A Distributed Scheduling Algorithm for Wireless Networks with Constant Overhead and Arbitrary Binary Interference.
Jean-Claude Bermond, Dorian Mazauric, Vishal Misra, Philippe Nain

To cite this version:

HAL Id: inria-00505519
https://inria.hal.science/inria-00505519
Submitted on 8 Sep 2022

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L’archive ouverte pluridisciplinaire HAL, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d’enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.
A Distributed Scheduling Algorithm for Wireless Networks with Constant Overhead and Arbitrary Binary Interference.

Jean-Claude Bermond¹, Dorian Mazauric¹,², Vishal Misra¹, Philippe Nain²

¹ : Mascotte, INRIA, I3S(CNRS/Univ of Nice-Sophia), Sophia Antipolis, France
² : Maestro, INRIA, Sophia Antipolis, France

Categories and Subject Descriptors: F.2.2 Nonnumerical Algorithms and Problems


Keywords: wireless network, transmission scheduling, interference, distributed algorithm, stability.

This work investigates distributed transmission scheduling in wireless networks. Due to interference constraints, “neighboring links” cannot be simultaneously activated, otherwise transmissions will fail. Hereafter, we consider any binary model of interference. We follow the model described by Bui, Sanghavi, and Srikant in [3]. We assume. There are two phases during each slot: first the control phase which determines what links will be activated next, followed by a transmission phase during which data is transmitted. We assume random arrivals of data (packets) on each link during each slot, so that a buffer (queue) is associated to each link. It takes exactly one time slot to transmit a packet on a link. Since nodes do not have a global knowledge of the network, our aim (like in [3]) is to design for the control phase a distributed algorithm which identifies a set of non-interfering links.

For example, in the primary node interference model (where two links interfere only if they are incident), a set of non-interfering edges is called a matching. In order to ensure the largest stability region of the system we want to choose links so that the sum of their weight (queue-length) is as large as possible so as to realize a maximum matching. The idea behind the maximum matching is to decrease the largest queue. Centralized algorithms have been proposed to solve this problem both for random arrivals in [7] and deterministic arrivals in [6]. To be efficient the control phase should be as short as possible; this is done by exchanging control messages during a constant number of mini-slots (constant overhead).

In this work we design the first fully distributed local algorithm with the following properties: it works for any arbitrary binary interference model; it has a constant overhead (independent of the size of the network and of the queue-lengths) and it requires no knowledge. Indeed the algorithm in [5] does not have a constant overhead, whereas the one described in [3] is only valid for the primary node interference model. Furthermore, both algorithms need to know the queue lengths of the “neighboring links”, which are difficult to obtain in a wireless network with interference. We prove that our algorithm gives a maximal set of active links (in each interference set, there is at least one active edge). We also give sufficient conditions for stability under Markovian assumptions. Finally, the performance of our algorithm (throughput, stability) is investigated and compared via simulations to that of previous schemes.

Let us sketch the main ideas of our algorithm Algorithm Log. Details, proofs and examples can be found in the full version [1].

We use a binary symmetric interference model and define the interference set of a link, that is the number of packets that can serve during one time slot if the link is active. Let q(t,e) (also called the weight of link e) be the number of packets in the queue of link e ∈ E at the beginning of step t ≥ 1. Algorithm Log concerns only the control phase which determines at each step the set of active links (on which data will be sent during the data phase). The control phase is divided into mini-slots.

A first idea consists in associating virtual weights (see below) to current queue-lengths. These virtual weights will be used to determine in each mini-slot which edges will send a control message. Only edges with weight greater than their capacity (q(t,e) ≥ c(e)) will participate in Algorithm Log. We then split the values of q(t,e)/c(e) into a small number K of disjoint intervals (classes) I_0,...,I_{K-1} with the constraint that the largest class I_{K-1} contains all the values greater than some value L: I_{K-1} = [L, +∞[. In the example (Figure 1), we choose c(e) = 1, L = 100 and K = 5; the five classes associated to the values of q(t,e) are: I_0 = [1, 10); I_1 = [10, 30); I_2 = [30, 60); I_3 = [60, 100] and I_4 = [100, +∞[.

