

# Partial Content Distribution on High Performance Networks

Eric Weigle<sup>1</sup>    Andrew A. Chien<sup>1,2</sup>

<sup>1</sup>University of California, San Diego

<sup>2</sup>Intel Research

29 June 2007 – HPDC 2007 Conference

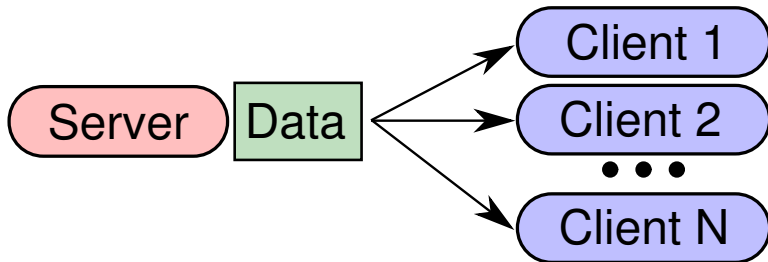
# Outline

- 1 Introduction - Background and Problem
- 2 Approach - Algorithms and Implementation
- 3 Evaluation - Environment and Experiments
- 4 Conclusions - Summary, Other Work, Questions

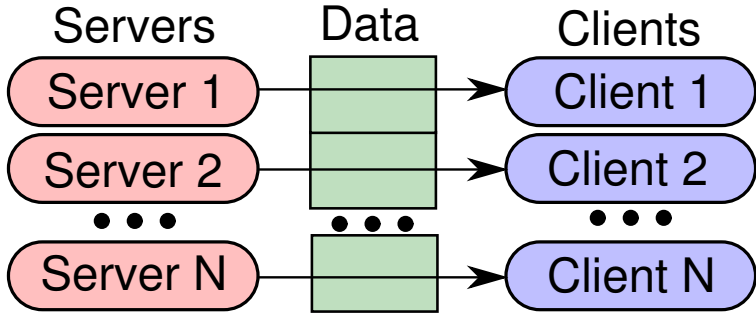
# Introduction

- Content distribution (traditional)
  - Simple model: One-to-many
  - Some known “optimal” approaches (Network coding, multicast meshes)
  
- *Partial* content distribution
  - Richer model: Many-to-many
  - Traditional CDNs don't work
  - High performance?

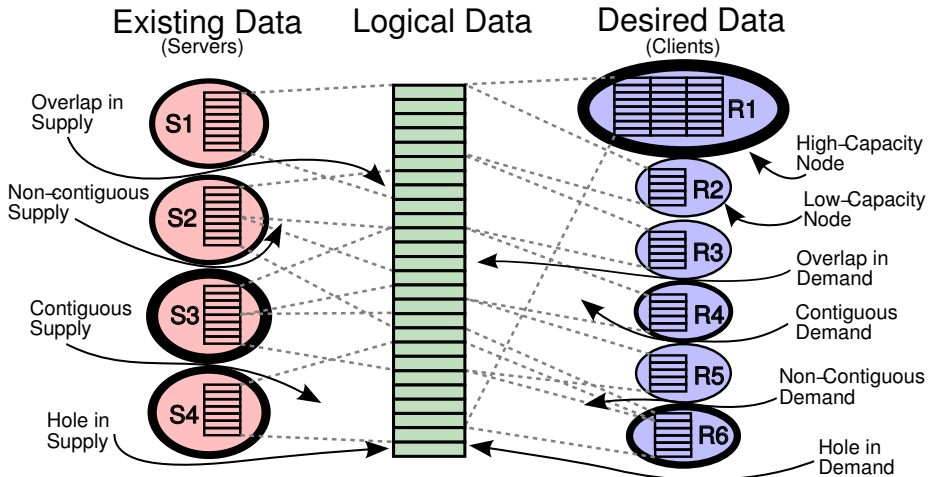
# Problem



# Problem



# Problem



# Outline

- 1 Introduction - Background and Problem
- 2 Approach - Algorithms and Implementation**
- 3 Evaluation - Environment and Experiments
- 4 Conclusions - Summary, Other Work, Questions

