# Byzantine Generals problem

Amirreza Taghizadeh

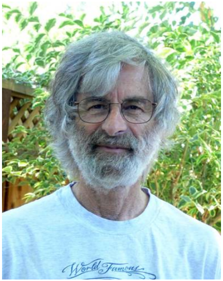
May 3, 2023

In this presentation, we aim to discuss a rproblem in distributed systems known as "**Byzantine generals problem"** proposed by **Leslie Lamport.**

# Outline

# Outline

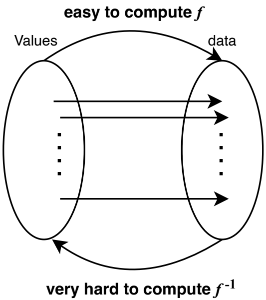
# Brief history of Lamport's works




- **Inventing LaTeX:** a group of macros, which can make the life of TeXusers a lot easier!!
- **One way authentication** in Whitfield Diffie's "*New directions in cryptography*" (1976)

# brief description of **One-way Function**

- A one way function $\boldsymbol{f}$ is a function that is **easy to compute** but whose **inverse is difficult to compute**:
- or to say $f$ is one-way function iff (adapted from[1]):
  1. for all value $v$, finding data object $d$ s.t. $\phi(d) = v$ is **computationally infeasible.**
  2. $f$ is not **one-to-one** or proving it otherwise is **computationally infeasible.**[2]




[1] Lamport, "*Constructing Digital Signatures from a One Way Function*"

[2] i.e. to say: $\forall$ data object $d' \neq d : \phi(d') \neq \phi(d)$

# brief description of **One-way Authentication**

**Definition (Digital signature)**

A digital signature created by **sender P** for **document m** is a data item $\sigma_p(m)$ that is when received together with **m**, one can determine (e.g. in a court of law) that **P** generated document **m**.
Hence A tool for determining **validity** of something sent.[1]

[1] Lamport, L. (1979) "*Constructing Digital Signatures from a One Way Function*"

# brief description of **One-way Authentication**

**Definition (One way authentication)**

It must be **easy for anyone** to recognize the signature as **authentic** but **impossible** for anyone other than the signer to produce it![1]

[1] Diffie, W. (1976)" *New Directions in Cryptography*"

# A practical Example of a **One-way function** in **one-way authentication**
# **Login Problem**

- User **A** enters Password $PW$ and computer store it as $f(PW)$
- where $f(PW)$ is a one-way function of <u>10 million instructions</u>
- and its inverse has $10^{30}$ more instructions (or computations), which practically makes it **noninvertible**
- for example, finding square root of $x_0$ given in $f(x) = x^2$ is much harder than computing $x^2$ at $x_0$.

# brief description of **One-way Authentication** *Cont'd*

- However, determining exactly what the one-way function should be is originally solved by **Lamport**
  which further lead to the publication of the paper: "*Constructing Digital Signatures from a One Way Function*"
- But how this solution relates to the ecosystem of **public keys** is <u>out of the scope of the presentation</u> and discussed in the paper: "*New Directions in Cryptography*" by Whitfield Diffie (1976)

# Brief history of Lamport's works

- **Inventing LaTeX:** a group of macros, which can make the life of TeXusers a lot easier!!
- **One way authentication** in Whitfield Diffie's "*New directions in cryptography*" (1976)
- **bakery algorithm** an algorithm to ensure mutual exclusion in concurrent processes
- that is to ensure a data structure is modified by at most one process at a time
  and no process is reading a data structure while it is being written by other processes.
- **Paxos algorithm**: an algorithm used in distributed systems for reaching consensus, used in distributed storage systems

[1] Silberschatz, "Database System Concepts", Ch. 19, P. 965

# Brief history of Lamport's works *cont'd*

- Time, clocks and ordering of events in a distributed system:
  - in a distributed system, sometimes it is **impossible** to say an event has happened before something else.
  - hence, relation "*happened before* is a partial ordering relation.
  - **Partial ordering? sounds familiar...**

**Definition (Partial ordering)**

Partial ordering relation is an ordering relation in which not all members of the set need to be comparable!

