# Some Synchronization Issues in OSPF Routing

Anne Bouillard, Claude Jard, Aurore Junier

# Some Synchronization Issues in OSPF Routing

Anne Bouillard[1], Claude Jard[2] and Aurore Junier[3]
[1] *ENS/INRIA, Paris, France*
[2] *LINA, University of Nantes, France*
[3] *INRIA, Rennes, France*
*Anne.Bouillard@ens.fr, Claude.Jard@univ-nantes.fr, Aurore.Junier@inria.fr*

Keywords:
OSPF routing; synchronization; simulation; Time Petri nets

Abstract:
A routing protocol such as OSPF has a cyclic behavior to regularly update its view of the network topology. Its behavior is divided into periods. Each period produces a flood of network information messages. We observe a regular activity in terms of messages exchanges and filling of receive buffers in routers. This article examines the consequences of possible overlap of activity between periods, leading to a buffer overflow. OSPF allows "out of sync" flows by considering an initial delay (phase). We study the optimum calculation of these offsets to reduce the load, while maintaining a short period to ensure a protocol reactive to topology changes. Such studies are conducted using a simulated Petri net model. A heuristic for determining initial delays is proposed. A core network in Germany serves as illustration.

# 1 INTRODUCTION

Routing protocols generally work in a dynamic environment where they have to constantly monitor changes. This function is implemented locally in routers by a programming loop that generates regular behaviors. Open Shortest Path First (OSPF) protocol (Moy, 1998) is an interesting example, widely used in networks. OSPF is a link-state protocol that performs internal IP routing. This protocol regularly fills the network with messages "hello" to monitor the changes of network topology and messages "link state advertisements" (LSA) to update the table of shortest paths in each router.

A lot of work (Francois et al., 2005; Basu and Riecke, 2001) has been devoted to stability issues. The stability is required if there is a change in the network state (e.g., a link goes down), all the nodes in the network are guaranteed to converge to the new network topology in finite time (in the absence of any other events). The question is difficult when the change is determined as a result of a bottleneck in a router (as possible in the OPSF-TE (Katz et al., 2003)). If the response to a congestion is the exchange of additional messages, the situation may become critical. But it has been proved (Basu and Riecke, 2001) that OSPF-TE is rather robust in that matter.

In this article we look at a related problem which is to focus on the possibilities of congestion of the input buffers of routers due to LSA traffic. Indeed, we believe that there are situations where the cyclical behavior of routers may cause harmful timings in which incoming messages collide in a very short time in front of routers.

In current implementations, the refresh cycle is very slow and congestion is unlikely in view of the routers' response time. Nevertheless, we address the question to increase the refresh rate to ensure better responsiveness to changes. This article shows a possibility of divergence, and discusses the possibilities of avoiding harmful synchronization by adjusting the phase shift of cyclical behavior.

The approach is as follows. We modeled LSAs exchanges using Time Petri Nets (in a fairly abstract representation). This model was simulated for a topology of 17 nodes representing the heart of an existing network in Germany (data provided by Alcatel). We then demonstrated the possibility of accumulation of messages for well-chosen parameter values. Accumulation is due to a possible overlap of refresh phases in terms of messages. To validate this model, and thus the reality of the observed phenomenon, we reproduced it on a network emulator available from Alcatel. Curves could indeed be replicated. Parameter values were different, but it was difficult to believe that the model scaled with respect to the rough abstraction performed. Once the problem identified, the question is then to try to solve it by computing optimum initial delays. Such a computation can be performed using linear integer programming on a simplified graphical model. We will show using simulation that the computed values are relevant to avoid message accumulation in front of routers.

The rest of the paper is organized as follows: we first present in section 2 the modeling of the LSA flooding process and its validation. In section 3, simulation shows a possible overload of buffers depending on the refresh period. Then, in section 4, we study a possible adjustment of the initial delays, which aims at minimizing the overload. We show how to compute these delays. The impact is then demonstrated using simulation.

# 2 TPN MODELING OF THE LSA FLOODING PROCESS

## 2.1 LSA flooding process

The network is represented by a directed graph $G = (V, E)$, where $V$ is a finite set of $n$ vertices (the routers) and $E$ is a binary relation on $V$ to represent the links. The $i^{th}$ router is denoted by $R_i$. The set $\mathcal{V}(R_i)$ denotes the set of neighbors of $R_i$, of cardinality $|\mathcal{V}(R_i)|$. To help the reader Table 1 gives the list of the main notations introduced in this paper.

The LSA flooding occurs periodically every $T_r$ seconds (30 minutes in the standard). Thus, the LSA flooding process starts at time $kT_r$, $\forall k \in \mathbb{N}$.

The LSA of a router $R_i$ records the content of its database. Then, $R_i$ shares this LSA (denoted $LSA_i$) with its neighbors to communicate its view of the network at the beginning of each period. The router $R_i$ sends $LSA_i$ after an *initial delay* $d_i$. More precisely, $R_i$ sends $LSA_i$ at $d_i + kT_r$, $\forall k \in \mathbb{N}$. Suppose that a router $R_j$ receives $LSA_i$ and that it starts processing it at time $t$. Then, $R_j$ ended the processing of $LSA_i$ at time $t + T_p$, where $T_p$ is the time needed by any router to process an LSA or an acknowledgment (Ack). During this processing, $R_j$ updates its database and sends a new LSA to its other neighbors if some new information is learned. Consequently, $R_j$ could send a new LSA at time $t + T_p$, and its neighbors will receive it at time $t + T_p + T_t$, where $T_t$ represents the time to send a message.

Note that any information received by $R_j$ can be taken into account if some properties are satisfied. The most important one is the age of the LSA. An LSA that is too old is simply ignored. In all cases, at time $t + T_p$, $R_j$ sends an Ack to $R_i$. The objective is to inform $R_i$ that $LSA_i$ has been correctly received. In parallel, $R_i$ waits for an Ack from all of its neighbors before a given time. If an Ack is not received before the end of this time, $R_i$ sends $LSA_i$ again until an Ack is properly received.

The LSA flooding process ends when every router has synchronized to the same database.

