Laboratoire de l'Informatique du Parallélisme

École Normale Supérieure de Lyon
Unité Mixte de Recherche CNRS-INRIA-ENS LYON n° 5668

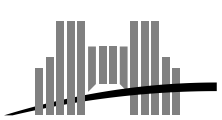
CENTRE NATIONAL DE LA RECHERCHE SCIENTIFIQUE

Optimizing the steady-state throughput of Broadcasts on heterogeneous platforms

Arnaud Legrand,
Olivier Beaumont,
Loris Marchal ,
Yves Robert

June 2003

Research Report N° 2003-34



École Normale Supérieure de Lyon
46 Allée d'Italie, 69364 Lyon Cedex 07, France
Téléphone : +33(0)4.72.72.80.37
Télécopieur : +33(0)4.72.72.80.80
Adresse électronique : `lip@ens-lyon.fr`

INRIA

Optimizing the steady-state throughput of Broadcasts on heterogeneous platforms

Arnaud Legrand, Olivier Beaumont, Loris Marchal , Yves Robert

June 2003

Abstract

In this paper, we consider the communications involved by the execution of a complex application, deployed on a heterogeneous "grid" platform. Such applications extensively use macro-communication schemes, for example to broadcast data items. Rather than aiming at minimizing the execution time of a single broadcast, we focus on the steady-state operation. We assume that there is a large number of messages to be broadcast in pipeline fashion, and we aim at maximizing the throughput, i.e. the (rational) number of messages which can be broadcast every time-step. We target heterogeneous platforms, modeled by a graph where resources have different communication and computation speeds. Achieving the best throughput may well require that the target platform is used in totality: we show that neither spanning trees nor DAGs are as powerful as general graphs.

We show how to compute the best throughput using linear programming, and how to exhibit a periodic schedule, first when restricting to a DAG, and then when using a general graph. The polynomial compactness of the description comes from the decomposition of the schedule into several broadcast trees that are used concurrently to reach the best throughput. It is important to point out that a concrete scheduling algorithm based upon the steady-state operation is asymptotically optimal, in the class of all possible schedules (not only periodic solutions).

Keywords: Scheduling, steady-state, broadcast, heterogeneous platforms.

Résumé

Nous nous intéressons ici aux communications qui ont lieu lors de l'exécution d'une application complexe distribuée sur un environnement hétérogène de type *"grille de calcul"*. De telles applications font un usage intensif de la diffusion de données à travers le réseau d'interconnexion. Nous cherchons à optimiser le débit d'une telle diffusion en régime permanent, en supposant qu'un grand nombre de messages doivent être diffusés successivement, comme c'est le cas pour le parallélisme de données. La plateforme hétérogène que nous visons est modélisée par un graphe où les différentes ressources (calcul ou communication) ont des vitesses différentes. Nous étudions la diffusion utilisant des sous-réseaux avec des topologies restreintes (sur des arbres ou des graphes acycliques dirigés), et montrons que celles-ci sont moins puissantes que des graphes généraux. Nous montrons comment calculer le débit optimal d'une diffusion et comment construire un ordonnancement périodique qui réalise ce débit.

Mots-clés: Ordonnancement, régime permanent, diffusion, plateforme hétérogène.

1 Introduction

Broadcasting in computer networks is the focus of a vast literature. The one-to-all broadcast, or single-node broadcast [14], is the most primary collective communication pattern: initially, only the source processor has the data that needs to be broadcast; at the end, there is a copy of the original data residing at each processor.

Parallel algorithms often require to send identical data to all other processors, in order to disseminate global information (typically, input data such as the problem size or application parameters). Numerous broadcast algorithms have been designed for parallel machines such as meshes, hypercubes, and variants (see among others [12, 27, 25, 13, 26]). The *one-to-all* MPI routine [23] is widely used, and particular case has been given to its efficient implementation on a large variety of platforms [11]. There are three main variants considered in the literature:

Atomic broadcast: the source message is atomic, i.e. cannot be split into packets. A single message is sent by the source processor, and forwarded across the network.

Pipelined broadcast: the source message can be split into an arbitrary number of packets, which may be routed in a pipelined fashion, possibly using different paths.

Series of broadcasts: the same source processor sends a series of atomic one-to-all broadcasts. The processing of these broadcasts can be pipelined.

For the first two problems, the goal is to minimize the total execution time (or makespan). For the third problem, the objective function is rather to optimize the throughput of the steady-state operation, i.e. the average amount of data broadcast per time-unit.

In the case of the *atomic broadcast*, there is no reason why a processor (distinct from the source) would receive the message twice. Therefore, the atomic broadcast is frequently implemented using a spanning tree. In the case of the *pipelined broadcast*, things get more complex: the idea is to use several edge-disjoint spanning trees to route simultaneously several fractions of the total message. Along each spanning tree, the message fraction is divided into packets, which are sent in a pipeline fashion, so as to minimize start-up idle times. See [27] for an illustration with two-dimensional meshes.

The *series of broadcasts* problems has been considered by Moore and Quinn [20], and by Desprez et al. [24], but with a different perspective: they consider that *distinct* processor sources successively broadcast one message, and their goal is to load-balance this series of communications. Here, we assume that the *same* source processor initiates all the broadcasts: this is closer to a master-slave paradigm where the master disseminates the information to the slaves in a pipelined fashion, for instance the data needed to solve a collection of (independent) problem instances.

The *series of broadcasts* resembles the *pipelined broadcast* problem in that we can solve the latter using an algorithm for the former: this amounts to fix the granularity, i.e. the size of the atomic messages (packets) that will be sent in pipeline. However, an efficient solution to the *pipelined broadcast* problem would require to determine the size of the packets as a function of the total message length.

In this paper, we re-visit the *series of broadcasts* and the *pipelined broadcast* problems in the context of heterogeneous computing platforms. Several authors have recently studied broadcasting with processors communicating with their neighbors along links with different capacities, and/or different start-up costs (see Section 8 on related work), but they mainly

restricted to the *atomic broadcast* problem. Our major result is the design of an optimal algorithm for the *series of broadcasts* problem, and of an asymptotically optimal algorithm for the *pipelined broadcast* problem. Both algorithms rely on tools such as linear programming, network flows and graph theory (Edmond's branching theorem).

The rest of the paper is organized as follows. The next section (Section 2) is devoted to the formal specification of our broadcast problems, including the description of the target heterogenous network, and of the operating mode of the resources (processors and communication links). Section 3 is devoted to comparing topologies for the *series of broadcasts* problem: we work out a toy example, and we compare the best throughput that can be achieved using a tree, a directed acyclic graph (DAG), or the full topology with cycles. In passing, we prove that determining the best broadcast tree is a NP-hard problem. In Section 4, we move to the design of the optimal steady-state algorithm, when the target network is a directed acyclic graph (DAG). Indeed, it turns out that computing the optimal throughput for the *series of broadcast* problem is much easier when restricting to DAGs than when dealing with arbitrary graphs (including cycles). We extend this result to the latter case of an arbitrary network graph in Section 5. The proof is technically involved, and some details may be left at the first reading. We also prove (Section 5.5) that the concrete scheduling algorithm based upon the steady-state operation is asymptotically optimal, in the class of all possible schedules, (not only periodic solutions). Section 6 deals with the *pipelined broadcast* problem: we design an asymptotically optimal algorithm, based upon the previous result and extended via fluid relaxation techniques due to Bertsimas and Gamarnik [3]. We report some experimental data in Section 7. Finally, we state some concluding remarks in Section 9.

2 Framework

The target architectural platform is represented by an edge-weighted directed graph $G = (V, E, c)$, as illustrated in Figure 1. Note that this graph may well include cycles and multiple paths. Let $p = |V|$ be the number of nodes. There is a *source* node P_s, which plays a particular role: it initially holds all the data to be broadcast. All the other nodes P_i, $1 \leqslant i \leqslant p, i \neq s$, are destination nodes which must receive all the data sent by P_s.

There are several scenarios for the operation of the processors, as discussed in Section 8. In this paper, we concentrate on the *one-port model*, where a processor node can simultaneously receive data from one of its neighbor, and send (independent) data to one of its neighbor. At any given time-step, there are at most two communications involving a given processor, one in emission and the other in reception.

Each edge $e_{j,k} : P_j \to P_k$ is labeled by a value $c_{j,k}$ which represents the time needed to communicate one unit-size message from P_j to P_k (start-up costs are dealt with below, for the *pipelined broadcast* problem). The graph is directed, and the time to communicate in the reverse direction, from P_k to P_j, provided that this link exists, is $c_{k,j}$. Note that if there is no communication link between P_j and P_k we let $c_{j,k} = +\infty$, so that $c_{j,k} < +\infty$ means that P_j and P_k are neighbors in the communication graph. We state the communication model more precisely: if P_j sends a unit-size message to P_k at time-step t, then

- P_k cannot initiate another receive operation before time-step $t + c_{j,k}$ (but it can perform a send operation),
- P_j cannot initiate another send operation before time-step $t + c_{j,k}$ (but it can perform

a receive operation).

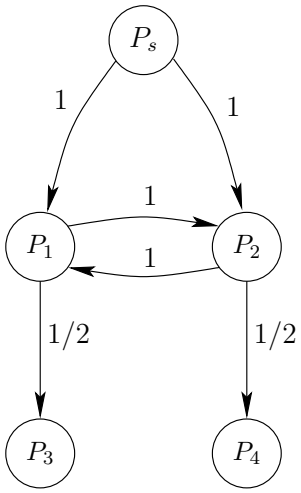


Figure 1: Simple network topology. The value of $c_{j,k}$ is indicated along each edge. The node P_s is the source of the broadcasts.

Series of broadcasts In the *series of broadcasts* problem, the source processor broadcasts a (potentially infinite) sequence of unit-size messages. Start-up costs are included in the values of the link capacities $c_{j,k}$. The optimization problem, which we denote as SERIES(V, E, c), is to maximize the throughput, i.e. the average number of broadcasts initiated per time-unit. We work out a little example in Section 3.1, using the platform represented in Figure 1.

Pipelined broadcast In the *pipelined broadcast* problem, the source processor broadcasts a large message of total size L. The message can be split into an arbitrary number of packets. The time to send a packet of size $n_{j,k}$ from P_j to P_k is $\beta_{j,k} + n_{j,k}c_{j,k}$. We include the start-up costs in the definition of the platform graph, which becomes $G = (V, E, c, \beta)$. The optimization problem, which we denote as PIPE(V, E, c, β), is to minimize the makespan, i.e. to find the number and size of the packets, and a routing scheme for each broadcast packet, so that the total execution time is as small as possible.

3 Comparing topologies for series of broadcasts

We start this section with an example, whose objective is to show the difficulty of the problem. We compare the best throughput that can be achieved using a tree, a directed acyclic graph (DAG), or the full topology with cycles. In passing, we prove that determining the best broadcast tree is a NP-hard problem.

3.1 Working out an example

3.1.1 Optimal solution

Consider the simple example of the network described on Figure 1. The best throughput that can be achieved on this network is 1, i.e. one message is broadcast every time-step after some

initialization phase. On one hand, since the source cannot send more than one message at each time-unit, the best throughput is less than or equal to 1. On the other hand, a feasible schedule for a series of broadcasts realizing this throughput is given in Figure 2. Here are a few comments to help read this figure:

- Messages are tagged by their number. Columns represent time-steps. The schedule is periodic, with period length $T = 2$. Steady-state is reached at time-step $t = 5$.