  - He proposed in that paper a partial ordering of "happened before" and gave a **distributed algorithm** for **extending it to a total ordering of events**

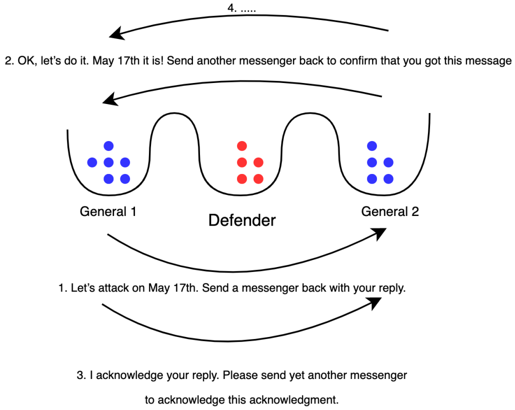
**But where is the "Byzantine generals" problem in the list?**

# Outline

# Brief history of the origination of the problem

- **Naming:** "Because it was posed as a cute problem about philosophers seated around a table, **Dijkstra's dining philosopher's problem** received much more attention than it deserves. (For example, it has probably received more attention in the theory community than the readers/writers problem, which illustrates the same principles and has much more practical importance.) . . . The popularity of the dining philosophers problem taught me that **the best way to attract attention to a problem is to present it in terms of a story**." -Lamport
- **Motivation:** Two General's problem. Byzantine General is a rather more general form of this problem.

# A brief Overview of Two general's problem
## Stating the problem through visualization!




**Note: All Acknowledgement messages were sent by assuming the defender not capturing any messengers**

# Outline

# A review of the Byzantine problem

- There is an **enemy city** and a group of **General** $i$, each deciding to reach **an agreed upon plan** (which is the exact definition of **consensus**) to whether *Attack* or *Retreat*
- and each general $i$ is equipped with a messaging method for sending value $v(i)(\in \{\text{Attack, Retreat}\})$ for communicating with each other
- AND there are a bunch of **traitorous generals** sending **conflicting** messages, aim to prevent **loyal generals to reach a plan**
- In fact, they send **arbitrary messages** to other generals.
- In order for them to a reach consensus, two conditions must be satisfied:
  1. Every loyal general must obtain the same information $v(1), v(2) \ldots, v(n)$
  2. The value sent by a loyal general should be used by all loyal generals

# A review of the Byzantine problem

# A review of the Byzantine problem *cont.*

How should the generals send their messages?
Let's examine how **a single general** $i$ **should send the message** $v(i)$ that is formally defined by: *why??* (which is made by grouping generals into two groups, namely **commander and lieutenant generals**)

## Definition (Byzantine Generals Problem)

A **commanding general** must send an order to his $n-1$ **lieutenant generals** s.t.:
IC1. All loyal lieutenants obey the same order.
IC2. If the commanding general is loyal, then every loyal lieutenant obeys the order he sends.

Note that IC2 $\implies$ IC1
***** Also note that,E.g. when General $n$ sends $v(n)$ lieutenant $n-1$ retrieves a message from $n-2$ generals and then apply a function $Majority(v(1), v(2), \dots, v(n-2))$ and adds $v(n)$ to the list $V_1$

# A review of the Byzantine problem *cont.*

Lamport gave a recursive algorithm based on **majority function** for the mentioned problem in case of having **oral messages** whose content is solely managed by the sender.

Unfortunately, the algorithm there are two problems concerning this algorithm:

1. **It is expensive** $\rightarrow O(n!)$
2. It only works only, in case of having $m$ traitors, for $n \geq 3m + 1$ generals

In order to deal with problem 2, he proposed an algorithm based on **unforgeable messages.**

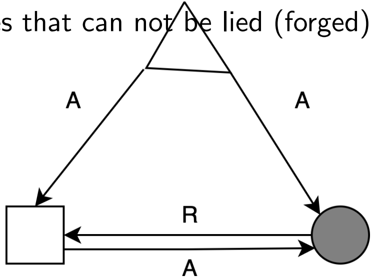
# Outline

# Signed messages

What was wrong with the oral messages?

Answer: because **traitors lie**, they **alter the contents** of the messages they receive and send.

Let's make messages that can not be lied (forged) or any alterations could be detected



# Formal **&abstract** Assumptions regarding Messages!

A1. Every Message that is sent is delivered correctly.
A2. There receiver of a message knows who sent it.
A3. The absence of a message can be detected.