## 2.2 The simulation model

Time Petri Net (TPN) (Jard and Roux, 2010) is an efficient tool to model discrete-event systems and to capture the inherent concurrency of complex systems. In the classical definition, transitions are fired over an interval of time. Here, transitions are fired at a fixed time. This assumption is justified by observations of actual OSPF traces whose data processing time does not vary that much. In our case, the formal definition of TPN is the following:

**Definition 2.1** (Time Petri Net)**.** *A Time Petri Net (TPN) is a tuple $(P, T, B, F, M_0, \varphi)$ where*

- *$P$ is a finite non-empty set of places;*
- *$T$ is a finite non-empty set of transitions;*
- *$B : P \times T \to \mathbb{N}$ is the backward incidence function;*
- *$F : T \times P \to \mathbb{N}$ is the forward incidence function;*
- *$M_0 : P \to \mathbb{N}$ is the initial marking function;*
- *$\varphi : T \to \mathbb{N}$ is the temporal mapping of transitions.*

The remainder of this part is devoted to the construction of the TPN that models message exchanges of the LSA flooding process. The objective is to model and observe the dynamic behavior of a given network.

**Router modeling** The TPN that models the behavior of the LSA flooding process in a router $R_i$ needs three timers: $d_i$, $T_r$ and $T_p$. Their functions are: creating $LSA_i$, managing a message received and retransmitting a received LSA when needed. Messages are processed one by one. The following paragraphs present each functional part of the TPN that models a router.

• *Place Processor* Initially this place contains one token, representing the processing resource of a router that is used to process LSAs and Acks. This place mimics the queuing mechanism of $R_i$ and guaranties that only one message is

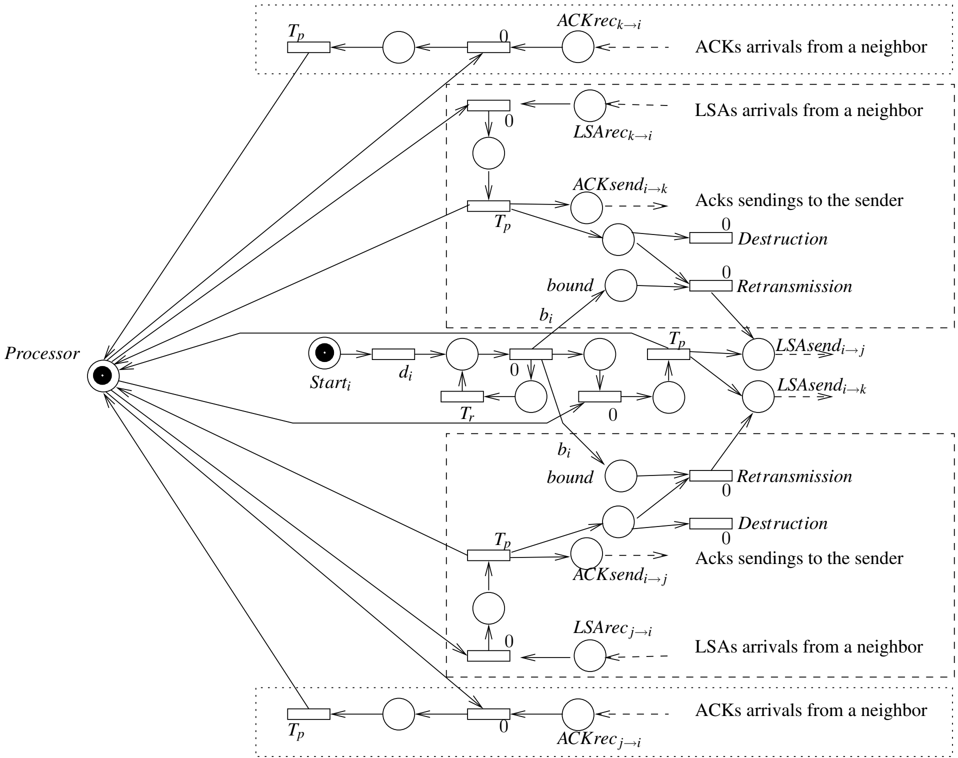


Figure 2: TPN of a router $R_i$ that has two neighbors, $R_j$ and $R_k$.

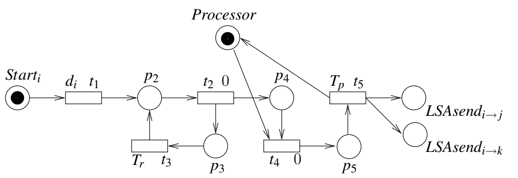


Figure 1: Part of TPN that creates the LSA of a router $R_i$.

processed at once. For each different kind of messages ($LSA_i$ and *Ack*) the processing mechanism is the following: an instantaneous transition is fired, to reserve the resource of $R_i$. Note that it can only be fired if a message is waiting. Then the successor transition with timing $T_p$ can be fired, modeling the processing time of the router, and *Processor* becomes marked again, enabling the processing of a new message.

• *Creation of LSA* Figure 1 represents the part of the TPN that creates $LSA_i$s at time $d_i + kT_r$, for $k \in \mathbb{N}$ in router $R_i$. Initially $Start_i$ contains one token, $t_1$ fires at time $d_i$ and a token appears in $p_2$ at time $d_i$ for the first time. Afterward, the cycle $p_2, t_2, p_3, t_3$ generates a token in $p_4$ at times $d_i + kT_r$, $k \in \mathbb{N}$. Those token will be processed using the mechanism described above, generating tokens in places $LSAsend_{i\to j}$, $R_j \in \mathcal{V}(R_i)$.

• *Reception of an Ack* (dotted rectangles on Figure 2) A token in $ACKrec_{j\to i}$ represents this event. It is processed using the mechanism described above and does not generate any new message.

• *Reception of an LSA from a neighbor* (dashed rectangles in Figure 2). A token in place $LSArec_{j\to i}$ represents this event. It is processed using the mechanism described above and generate an *Ack*, that is sent to the sender. It can also possibly generate an *LSA* message that will be retransmitted to its other neighbors (transition *Retransmission*). Otherwise, the token is destroyed (transition *Destruction*). In the flooding mechanism, an $LSA_j$ is retransmitted only if it is received for the first time during one flooding period. That way, the LSA flooding process ensures that every router converges to the same database before the end of every period. To model this, we bound the number of retransmissions per period (for $R_i$, the number of retransmissions of an LSA received from $R_j$ is $b_i$, that is modeled by placing $b_i$ tokens in each place *bound* of $R_i$ at the beginning of each period). The tokens are inserted in these places by weighted arcs between $t_2$ and each place *bound*.

• *Global TPN* Figure 2 represents the behavior for one router. Such a net is built for each router. Finally, place $LSAsend_{i\to j}$ (resp.

