Broadcast Trees for Heterogeneous Platforms

Olivier Beaumont, Loris Marchal and Yves Robert

Laboratoire de l'Informatique du Parallélisme
École Normale Supérieure de Lyon, France

Loris.Marchal@ens-lyon.fr
http://graal.ens-lyon.fr/~lmarchal

January 2005

Outline

Outline

Outline

Outline

Outline

Outline

Outline

Broadcasting data

- Key collective communication operation
- Start: one processor has the data
- End: all processors own a copy
- Vast literature about broadcast, `MPI_Bcast`
- Standard approach: use a spanning tree
- Finding the best spanning tree: NP-Complete problem (even in the telephone model)

Broadcasting data

- Key collective communication operation
- Start: one processor has the data
- End: all processors own a copy
- Vast literature about broadcast, `MPI_Bcast`
- Standard approach: use a spanning tree
- Finding the best spanning tree: NP-Complete problem (even in the telephone model)

Broadcasting data

- Key collective communication operation
- Start: one processor has the data
- End: all processors own a copy
- Vast literature about broadcast, `MPI_Bcast`
- Standard approach: use a spanning tree
- Finding the best spanning tree: NP-Complete problem (even in the telephone model)

Broadcasting data

- Key collective communication operation
- Start: one processor has the data
- End: all processors own a copy
- Vast literature about broadcast, `MPI_Bcast`
- Standard approach: use a spanning tree
- Finding the best spanning tree: NP-Complete problem (even in the telephone model)

Broadcasting data

- Key collective communication operation
- Start: one processor has the data
- End: all processors own a copy
- Vast literature about broadcast, `MPI_Bcast`
- Standard approach: use a spanning tree
- Finding the best spanning tree: NP-Complete problem (even in the telephone model)

Different broadcast problems

Broadcast large messages ⇒ pipelining strategies

- split the messages into slices (application level)
- route them concurrently, possibly using different spanning trees
- throughput optimization (relaxation of makespan minimization)

STA Singe Tree, Atomic message
heuristics to minimize makespan: FNF...

STP Single Tree, Pipelined series of messages

MTP Multiple Tree, Pipelined series of messages

- polynomial algorithm to find optimal solution (LP formulation)
- hard to implement ⇒ concentrate on STP

Different broadcast problems

Broadcast large messages ⇒ pipelining strategies

- split the messages into slices (application level)
- route them concurrently, possibly using different spanning trees
- throughput optimization (relaxation of makespan minimization)

STA Singe Tree, Atomic message
heuristics to minimize makespan: FNF...

STP Single Tree, Pipelined series of messages

MTP Multiple Tree, Pipelined series of messages

- polynomial algorithm to find optimal solution (LP formulation)
- hard to implement ⇒ concentrate on STP

Different broadcast problems

Broadcast large messages ⇒ pipelining strategies

- split the messages into slices (application level)
- route them concurrently, possibly using different spanning trees
- throughput optimization (relaxation of makespan minimization)

STA Singe Tree, Atomic message
heuristics to minimize makespan: FNF...

STP Single Tree, Pipelined series of messages

MTP Multiple Tree, Pipelined series of messages

- polynomial algorithm to find optimal solution (LP formulation)
- hard to implement ⇒ concentrate on STP

Different broadcast problems

Broadcast large messages ⇒ pipelining strategies

- split the messages into slices (application level)
- route them concurrently, possibly using different spanning trees
- throughput optimization (relaxation of makespan minimization)

STA Singe Tree, Atomic message
heuristics to minimize makespan: FNF...

STP Single Tree, Pipelined series of messages

MTP Multiple Tree, Pipelined series of messages

- polynomial algorithm to find optimal solution (LP formulation)
- hard to implement ⇒ concentrate on STP

Different broadcast problems

Broadcast large messages ⇒ pipelining strategies

- split the messages into slices (application level)
- route them concurrently, possibly using different spanning trees
- throughput optimization (relaxation of makespan minimization)

STA Singe Tree, Atomic message
heuristics to minimize makespan: FNF...

STP Single Tree, Pipelined series of messages

MTP Multiple Tree, Pipelined series of messages

- polynomial algorithm to find optimal solution (LP formulation)
- hard to implement ⇒ concentrate on STP

Different broadcast problems

Broadcast large messages ⇒ pipelining strategies

- split the messages into slices (application level)
- route them concurrently, possibly using different spanning trees
- throughput optimization (relaxation of makespan minimization)

STA Singe Tree, Atomic message
heuristics to minimize makespan: FNF...

STP Single Tree, Pipelined series of messages

MTP Multiple Tree, Pipelined series of messages

- polynomial algorithm to find optimal solution (LP formulation)
- hard to implement ⇒ concentrate on STP

Different broadcast problems

Broadcast large messages ⇒ pipelining strategies

- split the messages into slices (application level)
- route them concurrently, possibly using different spanning trees
- throughput optimization (relaxation of makespan minimization)

STA Singe Tree, Atomic message
heuristics to minimize makespan: FNF...

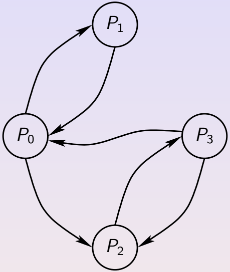
STP Single Tree, Pipelined series of messages

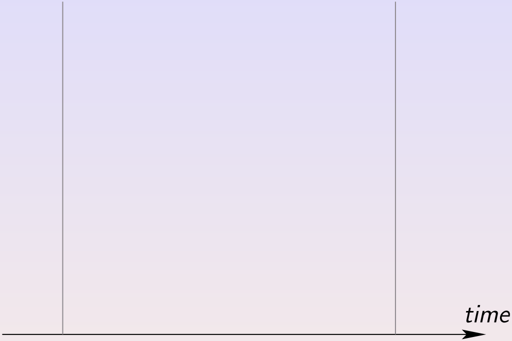
MTP Multiple Tree, Pipelined series of messages
- polynomial algorithm to find optimal solution (LP formulation)
- hard to implement ⇒ concentrate on STP