- At time-step $t = 1$, the source processor P_s sends the first message m_1 to P_1. At time-step $t = 2$, the source processor P_s sends the second message m_2 to P_2. Every odd-numbered step, P_s sends a new message to P_1, and every even-numbered step, P_s sends a new message to P_2

- P_1 is idle at time-steps $t = 1$ and $t = 3$: since it has not yet reached its steady-state, we have indicated fictitious messages (represented as crosses "×"), which it would have received from P_s if the computation had started earlier. At time-step $t = 2$, P_1 forwards the first message m_1 to P_2. Every even-numbered time-step, P_1 forwards to P_2 the message that it has received from P_s during the previous step.

- At step $t = 5$, P_1 forwards two-messages to P_3: message m_1 that it received from P_s at $t = 1$, and message m_2 that it received from P_2 at $t = 3$. Because the link is twice faster ($c_{1,3} = 1/2$), one time-step is enough for sending both messages. From then on, every odd-numbered time-step, P_1 sends two messages to P_3.

- P_2 operates in a similar fashion, alternately sending one message to P_1 and two messages to P_4.

- Overall, the steady-state is reached at time-step $t = 5$. One new broadcast is initiated by the source processor every time-step, so that the throughput of the schedule is equal to 1.

Period	1		2		3		4	
Time / Link	1	2	3	4	5	6	7	8
$P_s \to P_1$	m_1		m_3		m_5		m_7	
$P_s \to P_2$		m_2		m_4		m_6		m_8
$P_1 \to P_2$		m_1		m_3		m_5		m_7
$P_1 \to P_3$	×		×		m_1, m_2		m_3, m_4	
$P_2 \to P_1$	×		m_2		m_4		m_6	
$P_2 \to P_4$		×		m_1, m_2		m_3, m_4		m_5, m_6

Figure 2: An optimal schedule for the network of Figure 1.

We further use the example to illustrate the "superiority" of general graphs over DAGs, and of DAGs over spanning trees, for the SERIES problem.

3.1.2 Broadcast trees

As already pointed out, the atomic broadcast is frequently implemented using a spanning tree. This raises a natural question: what is the best throughput that can be achieved for the SERIES problem, using a single spanning tree to broadcast all the messages?

A broadcast tree $T = (V, E_T)$ is a subgraph of G, which is a spanning tree rooted at P_s, source of the broadcast. The broadcast tree can be used to broadcast r messages within a time-unit (in steady state) if the one-port constraints are satisfied:

$$\forall i \in V \qquad \sum_{j \in V, (P_i, P_j) \in E_T} r \times c_{i,j} \leqslant 1 \tag{1}$$

These are the constraints for outgoing messages: Equation (1) simply states that each node i needs the time to send the message to all of its children in the broadcast tree. As a node receives its messages from only one node (its parent in the tree), the constraint on incoming messages writes $r \times c_{f(i),i} \leqslant 1$, where $f(i)$ is the parent of i in T. This constraint is satisfied for i as soon as Equation (1) is verified for $f(i)$, so we can discard this constraint. In the following, we let $\mathrm{TP}(T)$ denote the throughput of a broadcast tree T, i.e. the maximum number of messages of unit size which can be broadcast using T in one time-unit.

What is the maximal throughput $\mathrm{TP}(T)$ that can be achieved using a sub-tree of the platform described on Figure 1? We can build two kinds of spanning trees: either both P_1 and P_2 are children of the source, or only one of them is a child of the source in the tree.

In the first case, where P_1 and P_2 are directly linked to the source, we obtain the broadcast tree of Figure 3. The labels on the figure are not communication capacities. Instead, they represent the (average) number of messages that circulate along each edge within one time-unit. The value $1/2$ means that one message is sent every two steps along the edge. Obviously, because of the one-port constraint for the source processor, this is the best throughput that can be achieved using this tree.

In the second case, one of the vertices P_1 and P_2 is not directly linked to the source. Without loss of generality, we assume that the edge (P_s, P_2) does not belong to the tree. This leads to the spanning tree of Figure 4, whose optimal throughput is $\mathrm{TP}(T) = 2/3$. Indeed, on one hand, the one-port constraint for processor P_1 states that P_1 needs 1.5 time-steps to transfer a message to its children P_2 and P_3, so we cannot achieve more than 2 broadcasts every 3 time-steps. We can indeed achieve this throughput $\mathrm{TP}(T) = 2/3$, as illustrated in the figure. Overall, this is the best throughput that can be obtained with a broadcast tree in this network.

3.1.3 Broadcast DAGs

We choose a less restrictive assumption and try to extract a Directed Acyclic Graph (DAG), instead of a broadcast tree, out of the network. Of course we look for a DAG with a single entry vertex, namely the source processor. Can we can get a better throughput than with a tree?

The answer is positive. There are only two candidates DAGs which do not reduce to spanning trees: the DAG shown on Figure 5, and its symmetric counterpart where the edge (P_1, P_2) is replaced by the edge (P_2, P_1). Without loss of generality, we restrict to the DAG of Figure 5. Because the first broadcast tree of Figure 3 is a subgraph of the DAG, we can achieve a throughput at least $1/2$. But we can get more. Figure 5 illustrates how to initiate 4

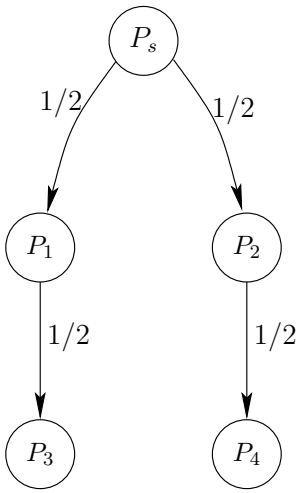


Figure 3: Broadcasting a message from P_s, using the first broadcast tree. Edges are labeled with the (average) number of messages that circulate within one time-unit.

broadcasts every 5 time-steps, hence a throughput 4/5. It turns out that this is the optimal solution with this DAG: we explain in Section 4.1 how to compute the best throughput for a DAG.

As a conclusion, we point out that the best throughput achieved for the SERIES problem strongly depends upon the graph structure allowed for transferring the messages. As the little example shows, restricting to trees is less powerful than using DAGs (troughput of $\frac{4}{5}$ instead of $\frac{2}{3}$), and restricting to DAGs is less powerful than using the full network graph (troughput of 1 instead of $\frac{4}{5}$).

It turns out that computing the optimal throughput for the SERIES problem is much easier when restricting to DAGs than when dealing with arbitrary graphs (including cycles). Therefore, we give the solution for DAGs in Section 4.1, to prepare for the difficult algorithm for general graphs (Section 5). Beforehand, we make a short digression to establish a complexity result: we show that finding the best broadcast tree (the one with the maximum throughput) for the SERIES problem is NP-Complete.

3.2 Finding the best broadcast tree

The problem of determining the best broadcast tree for the SERIES problem is stated as follows:

Definition 1 (BEST-BROADCAST-TREE(G)). *Given an edge-weighted directed graph $G = (G, V, c)$ representing a network, and a source s, find a broadcast tree T with maximum throughput* TP.

The associated decision problem is the following:

Definition 2 (BEST-BROADCAST-TREE-DEC(G,D)). *Given an edge-weighted graph $G = (V, E, c)$ representing a network, a source s, and a bound D, is it possible to find a broadcast tree T such that* TP $\geqslant D$*?*

Theorem 1. *BEST-BROADCAST-TREE-DEC(G,D) is NP-complete.*

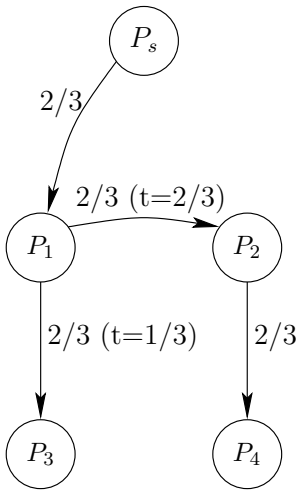


Figure 4: Broadcasting a message from P_s, using the second broadcast tree. Edges are labeled with the (average) number of messages that circulate within one time-unit. The time needed to transfer these messages is indicated between brackets.

Proof. First, BEST-BROADCAST-TREE-DEC(G,D) obviously belongs to the class NP. To prove its completeness, we proceed by a reduction from DEGREE-CONSTRAINT-SPANNING-TREE, which is known to be NP-complete [8]. An arbitrary instance $\mathcal{I}_1$ of DEGREE-CONSTRAINT-SPANNING-TREE is the following: given a graph $G_d = (V_d, E_d)$ and a positive integer $K \leqslant |V_d|$, does there exist a spanning tree for G_d in which no vertex has degree larger than K? We construct the following instance $\mathcal{I}_2$ of BEST-BROADCAST-TREE-DEC(G,D): we let $G = (V, E, c)$ where $V = V_d$, and we duplicate all edges in E_d to form the arcs of E: if the un-oriented edge (j, k) is in E_d, then both arcs (j, k) and (k, j) belong to E. We pick a source at random, i.e. any node $s \in V$. We define the communication costs as $1/(K-1)$ for all edges, except for the edges outgoing from s, whose weight is set to $1/K$. Finally, we let the bound $D = 1$. Clearly, the instance $\mathcal{I}_2$ can be constructed in time polynomial in the size of $\mathcal{I}_1$. We show that $\mathcal{I}_2$ admits a solution if and only if $\mathcal{I}_1$ does:

- Assume first that $\mathcal{I}_1$ admits a solution, and let T_d be a spanning tree of degree K in G_d. We orient T_d so as to transform it into an arborescence $T = (V, E_T)$ of G rooted at s. T is a broadcast tree in G, and we claim that its throughput is at least $D = 1$. Indeed, all nodes in T have an out-degree at most equal to $K - 1$, except s whose out-degree is at most K (because s has no parent in T). The time spent by s to send the message to its children in T is

$$\sum_{j \in V, (P_s, P_j) \in E_T} 1 \times c_{s,j} = \sum_{j \in V, (P_s, P_j) \in E_T} 1 \times 1/K \leqslant K \times 1/K = 1.$$

  The time spent by any other intermediate node i is similarly bounded:

$$\sum_{j \in V, (P_i, P_j) \in E_T} 1 \times c_{i,j} = \sum_{j \in V, (P_i, P_j) \in E_T} 1 \times 1/(K-1) \leqslant (K-1) \times 1/(K-1) = 1.$$

  Therefore TP $\geqslant D$, and $\mathcal{I}_2$ has a solution

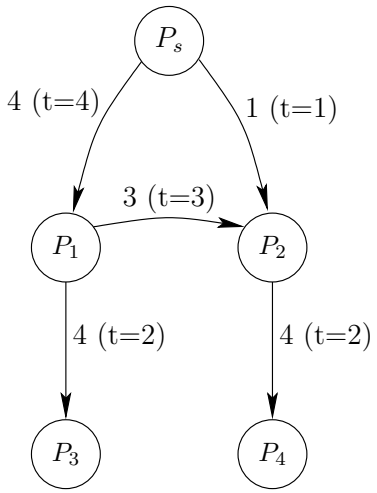


Figure 5: Broadcast on a DAG. 4 broadcasts are initiated every 5 time-steps.

