Slurpie: A Cooperative Bulk Data Transfer Protocol

J. L. Chiang
MNET Lab Meeting
March 2, 2006

Outline

• Introduction
• Design goals and challenges
• The Slurpie Protocol
• Performance evaluation
• Conclusions and discussions
Introduction

Overloaded

X

C C C C C ...

20mins 20mins 30mins

? mins/hours
Observations
Basic Idea
Design Goals

• **Scalable**
  – Maintain load at the server independent of the # of clients

• **Beneficial**
  – Minimized the client download times

• **Deployable**
  – Deployable w/o infrastructure support (except for a minimal loaded demultiplexing host)

• **Adaptive**
  – Adaptive download strategy based on network condition and client capacity

• **Compatible**
  – Require no server-side changes
Related Works

• Multicast
  – IP multicast, application layer multicast (EMS)

• Infrastructure-based
  – CDN, web caches, mirrors

• Peer-to-peer
  – CoopNet, BitTorrent

  • Design for small HTML files
  • All transfer involve the server

  • Explicitly provisioned for only certain load levels

  • Not adapt to varying bandwidth conditions
  • Not scale its # of neighbors as the group size increases
  • Tracker limits the scalability to thousands of nodes
The Slurpie Protocol

1. TS \(\rightarrow\) C7, C6, C5, C4, C3
   - Get file1

2. C8
   - add neighbor C4

3. C8
   - add neighbor C6
   - add neighbor C3

4. C8
   - Get A
   - Block A
   - HTTP Get
   - Data

MNET - J. L.
Challenges

• Maintain exact state about all peers downloading a file
• How the mesh is formed
• How updates are propagated
• Decide whether to contact server or peers
• How many blocks to divide the files into
• How many connections each peer should open
Mesh Formation

• TS maintains and returns the last $\phi$ seed nodes that queried the TS for that same file.
  – $\phi$ is a small constant.

• The newly joined node makes bidirectional links to a random set of these seed nodes, and tries to discover $\eta$ neighbors to maintain
  – $\eta$ is updated depending on available bandwidth.
  – $\eta \geq O(\log n)$ to ensure that the mesh stays connected with high probability.
  – $n$ is the estimated number of nodes in the mesh.
Group Size Estimation

- From random graph theory, for an $r$-regular graph, the mean distance $d$ between nodes is proportional to $\log_{r-1} n$.
- $n = O((r - 1)^d)$
  - Use $U$ updates to estimate $d$ by hop_count and $r$ by degree.

<table>
<thead>
<tr>
<th>$n$</th>
<th>20</th>
<th>50</th>
<th>100</th>
</tr>
</thead>
<tbody>
<tr>
<td>200</td>
<td>17.5%</td>
<td>13.2%</td>
<td>10.9%</td>
</tr>
<tr>
<td>1000</td>
<td>5.9%</td>
<td>4.2%</td>
<td>2.6%</td>
</tr>
<tr>
<td>5000</td>
<td>11.3%</td>
<td>7.5%</td>
<td>6.0%</td>
</tr>
<tr>
<td>10000</td>
<td>3.8%</td>
<td>0.8%</td>
<td>0.4%</td>
</tr>
</tbody>
</table>

TABLE II
% ERROR IN GROUP SIZE ESTIMATION
Update Propagation (1/2)

• The update message:
  – <IP-addr, port, block-list, hopcount, node-degree>

• The rate of updates passed along each link per second, $\sigma$, is subject to an AIMD flow control algorithm based on available bandwidth estimates.
Update Propagation (2/2)

- The update tree:
  - Bit vectors of $U$ nodes

![Update Tree Diagram]

None of known peers have this block. Try to approach the server (based on backoff).
Bandwidth Estimation (1/2)

• Three states:
  – underutilized, at-capacity, throttle-back

• $B_{max}$: coarse grained bandwidth setups
  – “Modem”, “T1/DSL”, “T3”, …

• $B_{act}$: actual achieved throughput over all data connections over a 1s interval

• $avgB$: moving average of $B_{act}$, along with a $std$
Bandwidth Estimation (2/2)

• **Underutilized:**
  – $B_{act} < B_{max} - std$
  – $\eta$ ++, $\sigma$ ++, data_connection ++

• **Throttle-back:**
  – $B_{act} > avgB + std$
  – $\eta$ --, $\sigma$/2, data_connection --

• **At-capacity:**
  – Others
  – No change on system parameters
Download Decisions

- When multiple peers have the same block, a peer is chosen at random.
- A node prefers to download from the peer with established connection to take advantage of an open TCP window.
- The bandwidth estimation algorithm is queried every second, if it returns underutilized, and there exist hosts that have blocks that the local node does not have, a new connection is opened.
Random Backoff (1/2)

• Control the load on the source server independent of peers in the system.

