Università degli Studi dell'Aquila

APPLICATION OF CONSENSUS PROBLEM WITH BYZANTINE NODES TO MONITORING PROBLEM

Source:

- "*Applying Byzantine Agreement Protocol to the Intrusion Detection Problem in distributed systems*" by Fernando C. Colon Osorio – Xiaoning Wang
- "*Dominanza su griglie con tolleranza a nodi bizantini*" – Degree thesis by F. Cellinese

Students:
Dell'Elce Luana
Cicchini Stefania

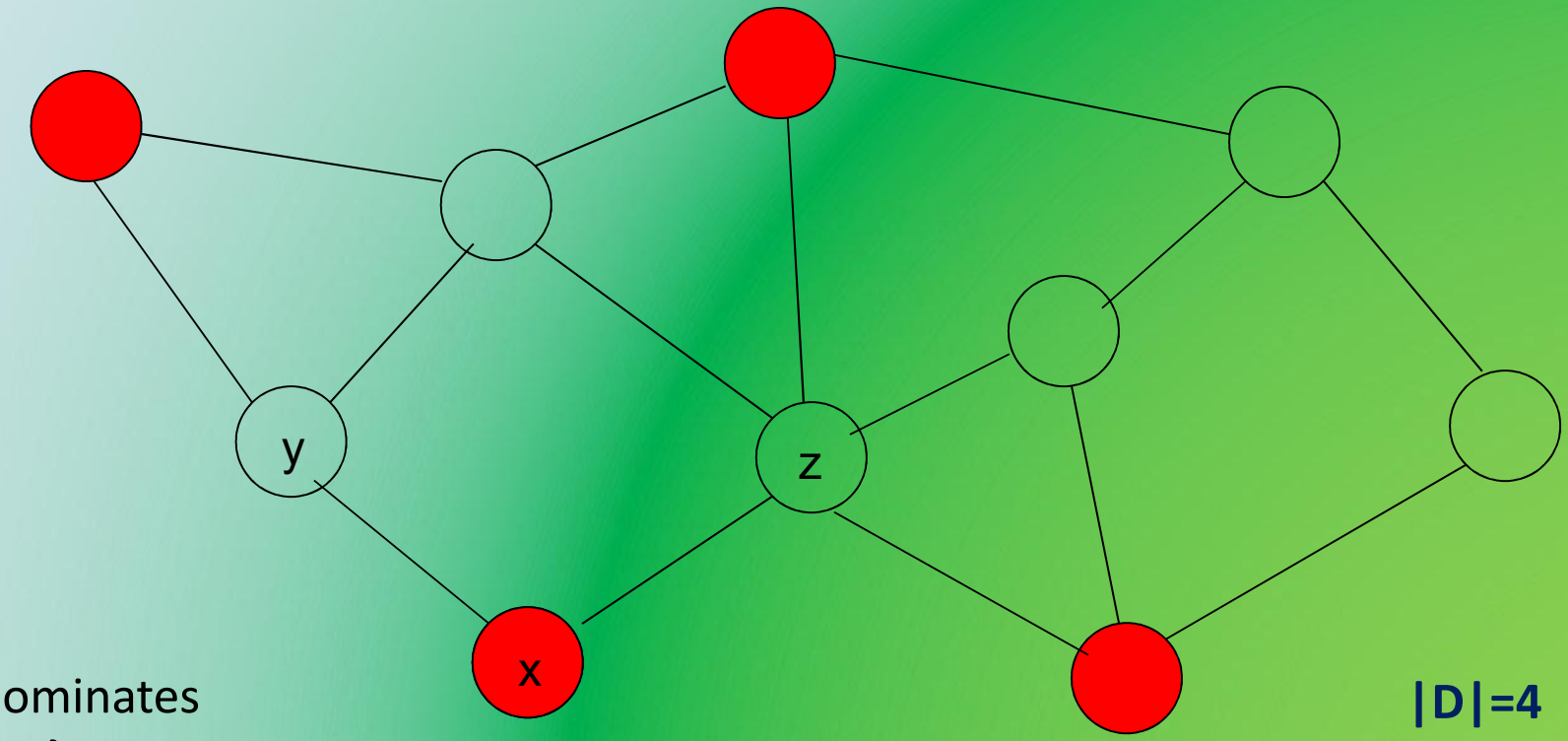
MONITORING PROBLEM

A major activity in Distributed Systems consists of monitoring whether all the system components work properly. This problem is very important for real-life applications whose *components' communication scheme* make use of an Distributed System which can be modelled by means Message Passing System.

At this purpose we introduce **Network Monitoring Systems (NMS)**, which can monitor any network for problems caused by node and links malfunctioning.
A NMS has two functions:
1.monitoring the network;
2.reporting a malfunctioning to the system administrator.