- Assume now that $\mathcal{I}_2$ admits a solution, and let $T = (V, E_T)$ be the broadcast tree in G whose throughput is at least $D = 1$. Because each edge outgoing from s has weight $1/K$, the number of such edges is at most K. Indeed, the constraints on outgoing messages from node s is $\sum_{j \in V, (P_s, P_j) \in E_T} \text{TP} \times c_{s,j} \leqslant 1$. But

$$\sum_{j \in V, (P_s, P_j) \in E_T} \text{TP} \times c_{i,j} = \sum_{j \in V, (P_s, P_j) \in E_T} \text{TP} \times 1/K = \delta^+(s) \times \frac{\text{TP}}{K}$$

  where $\delta^+(i)$ is number of children of node s in T. Since TP is at least one, this gives $\delta^+(s) \leqslant K$. Similarly for any intermediate node $i \neq s$: because each edge outgoing from i has weight $1/(K-1)$, the number of such edges is at most $K-1$, and the total degree of i is at most K (one parent and at most $K-1$ children). This shows that the arborescence T rooted at s can be viewed as an un-oriented spanning tree of degree less than or equal to K, hence providing a solution to $\mathcal{I}_1$.

■

4 Series of broadcasts on a DAG

In this section we provide an algorithm to compute the optimal solution to the SERIES(V, E, c) optimization problem, in the restricted case where the graph $G = (V, E)$ is a DAG.

4.1 Linear program for DAGs

In this section, we assume the network is organized as a DAG rooted at the source P_s, and that all nodes are reachable from the source. We start with a few definitions:

- $n_{j,k}$ is the (fractional) number of unit-size messages sent from processor P_j to processor P_k during one time-unit,
- $t_{j,k}$ is the fraction of time spent by processor P_j to send messages to P_k during one time-unit.

As above, $c_{j,k}$ is the time needed to perform the transfer of a unit-size message on edge (P_j, P_k). A first equation links the two previous quantities:

$$t_{j,k} = n_{j,k} \times c_{j,k} \tag{2}$$

The activity on edge (P_j, P_k) in one time-unit is bounded:

$$\forall P_j, \forall P_k \qquad 0 \leqslant t_{j,k} \leqslant 1 \tag{3}$$

The one-port model constraints are expressed by the following equations:

$$\forall P_j, \qquad \sum_{P_k,(P_j,P_k)\in E} t_{j,k} \leqslant 1 \qquad \text{(outgoing messages)} \tag{4}$$

$$\forall P_j, \qquad \sum_{P_k,(P_k,P_j)\in E} t_{k,j} \leqslant 1 \qquad \text{(incoming messages)} \tag{5}$$

Moreover, each node should receive the same (fractional) number of messages in one time-unit (that is the throughput TP):

$$\forall P_j \text{ with } j \neq s, \qquad \sum_{P_k,(P_k,P_j)\in E} n_{k,j} = \text{TP} \tag{6}$$

We summarize these equations in a linear program (with rational coefficients and unknowns):

SSBPDAG(G), Steady-State Series of Broadcasts Problem on a DAG
Maximize TP,
subject to

$\forall P_j, \forall P_k$	$t_{j,k} = n_{j,k} \times c_{j,k}$
$\forall P_j, \forall P_k$	$0 \leqslant t_{j,k} \leqslant 1$
$\forall P_j,$	$\sum_{P_k,(P_j,P_k)\in E} t_{j,k} \leqslant 1$
$\forall P_j,$	$\sum_{P_k,(P_k,P_j)\in E} t_{k,j} \leqslant 1$
$\forall P_j$ with $j \neq s,$	$\sum_{P_k,(P_k,P_j)\in E} n_{k,j} = \text{TP}$

We make a digression to prove a useful result on building a schedule before stating the optimal solution of the Series problem.

4.2 The edge coloring lemma

Given a set of processors operating under the one-port model, can we actually execute any set of communications within a prescribed time-bound T? Of course, a necessary constraint is that Equations 4 and 5 are satisfied by each processor during the time interval:

$$\forall P_j, \qquad \sum_{P_k,(P_j,P_k)\in E} t_{j,k} \leqslant T \qquad \text{(outgoing messages)}$$

$$\forall P_j, \qquad \sum_{P_k,(P_k,P_j)\in E} t_{k,j} \leqslant T \qquad \text{(incoming messages)}$$

However, it is not obvious that these necessary conditions are sufficient to build a schedule, because only independent communications (with disjoint sender and receiver pairs) can be scheduled simultaneously. Fortunately, the result is indeed true:

Lemma 1 (Edge Coloring Lemma). *Given a platform graph (V, E, c) and a set of communication times $x = \{x_{j,k}\}$ satisfying the one-port constraints for a period of T, we can exhibit in a polynomial time a valid schedule achieving these communications (such that a node never sends nor receives two messages at the same time).*

Proof. We transform the platform graph into a weighted bipartite graph by splitting each node P_j into an outgoing node P_j^{send} and an incoming node P_j^{recv}. Each edge from P_j^{send} to P_k^{recv} is weighted by the length of the communication $t_{j,k}$. At any given time-step, we can schedule at most two communications involving a given processor, one in emission and the other in reception. Thus, at a given time step, only communications corresponding to a matching in the bipartite graph can be performed simultaneously. Therefore, we need to decompose the weighted bipartite graph into a sum of matchings. The desired decomposition of the graph is in fact an edge coloring. The weighted edge coloring algorithm of [22, vol.A chapter 20] provides in time $O(|E|^2)$ a number of matchings which is polynomial in the size of the platform graph (in fact there are at most $|E|$ matchings). Moreover, the overall weight of the matchings is equal to the maximum weighted degree of any P_j^{send} or P_j^{recv} node, so that we can use these matchings to perform the different communications. We refer the reader to Section 5.4.2, where we provide the explicit communication scheduling using the set of matchings. ■

4.3 Optimality

We state the following result, which shows that the optimal solution to the SERIES problem can be found in polynomial time:

Theorem 2. *The solution of the* SSBPDAG*(G) linear program provides the optimal solution to the* SERIES *problem on a DAG: the value TP returned by the program is the maximum number of broadcasts that can be initiated per time-unit. Furthermore, it is possible to construct the corresponding optimal periodic schedule in time polynomial in size of the input DAG.*

Proof. Because G is a DAG, we can perform a topological sort to compute a height $h(j)$ of each vertex j, such that $h(j) < h(k)$ if and only if there is a path in G from j to k. Since each node is reachable from s and there is no cycle in G, we can assume that $h(s) = 0$. Let H be the maximal height of any node in the graph.

We define q as the least common integer multiple of all values of the variables ($n_{j,k}$, $t_{j,k}$, TP) in the solution of the linear program. We will first build a schedule to perform the broadcast of $q \times \text{TP}$ messages from the source in time $q \times H$, and then we show that this schedule can be pipelined to reach the throughput TP.

We build a schedule by induction on h. We prove that each node j gets all the $q \times \text{TP}$ messages at time $q \times h(i)$ and that the one-port constraints are satisfied:

- The source already has the messages at time $q \times h(0) = 0$.
- Consider a node P_j of height h. Assume by induction that all the nodes of height less than or equal to $h-1$ have received the $q \times TP$ messages at time $q \times (h-1)$. According to Equation (6), in the solution of the linear program, we have:

$$\forall P_j \text{ with } j \neq s, \qquad \sum_{P_k,(P_k,P_j)\in E} n_{k,j} = \text{TP}$$

so we have:

$$\sum_{P_k,(P_k,P_j)\in E} q \times n_{k,j} = q \times \mathrm{TP}$$

and $recv_j(P_k) = q \times n_{k,j}$ is an integer. Here, $recv_j(P_k)$ denotes the number of messages received by P_j from P_k.

We denote as $P_{k_0}, \ldots, P_{k_r}$ the nodes actually sending data to P_j, i.e. the nodes P_k such that $(k,j) \in E$ and $recv_j(P_k) \neq 0$. All these nodes P_k have an height less than or equal to $h-1$ since $(k,j) \in E$. By induction, they have received the messages at time $q \times (h-1)$. So we can schedule the following communications:

> during the time interval $[q \times (h-1), q \times h[$, P_{k_0} sends the $recv_j(P_{k_0})$ first messages of the $q \times \mathrm{TP}$ messages to be broadcast, then P_{k_1} sends the $recv_j(P_{k_1})$ following messages, ..., eventually P_{k_l} sends the $recv_j(P_{k_r})$ last messages.

As we had the contraint $\sum_{P_k,(P_k,P_j)\in E} t_{k,j} \leqslant 1$ for incoming messages, we have

$$\sum_{0 \leqslant u \leqslant r} t_{k_u,j} \times q \leqslant q,$$

so this series of receptions in P_j lasts less than the length q of the period $[q\times(h-1), q\times h[$.

There may be several nodes of height h, so we have to check whether the sending constraints are satisfied for the nodes P_k emitting messages to potentially many nodes. Each sender P_k will have the following sending time:

$$\begin{aligned}
\sum_{P_l,(P_k,P_l)\in E \text{ and } h(P_l)=h} q \times n_{k,l} \times c_{k,l} &\leqslant \sum_{P_l,(P_k,P_l)\in E} q \times n_{k,l} \times c_{k,l} \\
&\leqslant q \times \sum_{P_l,(P_k,P_l)\in E} t_{k,l} \\
&\leqslant q
\end{aligned}$$

Since the one-port constraints are satisfied both in emission and reception, using the result presented above, we can use the *Edge Coloring Lemma* to build a schedule achieving these communications in time q.

We now prove that this schedule can be pipelined to reach the throughput TP. This is done by pipelining the schedule with a period of $T = q$. First we prove that the one-port constraints are still satisfied, during one period $[m \times q, (m+1) \times q[$:

- a node P_j receives during this period the same amount of data that it received during period $[(h(P_j)-1) \times q, h(P_j) \times q[$, and we already proved that these communications can be performed in time q
- the time spent by node P_j to send data is:

$$\begin{aligned}
\sum_{P_k,(P_j,P_k)\in E} recv_k(P_j) \times c_{j,k} &= \sum_{P_k,(P_j,P_k)\in E} q \times n_{j,k} \times c_{j,k} \\
&= q \times t_{j,k} \leqslant q
\end{aligned}$$

At this point, we have a set of integer communications $t_{j,k}$, which have the properties of one-port model. Again, we use the *Edge Coloring Lemma* to build the final schedule.

We point out a technical subtlety here: because it arises from the linear program, $\log T$ is indeed a number polynomial in the problem size, but T itself is not, and describing what happens at every time-step would be exponential in the problem size. Fortunately, there is a polynomial number of matchings, hence a polynomial number of "events" that take place during the time period, and the periodic schedule can be described in a "compact" way, i.e. in polynomial size. Again, the reader is referred to Section 5.4.2 where a compact schedule of communications is provided and proved valid. ■

4.4 Back to the example

We come back to the example given in Figure 5, for which we claimed to obtain a throughput of 4/5: this is in fact the value returned by the linear program on this example. The values of $q \times n_{j,k}$ ($q = 5$) are given along the edges in Figure 5. A topological sort gives the following heights:

node	P_s	P_1	P_2	P_3	P_4
height	0	1	2	3	3

The schedule given in the proof is the following (the number of the messages sent is mentioned between brackets, $[0, 3]$ means messages number 0 to 3):

Link \ Period	0	1	2
$P_s \to P_1$	[0,3]		
$P_s \to P_2$		[0]	
$P_1 \to P_2$		[1,3]	
$P_1 \to P_3$			[0,3]
$P_2 \to P_4$			[0,3]

Once pipelined, it gives the following communications:

Link \ Period	0	1	2 3	4	5	6
$P_s \to P_1$	[0,3]	[4,7]	[8,11]	[12,15]	[16,19]	[20,23]
$P_s \to P_2$		[0]	[4]	[8]	[12]	[16]
$P_1 \to P_2$		[1,3]	[5,7]	[9,11]	[13,15]	[17,19]
$P_1 \to P_3$			[0,3]	[4,7]	[8,11]	[12,15]
$P_2 \to P_4$			[0,3]	[4,7]	[8,11]	[12,15]

The last step is to use the edge-coloring algorithm to create a schedule where several receptions or emissions never overlap on a node. The result is the following schedule:

Link \ Period / Time	1					2					3					4				
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
$P_s \to P_1$	0	1	2	3		4	5	6	7		8	9	10	11		12	13	14	15	
$P_s \to P_2$					×					0					4					8
$P_1 \to P_2$	×	×	×			1	2	3			5	6	7			9	10	11		
$P_1 \to P_3$				×	×				0,1	2,3				4,5	6,7				8,9	10,11
$P_2 \to P_4$	×	×				×	×				0,1	2,3				4,5	6,7			

5 Series of broadcasts on a general platform

In this section, we give the optimal solution to the SERIES problem for an arbitrary platform graph, which may include cycles. We proceed in several steps, and we use a couple of technically involved theoretical results from linear programming, network flows and graph theory.