Outline

Models

Network = directed graph $\mathcal{P} = (V, E)$

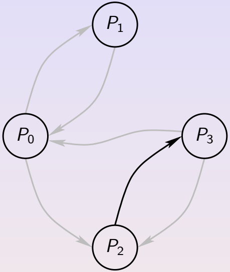


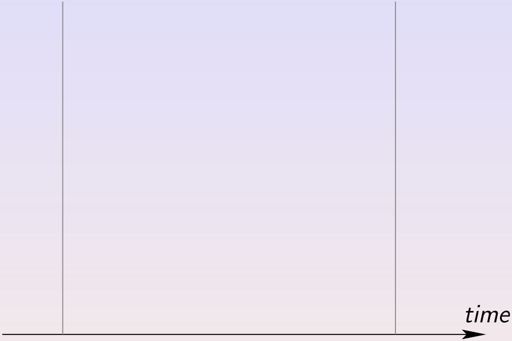


- General case: affine model (includes latencies)
- Common variant: sending and receiving processors busy during whole transfer

Models

Network = directed graph $\mathcal{P} = (V, E)$

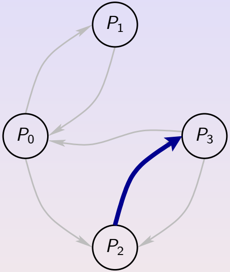


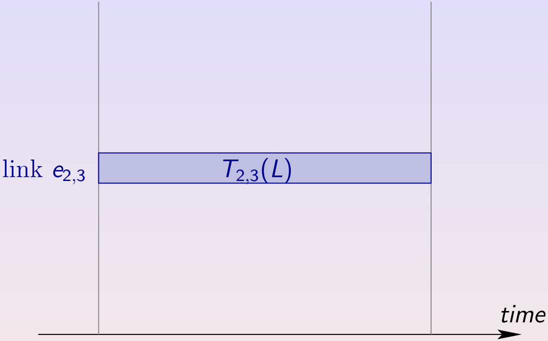


- General case: affine model (includes latencies)
- Common variant: sending and receiving processors busy during whole transfer

Models

Network $=$ directed graph $\mathcal{P} = (V, E)$

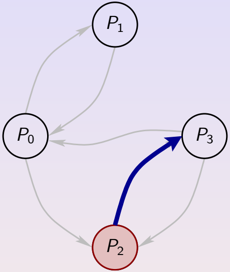


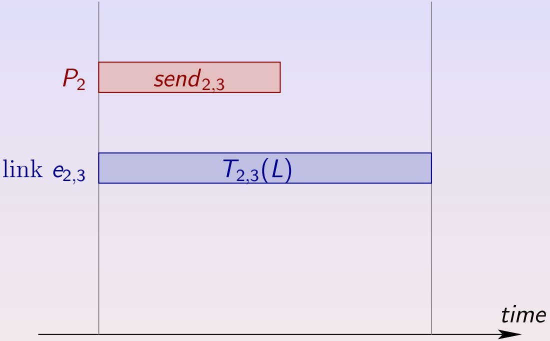


- General case: affine model (includes latencies)
- Common variant: sending and receiving processors busy during whole transfer

Models

Network = directed graph $\mathcal{P} = (V, E)$

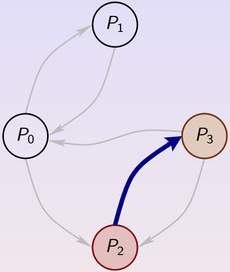


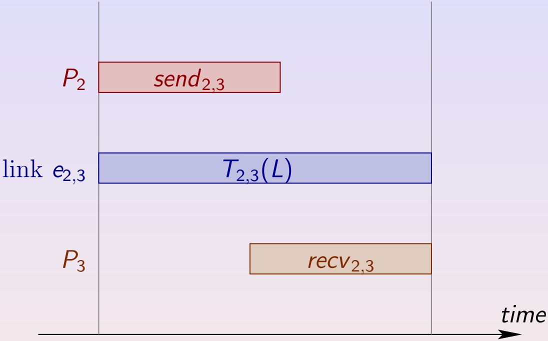


- General case: affine model (includes latencies)
- Common variant: sending and receiving processors busy during whole transfer

Models

Network = directed graph $\mathcal{P} = (V, E)$

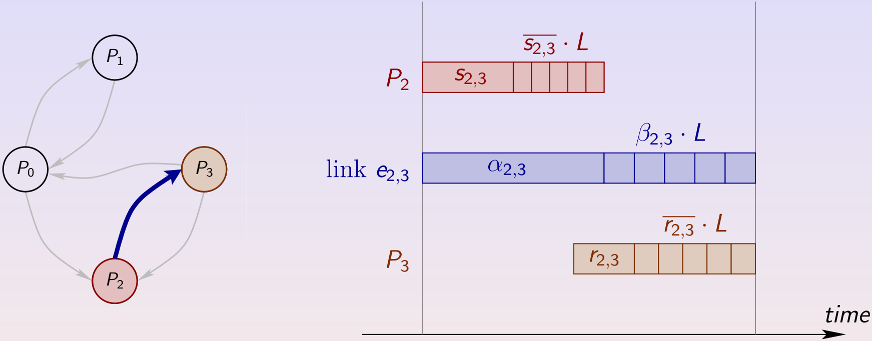




- General case: affine model (includes latencies)
- Common variant: sending and receiving processors busy during whole transfer

Models

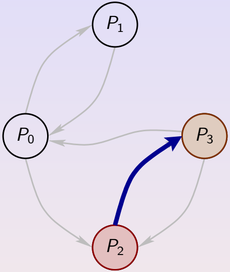
Network = directed graph $\mathcal{P} = (V, E)$

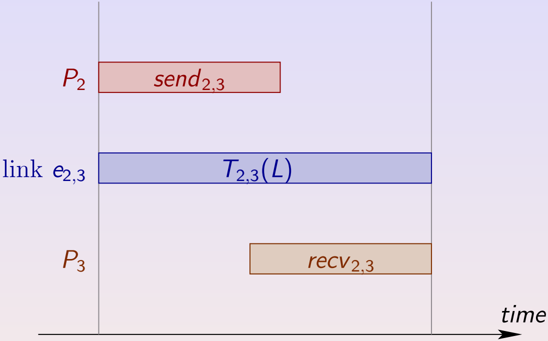


- General case: affine model (includes latencies)
- Common variant: sending and receiving processors busy during whole transfer