In particular, in a NMS, the status of nodes and edges is monitored through the use of sentinel nodes, which periodically exchange messages with adjacent nodes and then report some kind of information to the network administrator.

In the case in which a **sentinel** node communicate only with its *adjacent* nodes the monitoring problem is reduced to **dominance problem** in graphs, with the activity of selecting a set D of nodes (**dominators**) in a graph in order to have all the nodes of the graph within distance *at most 1* from at least a dominator.



x dominates {x,y,z}

|D|=4

We are going to study the possibility to calculate the **Minimum Dominating Set** tolerant to byzantine nodes, named **Liar's Dominating Set,** in graph with grid topology.

What are Byzantine nodes?

In a graph G=(V,E), a **byzantine node** is a sentinel which is not able to identify correctly intrusions or report to the system administrator a wrong position.

Let's give some notions:

Grid Graph: A graph G= (V, E) of m *x* n dimension has grid topology if for each pair of adjacent nodes (v_{ij}, $v_{i'j'}$) following definitions hold:

$i = i'$ and $|j-j'| = 1$

$j = j'$ and $|i-i'| = 1$.

Order of a node: In a grid m *x* n with vertices $v_{i,j}$, $1 \leq i \leq m$ and $1 \leq j \leq n$, node $v_{i,j}$ has order less than node $v_{k,p}$ either if $i < k$ or if $i = k$ and $j < p$.

Graph's grade: Given a graph G, the minimum grade of G, $\boldsymbol{\delta}(G)$, is the vertex's grade with the minimum number of incident edges.

Neighbourhood of a node: Given a node x, N[x] is the set of adjacent nodes to x, including x.

Dominating Set: Given a graph G = (V, E), a dominating set K is a set K c V such that, for each v ϵ V, it is adjacent at least to **one** node in K.

Doubly Dominating set: Given a graph G = (V, E), a doubly dominating set is a set K c V such that, for each v ϵ V, it is adjacent to at least to **two** nodes in K.

LDS (Liar's Dominating Set): Given a graph G = (V, E), LDS is a set K c V such that, K is a doubly dominating set with respect G, and for each pair of nodes (x, y) of V, the following inequality holds:

$$|(N[x] \cup N[y]) \cap K| \geq 3$$

Thanks to this set, it is possible locate intruder and byzantine nodes.

Minimum LDS: A minimum LDS is an LDS with minimum cardinality $\gamma_{LR}(G)$.

THE ALGORITHM

[by student: F. Cellinese]

The proposed algorithm will search **doubly dominating sets** rather than minimum LDS, checking then if they are LDS too.

The current algorithm is able to find a LDS on grid graphs even in presence of **one byzantine node**. Knowing that the algorithm constructs all possible doubly dominating sets, in an **increasing way** with respect to cardinality, so the first set that satisfies the LDS definition it is the minimum too.

This is a **branch and bound algorithm**, that executes a BFS operation (**Breadth-First Search - BFS**) over the **Branch Decision Tree (BDT)**. This ensure that every time an ID is examined, it will have a non-less number of dominating nodes than previous examined ID. Each node of the Branch Decision Tree (BDT) is the current state solution of the following problem:

"Try to dominate the *smallest* node non-doubly dominated".

At each step the algorithm examines an ID of the Branch Decision Tree.

An ID is a set of nodes capable to keep different information about each node:

- If a node is dominated;
- If a node is doubly dominated;
- If a node is dominating.

The algorithm *terminates* when it arrives to an ID in which dominating nodes match to the LDS definition.

The data structures chosen to implement the algorithm are a **queues**, a **Binary Search Tree (BST)** and a **generic set of nodes** which represents the ID of BDT.

In order to make possible the *horizontal* slide of IDs, the creating sets are always *insert in queues* and extract one by one, until one reaches to a solution.

However, using only a queue, the probability that from an ID, ones reach to an already created ID it's high, with corresponding redundancy. So to avoid this problem, one needs to put besides to queue a **Binary Search Tree**.

Moreover the use of BST makes the algorithm **faster**, reducing the cost of the *ID searching*: **O(log n)** instead of *O(n)*, where n is the number of IDs already computed.

PSEUDOCODE

(non-anonymous system)

Input: A graph G = (V, E) with grid topology;
A queue of sets, initially with an inner empty set;
A Binary Search Tree T, initially empty.

Output: The minimum LDS on G.

Procedure:

(1) One extracts the first set K from queue, in which one searches the *non-doubly dominated* node of *minimum order*.
If such node is not present go to step 2 otherwise go to step 3.