5.1 Sketch of proof

As before, the target platform graph is modeled by a directed graph $G = (V, E, c)$. Each edge $(P_j, P_k) \in E$ is labeled by its capacity $c_{j,k}$, i.e. the time needed to transfer a unit-size message from P_j to P_k. The transfer time for Z different messages between P_j and P_k is equal to $Zc_{j,k}$. Each node operates under the one-port model, so that both incoming and outgoing communications have to be performed sequentially.

There is a large number of unit-size messages to broadcast. Initially, the source processor P_s holds all these messages. Our aim is to derive a periodic algorithm that achieves the optimal throughput TP, defined as the ratio of the number of messages broadcast per time-period T in steady-state, over the duration T of the period. Not only do we have to compute the optimal throughput TP, but also we have to provide the actual construction of the periodic schedule. Our goal is to obtain a compact description of this schedule: the description of the behavior of each node during one period (i.e. the size of the code) must be polynomial in the size of the initial data. The sketch of our approach is the following:

1. We express the conditions that must be fulfilled at steady state by any periodic solution to the SERIES problem by means of a linear program. The solution of this linear program provides a lower bound for the completion time.

2. From the solution of the linear program, we derive a set of weighted trees that will be used to broadcast the different messages. We prove that the total weight of the trees enables us to reach the lower bound computed at the previous step.

3. From the set of trees, we derive a periodic solution, and we prove that it is possible to write the code of the broadcast algorithm with a size polynomial in the size of the initial data.

5.2 Lower bound

In what follows, we give a set of linear constraints that must be fulfilled by any periodic solution at steady-state. We normalize the solution so that one unit-size message is broadcast to each processor every T^* time-steps, and we aim at minimizing the period T^*. Note that this is the dual problem of Section 4.1, where we aimed at maximizing the number of messages broadcast per time-unit. However, we (try to) keep similar notations: $n_{j,k}$ denotes the number of messages that transit along edge (P_j, P_k), and $t_{j,k}$ is the total occupation time of that edge. But things get more complicated, and we need new variables $x_i^{j,k}$, as explained below.

For any node P_j, we denote by $\mathcal{N}^{out}(P_j)$ its output neighbors, i.e. the set of nodes P_k such that $(P_j, P_k) \in E$; similarly, $\mathcal{N}^{in}(P_j)$ is the set of the input neighbors of P_j, i.e. nodes P_k such that $(P_k, P_j) \in E$.

Since we deal with broadcast operations, the same messages are sent to all the nodes. But because of the pipelining, several different messages are likely to circulate simultaneously

in the network. We fictitiously distinguish the messages that are sent by the source P_s to each processor P_i, even in the end the same messages will have been sent, but maybe according to a different ordering, and via different routes. More precisely, we denote by $x_i^{j,k}$, $\forall P_i \in V$, $\forall (P_j, P_k) \in E$ the fractional number of unit-size messages sent by the source P_s to P_i and that transit on the edge between P_j and P_k:

Source and destination The first set of constraints states that the total number of messages destined to P_i and which are sent from the source P_s every period is indeed 1; also, the total number of messages which are actually received by P_i every period is also equal to 1:

$$\forall i, \quad \sum_{P_j \in \mathcal{N}^{out}(P_s)} x_i^{s,j} = 1 \tag{7}$$

$$\forall i \neq s, \quad \sum_{P_j \in \mathcal{N}^{in}(P_i)} x_i^{j,i} = 1 \tag{8}$$

Conservation law The second set of constraints states a conservation law at any intermediate processor $P_j \neq P_s, P_i$ for the messages sent to P_i:

$$\forall j, \quad P_j \neq P_s \text{ and } P_j \neq P_i, \quad \sum_{P_k \in \mathcal{N}^{in}(P_j)} x_i^{k,j} = \sum_{P_k \in \mathcal{N}^{out}(P_j)} x_i^{j,k} \tag{9}$$

This constraint reads: for each index i and each intermediate processor P_j, $j \neq i$, the number of messages destined to P_i which arrive at P_j each time-period is the same as the number of same type messages that go out of P_j. This conservation law is only valid in steady-state operation, it does not apply to the initialization and clean-up phases.

Link occupation The following set of constraints is related to the number of distinct messages that are transferred through each edge. Let us denote by $n_{j,k}$ the total number of messages that transit on the communication link between P_j and P_k. We know that for each i, the fraction $x_i^{j,k}$ of the messages sent to P_i does transit on this link. The main difficulty is that the messages transiting on the link and sent to different P_i's may be partly the same, since the same messages are overall sent to all the nodes. Therefore, the constraint $n_{j,k} = \sum_i x_i^{j,k}$, that would hold true for a scatter operation, may be too pessimistic. Since our aim is to find a lower bound for the execution time, we consider that all the messages transiting between P_j and P_k are all sub-sets of the same set, namely the largest one. In other words, we write the following constraints for the occupation time $t_{j,k}$ of the link (P_j, P_k):

$$\forall (P_j, P_k) \in E, \quad n_{j,k} = \max_i x_i^{j,k} \tag{10}$$

$$\forall (P_j, P_k) \in E, \quad t_{j,k} = n_{j,k} c_{j,k} \tag{11}$$

We also need to write down the constraints stating that communication ports for both incoming and outgoing communications are not saturated (one-port model). Let $t_j^{(in)}$ be the time spent by P_j for incoming communications, and $t_j^{(out)}$ the time spent for

out-going ones:

$$\forall j, \quad t_j^{(in)} = \sum_{P_k \in \mathcal{N}^{in}(P_j)} t_{k,j} \tag{12}$$

$$\forall j, \quad t_j^{(out)} = \sum_{P_k \in \mathcal{N}^{out}(P_j)} t_{j,k} \tag{13}$$

Execution time The last set of constraints is related to the overall period length T^* required for broadcasting a unit size message. The constraints simply state that T^* is larger than the occupation time of any edge and any in-coming or out-going communication port:

$$\forall j,k, \quad T^* \geqslant t_{j,k} \tag{14}$$

$$\forall j, \quad T^* \geqslant t_j^{(in)} \tag{15}$$

$$\forall j, \quad T^* \geqslant t_j^{(out)} \tag{16}$$

Finally, we gather all the constraints into the following linear program, which provides a lower bound for T^*, the time needed to broadcast one unit-size message:

Steady-State Broadcast Problem on a Graph SSB(G)
Minimize T^*,
subject to

$$\begin{cases}
\forall i, & \sum_{P_j \in \mathcal{N}^{out}(P_s)} x_i^{s,j} = 1 & (7)\\
\forall i \neq s, & \sum_{P_j \in \mathcal{N}^{in}(P_i)} x_i^{j,i} = 1 & (8)\\
\forall j,\ P_j \neq P_s \text{ and } P_i, & \sum_{P_k \in \mathcal{N}^{in}(P_j)} x_i^{k,j} = \sum_{P_k \in \mathcal{N}^{out}(P_j)} x_i^{j,k} & (9)\\
\forall (P_j,P_k) \in E, & n_{j,k} = \max_i x_i^{j,k} & (10)\\
\forall (P_j,P_k) \in E, & t_{j,k} = n_{j,k} c_{j,k} & (11)\\
\forall j, & t_j^{(in)} = \sum_{P_k \in \mathcal{N}^{in}(P_j)} t_{k,j} & (12)\\
\forall j, & t_j^{(out)} = \sum_{P_k \in \mathcal{N}^{out}(P_j)} t_{j,k} & (13)\\
\forall j,k, & T^* \geqslant t_{j,k} & (14)\\
\forall j, & T^* \geqslant t_j^{(in)} & (15)\\
\forall j, & T^* \geqslant t_j^{(out)} & (16)
\end{cases}$$

5.3 Weighted broadcast trees

The solution of the linear program clearly provides a lower bound for the period length needed to broadcast one unit-size message. Nevertheless, it is not clear that this bound can be achieved, because of the assumption stating that all the messages transiting on a given edge are all sub-sets of the largest set (Equation (11)). In this section, we first prove that it is possible to find a set of broadcast trees realizing exactly the lower bound, using Edmond's Branching theorem. Unfortunately, the number of trees produced by this theorem may be exponential in the problem size. Fortunately, there exists a weighted version of Edmond's Branching theorem, that produces the desired polynomial number of trees.

5.3.1 Broadcast trees and Edmond's Branching theorem

Edmond's Branching theorem applies to non-weighted graphs only, so we transform the previous graph, weighted by the $n_{j,k}$, into a multi-graph. Let us denote by N the least common multiple of all the denominators of the $n_{j,k}$'s and the $x_i^{j,k}$'s, so that $\forall i,j,k$, $Nn_{j,k}$ and $Nx_i^{j,k}$ have integer values. Moreover, let us denote by $G^{(m)} = (V,E)$ the multi-graph such that there exists exactly $Nn_{j,k}$ edges between P_j and P_k.

