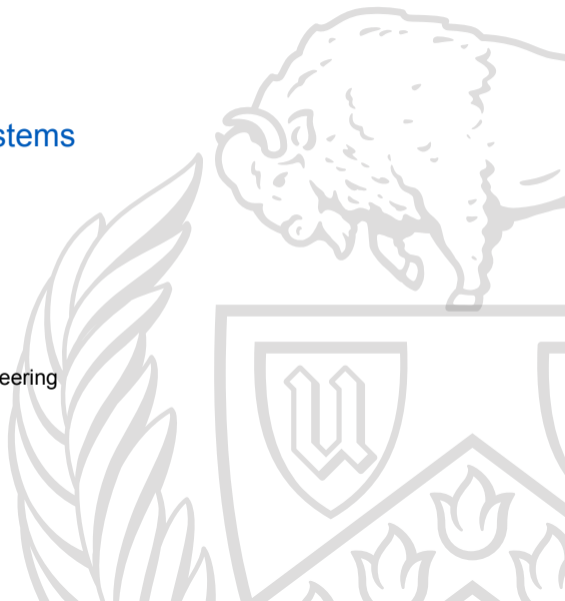


Leader Election

CSE 486/586: Distributed Systems

Ethan Blanton

Department of Computer Science and Engineering
University at Buffalo



Centralized Decisions

It can often **simplify protocols** to make centralized decisions.

For example:

- **Total ordering** with a central scheduler
- Assignment of **globally unique names**

There are significant disadvantages to centralizing:

- **Single points of failure**
- Centralized **trust**
- **Latency**¹

¹This is absolutely a tradeoff!

Leader Elections

Leader elections can address the single point of failure.

They select a centralized decision-maker at run time.

They are most often used:

- When a distributed protocol is bootstrapping
- After a leader has failed

In the latter case they require a failure detector.

Election Properties

Leader elections can be called at different times:

- When **any process wants to**
- When **the current leader wants to**
- When **any process detects leader failure**
- When **some set of processes** detects leader failure
- At a **predetermined time** or protocol state
- ...

Which policy is used depends on the application!

The Bully Algorithm

The **Bully Algorithm** [4] was one of the first to be described.

It takes its name from the property that the **biggest process** wins.

It assumes:

- Every process has a unique ID
- Process IDs form a **total ordering**
- Communication is reliable
- Messages are delivered **in bounded time**

Any process can start an election **at any time**.

States

Processes in the Bully Algorithm have **two states**:

- Normal
- Election
- Waiting

Processes in the normal state are **doing what they do**.

Processes in the **election state** are electing a leader.

Processes in the **waiting state** are awaiting results.

The **safety property** must hold:

If p_i and p_j are both in the normal state, they agree on the current leader.

Starting the Election

Suppose that process p_i wishes to start an election.
(Perhaps it thinks the current leader has failed?)

First it moves to the **election** state.

It **proposes itself** as leader to **all processes with larger IDs**.

If any process of larger ID responds, it waits for a new leader.

If no process of larger ID responds, **it declares itself leader**.

Participating in Election

Suppose that some process p_j hears an election proposal from p_i .

This means that p_j 's ID is larger than p_i 's ID.
(To whom did p_i send proposals?)

It sends a message to p_i stating that it is alive.

It then **starts an election**.

Processes that do not hear a proposal **hear the results**.

Distributing the Results

Once a leader is elected, the **results must be distributed**.

In order to **maintain agreement**, this is not quite trivial.

The new leader sends a **special message** to all processes.

(All processes now have **smaller IDs** than the newly-elected leader!)

Every process hearing this message moves to the **waiting** state.

Once every process is in the waiting state, the new leader **announces its election**.

Every process moves to the **normal** state

Liveness

This protocol is **vulnerable to deadlock!**

1. If a process p_j responds to p_i but fails **before electing itself**
2. If the elected process fails after putting processes in the waiting state and **before declaring victory**

In both cases:

After **some timeout**, a blocked process **starts a new election**.