(2) If K it's *doubly dominating* then is a potential candidate being an LDS.
One examines each possible pair of nodes (x, y) such that holds the following:

$$|(N[x] \cup N[y]) \cap K| \geq 3.$$

In that case, the algorithm terminates and returns solution k.
Otherwise come back to step 1.

(3) Once the first non-doubly dominated node v is identified, one needs dominate it through itself and all its adjacent nodes. For each of those nodes:

- If node Y is already *dominating*, it is deleted from K and one examines the next one;
- One creates a new set U = K;
- Y become *dominating* in U;
- If U does not belong to T, one inserts U into queue and in the BST, otherwise U is deleted.

(4) One eliminates K from T and one executes again step 1.

APPLICATION OF BYZANTINE AGREEMENT PROTOCOLS TO THE INTRUSION DETECTION PROBLEM

- Necessary and Sufficient *conditions* for application of **Byzantine Agreement Protocol** to the **Intrusion Detection Problem.**
- Designing an **Intrusion Detection & Countermeasure System, SAFE** developed at the **Worcester Polytechnic Institute System Security Research Laboratory** (WSSRL)

The key to such a system is a distributed module:

The distributed Trust Manager or **TTM.**

The **TTM** serves three major system functions:

- Using a **Trust Relationship Matrix,** one can know which logical nodes in the system can be **trusted**;
- Using **Byzantine Agreement Protocol**, one can identify and isolate nodes that have been compromised;
- Preventing the trust system from being partition (**Quorum formation**).

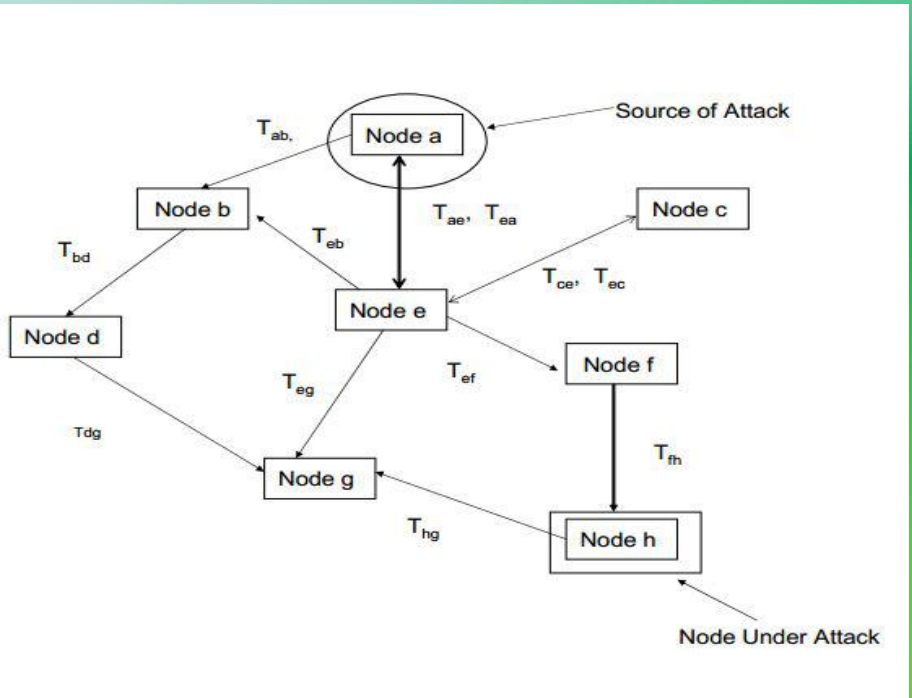
In **SAFE**, the **Primary Security Mechanism** used is the creation, updating, and maintenance of a **Trust Relationship Matrix**.
This matrix, which is managed by the **Trust Manager**,
contains update informations on the **Trust Relationship**
between all the nodes in the system.

Definition:

Given a **Network Monitoring System G=(V,E)**
and nodes **i** and **j** of **V**,
each element $T_{ij}(t)$ of **Trust Relationship Matrix** represents
the level of trust that node **i** has with respect to node ***j*** at time **t**.
$T_{ij}(t) = 0$ denotes the lack of trust between **i** and **j**, at time **t**.
$T_{ij}(t)$ is called **Trust Function.**

Trust Relationship Matrix is not necessarily symmetric,
because the processors **i** and **j** may have a
different level of **trust** with each other.

Let's see an example:



Node **a** is the ***source*** **of the intruder attack.**
Node **h** is the ***target*** **of the intruder attack**.
Since there is not a direct trust relationship between node **a** and node **h**, the **intruder** is forced to a set of **attempted intrusions** into nodes **e** and **f**, before attempting to compromise ***h***.
Due to the topology of the Trust Relationships between nodes, compromising any node other than nodes e or f will not allow the intruder to compromise the *target* node *h*.

