Large Scale Security Systems

Project Part 2 - Mitigation Methods

Marcos Caramalho (114834, University of Aveiro)

2025-01-18

Contents

1 Introduction

This project builds upon a previous implementation of a large-scale security monitoring system, extending it with attack tolerance mechanisms. While part 1 focused on attack detection and response, this implementation addresses the challenge of maintaining system reliability in the presence of potentially compromised components.

The key objectives for this implementation were:

1) Adding Moving Target Defense capabilities
2) Implementing Byzantine Fault Tolerance
3) Usage of a consensus protocol, such as Paxos

The code for this can be found in the following git repository, under project-2: https://github.com/rezzmk/slsa-poc

2 State of the Art

2.1 Moving Target Defense

Moving Target Defense (MTD) represents a paradigm shift from static defenses to dynamic security measures. Modern MTD approaches typically include:

- Dynamic network address assignment
- Service diversification
- Runtime environment changes
- Port hopping techniques

The implementation build for this project focuses on port randomization, where the service endpoints regularly change their network footprint in order to change the surface of attack, which in turn changes the lay of the land and makes it more difficult for an attacker to do its bidding.

2.2 Byzantine Fault Tolerance

Byzantine Fault Tolerance (BFT) addresses the challenge of reaching consensus when some nodes might behave maliciously. Key developments in this area include:

- Practical Byzantine Fault Tolerance (PBFT)
- Consensus protocols like Paxos and Raft
- Blockchain consensus mechanisms

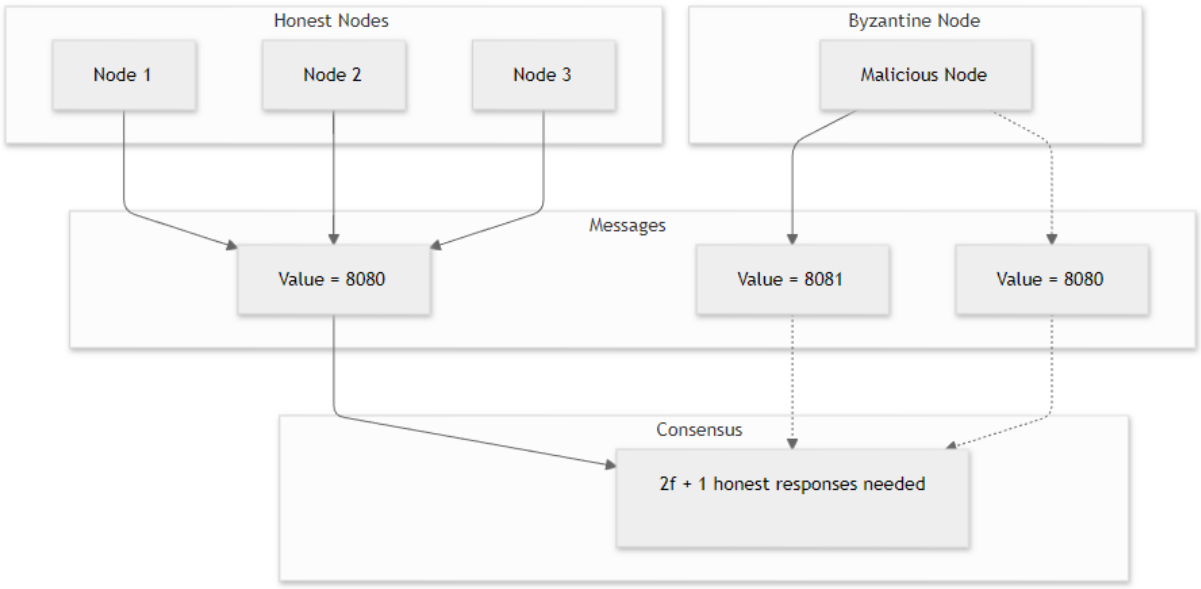
This implementation uses a BFT-enhanced version of Paxos for consensus on port assignments, requiring agreement from honest nodes while tolerating malicious behavior.

Essentially, for BFT we need:

1) Node Requirements
    1) Need 3m+1 total nodes to tolerate m malicious nodes
    2) At least 2m+1 honest nodes required for consensus
    3) Malicious nodes can send arbitrary messages
2) Detection Mechanisms
    1) Message verification
    2) Behavior monitoring
    3) Conflict detection

Some modern applications for BFTs are blockchain systems, distributed systems and critical infrastructures.