$\rightarrow$ **Oral messages**

A4. (a) A loyal general's signature cannot be forged (**or changed!!**) and any alteration of the contents of his signed messages can be detected

(b) Anyone can verify the authenticity of a general's signature

$\downarrow$

**Signed messages**

BUT, in case of assuming A4, What is the purpose of a traitor?????

# Signed messages algorithm

Note that we know **the number of m** traitors when running the Alg.

1. Commander sends a **message** having a value $v$ and a signature (which is a sequence of IDs) $\rightarrow$ $\boldsymbol{v : 0}$
2. each lieutenant $i$ receives the message of length $k$, **adds the** $v$ **to a** $V_i$ **set**, **adds his ID** to the message and **sends it to those child lieutenants not having received this message before**
3. when lieutenant $i$ receives no more messages, the lieutenant $i$ applies the **a Choice function** to $V_i$ in order to retrieve an order.

What is that **Choice function?**

# Choice function

The Choice function could be any **aggregate function** (such as median, average, etc) BUT, it needs to have two essential properties:

1. if Set $V_i$ consists of single value $v$ Then, $Choice(V_i) = v$
2. $Choice(\emptyset) = RETREAT$

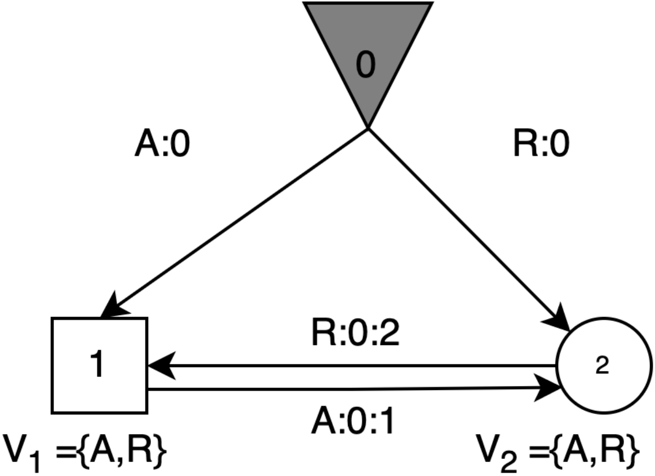
# Formal statement of Signed messages $SM(m)$

Initially $V_i = \emptyset$
(1) The commander signs and sends message $v : 0$ to all lieutenants
(2) For each $i$:
(A) If Lieutenant $i$ receives a message of the form $v : 0$ from the commander and he hasn't receieved any order, then:
(i) he lets $V_i = v$
(ii) he sends the message $v : 0 : i$ to every other lieutenant.

(B) If Lieutenant $i$ receives a message of the form $v : 0 : j_1 \cdots : j_k$ **and** $v \notin V_i$ then:
(i) he adds $v$ to $V_i$;
(ii) if $k < m$, then he sends the message $v : 0 : j_1 \cdots : j_k : i$ to every lieutenant other than $j_1, \ldots, j_k$

(3) For each $i$: When Lieutenant i will receive no more messages, he obeys the order $Choice(V_i)$

# The impossible case revisited



- The proposed algorithm **increased fault tolerance**, but it is still **expensive** $\rightarrow O(n!)$
- In what follows, we give a description of a more **practical and efficient algorithm** which can tolerate Byzantine-faults (i.e. arbitrary messages and faults in nodes) in asynchronous systems, proposed by *Castro M. & Liskov B.*
- but first let us define what is meant by **synchronous and asynchronous systems.**

# Outline

### Definition (Synchronous distributed system)

A distributed system comprising of processes is synchronous iff it has the following properties[1]

1. There is a real-time physical clock which provides synchronization among multiple processes
2. A known upper bound $\epsilon$ on processing delays
3. A known upper bound $\delta$ on message transmission delays.

[a] "*Reliable and Secure Distributed Programming*" A book by Rodrigues L. & Cachin

# Asynchronous systems → Lamport's Time clocks revisted

**Definition (Asynchronous system[1])**

In such systems, there is no time assumptions about processes. That is, we do not have **physical clock**. Instead **Time** is defined with respect to communication and measured by a *Logical clock*.