# Proposed Solution

- 1 Build measurement/metadata infrastructure
  - Capture **current** state:  
network latency, capacity, block location
  - Capture **desired** state: block location
- 2 Apply transfer scheduling algorithms
  - Various optimization techniques;  
e.g. linear programming, network flow, hill climbing,  
network coding, ...
  - **Claim:** A greedy proportional-sharing algorithm is  
simple, efficient, and produces good results

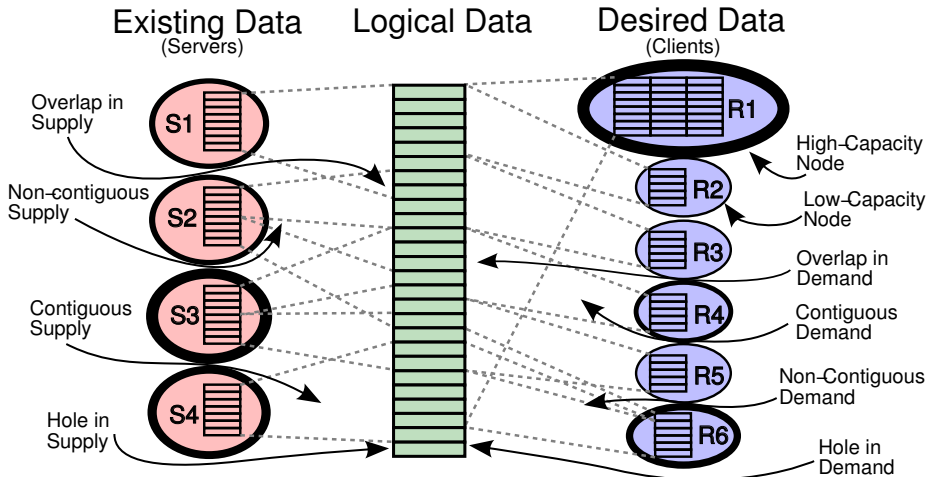
# The Composite Endpoint Protocol

- CEP provides infrastructure and scheduling
  - Infrastructure: “Intelligent glue”
  - Scheduling: proportional supply/demand sharing but not necessarily *fair* sharing.
  - Nothing to do with Complex Events; but it’s got a database, and is event driven...
- Exists in two forms:
  - 1 Applications, like a `gsiftp` analogue
  - 2 Shared library + API: what we’ll use here

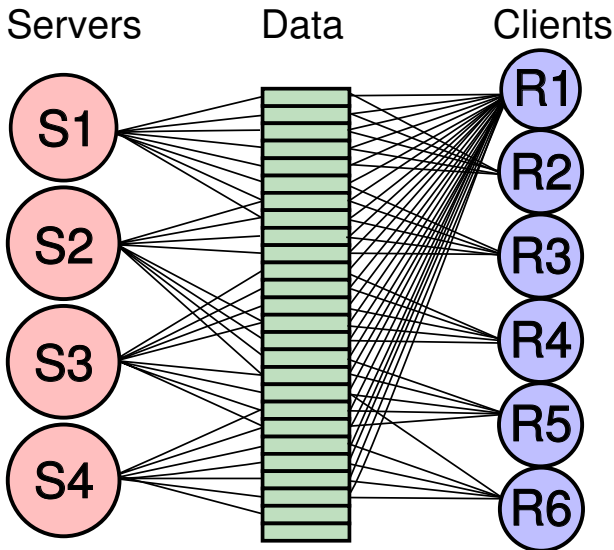
# Canonical Input: Segment/Block Graph

- A graph representing constraints
  - Nodes are senders, receivers, and simple data ranges
  - Directed **data** edges between those who provide/require
  - **Network** edges mirroring physical topology (if known)
- Find a weighting of links
  - Proportion of outgoing bandwidth allocated to each chunk of data.

# Canonical Input: Segment/Block Graph



# Canonical Input: Segment/Block Graph



# Basic Greedy Algorithm (Low Sharing)

- 1 Calculate proportional *demand*:
  - Sum demand over all receivers/data, proportional to their speed
  - i.e. How fast everybody wants to download each block
- 2 Calculate proportional *supply*:
  - Each sender allocates capacity proportional to remaining demand
- 3 Determine rate schedule:
  - Each sender equally allocates rate to receivers per segment.
- 4 Execute Transfer
  - Nodes locally optimize
  - *Recalculate as necessary*

# Optimizing for High Sharing

Two-phase process; 'seed' and 'feed'

- 1 Seed: Distribute blocks intelligently;  
→ Supply proportional to demand.