Edmond's branching theorem [28] shows the relationship between the number (denoted as $\kappa(G,P_0)$) of edges whose deletion makes some vertex P_i unreachable from P_s and the number of edge-disjoint spanning trees rooted at P_s:

Theorem 3 (Edmond's Branching Theorem). *The number of edge-disjoint spanning trees rooted at P_0 is exactly $\kappa(G,P_0)$.*

Lemma 2. $\kappa(G,P_0) \geqslant N$

Proof. Consider any $P_i \in V$ distinct from the source P_s. The values $x_i^{j,k}$ define a flow of total weight N between P_s and P_i. Indeed, we have:

$$\begin{cases} \forall i, & \sum_{P_j \in \mathcal{N}^{out}(P_s)} Nx_i^{s,j} = N & \text{by (7)} \\ \forall j, & \sum_{P_j \in \mathcal{N}^{in}(P_i)} Nx_i^{j,i} = N & \text{by (8)} \\ \forall j,\ P_j \neq P_0 \text{ and } P_i, & \sum_{P_k \in \mathcal{N}^{in}(P_j)} Nx_i^{j,k} = \sum_{P_k \in \mathcal{N}^{out}(P_j)} Nx_i^{k,j} & \text{by (9)} \end{cases}$$

Therefore, by the Max-flow, Min-cut Theorem of Ford and Fulkerson [7], the minimal cut of G between P_s and P_i is at least N, so that at least N edges have to be deleted in order to disconnect P_s and P_i. Since the above property holds true for any P_i, then $\kappa(G,P_0) \geqslant N$. ■

Lemma 3. $\kappa(G,P_0) \leqslant N$

Proof. Suppose that $\kappa(G,P_0) = N' > N$. Then, by the Max-flow, Min-cut Theorem of Ford and Fulkerson, for each P_i, there exists a flow a weight N' in G between P_s and P_i. Let $y_i^{j,k}$ denote the value of this flow on the edge between P_j and P_k (clearly, $y_i^{j,k} \leqslant Nn_{j,k}$ by construction), and let us denote by $z_i^{j,k} = \frac{y_i^{j,k}}{N'}$, so that the $z_i^{j,k}$'s define a flow of weight 1 between P_s and P_i. Then,

$$\begin{cases} \forall i, & \sum_{P_j \in \mathcal{N}^{out}(P_s)} z_i^{s,j} = 1 & (7) \\ \forall i, & \sum_{P_j \in \mathcal{N}^{in}(P_i)} z_i^{i,j} = 1 & (8) \\ \forall j,\ P_j \neq P_s \text{ and } P_j \neq P_i, & \sum_{P_k \in \mathcal{N}^{in}(P_j)} z_i^{j,k} = \sum_{P_k \in \mathcal{N}^{out}(P_j)} z_i^{k,j} & (9) \\ \forall (P_j,P_k) \in E, & n'_{j,k} = \max_i z_i^{j,k} \leqslant \frac{N}{N'} n_{j,k} & (10) \\ \forall (P_j,P_k) \in E, & t'_{j,k} = n'_{j,k} c_{j,k} \leqslant \frac{N}{N'} t_{j,k} & (11) \\ \forall j, & t_j^{'(in)} = \sum_{P_k \in \mathcal{N}^{out}(P_j)} t'_{j,k} \leqslant \frac{N}{N'} t_j^{(in)} & (12) \\ \forall j, & t_j^{'(out)} = \sum_{P_k \in \mathcal{N}^{out}(P_j)} t'_{j,k} \leqslant \frac{N}{N'} t_j^{(out)} & (13) \end{cases}$$

Therefore, there would exist a solution of the linear program with a completion time of $\frac{N}{N'}T^* < T^*$, which is a contradiction. Thus, $\kappa(G,P_0) \leqslant N$. ■

Finally we have the following theorem:

Theorem 4. $\kappa(G,P_0) = N$.

Therefore, by Edmond's Branching theorem, there exist N disjoint broadcast trees in G^m. There exist several implementations of Edmond's Branching theorem, but the number of different trees is of order $O(N)$. Unfortunately, a solution consisting of N broadcast trees is not compact enough for our purpose, since its encoding would take at least of order $O(N|V|)$. Indeed, since N is the least common multiple of the denominators of the $x_i^{j,k}$'s and the $n_{j,k}$'s, it can be encoded in size of order $|V||E|\log(\max(x_i^{j,k}, n_{j,k}))$. Moreover, the $x_i^{j,k}$'s and the $n_{j,k}$'s are the solution of a linear system, whose right-hand side and left-hand size matrix coefficients are initial data. Therefore, N can be encoded in polynomial size. Nevertheless, the encoding of the trees would take at least $|V|N$ bits, and would therefore be exponential in the size of original data. Fortunately, there exists a weighted version of Edmond's Branching theorem which produces a polynomial number of trees, as shown in next section.

5.3.2 Weighted version of Edmond's Branching Theorem

We use the following result, whose proof can be found in [22, vol.B chapter 53]:

Theorem 5. *Let $G = (V, E, Nn_{j,k})$ denote a weighted directed graph. There exist k_T trees $T_1, \ldots, T_{k_T}$ trees, with integer weights $\lambda_1, \ldots, \lambda_{k_T}$, such that*

$$\forall j,k, \qquad \sum_l \lambda_l \chi^T_{j,k}(T_l) \leqslant N n_{j,k},$$

where $\chi^T_{j,k}(T_l) = 1$ if $(P_j, P_k) \in T_l$ and 0 otherwise, and such that $\sum_l \lambda_l$ is maximized. Moreover, the trees can be found in strongly polynomial time and by construction,

$$k_T \leqslant |V|^3 + |E|.$$

Lemma 4. $\sum_l \lambda_l = \kappa(G, P_0) = N$.

Proof. The proof of this lemma is quite similar to the proof of Lemma 3 and uses the transformation of G into a multi-graph. ∎

Lemma 5. *The set of trees can be encoded in polynomial size with respect to initial data.*

Proof. The number of trees is bounded by $|V|^3 + |E|$ and therefore, the set of trees can be encoded in size of order $|V|(|V|^3 + |E|)$. Moreover, $\forall l$, $\lambda_l \leqslant N \max n_{j,k}$, and both N and $\max n_{j,k}$, can be encoded in polynomial size with respect to the initial data, as proved above. ∎

Therefore, the weighted version of Edmond's Branching theorem produces in polynomial time a set of weighted trees, whose encoding is compact enough, for our purpose. We will use these trees in order to broadcast the different messages. In what follows, let $m_{j,k}$ be the overall number of messages that transit between P_j and P_k on the different trees, i.e.

$$m_{j,k} = \sum_l \lambda_l \chi^T_{j,k}(T_l) \leqslant N n_{j,k} \tag{17}$$

Moreover, since the overall weight of the trees is N, and all the trees span the whole platform, we have:

$$\forall k, \sum_{P_j \in \mathcal{N}^{in}(P_k)} m_{j,k} = N \tag{18}$$

To conclude this section, we point out that we may have $m_{j,k} < Nn_{j,k}$ on some edges. Consider the toy-example of Figure 6. Not all communications arising from the linear program SSB(G) are actually used in the trees: some are discarded, because they do not improve the throughput of the broadcasts; but they do not interfere with other communications either. In other words, these communications are "useless" but "harmless".

5.4 Communication scheduling

Our goal is to use the broadcast trees defined above to perform the series of broadcasts. Thus, we need to find a schedule for communications. Indeed, since several broadcast trees will be used, node P_k will receive messages from several nodes P_j and, since P_k is only able to handle one receiving operation at the same time, communications to P_k (and from P_j) need to be scheduled carefully. We re-visit the Edge Coloring Lemma of Section 4.2 with more details, so as to extract disjoint matchings out of the set of communications: in a word, the situation is more complex here, because of the need to partition the matchings themselves into the different broadcast trees which they intersect with.

5.4.1 Weighted bipartite graph

As in Section 4.2, we construct a weighted bipartite graph $G^M = (V', E', m_{j,k}c_{j,k})$ to represent the set of communications. Let us denote

$$V' = V^{out} \cup V^{in} = (P_1^{out}, \ldots, P_p^{out}) \cup (P_1^{in}, \ldots, P_p^{in}),$$

where $p = |V|$ is the number of processors. In the bipartite graph, the edge between P_j^{out} and P_k^{in} is weighted by the quantity $m_{j,k}c_{j,k}$, which is the time necessary to transfer the overall amount of data transiting on this edge on the different trees. In order to schedule the communications, we use the refined version of the *Edge Coloring Lemma* (see [22, vol.A chapter 20]):

Theorem 6. *Let $G^M = (V, E', m_{j,k}c_{j,k})$ be a bipartite weighted graph. There exist k_M matchings $M_1, \ldots, M_{k_M}$, with integer weights $\mu_1, \ldots, \mu_{k_M}$, such that*

$$\forall j, k, \qquad \sum_i \mu_i \chi^M_{j,k}(M_i) = m_{j,k}c_{j,k} \tag{19}$$

where $\chi^M_{j,k}(M_i) = 1$ if $(P_j, P_k) \in M_i$ and 0 otherwise, and

$$\sum_i \mu_i = \max(\max_j \sum_k m_{j,k}c_{j,k}, \max_k \sum_j m_{j,k}c_{j,k}).$$

Moreover, the matchings can be found in strongly polynomial time and by construction,

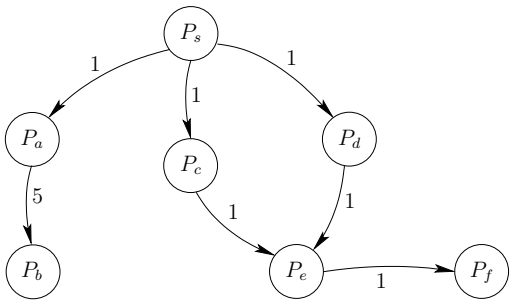
$$k_M \leqslant |E|.$$

Lemma 6. $\sum_i \mu_i \leqslant NT^*$.

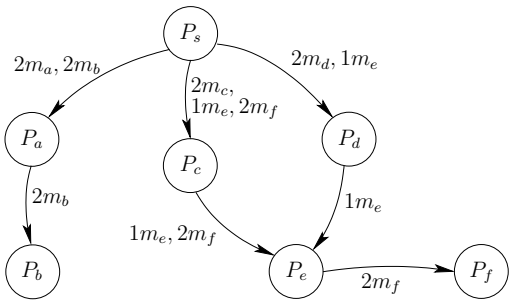
Proof. By Equation(17), $m_{j,k} \leqslant Nn_{j,k}$. Thus,

$$\sum_j m_{j,k}c_{j,k} \leqslant N \sum_j n_{j,k}c_{j,k} \leqslant NT^* \text{ by (13) and (16)}$$

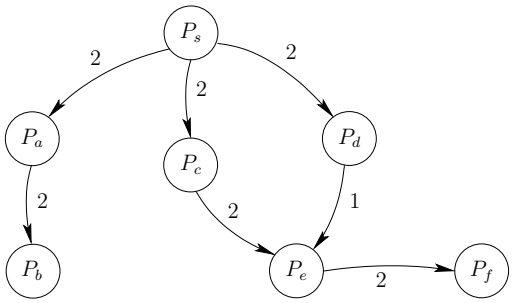
$$\text{and } \sum_k m_{j,k}c_{j,k} \leqslant N \sum_k n_{j,k}c_{j,k} \leqslant NT^* \text{ by (12) and (15)}$$



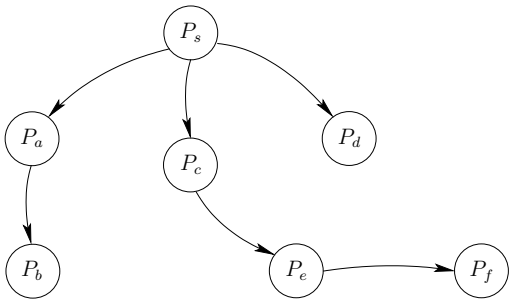
(a) Topology Graph, with the communication cost of each edge.