A high level view of this project's implementation can be seen next:



2.3 Paxos Consensus

Paxos represents one of the most known distributed consensus protocols, allowing a distributed system to reach agreement despite partial failures. The protocol operates in two main phases:

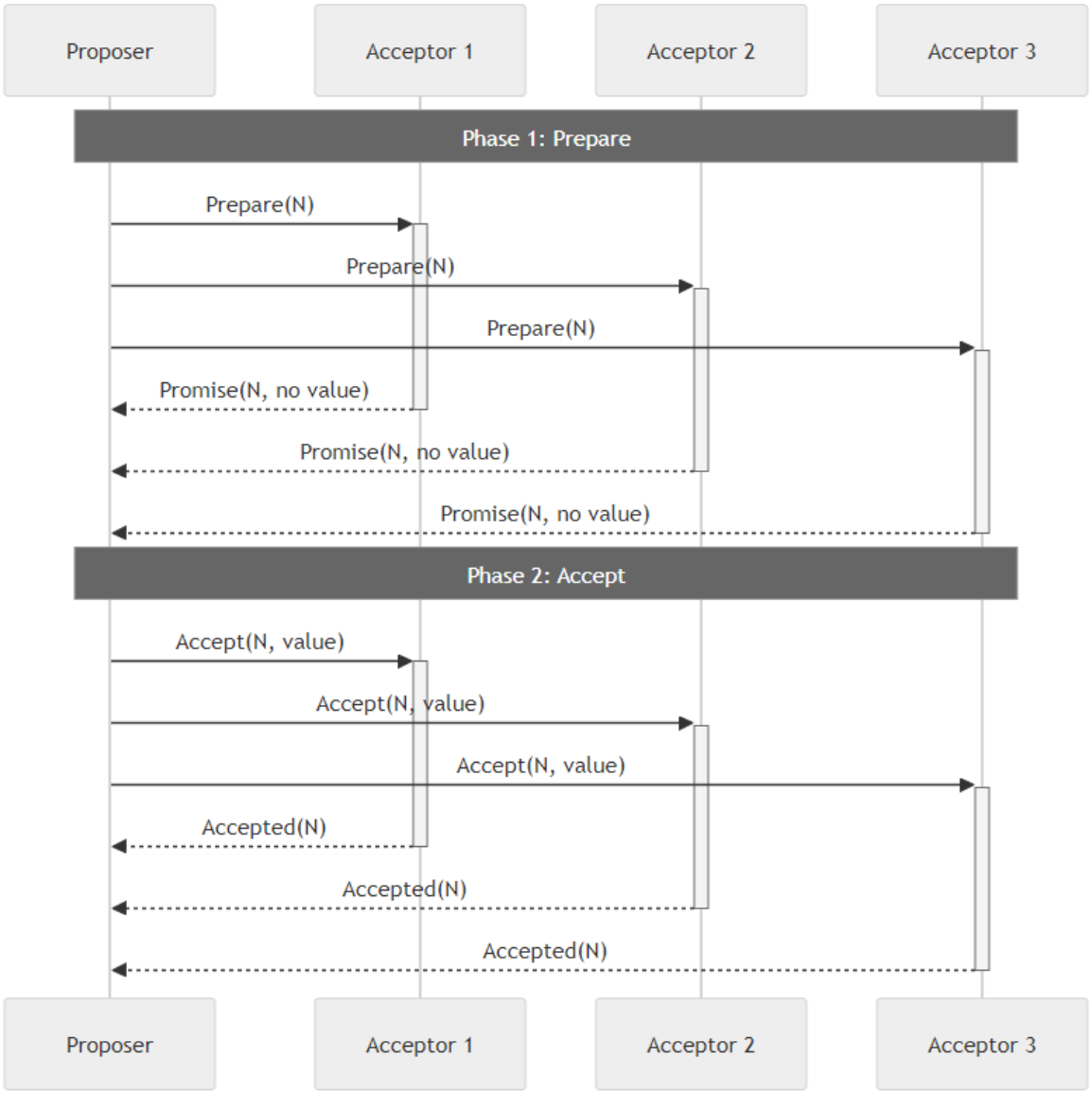
1) Prepare Phase (1a/1b)