If the same potential leader is still alive, **its election will complete**.

If it is not, **the next-largest node ID** will be elected.

Correctness

How do we know the correct process of **largest ID** is elected?

Suppose that some process p_i with ID smaller than p_j is elected.

We know that:

- p_i sent an election proposal to p_j .
- p_j did not send an election announcement to p_i .

Therefore, p_j must have failed!

Efficiency of the Bully Algorithm

In the **worst case**, the bully algorithm sends $O(n^2)$ messages.

Consider: p_i of the **lowest priority** starts an election.

Every p_j of higher priority will **also start an election**.

In the best case, it sends $O(n)$ messages.

If the correct process of **highest remaining ID** continually fails during election, it can time out many times in succession.

Ring Election

Another interesting and simple protocol uses a **ring** [2].

Processes are arranged in a **communication ring**.

Each process has a **clockwise** and **counterclockwise** neighbor.

Every process has an **ID**, and IDs form a **total ordering**.

The algorithm will elect the correct process with the **largest ID**.

The Protocol

Any process may start an election at any time.

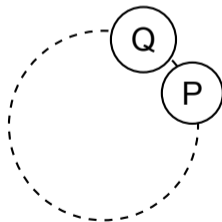
To start an election, a process **sends an election message** containing its own ID **counterclockwise**.

On receiving an election message, each process:

- **Declares victory** if the message contains its ID
- Forwards the message if the message ID is **larger than** its own ID
- Forwards **its own ID** if the ID is **smaller**

Efficiency of the Ring Algorithm

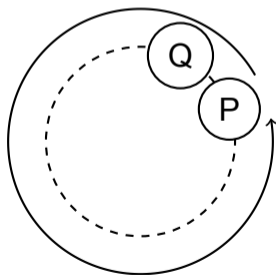
If there are no failures during election, this algorithm is $O(n)$.



Efficiency of the Ring Algorithm

If there are no failures **during election**, this algorithm is $O(n)$.

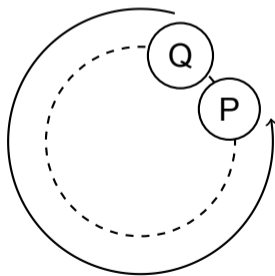
If the process P of **largest ID** starts the election, n messages are sent.



Efficiency of the Ring Algorithm

If there are no failures **during election**, this algorithm is $O(n)$.

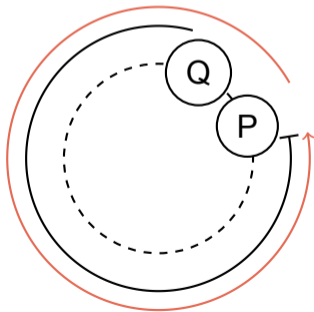
If the process Q **immediately counterclockwise** of the winner starts it, $2n - 1$ messages are sent.



Efficiency of the Ring Algorithm

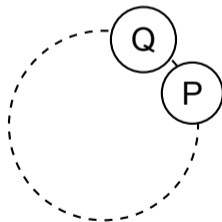
If there are no failures during election, this algorithm is $O(n)$.

If the process Q immediately counterclockwise of the winner starts it, $2n - 1$ messages are sent.



Efficiency of the Ring Algorithm

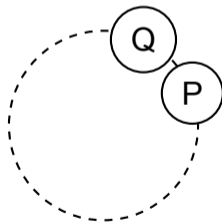
If there are no failures during election, this algorithm is $O(n)$.



Efficiency of the Ring Algorithm

If there are no failures during election, this algorithm is $O(n)$.

If processes die during election, it can be n^2 .



Other Properties of the Ring Algorithm

This protocol is **also vulnerable** to deadlock.

Timeouts can once again be used to solve this.

Note that **even though processes send only counterclockwise**, every process must know **or be able to find** other processes.

Consider what happens when P 's counterclockwise neighbor fails!

If we want the guarantees of the bully algorithm, we need **another round**.

Unique Identifiers

Both of these algorithms require **unique IDs**.