Models

Network = directed graph $\mathcal{P} = (V, E)$

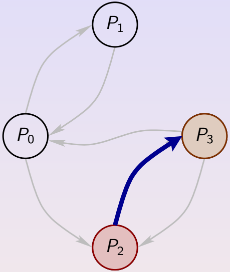


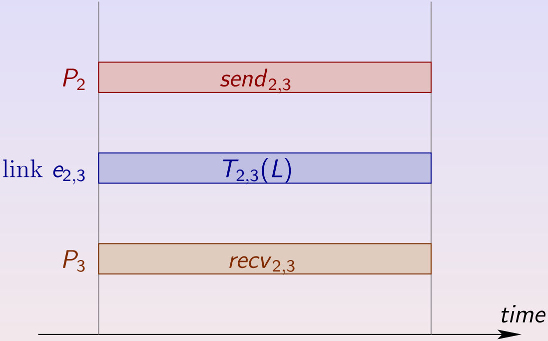


- General case: affine model (includes latencies)
- Common variant: sending and receiving processors busy during whole transfer

Models

Network = directed graph $\mathcal{P} = (V, E)$

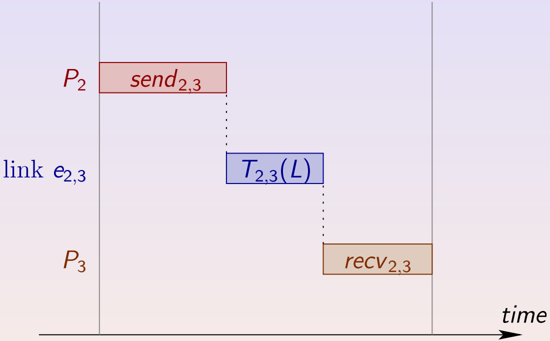




- General case: affine model (includes latencies)
- Common variant: sending and receiving processors busy during whole transfer

Multi-port

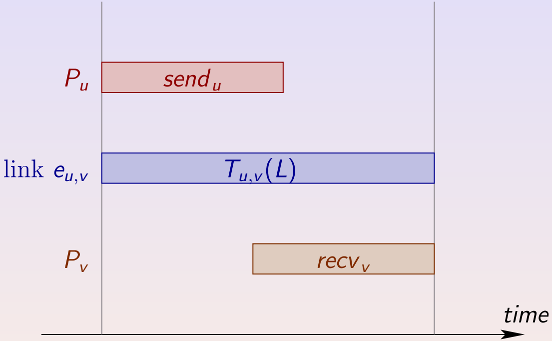
- Banikazemi et al.
  no overlap between link and processor occupation:



$\Rightarrow$ methodology to instantiate parameters

Multi-port

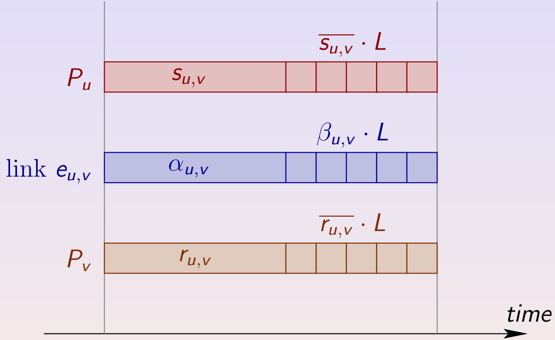
- Bar-Noy et al.
  occupation time of sender P_u independent of target P_v



not *fully* multi-port model, but allows for starting a new transfer from P_u without waiting for previous one to finish

One-port

- Bhat et al.
  same parameters for sender P_u, link $e_{u,v}$ and receiver P_v

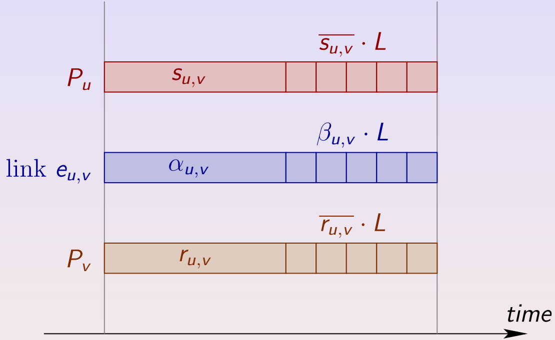


Two flavors:

- bidirectional: simultaneous send *and* receive transfers allowed
- unidirectional: only one send or receive transfer at a given time-step

One-port

- Bhat et al.
  same parameters for sender P_u, link $e_{u,v}$ and receiver P_v

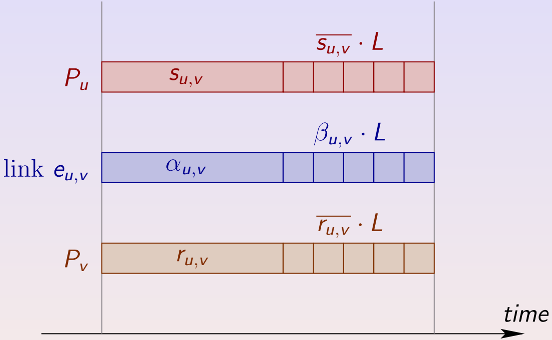


Two flavors:

- bidirectional: simultaneous send *and* receive transfers allowed
- unidirectional: only one send or receive transfer at a given time-step

One-port

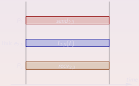
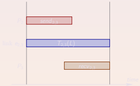
- Bhat et al.
  same parameters for sender P_u, link $e_{u,v}$ and receiver P_v



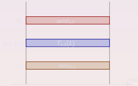
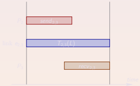
Two flavors:

- bidirectional: simultaneous send *and* receive transfers allowed
- unidirectional: only one send or receive transfer at a given time-step

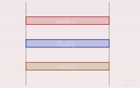
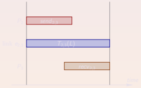
Framework

- Platform graph $\mathcal{P} = (V, E)$
- Source processor P_{source}
- Goal: broadcast a series of messages to all other nodes
- Transfers of successive messages are pipelined
- Send messages along a spanning tree
- Find a spanning tree with good throughput
  (neglect initialization and clean-up phases)
- Bidirectional one-port model: 
- Multi-port model: 

Framework

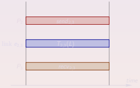
- Platform graph $\mathcal{P} = (V, E)$
- Source processor P_{source}
- Goal: broadcast a series of messages to all other nodes
- Transfers of successive messages are pipelined
- Send messages along a spanning tree
- Find a spanning tree with good throughput
  (neglect initialization and clean-up phases)
- Bidirectional one-port model: 
- Multi-port model: 