Run basic greedy algorithm; cache values

For each receiver:

Seed a block when:

I want the block

I have available upload capacity

Block's proportional supply not met

- 2 Feed: Run greedy transfer as above.

# Implementation

- BitTorrent is remarkably popular
  - 80% of the background traffic on the Internet
  - We use the BitTornado client
    - had features, source (Python), performance
- Our Changes/Additions:
  - Added block-level API/selective **block** download
  - Integrated CEP code, new **transfer scheduling algorithms**
  - Parameter tuning

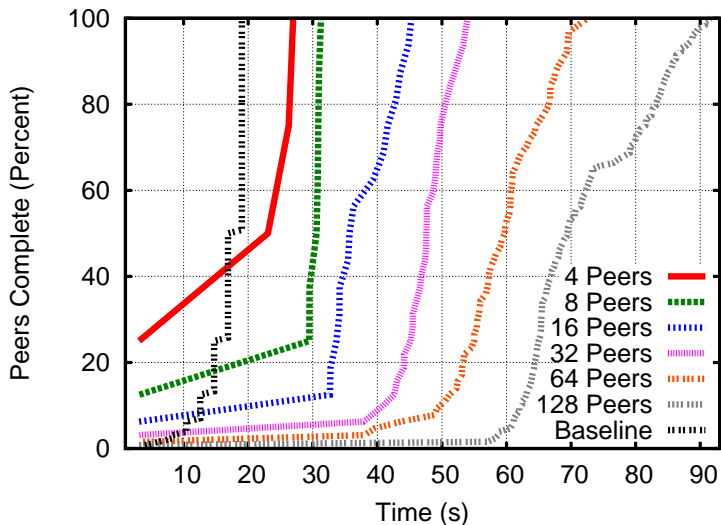
# Outline

- 1 Introduction - Background and Problem
- 2 Approach - Algorithms and Implementation
- 3 Evaluation - Environment and Experiments
- 4 Conclusions - Summary, Other Work, Questions

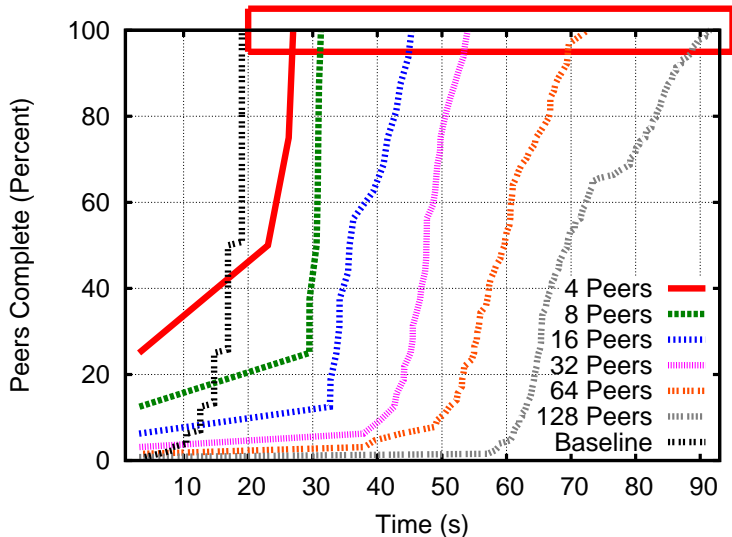
# Evaluation Environment

- Big picture: built the thing, now run it.
  - 1 Get a useful baseline
  - 2 test performance:  
bandwidth, latency, computation, fairness
- Use a typical “High Performance” cluster
  - 1Gbps links in the LAN, 10Gbps in the WAN
  - $\approx 300$  nodes available through sge batch scheduler
  - $\geq 1.6\text{GHz}$  CPU,  $\geq 2\text{GB}$  memory
- Dummynet to model higher-latency networks