1) Proposer sends prepare request with number N
2) Acceptors promise not to accept proposals < N
3) Acceptors return any previously accepted values

2) Accept Phase (2a/2b)

1) Proposer sends accept request with value
2) Acceptors accept if no higher numbered proposal seen
3) Majority acceptance required for consensus

This flow can be seen in the next diagram:

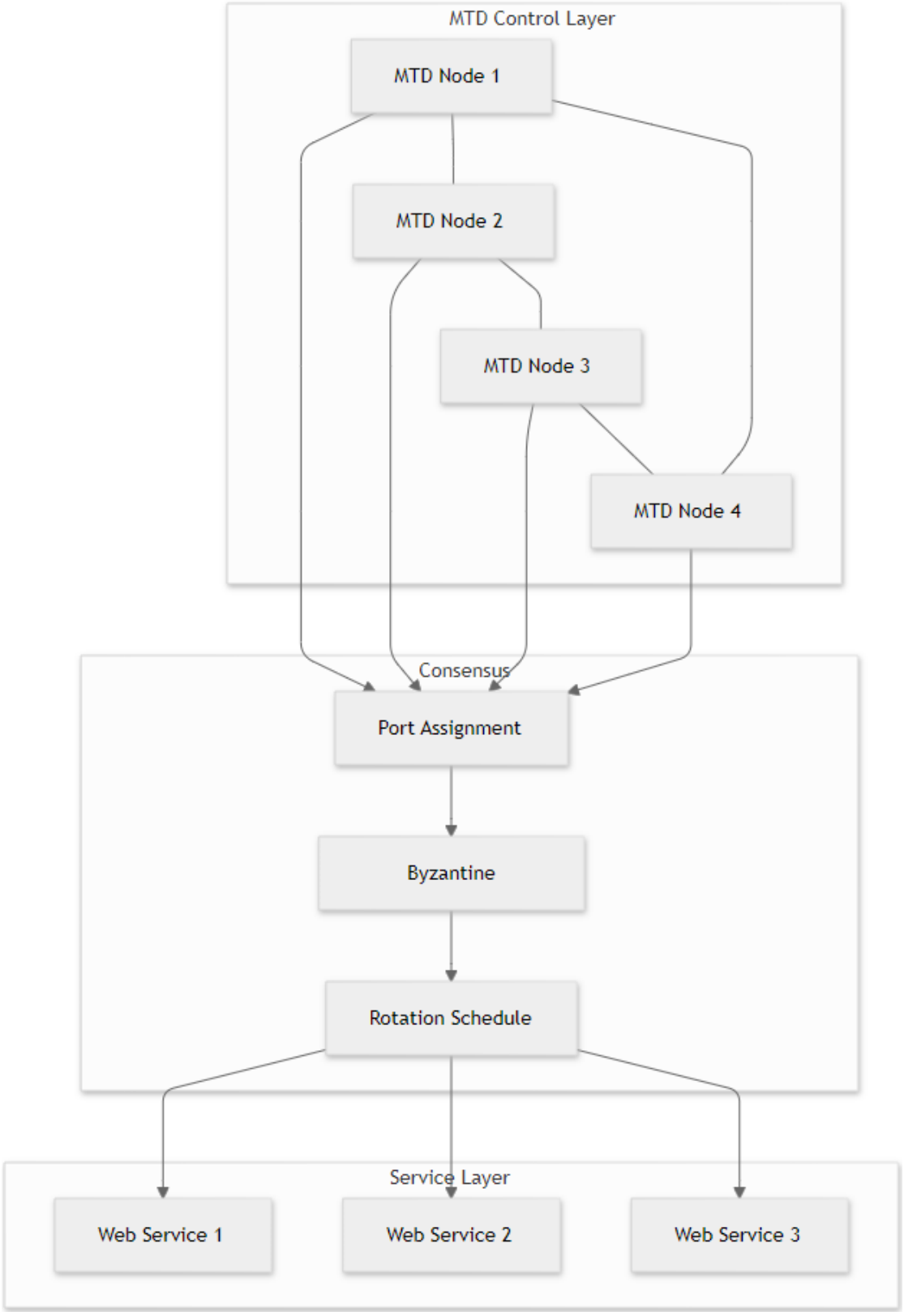


System Architecture

The system architecture builds upon the previous monitoring infrastructure while adding new components for attack tolerance. The key additions include:

1) MTD Control Layer
    1) Multiple MTD nodes for fault tolerance
    2) Distributed consensus for port assignments
    3) Byzantine behavior detection
2) Consensus Layer
    1) BFT-enhanced Paxos Implementation
    2) Malicious behavior detection
    3) Port assignment coordination
3) Integration Layer
    1) Service port management
    2) Rotation scheduling
    3) Container orchestration

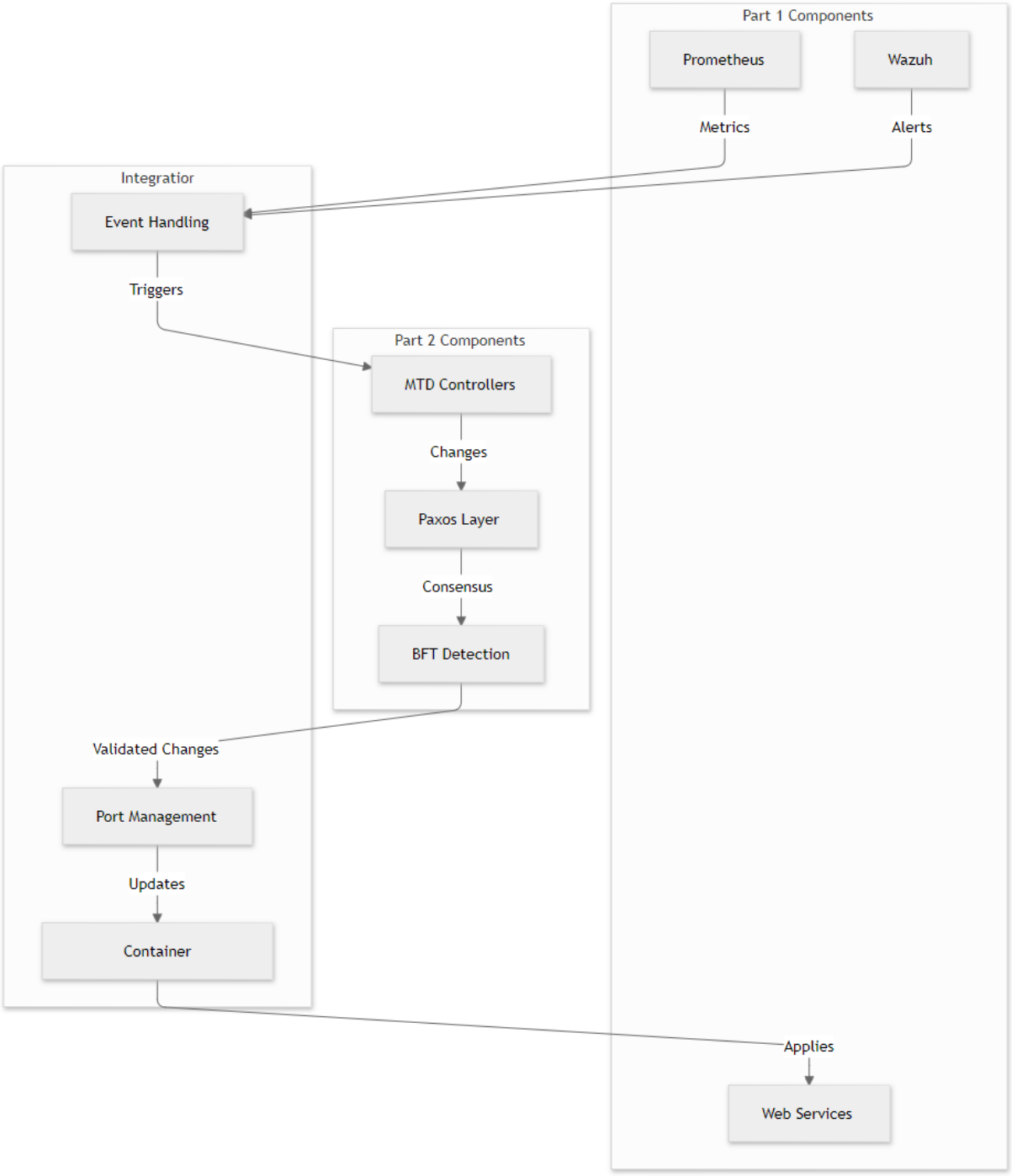
Below is a diagram that shows this at a high-level:


MTD Control Layer
MTD Node 1
MTD Node 2
MTD Node 3
MTD Node 4
Consensus
Port Assignment
Byzantine
Rotation Schedule
Service Layer
Web Service 1
Web Service 2
Web Service 3

3 Implementation Details

3.1 System Integration

Integration with the existing infrastructure from part 1 was taken into consideration and this was the end result:



3.2 Moving Target Defense Implementation

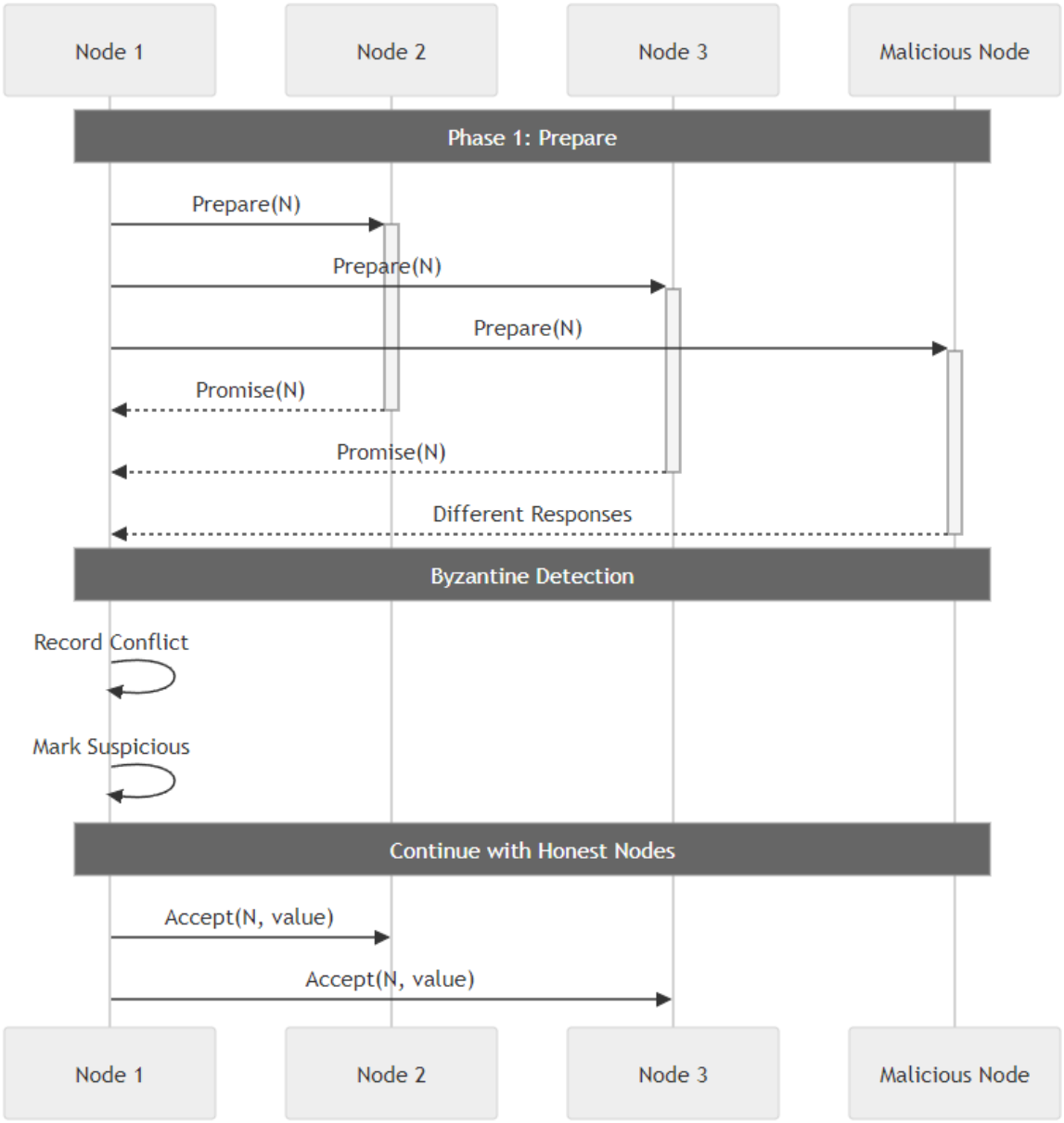
The MTD implementation focuses on port randomization with consensus-based decision making:

```python
class MTDService:
    def __init__(self):
        self.client = docker.from_env()
        self.port_range = range(8000, 9000)
        self.scheduler = BackgroundScheduler()
        self.bft_state = BFTState()

    def initiate_rotation(self):
        """Initiate a port rotation with consensus"""
        services = ['dotnet-ws1', 'dotnet-ws2', 'dotnet-ws3']
        new_assignments = {}
        for service in services:
            new_port = self.get_random_port()
            new_assignments[service] = new_port

        success, result = self.propose_port_changes(new_assignments)
        if success:
            return self.execute_rotation(result)
```