Here is Lamport's Algorithm for finding something has "happened before" something else.

[1] "*Reliable and Secure Distributed Programming*" A book by Rodrigues L. & Cachin

# A brief description of Lamport's Algorithm for Time clocks

1. Each process $p$ keeps an ineger called *logical clock* $l_p$ initially 0.
2. Whenver an event occurs at Node $p$, the logical clock $l_p$ is incremented by one unit.
3. When a process **sends a message,** it adds a timestamp to the message with the value of its logical clock at the moment the message is sent. The timestamp of an event $e$ is denoted by $t(e)$.
4. When a Node $p$ receives a message $m$ with timestamp $t_m$, Node $p$ increments its logical clock in the following way: $l_p := \max\{l_p, t_m\} + 1$

# A brief description of Lamport's Algorithm for Time clocks *cont.*

- now that we have a mechanism for time in asynchronous world, let us define the *happened before* relation

**Definition (Happened before relation $e_1 \rightarrow e_2$)**

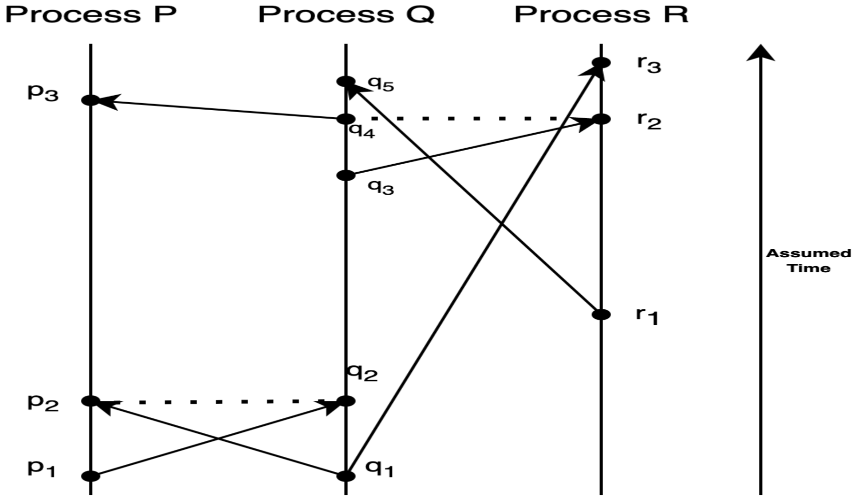
For event $e_1$ and $e_2$ we say "$e_1$ *has happened before* $e_2$" ($e_1 \rightarrow e_2$) if $e_1$ and $e_2$ have the following conditions:

1. $e_1$ and $e_2$ occurred at the **same process p** and $e_1$ occurred before $e_2$ (i.e. $t(e_2) = t(e_1) + 1$)
2. $e_1$ corresponds to the transmission of a message $\boldsymbol{m}$ at process $p$ and $e_2$ to the reception of $m$ at process $q$.
3. $\rightarrow$ be transitive.

[1] *"Reliable and Secure Distributed Programming"* A book by Rodrigues L. & Cachin

# A brief description of Lamport's Algorithm for Time clocks *cont.*

The above definition implies that if $e_1 \rightarrow e_2 \implies t(e_1) < t(e_2)$



# Practical Byzantine Fault Tolerance (PBFT)

This Algorithm describes the implementation of a
**Byzantine-fault-tolerant disributed file system.**
Assumptions:

1. We have a **asynchronous distributed system** $\implies$ network may have: **Delays, failure to deliver messages; duplicate them, delivery out of order**
2. For Byzantine assumption (nodes behaving arbitrarily): Independent node failures $\implies$ each node runs service code **independently**, have different root password and a different admin.
3. Use cryptographic techniques to prevent spoofing and to detect corrupted messages $\rightarrow$ messages have *public key signatures*, message authentication codes.
4. A strong adversary coordinates faulty nodes, delay communication and delay correct nodes
5. The adversary cannot violate cryptographic techniques used by nodes.