$ACKsend_{i\to j}$) is connected to place $LSArec_{i\to j}$ (resp. $ACKrec_{i\to j}$) by inserting a transition $LSA_{i\to j}$ (resp. $ACK_{i\to j}$) with firing time $T_t$ between them.

## 2.3 Model validation

We performed our experimentations on the 17-node German telecommunication network represented in Figure 3. This article focuses on the study of router $R_8$ that has the largest number of neighbors ($|\mathcal{V}(R_8)| = 6$).

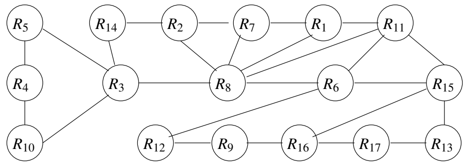


Figure 3: German telecommunication network.

The arrivals of LSAs and Acks in the actual network are captured by an emulation using the Quagga Routing Software Suite (Ishiguro, 2012), where each node is set from an Ubuntu Linux machine that hosts a running instance of the Quagga Routing Software Suite. Figure 4 represents the arrival of messages in $R_8$ by the emulation of the LSA flooding on the German topology during 8000s with $T_r = 1800$s.

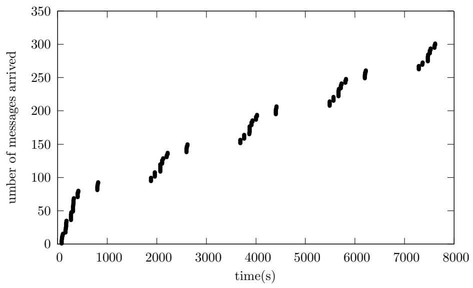


Figure 4: Emulation of the arrivals to $R_8$.

During the emulation, the processors of routers are parametrized with a 900 MHz CPU, and the mean size of an LSA (resp. an Ack) is 96 bytes (resp. 63 bytes). The processing time of an LSA (resp. an Ack) is approximately 0.8 $\mu$s (resp. 0.5 $\mu$s). The transmission time of an LSA (resp. an Ack) in 96 ms (resp. 64 ms).

Unfortunately, these parameters can not be used directly to parametrize the TPN, as the TPN only represents the behavior of the LSA flooding process. However, an actual router is much more loaded. Thus, $T_p$ and $T_t$ must be adjusted to include the whole load of the router.

The simulations presented in this article are produced by the software Renew (see (Kummer et al., 2003)) which can simulate Time Petri Nets. Note that the TPN are automatically generated (the TPN that models the German Telecommunication network is not represented here due to its size). Figure 5 represents the simulation of message arrivals using the TPN where $T_r = 1800$s, $T_p = 15$s, $T_t = 30$s. To correspond to the sendings emulated in Figure 4 the number of LSAs retransmitted per neighbour during a period is $b_i = \lceil \frac{(n-1)}{4|\mathcal{V}(R_i)|} \rceil$.

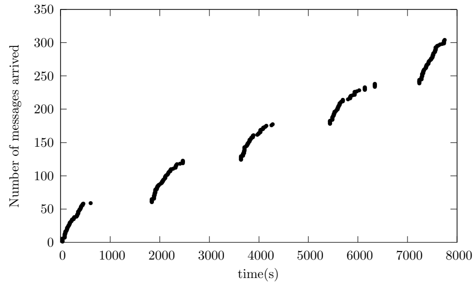


Figure 5: Message arrivals to $R_8$ with $T_r = 1800$s.

One can observe that Figure 4 and 5 are quite similar: the parameters chosen as above are defined to represent the actual behavior of an LSA flooding process. The two curves are both composed of periods that last 1800s. They show on each period a burst of message arrivals that lasts approximately 800s, then message arrivals stop until the next period. We therefore conclude that our abstract model correctly captures the phenomenon of LSA flooding.

From now on we fix the parameters ($(b_i)_{i\in\{1,\dots,n\}}$, $T_p$, $T_t$ and $T_r$) as defined above.

# 3 STUDY OF PERIOD LENGTH

We study here the effect of the period length $T_r$ on both message arrivals and queue length. We first discuss the normal case where $T_r = 1800$s. Then, we present a congested case where $T_r = 514$s. Finally, we observe a limit case where $T_r = 1000$s.

## 3.1 Low traffic case

Figure 6 represents the simulated queue length of $R_8$ during $10^5$s (approx. 1 day), where $T_r = 1800$s. One can observe a lot of fluctuations. At the beginning of each period $R_8$ receives and processes messages. However, the number of messages that are received is much larger than those which are

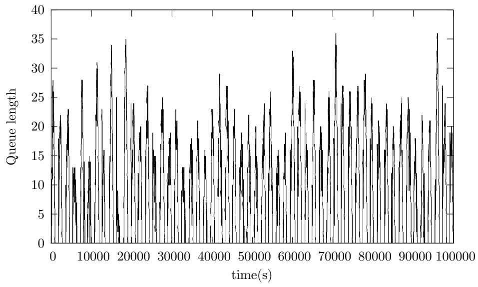


Figure 6: Buffer length of $R_8$ with $T_r = 1800$s.

processed. Consequently, the queue length increases. Afterward, the sendings stop, and $R_i$ keeps processing messages. The queue length decreases.

## 3.2 Congested case

Figure 7 represents the message arrivals in $R_8$ during 8000s, and Figure 8 the queue length of $R_8$ during $10^5$s, where $T_r = 514$s. One can observe that messages arrive continuously on router $R_8$. Then, $R_8$ is never idle and never empties its queue. Consequently the queue length permanently increases.

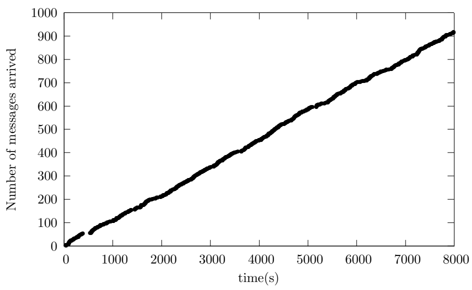


Figure 7: Message arrivals to $R_8$ with $T_r = 514$s.

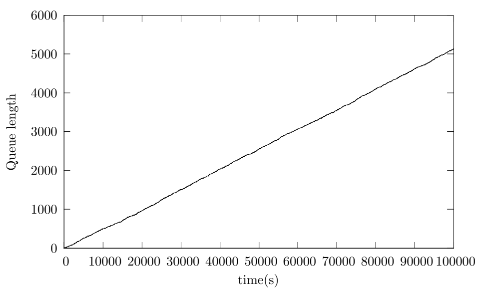


Figure 8: Buffer size of $R_8$ with $T_r = 514$s.