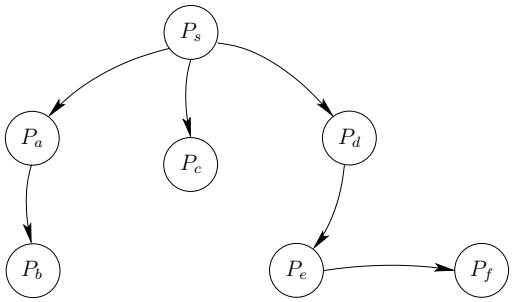
(b) Result Graph ($Nx_i^{j,k}$)



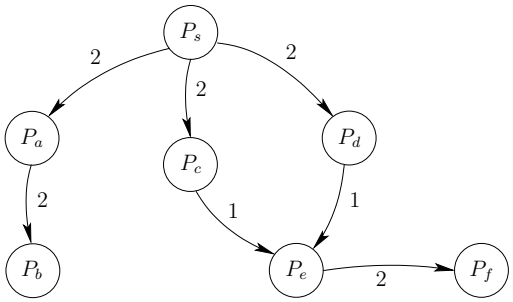
(c) Graph of the $Nn_{j,k} = \max_i (Nx_i^{j,k})$



(d) First broadcast tree, $\lambda_1 = 1$



(e) Second broadcast tree, $\lambda_2 = 1$



(f) Sum of both trees - graph of the $m_{j,k}$'s

Figure 6: Example where $m_{j,k} < Nn_{j,k}$. The optimal steady-state broadcast time T^* for one message is 5 time-units, due to edge (P_a, P_b). Figure 6(b) describes the results multiplied by the least common multiple $N = 2$, and Figure 6(c) reports the maximum values of $Nx_i^{j,k}$ on each edge. Figures 6(d) and 6(e) are the two broadcast trees extracted from the previous figure, each of them with a weight of $\lambda_l = 1$. Finally, Figure 6(f) represents the sum of these trees. On the edge (P_c, P_e), we have $m_{c,e} < Nn_{c,e}$: this edge is used by only one broadcast tree, so $m_{c,e} = 1$, whereas $Nn_{c,e} = 2$ because all messages targeting P_f are supposed to go through this edge in the optimal solution given by the linear solver, which is not the choice made when we use trees.

Thus, since $\sum_i \mu_i = \max(\max_j \sum_k m_{j,k}c_{j,k}, \max_k \sum_j m_{j,k}c_{j,k})$, then

$$\sum_i \mu_i \leqslant NT^*.$$

In fact, the inequality is indeed an equality, but the simplest way to show it is to exhibit the periodic schedule (see below).

■

5.4.2 Broadcasting algorithm

In this section, we give the precise communication scheduling during one period, i.e. the sketch of the code used to implement the broadcasts in steady state. Let us define, $\forall (P_j, P_k)$ such that $m_{j,k} \neq 0$,

$$M^{(j,k)} = \{i, \quad (P_j^{out}, P_k^{in}) \in M_i\} \text{ the set of matchings containing } (P_j^{out}, P_k^{in})$$

and

$$T^{(j,k)} = \{l, \quad (P_j, P_k) \in T_l\} \text{ the set of trees containing } (P_j, P_k).$$

Thus, we can notice that,

$$\text{by (19)}, \forall (P_j, P_k), \sum_{i \in M^{(j,k)}} \mu_i = m_{j,k}c_{j,k}$$
$$\text{and by (17)}, \forall (P_j, P_k), \sum_{l \in T^{(j,k)}} \lambda_l = m_{j,k}.$$

Let us denote by

$$s = \text{lcm}_{j,k} \left(\sum_{i \in M^{(j,k)}} \mu_i \right) \tag{20}$$

In the following, we exhibit an optimal periodic schedule: the period length is $T^{\text{per}} = NsT^*$, and Ns messages are broadcast every T^{per} time-steps, thereby achieving the optimal throughput $1/T^*$.

Let $m_j^l(q)$ be the set of messages received by node P_j from its father in the tree T_l during the q-th period. The sketch of the scheduling algorithm during the r-th period is the following:

Step i		(communications corresponding to matching M_i)
	$\forall (P_j, P_k), \quad i \in M^{(j,k)}$	
	$\forall l, \ (P_j, P_k) \in T_l$	
		P_j sends the $\frac{\mu_i s \lambda_l}{\sum_{i \in M^{(j,k)}} \mu_i}$ messages of the set $m_j^l(r-1)$ to P_k
		$m_k^l(p) = m_k^l(r) \cup m_j^l(r-1)$ the messages sent by P_j
	end	
end		

We prove the correctness of the algorithm as follows:

Duration of step i In order to estimate the duration of step i, we need to evaluate, for each P_j such that $(P_j^{out}, P_k^{in}) \in M_i$, the time needed by P_j to send all the messages:

$$\begin{aligned}\sum_{l \in T^{(j,k)}} \frac{\mu_i s \lambda_l c_{j,k}}{\sum_{i \in M^{(j,k)}} \mu_i} &= \frac{\mu_i s}{\sum_{i \in M^{(j,k)}} \mu_i} \left(\sum_{l \in T^{(j,k)}} \lambda_l \right) c_{j,k} \\ &= \frac{\mu_i s}{\sum_{i \in M^{(j,k)}} \mu_i} m_{j,k} c_{j,k} \qquad \text{by (17)} \\ &= \mu_i s \qquad \text{by (19)}\end{aligned}$$

This result does not depend on j. Furthermore, the communications involving different P_j's can be handled in parallel, because they belong to a matching. Therefore, step i can be executed within $\mu_i s$ time-units.

Length of the period The duration of the period T^{per} is the sum of the duration of the different steps:

$$\sum_i \mu_i s \leqslant N T^* s = T^{\text{per}}.$$

Number of messages $M(r,j,k)$ received by P_k and coming from P_j during the r-th period:

$$\begin{aligned}M(r,j,k) &= \sum_{i \in M^{(j,k)}} \sum_{l \in T^{(j,k)}} \frac{\mu_i s \lambda_l}{\sum_{i \in M^{(j,k)}} \mu_i} \\ &= s \sum_{l \in T^{(j,k)}} \lambda_l \\ &= s m_{j,k} \qquad \text{by (17)}\end{aligned}$$

Total number of messages received by P_k during the r-th period. Since all the messages are sent along the edges of the different trees, all the messages received by P_k are different, and are different from those received during previous periods. Therefore, the overall number of messages received by node P_k during one period is given by

$$s \sum_j m_{j,k} = sN \text{ by (18)}.$$

Therefore, during one period of duration $T^{\text{per}} = NsT^*$, each node receives exactly Ns new different messages. Therefore, the overall throughput of the SERIES algorithm during one period is $\frac{1}{T^*}$, hence its optimality. Finally, because the actual length of the period is the sum of the duration of the different steps, we derive that $\sum_i \mu_i s = T^{\text{per}}$, hence $\sum_i \mu_i = NT^*$, as claimed in the proof of Lemma 6.

5.5 Asymptotic optimality

In this section, we prove that the previous periodic schedule is asymptotically optimal: basically, no scheduling algorithm (even non periodic) can execute more broadcast operations in a given time-frame than ours, up to a constant number of operations. This section is devoted to the formal statement of this result, and to the corresponding proof.

Given a platform graph $G = (V, E, c)$, a source processor P_s holding an infinite number of unit-size messages, and a time bound K, define $opt(G, K)$ as the optimal number of messages that can be received by every target processor in a succession of broadcast operations from P_s, within K time-units. Let $\mathrm{TP}(G)$ be the solution of the linear program SSB(G) of Section 5.2 applied to this platform graph G. We have the following result:

Lemma 7. $opt(G, K) \leqslant \frac{1}{\mathrm{TP}(G)} \times K$

Proof. Consider an optimal schedule, such that the number of messages broadcast by the source processor within the K time-units is maximal, i.e. is equal to $opt(G, K)$. Each processor P_i receives $opt(G, K)$ messages from the source P_s. Each of these messages has followed a given route to reach P_i. For each edge (P_j, P_k), let $N_i^{j,k}$ be the number of messages whose final destination is P_i and which have been sent by P_j to P_k along the edge. Let $T_{j,k}$ be the total occupation time of the edge (P_j, P_k). Then the following equations hold true:

- $\forall P_j, P_k, \qquad T_{j,k} \geqslant \max_i N_i^{j,k} \times c_{j,k}$
- $\forall P_j, \forall P_k, \qquad 0 \leqslant T_{j,k} \leqslant K$
- $\forall P_j, \sum_{P_k,(P_j,P_k)\in E} T_{j,k} \leqslant K$ (time for P_j to send messages in the one-port model)
- $\forall P_j, \sum_{P_k,(P_k,P_j)\in E} T_{k,j} \leqslant K$ (time for P_j to receive messages in the one-port model)
- $\forall P_i, \forall P_j \neq P_i, P_s, \sum_{P_k,(P_j,P_k)\in E} N_i^{j,k} = \sum_{P_k,(P_k,P_j)\in E} N_i^{k,j}$ (conservation law for messages forwarded by P_j to P_i)
- $\forall P_i, P_i \neq P_s, opt(G, K) = \sum_{P_j,(P_s,P_j)\in E} N_i^{s,j}$ (same number of messages broadcast by the source to each node)
- $\forall P_i, P_i \neq P_s, opt(G, K) = \sum_{P_j,(P_j,P_i)\in E} N_i^{j,i}$ (same number of messages received by each node)

Let $x_k^{i,j} = \frac{N_i^{j,k}}{opt(G,K)}$, $n_{j,k} = \max_i x_i^{j,k}$, $t_{j,k} = n_{j,k}c_{j,k}$, $t_j^{(in)} = \sum_{P_k\in\mathcal{N}^{in}(P_j)} t_{k,j}$, and $t_j^{(out)} = \sum_{P_k\in\mathcal{N}^{out}(P_j)} t_{j,k}$. Let $T(K) = \frac{K}{opt(G,K)}$, we have $T(K) \geqslant t_{j,k}$, $T(K) \geqslant t_j^{(in)}$ and $T(K) \geqslant t_j^{(out)}$. All the equations of the linear program hold, hence $T(K) \geqslant \mathrm{TP}(G)$, since TP is the optimal value. This concludes the proof. ∎

Again, this lemma states that no schedule can send more messages that the steady-state. There remains to bound the loss due to the initialization and the clean-up phase in our periodic solution, to come up with a well-defined scheduling algorithm based upon steady-state operation. Consider the following algorithm (assume that K is large enough):

- Solve the linear program for SSB(G), compute the throughput $1/\mathrm{TP}(G)$. Determine the period T such that every communication time is an integer.

- For each processor P_i, $i \neq s$, for each message type m_k destined to P_k, we use a dedicated buffer on processor P_i, whose size is $buffer_{P_i,m_k}$. In steady-state mode, the buffer contains as many messages of each type as the number sent during one period. Note that this is the same quantity as the number of messages of type m_k received by P_i within each period. Note that all the quantities $buffer_{P_i,m_k}$ are independent of K: they only depend upon the characteristics of the platform.

- Initialization phase: fill up all buffers. This operation requires the source P_s to send a constant number of messages, independent of K. These messages are routed sequentially, along the paths used by the steady-state operation. The time needed per message is bounded by the weight of the longest path used in the routing. Altogether, this initialization requires a constant number I of time-steps.

- Similarly, let J be the time needed by the following clean-up operation: each processor sends all the messages which are stored in its buffer at the end of the last period, to their final destination processor. All these communications are performed sequentially. Again, J is a constant independent of K.

- Let $r = \lfloor \frac{K-I-J}{T} \rfloor$.

- Steady-state scheduling: during r periods of time T, operate the platform in steady-state, according to the solution of SSB(G).

- Clean-up during the J last time-units: processors forward all their messages to their final destination. No processor is active during the very last units ($K - I - J$ may not be evenly divisible by T).

- The number of messages broadcast by this algorithm within K time-units is equal to $steady(G, K) = (r + 1) \times T \times \frac{1}{\mathrm{TP}(G)}$.