3.3 Byzantine Fault Tolerance Integration

Integrating BFT on the consensus protocol, for MTD, was done in a way that could be testable for the PoC provided in the project's assets. Obviously in a real environment we wouldn't have this, but in this case, a special environment variable called MALICIOUS was created, allowing us to quickly set a node to malicious by just setting this in the docker-compose.yml file of the infrastructure. The mtd-service then takes care of looking at this for each node and seeing how to act.

Below is a high level flow diagram of how this works:



Essentially, the BFT implementation includes:

1) Message History Tracking

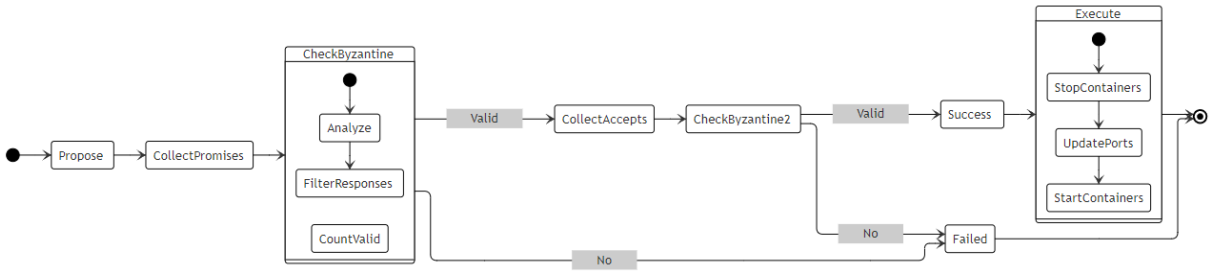
```python
def record_message(self, node_id, msg_type, message):
    """Record message for Byzantine detection"""
    if node_id not in self.message_history:
        self.message_history[node_id] = []
    self.message_history[node_id].append((msg_type, message))
```

2) Conflict Detection

```python
def check_byzantine_behavior(self, node_id):
    """Check if a node shows Byzantine behavior"""
    messages = self.message_history[node_id]
    for i in range(len(messages)):
        for j in range(i + 1, len(messages)):
            if self._are_messages_conflicting(messages[i], messages[j])
                :
                return True
    return False
```

3.4 Paxos Consensus Implementation

To explain the entire flow for the paxos implementation, it is easier to do so with a diagram, like the one found in the next image:



The MTD service has RESTful endpoints that take care of the multiple stages in the paxos consensus protocol.

4 Conclusions

This implementation successfully combines Moving Target Defense with Byzantine Fault Tolerance, as well as Paxos to create a robust, attack-tolerant system. The key achievements include:

1) Technical Achievements
    1) Successfully implemented BFT-enabled MTD
    2) Integrated with existing monitoring infrastructure
    3) Demonstrated resilience against Byzantine behavior
2) System Benefits
    1) Enhanced security through dynamic port assignment
    2) Reliable operation despite malicious nodes
    3) Automated response to detected threats

3) Project Objectives

    1) Met all core requirements
    2) Demonstrated practical attack tolerance
    3) Created maintainable, extensible solution

The system proves the feasibility of combining MTD with BFT in practical applications, with a consensus protocol. The implementation provides a solid foundation for further development of attack-tolerant systems in large-scale environments.