# As-Far-As-Possible LE Algorithm Implementation

This document is a protocol of an assignment from subject *NI-DSV* at FIT CTU in Prague

Jan Troják
*Faculty of Information Technology*
*Czech Technical University in Prague*
Prague, Czechia
trojaj12@fit.cvut.cz

## I. Introduction

Decentralized systems are systems where no parts rely on one central part. Those systems can be used in computer science as part of solutions with specific requirements like high availability. Even though those structures are meant to work without any chosen authority, there is often a need to agree on one common knowledge. One such common agreement, considered one of the most known, is the leader election problem.

### A. Paper overview

This paper is about one possible Implementation of leader election: the *As-Far-As Possible* (AFA) leader election algorithm.

In the first part of the paper, the theory of the leader election problem and *As-Far-As Possible* will be introduced. The flow section will describe a possible algorithm implementation using *gRPC*. In the last part of the paper, the results of algorithm execution on various topologies will be covered.

### B. Common language and restrictions

In the document, distributed systems will be discussed. As distributed systems, we will consider any formation of linked computation nodes, forming a connected graph (a path exists between each node and any other node). For communication purposes, we will assume three axioms:

- **MConnectivity (CN)**
  The communication topology G is strongly
- **Bidirectional links (BL)**
  Communication between any two nodes that are directly connected can be performed in both directions
- **Total reliability (TR)**
  Neither any failures will occur during nor any failures occurred before the algorithm execution. *[1]*

All nodes in the distributed graph will be considered trusted and execute the same code as others.

## II. Leader election problem

Choosing a single node in the distributed system recognized by all others is known as a leader election (LE) problem. This problem is not solvable in the conditions specified in *Section I.B*. In general, finding an asymmetric property in a possible symmetric structure is impossible. That is why a new property needs to be defined to grant the possibility (correctness and finite termination) of an algorithm solving the LE problem. *[2]*

- **Unique identifier (UI):**
  For each node in the graph, there is a unique identifier that the node can use

This property breaks the possible symmetry in the graph, making the LE problem solvable.

In the flowing text, let's consider that such property is quarantined and identifiers are comparable.

## III. As-far-as possible

*As-Far-AS possible (AFA)* is one of the algorithms that can solve the LE problem on a circle graph. AFA is a straightforward solution that locates the nodes with the lowest identifier and promotes the node to the leader. After the election, the algorithm sends the final message to all nodes (broadcast) with information that the leader was elected and specifies the leader's identification. At the end of the AFA, all nodes know who the leader is, and the leader node knows it has been elected as a leader.

The AFA algorithm is based on the *All-Way-Around (AWE)* LE algorithm, which broadcasts each node's ID. AFA improves the AWE algorithm so it has lower message complexity. *[2]*

### A. Algorithm

Each node has a UID (unique identification), which is assumed to be a natural number. AFA uses two types of messages:

- ElectionMessage (EM)
- TerminationMessage (TM)

Each node can be in one of the flowing states:

- Candidate (C),
- Defeated (D)
- Elected (E).

Each node starts in a sleeping stage with knowledge of its neighbors and UID. When a node wakes up (starts the algorithm), it sends an EM message to one of its neighbors with its UID and sets its state to C. If a node is wakened up by receiving EM, it sets its state to C, and reads the ID received in the EM message. If the received message is

```
service AFA {
  rpc Ping(PingRequest) returns (Empty);
  rpc SendElectionMessage(ElectionMessage) returns (Empty);
  rpc SendTerminationMessage(TerminationMessage) returns (Empty);
}

message PingRequest { int32 source_id = 1; int32 destination_id = 2; }
message ElectionMessage { int32 source_id = 1; }
message TerminationMessage { int32 leader_id = 1; }
```

Code 1: Protocol Specification

smaller than the most minor known ID (initially its own UID), it updates the most minor known ID and sends it to the next node. The nodes switch to D states if the update was performed.

Three possibilities can occur when a node receives EM and is already in C state.

1) Received ID is greater than most minor know ID in that case, EM is discarded
2) Received ID is more minor than smallest known ID node updates its most minor known ID and sends the EM message with the most minor known ID to the neighbour
3) Received ID is the same as the smaller known ID node can set its state to L and become the leader

After a finite amount of time, one of the nodes (with the most minor UID) will be in a L, and it will know about it. In that phase, it can send a TM broadcast (message around the circle) that it has been elected as a leader. Once the broadcast is done, all nodes will agree on one leader.

AFA has message complexity:

$$MC(\text{AFA}, R(C)) \leq n + \sum_{i=1}^{n} i = \frac{n(n+3)}{2}$$

*[2]*

## IV. Implementation

The AFA leader election algorithm was implemented in a distributed environment using *gRPC* for communication. The language used to implement the algorithm was Golang. This section will describe key components of the algorithm, including the protocol used and other implementation-specific decisions.

### A. Protocol

For direct communication between nodes, the *gRPC* protocol was used. Its proto-buffer specification can be seen at *Code 1*. *Ping* message is used only for testing topology - it is not used in AFA. *ElectionMessages* and *TerminationMessage* is used as specified in *Section III.A*.