Such a topological constraint among nodes,
brings **Significant Advantages** over other approaches.

Once, the Intrusion Detection Problem is formulated,
then, well known *solutions* to the **Byzantine Generals Problem** are readily available.
This kind of solution can be used in the design of the Trust Manager

At this purpose, we present the **Byzantine Agreement Protocol (BAP),** which aims to establish a **fault-tolerant agreement** when one or more nodes in a system have been compromised or failed.

THE BYZANTINE GENERALS PROBLEM

The **Byzantine Generals Problem**:
There are ***several divisions*** of a Byzantine army camped outside an enemy city, each division commanded by its own **general**.
The generals can communicate with other ones only by messenger.
They must decide upon a **common plan of action**. Some of the generals may be **traitors**, trying to prevent the **loyal generals** from reaching agreement.

The generals must have an algorithm to guarantee that:

- All **loyal generals** decide upon the same plan of action. The **traitors** may do anything they wish.

The algorithm must guarantee such a condition, regardless of what the traitors do.

- A small number of **traitors** cannot cause the loyal generals to adopt a bad plan.

THE BYZANTINE AGREEMENT PROTOCOL (BAP)

The **Byzantine Agreement Protocol algorithm** is a distributed algorithm designed to achieve consensus.

Several processes achieve consensus if they all agree on some allowed value called the **outcome**;

The **interface to consensus** has two *actions*:

- **allow a value**,
- **read the outcome**.

A consensus algorithm *terminates* when all non-faulty (uncompromised) processes know the outcome.

Let's substitute generals for nodes in distributed systems, and consensus for the need to agree on which nodes are safe/sane, **so** the problem of identifying a compromised node can be easily described as follow:

In SAFE there are several nodes, which cooperate with each other to detect intrusions.

Each node runs an autonomous agent, the **TTM**, which continuously sends messages to other nodes.

The message that is sends has two possible values:

- "**keep sane**" or "**0**";
- "**I am potentially compromised**" or "**1**".*

In this context, the nodes that can be **trusted** should be able to determine which nodes are compromised and arrive to a consensus.
A **small number of nodes** that have been compromised, should not be able cause the other nodes to adopt a *wrong* or even *malevolent* message.

The Byzantine Agreement Protocol (BAP), by which we can detect intrusion and minimize the success of attacks, fits this context perfectly.

Threshold:

There is a **threshold *t***:
If more than *t* nodes are compromised, the BAP will also fail.
The resiliency of the algorithms to the number of nodes compromised depends on the **communication mechanism** used to exchange messages, and its characterization by the **value** of *t*.

In order to improve **the intrusion-resiliency of SAFE**, as well as to make the implementation of the **Byzantine Agreement protocol** simpler in our system, we will assume the presence of a communication channel, which is both **reliable** and **secure**.

Signed Message Algorithm (SM)

In the **communication mechanism** used by SAFE, messages are **signed**, so it's always possible *determine* the ***sender*** of a message.

The **BAP** for signed messages is the **Signed Message Algorithm (SM)**, which can achieve consensus if *at most* **n - 2 nodes** have been compromised, among a total of **n** nodes, in the following way:

(1) The commander ***signs*** and **sends** its message to every node N_i it can reach directly;

(2) **E**ach node N_i:

- **(a)** Maintains a **running list** of all the messages it has received;
- **(b)** Then, it **signs** (authenticates) all the messages that node it has received, and then **sends a copy** with its signature to all other nodes that:
  - **(i)** Are directly connected (one hop away) to node N_i;
  - **(ii)** Whose messages he has not received yet;
- **(c)** The node will repeat step (b) until node N_i does not received any additional signed messages;

(3) N_i then will arrive at a decision on the course of action, based on all the signed messages.

It is also well noting that a *compromised* node will not necessarily follow the above **Signed Message Algorithm**. For that matter, a set of compromised nodes may collude with each other in an attempt to mislead or compromise all other sane nodes.

Collusion may take any of the following forms:

- *Falsify signatures*, i.e., use each other signatures as part of the authentication process;
- *Forge messages*;
- *Send incorrect values* or *not send any values* at all.

At least (n-2) trusted nodes could exchange messages to arrive at consensus, the Byzantine Agreement Protocol (BAP) holds.

This is **true if and only if** we can guarantee the following:

(A1) Every message that is sent is delivered correctly.

(A2) The receiver of a message knows who sent it.

(A3) **(a)** A sane node's signature cannot be forged, and any alteration of the contents of his signed messages can be detected.

(A3) **(b)** Anyone can verify the **authenticity** of a general's signature.

(A4) The **absence of a message** can be detected.

THE END

Thanks for attention!