses can also be considered "features"…

2. A centralized server to enforce fair sharing could be used: Every data exchange is monitored by the server.

Weaknesses of this approach:

- Limited scalability
- Single point of failure…

For example, Dandelion uses the approach!

Motivation: Why not Tit-for-Tat?

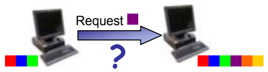
Tit-for-tat is believed to be the most effective strategy to enforce collaboration.

Initially cooperate and then respond in kind to the other peer's previous action!

Why isn't this simple strategy used in file sharing networks???

Short answer: Because it does not work (if applied directly):

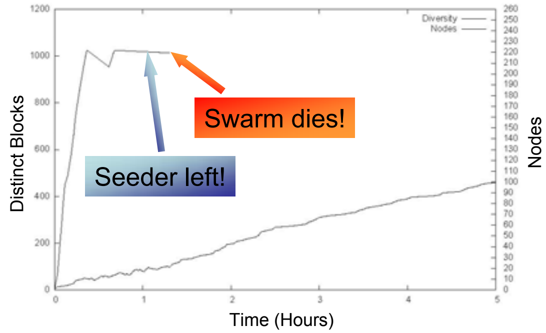
- Bootstrap problem: Initially, peers have nothing to share.
- Deadlocks: Nothing to offer to other peers!



Motivation: Selfish Behavior

System based on T4T:

- Peers exchange blocks using the tit-for-tat strategy.
- Peers leave after downloading all blocks.
- Single seeder leaves after ≈1h.



→ 17 minutes later, peers can no longer finish their downloads, because some blocks are not available anymore!

→ Such a system is inefficient (deadlocks) and often fails (peers leave) in selfish environments!

What can be done to solve this problem...?

Outline

I. Motivation

II. System Overview

III. Evaluation

IV. Conclusion

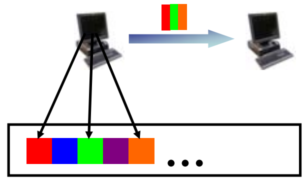
System Overview: Source Coding

Basic idea: If m blocks from a much larger set of blocks suffice to reconstruct the file and much more than m blocks are in circulation („block diversity"), the deadlock problem can be mitigated!

How can the block diversity be increased? The blocks are encoded at the seeders (source coding):

k random blocks are combined into a new block.

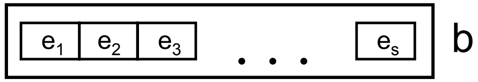
The total number of blocks increases from m to *m choose k* (# blocks $\in O(m^k)$)!



How are the blocks encoded?

System Overview: Finite Field

Each block b is interpreted as a sequence of elements e_i from a finite alphabet.

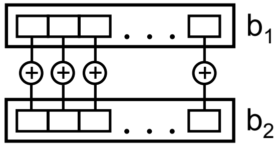


In network coding, the field GF(2^q) is used!

The elements are taken from the finite field GF(2^x - 1) → Computations are carried out modulo the *Mersenne prime number* P = 2^x - 1.

(For example, we used P = 2^{31} - 1 and a block size of 128 KB, resulting in s = 33,825 elements per block)

When two blocks b_1 and b_2 are combined, the elements at the same positions are added up!



ADD, not XOR!

System Overview: Algebra

All basic arithmetic operations are efficient:

What:	How:
e_1+e_2	Bitwise addition + add carry-over bit
-e	Flip all bits
$e_1 \times e_2$	Bitwise multiplication (using addition from above)
1/e	Extended Euclidean Algorithm

How many blocks are added up?

How can the original blocks be reconstructed?

System Overview: Small Parameter k

Simulations show: Combining k = 2, 3... blocks suffices to boost the block diversity.

However, k must be larger, otherwise the resulting coefficient matrix C does not have rank m!

If rank(C) < m, **more than m blocks** have to be downloaded!

If k is slightly larger than log m, the rank of C is practically always m!

→ Exactly m blocks have to be downloaded, which is optimal!

Advantages over regular network coding:

No coding at the leechers required!

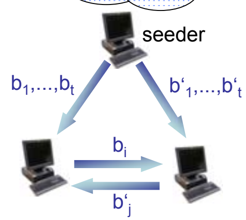
- Every block occurs at most once in every encoded block → Simple bitmap as a representation is enough!
- The leecher strategy is simple: Play tit-for-tat with all neighboring peers and download every encoded block that is not locally available!

System Overview: Seeder Strategy

The following rules prevent free riders from exploiting the seeders:

Chosen randomly from all *m choose k* possible blocks!

- Each peer can download ***only*** a small, *specific* pseudo-random subset of the blocks!
- If there are n peers and m blocks, the seeders adaptively set the size of this subset to $t \approx m/n$.



Advantages of this approach:

- Different peers obtain entirely different blocks.
→ Large block diversity!
- Leechers are forced to collaborate.
- Seeders have to provide only little data.

It is cheap to be (and remain) a seeder!

Outline

I. Motivation

II. System Overview

III. Evaluation

IV. Conclusion

Evaluation: Decoding Time

Reconstruct the original blocks:

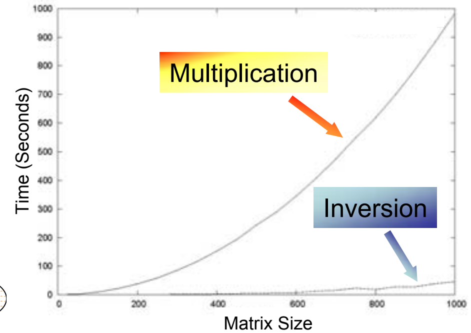
→ Invert the coefficient matrix C

→ Multiply C^{-1} with all blocks

Inversion: $O(m^3)$ time

Multiplication: $O(m^2 s)$ time

s ≫ m: Multiplication dominates the decoding time!