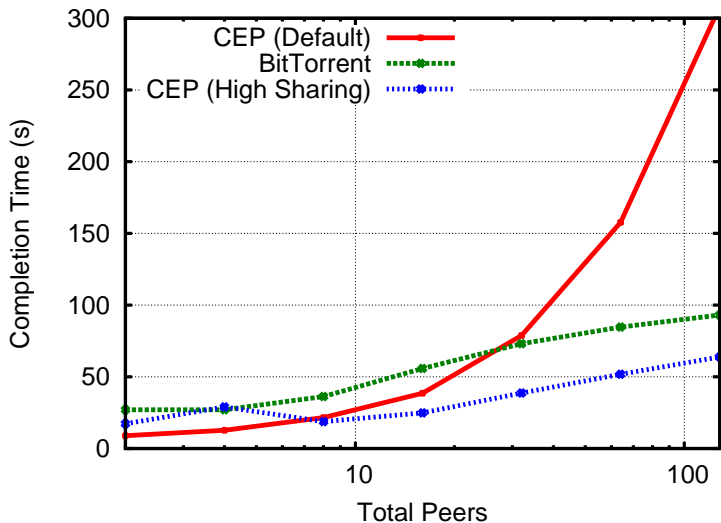
# BitTorrent 1 $\rightarrow$ $n$ Distribution



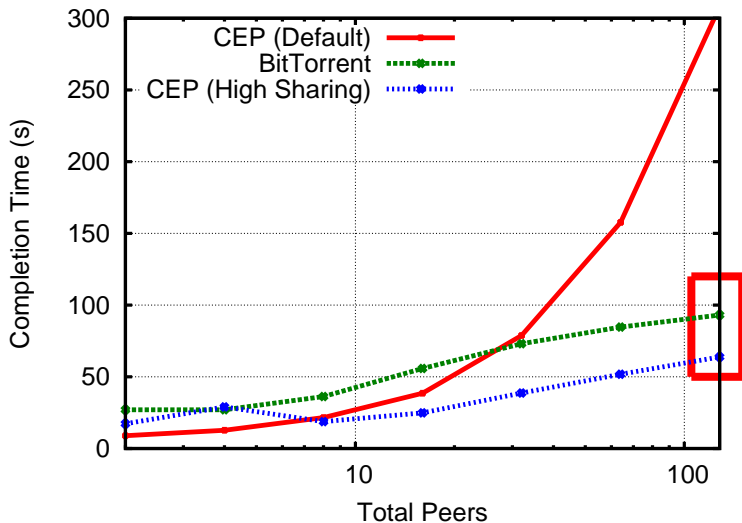
# BitTorrent 1 $\rightarrow$ $n$ Distribution