# Practical Byzantine Fault Tolerance (PBFT)

Properties:

1. The algorithm used to implement any **replicated service** with *state* and *some operations*, including but not limited to *read and writes on replicas*
2. There are some **faulty** and **non-faulty clients(requesting to invoke operations)** and **replicas** (we have $f$ faulty replicas and $n = 3f + 1$ number of all replicas))
3. The Algorithm both provides **safety** and **liveness**
   1. **Safety (Accuracy):** replicated service satisfies linearizability.
   2. **Liveness (Completeness):** Clients eventually receive replies to their requests, provided at most $\lfloor \frac{n-1}{3} \rfloor$ are faulty and $\underline{delay(t)}$ does not grow faster than $t$ indefinitely.

4. Algorithm does not rely on **synchrony** to provide **safety**
5. However, **It must** rely on synchrony to provide **liveness** $\rightarrow$ **FLP's impossibility**
6. The algorithm uses a 3-phase commit protocol: Pre-prepare, prepare, commit

# PBFT Algorithm

- There is a primary replica in each request of a client c.
- Replicas move through a seccession of configurations called **views**.
- in a view, one replica is the primary and others are backups.

# PBFT Alg. Cont.

A very brief description of the Algorithm:

- A client sends a request to invoke a service operation to the primary replica
- The client's request message is of the form $< REQUEST, o, t, c >_{\sigma_c}$
- The primary multicasts the request to the backups
- Replicas execute the request and send a reply to the client
- The client waits for $f + 1$ replies from different replicas with the same result; this the result of the operation.
- The replica's reply is of the form $< REQUEST, \nu, t, c, i, r >_{\sigma_i}$

# PBFT Alg. Cont.

- The client waits for $f+1$ replies with valid signatures from different replicas, and with the same $t$ and $r$, before accepting the result $r$
- If client doesn't receive replies soon, it broadcasts the request to all replicas.
- If the request has already been processed, the replicas simply re-send the reply;
- otherwise, if the replica is not the primary, it relays the request to the primary.
- If the primary doesn't multicast the request to the group, it will eventually be suspected to be faulty by enough replicas to cause a view change.
- **The algorithm is proved to be Only 3% slower than non-replicated implementation of NFS**

# Outline

# Topics Discussed

- Appreciation of Lamport's Other Works: LaTeX, Paxos, Bakery algorithm, One-way authentication, **Time clocks in Asynchronous Distributed systems** etc.
- What is the Byzantine generals problem and how does it relate to the problem of solving **consensus**?
- Using Oral messaging to solve Byzantine-fault problem + **its weaknesses**$\rightarrow n \geq 3m + 1$
- What is the structure of **signed and unforgeable messages** and How does it solve Oral Messaging?
- Defining time with **logical clocks** in a system **without a global physical clock** (Asynchronous Systems)
- Is there a more efficient algorithm for Solving Byzantine failure in asynchronous System? $\rightarrow$ Practical Byzantine-fault Tolerant Algorithm

# References:


L. Lamport, R. Shostak, and M. Pease
The Byzantine Generals Problem
*ACM Transactions on Programming Languages and Systems (1982).*

M. Fishcher, N. Lynch, and M. Paterson
Impossibility of Distributed Consensus with One Faulty Process
*Journal of the Association for Computing Machinery, Vol. 32, No. 2, (1985).*

C. Cachin, R. Guerraoui and L. Rodrigues
Introduction to Reliable and Secure Distributed Programming, 2nd Edition
*Springer (2011), p. 44-48.*

A. Silberschatz, H. Korth, and S. Sudarshan
Database System Concepts, 7th Edition
*McGraw-Hill Education (2020), p. 965*

L. Lamport
Time, Clocks, and the Ordering of Events in a Distributed System
*Journal of the Association for Computing Machinery, Vol. 21, No. 7, (1978)*

# References:


M. Castro, B. Liskov
Practical Byzantine Fault Tolerance
*Third Symposium on Operating Systems Design and Implementation, New Orleans, (1999)*

W. Diffie and M. Hellman
New Directions in Cryptography
*IEEE Transactions on Information Theory, Vol. IT-22, No. 6 (1976)*

L. Lamport cd
Constructing Digital Signatures from a One Way Function
*SRI International (1979)*

F. A. Parand
Failure and Consensus
*(.ppt) file used as teaching material in Distributed Systems class by Prof. Parand, Allameh Tabatabai University*

Any Questions?