When K is small, many interfering edges might belong to the same interval. To differentiate them, we also give to each edge a color (integer), γ(e), in such a way that two interfering edges have different colors. Let C be the number of colors. In order to get different rankings in consecutive slots, we will permute the colors. To this end, we associate in the slot t the value γ(t,e) = ((γ(e) + t - 2) mod C + 1. Figure 1(b) shows a coloring for a cycle of length 9, with C = 3, when the primary node interference model is used (i.e. two edges interfere if they are incident). The coloring is given at some step t; the first edge will get color 3 at step t + 1, then color 1 at step t + 2, color 2 again at step t + 3 and so on.

Finally, we assign to an edge of class i a virtual weight q_i(t,e) = C * i + γ(t,e). The virtual weights satisfy the two properties: for any edge e, 0 < q_i(t,e) ≤ CK ∀t ≥ 1, and if e′ ∈ e(e) (and e ∈ e(e′)) then, q_i(t,e) ≠ q_i(t,e′). Let T = [log_2(CK) + 1. We associate to q_i(t,e) a control vector v_{i,e}(t) = (v_{i,e}(1),...,v_{i,e}(T)) where v_{i,e}(i), 1 ≤ i ≤ T, corresponds to the ith bit of q_i(t,e). Table 1 gives these values for the example in Figure 1(c).

Another main idea of Algorithm Log is to use interference as information. More precisely at each mini-slot of Algorithm Log, an edge can be active, inactive, or still undetermined. The control phase is itself divided into subphases. The first subphase consists of T mini-slots. During a mini-slot t, 1 ≤ t ≤ T, e ∈ E sends a control message if and only if e is still undetermined and v_{i,e}(i) = 1. We have the three rules:

Copyright is held by the author/owner(s). SIGMETRICS'10, June 14–18, 2010, New York, New York, USA. ACM 978-1-4503-0038-4/10/06.
if \( (b) \) edge in \( \epsilon \), if processes, we establish a sufficient stability condition by using the assumptions that the arrival processes are independent renewal

shows the application of \( \text{Algorithm Log} \) of active links, meeting the interference constraints. Figure 1(d-g) and does not send a control message, then

\[ \alpha \]

inactive edges without active links in their interference sets (Fig

After subphase

If we choose \( \alpha \), then for any edge \( e \in \epsilon \), where the two active edges do not form a maximal matching. To deal with this situation, we apply

\[ \text{Algorithm Log} \]

In 

In [1] we analyze the stability of \( \text{Algorithm Log} \). Under the assumptions that the arrival processes are independent renewal processes, we establish a sufficient stability condition by using the approach in [2, Theorem 1]. To state this condition, let \( A_t(e) \) be the number of arrivals on link \( e \) in slot \( t \). Define \( H(e) \) as the maximum number of links that can be scheduled if link \( e \) is not scheduled and

Let \( H := \max(1, \max_{e \in \epsilon} H(e)) \). We show in [1] that the load vector \( b = (E[A_t(e), e \in E]) \) stabilizes \( \text{Algorithm Log} \) if there exists \( \epsilon > 0 \) such the vector \( CH/(C + H - 1))(b + \epsilon c) \) belongs to the capacity region introduced by Tassiulas et al [8], where \( c := (e(E), e \in E) \) is the link capacity vector. We conjecture that if for any edge \( e \), \( \sum_{e \in \epsilon \in \epsilon \cup \{e\}} E[A_t(e')] < \min_{e \in \epsilon \cup \{e\}} c(e') \), then \( \text{Algorithm Log} \) is stable.

In [1] we report many simulation results which confirm this conjecture and show a very good behavior of the algorithm and better than that of [3]. In particular \( \text{Algorithm Log} \) gives a set of active edges of large weight and the size of the queues remains small.

Acknowledgment: this work has been partially supported by région PACA, ANR AGAPE and DIMAGREEN, and European project IST FET AEOLUS.

1. REFERENCES


<table>
<thead>
<tr>
<th>( q(e) )</th>
<th>( q(e) )</th>
<th>class</th>
<th>( \gamma(e) )</th>
<th>( \gamma(e) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>10</td>
<td>10</td>
<td>1</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>10</td>
<td>10</td>
<td>1</td>
<td>5</td>
<td>2</td>
</tr>
<tr>
<td>10</td>
<td>10</td>
<td>0</td>
<td>3</td>
<td>6</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>100</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>100</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

**Table 1:** \( (q(e), v = v_{t,e}) \) for every possible pair \( (q(e), \gamma(e)) \) for \( \text{Algorithm Log} \): \( C=3, K=5, L=100 \)