## 3.3 Limit case

Figure 9 represents the message arrivals in $R_8$ during 8000s, and Figure 10 shows the queue length of router $R_8$ during $10^5$s, where $T_r = 1000$s. This time, the sendings of a period are not merged with the sendings of the next period. Then, each period is long enough so that $R_8$ can process messages from its queue before the beginning of the next one. Figure 10 shows the fluctuations of the queue length that correspond to this. However the queue length is not empty at the end of each period. Consequently, the stability of this router is not ensured.

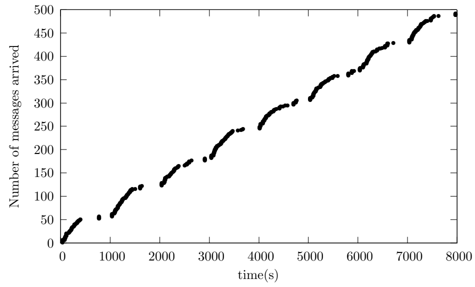


Figure 9: Message arrivals to $R_8$ with $T_r = 1000$s.

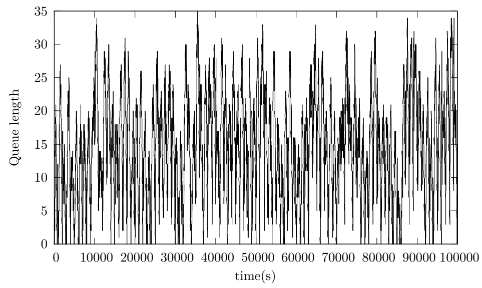


Figure 10: Buffer length of $R_8$ with $T_r = 1000$s.

## 3.4 Sufficient condition for congestion

Suppose being in the worst case where each router learns some new information from each router and let us now focus on the quantity of messages received during a period.

**Theorem 3.1.** *Let $n(j)$ be the number of messages received by a router $R_j$ during a flooding period $T_r$. Then*

$$n(j) \geqslant n(|\mathcal{V}(R_j)|).$$

*Proof.* Let us first focus on the case of networks with a tree topology. In this case, we show that the above inequality is in fact an equality. Two kinds of messages can be received: LSAs and Acks. Let us first count the number of messages received by router $R_j$ concerning the flooding from router $R_i$. Consider $R_i$ as the root of the tree, $R_j$ can receive $LSA_i$ from its father only: $R_j$ will receive one and only once $LSA_i$. Afterward $R_j$ sends $LSA_i$ to its children and will receive an *Ack* (as illustrated in Figure 11). As a consequence, the number of messages received for the flooding of $LSA_i$ is the number of neighbors of $R_j$. Consider the flooding of $LSA_j$. The router $R_j$ sends the LSA to its neighbors and will receive an *Ack* from them. Globally, $R_j$ will then receive exactly $n(|\mathcal{V}(R_j)|)$ messages.

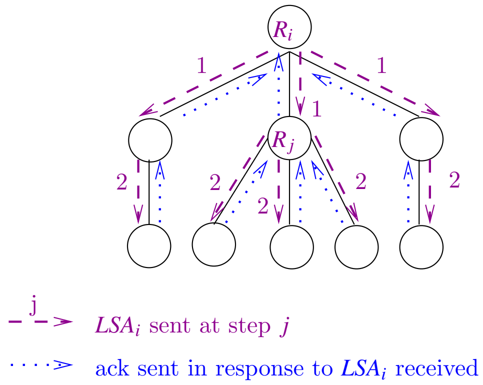

Figure 11: Flooding of $LSA_i$: LSA and ACKs transmissions in a tree topology.

For networks with a general topology, one can observe that the flooding of $LSA_i$ defines a spanning tree of the graph: $(R_j, R_k)$ is an edge of the spanning tree if $R_k$ first received $LSA_i$ from $R_j$. Then for the flooding of $LSA_i$, $R_j$ receives at least the messages it would received if the topology were the spanning tree, which gives the desired inequality. □

The number of messages processed by router $R_j$ during a flooding period is $1 + n(j)$: it processes the received messages plus $LSA_j$. Define $N(j)$ the number of messages processed during a flooding period by $R_j$, we have

$$N(j) = n(|\mathcal{V}(R_j)|) + 1.$$

If a router can not process every message of its buffer before the end of each period a congestion occurs. Also, given the minimal bound of Theorem 3.1 the congestion is ensured by the following threshold on $T_r$.

**Lemma 3.2.** *If $T_r < T_p N(j)$ then the queue length of $R_j$ tends to infinity.*

*Proof.* The proof is straightforward from Theorem 3.1. □

*Consider the tree topology network of Figure 11. Theorem 3.1 ensures that the number of messages received by $R_j$ ($|\mathcal{V}(R_j)| = 4$) is $N(j) = 9 \times 4 + 1 = 37$. Therefore, if $T_p$ is set to 15 s in the TPN, if $T_r < 15 \times 37 = 555$ s the network is congested. Simulation of the TPN, representing this topology, with $T_p = 15s$, $T_r = 554s$, $T_t = 30s$*

**Example 3.3** (Simulation of TPN by Renew software)**.**

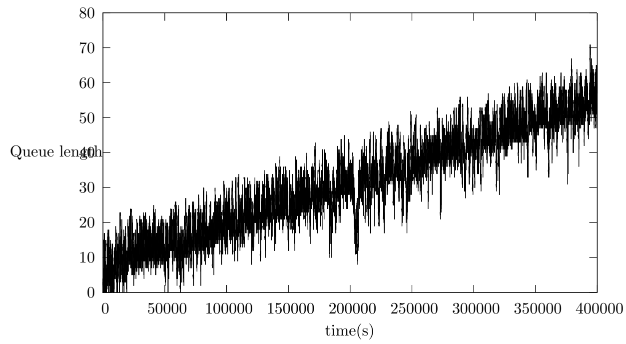

Figure 12: Queue length of $R_3$ with $T_r = 554$s of tree topology.