• Ideally, the host with the best connection to the server would be the sole machine connected to the server, and everyone else receive their data from this host.
  – Finding the best host is difficult.
  – The best host could download the data and then leave.
Random Backoff (2/2)

- Every time period $\tau$, each eligible peer decides to go to the server with probability $k/n$.
  - $n$ is the estimate of the nodes in the system.
  - $k$ is a small constant.
  - (setting $k=1$ results in no connections at the server for about 30% of extended periods of time)
  - (Setting $k=3$ keeps at least one connection at the server about 90% of the time)
  - $\tau$ is chosen to be long enough for progress but short enough for fairness.
Block Size

- Fixed 256KB block size is chosen.
  - It is the smallest size at which the TCP overhead was effectively amortized (<1%).
  - It keeps the bit vector to a manageable size for large files (50 bytes for a 100MB file).
Experimental Setup

- Local Testbed Setup
- PlanetLab Setup

![Diagram of Local Testbed Setup]

48 Linux Clients
1 Gb/s
10 Mb/s
100 Mb/s

![Diagram of PlanetLab Setup]

48 Linux Clients
10Mbps
95Mbps
1Gbps

Internet
Internet2
# Parameter Setup

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>$k$</td>
<td>$k/n$ clients go to server</td>
<td>3</td>
</tr>
<tr>
<td>$\tau$</td>
<td>Server connection length</td>
<td>4 seconds</td>
</tr>
<tr>
<td>$\sigma$</td>
<td>Initial Update Rate</td>
<td>8/second</td>
</tr>
<tr>
<td>$\eta$</td>
<td>Initial Number of Neighbors</td>
<td>10</td>
</tr>
<tr>
<td>$m$</td>
<td>Mirror Time (described below)</td>
<td>2 seconds</td>
</tr>
<tr>
<td>$U$</td>
<td>Number of Updates Stored</td>
<td>100</td>
</tr>
<tr>
<td>$\psi$</td>
<td>Per File State at Topology Server</td>
<td>5</td>
</tr>
</tbody>
</table>

**TABLE I**

**Default Slurpie Parameters**
Completion Times (Local)

Factor = experiment time / baseline time
Completion Times (PlanetLab)

![Graph showing completion times for Slurpie and Bittorrent]
CDF of Completion Times

![CDF of Completion Times Graph]

Slurpie
BitTorrent
(both with 48 clients)
Completion time of Client

Completion Time for Single Client with No Contention (93 seconds)

Slurpie w/ 245 client, each arrives 3s after previous one
Effect of Benevolence

![Graph showing the effect of benevolence over time.](Image)
Effect of Backoff on Server

![Graph showing the effect of backoff on server connections over time.]

- No Backoff
- With Backoff, $k=3$
Effect of Backoff on Client

![Graph showing the effect of backoff on client performance. The x-axis represents the number of clients, 3s apart, while the y-axis represents factor improvement. Two lines are shown: one for backoff on (solid line) and one for no backoff (dashed line). The graph indicates a significant increase in factor improvement with backoff enabled.](image-url)
Effect of Flash Crowds

![Graph showing the effect of flash crowds on server connections]

- 10 clients
- 20 clients
- 32 clients
- 48 clients
Summary

• Slurpie protocol fulfills the design goals of system scalability, improved client performance and insulation of the server from load variance in the client population.

• Future works
  – Internet-wide deployment
    • initial mirror sites, slurpie proxy
  – Better estimation of network size
Discussions (1/3)

- Clients download parts of files from other clients without accessing highly contested server resources.
- BT v.s. Slurpie

Effect of tit-for-tat
Discussions (2/3)

- Effect of piece (block) selection algorithms
  - Last block problem, peer utility
Discussions (3/3)

• A larger # of clients should be experimented.
• The overhead of update propagation should be considered.
• Which data connection should be closed when throttle-back.