Reducing the decoding time:

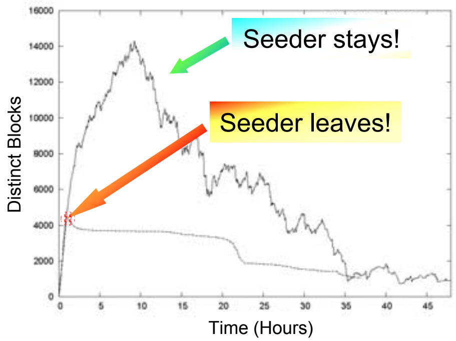
Increase the block size? → Freeloading becomes possible!

Group blocks together? → Reduces the decoding time! → Creates dependencies...

Evaluation: Block Diversity

Simulation scenario: Flash crowd!

- 2000 peers arrive
- Peers leave after downloading m=1024 blocks
- Block size = 128 KB
- # Blocks combined: k = 12



Two cases:

1) One seeder stays forever
2) The seeder leaves after uploading 4·m blocks

2‰ of all data that needs to be exchanged!

We learn that:

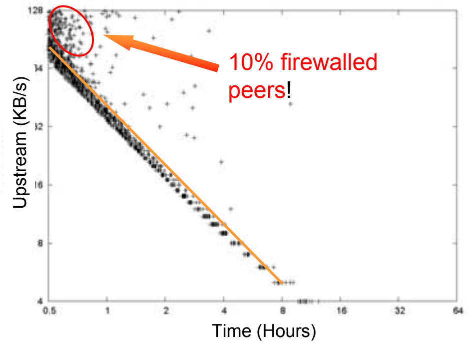
• The block diversity in the first case is larger!

• In the second case, the block diversity is large enough!!!

Evaluation: Download Time

The download time correlates with the upload bandwidth!

→ This indicates that the system is fair!

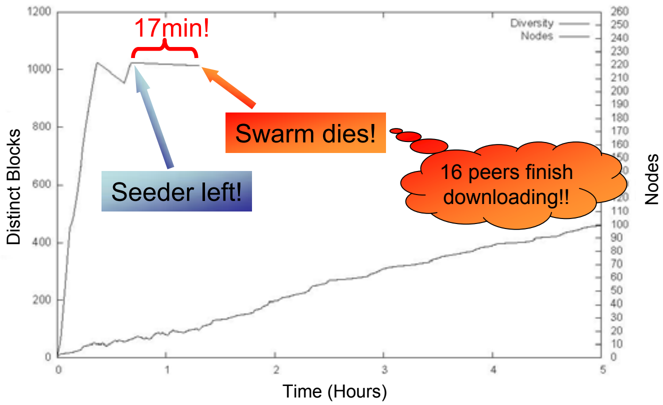


Firewalled peers cannot open enough connections to other peers

→ Longer download times!

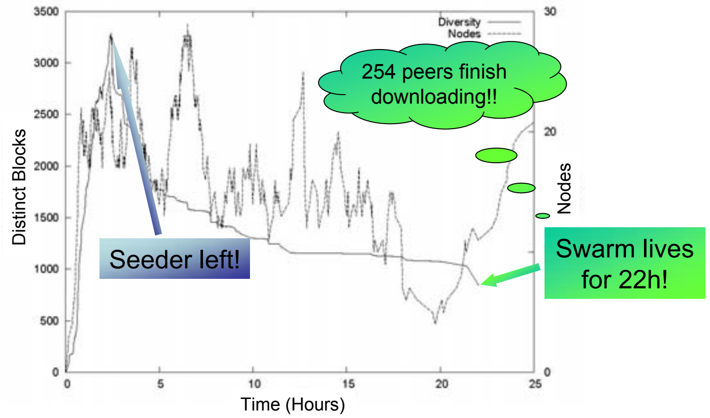
Normally ≈ 10 connections!

Evaluation: Performance Without Coding



- 300 peers arrive
- Peers leave after downloading m=1024 blocks
- Block size = 128 KB
- Seeder leaves after uploading 4·m blocks (≈1h)

Evaluation: Performance With Coding



- 300 peers arrive
- Peers leave after downloading m=1024 blocks
- Block size = 128 KB
- Seeder leaves after uploading 4·m blocks (≈1h)

Outline

Conclusion

- Source coding to ensure fairness
  - Increased block diversity keeps network alive!
  - New seeder strategy: Seeders cannot be exploited.
  - Leechers must engage in fair tit-for-tat exchanges.
- Different encoding technique
  - Simple block representation!
  - The matrix can be kept sparse!
- Main challenge
  - Reducing the decoding time...

Tit for tat

Questions and Comments?

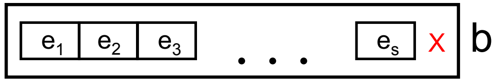
Thank you for your attention!

Thomas Locher
Distributed Computing Group
ETH Zurich, Switzerland
lochert@tik.ee.ethz.ch
http://dcg.ethz.ch/members/thomasl.html

Additional Slide: Disadvantages

The system has a few disadvantages:

- Computations modulo $P = 2^x - 1$ → $\overbrace{00\ldots0}^{x} \equiv \overbrace{11\ldots1}^{x} \pmod{P}$!
- One bit X „missing" in the encoding of each block:



A „helper block" solves the two problems:

- Store indices where the element 11...1 occurs
- Store the last bit of each block separately

> Very rare in **compressed files!**

> Only **1KB** if file size is **1GB!**