*has been made during $4.10^5$ s to illustrate this result. The evolution of the queue length of router $R_j$ is shown in Figure 12. The queue length of $R_j$ clearly increases during the simulation, showing that the network is congested. Finally, as the simulation has been made with the largest period length that ensures congestion, during each period, $R_j$ has enough time to process many messages from his queue. Consequently, one can observe that the queue length varies a lot.*

# 4 COMPUTING OPTIMUM INITIAL DELAYS

In Section 2.2, we emulated the flooding phenomenon of the OSFP protocol using Time Petri nets. The initial idea was to consider initial delays for each router as parameters. The question is then to infer constraints on these parameters that ensure a minimum size of the input buffers. Even if this kind of question can be theoretically solved using symbolic model-checking (Lime et al., 2009), the computation complexity is high. The state of the art of the current existing tools did not allow us to automatically produce such symbolic constraints.

In order to compute initial delays, we adopt the following method. We only take into account the message contributing to the flooding mechanism: when an LSA message concerning router $R_j$ is received at router $R_i$, it is forwarded only if it is received for the first time. Then, we will model neither the LSA messages that are not the first to be received at a node, nor the Acknowledgments.

## 4.1 Constraints modeling

Our goal is to perform the floodings as closed as possible while interacting as little as possible.

We say that two floodings do not interact if, for each router, the first LSA received from those two floodings in that router are not queued at the same time.

More formally, we consider a graph $G = (V, E)$, where $V = \{R_1, \ldots, R_n\}$ is the set of routers and $E \subseteq V \times V$ is the set of links between the routers. If $(R_i, R_j) \in E$, then $\tau_{i,j}$ denotes the transmission time between $R_i$ and $R_j$, and $\tau_{i,j} = \infty$ if $(R_i, R_j) \notin E$. The sojourn time of a message in $R_i$, between its reception and its forwarding, belongs to the interval $[\delta_i, \Delta_i[$. This time also holds for the source of messages.

Let us first compute the intervals of time $I_{i,j}$ when the first LSA originating from $R_i$ is received in $R_j$ if the flooding starts at time 0. If $i = j$, then $I_{i,i} = [0, 0]$, and otherwise, we have $I_{i,j} = [\alpha_{i,j}, \beta_{i,j}[$ where $\alpha_{i,j} = \min_{k \in \{1,\ldots,n\}} \alpha_{i,k} + \delta_k + \tau_{k,j}$ and $\beta_{i,j} = \min_{k \in \{1,\ldots,n\}} \beta_{i,k} + \Delta_k + \tau_{k,j}$.

The quantities $\alpha_{i,k} + \delta_k$ and $\beta_{i,k} + \Delta_k$ respectively represent the minimal and the maximal departure times from $R_k$.

For the computation of both $\alpha_{i,j}$ and $\beta_{i,j}$, we recognize the computation of a shortest path in a graph with respective edge lengths $(\delta_i + \tau_{i,j})$ and $(\Delta_i + \tau_{i,j})$. Let $\alpha = (\alpha_{i,j})$ and $\beta = (\beta_{i,j})$ the matrices of the shortest-paths. They can, for example, be computed using the Floyd-Warshall algorithm. Now, the messages originating from $R_i$ are present in $R_j$ during an interval of time included in $[\alpha_{i,j}, \beta_{i,j} + \Delta_j[ = [\alpha_{i,j}, \gamma_{i,j}[$. We denote by $D_{i,j}$ this interval and $D$ the matrix of these intervals.

**Example 4.1** (Sojourn times in the routers)**.**

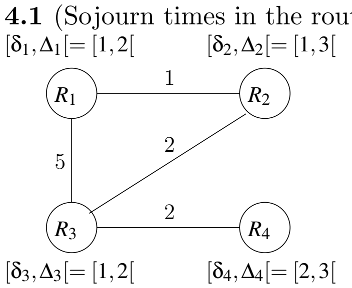


Figure 13: Example of a toy topology.

*Figure 13 represents a toy topology with 4 vertices. Matrix D is then:*

$$D = \begin{pmatrix} [0,2[ & [2,6[ & [5,9[ & [8,14[ \\ [2,6[ & [0,3[ & [3,7[ & [6,12[ \\ [5,9[ & [3,7[ & [0,2[ & [3,7[ \\ [9,14[ & [7,12[ & [4,7[ & [0,3[ \end{pmatrix}.$$

Now, if the flooding from server $R_i$ starts at time $d_i$, its first LSA received by $R_j$ is present in that server at most in the interval $d_i + D_{i,j} = [d_i + \alpha_{i,j}, d_i + \gamma_{i,j}]$.

Then, in order to have no interference between the floodings in router $R_j$, the family of intervals $(d_i + D_{i,j})_{i \in \{1,\ldots,n\}}$ must be two-by-two disjoint, and to have no interference at all, the following condition must hold:

$$\forall i, j, k \in \{1, \ldots, n\},\ i \neq k \Rightarrow d_i + D_{i,j} \cap d_k + D_{k,j} = \emptyset,$$

that is,

$$\forall i, j, k \in \{1, \ldots, n\},\ i \neq k \Rightarrow \begin{cases} d_i + \gamma_{i,j} \leq d_k + \alpha_{k,j} & \text{or} \\ d_k + \gamma_{k,j} \leq d_i + \alpha_{i,j}. \end{cases}$$

For each triple $(i, j, k)$, the two constraints above are exclusive: as $\gamma_{i,j} > \alpha_{i,j}$, if one holds, necessarily, the other one does not hold.

Now, if we don't consider the first flooding from each router only, we have to study the interferences between the first and second flooding from each router (if there is no interference between those two sets of flooding, then there will be no interference at all).

If the flooding period is $T$, then the constraints must then be transform in

$$\forall i, j, k \in \{1, \ldots, n\}, \quad \begin{array}{ll} d_i + \gamma_{i,j} \leq d_k + \alpha_{k,j} & \text{or} \\ d_k + \gamma_{k,j} \leq d_i + \alpha_{i,j} & \text{and} \\ d_k + \gamma_{k,j} \leq d_i + T + \alpha_{i,j} & \text{and} \\ d_i + \gamma_{i,j} \leq d_k + T + \alpha_{k,j} & \end{array} \tag{1}$$

The two cases are illustrated on Figure 14. Note that, depending on which of the two first constraint is satisfied, one of the two last inequalities is trivially satisfied.