How do we get those **without a coordinator**?

We have mentioned **cryptographic hashes** before.

How they can be used depends on **what we're defending against**.

Disclaimer

This is **not a security course**.

Our coverage of security issues will be superficial.

It is easy to **draw false conclusions** with such analysis.

Take a security course!

Threat Models

We must define a **threat model** in order to answer this.

The threat model captures:

- Whether you expect to have **adversaries**
- What kind of resources the adversaries will have
- What failures you are protecting against

For example:

Adversarial processes may try to adopt a process ID larger than any process in the system in order to become the leader. They can spend up to s CPU seconds to accomplish this.

Proof of Work

A common technique for combating this is **proof of work**.

Proof of work participants must compute a function f that [3]:

- Takes some time to compute
- Is not easily precomputed or amortized
- Given x and y , it is easy to determine if $y = f(x)$

This forces a process to **invest effort** in a system.

Example Proof of Work

S/Kademlia [1] proposes a proof of work to prevent **flooding the DHT** with node IDs.

Each node is identified by a **cryptographic hash**.

That hash must meet several properties:

- It must be the hash of a **public key** k in a public key cryptosystem: $h = \text{SHA1}(k)$
- If $i = \text{SHA1}(h)$; i must have b **leading zero bits**

Generating a public key is slow, and **selecting for b is hard**.

This means that a process must **generate many keys**, slowly.

Safer Proofs

SHA-1 is **badly broken** and should not be used.

Generating public/private key pairs is easier than it used to be.

Proof of work must be **parameterized** and **updated**.

Some functions can be **arbitrarily iterated**. For example:
 $H(H(H(\dots)))$ for some hash function H .

If **generating an ID** is hard, generating **a specific ID** is harder.

Suppose it takes s seconds to compute an ID.

Generating the largest ID in a pool now takes $\gg s$ seconds!

Summary

- Centralized authority doesn't mean **permanent authority**
- Distributed elections can be held
 - Bully algorithm
 - Ring algorithm
- Global identifiers **keep cropping up**
- **Proof of work** can make global IDs safer
- Security guarantees require **threat models**

References I

Required Readings

- [2] Ernest Chang and Rosemary Roberts. “An Improved Algorithm for Decentralized Extrema-finding in Circular Configurations of Processes”. In: *Communications of the ACM* 22.5 (May 1979), pp. 281–283. DOI: 10.1145/359104.359108. URL: https://search.lib.buffalo.edu/permalink/01SUNY_BUF/12pkqkt/cdi_crossref_primary_10_1145_359104_359108.

Optional Readings

References II

- [1] Ingmar Baumgart and Sebastien Mies. “S/Kademlia: A Practicable Approach towards Secure Key-Based Routing”. In: *International Conference on Parallel and Distributed Systems*. Dec. 2007, pp. 1–8. DOI: 10.1109/ICPADS.2007.4447808. URL: https://search.lib.buffalo.edu/permalink/01SUNY_BUF/12pkqkt/cdi_ieee_primary_4447808.
- [3] Cynthia Dwork and Moni Naor. “Pricing via Processing or Combatting Junk Mail”. In: *International Cryptology Conference*. Vol. 740. Springer-Verlag, Aug. 1992, pp. 139–147. DOI: 10.1007/3-540-48071-4_10. URL: https://link.springer.com/chapter/10.1007\%2F3-540-48071-4_10.

References III

- [4] Hector Garcia-Molina. “Elections in a Distributed Computing System”. In: *IEEE Transactions on Computers* C-31.1 (Jan. 1982), pp. 48–59. DOI: 10.1109/TC.1982.1675885. URL: https://search.lib.buffalo.edu/permalink/01SUNY_BUF/12pkqkt/cdi_crossref_primary_10_1109_TC_1982_1675885.

Copyright 2021 Ethan Blanton, All Rights Reserved.

Reproduction of this material without written consent of the author is prohibited.

To retrieve a copy of this material, or related materials, see <https://www.cse.buffalo.edu/~eblanton/>.