### B. Initial starting phase

Since AFA expects correct ring topology, there is a preparation phase that checks whether the needed topology is available or not. Once a node is started, it tries to send a *Ping* message to itself. This means it sends a message around the whole topology. Once node receives *Ping* message addressed to it's self, node can start the AFA algorithm, since topology is reddy. To avoid any errors when sending messages, while topology is not completely started yet, *Ping* messages use grpc.WaitForReady(true), a parameter that Google's grpc lib provides.

### C. Algorithm implementaion

The whole algorithm is implemented in a Server structure, which implements all the *gRPC* calls. The state, configuration, and connections are held in a single structure (*Code 2*).

```
type Server struct {
  pb.UnimplementedAFA
  Node node.Node
  rightNode node.Node
  RightNodeClient pb.AFAClient
  connection *grpc.ClientConn

  State LEState
  lowestKnownId int
  end chan boolcolbreak
  once sync.Once
}
```

Code 2: Protocol Specification

### D. Termination of the program

Once the leader is selected, the elected node sends TM. This message announces the leader. It's forwarded along the ring topology and also announces (via the closing channel) that the caller of the AFA algorithm is done. If the primary process receives a closing message, it shuts down the server and returns an error code based on the role in the topology (leader or not).

| Size | Min | Max | Median | Average |
|---|---|---|---|---|
| 10 | 15 | 39 | 25 | 24 |
| 11 | 18 | 41 | 29 | 28 |
| 12 | 22 | 47 | 32 | 32 |
| 13 | 21 | 44 | 33 | 32 |
| 14 | 22 | 51 | 37 | 36 |
| 15 | 24 | 54 | 38 | 37 |
| 16 | 25 | 65 | 42 | 41 |
| 17 | 27 | 64 | 44 | 43 |
| 18 | 30 | 68 | 47 | 47 |
| 19 | 28 | 74 | 48 | 48 |
| 20 | 28 | 71 | 53 | 52 |
| 21 | 31 | 82 | 52 | 52 |
| 22 | 32 | 87 | 58 | 57 |
| 23 | 39 | 89 | 57 | 57 |
| 24 | 35 | 101 | 59 | 60 |
| 25 | 33 | 98 | 63 | 63 |
| 26 | 27 | 102 | 67 | 67 |
| 27 | 34 | 101 | 66 | 66 |
| 28 | 39 | 108 | 71 | 71 |
| 29 | 47 | 106 | 72 | 72 |
| 30 | 48 | 101 | 70 | 71 |
| 31 | 39 | 145 | 74 | 75 |
| 32 | 43 | 107 | 75 | 74 |
| 33 | 50 | 124 | 81 | 81 |
| 34 | 48 | 115 | 78 | 77 |
| 35 | 55 | 127 | 82 | 83 |
| 36 | 54 | 123 | 84 | 84 |
| 37 | 54 | 127 | 85 | 84 |
| 38 | 52 | 138 | 87 | 88 |
| 39 | 53 | 127 | 88 | 88 |
| 40 | 54 | 148 | 93 | 93 |
| 41 | 47 | 136 | 93 | 92 |
| 42 | 63 | 155 | 99 | 99 |
| 43 | 64 | 154 | 99 | 98 |
| 44 | 63 | 140 | 100 | 100 |
| 45 | 67 | 148 | 104 | 104 |
| 46 | 64 | 154 | 108 | 107 |
| 47 | 65 | 170 | 109 | 108 |
| 48 | 65 | 161 | 110 | 109 |
| 49 | 68 | 185 | 113 | 114 |

Table 1: Aggregation of number of EM messages used for given size of topology

### E. *Threading and synchronization*

There are also threads among the main and server threads that check if the node was elected as a leader and other operational tasks. The concurrency threading model is implemented using native go routines. Synchronization is done using native go *channels* and native *sync* library.

## V. Tests

An implementation that is described was tested on 4096 possible topologies. Topologies were used as provided by FIT CTU[1]. All tests passed. In the table *Table 1*, you can see how many EM were sent for the given topology size.

[1] https://courses.fit.cvut.cz/NI-DSV/tutorials/files/topologies.txt

## List of Abbreviations

| | |
|---|---|
| RPC | Remote procedure call |
| gRPC | Google's remote procedure call |
| AFA | As-Far-As possible algorithm |
| CN | MConnectivity |
| BL | Bidirectional links |
| TR | Total reliability |
| LE | Leader election |
| UI | Unique identifier |
| AWE | All-Way-Around algorithm |
| UID | Unique identificator |
| ID | Identificator |
| EM | ElectionMessage |
| TM | TerminationMessage |
| C | Candidate state |
| D | Defeated state |
| E | Elected state |
| FIT | Faculty of Information Technology |
| CTU | Czech Technical University in Prague], |

## References


[1] prof. Ing. Pavel Tvrdík CSc., *NI-DSV Distribuované systémy a výpočty*, Úvod do teorie distribuovaných systémů a algoritmů. České vysoké učení technické v Praze Fakulta informačních technologií: Katedra počítačových systémů, 2024.

[2] prof. Ing. Pavel Tvrdík CSc., *NI-DSV Distribuované systémy a výpočty*, Volba vůdce 1. České vysoké učení technické v Praze Fakulta informačních technologií: Katedra počítačových systémů, 2024.