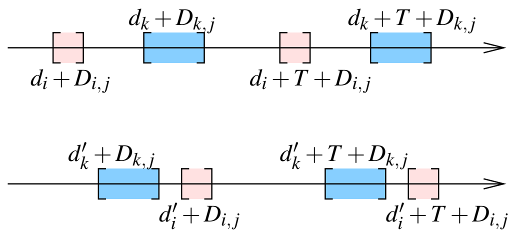


Figure 14: Different possibilities for the constraints. In the first case, $d_i + D_{i,j}$ is before $d_k + D_{k,j}$ and in the second case, $d'_k + D_{k,j}$ is before $d'_i + D_{i,j}$, but in both cases, $d_k + D_{k,j}$ is before $d_i + T + D_{i,j}$ and $d_i + D_{i,j}$ is before $d_k + T + D_{k,j}$

The problem we want to solve is then to find $(d_i)_{i \in \{1,\ldots,n\}}$ such that all the constraints are satisfied and $T$ is minimized.

**Theorem 4.2.** *Given $(\alpha_{i,j})_{i,j \in \{1,\ldots,n\}}$, $(\gamma_{i,j})_{i,j \in \{1,\ldots,n\}}$ and $T$, the problem of finding $(d_i)_{i \in \{1,\ldots,n\}}$ satisfying the constraints of Equation (1) is NP-complete.*

*Proof.* The problem is trivially in NP as for any assignment of $(d_i)$ and period $T$, it is possible to check in polynomial time if the constraints are satisfied (there are $O(n^3)$ constraints).

Now, to show that the problem is NP-hard, we reduce the salesman problem with triangular inequality to that problem.

Suppose a complete weighted graph, with positive weights of the edges $w(u,v)$, satisfying the triangular inequality: for all vertices $u,v,x$, $w(u,x)+w(x,v) \le w(u,v)$. Set $\gamma_{i,j} = \max_{k\in\{1,\dots,n\}} w(k,i)$ and $\alpha_{i,j} = \gamma_{k,j} - w(i,k)$.

This assignment of the variables is made in such a way that if for some $j$, $d_i - d_k \ge \gamma_{k,j} - \alpha_{i,j}$, then this holds for all $j$, as $\gamma_{k,j} - \alpha_{i,j} = w_{i,k}$.

Now, let $(d_i)$ and $T$ be a solution of our problem. There is a Hamiltonian cycle of weight $W \le T$ in the graph: suppose, without loss of generality that $d_1 \le d_2 \le \dots \le d_n$.

Then, $w(1,2)+w(2,3)+\dots+w(n,1) \le (d_2-d_1)+(d_3-d_2)+\dots+(d_1-d_n+T) = T$.

Conversely, suppose that there is a Hamiltonian cycle of weight $W$, corresponding without loss of generality to the cycle $1,2,\dots,n$. Set $d_1 = 0$ and $d_i = d_{i-1} + w(i-1,i)$. We have for all $i,j$ $d_i$ every constraint is satisfied and $T = W$ is a possible period: if $k > i$, $d_k - d_i = w(i,i+1)+\dots+w(k-1,k) \ge w(i,k)$. Moreover, $(d_i+W)-d_k = w(k,k+1)+\dots+w(n,i)+\dots+w(i-1,i) \ge w(k,i)$.

Hence, we have a Hamiltonian path of length at most $T$ if and only if we can find a solution to our problem with period at most $T$: the problem is NP-hard. □

## 4.2 Exact solution with linear programming

This problem can be solved with a linear program using both integer and non-integer variables. The trick is to encode the constraints

$$\begin{aligned} d_i + \gamma_{i,k} &\le d_k + \alpha_{k,j} \quad \text{or} \\ d_k + \gamma_{k,j} &\le d_i + \alpha_{i,j} \end{aligned}$$

into a linear program, and this is why we introduce integer variables.

First, this set of constraints can be rewritten in

$$d_k - d_i \ge b_{i,k,j} \text{ or } d_i - d_k \ge b_{k,i,j}$$

with $b_{i,k,j} = \gamma_{i,j} - \alpha_{k,j}$. Set $B = \max_{i,j,k} b_{i,k,j}$.

**Lemma 4.3.** *There is a solution of this problem where for all $i \in \{1,\dots,n\}$, $d_i \in [0,nB]$.*

*Proof.* The assignment $d_i = (i-1)B$ is a solution of the problem. Indeed, $\forall i < k$, $\forall j \in \{1,\dots,n\}$, $d_k - d_i = (k-i)B \ge B \ge b_{i,k,j}$. Moreover, $\forall i,k,j$, $d_k - d_i = (n-k+i)B \ge B \ge b_{i,k,j}$. □

**Lemma 4.4.** *The following sets of constraints are equivalent.*

*(i)* $d_i, d_k \in [0,nB]$ *and* $(d_k - d_i \ge b_{i,k,j}$ *or* $d_i - d_k \ge b_{k,i,j})$

*(ii)* $d_i, d_k \in [0,nB]$, $q \in \{0,1\}$ *and* $d_k - d_i + (1-q)nB \ge b_{i,k,j}$ *and* $d_i - d_k + qnB \ge b_{k,i,j}$.

*Proof.* Suppose that the constraints $(i)$ are satisfied. Either $d_k - d_i \ge b_{i,j,k}$ and the constraints in $(ii)$ with $q = 1$ are satisfied (we have the two constraints $d_k - d_i \ge b_{i,j,k}$ and $d_i - d_k + nB \ge nB \ge b_{k,i,j}$); or $d_i - d_k > b_{k,i,j}$ and similarly, the constraints in $(ii)$ with $q = 0$ are satisfied.

Suppose now that the constraints $(ii)$ are satisfied. If $q = 1$, then, trivially, $d_k - d_i \ge b_{i,j,k}$ and if $q = 0$, then $d_i - d_k \ge b_{k,i,j}$. □

Consequently, the linear program is

Minimize $T$ under the constraints

$$\forall i,j,k \in \{1,\dots,n\},\ i \ne k,$$
$$0 \le d_i \le nB$$
$$\begin{cases} q_{i,k,j} \in \{0,1\} \\ d_k - d_i + (1-q_{i,j,k})nB \ge b_{i,k,j} \\ d_i - d_k + q_{i,j,k}nB \ge b_{k,i,j} \\ d_k - d_i \le T - \max_{j\in\mathbb{N}_n} b_{k,i,j} \end{cases}$$

**Example 4.5.** *The toy example above gives $T = 28$, with $d_1 = 0$, $d_2 = 21$, $d_3 = 14$ and $d_4 = 5$.*

Computing this exact solution is possible but has two drawbacks. First, as the problem is NP-complete, computing the initial delays in larger networks may be untractable. Second, this solution does not exhibit monotony properties. For example, if the linear program lead to a period $T$ and the target period is $T' > T$, it might be better to stretch the values $d_i - d_k$ to $(d_i - d_k)T'/T$. It is unfortunately not ensured with the solution found. In the next paragraph, we show how to compute a solution complying with this additional constraint.