Framework

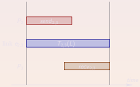
- Platform graph $\mathcal{P} = (V, E)$
- Source processor P_{source}
- Goal: broadcast a series of messages to all other nodes
- Transfers of successive messages are pipelined
- Send messages along a spanning tree
- Find a spanning tree with good throughput
  (neglect initialization and clean-up phases)
- Bidirectional one-port model: 
- Multi-port model: 

Framework

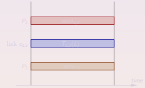
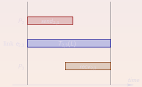
- Platform graph $\mathcal{P} = (V, E)$
- Source processor P_{source}
- Goal: broadcast a series of messages to all other nodes
- Transfers of successive messages are pipelined
- Send messages along a spanning tree
- Find a spanning tree with good throughput
  (neglect initialization and clean-up phases)
- Bidirectional one-port model:



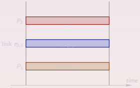
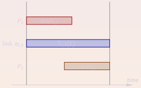
- Multi-port model:



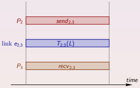
Framework

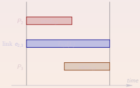
- Platform graph $\mathcal{P} = (V, E)$
- Source processor P_{source}
- Goal: broadcast a series of messages to all other nodes
- Transfers of successive messages are pipelined
- Send messages along a spanning tree
- Find a spanning tree with good throughput (neglect initialization and clean-up phases)
- Bidirectional one-port model: 
- Multi-port model: 

Framework

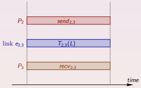
- Platform graph $\mathcal{P} = (V, E)$
- Source processor P_{source}
- Goal: broadcast a series of messages to all other nodes
- Transfers of successive messages are pipelined
- Send messages along a spanning tree
- Find a spanning tree with good throughput
  (neglect initialization and clean-up phases)
- Bidirectional one-port model: 
- Multi-port model: 

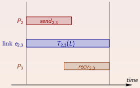
Framework

- Platform graph $\mathcal{P} = (V, E)$
- Source processor P_{source}
- Goal: broadcast a series of messages to all other nodes
- Transfers of successive messages are pipelined
- Send messages along a spanning tree
- Find a spanning tree with good throughput
  (neglect initialization and clean-up phases)
- Bidirectional one-port model: 

- Multi-port model: 

Framework

- Platform graph $\mathcal{P} = (V, E)$
- Source processor P_{source}
- Goal: broadcast a series of messages to all other nodes
- Transfers of successive messages are pipelined
- Send messages along a spanning tree
- Find a spanning tree with good throughput (neglect initialization and clean-up phases)
- Bidirectional one-port model: 

- Multi-port model: 

Outline

One port-model

- Processors involved in one (sending or receiving) communication
- Duration of a transfer = f (link $e_{u,v}$)

$$send_{u,v}(L) = recv_{u,v}(L) = T_{u,v}(L) = T_{u,v}$$

One port-model

- Processors involved in one (sending or receiving) communication
- Duration of a transfer = f (link $e_{u,v}$)

$$send_{u,v}(L) = recv_{u,v}(L) = T_{u,v}(L) = T_{u,v}$$

Simple Platform Pruning

- Idea: delete edges of maximum weight, until we have a tree
- Algorithm:

SIMPLE-PLATFORM-PRUNING($\mathcal{P}$, P_{source})
  $TreeEdges \leftarrow$ all edges of E
  while $|TreeEdges| > n - 1$ **do**
    $L \leftarrow$ edges of $TreeEdges$ sorted by non-increasing weight $T_{u,v}$
    for each edge $e \in L$ **do**
      if the graph $(V, TreeEdges \setminus \{e\})$ is still connected **then**
        $TreeEdges \leftarrow TreeEdges \setminus \{e\}$
  return $(V, TreeEdges)$

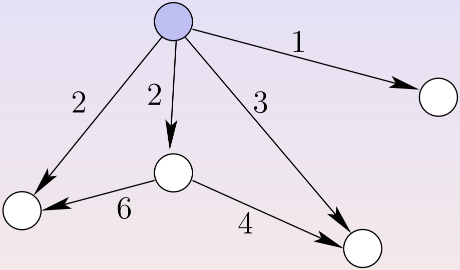
Simple Platform Pruning

- Idea: delete edges of maximum weight, until we have a tree
- Algorithm:

SIMPLE-PLATFORM-PRUNING($\mathcal{P}$, P_{source})
    $TreeEdges \leftarrow$ all edges of E
    while $|TreeEdges| > n - 1$ **do**
        $L \leftarrow$ edges of $TreeEdges$ sorted by non-increasing weight $T_{u,v}$
        for each edge $e \in L$ **do**
            if the graph $(V, TreeEdges \backslash \{e\})$ is still connected **then**
                $TreeEdges \leftarrow TreeEdges \backslash \{e\}$
    return $(V, TreeEdges)$

Simple Platform Pruning

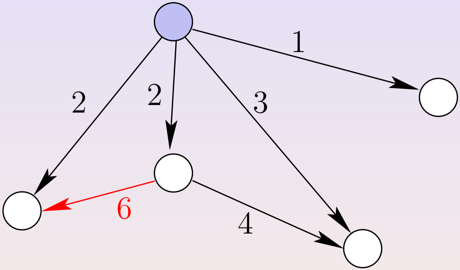
- Example of simple pruning:



Topology, costs of edges $T_{u,v}$

Simple Platform Pruning

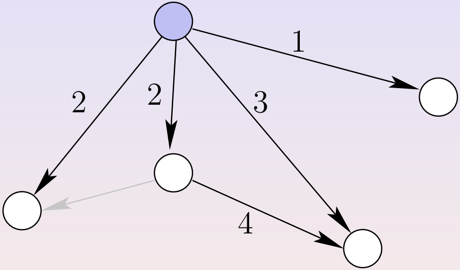
- Example of simple pruning:



Choosing and pruning edge of maximum weight

Simple Platform Pruning

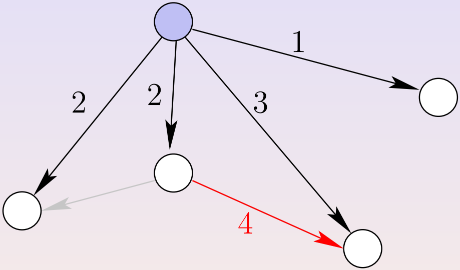
- Example of simple pruning:



Choosing and pruning edge of maximum weight

Simple Platform Pruning

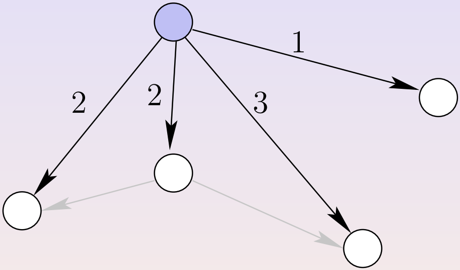
- Example of simple pruning:



Choosing and pruning edge of maximum weight

Simple Platform Pruning

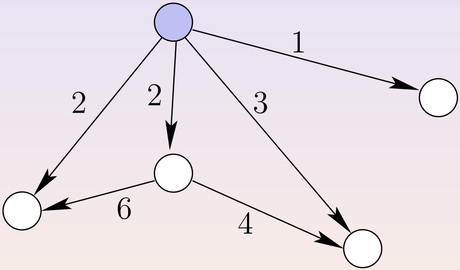
- Example of simple pruning:



Achievable throughput: 1/8

Refined Platform Pruning

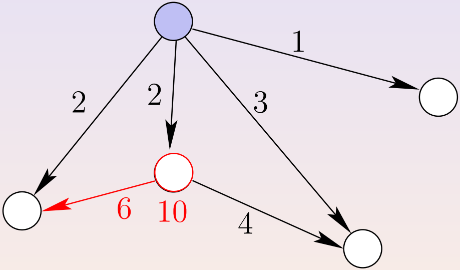
- Idea:
  - at each step, compute the out-degree of each node
  - prune an edge from a node whose out-degree is maximum
- Example:



Topology, costs of edges $T_{u,v}$

Refined Platform Pruning

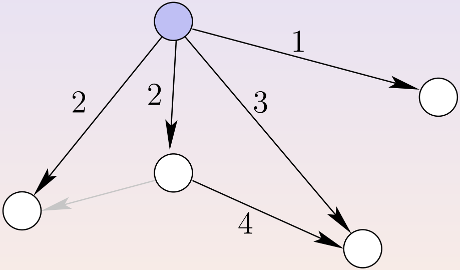
- Idea:
  - at each step, compute the out-degree of each node
  - prune an edge from a node whose out-degree is maximum
- Example:



Choosing maximum out-degree node, then maximum edge

Refined Platform Pruning

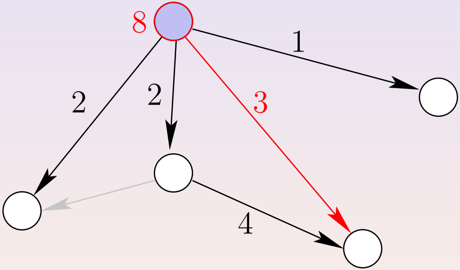
- Idea:
  - at each step, compute the out-degree of each node
  - prune an edge from a node whose out-degree is maximum
- Example:



Choosing maximum out-degree node, then maximum edge

Refined Platform Pruning

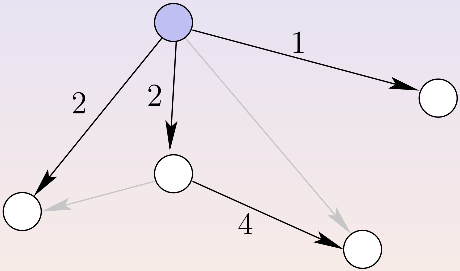
- Idea:
  - at each step, compute the out-degree of each node
  - prune an edge from a node whose out-degree is maximum
- Example:



Choosing maximum out-degree node, then maximum edge

Refined Platform Pruning

- Idea:
  - at each step, compute the out-degree of each node
  - prune an edge from a node whose out-degree is maximum
- Example:



Achievable throughput: 1/5

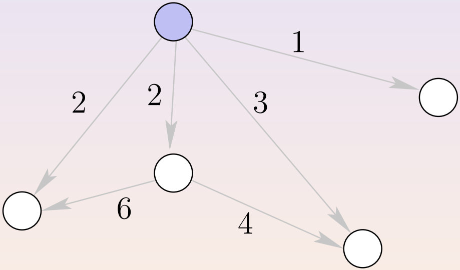
Refined Platform Pruning

REFINED-PLATFORM-PRUNING($\mathcal{P}, P_{source}$)

$TreeEdges \leftarrow$ all edges of E
for each $u \in V$ **do**
    $OutDegree(u) \leftarrow \sum_{v,\ (u,v) \in E} T_{u,v}$
while $|TreeEdges| > n - 1$ **do**
    $SortedNodes \leftarrow$ nodes sorted by non-increasing value of $OutDegree(u)$
    for $u \in SortedNodes$ **do**
        $L \leftarrow$ edges sorted by decreasing weight $T_{u,v}$
        for each edge $e = (u, v) \in L$ **do**
            if the graph $(V, TreeEdges \backslash \{e\})$ is still connected **then**
                $TreeEdges \leftarrow TreeEdges \backslash \{e\}$
                $OutDegree(u) \leftarrow OutDegree(u) - T_{u,v}$
                goto 4
return $(V, TreeEdges)$

Growing a Minimum Weighted Out-Degree Tree

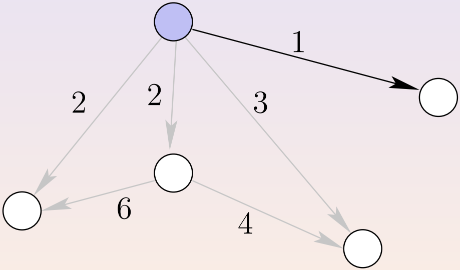
- Idea: grow a tree as in Prim's algorithm
- At each step, choose an edge optimizing metric
- Our metric:
  - minimize the weighted out-degree of each node in the tree
- Example:



Achievable throughput: 1/5

Growing a Minimum Weighted Out-Degree Tree

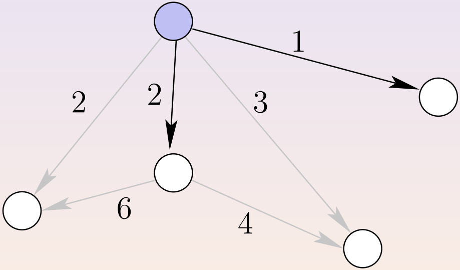
- Idea: grow a tree as in Prim's algorithm
- At each step, choose an edge optimizing metric
- Our metric:
  - minimize the weighted out-degree of each node in the tree
- Example:



Achievable throughput: 1/5

Growing a Minimum Weighted Out-Degree Tree

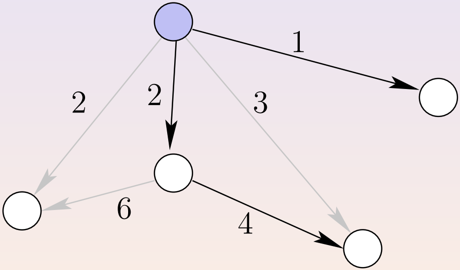
- Idea: grow a tree as in Prim's algorithm
- At each step, choose an edge optimizing metric
- Our metric:
  - minimize the weighted out-degree of each node in the tree
- Example:



Achievable throughput: 1/5

Growing a Minimum Weighted Out-Degree Tree

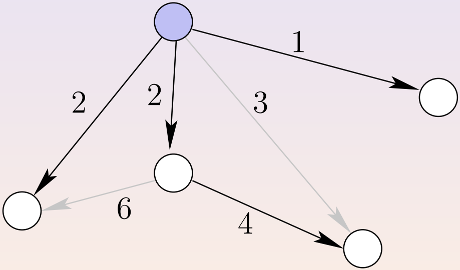
- Idea: grow a tree as in Prim's algorithm
- At each step, choose an edge optimizing metric
- Our metric:
  - minimize the weighted out-degree of each node in the tree
- Example:



Achievable throughput: 1/5

Growing a Minimum Weighted Out-Degree Tree

- Idea: grow a tree as in Prim's algorithm
- At each step, choose an edge optimizing metric
- Our metric:
  - minimize the weighted out-degree of each node in the tree
- Example:



Achievable throughput: $1/5$

Growing a Minimum Weighted Out-Degree Tree

GROWING-MINIMUM-WEIGHTED-OUT-DEGREE-TREE($\mathcal{P}$, P_{source})

  $TreeEdges \leftarrow \emptyset$
  $TreeVertices \leftarrow \{P_{source}\}$
  for each edge $e = (u, v)$ **do**
    $cost(u, v) \leftarrow T_{u,v}$
  while $TreeVertices \neq V$ **do**
    choose the link (u, v) such that $u \in TreeVertices$,
    $v \notin TreeVertices$ and (u, v) has minimum value $cost(u, v)$
    $TreeVertices \leftarrow TreeVertices \cup \{v\}$
    $TreeEdges \leftarrow TreeEdges \cup \{(u, v)\}$
    for each edge $(u, w) \notin TreeEdges$ **do**
      $cost(u, w) \leftarrow cost(u, w) + cost(u, v)$
  return ($TreeVertices$, $TreeEdges$)

Binomial tree heuristic

- For sake of comparison
- Close to MPI_Bcast
- Construct a binomial tree (without topological information)

Binomial tree heuristic

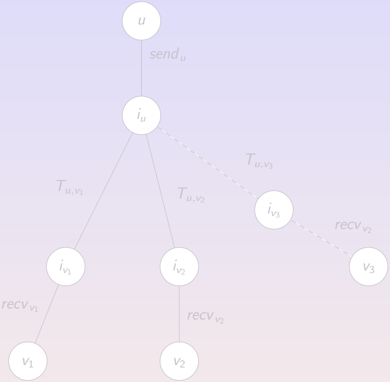
- For sake of comparison
- Close to `MPI_Bcast`
- Construct a binomial tree (without topological information)

Binomial tree heuristic

- For sake of comparison
- Close to `MPI_Bcast`
- Construct a binomial tree (without topological information)

Multi-port

- Adapt the growing-tree heuristic to multi-port model
- Congestion may come from:
  - the number of send operations from P_u,
  - the length of a transfer $P_u \to P_v$
- New computation of out-degree:

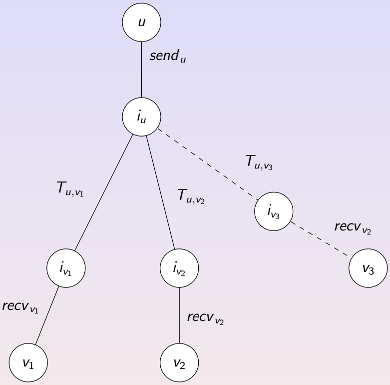


$$T_{period} = \max\left(\delta_{out}(P_u) \times send_u, \max_i(T_{u,v_i})\right)$$

Multi-port

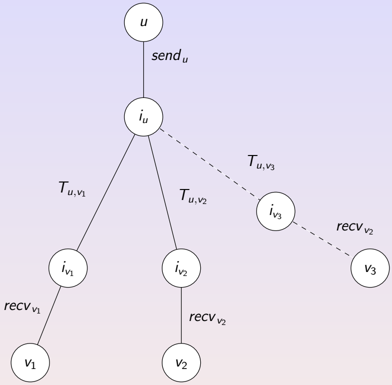
- Adapt the growing-tree heuristic to multi-port model
- Congestion may come from:
  - the number of send operations from P_u,
  - the length of a transfer $P_u \rightarrow P_v$
- New computation of out-degree:

$$T_{period} = \max\left(\delta_{out}(P_u) \times send_u, \max_i(T_{u,v_i})\right)$$



Multi-port

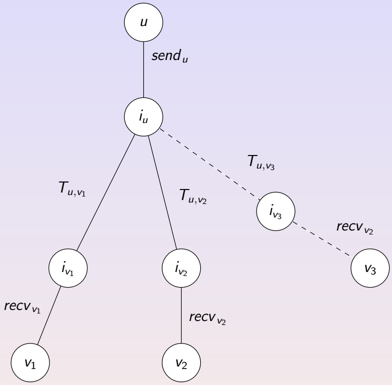
- Adapt the growing-tree heuristic to multi-port model
- Congestion may come from:
  - the number of send operations from P_u,
  - the length of a transfer $P_u \to P_v$
- New computation of out-degree:



$$T_{period} = \max\left(\delta_{out}(P_u) \times send_u, \max_i(T_{u,v_i})\right)$$