Clearly, the initialization and clean-up phases would be shortened for an actual implementation, using parallel routing and distributed computations. But on the theoretical side, we do not need to refine the previous bound, because it is sufficient to prove the following result:

Theorem 7. *The previous scheduling algorithm based on the steady-state operation is asymptotically optimal:*

$$\lim_{K \to +\infty} \frac{steady(G, K)}{opt(G, K)} = 1.$$

Proof. Using the previous lemma, $opt(G, K) \leqslant \frac{1}{\mathrm{TP}(G)} \times K$. From the description of the algorithm, we have $steady(G, K) = (r + 1) \times T \times \frac{1}{\mathrm{TP}(G)} \geqslant (K - I - J) \times \frac{1}{\mathrm{TP}(G)}$, hence the result because I, J, and $\mathrm{TP}(G)$ are constants independent of K. ■

6 Pipelined broadcast

In the *pipelined broadcast* problem, the source processor broadcasts a single (large) message of total size L, which can be split into an arbitrary number of packets. To be realistic, the model must include start-up overheads in the communication times: otherwise, with a cost linear in the packet size, the best solution would be to have an infinite number of infinitely

small packets. Therefore, in this section we assume that the time to send a packet of size $n_{j,k}$ from P_j to P_k is $\beta_{j,k} + n_{j,k}c_{j,k}$. We include the start-up costs in the definition of the platform graph, which becomes $G = (V, E, c, \beta)$. The Pipe(V, E, c, β) problem is to minimize the time needed to broadcast the initial message of size L, i.e. to find the number and size of the packets, and a routing scheme for each packet, so that the total execution time is as small as possible.

Using again the periodic scheme described in Section 5, we can extend Theorem 7 and prove a result of asymptotic optimality for the Pipe optimization problem. This result is inspired by the work of Bertsimas and Gamarnik [3], who use a fluid relaxation technique to prove the asymptotic optimality of a simpler packet routing problem.

Let $T_{opt}(L)$ be the optimal time to broadcast a message of size L from the source processor P_s on the platform $G = (V, E, c, \beta)$. Assume that L is large enough, and let $\nu = \lceil \sqrt{L} \rceil$, so that $(\nu-1)^2 \leqslant L \leqslant \nu^2$. By padding the message with empty data at the end, we can consider to have ν messages of size ν to broadcast. We use the asymptotically optimal algorithm of Section 5.5 (which is based on the steady-state operation) to broadcast these ν messages. This requires a time $T_{pipe}(L)$. If L, hence ν, is large enough, this scheme remains asymptotically optimal, as shown below:

Theorem 8. *The previous scheduling algorithm based on the steady-state operation is asymptotically optimal:*

$$\lim_{L \to +\infty} \frac{T_{opt}(L)}{T_{pipe}(L)} = 1.$$

Proof. Let $w = \max_{j,k} \frac{\beta_{j,k}}{c_{j,k}}$ so that $\beta_{j,k} + \nu c_{j,k} \leqslant (\nu + w)c_{j,k}$. In steady-state, it takes T time-units to broadcast $\frac{1}{\mathrm{TP}} \times T$ unit-size messages, where TP is the solution of the linear program SSB(G), and T the integer period. Therefore, it takes $(\nu+w)T$ time-units to broadcast $\frac{1}{\mathrm{TP}} \times T$ messages of length $\nu + w$. As shown in Section 5.5, it takes a finite number of periods to reach and to quit the steady-state mode, so that it requires no more than $\left(\frac{\nu.\mathrm{TP}}{T} + C\right)((\nu + w)T)$ time-units to broadcast ν messages of length $\nu + w$. Finally, we have

$$T_{pipe}(L) \leqslant \nu^2 \mathrm{TP} + O(\nu) = L.\mathrm{TP} + O(\sqrt{L})$$

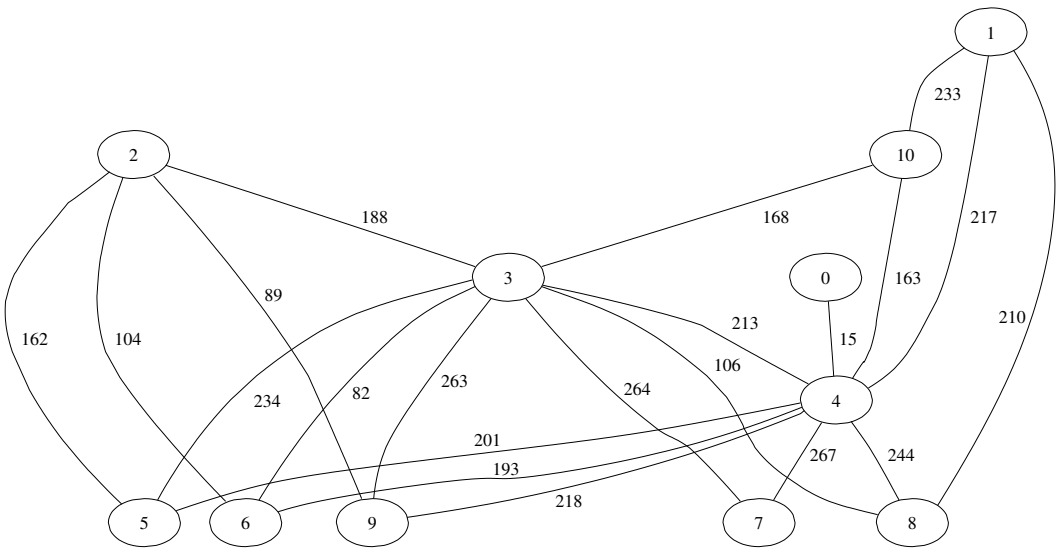
Conversely, Lemma 7 shows that $L \leqslant \frac{1}{\mathrm{TP}} \times T_{opt}(L)$, even when latencies are not taken into account, so that we have *a fortiori* that $T_{opt}(L) \geqslant L \times \mathrm{TP}$. This concludes the proof. ■

7 Experiments

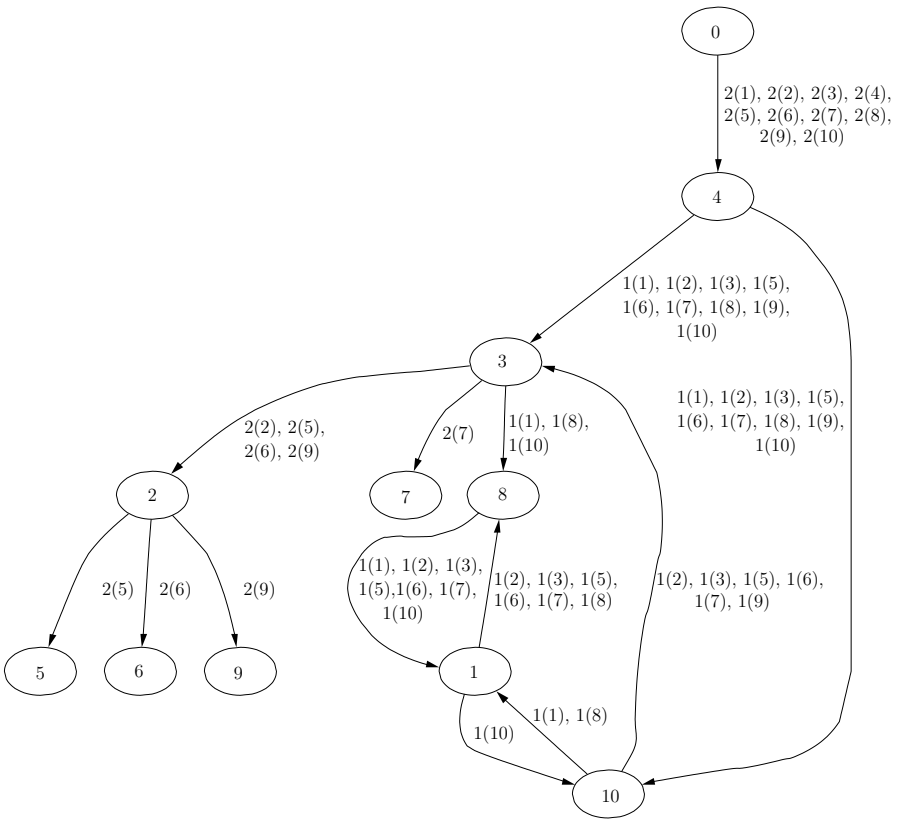
In this section, we work out a complete example. The platform is generated by Tiers, a random generator of topology [6]. The bandwidth of the links are randomly chosen, and the topology is represented on Figure7(a).

Figure 7(b) shows the results of the linear program SSB(G). The edges of this graph represent communications, and their label is a list of transfers: if edge (i, j) has the item $y(k)$ in its list, it means that $Nx_k^{i,j} = y$, so in the steady-state integer solution, y messages go through edge (i, j) to reach P_k. Here, the throughput achieved is 2 messages per period of 152 time-units.

From these communications, we extract two broadcast trees, which are represented on Figure 8, where both the logical tree and the communications extracted from Figure 7(b) are



(a) Topology. Edge e is labeled by its bandwidth $bw(e)$. The cost of a transfer is $c(e) = 1000/bw(i)$ for a single message.



(b) Communication graph

Figure 7: Experiments on a given topology.

mentioned. We point out that not all communications arising from the linear program SSB(G) are actually used in the trees: some are redundant (hence useless). The same observation was made for the toy example at the end of Section 5.3.2. For example, there is a cycle between node P_1 and P_8 for transfers, whose targets are nodes P_3, P_5, P_6 and P_7. These communications do not improve the throughput of the broadcast, but they do not interfere with other communications: indeed, the maximum of all communications on these edges is $Nx^{1,8} = Nx^{8,1} = 1$. Extracting trees from the solution of the linear program enables us to neglect such "parasitic" communications.

8 Related Work

The *atomic broadcast* problem has been studied under different models to deal with the heterogeneity of the target architecture.

Banikazemi et al. [1] consider a simple model in which the heterogeneity among processors is characterized by the speed of the sending processors. In this model, the interconnection network is fully connected (a complete graph), and each processor P_i requires t_i time-units to send a (normalized) message to any other processor. The authors discuss that this simple model of heterogeneity can well describe the different communication delays in a heterogeneous cluster. They introduce the Fastest Node First (FNF) heuristic: to construct a good broadcast tree, it is better to put fastest processors (processors that have the smallest sending time) at the top of tree. Some theoretical results (NP-completeness and approximation algorithms) have been developed for the problem of broadcasting a message in this model: see [9, 17, 16].

A more complex model is introduced in [2]: it takes not only the time needed to send a message into account, but also the time spent for the transfer through the network, and the time needed to receive the message. All these three components have a fixed part, and a part proportional to the length of the message.