## 4.3 Heuristic using a greedy algorithm

To simplify the problem we only use strongest constraints: with $c_{i,k} = \max_{k\in\mathbb{N}_n} b_{i,k,j}$,

$$c_{i,k} \le d_k - d_i \le T - c_{k,i} \quad \text{or} \quad c_{k,i} \le d_i - d_k \le T - c_{i,k}. \tag{2}$$

**Lemma 4.6.** *If $(d_i)_{i\in\{1,\dots,n\}}$ is a solution to the constraints of Eq. (2) with a period $T$, then for $T' > T$, $(\frac{T'}{T} d_i)$ is a solution for the same constraints with period $T'$.*

*Proof.* If $c_{i,k} \le d_k - d_i \le T - c_{k,i}$, then as $\frac{T'}{T} \ge 1$, $\frac{T'}{T}(d_k - d_i) \ge d_k - d_i \ge c_{i,k}$. Second, $\frac{T'}{T}(d_k - d_i) = \frac{T'}{T}(T - c_{k,i}) = T' - \frac{T'}{T} c_{k,i} \le T' - c_{i,k}$. □

Solving these constraints is still a NP-complete problem. In fact the proof of Theorem 4.2 is valid in this case.

Now, in order to assign the values, we can use the greedy algorithm presented in Algorithm 1. At each step, the algorithm assigns one initial delay, that is chosen to be the smallest as possible, given the initial delays already assigned, while satisfying the constraints set by them.

**Algorithm 1**: Initial delays computation.

**Data**: $c_{i,j}$.
**Result**: $d_1, \dots, d_n$, $T$.

1 **begin**
2 $\quad D \leftarrow \emptyset$ ;
3 $\quad S \leftarrow \{1, \dots, n\}$;
4 $\quad$ **foreach** $i \in S$ **do** $d_i \leftarrow 0$;
5 $\quad$ **while** $S \neq \emptyset$ **do**
6 $\quad\quad s \leftarrow \mathrm{Argmin}_{i \in S} d_i$;
7 $\quad\quad S \leftarrow S \setminus \{s\}$;
8 $\quad\quad$ **foreach** $i \in S$ **do**
$\quad\quad\quad d_i \leftarrow \max(d_i, d_s + c_{s,i})$;
9 $\quad\quad$ **foreach** $i \in D$ **do**
$\quad\quad\quad T \leftarrow \max(T, d_s - d_i + c_{s,i})$;
10 $\quad\quad D \leftarrow D \cup \{s\}$;
11 **end**

**Lemma 4.7.** *At each step of the algorithm, the constraints (2) such that $i, k \in D$ are satisfied.*

*Proof.* We show the result by induction. When $D = \emptyset$ or $|D| = 1$, then this is obviously true as no constraints are involved. Suppose this is true for $D$ and let $s$ the next element that is added to $D$ in the algorithm. From line 8, we know that $d_s \geq \max_{i \in D} d_i + c_{i,s}$. Then, for all $i \in D$, $d_s - d_i \geq c_{i,s}$. Now, from line 9, for all $i \in D$, $T \geq d_s - d_i + c_{s,i}$, so $d_s - d_i \leq T - c_{s,i}$. So, the constraints involving $s$ are satisfied. Now, if the constraints between $i$ and $j$, $i, j \in D$ are satisfied at one step of the algorithm, they will remain satisfied during the following steps, as $T$ can only increase. $\square$

**Example 4.8** (Application of Algorithm 1)**.** *With our toy example, we have*

$$C = (c_{i,j}) = \begin{pmatrix} 0 & 8 & 11 & 14 \\ 6 & 0 & 9 & 12 \\ 9 & 7 & 0 & 7 \\ 14 & 12 & 9 & 0 \end{pmatrix}.$$

*If* 1 *is chosen first ($d_i = 0\ \forall i \in \{1,2,3,4\}$), the values are updates to $d_1 = 0$, $d_2 = \max(0, d_1 + c_{1,2}) = 8$, $d_3 = 11$ and $d_4 = 14$; $T = 0$. Then,* 2 *is chosen and we get $d_3 = \max(d_3, d_2 + c_{2,3}) = 17$ and $d_4 = 20$; $T = \max(T, d_2 - d_1 + c_{2,1}) = 14$. Finally, we have $d_1 = 0$, $d_2 = 8$, $d_3 = 17$, $d_4 = 24$ and $T = 38$.*

Note that this problem could also have been solved using a linear program (with integer variables), by replacing the variables $q_{i,k,j}$ in the linear program of the previous paragraph by $q_{i,k}$: forgetting the parameter $j$, exactly leads to the same constraints of Equation (2). In this case, we find $T = 36$, with $d_1 = 0$, $d_2 = 30$, $d_3 = 11$ and $d_4 = 18$. Our heuristic is near this optimal.

In the next lemma, we assume that our target period is $T' < T$, that is, we are not able to find a solution so that there is at most one message in the queues of the routers. We assume here that the sojourn time of a message does not depend on the queue length.

**Lemma 4.9.** *Let $(d_i)$ be a solution for the initial delays with period $T$. The same assignment with period $T' < T$ ensures that in each router, there are never more than $\lceil \frac{T}{T'} \rceil$ messages simultaneously.*

*Proof.* Set $\lceil \frac{T}{T'} \rceil = q$. We number the messages: $m_i^j$ is the $j$-th message originating from router $i$. For $\ell \in \{0, \dots, q-1\}$, in each server, simultaneously, there cannot be several messages among $(m_i^{kq+\ell})_{k \in \mathbb{N}, i \in \mathbb{N}_n}$, because $qT' \geq T$. As a consequence, there cannot be more than $q$ messages in a router. $\square$

## 4.4 Simulation results with initial delays

In this section, we present simulations of the TPN modeling the German telecommunication network with initial delays defined by Algorithm 1 in the stable case ($T_r = 1800$s).