Multi-port

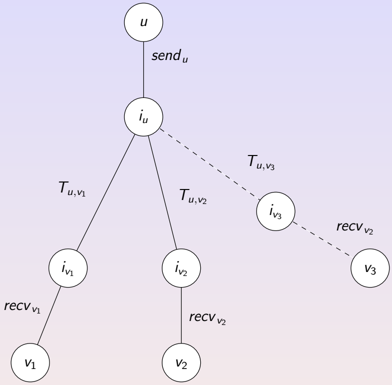
- Adapt the growing-tree heuristic to multi-port model
- Congestion may come from:
  - the number of send operations from P_u,
  - the length of a transfer $P_u \rightarrow P_v$
- New computation of out-degree:



$$T_{period} = \max\left(\delta_{out}(P_u) \times send_u, \max_i(T_{u,v_i})\right)$$

Multi-port

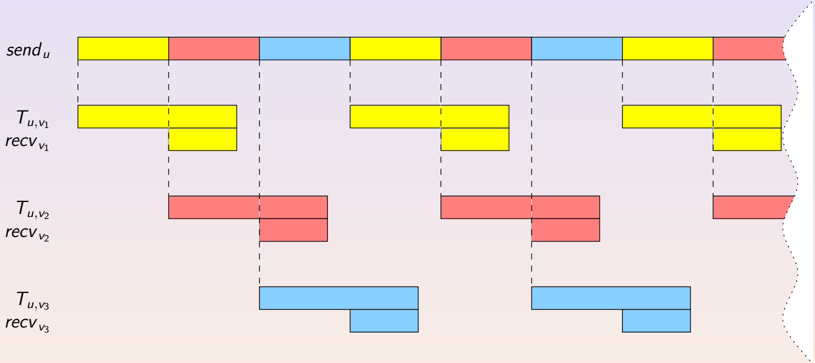
- Adapt the growing-tree heuristic to multi-port model
- Congestion may come from:
  - the number of send operations from P_u,
  - the length of a transfer $P_u \rightarrow P_v$
- New computation of out-degree:



$$T_{period} = \max\left(\delta_{out}(P_u) \times send_u, \max_i(T_{u,v_i})\right)$$

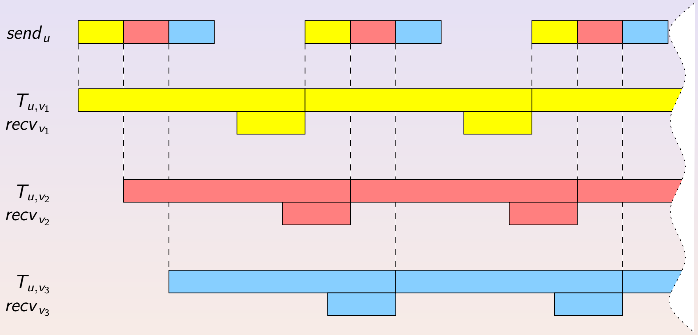
Multi-port

- Case where throughput is bounded by the serialized $send_u$:
- Case where throughput is bounded by the longest link occupation $T_{u,v}$:



Multi-port

- Case where throughput is bounded by the serialized $send_u$:
- Case where throughput is bounded by the longest link occupation $T_{u,v}$:



Outline

LP formulation

- Solving the MTP problem with LP formulation:
  - variables: average number of messages going through each link
  - constraints: one-port model constraints, link occupation
- Solution of LP ⇒ network utilization to reach best throughput
- Complicated algorithm to reconstruct optimal set of trees for MTP, not needed here
- Use results output by LP, optimal solution $\mathcal{S}_{\text{opt}}$:
  - TP = optimal throughput
  - $n_{u,v}$ = number of messages through edge $e_{u,v}$ in one time-unit in $\mathcal{S}_{\text{opt}}$

LP formulation

- Solving the MTP problem with LP formulation:
  - variables: average number of messages going through each link
  - constraints: one-port model constraints, link occupation
- Solution of LP ⇒ network utilization to reach best throughput
- Complicated algorithm to reconstruct optimal set of trees for MTP, not needed here
- Use results output by LP, optimal solution $\mathcal{S}_{\text{opt}}$:
  - TP = optimal throughput
  - $n_{u,v}$ = number of messages through edge $e_{u,v}$ in one time-unit in $\mathcal{S}_{\text{opt}}$

LP formulation

- Solving the MTP problem with LP formulation:
  - variables: average number of messages going through each link
  - constraints: one-port model constraints, link occupation
- Solution of LP $\Rightarrow$ network utilization to reach best throughput
- Complicated algorithm to reconstruct optimal set of trees for MTP, not needed here
- Use results output by LP, optimal solution $\mathcal{S}_{\text{opt}}$:
  - TP = optimal throughput
  - $n_{u,v}$ = number of messages through edge $e_{u,v}$ in one time-unit in $\mathcal{S}_{\text{opt}}$

LP formulation

- Solving the MTP problem with LP formulation:
  - variables: average number of messages going through each link
  - constraints: one-port model constraints, link occupation
- Solution of LP $\Rightarrow$ network utilization to reach best throughput
- Complicated algorithm to reconstruct optimal set of trees for MTP, not needed here
- Use results output by LP, optimal solution $\mathcal{S}_{\text{opt}}$:
  - TP = optimal throughput
  - $n_{u,v}$ = number of messages through edge $e_{u,v}$ in one time-unit in $\mathcal{S}_{\text{opt}}$

LP-based heuristics

- Communication graph pruning:
  - similar to the previous pruning heuristic
  - based on the communication graph, labeled with $n_{u,v}$ values
  - prune edges carrying the fewest messages in $\mathcal{S}_{opt}$
- Growing a spanning tree over the communication graph
  - start from the communication graph of $\mathcal{S}_{opt}$
  - grow a tree, selecting edges with maximal number of messages

LP-based heuristics

- Communication graph pruning:
  - similar to the previous pruning heuristic
  - based on the communication graph, labeled with $n_{u,v}$ values
  - prune edges carrying the fewest messages in $\mathcal{S}_{\text{opt}}$
- Growing a spanning tree over the communication graph
  - start from the communication graph of $\mathcal{S}_{\text{opt}}$
  - grow a tree, selecting edges with maximal number of messages

Outline

Platform

Simulations using both the one-port and multi-port models

1. random generation of platforms, with parameters:

number of nodes	:	10, 20,. . . , 50
density	:	0.04, 0.08,. . . , 0.20
$T_{u,v}$	:	Gaussian distribution
	:	(mean=100MB/s, deviation=20MB/s)
$send_{u,v}$	:	$0.80 \cdot \min_{w,(u,w)\in E} \{T_{u,w}\}$

(for each set of parameters, 10 different configurations generated)