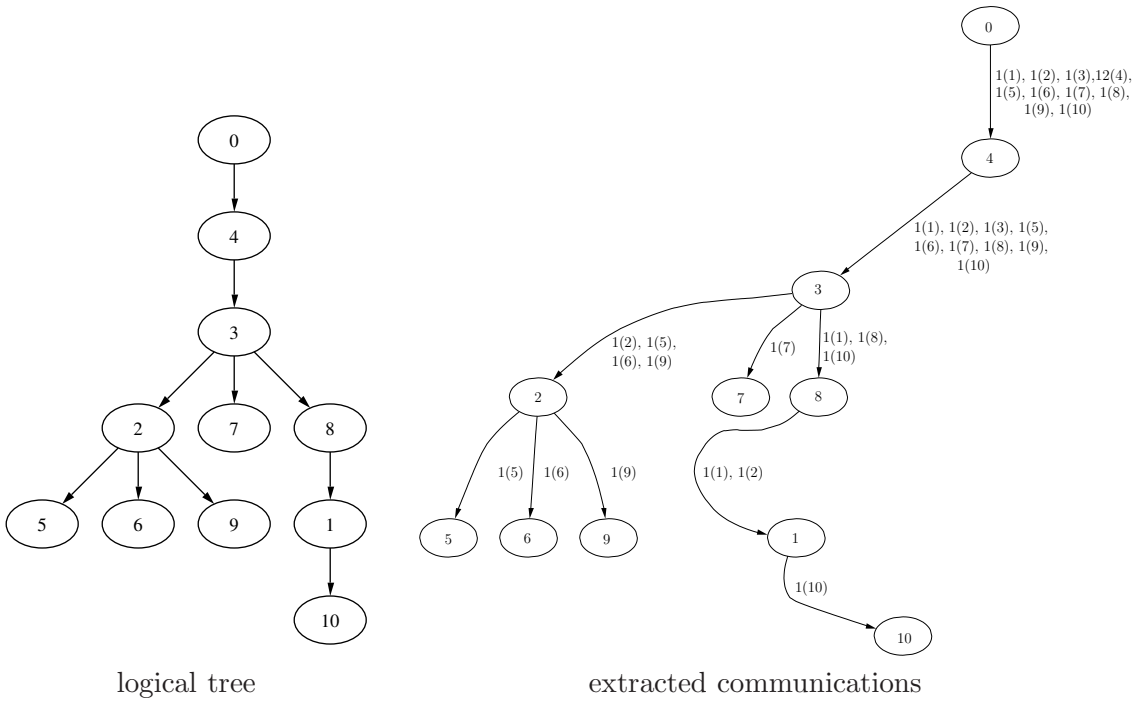
Yet another model of communication is introduced in [5, 4]: the time needed to transfer the message between any processor pair (P_i, P_j) is supposed to be divided into a start-up cost $T_{i,j}$ and a part depending on the size m of the message and the transmission rate $B_{i,j}$ between the two processors, $\frac{m}{B_{i,j}}$. Since the message size is a constant in the case of a broadcast, the total communication time between P_i and P_j is $C_{i,j} = T_{i,j} + \frac{m}{B_{i,j}}$. In [5], some heuristics are proposed for the broadcast and the multicast using this model.

All previous models assume the *one port* protocol, which we used throughout this paper: a given processor can send data to at most one neighbor processor at a time. Usually, overlapping this operation with one receiving (of independent data) is allowed.

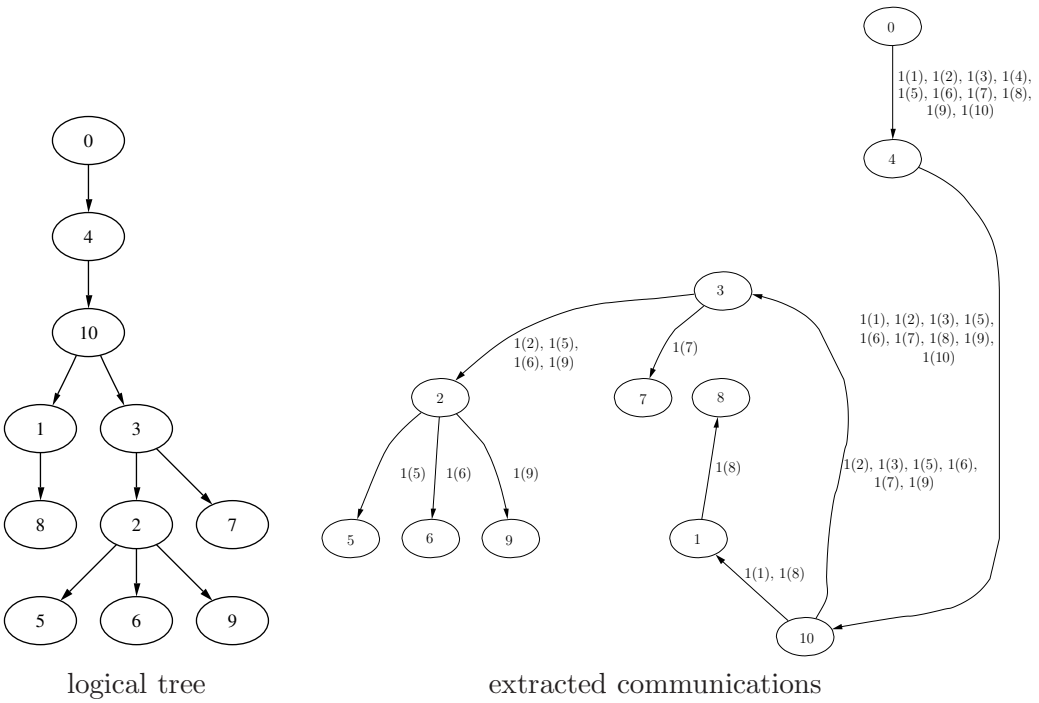
Other collective communications, such as multicast,scatter, all-to-all, gossiping and gather/reduce have been studied in the context of heterogeneous platforms: see [19, 10, 18, 15, 21] among others.

9 Conclusion

In this paper, we have studied several broadcasting problems on heterogeneous platforms. Our major objective was to maximize the throughput that can be achieved in steady-state mode, when a large number of same-size broadcasts are done in a pipeline fashion, or when a single large message is split into packets that are broadcast in pipeline fashion too. Achieving the best throughput may well require that the target platform is used in totality: we have



(a) First broadcast tree (broadcasting 1 message)



(b) Second broadcast tree (broadcasting 1 message)

Figure 8: Broadcast trees.

shown neither spanning trees nor DAGs are powerful enough. In passing, we have proved that finding the broadcast tree that achieves the best throughput is a NP-complete problem.

We have shown how to compute the best throughput using linear programming, and how to exhibit a periodic schedule, first when restricting to a DAG, and then when using a general graph. The polynomial compactness of the description comes from the decomposition of the schedule into several broadcast trees that are used concurrently to reach the best throughput. It is important to point out that a concrete scheduling algorithm based upon the steady-state operation is asymptotically optimal, in the class of all possible schedules (not only periodic solutions).

An interesting problem would be to extend this work to the case of the multicast operation, where the target processors (the receivers) form a strict subset of the computing resources. In this case, even determining the best throughput in steady-state mode seems to be a challenging problem.

References

[1] M. Banikazemi, V. Moorthy, and D. K. Panda. Efficient collective communication on heterogeneous networks of workstations. In *Proceedings of the 27th International Conference on Parallel Processing (ICPP'98)*. IEEE Computer Society Press, 1998.

[2] M. Banikazemi, J. Sampathkumar, S. Prabhu, D.K. Panda, and P. Sadayappan. Communication modeling of heterogeneous networks of workstations for performance characterization of collective operations. In *HCW'99, the 8th Heterogeneous Computing Workshop*, pages 125–133. IEEE Computer Society Press, 1999.

[3] D. Bertsimas and D. Gamarnik. Asymptotically optimal algorithm for job shop scheduling and packet routing. *Journal of Algorithms*, 33(2):296–318, 1999.

[4] P.B. Bhat, C.S. Raghavendra, and V.K. Prasanna. Adaptive communication algorithms for distributed heterogeneous systems. *Journal of Parallel and Distributed Computing*, 59(2):252–279, 1999.

[5] P.B. Bhat, C.S. Raghavendra, and V.K. Prasanna. Efficient collective communication in distributed heterogeneous systems. In *ICDCS'99 19th International Conference on Distributed Computing Systems*, pages 15–24. IEEE Computer Society Press, 1999.

[6] Kenneth L. Calvert, Matthew B. Doar, and Ellen W. Zegura. Modeling internet topology. *IEEE Communications Magazine*, 35(6):160–163, June 1997. Available at `http://citeseer.nj.nec.com/calvert97modeling.html`.

[7] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms*. The MIT Press, 1990.

[8] M. R. Garey and D. S. Johnson. *Computers and Intractability, a Guide to the Theory of NP-Completeness*. W. H. Freeman and Company, 1991.

[9] N.G. Hall, W.-P. Liu, and J.B. Sidney. Scheduling in broadcast networks. *Networks*, 32(14):233–253, 1998.

[10] J.-I. Hatta and S. Shibusawa. Scheduling algorithms for efficient gather operations in distributed heterogeneous systems. In *2000 International Conference on Parallel Processing (ICPP'2000)*. IEEE Computer Society Press, 2000.

[11] K. Hwang and Z. Xu. *Scalable Parallel Computing*. McGraw-Hill, 1998.

[12] S. L. Johnsson and C.-T. Ho. Optimum broadcasting and personalized communication in hypercubes. *IEEE Trans. Computers*, 38(9):1249–1268, 1989.

[13] H. Ko, S. Latifi, and P.K. Srimani. Near-optimal broadcast in all-port wormhole-routed hypercubes using error-correcting codes. *IEEE Trans. Parallel and Distributed Systems*, 11(3):247–260, 2000.

[14] V. Kumar, A. Grama, A. Gupta, and G. Karypis. *Introduction to Parallel Computing*. The Benjamin/Cummings Publishing Company, Inc., 1994.

[15] R. Libeskind-Hadas, J. R. K. Hartline, P. Boothe, G. Rae, and J. Swisher. On multicast algorithms for heterogeneous networks of workstations. *Journal of Parallel and Distributed Computing*, 61(11):1665–1679, 2001.

[16] P. Liu. Broadcast scheduling optimization for heterogeneous cluster systems. *Journal of Algorithms*, 42(1):135–152, 2002.

[17] P. Liu and T.-H. Sheng. Broadcast scheduling optimization for heterogeneous cluster systems. In *SPAA'2000, 12th Annual ACM Symposium on Parallel Algorithms and Architectures*, pages 129–136. ACM Press, 2000.

[18] P. Liu and D.-W. Wang. Reduction optimization in heterogeneous cluster environments. In *14th International Parallel and Distributed Processing Symposium (IPDPS'2000)*. IEEE Computer Society Press, 2000.

[19] B. Lowekamp and A. Beguelin. Eco: Efficient collective operations for communication on heterogeneous networks. In *10th International Parallel and Distributed Processing Symposium (IPDPS'96)*. IEEE Computer Society Press, 1996.

[20] J.A. Moore and M.J. Quinn. Generating an efficient broadcast sequence using reflected gray codes. *IEEE Trans. Parallel and Distributed Systems*, 8(11):1117–1122, 1997.

[21] F. Ooshita, S. Matsumae, and T. Masuzawa. Efficient gather operation in heterogeneous cluster systems. In *Proceedings of the 16th International Symposium on High Performance Computing Systems and Applications (HPCS'02)*. IEEE Computer Society Press, 2002.

[22] A. Schrijver. *Combinatorial Optimization: Polyhedra and Efficiency*, volume 24 of *Algorithms and Combinatorics*. Springer-Verlag, 2003.

[23] M. Snir, S. W. Otto, S. Huss-Lederman, D. W. Walker, and J. Dongarra. *MPI the complete reference*. The MIT Press, 1996.

[24] Y-C. Tseng, S-Y. Wang, and C-W. Ho. Performance complexity of lu factorization with efficient pipelining and overlap on a multiprocessor. *Parallel Processing Letters*, 5(2):44–61, 1995.

[25] Y.-C. Tseng, S.-Y. Wang, and C.-W. Ho. Efficient broadcasting in wormhole-routed multicomputers: a network-partitioning approach. *IEEE Trans. Parallel and Distributed Systems*, 10(1):44–61, 1999.

[26] S-Y. Wang and Y-C. Tseng. Algebraic foundations and broadcasting algorithms for wormhole-routed all-port tori. *IEEE Trans. Computers*, 49(3):246–258, 2000.

[27] J. Watts and R. Van De Geijn. A pipelined broadcast for multidimensional meshes. *Parallel Processing Letters*, 5(2):281–292, 1995.

[28] D. B. West. *Introduction to Graph Theory*. Prentice Hall, 1996.