We first need to define the transmission and sojourn times used by the algorithm:

- the transmission time has already been defined to $\tau_{ij} = T_t = 30s$, for all the links of the network;
- for each router $R_i$, the sojourn time is at least equal to the processing time $\delta_i = T_p = 15s$, the time to process the message where the queue is empty. The maximum sojourn time is extracted from the simulation of the TPN of Section 2 (with no initial delays). During the simulation, the maximum queue length is $Q_i$ in router $R_i$. Then we take $\Delta_i = Q_i T_p$.

  Note that doing this enables to take into account all the messages from the LSA flooding mechanism, and not only the first LSA message in each router.

The maximal queue length of each router is extracted from a simulation of the TPN during approximately 3.5 days ($3.10^5$s). Here is the list of each maximal queue length: $Q =$

$(7,8,13,2,2,17,8,37,4,5,13,2,2,3,13,6,2)$. Then, Algorithm 1 returns the following initial delays:

$$d = (0,105,1200,810,75,255,420,1335,1035, 1080,1155,1530,630,330,780,330,1680).$$

Furthermore, Algorithm 1 computes $T_{rMax} = 16695$s.

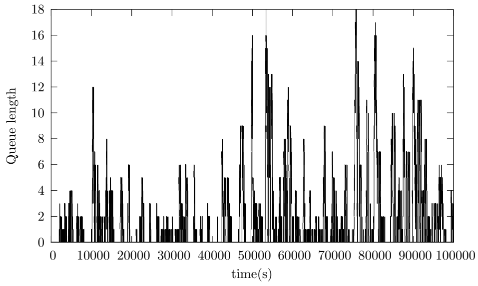


Figure 15: Buffer length of $R_8$ with $T_r = 1800$s and initial delays.

Figure 15 represents the result of the TPN simulation with initial delays listed above when $T_r = 1800$s. The maximum queue length for router $R_8$ is now $Max_8 = 25$, which gives a significant improvement: it was $Max_8 = 37$ without the computation of initial delays. Moreover, the queue length is most of the time below 10.

# 5 CONCLUSION

This article presents a method usable for the OSPF protocol and cyclic protocols that use delay parameters. This method aims at increasing the reactivity of the network to topology changes, and at minimizing the queue length of routers. Algorithm 1 provides an efficient way to spread messages over the whole period. Furthermore, it shows to be a good tool to reduce queue lengths.

# REFERENCES


Basu, A. and Riecke, J. (2001). Stability issues in ospf routing. In *Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications*, SIGCOMM '01, pages 225–236, New York, NY, USA. ACM.

Francois, P., Filsfils, C., Evans, J., and Bonaventure, O. (2005). Achieving sub-second igp convergence in large ip networks. *SIGCOMM Comput. Commun. Rev.*, 35(3):35–44.

Ishiguro, K. (2012). Quagga, a routing software package for tcp/ip networks, http://www.nongnu.org/quagga/.

Jard, C. and Roux, O. H. (2010). *Communicating Embedded Systems, Sofware and Design, Formal Methods*. ISTE and Wiley.

Katz, D., Kompella, K., and Yeung, D. (2003). Traffic Engineering (TE) Extensions to OSPF Version 2. Updated by RFC 4203.

Kummer, O., Wienberg, F., Duvigneau, M., Kohler, M., Moldt, D., and Rolke, H. (2003). Renew the reference net workshop. In *mi*.

Lime, D., Roux, O. H., Seidner, C., and Traonouez, L.-M. (2009). Romeo: A parametric model-checker for petri nets with stopwatches. In Kowalewski, S. and Philippou, A., editors, *TACAS*, volume 5505 of *Lecture Notes in Computer Science*, pages 54–57, York, United Kingdom. Springer.

Moy, J. (1998). RFC 2328 OSPF v2. Technical report.


| Notation | Full name |
|---|---|
| $G = (V, E)$ | directed graph representing the network |
| $n$ | number of routers |
| $R_i$ | $i^{th}$ router in the network |
| $\mathcal{V}(R_i)$ | set of neighbors of $R_i$ |
| $\lvert\mathcal{V}(R_i)\rvert$ | cardinality of $\mathcal{V}(R_i)$ |
| $d_i$ | initial delay of $R_i$ |
| $b_i$ | number of retransmission of an LSA received from a neighbor of $R_i$ |
| $LSA_i$ | link state advertisement message of $R_i$ |
| Ack | acknowledgment message |
| $T_r$ (or $T$) | period length of the LSA flooding process |
| $T_p$ | processing time of messages |
| $T_t$ | time to send a message |
| (P,T,B,F,$M_0$,$\varphi$) | a Time Petri Net (TPN) |
| $Start_i$ | initial place of TPN to create $LSA_i$s |
| $LSAsend_{i\to j}$ | place to send $LSA_i$ to $R_j$ |
| $ACKsend_{i\to j}$ | place to send an Ack from $R_i$ to $R_j$ |
| $LSArec_{j\to i}$ | place to receive $LSA_j$ in $R_i$ |
| $ACKrec_{j\to i}$ | place to receive an Ack from $R_j$ in $R_i$ |
| $Processor$ | place to guaranty that one message is processed at a time |
| $bound$ | place to bound the number of retransmission from a neighbor |
| $Retransmission$ | place to retransmit a received LSA |
| $Destruction$ | place to destroy a received LSA |
| $n(j)$ (resp. $N(j)$) | number of messages received (resp. processed) by $R_j$ during $T_r$ |
| $\tau_{i,j}$ | transmission time between $R_i$ and $R_j$ |
| $[\delta_i, \Delta_i[$ | sojourn time of a message in $R_i$ |
| $I_{i,j} = [\alpha_{i,j}, \beta_{i,j}[$ | time of first $LSA_i$ received in $R_j$ |
| $\alpha = (\alpha_{i,j})$ | matrix of values $\alpha_{i,j}$ |
| $\beta = (\beta_{i,j})$ | matrix of values $\beta_{i,j}$ |
| $D = (D_{i,j})$ | $D_{i,j} = [\alpha_{i,j}, \gamma_{i,j}[$ with $\gamma_{i,j} = \beta_{i,j} + \Delta_j$ |
| $Q = (Q_i)$ | maximal queue length of $R_i$ |
| $b_{i,k,j}$ and $B$ | $b_{i,k,j} = \gamma_{i,j} - \alpha_{k,j}$ and $B = \max_{i,k,j} b_{i,k,j}$ |
| $C = (c_{i,k})$ | $c_{i,k} = \max_{k\in\mathbb{N}_n} b_{i,k,j}$ |

Table 1: List of main notations.