2. realistic platforms generated by Tiers:
   - 100 platforms with 30 nodes
   - 100 platforms with 65 nodes
   - density between 0.05 and 0.15

Platform

Simulations using both the one-port and multi-port models

1. random generation of platforms, with parameters:

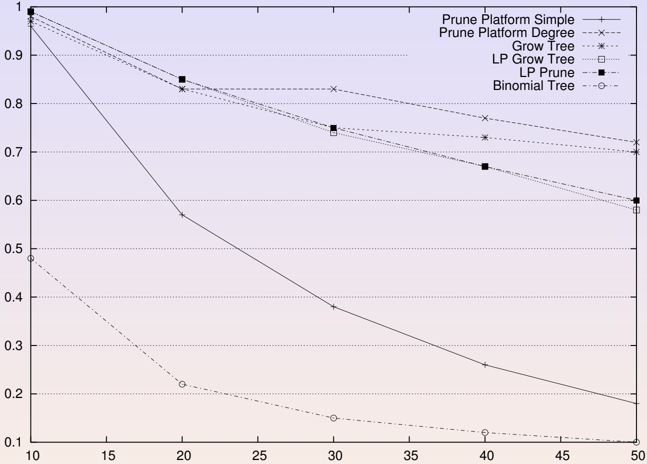
number of nodes :	10, 20,. . . , 50
density :	0.04, 0.08,. . . , 0.20
$T_{u,v}$:	Gaussian distribution
:	(mean=100MB/s, deviation=20MB/s)
$send_{u,v}$:	$0.80 \cdot \min_{w,(u,w)\in E} \{T_{u,w}\}$

   (for each set of parameters, 10 different configurations generated)

2. realistic platforms generated by Tiers:
   - 100 platforms with 30 nodes
   - 100 platforms with 65 nodes
   - density between 0.05 and 0.15

Results, one-port, random platforms

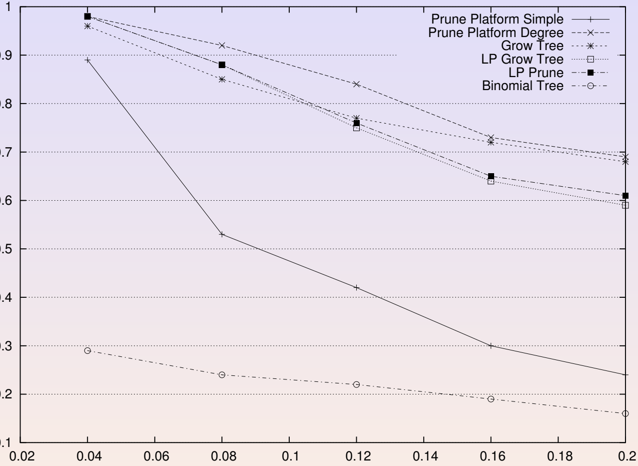
- Performance versus number of nodes



Y axis: relative average performance compared to the optimal solution for MTP

Results, one-port, random platforms

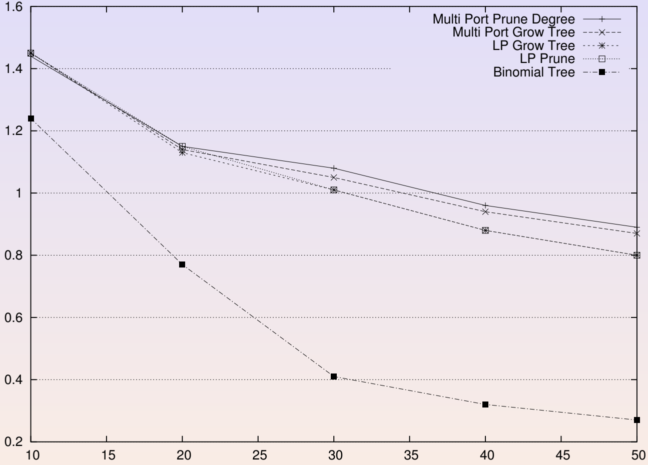
- Performance versus density



Y axis: relative average performance compared to the optimal solution for MTP

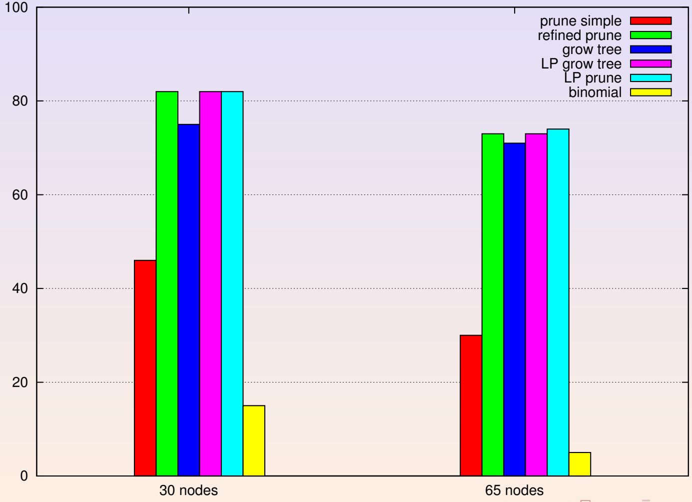
Results, multi-port, random platforms

- Performance versus number of nodes



Results, one-port, realistic platforms

- Performance of the one-port heuristics on two types of platforms generated by TIERS



Analysis

- For the one-port model:
  - small platforms: results close to the optimal
  - large platforms: "advanced" heuristics within 60% of the optimal
  - simple pruning heuristic: not scalable
  - binomial heuristic: very poor results
- Under multi-port assumption:
  - binomial heuristic performs slightly better
  - adapted heuristic (Growing-Tree): much better results
  - LP-based heuristics perform well

Analysis

- For the one-port model:
  - small platforms: results close to the optimal
  - large platforms: "advanced" heuristics within 60% of the optimal
  - simple pruning heuristic: not scalable
  - binomial heuristic: very poor results
- Under multi-port assumption:
  - binomial heuristic performs slightly better
  - adapted heuristic (Growing-Tree): much better results
  - LP-based heuristics perform well

Outline

Conclusion

- Designing efficient algorithms to broadcast data
- Use pipelining techniques, focus on steady-state
- Using multiple trees (MTP): polynomial algorithm, but difficult to enforce in practice
- Using a single tree (STP): NP-Complete
- Design heuristics for STP, possibly using MTP linear program
- Avoid binomial approach (MPI)