Large-Scale Distributed Systems
Project 3: Firefly-inspired synchronization

Laurent Hayez

December 14, 2015

Table of contents

1 Introduction

In nature, fireflies produce light in order to attract mates or prey. One interesting feature of these beetles is that when they emit light in group, at some point, they do it in a synchronized manner, just by looking at when their neighbours emit light. This feature is interesting in large-scale distributed systems, as synchronization might be required, but one node does not know every other nodes.

The objective is thus to inspire ourselves from fireflies to try to synchronize nodes in a decentralized manner. At first we will detail the protocol skeleton and explain how the core of the protocols will work. Then we will look at a two models called "phase-advance" and "phase-delay" and briefly analyze them. The main and final part will be the "adaptive Ermentrout model" which is more representative of the reality. We will explain the implementation specifities and analyze this model in different situations.

2 Protocol skeleton

According to the paper "Firefly-inspired Heartbeat Synchronization in Overlay Networks", the skeleton for the different algorithms is composed of two main functions, namely activeThread and passiveThread. We provide the pseudo code for the implementation in Algorithm 2.1.

In the different protocols, a node is an oscillator characterized by its phase φ and the cycle length Δ. We define φ as a sawtooth function with domain $[0, 1]$ such that $\frac{d\varphi}{dt} = \frac{1}{\Delta}$. This is represented in Figure 1.

When φ reaches 1, the node will send a flash to a set of neighbour nodes, and φ is reset to 0. The cycle length, depending on the model chosen, can be the same or different for all nodes. The function updatePhi will differ in our implementations, but we will come back on this when needed.

The core of the synchronization protocol is the function processFlash, i.e. what a node does when it receives a flash. This function is responsible of how φ is updated. Depending on the implementation, φ will be updated, or Δ will be updated and will affect φ.

The underlying overlay network protocol used for a node to know its neighbours is the Peer Sampling Service (PSS).

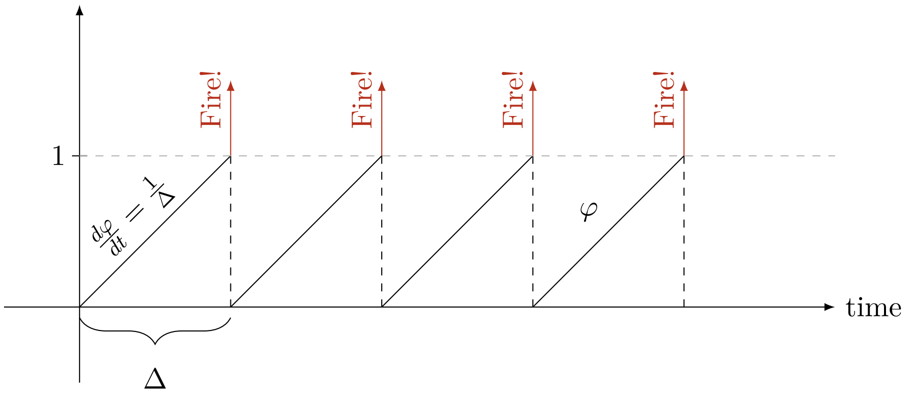

Figure 1: Representation of φ and its relation with Δ

We briefly discuss how the protocol skeleton works before we move to the implementation of the phase-advance and phase-delay protocols.

For every implementations, we start by initializing the PSS at each node, and wait two minutes to have a consistent view. Then we start a thread activeThread and a periodic thread updatePhi. As we defined φ to be such that $\frac{d\varphi}{dt} = \frac{1}{\Delta}$, we have $\varphi = \int \varphi dt$, which explains how we update φ. If $\varphi \geq 1$, we fire the event "Flash!" that triggers activeThread to send that it emitted a flash to the nodes in its view. In updatePhi, we then set φ to 0, as in Figure 1 and call activeThread to wait for a new flash.

3 Phase-advance and phase-delay protocols

3.1 Implementation

The fundamental difference with the protocol skeleton here is the implementation of the processFlash function. As the two protocols are almost the same, we added the boolean phase_advance to signify the usage of the phase-advance or phase-delay protocol. The pseudo-code of the implementation is given in Algorithm 3.1.

3.2 Analysis of the protocols

We tried our implementation with 100 nodes per experiment and $\Delta = 1, 2$ and 5. For the PSS, we used a view of 10 peers with a period of 20 seconds for each update. We used a random selection for the peer selection, and we used a swapping parameter of 3 and a healing parameter of 2 for the view exchange policy.

We start by analyzing the convergence of the phase-advance protocols.

Algorithm 2.1 Skeleton for the Firefly algorithms

```
Variables:
φ                                                          ▷ phase
Δ                                                          ▷ cycle length
```

$$\text{update\_phi\_period} = \begin{cases} \frac{\Delta}{10} & \text{if } \Delta < 1 \\ \frac{1}{10\Delta} & \text{if } \Delta \geq 1 \end{cases}$$

function SENDFLASH()
  $P \leftarrow$ view from PSS
  send flash to all peers in P
end function

function PROCESSFLASH()
  depends on the implementation
end function

function UPDATEPHI()
  if $\varphi < 1$ **then**
    $\varphi \leftarrow \varphi + \frac{1}{\Delta} \cdot$update_phi_period
  else
    fire event "Flash!"
    $\varphi \leftarrow 0$
    start new thread ACTIVETHREAD
  end if
end function

function ACTIVETHREAD()
  wait for the event "Flash!"
  sendFlash()
end function

function PASSIVETHREAD()
  receive flash
  processFlash()
end function

Algorithm 3.1 processFlash for the phase-advance and phase-delay protocols

Variables:
phase_advance $\leftarrow$ true or false

function PROCESSFLASH()
  if phase_advance **then**
    $\varphi \leftarrow 1$
  else
    $\varphi \leftarrow 0$
  end if
end function

4 Conclusion