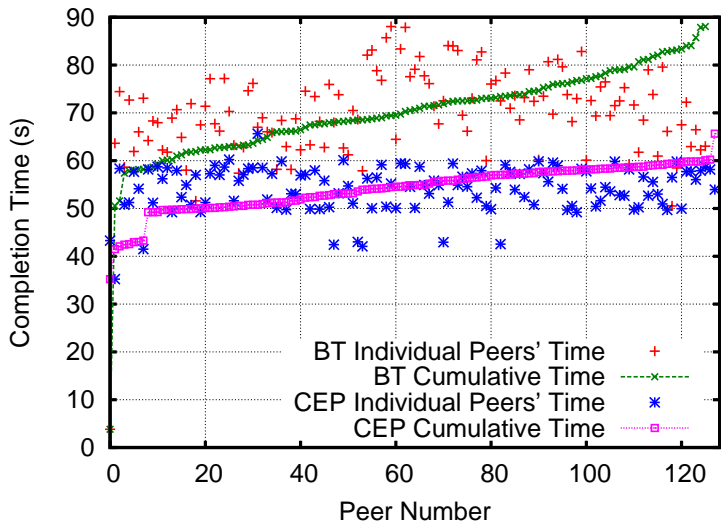
# Scalability; High Sharing Transfer



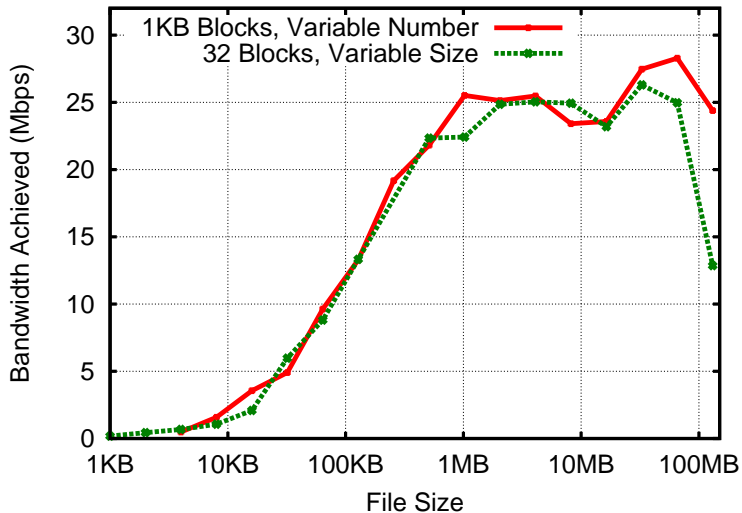
# Scalability; High Sharing Transfer



# Fairness



# High-Performance Tuning: Block Size



# Low-Performance Tuning: The Rest

Tuning Method	Time (s)	
	Mean	Ratio
Baseline	48.1	—
No Double Check	48.9	+1%
More Uploads	52.3	+8%
More Unchokes	52.5	+9%
Super-Seed	90.6	+88%
Larger Slices	154.9	+222%

# “Sharing” Quantified

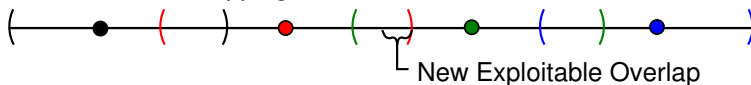
Sharing =  $avg_{\forall i}(avg_{\forall j}((|\{b_i\} \cap \{b_j\}|)/|\{b_i\}|))$

- 0 when nodes want **different** blocks
- $\frac{1}{2}$  when half want one set, half another
- 1 when nodes want the **same** blocks

