A Stable Broadcast Algorithm

Kei Takahashi             Hideo Saito
Takeshi Shibata           Kenjiro Taura
(The University of Tokyo, Japan)
Broadcasting Large Messages

- To distribute the same, but large data to many nodes
  - Ex: content delivery

- Widely used in parallel processing
Usually, in a broadcast transfer, the source can deliver **much less** data than a single transfer from the source.
Pipeline-manner transfers improve the performance.

Even in a pipeline transfer, nodes with small bandwidth (slow nodes) may degrade receiving bandwidth of all other nodes.
1. Propose an idea of **Stable Broadcast**

*In a stable broadcast:*

- Slow nodes *never* degrade receiving bandwidth to other nodes

- **All nodes** receive the *maximum* possible amount of data
Contributions (cont.)

2. Propose a stable broadcast algorithm for tree topologies
   - **Proved** to be stable in a theoretical model
   - Improve performances in general graph networks

3. In a real-machine experiment, our algorithm achieved **2.5 times** the aggregate bandwidth than the previous algorithm (FPFR)
- Introduction
- **Problem Settings**
- Related Work
- Proposed Algorithm
- Evaluation
- Conclusion
Problem Settings

1. Target: large message broadcast

2. Only computational nodes handle messages
3. Only bandwidth matters for large messages

\[ \text{(Transfer time)} = (\text{Latency}) + \frac{1\text{GB}}{(1\text{Gbps})} \]

50msec

4. Bandwidth is only limited by link capacities

Assume that nodes and switches have enough processing throughput
5. Bandwidth-annotated topologies are given in advance

- Bandwidth and topologies can be rapidly inferred

  - Naganuma et al. Improving Efficiency of Network Bandwidth Estimation Using Topology Information (SACSIS 2008, Tsukuba, Japan)
Evaluation of Broadcast

- Previous algorithms evaluated broadcast by completion time

- However, it cannot evaluate the effect of slowly receiving nodes
  - It is desirable that each node receives as much data as possible

- **Aggregate Bandwidth** is a more reasonable evaluation criterion in many cases
**Definition of Stable Broadcast**

- **All nodes** receive **maximum** possible bandwidth

  - Receiving bandwidth for each node does not lessen by adding other nodes to the broadcast

---

**Single Transfer**

- Single Transfer

---

**Broadcast**

- 100 → 10 → 120 → 100
Properties of Stable broadcast

- **Maximize** aggregate bandwidth
- **Minimize** completion time
Introduction

Problem Settings

**Related Work**

Proposed Algorithm

Evaluation

Conclusion
Single-Tree Algorithms

- Flat tree:
  - The outgoing link from the source becomes a bottleneck

- Random Pipeline:
  - Some links used many times become bottlenecks

- Depth-first Pipeline:
  - Each link is only used once, but fast nodes suffer from slow nodes

- Dijkstra:
  - Fast nodes do not suffer from slow nodes, but some link are used many times

< A Stable Broadcast Algorithm >
Kei Takahashi, Hideo Saito, Takeshi Shibata and Kenjiro Taura
FPFR Algorithm [†]

FPFR (Fast Parallel File Replication) has improved the aggregate bandwidth from algorithms that use only one tree.

Idea:

1. Construct multiple spanning trees
2. Use these trees in parallel

[†] Izmailov et al. Fast Parallel File Replication in Data Grid. (GGF-10, March 2004.)
Iteratively construct spanning trees

- Create a spanning tree ($T_n$) by tracing every destination.
- Set the throughput ($V_n$) to the bottleneck bandwidth in $T_n$.
- Subtract $V_n$ from the remaining bandwidth of each link.

**First Tree ($T_1$)**

**Second Tree ($T_2$)**
Each tree sends different fractions of data in parallel

- The proportion of data sent through each tree may be optimized by linear programming (*Balanced Multicasting*[^†^])

Problems of FPFR

- In FPFR, slow nodes degrade receiving bandwidth to other nodes.
- For tree topologies, FPFR only outputs one depth-first pipeline, which cannot utilize the potential network performance.
Our Algorithm

- Modify FPFR algorithm
  - Create both **spanning** trees and **partial trees**

- **Stable** for tree topologies whose links have the same bandwidth in both directions
Tree Constructions

- Iteratively construct trees
  - Create a tree $T_n$ by tracing every destination
  - Set the throughput $V_n$ to the bottleneck in $T_n$
  - Subtract $V_n$ from the remaining capacities

T1: First Tree (Spanning)
- Throughput of $T_1$
- $V_1$

T2: Second Tree (Partial Tree)
- $V_2$

T3: Third Tree (Partial Tree)
- $V_3$
Send data proportional to the tree throughput \( V_n \)

Example:

- Stage 1: use T1, T2 and T3
- Stage 2: use T1 and T2 to send data previously sent by T3
- Stage 3: use T1 to send data previously sent by T2
Properties of Our Algorithm

1. Our algorithm is **Stable** for tree topologies (whose links have the same capacities in both directions)
   - Every node receives **maximum** bandwidth

2. For any topology, it achieves **greater aggregate bandwidth** than the baseline algorithm (FPFR)
   - Fully utilize link capacity by using **partial trees**

3. It has small calculation cost to create a broadcast plan
Introduction

Problem Settings

Related Work

Proposed Algorithm

Evaluation

Conclusion
(1) Simulations

- Simulated 5 broadcast algorithms using a real topology
- Compared the aggregate bandwidth of each method
  - Many bandwidth distributions
  - Broadcast to 10, 50, and 100 nodes
  - 10 kinds of conditions (src, dest)
Compared Algorithms

Flat Tree

Random

Dijkstra

Depth-First (FPFR)

... and OURS
**Result of Simulations**

- Mixed two kinds of Links (100 and 1000)
  - Vertical axis: speedup from *FlatTree*
  - 40 times more than *random*,
  - 3 times more than *depth-first* (FPFR) with 100 nodes

<table>
<thead>
<tr>
<th>Number of Destinations</th>
<th>Ours</th>
<th>DepthFirst</th>
<th>Dijkstra</th>
<th>Random (best)</th>
<th>Random</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>50</td>
<td>45</td>
<td></td>
<td>42</td>
<td></td>
<td></td>
</tr>
<tr>
<td>100</td>
<td>88</td>
<td></td>
<td>77</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

![Diagram](image)
Tested 8 bandwidth distributions

- Uniform distribution (500-1000)
- Uniform distribution (100-1000)
- Mixed 100 and 1000 links
- Uniform distribution (500-100) between switches

(for each distribution, tested two conditions that bandwidth of both directions are the same and different)

Our method achieved the largest bandwidth in 7/8 cases

- Large improvement especially in large bandwidth variance
- In a uniform distribution (100-1000) and link bandwidth in two directions are different, Dijkstra achieved 2% more aggregate bandwidth
(2) Real Machine Experiment

- Performed broadcasts in 4 clusters
  - Number of destinations: 10, 47 and 105 nodes
  - Bandwidths of each link: (10M - 1Gbps)

- Compared the aggregate bandwidth in 4 algorithms
  1. Our algorithm
  2. Depth-first (FPFR)
  3. Dijkstra
  4. Random (Best among 100 trials)
Also, we calculated the theoretical maximum aggregate bandwidth:

The total of the receiving bandwidth in a case of separate direct transfer from the source to each destination.
For 105 nodes broadcast, **2.5 times** more bandwidth than the baseline algorithm *DepthFirst (FPFR)*

However, our algorithm stayed 50-70% the aggregate bandwidth compared to the theoretical maximum

- Computational nodes cannot fully utilize up/down network

**Evaluation of Aggregate Bandwidth**

<table>
<thead>
<tr>
<th>(Gbps)</th>
<th>Ours</th>
<th>DepthFirst</th>
<th>Dijkstra</th>
<th>Random</th>
<th>Simulated</th>
</tr>
</thead>
<tbody>
<tr>
<td>(a) Broadcast (10 Nodes)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(Gbps)</td>
<td>Ours</td>
<td>DepthFirst</td>
<td>Dijkstra</td>
<td>Random</td>
<td>Simulated</td>
</tr>
<tr>
<td>(b) Broadcast (47 Nodes)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(Gbps)</td>
<td>Ours</td>
<td>DepthFirst</td>
<td>Dijkstra</td>
<td>Random</td>
<td>Simulated</td>
</tr>
<tr>
<td>(c) Broadcast (105 Nodes)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Evaluation of Stability

- Compared aggregate bandwidth of 9 nodes before/after adding one slow node
  - Unlike DepthFirst(FPFR), existing nodes do not suffer from adding a slow node in our algorithm
  - Achieved 1.6 times bandwidth than Dijkstra

![Diagram showing network changes and bandwidth comparison](image)
Introduction
Problem Settings
Related Work
Our Algorithm
Evaluation
Conclusion
Introduction the notion of Stable Broadcast
- Slow nodes never degrade receiving bandwidth of fast nodes

Proposed a stable broadcast algorithm for tree topologies
- Theoretically proved
- 2.5 times the aggregate bandwidth in real machine experiments
- Confirmed speedup in simulations with many different conditions
Future Work

- Algorithm that maximizes aggregate bandwidth in general graph topologies

- Algorithm that changes relay schedule by detecting bandwidth fluctuations
Algorithm that maximizes aggregate bandwidth in general graph topologies

Algorithm that changes relay schedule by detecting bandwidth fluctuations

Future work