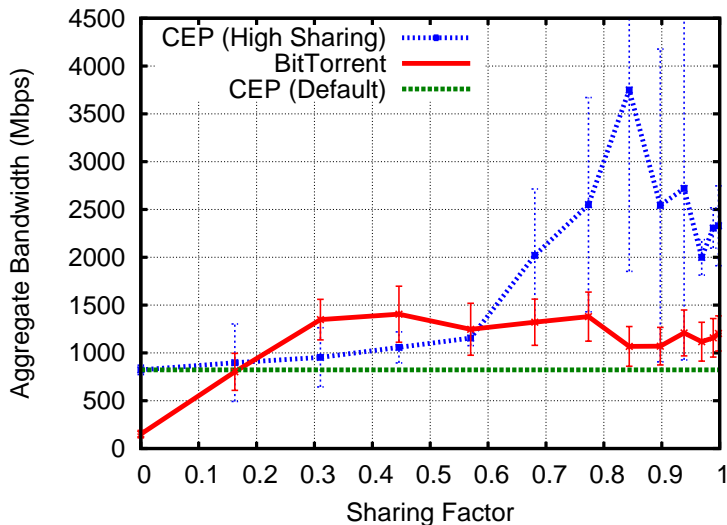
Virtual Line: Disjoint Peer Data



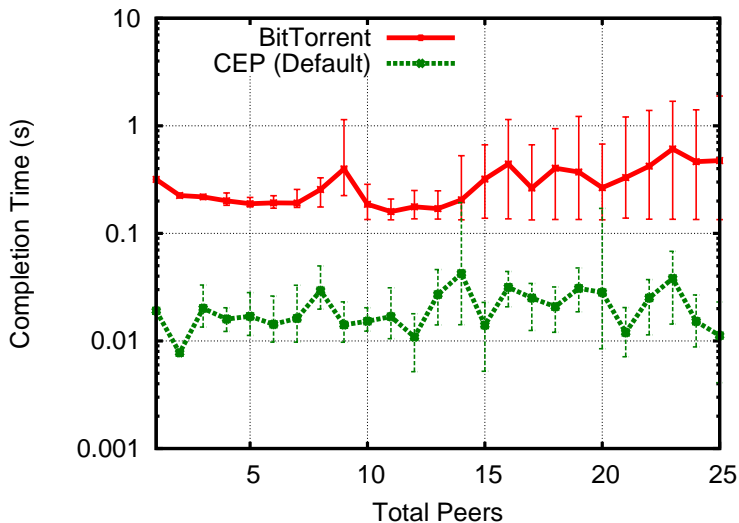
Virtual Line: Overlapping Peer Data



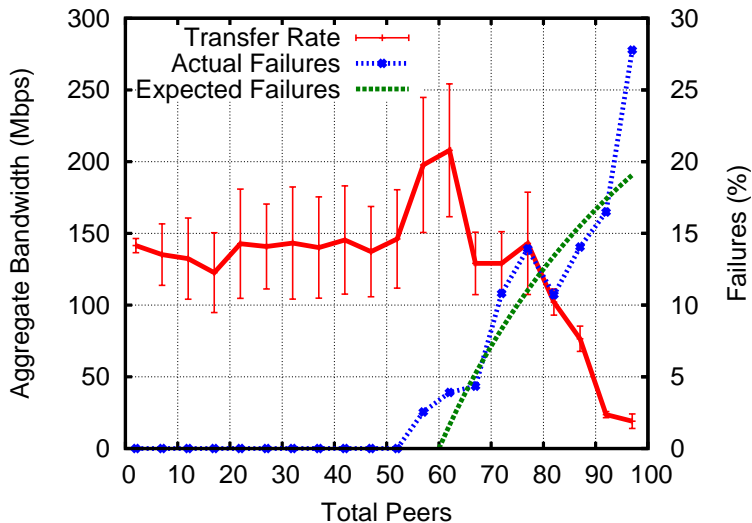
# Sharing and Performance



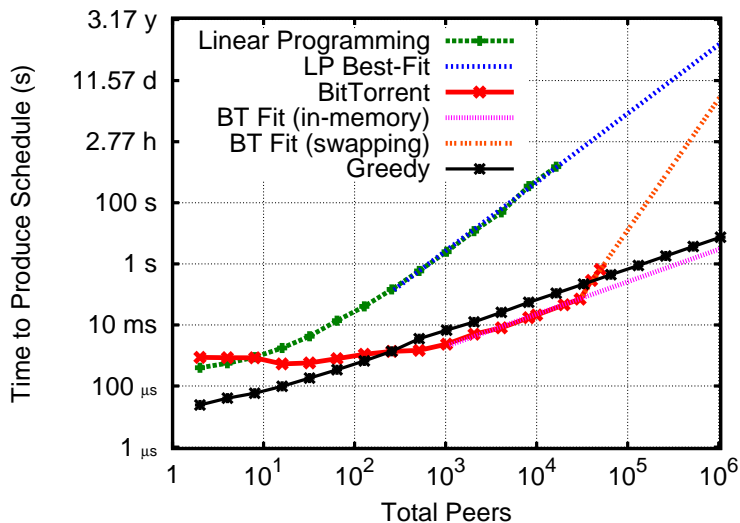
# Metadata(1): Look-up Latency



# Metadata(2): Failure of Striped BT



# Scheduling Cost



# Outline

- 1 Introduction - Background and Problem
- 2 Approach - Algorithms and Implementation
- 3 Evaluation - Environment and Experiments
- 4 Conclusions - Summary, Other Work, Questions

# Summary

Efficient, reasonably scalable, high-speed transfers under a variety of constraints.

- Partial content distribution is more general
- Default: Higher performance in low-sharing environments
- Optimized: Higher performance in high sharing environments
- In WAN, TCP performance determines CEP performance, leading to...

# Future & Related Work

- Other protocols such as:  
HS-TCP, FAST, UDT, RBUDP, etc.
- Better pipelining in 'seed' and 'feed' phases, or an equivalent fully distributed algorithm
  - A DHT (e.g. Pastry, Chord) isn't quite enough
  - Nor are erasure/network coding techniques
- Traditional CDNs; Bullet, SRM, Akamai, etc.
- Distributed file systems/memory; Lustre, GFS, etc.
- Logistical Networking (LoCI)
- Other scheduling ( $\lambda$ s, network flow, etc.)
- And the person in the audience I'm neglecting

Questions/Comments?



"Eric Weigle" <ehw@csag.ucsd.edu>

# References

- OptIPuter Project: <http://www.optiputer.net/>

# Erasure Codes

